OpenShift Container Platform 3.4 Installation and Configuration

---

OpenShift Container Platform 3.4 Installation and Configuration

Red Hat OpenShift Documentation Team

OpenShift Container Platform 3.4 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 Container Platform Installation and Configuration topics cover the basics of installing and configuring OpenShift Container Platform in your environment. Configuration, management, and logging are also covered. Use these topics for the one-time tasks required quickly set up your OpenShift Container Platform environment and configure it based on your organizational needs.

For day to day cluster administrator tasks, see Cluster Administration.

CHAPTER 2. INSTALLING A CLUSTER

2.1. PLANNING

2.1.1. Initial Planning

For production environments, several factors influence installation. Consider the following questions as you read through the documentation:

- *Which installation method do you want to use?* The Installation Methods section provides some information about the quick and advanced installation methods.
- *How many hosts do you require in the cluster?* The Environment Scenarios section provides multiple examples of Single Master and Multiple Master configurations.
- *How many pods are required in your cluster?* The Sizing Considerations section provides limits for nodes and pods so you can calculate how large your environment needs to be.
- *Is high availability required?* High availability is recommended for fault tolerance. In this situation, you might aim to use the Multiple Masters Using Native HA example as a basis for your environment.
- *Which installation type do you want to use: RPM or containerized?* Both installations provide a working OpenShift Container Platform environment, but you might have a preference for a particular method of installing, managing, and updating your services.

2.1.2. Installation Methods

Both the quick and advanced installations methods are supported for development and production environments. If you want to quickly get OpenShift Container Platform up and running to try out for the first time, use the quick installer and let the interactive CLI guide you through the configuration options relevant to your environment.

For the most control over your cluster's configuration, you can use the advanced installation method. This method is particularly suited if you are already familiar with Ansible. However, following along with the OpenShift Container Platform documentation should equip you with enough information to reliably deploy your cluster and continue to manage its configuration post-deployment using the provided Ansible playbooks directly.

If you install initially using the quick installer, you can always further tweak your cluster's configuration and adjust the number of hosts in the cluster using the same installer tool. If you wanted to later switch to using the advanced method, you can create an inventory file for your configuration and carry on that way.

2.1.3. Sizing Considerations

Determine how many nodes and pods you require for your OpenShift Container Platform cluster. Cluster scalability correlates to the number of pods in a cluster environment. That number influences the other numbers in your setup.

The following table provides the maximum sizing limits for nodes and pods:

Type	Maximum
Maximum nodes per cluster	1000
Maximum pods per cluster	120,000
Maximum pods per node	250
Maximum pods per core	10

Important

Oversubscribing the physical resources on a node affects resource guarantees the Kubernetes scheduler makes during pod placement. Learn what measures you can take to avoid memory swapping.

Determine how many pods are expected to fit per node:

```
Maximum Pods per Cluster / Expected Pods per Node = Total Number of
Nodes
```

Example Scenario

If you want to scope your cluster for 2200 pods per cluster, you would need at least 9 nodes, assuming that there are 250 maximum pods per node:

```
2200 / 250 = 8.8
```

If you increase the number of nodes to 20, then the pod distribution changes to 110 pods per node:

```
2200 / 20 = 110
```

2.1.4. Environment Scenarios

This section outlines different examples of scenarios for your OpenShift Container Platform environment. Use these scenarios as a basis for planning your own OpenShift Container Platform cluster.

Note

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

2.1.4.1. Single Master and Multiple Nodes

The following table describes an example environment for a single master (with embedded **etcd**) and two nodes:

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

2.1.4.2. 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 Container Platform's embedded **etcd** is not supported.

2.1.4.3. Multiple Masters Using Native HA

The following describes an example environment for three masters, one HAProxy load balancer, three **etcd** hosts, and two nodes using the **`native`** HA method:

Host Name	Infrastructure Component to Install
master1.example.com	Master (clustered using native HA) and node
master2.example.com	
master3.example.com	
lb.example.com	HAProxy to load balance API master endpoints
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 Container Platform's embedded **etcd** is not supported.

2.1.4.4. Stand-alone Registry

You can also install OpenShift Container Platform to act as a stand-alone registry using the OpenShift Container Platform's integrated registry. See Installing a Stand-alone Registry for details on this scenario.

2.1.5. RPM vs Containerized

An RPM installation installs all services through package management and configures services to run within the same user space, while a containerized installation configures installs services using container images and runs separate services in individual containers.

The default method for installing OpenShift Container Platform on Red Hat Enterprise Linux (RHEL) uses RPMs. Alternatively, you can use the containerized method, which deploys containerized OpenShift Container Platform master and node components. When targeting a RHEL Atomic Host system, the containerized method is the only available option, and is automatically selected for you based on the detection of the ***/run/ostree-booted*** file.

The following table outlines the differences between the RPM and Containerized methods:

Type	RPM	Containerized
Installation Method	Packages via **`yum`**	Container images via **`docker`**
Service Management	**`systemd`**	**`docker`** and **`systemd`** units
Operating System	Red Hat Enterprise Linux	Red Hat Enterprise Linux or Red Hat Atomic Host

The Containerized Installation Preparation section provides more details on configuring your installation to use containerized services.

2.2. PREREQUISITES

2.2.1. System Requirements

The following sections identify the hardware specifications and system-level requirements of all hosts within your OpenShift Container Platform environment.

2.2.1.1. Red Hat Subscriptions

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

Important

OpenShift Container Platform 3.4 requires Docker 1.12.

2.2.1.2. Minimum Hardware Requirements

The system requirements vary per host type:

Masters	- Physical or virtual system, or an instance running on a public or private IaaS. - Base OS: RHEL 7.3 or later with "Minimal" installation option, or RHEL Atomic Host 7.3.2 or later. RHEL 7.2 is also supported using Docker 1.12 and its dependencies. - 2 vCPU. - Minimum 16 GB RAM. - Minimum 40 GB hard disk space for the file system containing ***/var/***.
Nodes	- Physical or virtual system, or an instance running on a public or private IaaS. - Base OS: RHEL 7.3 or later with "Minimal" installation option, or RHEL Atomic Host 7.3.2 or later. RHEL 7.2 is also supported using Docker 1.12 and its dependencies. - NetworkManager 1.0 or later. - 1 vCPU. - Minimum 8 GB RAM. - Minimum 15 GB hard disk space for the file system containing ***/var/***. - An additional minimum 15 GB unallocated space to be used for Docker's storage back end; see Configuring Docker Storage.
External etcd Nodes	- Minimum 20 GB hard disk space for etcd data. - Consult Hardware Recommendations to properly size your etcd nodes. - Currently, OpenShift Container Platform stores image metadata in etcd. You must periodically prune old images. If you are planning to leverage a large number of images, place etcd on machines with large amounts of memory and fast SSD drives.

Important

OpenShift Container Platform only supports servers with x86_64 architecture.

Note

Meeting the ***/var/*** file system sizing requirements in RHEL Atomic Host requires making changes to the default configuration. See Managing Storage in Red Hat Enterprise Linux Atomic Host for instructions on configuring this during or after installation.

2.2.1.3. Production Level Hardware Requirements

Test or sample environments function with the minimum requirements. For production environments, the following recommendations apply:

Master Hosts

In a highly available OpenShift Container Platform cluster with external etcd, a master host should have 1 CPU core and 1.5 GB of memory, on top of the defaults in the table above,

for each 1000 pods. Therefore, the recommended size of master host in an OpenShift Container Platform cluster of 2000 pods would be 2 CPU cores and 5 GB of RAM, in addition to the minimum requirements for a master host of 2 CPU cores and 8 GB of RAM.

When planning an environment with multiple masters, a minimum of three etcd hosts as well as a load-balancer between the master hosts, is required.

The OpenShift Container Platform master caches deserialized versions of resources aggressively to ease CPU load. However, in smaller clusters of less than 1000 pods, this cache can waste a lot of memory for negligible CPU load reduction. The default cache size is 50000 entries, which, depending on the size of your resources, can grow to occupy 1 to 2 GB of memory. This cache size can be reduced using the following setting the in ***master-config.yaml***:

```
kubernetesMasterConfig:
  apiServerArguments:
    deserialization-cache-size:
    - "1000"
```

Node Hosts

The size of a node host depends on the expected size of its workload. As an OpenShift Container Platform cluster administrator, you will need to calculate the expected workload, then add about 10 per cent for overhead. For production environments, allocate enough resources so that node host failure does not affect your maximum capacity.

Use the above with the following table to plan the maximum loads for nodes and pods:

Host	Sizing Recommendation
Maximum nodes per cluster	1000
Maximum pods per cluster	120000
Maximum pods per nodes	250
Maximum pods per core	10

Important

Oversubscribing the physical resources on a node affects resource guarantees the Kubernetes scheduler makes during pod placement. Learn what measures you can take to avoid memory swapping.

2.2.1.4. Configuring Core Usage

By default, OpenShift Container Platform masters and nodes use all available cores in the system they run on. You can choose the number of cores you want OpenShift Container Platform to use by

setting the **GOMAXPROCS** environment variable.

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

```
# export GOMAXPROCS=1
```

2.2.1.5. SELinux

Security-Enhanced Linux (SELinux) must be enabled on all of the servers before installing OpenShift Container Platform 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
```

2.2.1.6. Security Warning

OpenShift Container Platform runs 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/engine/security/security/

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

2.2.2. Environment Requirements

The following section defines the requirements of the environment containing your OpenShift Container Platform configuration. This includes networking considerations and access to external services, such as Git repository access, storage, and cloud infrastructure providers.

2.2.2.1. DNS

OpenShift Container Platform requires a fully functional DNS server in the environment. This is ideally a separate host running DNS software and can provide name resolution to hosts and containers running on the platform.

Important

Adding entries into the ***/etc/hosts*** file on each host is not enough. This file is not copied into containers running on the platform.

Key components of OpenShift Container Platform run themselves inside of containers and use the following process for name resolution:

1. By default, containers receive their DNS configuration file (***/etc/resolv.conf***) from their host.
2. OpenShift Container Platform then inserts one DNS value into the pods (above the node's nameserver values). That value is defined in the ***/etc/origin/node/node-config.yaml*** file by the **`dnsIP`** parameter, which by default is set to the address of the host node because the host is using **dnsmasq**.
3. If the **`dnsIP`** parameter is omitted from the ***node-config.yaml*** file, then the value defaults to the kubernetes service IP, which is the first nameserver in the pod's ***/etc/resolv.conf*** file.

As of OpenShift Container Platform 3.2, **dnsmasq** is automatically configured on all masters and nodes. The pods use the nodes as their DNS, and the nodes forward the requests. By default, **dnsmasq** is configured on the nodes to listen on port 53, therefore the nodes cannot run any other type of DNS application.

Note

NetworkManager is required on the nodes in order to populate **dnsmasq** with the DNS IP addresses.

The following is an example set of DNS records for the Single Master and Multiple Nodes scenario:

```
master    A   10.64.33.100
node1     A   10.64.33.101
node2     A   10.64.33.102
```

If you do not have a properly functioning DNS environment, you could experience failure with:

- Product installation via the reference Ansible-based scripts
- Deployment of the infrastructure containers (registry, routers)
- Access to the OpenShift Container Platform web console, because it is not accessible via IP address alone

2.2.2.1.1. Configuring Hosts to Use DNS

Make sure each host in your environment is configured to resolve hostnames from your DNS server. The configuration for hosts' DNS resolution depend on whether DHCP is enabled. If DHCP is:

- Disabled, then configure your network interface to be static, and add DNS nameservers to NetworkManager.
- Enabled, then the NetworkManager dispatch script automatically configures DNS based on the DHCP configuration. Optionally, you can add a value to **dnsIP** in the ***node-config.yaml*** file to prepend the pod's ***resolv.conf*** file. The second nameserver is then defined by the host's first nameserver. By default, this will be the IP address of the node host.

Note

For most configurations, do not set the **openshift_dns_ip** option during the advanced installation of OpenShift Container Platform (using Ansible), because this option overrides the default IP address set by **dnsIP**.

Instead, allow the installer to configure each node to use **dnsmasq** and forward requests to SkyDNS or the external DNS provider. If you do set the **openshift_dns_ip** option, then it should be set either with a DNS IP that queries SkyDNS first, or to the SkyDNS service or endpoint IP (the Kubernetes service IP).

To properly check that hosts are correctly configured to resolved to your DNS server:

1. Check the contents of ***/etc/resolv.conf***:

```
$ cat /etc/resolv.conf
# Generated by NetworkManager
search example.com
nameserver 10.64.33.1
# nameserver updated by /etc/NetworkManager/dispatcher.d/99-
origin-dns.sh
```

   In this example, 10.64.33.1 is the address of our DNS server.

2. Test the DNS servers listed in ***/etc/resolv.conf*** are able to resolve to the addresses of all the masters and nodes in your OpenShift Container Platform environment:

```
$ dig <node_hostname> @<IP_address> +short
```

   For example:

```
$ dig master.example.com @10.64.33.1 +short
10.64.33.100
$ dig node1.example.com @10.64.33.1 +short
10.64.33.101
```

2.2.2.1.2. Disabling dnsmasq

If you want to disable **dnsmasq** (for example, if your ***/etc/resolv.conf*** is managed by a configuration tool other than NetworkManager), then set **openshift_use_dnsmasq** to **false** in the Ansible playbook.

However, certain containers do not properly move to the next nameserver when the first issues **SERVFAIL**. Red Hat Enterprise Linux (RHEL)-based containers do not suffer from this, but certain versions of **uclibc** and **musl** do.

2.2.2.1.3. Configuring a DNS Wildcard

Optionally, configure a wildcard for the router to use, so that you do not need to update your DNS configuration when new routes are added.

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

For example, create a wildcard DNS entry for **cloudapps** that has a low time-to-live value (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.

> **Warning**
>
> In your ***/etc/resolv.conf*** file on each node host, ensure that the DNS server that has the wildcard entry is not listed as a nameserver or that the wildcard domain is not listed in the search list. Otherwise, containers managed by OpenShift Container Platform may fail to resolve host names properly.

2.2.2.2. Network Access

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.

2.2.2.2.1. NetworkManager

NetworkManager, a program for providing detection and configuration for systems to automatically connect to the network, is required.

2.2.2.2.2. Required Ports

The OpenShift Container Platform installation automatically creates a set of internal firewall rules on each host using **`iptables`**. However, if your network configuration uses an external firewall, such as a hardware-based firewall, you must ensure infrastructure components can communicate with each other through specific ports that act as communication endpoints for certain processes or services.

Ensure the following ports required by OpenShift Container Platform are open on your network and configured to allow access between hosts. Some ports are optional depending on your configuration and usage.

Table 2.1. Node to Node

4789	UDP	Required for SDN communication between pods on separate hosts.

Table 2.2. Nodes to Master

53 or **8053**	TCP/ UDP	Required for DNS resolution of cluster services (SkyDNS). Installations prior to 3.2 or environments upgraded to 3.2 use port 53. New installations will use 8053 by default so that **dnsmasq** may be configured.
4789	UDP	Required for SDN communication between pods on separate hosts.
443 or **8443**	TCP	Required for node hosts to communicate to the master API, for the node hosts to post back status, to receive tasks, and so on.

Table 2.3. Master to Node

4789	UDP	Required for SDN communication between pods on separate hosts.
10250	TCP	The master proxies to node hosts via the Kubelet for **oc** commands.

Note

In the following table, **(L)** indicates the marked port is also used in *loopback mode*, enabling the master to communicate with itself.

In a single-master cluster:

- Ports marked with **(L)** must be open.
- Ports not marked with **(L)** need not be open.

In a multiple-master cluster, all the listed ports must be open.

Table 2.4. Master to Master

53 (L) or **8053** (L)	TCP/ UDP	Required for DNS resolution of cluster services (SkyDNS). Installations prior to 3.2 or environments upgraded to 3.2 use port 53. New installations will use 8053 by default so that **dnsmasq** may be configured.

2049 (L)	TCP/ UDP	Required when provisioning an NFS host as part of the installer.
2379	TCP	Used for standalone etcd (clustered) to accept changes in state.
2380	TCP	etcd requires this port be open between masters for leader election and peering connections when using standalone etcd (clustered).
4001 (L)	TCP	Used for embedded etcd (non-clustered) to accept changes in state.
4789 (L)	UDP	Required for SDN communication between pods on separate hosts.

Table 2.5. External to Load Balancer

9000	TCP	If you choose the **`native`** HA method, optional to allow access to the HAProxy statistics page.

Table 2.6. External to Master

443 or **8443**	TCP	Required for node hosts to communicate to the master API, for node hosts to post back status, to receive tasks, and so on.

Table 2.7. IaaS Deployments

22	TCP	Required for SSH by the installer or system administrator.
53 or **8053**	TCP/ UDP	Required for DNS resolution of cluster services (SkyDNS). Installations prior to 3.2 or environments upgraded to 3.2 use port 53. New installations will use 8053 by default so that **dnsmasq** may be configured. Only required to be internally open on master hosts.
80 or **443**	TCP	For HTTP/HTTPS use for the router. Required to be externally open on node hosts, especially on nodes running the router.

1936	TCP	For router statistics use. Required to be open when running the template router to access statistics, and can be open externally or internally to connections depending on if you want the statistics to be expressed publicly.
4001	TCP	For embedded etcd (non-clustered) use. Only required to be internally open on the master host. **4001** is for server-client connections.
2379 and **2380**	TCP	For standalone etcd use. Only required to be internally open on the master host. **2379** is for server-client connections. **2380** is for server-server connections, and is only required if you have clustered etcd.
4789	UDP	For VxLAN use (OpenShift SDN). Required only internally on node hosts.
8443	TCP	For use by the OpenShift Container Platform web console, shared with the API server.
10250	TCP	For use by the Kubelet. Required to be externally open on nodes.

Notes

- In the above examples, port **4789** is used for User Datagram Protocol (UDP).
- When deployments are using the SDN, the pod network is accessed via a service proxy, unless it is accessing the registry from the same node the registry is deployed on.
- OpenShift Container Platform internal DNS cannot be received over SDN. Depending on the detected values of **`openshift_facts`**, or if the **`openshift_ip`** and **`openshift_public_ip`** values are overridden, it will be the computed value of **`openshift_ip`**. For non-cloud deployments, this will default to the IP address associated with the default route on the master host. For cloud deployments, it will default to the IP address associated with the first internal interface as defined by the cloud metadata.
- The master host uses port **10250** to reach the nodes and does not go over SDN. It depends on the target host of the deployment and uses the computed values of **`openshift_hostname`** and **`openshift_public_hostname`**.

Table 2.8. Aggregated Logging

9200	TCP	For Elasticsearch API use. Required to be internally open on any infrastructure nodes so Kibana is able to retrieve logs for display. It can be externally opened for direct access to Elasticsearch by means of a route. The route can be created using **`oc expose`**.

9300	TCP	For Elasticsearch inter-cluster use. Required to be internally open on any infrastructure node so the members of the Elasticsearch cluster may communicate with each other.

2.2.2.3. Persistent Storage

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

The Installation and Configuration Guide provides instructions for cluster administrators on provisioning an OpenShift Container Platform cluster with persistent storage using NFS, GlusterFS, Ceph RBD, OpenStack Cinder, AWS Elastic Block Store (EBS), GCE Persistent Disks, and iSCSI.

2.2.2.4. Cloud Provider Considerations

There are certain aspects to take into consideration if installing OpenShift Container Platform on a cloud provider.

2.2.2.4.1. Configuring a Security Group

When installing on AWS or OpenStack, ensure that you set up the appropriate security groups. These are some ports that you should have in your security groups, without which the installation will fail. You may need more depending on the cluster configuration you want to install. For more information and to adjust your security groups accordingly, see Required Ports for more information.

All OpenShift Container Platform Hosts	- tcp/22 from host running the installer/Ansible
etcd Security Group	- tcp/2379 from masters - tcp/2380 from etcd hosts
Master Security Group	- tcp/8443 from 0.0.0.0/0 - tcp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 - udp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 - tcp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2 - udp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2

Node Security Group	- tcp/10250 from masters - tcp/4789 from nodes
Infrastructure Nodes (ones that can host the OpenShift Container Platform router)	- tcp/443 from 0.0.0.0/0 - tcp/80 from 0.0.0.0/0

If configuring ELBs for load balancing the masters and/or routers, you also need to configure Ingress and Egress security groups for the ELBs appropriately.

2.2.2.4.2. Overriding Detected IP Addresses and Host Names

Some deployments require that the user override the detected host names and IP addresses for the hosts. To see the default values, run the **`openshift_facts`** playbook:

```
# ansible-playbook playbooks/byo/openshift_facts.yml
```

Now, verify the detected common settings. If they are not what you expect them to be, you can override them.

The Advanced Installation topic discusses the available Ansible variables in greater detail.

Variable	Usage
`hostname`	- Should resolve to the internal IP from the instances themselves. - **`openshift_hostname`** overrides.
`ip`	- Should be the internal IP of the instance. - **`openshift_ip`** will overrides.
`public_hostname`	- Should resolve to the external IP from hosts outside of the cloud. - Provider **`openshift_public_hostname`** overrides.
`public_ip`	- Should be the externally accessible IP associated with the instance. - **`openshift_public_ip`** overrides.
`use_openshift_sdn`	- Should be true unless the cloud is GCE. - **`openshift_use_openshift_sdn`** overrides.

Warning

If **openshift_hostname** is set to a value other than the metadata-provided **private-dns-name** value, the native cloud integration for those providers will no longer work.

In AWS, situations that require overriding the variables include:

Variable	Usage
hostname	The user is installing in a VPC that is not configured for both **DNS hostnames** and **DNS resolution**.
ip	Possibly if they have multiple network interfaces configured and they want to use one other than the default. You must first set **openshift_node_set_node_ip** to **True**. Otherwise, the SDN would attempt to use the **hostname** setting or try to resolve the host name for the IP.
public_hostname	- A master instance where the VPC subnet is not configured for **Auto-assign Public IP**. For external access to this master, you need to have an ELB or other load balancer configured that would provide the external access needed, or you need to connect over a VPN connection to the internal name of the host. - A master instance where metadata is disabled. - This value is not actually used by the nodes.
public_ip	- A master instance where the VPC subnet is not configured for **Auto-assign Public IP**. - A master instance where metadata is disabled. - This value is not actually used by the nodes.

If setting **openshift_hostname** to something other than the metadata-provided **private-dns-name** value, the native cloud integration for those providers will no longer work.

For EC2 hosts in particular, they must be deployed in a VPC that has both **DNS host names** and **DNS resolution** enabled, and **openshift_hostname** should not be overridden.

2.2.2.4.3. Post-Installation Configuration for Cloud Providers

Following the installation process, you can configure OpenShift Container Platform for AWS, OpenStack, or GCE.

2.3. HOST PREPARATION

2.3. HOST PREPARATION

2.3.1. Operating System Requirements

A base installation of RHEL 7.3 or later or RHEL Atomic Host 7.3.2 or later is required for master and node hosts. RHEL 7.2 is also supported using Docker 1.12 and its dependencies. See the following documentation for the respective installation instructions, if required:

- Red Hat Enterprise Linux 7 Installation Guide
- Red Hat Enterprise Linux Atomic Host 7 Installation and Configuration Guide

2.3.2. Host Registration

Each host must be registered using Red Hat Subscription Manager (RHSM) and have an active OpenShift Container Platform 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 --matches '*OpenShift*'
```

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

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

4. Disable all yum repositories:

    a. Disable all the enabled RHSM repositories:

    ```
    # subscription-manager repos --disable="*"
    ```

    b. List the remaining yum repositories and note their names under **`repo id`**, if any:

    ```
    # yum repolist
    ```

    c. Use **`yum-config-manager`** to disable the remaining yum repositories:

    ```
    # yum-config-manager --disable <repo_id>
    ```

    Alternatively, disable all repositories:

    ```
     yum-config-manager --disable \*
    ```

    Note that this could take a few minutes if you have a large number of available repositories

5. Enable only the repositories required by OpenShift Container Platform 3.4:

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.4-rpms"
```

2.3.3. Installing Base Packages

For RHEL 7 systems:

1. Install the following base packages:

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

2. Update the system to the latest packages:

```
# yum update
```

3. Install the following package, which provides OpenShift Container Platform 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
```

4. Install the following ***-excluder** packages on each RHEL 7 system, which helps ensure your systems stay on the correct versions of **atomic-openshift** and **docker** packages when you are not trying to upgrade, according to the OpenShift Container Platform version:

```
# yum install atomic-openshift-excluder atomic-openshift-docker-
excluder
```

5. The ***-excluder** packages add entries to the **exclude** directive in the host's ***/etc/yum.conf*** file when installed. Run the following command on each host to remove the **atomic-openshift** packages from the list for the duration of the installation.

```
# atomic-openshift-excluder unexclude
```

For RHEL Atomic Host 7 systems:

1. Ensure the host is up to date by upgrading to the latest Atomic tree if one is available:

```
# atomic host upgrade
```

2. After the upgrade is completed and prepared for the next boot, reboot the host:

```
# systemctl reboot
```

2.3.4. Installing Docker

At this point, you should install Docker on all master and node hosts. This allows you to configure your Docker storage options before installing OpenShift Container Platform.

1. For RHEL 7 systems, install Docker 1.12.

Note

On RHEL Atomic Host 7 systems, Docker should already be installed, configured, and running by default.

The **atomic-openshift-docker-excluder** package that was installed in Installing Base Packages should ensure that the correct version of Docker is installed in this step:

```
# yum install docker
```

After the package installation is complete, verify that version 1.12 was installed:

```
# docker version
```

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'
```

If using the Quick Installation method, you can easily script a complete installation from a kickstart or cloud-init setup, change the default configuration file:

```
# sed -i '/OPTIONS=.*/c\OPTIONS="--selinux-enabled --insecure-
registry 172.30.0.0/16"' \
/etc/sysconfig/docker
```

Note

The Advanced Installation method automatically changes ***/etc/sysconfig/docker***.

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

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.

Note

After the initial OpenShift Container Platform installation is complete, you can choose to secure the integrated Docker registry, which involves adjusting the `--insecure-registry` option accordingly.

2.3.5. Configuring Docker Storage

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.

For RHEL Atomic Host

The default storage back end for Docker on RHEL Atomic Host is a thin pool logical volume, which is supported for production environments. You must ensure that enough space is allocated for this volume per the Docker storage requirements mentioned in System Requirements.

If you do not have enough allocated, see Managing Storage with Docker Formatted Containers for details on using **docker-storage-setup** and basic instructions on storage management in RHEL Atomic Host.

For RHEL

The default storage back end for Docker on RHEL 7 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.

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 Container Platform, 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 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 Container Platform.

2.3.5.1. 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.

2.3.5.2. Managing Container Logs

Sometimes a container's log file (the ***/var/lib/docker/containers/<hash>/<hash>-json.log*** file on the node where the container is running) can increase to a problematic size. You can manage this by configuring Docker's **`json-file`** logging driver to restrict the size and number of log files.

Option	Purpose

Option	Purpose
`--log-opt max-size`	Sets the size at which a new log file is created.
`--log-opt max-file`	Sets the file on each host to configure the options.

For example, to set the maximum file size to 1MB and always keep the last three log files, edit the ***/etc/sysconfig/docker*** file to configure **`max-size=1M`** and **`max-file=3`**:

```
OPTIONS='--insecure-registry=172.30.0.0/16 --selinux-enabled --log-opt
max-size=1M --log-opt max-file=3'
```

Next, restart the Docker service:

```
# systemctl restart docker
```

2.3.5.3. Viewing Available Container Logs

Container logs are stored in the ***/var/lib/docker/containers/<hash>/*** directory on the node where the container is running. For example:

```
# ls -lh
/var/lib/docker/containers/f088349cceac173305d3e2c2e4790051799efe363842
fdab5732f51f5b001fd8/
total 2.6M
-rw-r--r--. 1 root root 5.6K Nov 24 00:12 config.json
-rw-r--r--. 1 root root 649K Nov 24 00:15
f088349cceac173305d3e2c2e4790051799efe363842fdab5732f51f5b001fd8-
json.log
-rw-r--r--. 1 root root 977K Nov 24 00:15
f088349cceac173305d3e2c2e4790051799efe363842fdab5732f51f5b001fd8-
json.log.1
-rw-r--r--. 1 root root 977K Nov 24 00:15
f088349cceac173305d3e2c2e4790051799efe363842fdab5732f51f5b001fd8-
json.log.2
-rw-r--r--. 1 root root 1.3K Nov 24 00:12 hostconfig.json
drwx------. 2 root root    6 Nov 24 00:12 secrets
```

See Docker's documentation for additional information on how to Configure Logging Drivers.

2.3.6. 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.

2.3.7. What's Next?

If you are interested in installing OpenShift Container Platform using the containerized method (optional for RHEL but required for RHEL Atomic Host), see Containerized Components to prepare your hosts.

When you are ready to proceed, you can install OpenShift Container Platform using the quick installation or advanced installation method.

If you are installing a stand-alone registry, continue with Installing a Stand-alone Registry.

2.4. CONTAINERIZED COMPONENTS

2.4.1. Overview

This section explores some of the preparation required to install OpenShift Container Platform as a set of services within containers. This applies to hosts using either Red Hat Enterprise Linux or Red Hat Atomic Host.

- For the quick installation method, you can choose between the RPM or containerized method on a per host basis during the interactive installation, or set the values manually in an installation configuration file.
- For the advanced installation method, you can set the Ansible variable **`containerized=true`** in an inventory file on a cluster-wide or per host basis.

Note

When installing an environment with multiple masters, the load balancer cannot be deployed by the installation process as a container. See Advanced Installation for load balancer requirements using the native HA method.

The following sections detail the preparation for a containerized OpenShift Container Platform installation.

2.4.2. Required Images

Containerized installations make use of the following images:

- **openshift3/ose**
- **openshift3/node**
- **openshift3/openvswitch**
- **registry.access.redhat.com/rhel7/etcd**

By default, all of the above images are pulled from the Red Hat Registry at registry.access.redhat.com.

If you need to use a private registry to pull these images during the installation, you can specify the registry information ahead of time. For the advanced installation method, you can set the following Ansible variables in your inventory file, as required:

```
cli_docker_additional_registries=<registry_hostname>
cli_docker_insecure_registries=<registry_hostname>
cli_docker_blocked_registries=<registry_hostname>
```

For the quick installation method, you can export the following environment variables on each target host:

```
# export OO_INSTALL_ADDITIONAL_REGISTRIES=<registry_hostname>
# export OO_INSTALL_INSECURE_REGISTRIES=<registry_hostname>
```

Blocked Docker registries cannot currently be specified using the quick installation method.

The configuration of additional, insecure, and blocked Docker registries occurs at the beginning of the installation process to ensure that these settings are applied before attempting to pull any of the required images.

2.4.3. CLI Wrappers

When using containerized installations, a CLI wrapper script is deployed on each master at ***/usr/local/bin/openshift***. The following set of symbolic links are also provided to ease administrative tasks:

Symbolic Link	Usage
/usr/local/bin/oc	Developer CLI
/usr/local/bin/oadm	Administrative CLI
/usr/local/bin/kubectl	Kubernetes CLI

The wrapper spawns a new container on each invocation, so you may notice it run slightly slower than native CLI operations.

The wrapper scripts mount a limited subset of paths:

- ***~/.kube***
- ***/etc/origin/***
- ***/tmp/***

Be mindful of this when passing in files to be processed by the **oc** or **oadm** commands. You may find it easier to redirect the input, for example:

```
# oc create -f - < my-file.json
```

Note

The wrapper is intended only to be used to bootstrap an environment. You should install the CLI tools on another host after you have granted **cluster-admin** privileges to a user. See Managing Role Bindings and Get Started with the CLI for more information.

2.4.4. Starting and Stopping Containers

The installation process creates relevant **systemd** units which can be used to start, stop, and poll services using normal **systemctl** commands. For containerized installations, these unit names match those of an RPM installation, with the exception of the **etcd** service which is named **etcd_container**.

This change is necessary as currently RHEL Atomic Host ships with the **etcd** package installed as part of the operating system, so a containerized version is used for the OpenShift Container Platform installation instead. The installation process disables the default **etcd** service. The **etcd** package is slated to be removed from RHEL Atomic Host in the future.

2.4.5. File Paths

All OpenShift configuration files are placed in the same locations during containerized installation as RPM based installations and will survive **os-tree** upgrades.

However, the default image stream and template files are installed at ***/etc/origin/examples/*** for containerized installations rather than the standard ***/usr/share/openshift/examples/***, because that directory is read-only on RHEL Atomic Host.

2.4.6. Storage Requirements

RHEL Atomic Host installations normally have a very small root file system. However, the etcd, master, and node containers persist data in the ***/var/lib/*** directory. Ensure that you have enough space on the root file system before installing OpenShift Container Platform; see the System Requirements section for details.

2.4.7. Open vSwitch SDN Initialization

OpenShift SDN initialization requires that the Docker bridge be reconfigured and that Docker is restarted. This complicates the situation when the node is running within a container. When using the Open vSwitch (OVS) SDN, you will see the node start, reconfigure Docker, restart Docker (which restarts all containers), and finally start successfully.

In this case, the node service may fail to start and be restarted a few times because the master services are also restarted along with Docker. The current implementation uses a workaround which relies on setting the **`Restart=always`** parameter in the Docker based **systemd** units.

2.5. QUICK INSTALLATION

2.5.1. Overview

The *quick installation* method allows you to use an interactive CLI utility, the **`atomic-openshift-installer`** command, to install OpenShift Container Platform across a set of hosts. This installer can deploy OpenShift Container Platform components on targeted hosts by either installing RPMs or running containerized services.

This installation method is provided to make the installation experience easier by interactively gathering the data needed to run on each host. The installer is a self-contained wrapper intended for usage on a Red Hat Enterprise Linux (RHEL) 7 system. While RHEL Atomic Host is supported for running containerized OpenShift Container Platform services, the installer is provided by an RPM not available by default in RHEL Atomic Host, and must therefore be run from a RHEL 7 system. The host initiating the installation does not need to be intended for inclusion in the OpenShift Container Platform cluster, but it can be.

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 installer to:

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

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

Note

To install OpenShift Container Platform as a stand-alone registry, see Installing a Stand-alone Registry.

2.5.2. Before You Begin

The installer allows you to install OpenShift Container Platform 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 Container Platform 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 Container Platform, 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
- Whether to use the RPM or containerized method
- Internal and external IP addresses

If you are interested in installing OpenShift Container Platform using the containerized method (optional for RHEL but required for RHEL Atomic Host), see RPM vs Containerized to ensure that you understand the differences between these methods, then return to this topic to continue.

After following the instructions in the Prerequisites topic and deciding between the RPM and containerized methods, you can continue to running an interactive or unattended installation.

2.5.3. 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 Container Platform 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, verify the installation.

2.5.4. Defining an Installation Configuration File

The installer 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: v2 1
variant: openshift-enterprise 2
variant_version: 3.4 3
ansible_log_path: /tmp/ansible.log 4
deployment:
  ansible_ssh_user: root 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
    roles: 11
      - master
      - node
    containerized: 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_labels: {'region': 'infra'} 14
    roles:
      - node
    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
    roles:
      - node
    connect_to: 10.0.0.3
  roles: 15
    master:
      <variable_name1>: "<value1>" 16
      <variable_name2>: "<value2>"
    node:
      <variable_name1>: "<value1>" 17
```

1 The version of this installation configuration file. As of OpenShift Container Platform 3.3,

the only valid version here is **v2**.

2

The OpenShift Container Platform variant to install. For OpenShift Container Platform, set this to **`openshift-enterprise`**.

3

A valid version of your selected variant: **`3.4`**, **`3.3`**, **`3.2`**, or **`3.1`**. If not specified, this defaults to the latest version for the specified variant.

4

Defines where the Ansible logs are stored. By default, this is the ***/tmp/ansible.log*** file.

5

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.

6

Defines a list of the hosts onto which you want to install the OpenShift Container Platform 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 installer, and you are asked to confirm the details. If undefined for an unattended installation, the installation fails.

11

Determines the type of services that are installed. Specified as a list.

12

If set to **true**, containerized OpenShift Container Platform services are run on target master and node hosts instead of installed using RPM packages. If set to **false** or unset,

the default RPM method is used. RHEL Atomic Host requires the containerized method, and is automatically selected for you based on the detection of the ***/run/ostree-booted*** file. See RPM vs Containerized for more details.

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.

14

Node labels can optionally be set per-host.

15

Defines a dictionary of roles across the deployment.

16 17

Any ansible variables that should only be applied to hosts assigned a role can be defined. For examples, see Configuring Ansible.

2.5.5. 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 installer 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 installer with the **-u** flag:

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

By default in interactive or unattended mode, the installer 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, verify the installation.

2.5.6. Verifying the Installation

After the installation completes:

1. Verify that the master is started and nodes are registered and reporting in **Ready** status. **On the master host**, run the following as root:

```
# oc get nodes

NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

To verify that the web console is installed correctly, use the master host name and the console port number to access the console with a web browser.

For example, for a master host with a hostname of **`master.openshift.com`** and using the default port of **`8443`**, the web console would be found at:

```
https://master.openshift.com:8443/console
```

Now that the install has been verified, run the following command on each master and node host to add the **atomic-openshift** packages back to the list of yum excludes on the host:

```
# atomic-openshift-excluder exclude
```

Then, see What's Next for the next steps on configuring your OpenShift Container Platform cluster.

2.5.7. Uninstalling OpenShift Container Platform

You can uninstall OpenShift Container Platform from all hosts in your cluster using the installer's **`uninstall`** command. By default, the installer uses the installation configuration file located at ***~/.config/openshift/installer.cfg.yml*** if the file exists:

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

Alternatively, you can specify a different location for the configuration file using the **`-c`** option:

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

See the advanced installation method for more options.

2.5.8. What's Next?

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

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

- Configure the automatically-deployed integrated Docker registry.
- Configure the automatically-deployed router.

2.6. ADVANCED INSTALLATION

2.6.1. Overview

A reference configuration implemented using Ansible playbooks is available as the *advanced installation* method for installing a OpenShift Container Platform cluster. 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.

While RHEL Atomic Host is supported for running containerized OpenShift Container Platform services, the advanced installation method utilizes Ansible, which is not available in RHEL Atomic Host, and must therefore be run from a RHEL 7 system. The host initiating the installation does not need to be intended for inclusion in the OpenShift Container Platform cluster, but it can be.

Alternatively, you can use the quick installation method if you prefer an interactive installation experience.

Note

To install OpenShift Container Platform as a stand-alone registry, see Installing a Stand-alone Registry.

2.6.2. Before You Begin

Before installing OpenShift Container Platform, 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.

If you are interested in installing OpenShift Container Platform using the containerized method (optional for RHEL but required for RHEL Atomic Host), see RPM vs Containerized to ensure that you understand the differences between these methods, then return to this topic to continue.

After following the instructions in the Prerequisites topic and deciding between the RPM and containerized methods, you can continue in this topic to Configuring Ansible.

2.6.3. 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 Container Platform 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, including using multiple masters for high availability. 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 advanced installation.

2.6.3.1. Configuring Host Variables

To assign environment variables to hosts during the Ansible installation, 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 2.9. 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).

Variable	Purpose
`containerized`	If set to **true**, containerized OpenShift Container Platform services are run on target master and node hosts instead of installed using RPM packages. If set to **false** or unset, the default RPM method is used. RHEL Atomic Host requires the containerized method, and is automatically selected for you based on the detection of the ***/run/ostree-booted*** file. See RPM vs Containerized for more details. Containerized installations are supported starting in OpenShift Container Platform 3.1.1.
`openshift_node_labels`	This variable adds labels to nodes during installation. See Configuring Node Host Labels for more details.
`openshift_node_kubelet_args`	This variable is used to configure `kubeletArguments` on nodes, such as arguments used in container and image garbage collection, and to specify resources per node. `kubeletArguments` are key value pairs that are passed directly to the Kubelet that match the Kubelet's command line arguments. `kubeletArguments` are not migrated or validated and may become invalid if used. These values override other settings in node configuration which may cause invalid configurations. Example usage: **{'image-gc-high-threshold': ['90'],'image-gc-low-threshold': ['80']}**.
`openshift_hosted_router_selector`	Default node selector for automatically deploying router pods. See Configuring Node Host Labels for details.
`openshift_registry_selector`	Default node selector for automatically deploying registry pods. See Configuring Node Host Labels for details.
`openshift_docker_options`	This variable configures additional Docker options within ***/etc/sysconfig/docker***, such as options used in Managing Container Logs. Example usage: **"--log-driver json-file --log-opt max-size=1M --log-opt max-file=3"**.

2.6.3.2. Configuring Cluster Variables

To assign environment variables during the Ansible install that apply more globally to your OpenShift Container Platform 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/origin/master/htpasswd'}]

openshift_master_default_subdomain=apps.test.example.com
```

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

Table 2.10. 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_become`	If **`ansible_ssh_user`** is not **root**, this variable must be set to **true** and the user must be configured for passwordless **sudo**.
`containerized`	If set to **true**, containerized OpenShift Container Platform services are run on all target master and node hosts in the cluster instead of installed using RPM packages. If set to **false** or unset, the default RPM method is used. RHEL Atomic Host requires the containerized method, and is automatically selected for you based on the detection of the ***/run/ostree-booted*** file. See RPM vs Containerized for more details. Containerized installations are supported starting in OpenShift Container Platform 3.1.1.
`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.

Variable	Purpose
`openshift_master_cluster_method`	Optional. This variable defines the HA method when deploying multiple masters. Supports the **`native`** method. See Multiple Masters for more information.
`openshift_rolling_restart_mode`	This variable enables rolling restarts of HA masters (i.e., masters are taken down one at a time) when running the upgrade playbook directly. It defaults to **`services`**, which allows rolling restarts of services on the masters. It can instead be set to **`system`**, which enables rolling, full system restarts and also works for single master clusters.
`os_sdn_network_plugin_name`	This variable configures which OpenShift SDN plug-in to use for the pod network, which defaults to **redhat/openshift-ovs-subnet** for the standard SDN plug-in. Set the variable to **redhat/openshift-ovs-multitenant** to use the multitenant plug-in.
`openshift_master_identity_providers`	This variable overrides the identity provider, which defaults to Deny All.
`openshift_master_named_certificates`	These variables are used to configure custom certificates which are deployed as part of the installation. See Configuring Custom Certificates for more information.
`openshift_master_overwrite_named_certificates`	
`openshift_master_session_name`	These variables override defaults for session options in the OAuth configuration. See Configuring Session Options for more information.
`openshift_master_session_max_seconds`	
`openshift_master_session_auth_secrets`	

Variable	Purpose
`openshift_master_session_encryption_secrets`	
`openshift_master_portal_net`	This variable configures the subnet in which services will be created within the OpenShift Container Platform SDN. This network block should be private and must not conflict with any existing network blocks in your infrastructure to which pods, nodes, or the master may require access to, or the installation will fail. Defaults to **172.30.0.0/16**, and **cannot** be re-configured after deployment. If changing from the default, avoid **172.16.0.0/16**, which the **docker0** network bridge uses by default, or modify the **docker0** network.
`openshift_master_default_subdomain`	This variable overrides the default subdomain to use for exposed routes.
`openshift_node_proxy_mode`	This variable specifies the service proxy mode to use: either **iptables** for the default, pure-**iptables** implementation, or **userspace** for the user space proxy.
`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 must not conflict with existing network blocks in your infrastructure to which pods, nodes, or the master may require access. Defaults to **10.128.0.0/14** and **cannot** be arbitrarily re-configured after deployment, although certain changes to it can be made in the SDN master configuration.
`osm_host_subnet_length`	This variable specifies the size of the per host subnet allocated for pod IPs by OpenShift Container Platform SDN. Defaults to **9** which means that a subnet of size /23 is allocated to each host; for example, given the default 10.128.0.0/14 cluster network, this will allocate 10.128.0.0/23, 10.128.2.0/23, 10.128.4.0/23, and so on. This **cannot** be re-configured after deployment.

Variable	Purpose
openshift_use_flannel	This variable enables **flannel** as an alternative networking layer instead of the default SDN. If enabling **flannel**, disable the default SDN with the **openshift_use_openshift_sdn** variable. For more information, see Using Flannel.
openshift_docker_additional_registries	OpenShift Container Platform adds the specified additional registry or registries to the Docker configuration.
openshift_docker_insecure_registries	OpenShift Container Platform adds the specified additional insecure registry or registries to the Docker configuration.
openshift_docker_blocked_registries	OpenShift Container Platform adds the specified blocked registry or registries to the Docker configuration.
openshift_hosted_metrics_public_url	This variable sets the host name for integration with the metrics console. The default is **https://hawkular-metrics.{{openshift_master_default_subdomain}}/hawkular/metrics** If you alter this variable, ensure the host name is accessible via your router.

2.6.3.3. Configuring a Registry Location

If you are using an image registry other than the default at **registry.access.redhat.com**, specify the desired registry within the ***/etc/ansible/hosts*** file.

```
oreg_url=example.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
```

Table 2.11. Registry Variables

Variable	Purpose
oreg_url	Set to the alternate image location. Necessary if you are not using the default registry at **registry.access.redhat.com**.

Variable	Purpose
openshift_examples_modify_imagestreams	Set to **true** if pointing to a registry other than the default. Modifies the image stream location to the value of **oreg_url**.

2.6.3.4. Configuring Global Proxy Options

If your hosts require use of a HTTP or HTTPS proxy in order to connect to external hosts, there are many components that must be configured to use the proxy, including masters, Docker, and builds. Node services only connect to the master API requiring no external access and therefore do not need to be configured to use a proxy.

In order to simplify this configuration, the following Ansible variables can be specified at a cluster or host level to apply these settings uniformly across your environment.

Note

See Configuring Global Build Defaults and Overrides for more information on how the proxy environment is defined for builds.

Table 2.12. Cluster Proxy Variables

Variable	Purpose
openshift_http_proxy	This variable specifies the **HTTP_PROXY** environment variable for masters and the Docker daemon.
openshift_https_proxy	This variable specifices the **HTTPS_PROXY** environment variable for masters and the Docker daemon.
openshift_no_proxy	This variable is used to set the **NO_PROXY** environment variable for masters and the Docker daemon. This value should be set to a comma separated list of host names or wildcard host names that should not use the defined proxy. This list will be augmented with the list of all defined OpenShift Container Platform host names by default.

Variable	Purpose
openshift_generate_no_proxy_hosts	This boolean variable specifies whether or not the names of all defined OpenShift hosts and ***.cluster.local** should be automatically appended to the **NO_PROXY** list. Defaults to **true**; set it to **false** to override this option.
openshift_builddefaults_http_proxy	This variable defines the **HTTP_PROXY** environment variable inserted into builds using the **BuildDefaults** admission controller. If **openshift_http_proxy** is set, this variable will inherit that value; you only need to set this if you want your builds to use a different value.
openshift_builddefaults_https_proxy	This variable defines the **HTTPS_PROXY** environment variable inserted into builds using the **BuildDefaults** admission controller. If **openshift_https_proxy** is set, this variable will inherit that value; you only need to set this if you want your builds to use a different value.
openshift_builddefaults_no_proxy	This variable defines the **NO_PROXY** environment variable inserted into builds using the **BuildDefaults** admission controller. If **openshift_no_proxy** is set, this variable will inherit that value; you only need to set this if you want your builds to use a different value.
openshift_builddefaults_git_http_proxy	This variable defines the HTTP proxy used by **git clone** operations during a build, defined using the **BuildDefaults** admission controller. If **openshift_builddefaults_http_proxy** is set, this variable will inherit that value; you only need to set this if you want your **git clone** operations to use a different value.

Variable	Purpose
openshift_builddefaults_git_https_proxy	This variable defines the HTTPS proxy used by **git clone** operations during a build, defined using the **BuildDefaults** admission controller. If **openshift_builddefaults_https_proxy** is set, this variable will inherit that value; you only need to set this if you want your **git clone** operations to use a different value.

2.6.3.5. 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. Other than **region=infra** (discussed below), the actual label names and values are arbitrary and can be assigned however you see fit per your cluster's requirements.

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. In the following example, labels are set for a region called **primary** and a zone called **east**:

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

The **openshift_router_selector** and **openshift_registry_selector** Ansible settings are set to **region=infra** by default:

```
# default selectors for router and registry services
# openshift_router_selector='region=infra'
# openshift_registry_selector='region=infra'
```

The default router and registry will be automatically deployed if nodes exist that match the selector settings above. For example:

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

2.6.3.6. Marking Masters as Unschedulable Nodes

Any hosts you designate as masters during the installation process should also be configured as nodes by adding them to the **[nodes]** section so that the masters are configured as part of the OpenShift Container Platform SDN.

However, in order to ensure that your masters are not burdened with running pods, you can make them unschedulable by adding the **openshift_scheduleable=false** option any node that is also a master. For example:

```
[nodes]
master.example.com openshift_node_labels="
{'region':'infra','zone':'default'}" openshift_schedulable=false
```

2.6.3.7. Configuring Session Options

Session options in the OAuth configuration are configurable in the inventory file. By default, Ansible populates a **sessionSecretsFile** with generated authentication and encryption secrets so that sessions generated by one master can be decoded by the others. The default location is ***/etc/origin/master/session-secrets.yaml***, and this file will only be re-created if deleted on all masters.

You can set the session name and maximum number of seconds with **openshift_master_session_name** and **openshift_master_session_max_seconds**:

```
openshift_master_session_name=ssn
openshift_master_session_max_seconds=3600
```

If provided, **openshift_master_session_auth_secrets** and **openshift_master_encryption_secrets** must be equal length.

For **openshift_master_session_auth_secrets**, used to authenticate sessions using HMAC, it is recommended to use secrets with 32 or 64 bytes:

```
openshift_master_session_auth_secrets=
['DONT+USE+THIS+SECRET+b4NV+pmZNSO']
```

For **openshift_master_encryption_secrets**, used to encrypt sessions, secrets must be 16, 24, or 32 characters long, to select AES-128, AES-192, or AES-256:

```
openshift_master_session_encryption_secrets=
['DONT+USE+THIS+SECRET+b4NV+pmZNSO']
```

2.6.3.8. Configuring Custom Certificates

Custom serving certificates for the public host names of the OpenShift Container Platform API and web console can be deployed during an advanced installation and are configurable in the inventory file.

Note

Custom certificates should only be configured for the host name associated with the **publicMasterURL** which can be set using **openshift_master_cluster_public_hostname**. Using a custom serving certificate for the host name associated with the **masterURL** (**openshift_master_cluster_hostname**) will result in TLS errors as infrastructure components will attempt to contact the master API using the internal **masterURL** host.

Certificate and key file paths can be configured using the **openshift_master_named_certificates** cluster variable:

```
openshift_master_named_certificates=[{"certfile":
"/path/to/custom1.crt", "keyfile": "/path/to/custom1.key"}]
```

File paths must be local to the system where Ansible will be run. Certificates are copied to master hosts and are deployed within the ***/etc/origin/master/named_certificates/*** directory.

Ansible detects a certificate's **Common Name** and **Subject Alternative Names**. Detected names can be overridden by providing the **"names"** key when setting **openshift_master_named_certificates**:

```
openshift_master_named_certificates=[{"certfile":
"/path/to/custom1.crt", "keyfile": "/path/to/custom1.key", "names":
["public-master-host.com"]}]
```

Certificates configured using **openshift_master_named_certificates** are cached on masters, meaning that each additional Ansible run with a different set of certificates results in all previously deployed certificates remaining in place on master hosts and within the master configuration file.

If you would like **openshift_master_named_certificates** to be overwritten with the provided value (or no value), specify the **openshift_master_overwrite_named_certificates** cluster variable:

```
openshift_master_overwrite_named_certificates=true
```

For a more complete example, consider the following cluster variables in an inventory file:

```
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.openshift.com
openshift_master_cluster_public_hostname=custom.openshift.com
```

To overwrite the certificates on a subsequent Ansible run, you could set the following:

```
openshift_master_named_certificates=[{"certfile":
"/root/STAR.openshift.com.crt", "keyfile":
"/root/STAR.openshift.com.key", "names": ["custom.openshift.com"]}]
openshift_master_overwrite_named_certificates=true
```

2.6.4. Configuring Cluster Metrics

Cluster metrics are not set to automatically deploy by default. Set the following to enable cluster metrics when using the advanced install:

```
[OSEv3:vars]

openshift_hosted_metrics_deploy=true
```

2.6.4.1. Metrics Storage

The **openshift_hosted_metrics_storage_kind** variable must be set in order to use persistent storage. If **openshift_hosted_metrics_storage_kind** is not set, then cluster metrics data is stored in an **EmptyDir** volume, which will be deleted when the Cassandra pod terminates.

There are three options for enabling cluster metrics storage when using the advanced install:

Option A - NFS Host Group

When the following variables are set, an NFS volume is created during an advanced install with path **<nfs_directory>/<volume_name>** on the host within the [nfs] host group. For example, the volume path using these options would be **/exports/metrics**:

```
[OSEv3:vars]

openshift_hosted_metrics_storage_kind=nfs
openshift_hosted_metrics_storage_access_modes=['ReadWriteOnce']
openshift_hosted_metrics_storage_nfs_directory=/exports
openshift_hosted_metrics_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_metrics_storage_volume_name=metrics
openshift_hosted_metrics_storage_volume_size=10Gi
```

Option B - External NFS Host

To use an external NFS volume, one must already exist with a path of **<nfs_directory>/<volume_name>** on the storage host.

```
[OSEv3:vars]

openshift_hosted_metrics_storage_kind=nfs
openshift_hosted_metrics_storage_access_modes=['ReadWriteOnce']
openshift_hosted_metrics_storage_host=nfs.example.com
openshift_hosted_metrics_storage_nfs_directory=/exports
openshift_hosted_metrics_storage_volume_name=metrics
openshift_hosted_metrics_storage_volume_size=10Gi
```

The remote volume path using the following options would be **nfs.example.com:/exports/metrics**.

Option C - Dynamic

Use the following variable if your OpenShift Container Platform environment supports dynamic volume provisioning for your cloud platform:

```
[OSEv3:vars]

#openshift_hosted_metrics_storage_kind=dynamic
```

2.6.5. Single Master Examples

You can configure an environment with a single master and multiple nodes, and either a single embedded **etcd** or multiple external **etcd** hosts.

Note

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

Single Master and Multiple Nodes

The following table describes an example environment for a single master (with embedded **etcd**) 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_become must be set to
true
#ansible_become=true

deployment_type=openshift-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/origin/master/htpasswd'}]
```

```
# 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***.

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 Container Platform'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 2.3. Single Master, Multiple etcd, and Multiple Nodes Inventory File

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

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-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/origin/master/htpasswd'}]

# 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***.

2.6.6. Multiple Masters Examples

You can configure an environment with multiple masters, multiple **etcd** hosts, and multiple nodes. Configuring multiple masters for high availability (HA) ensures that the cluster has no single point of failure.

Note

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

When configuring multiple masters, the advanced installation supports the following high availability (HA) method:

native	Leverages the native HA master capabilities built into OpenShift Container Platform and can be combined with any load balancing solution. If a host is defined in the **[lb]** section of the inventory file, Ansible installs and configures HAProxy automatically as the load balancing solution. If no host is defined, it is assumed you have pre-configured a load balancing solution of your choice to balance the master API (port 8443) on all master hosts.

Note

For more on the high availability master architecture, see Kubernetes Infrastructure.

Note the following when using the **native** HA method:

- The advanced installation method does not currently support multiple HAProxy load balancers in an active-passive setup. See the Load Balancer Administration documentation for post-installation amendments.
- In a HAProxy setup, controller manager servers run as standalone processes. They elect their active leader with a lease stored in **etcd**. The lease expires after 30 seconds by default. If a failure happens on an active controller server, it will take up to this number of seconds to elect another leader. The interval can be configured with the **osm_controller_lease_ttl** variable.

To configure multiple masters, refer to the following section.

Multiple Masters with Multiple etcd, and Using Native HA

The following describes an example environment for three masters, one HAProxy load balancer, three **etcd** hosts, and two nodes using the **native** HA method:

Host Name	Infrastructure Component to Install
master1.example.com	Master (clustered using native HA) and node
master2.example.com	
master3.example.com	
lb.example.com	HAProxy to load balance API master endpoints

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 Container Platform's embedded **etcd** is not supported.

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

Example 2.4. Multiple Masters Using HAProxy Inventory File

```
# Create an OSEv3 group that contains the master, nodes, etcd, and
lb groups.
# The lb group lets Ansible configure HAProxy as the load balancing
solution.
# Comment lb out if your load balancer is pre-configured.
[OSEv3:children]
masters
nodes
etcd
lb

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-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/origin/master/htpasswd'}]

# Native high availbility cluster method with optional load balancer.
# If no lb group is defined installer assumes that a load balancer
has
# been preconfigured. For installation the value of
# openshift_master_cluster_hostname must resolve to the load balancer
# or to one or all of the masters defined in the inventory if no
load
# balancer is present.
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-cluster.example.com
openshift_master_cluster_public_hostname=openshift-
cluster.example.com

# apply updated node defaults
openshift_node_kubelet_args={'pods-per-core': ['10'], 'max-pods':
['250'], 'image-gc-high-threshold': ['90'], 'image-gc-low-
threshold': ['80']}

# override the default controller lease ttl
#osm_controller_lease_ttl=30

# enable ntp on masters to ensure proper failover
openshift_clock_enabled=true

# 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

# Specify load balancer host
[lb]
lb.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***.

Multiple Masters with Master and etcd on the Same Host, and Using Native HA

The following describes an example environment for three masters with **etcd** on each host, one HAProxy load balancer, and two nodes using the **`native`** HA method:

Host Name	Infrastructure Component to Install
master1.example.com	Master (clustered using native HA) and node with etcd on each host
master2.example.com	
master3.example.com	
lb.example.com	HAProxy to load balance API master endpoints
node1.example.com	Node
node2.example.com	

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

```
# Create an OSEv3 group that contains the master, nodes, etcd, and lb
groups.
# The lb group lets Ansible configure HAProxy as the load balancing
solution.
# Comment lb out if your load balancer is pre-configured.
[OSEv3:children]
masters
nodes
etcd
lb

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-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/origin/master/htpasswd'}]
```

```
# Native high availbility cluster method with optional load balancer.
# If no lb group is defined installer assumes that a load balancer has
# been preconfigured. For installation the value of
# openshift_master_cluster_hostname must resolve to the load balancer
# or to one or all of the masters defined in the inventory if no load
# balancer is present.
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-cluster.example.com
openshift_master_cluster_public_hostname=openshift-cluster.example.com

# override the default controller lease ttl
#osm_controller_lease_ttl=30

# host group for masters
[masters]
master1.example.com
master2.example.com
master3.example.com

# host group for etcd
[etcd]
master1.example.com
master2.example.com
master3.example.com

# Specify load balancer host
[lb]
lb.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***.

2.6.7. Running the Advanced Installation

After you have configured Ansible by defining an inventory file in ***/etc/ansible/hosts***, you can run the advanced installation using the following playbook:

```
# 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.

2.6.8. Verifying the Installation

After the installation completes:

1. Verify that the master is started and nodes are registered and reporting in **Ready** status. **On the master host**, run the following as root:

```
# oc get nodes

NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

To verify that the web console is installed correctly, use the master host name and the console port number to access the console with a web browser.

For example, for a master host with a hostname of **`master.openshift.com`** and using the default port of **8443**, the web console would be found at:

```
https://master.openshift.com:8443/console
```

Now that the install has been verified, run the following command on each master and node host to add the **atomic-openshift** packages back to the list of yum excludes on the host:

```
# atomic-openshift-excluder exclude
```

Note

The default port for the console is **8443**. If this was changed during the installation, the port can be found at **openshift_master_console_port** in the ***/etc/ansible/hosts*** file.

Multiple etcd Hosts

If you installed multiple **etcd** hosts:

1. On a master 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,htt
ps://etcd3.example.com:2379 \
    --ca-file=/etc/origin/master/master.etcd-ca.crt \
    --cert-file=/etc/origin/master/master.etcd-client.crt \
    --key-file=/etc/origin/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,htt
ps://etcd3.example.com:2379 \
```

```
    --ca-file=/etc/origin/master/master.etcd-ca.crt \
    --cert-file=/etc/origin/master/master.etcd-client.crt \
    --key-file=/etc/origin/master/master.etcd-client.key member
list
```

Multiple Masters Using HAProxy

If you installed multiple masters using HAProxy as a load balancer, browse to the following URL according to your **[lb]** section definition and check HAProxy's status:

```
http://<lb_hostname>:9000
```

You can verify your installation by consulting the HAProxy Configuration documentation.

2.6.9. Uninstalling OpenShift Container Platform

You can uninstall OpenShift Container Platform hosts in your cluster by running the ***uninstall.yml*** playbook. This playbook deletes OpenShift Container Platform content installed by Ansible, including:

- Configuration
- Containers
- Default templates and image streams
- Images
- RPM packages

The playbook will delete content for any hosts defined in the inventory file that you specify when running the playbook. If you want to uninstall OpenShift Container Platform across all hosts in your cluster, run the playbook using the inventory file you used when installing OpenShift Container Platform initially or ran most recently:

```
# ansible-playbook [-i /path/to/file] \
    /usr/share/ansible/openshift-ansible/playbooks/adhoc/uninstall.yml
```

2.6.9.1. Uninstalling Nodes

You can also uninstall node components from specific hosts using the ***uninstall.yml*** playbook while leaving the remaining hosts and cluster alone:

> **Warning**
>
> This method should only be used when attempting to uninstall specific node hosts and not for specific masters or etcd hosts, which would require further configuration changes within the cluster.

1. First follow the steps in Deleting Nodes to remove the node object from the cluster, then continue with the remaining steps in this procedure.

2. Create a different inventory file that only references those hosts. For example, to only delete content from one node:

```
[OSEv3:children]
nodes 1

[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-enterprise

[nodes]
node3.example.com openshift_node_labels="{'region': 'primary',
'zone': 'west'}" 2
```

1 Only include the sections that pertain to the hosts you are interested in uninstalling.

2 Only include hosts that you want to uninstall.

3. Specify that new inventory file using the **-i** option when running the ***uninstall.yml*** playbook:

```
# ansible-playbook -i /path/to/new/file \
    /usr/share/ansible/openshift-
ansible/playbooks/adhoc/uninstall.yml
```

When the playbook completes, all OpenShift Container Platform content should be removed from any specified hosts.

2.6.10. 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/kubernetes/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 using bare metal machines, run the following on all hosts:

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

# rm -rf /etc/origin /var/lib/openshift /etc/etcd \
    /var/lib/etcd /etc/sysconfig/atomic-openshift*
```

```
/etc/sysconfig/docker* \
    /root/.kube/config /etc/ansible/facts.d /usr/share/openshift
```

2.6.11. What's Next?

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

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

2.7. DISCONNECTED INSTALLATION

2.7.1. Overview

Frequently, portions of a datacenter may not have access to the Internet, even via proxy servers. Installing OpenShift Container Platform in these environments is considered a disconnected installation.

An OpenShift Container Platform disconnected installation differs from a regular installation in two primary ways:

- The OpenShift Container Platform software channels and repositories are not available via Red Hat's content distribution network.
- OpenShift Container Platform uses several containerized components. Normally, these images are pulled directly from Red Hat's Docker registry. In a disconnected environment, this is not possible.

A disconnected installation ensures the OpenShift Container Platform software is made available to the relevant servers, then follows the same installation process as a standard connected installation. This topic additionally details how to manually download the container images and transport them onto the relevant servers.

Once installed, in order to use OpenShift Container Platform, you will need source code in a source control repository (for example, Git). This topic assumes that an internal Git repository is available that can host source code and this repository is accessible from the OpenShift Container Platform nodes. Installing the source control repository is outside the scope of this document.

Also, when building applications in OpenShift Container Platform, your build may have some external dependencies, such as a Maven Repository or Gem files for Ruby applications. For this reason, and because they might require certain tags, many of the Quickstart templates offered by OpenShift Container Platform may not work on a disconnected environment. However, while Red Hat container images try to reach out to external repositories by default, you can configure OpenShift Container Platform to use your own internal repositories. For the purposes of this document, we assume that such internal repositories already exist and are accessible from the OpenShift Container Platform nodes hosts. Installing such repositories is outside the scope of this document.

Note

You can also have a Red Hat Satellite server that provides access to Red Hat content via an intranet or LAN. For environments with Satellite, you can synchronize the OpenShift Container Platform software onto the Satellite for use with the OpenShift Container Platform servers.

Red Hat Satellite 6.1 also introduces the ability to act as a Docker registry, and it can be used to host the OpenShift Container Platform containerized components. Doing so is outside of the scope of this document.

2.7.2. Prerequisites

This document assumes that you understand OpenShift Container Platform's overall architecture and that you have already planned out what the topology of your environment will look like.

2.7.3. Required Software and Components

In order to pull down the required software repositories and container images, you will need a Red Hat Enterprise Linux (RHEL) 7 server with access to the Internet and at least 100GB of additional free space. All steps in this section should be performed on the Internet-connected server as the root system user.

2.7.3.1. Syncing Repositories

Before you sync with the required repositories, you may need to import the appropriate GPG key:

```
# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

If the key is not imported, the indicated package is deleted after syncing the repository.

To sync the required repositories:

1. Register the server with the Red Hat Customer Portal. You must use the login and password associated with the account that has access to the OpenShift Container Platform subscriptions:

   ```
   # subscription-manager register
   ```

2. Attach to a subscription that provides OpenShift Container Platform channels. You can find the list of available subscriptions using:

   ```
   # subscription-manager list --available --matches '*OpenShift*'
   ```

   Then, find the pool ID for the subscription that provides OpenShift Container Platform, and attach it:

   ```
   # subscription-manager attach --pool=<pool_id>
   # subscription-manager repos --disable="*"
   # subscription-manager repos \
       --enable="rhel-7-server-rpms" \
   ```

```
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.4-rpms"
```

3. The **yum-utils** command provides the **reposync** utility, which lets you mirror yum repositories, and **createrepo** can create a usable **yum** repository from a directory:

```
# yum -y install yum-utils createrepo docker git
```

You will need up to 110GB of free space in order to sync the software. Depending on how restrictive your organization's policies are, you could re-connect this server to the disconnected LAN and use it as the repository server. You could use USB-connected storage and transport the software to another server that will act as the repository server. This topic covers these options.

4. Make a path to where you want to sync the software (either locally or on your USB or other device):

```
# mkdir -p </path/to/repos>
```

5. Sync the packages and create the repository for each of them. You will need to modify the command for the appropriate path you created above:

```
# for repo in \
rhel-7-server-rpms rhel-7-server-extras-rpms \
rhel-7-server-ose-3.4-rpms
do
  reposync --gpgcheck -lm --repoid=${repo} --
download_path=/path/to/repos
  createrepo -v </path/to/repos/>${repo} -o
</path/to/repos/>${repo}
done
```

2.7.3.2. Syncing Images

To sync the container images:

1. Start the Docker daemon:

```
# systemctl start docker
```

2. Pull all of the required OpenShift Container Platform containerized components. Replace **<tag>** with **v3.4.1.5** for the latest version.

```
# docker pull registry.access.redhat.com/openshift3/ose-haproxy-
router:<tag>
# docker pull registry.access.redhat.com/openshift3/ose-deployer:
<tag>
# docker pull registry.access.redhat.com/openshift3/ose-sti-
builder:<tag>
# docker pull registry.access.redhat.com/openshift3/ose-docker-
builder:<tag>
# docker pull registry.access.redhat.com/openshift3/ose-pod:<tag>
# docker pull registry.access.redhat.com/openshift3/ose-docker-
registry:<tag>
```

3. Pull all of the required OpenShift Container Platform containerized components for the additional centralized log aggregation and metrics aggregation components. Replace **<tag>** with **3.4.1** for the latest version.

```
# docker pull registry.access.redhat.com/openshift3/logging-
deployer:<tag>
# docker pull registry.access.redhat.com/openshift3/logging-
elasticsearch:<tag>
# docker pull registry.access.redhat.com/openshift3/logging-
kibana:<tag>
# docker pull registry.access.redhat.com/openshift3/logging-
fluentd:<tag>
# docker pull registry.access.redhat.com/openshift3/logging-
curator:<tag>
# docker pull registry.access.redhat.com/openshift3/logging-auth-
proxy:<tag>
# docker pull registry.access.redhat.com/openshift3/metrics-
deployer:<tag>
# docker pull registry.access.redhat.com/openshift3/metrics-
hawkular-metrics:<tag>
# docker pull registry.access.redhat.com/openshift3/metrics-
cassandra:<tag>
# docker pull registry.access.redhat.com/openshift3/metrics-
heapster:<tag>
```

4. Pull the Red Hat-certified Source-to-Image (S2I) builder images that you intend to use in your OpenShift environment. You can pull the following images:

   - jboss-eap70-openshift
   - jboss-amq-62
   - jboss-datagrid65-openshift
   - jboss-decisionserver62-openshift
   - jboss-eap64-openshift
   - jboss-eap70-openshift
   - jboss-webserver30-tomcat7-openshift
   - jboss-webserver30-tomcat8-openshift
   - mongodb
   - mysql
   - nodejs
   - perl
   - php
   - postgresql
   - python

- redhat-sso70-openshift
- ruby

Make sure to indicate the correct tag specifying the desired version number. For example, to pull both the previous and latest version of the Tomcat image:

```
# docker pull \
registry.access.redhat.com/jboss-webserver-3/webserver30-
tomcat7-openshift:latest
# docker pull \
registry.access.redhat.com/jboss-webserver-3/webserver30-
tomcat7-openshift:1.1
```

2.7.3.3. Preparing Images for Export

Container images can be exported from a system by first saving them to a tarball and then transporting them:

1. Make and change into a repository home directory:

```
# mkdir </path/to/repos/images>
# cd </path/to/repos/images>
```

2. Export the OpenShift Container Platform containerized components:

```
# docker save -o ose3-images.tar \
    registry.access.redhat.com/openshift3/ose-haproxy-router \
    registry.access.redhat.com/openshift3/ose-deployer \
    registry.access.redhat.com/openshift3/ose-sti-builder \
    registry.access.redhat.com/openshift3/ose-docker-builder \
    registry.access.redhat.com/openshift3/ose-pod \
    registry.access.redhat.com/openshift3/ose-docker-registry
```

3. If you synchronized the metrics and log aggregation images, export:

```
# docker save -o ose3-logging-metrics-images.tar \
    registry.access.redhat.com/openshift3/logging-deployer \
    registry.access.redhat.com/openshift3/logging-elasticsearch \
    registry.access.redhat.com/openshift3/logging-kibana \
    registry.access.redhat.com/openshift3/logging-fluentd \
    registry.access.redhat.com/openshift3/logging-auth-proxy \
    registry.access.redhat.com/openshift3/metrics-deployer \
    registry.access.redhat.com/openshift3/metrics-hawkular-
metrics \
    registry.access.redhat.com/openshift3/metrics-cassandra \
    registry.access.redhat.com/openshift3/metrics-heapster
```

4. Export the S2I builder images that you synced in the previous section. For example, if you synced only the Tomcat image:

```
# docker save -o ose3-builder-images.tar \
    registry.access.redhat.com/jboss-webserver-3/webserver30-
tomcat7-openshift:latest \
```

```
    registry.access.redhat.com/jboss-webserver-3/webserver30-
tomcat7-openshift:1.1
```

2.7.4. Repository Server

During the installation (and for later updates, should you so choose), you will need a webserver to host the repositories. RHEL 7 can provide the Apache webserver.

Option 1: Re-configuring as a Web server

If you can re-connect the server where you synchronized the software and images to your LAN, then you can simply install Apache on the server:

```
# yum install httpd
```

Skip to Placing the Software.

Option 2: Building a Repository Server

If you need to build a separate server to act as the repository server, install a new RHEL 7 system with at least 110GB of space. On this repository server during the installation, make sure you select the **Basic Web Server** option.

2.7.4.1. Placing the Software

1. If necessary, attach the external storage, and then copy the repository files into Apache's root folder. Note that the below copy step (**`cp -a`**) should be substituted with move (**`mv`**) if you are repurposing the server you used to sync:

```
# cp -a /path/to/repos /var/www/html/
# chmod -R +r /var/www/html/repos
# restorecon -vR /var/www/html
```

2. Add the firewall rules:

```
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload
```

3. Enable and start Apache for the changes to take effect:

```
# systemctl enable httpd
# systemctl start httpd
```

2.7.5. OpenShift Container Platform Systems

2.7.5.1. Building Your Hosts

At this point you can perform the initial creation of the hosts that will be part of the OpenShift Container Platform environment. It is recommended to use the latest version of RHEL 7 and to perform a minimal installation. You will also want to pay attention to the other OpenShift Container Platform-specific prerequisites.

Once the hosts are initially built, the repositories can be set up.

2.7.5.2. Connecting the Repositories

On all of the relevant systems that will need OpenShift Container Platform software components, create the required repository definitions. Place the following text in the ***/etc/yum.repos.d/ose.repo*** file, replacing **<server_IP>** with the IP or host name of the Apache server hosting the software repositories:

```
[rhel-7-server-rpms]
name=rhel-7-server-rpms
baseurl=http://<server_IP>/repos/rhel-7-server-rpms
enabled=1
gpgcheck=0
[rhel-7-server-extras-rpms]
name=rhel-7-server-extras-rpms
baseurl=http://<server_IP>/repos/rhel-7-server-extras-rpms
enabled=1
gpgcheck=0
[rhel-7-server-ose-3.4-rpms]
name=rhel-7-server-ose-3.4-rpms
baseurl=http://<server_IP>/repos/rhel-7-server-ose-3.4-rpms
enabled=1
gpgcheck=0
```

2.7.5.3. Host Preparation

At this point, the systems are ready to continue to be prepared following the OpenShift Container Platform documentation.

Skip the section titled **Host Registration** and start with **Installing Base Packages**.

2.7.6. Installing OpenShift Container Platform

2.7.6.1. Importing OpenShift Container Platform Containerized Components

To import the relevant components, securely copy the images from the connected host to the individual OpenShift Container Platform hosts:

```
# scp /var/www/html/repos/images/ose3-images.tar
root@<openshift_host_name>:
# ssh root@<openshift_host_name> "docker load -i ose3-images.tar"
```

If you prefer, you could use **wget** on each OpenShift Container Platform host to fetch the tar file, and then perform the Docker import command locally. Perform the same steps for the metrics and logging images, if you synchronized them.

On the host that will act as an OpenShift Container Platform master, copy and import the builder images:

```
# scp /var/www/html/images/ose3-builder-images.tar
root@<openshift_master_host_name>:
# ssh root@<openshift_master_host_name> "docker load -i ose3-builder-
images.tar"
```

2.7.6.2. Running the OpenShift Container Platform Installer

You can now choose to follow the quick or advanced OpenShift Container Platform installation instructions in the documentation.

2.7.6.3. Creating the Internal Docker Registry

You now need to create the internal Docker registry.

2.7.7. Post-Installation Changes

In one of the previous steps, the S2I images were imported into the Docker daemon running on one of the OpenShift Container Platform master hosts. In a connected installation, these images would be pulled from Red Hat's registry on demand. Since the Internet is not available to do this, the images must be made available in another Docker registry.

OpenShift Container Platform provides an internal registry for storing the images that are built as a result of the S2I process, but it can also be used to hold the S2I builder images. The following steps assume you did not customize the service IP subnet (172.30.0.0/16) or the Docker registry port (5000).

2.7.7.1. Re-tagging S2I Builder Images

1. On the master host where you imported the S2I builder images, obtain the service address of your Docker registry that you installed on the master:

```
# export REGISTRY=$(oc get service docker-registry -t
'{{.spec.clusterIP}}{{"\n"}}')
```

2. Next, tag all of the builder images that you synced and exported before pushing them into the OpenShift Container Platform Docker registry. For example, if you synced and exported only the Tomcat image:

```
# docker tag \
registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat7-
openshift:1.1 \
$REGISTRY:5000/openshift/webserver30-tomcat7-openshift:1.1
# docker tag \
registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat7-
openshift:latest \
$REGISTRY:5000/openshift/webserver30-tomcat7-openshift:1.2
# docker tag \
registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat7-
openshift:latest \
$REGISTRY:5000/openshift/webserver30-tomcat7-openshift:latest
```

2.7.7.2. Creating an Administrative User

Pushing the container images into OpenShift Container Platform's Docker registry requires a user with **cluster-admin** privileges. Because the default OpenShift Container Platform system administrator does not have a standard authorization token, they cannot be used to log in to the Docker registry.

To create an administrative user:

1. Create a new user account in the authentication system you are using with OpenShift Container Platform. For example, if you are using local **htpasswd**-based authentication:

```
# htpasswd -b /etc/openshift/openshift-passwd <admin_username>
<password>
```

2. The external authentication system now has a user account, but a user must log in to OpenShift Container Platform before an account is created in the internal database. Log in to OpenShift Container Platform for this account to be created. This assumes you are using the self-signed certificates generated by OpenShift Container Platform during the installation:

```
# oc login --certificate-authority=/etc/origin/master/ca.crt \
    -u <admin_username> https://<openshift_master_host>:8443
```

3. Get the user's authentication token:

```
# MYTOKEN=$(oc whoami -t)
# echo $MYTOKEN
iwo7hc4XilD2KOLL4V1O55ExH2VlPmLD-W2-JOd6Fko
```

2.7.7.3. Modifying the Security Policies

1. Using **oc login** switches to the new user. Switch back to the OpenShift Container Platform system administrator in order to make policy changes:

```
# oc login -u system:admin
```

2. In order to push images into the OpenShift Container Platform Docker registry, an account must have the **image-builder** security role. Add this to your OpenShift Container Platform administrative user:

```
# oadm policy add-role-to-user system:image-builder
<admin_username>
```

3. Next, add the administrative role to the user in the **openshift** project. This allows the administrative user to edit the **openshift** project, and, in this case, push the container images:

```
# oadm policy add-role-to-user admin <admin_username> -n
openshift
```

2.7.7.4. Editing the Image Stream Definitions

The **openshift** project is where all of the image streams for builder images are created by the installer. They are loaded by the installer from the ***/usr/share/openshift/examples*** directory. Change all of the definitions by deleting the image streams which had been loaded into OpenShift Container Platform's database, then re-create them:

1. Delete the existing image streams:

```
# oc delete is -n openshift --all
```

2. Make a backup of the files in ***/usr/share/openshift/examples/*** if you desire. Next, edit the file ***image-streams-rhel7.json*** in the ***/usr/share/openshift/examples/image-streams*** folder. You will find an image stream section for each of the builder images. Edit the **spec** stanza to point to your internal Docker registry.

   For example, change:

```
"spec": {
  "dockerImageRepository":
"registry.access.redhat.com/rhscl/mongodb-26-rhel7",
```

   to:

```
"spec": {
  "dockerImageRepository": "172.30.69.44:5000/openshift/mongodb-
26-rhel7",
```

   In the above, the repository name was changed from **rhscl** to **openshift**. You will need to ensure the change, regardless of whether the repository is **rhscl**, **openshift3**, or another directory. Every definition should have the following format:

```
<registry_ip>:5000/openshift/<image_name>
```

   Repeat this change for every image stream in the file. Ensure you use the correct IP address that you determined earlier. When you are finished, save and exit. Repeat the same process for the JBoss image streams in the ***/usr/share/openshift/examples/xpaas-streams/jboss-image-streams.json*** file.

3. Load the updated image stream definitions:

```
# oc create -f /usr/share/openshift/examples/image-streams/image-
streams-rhel7.json -n openshift
# oc create -f /usr/share/openshift/examples/xpaas-streams/jboss-
image-streams.json -n openshift
```

2.7.7.5. Loading the Container Images

At this point the system is ready to load the container images.

1. Log in to the Docker registry using the token and registry service IP obtained earlier:

```
# docker login -u adminuser -e mailto:adminuser@abc.com \
   -p $MYTOKEN $REGISTRY:5000
```

2. Push the Docker images:

```
# docker push $REGISTRY:5000/openshift/webserver30-tomcat7-
openshift:1.1
```

```
# docker push $REGISTRY:5000/openshift/webserver30-tomcat7-
openshift:1.2
# docker push $REGISTRY:5000/openshift/webserver30-tomcat7-
openshift:latest
```

3. Verify that all the image streams now have the tags populated:

```
# oc get imagestreams -n openshift
NAME                                     DOCKER REPO
TAGS                                         UPDATED
jboss-webserver30-tomcat7-openshift  $REGISTRY/jboss-webserver-
3/webserver30-jboss-tomcat7-openshift  1.1,1.1-2,1.1-6 + 2
more...              2 weeks ago
...
```

2.7.8. Installing a Router

At this point, the OpenShift Container Platform environment is almost ready for use. It is likely that you will want to install and configure a router.

2.8. INSTALLING A STAND-ALONE REGISTRY

2.8.1. Overview

OpenShift Container Platform is a fully-featured platform-as-a-service (PaaS) enterprise solution that includes an embedded container registry. However, it can alternatively be installed as a stand-alone container registry to run on-premise or in the cloud.

The stand-alone registry from OpenShift Container Platform provides the following capabilities:

- A user-focused registry web console.
- Secured traffic by default, served via TLS.
- Global identity provider authentication.
- A project namespace model to enable teams to collaborate through role-based access control (RBAC) authorization.
- A Kubernetes-based cluster to manage services.
- An image abstraction called image streams to enhance image management.

2.8.2. Minimum Hardware Requirements

Installing OpenShift Container Platform as a stand-alone registry has the following hardware requirements:

- Physical or virtual system, or an instance running on a public or private IaaS.
- Base OS: RHEL 7.1 or later with "Minimal" installation option, or RHEL Atomic Host 7.3.2 or later.
- NetworkManager 1.0 or later

- 2 vCPU.
- Minimum 16 GB RAM.
- Minimum 15 GB hard disk space for the file system containing ***/var/***.
- An additional minimum 15 GB unallocated space to be used for Docker's storage back end; see Configuring Docker Storage for details.

Important

OpenShift Container Platform only supports servers with x86_64 architecture.

Note

Meeting the ***/var/*** file system sizing requirements in RHEL Atomic Host requires making changes to the default configuration. See Managing Storage in Red Hat Enterprise Linux Atomic Host for instructions on configuring this during or after installation.

2.8.3. Supported System Topologies

The following system topologies are supported when running OpenShift Container Platform as a stand-alone registry:

All-in-one	A single host that includes the master, node, etcd, and registry components.
Multiple Masters (Highly-Available)	Three hosts with all components included on each (master, node, etcd, and registry), with the masters configured for native high-availability.

2.8.4. Host Preparation

Before installing a stand-alone registry, all of the same steps detailed in the Host Preparation topic for installing a full OpenShift Container Platform PaaS must be performed. This includes registering and subscribing the host(s) to the proper repositories, installing or updating certain packages, and setting up Docker and its storage requirements.

Follow the steps in the Host Preparation topic, then continue to Installation Methods.

2.8.5. Installation Methods

To install a stand-alone registry, use either of the standard installation methods (quick or advanced) used to install any variant of OpenShift Container Platform.

2.8.5.1. Quick Installation for Stand-alone Registries

When using the quick installation method to install a stand-alone registry, start the interactive installation by running:

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

Then follow the on-screen instructions to install a new registry. The installation questions will be largely the same as if you were installing a full OpenShift Container Platform PaaS, but when you reach the following screen:

```
Which variant would you like to install?


(1) OpenShift Container Platform
(2) Registry
```

Be sure to choose **2** to follow the registry installation path.

Note

For further usage details on the quick installer in general, see the full topic at Quick Installation.

2.8.5.2. Advanced Installation for Stand-alone Registries

When using the advanced installation method to install a stand-alone registry, use the same steps for installing a full OpenShift Container Platform PaaS using Ansible described in the full Advanced Installation topic. The main difference is that you must set **`deployment_subtype=registry`** in the inventory file within the **[OSEv3:vars]** section for the playbooks to follow the registry installation path.

See the following example inventory files for the different supported system topologies:

Example 2.5. All-in-one Stand-alone Registry 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_become must be set to
true
#ansible_become=true

deployment_type=openshift-enterprise
deployment_subtype=registry 1
```

```
# 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/origin/master/htpasswd'}]

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

# host group for nodes, includes region info
[nodes]
registry.example.com openshift_schedulable=true
openshift_node_labels="{'region': 'infra', 'zone': 'default'}" 2
```

1

Set **`deployment_subtype=registry`** to ensure installation of the stand-alone registry.

2

Set **`openshift_schedulable=true`** on the node entry to make the single node schedulable for pod placement.

Example 2.6. Multiple Masters (Highly-Available) Stand-alone Registry Inventory File

```
# Create an OSEv3 group that contains the master, nodes, etcd, and
lb groups.
# The lb group lets Ansible configure HAProxy as the load balancing
solution.
# Comment lb out if your load balancer is pre-configured.
[OSEv3:children]
masters
nodes
etcd
lb

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-enterprise
deployment_subtype=registry 1

# 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/origin/master/htpasswd'}]

# Native high availability cluster method with optional load
balancer.
# If no lb group is defined installer assumes that a load balancer
has
# been preconfigured. For installation the value of
# openshift_master_cluster_hostname must resolve to the load balancer
# or to one or all of the masters defined in the inventory if no
load
# balancer is present.
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-cluster.example.com
openshift_master_cluster_public_hostname=openshift-
cluster.example.com

# apply updated node defaults
openshift_node_kubelet_args={'pods-per-core': ['10'], 'max-pods':
['250'], 'image-gc-high-threshold': ['90'], 'image-gc-low-
threshold': ['80']}

# override the default controller lease ttl
#osm_controller_lease_ttl=30

# enable ntp on masters to ensure proper failover
openshift_clock_enabled=true

# 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

# Specify load balancer host
[lb]
lb.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'}"
```

1

> Setting **`deployment_subtype=registry`** ensures installation of the stand-alone registry.

After you have configured Ansible by defining an inventory file in ***/etc/ansible/hosts***, you can run the advanced installation using the following playbook:

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

Note

For more detailed usage information on the advanced installation method, including a comprehensive list of available Ansible variables, see the full topic at Advanced Installation.

CHAPTER 3. SETTING UP THE REGISTRY

3.1. REGISTRY OVERVIEW

3.1.1. About the Registry

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

3.1.2. Integrated or Stand-alone Registries

During an initial installation of a full OpenShift Container Platform cluster, it is likely that the registry was deployed automatically during the installation process. If it was not, or if you want to further customize the configuration of your registry, see Deploying a Registry on Existing Clusters.

While it can be deployed to run as an integrated part of your full OpenShift Container Platform cluster, the OpenShift Container Platform registry can alternatively be installed separately as a stand-alone container image registry.

To install a stand-alone registry, follow Installing a Stand-alone Registry. This installation path deploys an all-in-one cluster running a registry and specialized web console.

3.2. DEPLOYING A REGISTRY ON EXISTING CLUSTERS

3.2.1. Overview

If the integrated registry was not previously deployed automatically during the initial installation of your OpenShift Container Platform cluster, or if it is no longer running successfully and you need to redeploy it on your existing cluster, see the following sections for options on deploying a new registry.

Note

This topic is not required if you installed a stand-alone registry.

3.2.2. 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/origin/master/admin.kubeconfig \ 1
    --service-account=registry \ 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

`--service-account` is the service account used to run the registry's pod.

3

Required to pull the correct image for OpenShift Container Platform.

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
```

3.2.3. Deploying the Registry as a DaemonSet

Use the **`oadm registry`** command to deploy the registry as a DaemonSet with the **`--daemonset`** option.

Daemonsets ensure that when nodes are created, they contain copies of a specified pod. When the nodes are removed, the pods are garbage collected.

For more information on DaemonSets, see the DaemonSet documentation.

Note

Using DaemonSets with OpenShift Container Platform is a work in progress, and the only supported process is creating a registry. Currently, using deployment configurations to deploy a registry is recommended.

3.2.4. Registry Compute Resources

By default, the registry is created with no settings for compute resource requests or limits. For production, it is highly recommended that the deployment configuration for the registry be updated to set resource requests and limits for the registry pod. Otherwise, the registry pod will be considered a **BestEffort** pod.

See Compute Resources for more information on configuring requests and limits.

3.2.5. Storage for the Registry

The registry stores container 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.

3.2.5.1. Production Use

For production use, attach a remote volume or define and use the persistent storage method of your

choice.

For example, to use an existing persistent volume claim:

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

Note

See Known Issues if using a scaled registry with a shared NFS volume.

3.2.5.1.1. Use Amazon S3 as a Storage Back-end

There is also an option to use Amazon Simple Storage Service storage with the internal Docker registry. It is a secure cloud storage manageable through AWS Management Console. To use it, the registry's configuration file must be manually edited and mounted to the registry pod. However, before you start with the configuration, look at upstream's recommended steps.

Take a default YAML configuration file as a base and replace the **filesystem** entry in the **storage** section with **s3** entry such as below. The resulting storage section may look like this:

```
storage:
  cache:
    layerinfo: inmemory
  delete:
    enabled: true
  s3:
    accesskey: awsaccesskey
    secretkey: awssecretkey
    region: us-west-1
    regionendpoint: http://myobjects.local
    bucket: bucketname
    encrypt: true
    keyid: mykeyid
    secure: true
    v4auth: false
    chunksize: 5242880
    rootdirectory: /s3/object/name/prefix
```

All of the **s3** configuration options are documented in upstream's driver reference documentation.

Overriding the registry configuration will take you through the additional steps on mounting the configuration file into pod.

Warning

When the registry runs on the S3 storage back-end, there are reported issues.

3.2.5.2. Non-Production Use

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, create the registry and specify that it use the **registry** service account that was created during installation:

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

Important

The Docker registry pod runs as user **1001**. This user must be able to write to the host directory. You may need to change directory ownership to user ID **1001** with this command:

```
$ sudo chown 1001:root <path>
```

3.2.6. Enabling the Registry Console

OpenShift Container Platform provides a web-based interface to the integrated registry. This registry console is an optional component for browsing and managing images. It is deployed as a stateless service running as a pod.

Note

If you installed OpenShift Container Platform as a stand-alone registry, the registry console is already deployed and secured automatically during installation.

Important

If Cockpit is already running, you'll need to shut it down before proceeding in order to avoid a port conflict (9090 by default) with the registry console.

3.2.6.1. Deploying the Registry Console

Important

You must first have exposed the registry.

1. Create a passthrough route in the **default** project. You will need this when creating the registry console application in the next step.

```
$ oc create route passthrough --service registry-console \
    --port registry-console \
    -n default
```

2. Deploy the registry console application. Replace **<openshift_oauth_url>** with the URL of the OpenShift Container Platform OAuth provider, which is typically the master.

```
$ oc new-app -n default --template=registry-console \
    -p
OPENSHIFT_OAUTH_PROVIDER_URL="https://<openshift_oauth_url>:8443"
\
    -p REGISTRY_HOST=$(oc get route docker-registry -n default --
template='{{ .spec.host }}') \
    -p COCKPIT_KUBE_URL=$(oc get route registry-console -n
default --template='https://{{ .spec.host }}')
```

Note

If the redirection URL is wrong when you are trying to log in to the registry console, check your OAuth client with **oc get oauthclients**.

1. Finally, use a web browser to view the console using the route URI.

3.2.6.2. Securing the Registry Console

By default, the registry console generates self-signed TLS certificates if deployed manually per the steps in Deploying the Registry Console. See Troubleshooting the Registry Console for more information.

Use the following steps to add your organization's signed certificates as a secret volume. This assumes your certificates are available on the the **oc** client host.

1. Create a ***.cert*** file containing the certificate and key. Format the file with:

   - One or more **BEGIN CERTIFICATE** blocks for the server certificate and the intermediate certificate authorities
   - A block containing a **BEGIN PRIVATE KEY** or similar for the key. The key must not be encrypted

     For example:

```
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBg
NV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDDA
ls
...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBg
NV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDDA
ls
...
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCyOJ5garOYw0
sm
8TBCDSqQ/H1awGMzDYdB11xuHHsxYS2VepPMzMzryHR137I4dGFLhvdTvJUH8l
US
...
-----END PRIVATE KEY-----
```

The registry console loads a certificate from the ***/etc/cockpit/ws-certs.d*** directory. It uses the last file with a ***.cert*** extension in alphabetical order. Therefore, the ***.cert*** file should contain at least two PEM blocks formatted in the OpenSSL style.

If no certificate is found, a self-signed certificate is created using the **`openssl`** command and stored in the ***0-self-signed.cert*** file.

2. Create the secret:

```
$ oc secrets new console-secret \
    /path/to/console.cert
```

3. Add the secrets to the **registry-console** deployment configuration:

```
$ oc volume dc/registry-console --add --type=secret \
    --secret-name=console-secret -m /etc/cockpit/ws-certs.d
```

This triggers a new deployment of the registry console to include your signed certificates.

3.2.6.3. Troubleshooting the Registry Console

3.2.6.3.1. Debug Mode

The registry console debug mode is enabled using an environment variable. The following command redeploys the registry console in debug mode:

```
$ oc set env dc registry-console G_MESSAGES_DEBUG=cockpit-ws,cockpit-
wrapper
```

Enabling debug mode allows more verbose logging to appear in the registry console's pod logs.

3.2.6.3.2. Display SSL Certificate Path

To check which certificate the registry console is using, a command can be run from inside the console pod.

1. List the pods in the **default** project and find the registry console's pod name:

```
$ oc get pods -n default
NAME                       READY     STATUS    RESTARTS   AGE
registry-console-1-rssrw   1/1       Running   0          1d
```

2. Using the pod name from the previous command, get the certificate path that the **cockpit-ws** process is using. This example shows the console using the auto-generated certificate:

```
$ oc exec registry-console-1-rssrw remotectl certificate
certificate: /etc/cockpit/ws-certs.d/0-self-signed.cert
```

3.3. ACCESSING THE REGISTRY

3.3.1. Viewing Logs

To view the logs for the Docker registry, use the **`oc logs`** command with the deployment config:

```
$ oc logs dc/docker-registry
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
```

3.3.2. File Storage

Tag and image metadata is stored in OpenShift Container Platform, 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/ede17b139a271d6b1331
ca3d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331
ca3d83c648c24f92cece5f89d95ac6c34ce751111810/data (2)
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd
9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd
9fd84406680ae93d633cb16422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb
26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb
26f259582bb33502bdb0fcf5011e03c60577c4284845/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/rev
isions
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2
752a5c068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2
752a5c068b1cf/signatures (5)
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2
752a5c068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2
752a5c068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24
f92cece5f89d95ac6c34ce751111810
```

```
/registry/docker/registry/v2/repositories/p1/pause/_manifests/rev
isions/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2
752a5c068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24
f92cece5f89d95ac6c34ce751111810/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
/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/link 9
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/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.

3.3.3. Accessing the Registry Directly

For advanced usage, you can access the registry directly to invoke **`docker`** commands. This allows you to push images to or pull them from the integrated registry directly using operations like **`docker push`** or **`docker pull`**. To do so, you must be logged in to the registry using the **`docker login`** command. The operations you can perform depend on your user permissions, as described in the following sections.

3.3.3.1. User Prerequisites

To access the registry directly, the user that you use must satisfy the following, depending on your intended usage:

- For any direct access, you must have a regular user, if one does not already exist, for your preferred identity provider. A regular user can generate an access token required for logging in to the registry. System users, such as **system:admin**, cannot obtain access tokens and, therefore, cannot access the registry directly.

  For example, if you are using **`HTPASSWD`** authentication, you can create one using the following command:

  ```
  # htpasswd /etc/origin/openshift-htpasswd <user_name>
  ```

- The user must have the **system:registry** role. To add this role:

  ```
  # oadm policy add-role-to-user system:registry <user_name>
  ```

- Have the **admin** role for the project associated with the Docker operation. For example, if accessing images in the global **openshift** project:

  ```
  $ oadm policy add-role-to-user admin <user_name> -n openshift
  ```

- For writing or pushing images, for example when using the **`docker push`** command, the user must have the **system:image-builder** role. To add this role:

  ```
  $ oadm policy add-role-to-user system:image-builder <user_name>
  ```

For more information on user permissions, see Managing Role Bindings.

3.3.3.2. Logging in to the Registry

Note

Ensure your user satisfies the prerequisites for accessing the registry directly.

To log in to the registry directly:

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

```
$ oc login
```

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_ip>:<port>
```

3.3.3.3. Pushing and Pulling Images

After logging in to the registry, you can perform **`docker pull`** and **`docker push`** operations against your registry.

Important

You can pull arbitrary images, but if you have the **system:registry** role added, you can only push images to the registry in your project.

In the following examples, we use:

Component	Value
<registry_ip>	**`172.30.124.220`**
<port>	**`5000`**
<project>	**`openshift`**
<image>	**`busybox`**

<tag>	omitted (defaults to **latest**)

1. Pull an arbitrary image:

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

2. Tag the new image with the form **<registry_ip>:<port>/<project>/<image>**. The project name **must** appear in this pull specification for OpenShift Container Platform to correctly place and later access the image in the registry.

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

Note

Your regular user must have the **system:image-builder** role for the specified project, which allows the user to write or push an image. Otherwise, the **docker push** in the next step will fail. To test, you can create a new project to push the **busybox** image.

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

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f7740
3cab55
```

3.4. SECURING AND EXPOSING THE REGISTRY

3.4.1. 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=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --signer-serial=/etc/origin/master/ca.serial.txt \
    --hostnames='docker-
registry.default.svc.cluster.local,172.30.124.220' \
    --cert=/etc/secrets/registry.crt \
    --key=/etc/secrets/registry.key
```

   If the router will be exposed externally, add the public route host name in the **`--hostnames`** flag:

```
--hostnames='mydocker-registry.example.com,docker-
registry.default.svc.cluster.local,172.30.124.220
```

Note

The **`oadm ca create-server-cert`** command generates a certificate that is valid for two years. This can be altered with the **`--expire-days`** option, but for security reasons, it is recommended to not make it greater than this value.

4. Create the secret for the registry certificates:

```
$ oc secrets new registry-secret \
    /etc/secrets/registry.crt \
    /etc/secrets/registry.key
```

5. Add the secret to the registry pod's service accounts (including the **default** service account):

```
$ oc secrets link registry registry-secret
$ oc secrets link default  registry-secret
```

Note

Limiting secrets to only the service accounts that reference them is disabled by default. This means that if **`serviceAccountConfig.limitSecretReferences`** is set to **`false`** (the default setting) in the master configuration file, linking secrets to a service is not required.

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 set 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. Update the scheme used for the registry's liveness probe from HTTP to HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec":
{"containers":[{
    "name":"registry",
    "livenessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

9. If your registry was initially deployed on OpenShift Container Platform 3.2 or later, update the scheme used for the registry's readiness probe from HTTP to HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec":
{"containers":[{
    "name":"registry",
    "readinessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

10. Validate the registry is running in TLS mode. Wait until the latest **docker-registry** deployment completes and verify the Docker logs for the registry container. You should find an entry for **`listening on :5000, tls`**.

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

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

```
$ dcertsdir=/etc/docker/certs.d
$ destdir_addr=$dcertsdir/172.30.124.220:5000
$ destdir_name=$dcertsdir/docker-
registry.default.svc.cluster.local:5000

$ sudo mkdir -p $destdir_addr $destdir_name
$ sudo cp ca.crt $destdir_addr     1
$ sudo cp ca.crt $destdir_name
```

1

The ***ca.crt*** file is a copy of ***/etc/origin/master/ca.crt*** on the master.

12. 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
```

13. 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:
    sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f7740
    3cab55
    ```

3.4.2. 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 a passthrough route via the **oc create route passthrough** command, specifying the registry as the route's service. By default, the name of the created route is the same as the service name.

   For example:

   ```
   $ oc get svc
   NAME              CLUSTER_IP       EXTERNAL_IP   PORT(S)
   SELECTOR                  AGE
   docker-registry   172.30.69.167    <none>        5000/TCP
   docker-registry=default   4h
   kubernetes        172.30.0.1       <none>
   443/TCP,53/UDP,53/TCP   <none>                    4h
   router            172.30.172.132   <none>        80/TCP
   router=router             4h

   $ oc create route passthrough    \
       --service=docker-registry    \ 1
       --hostname=<host>
   route "docker-registry" created     2
   ```

1

Specify the registry as the route's service.

2

The route name is identical to the service name.

```
$ oc get route/docker-registry -o yaml
apiVersion: v1
kind: Route
metadata:
  name: docker-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:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147
d8ca31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147
d8ca31
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.

3.4.2.1. Exposing a Secure Registry

Instead of logging in to the registry from within the OpenShift Container Platform cluster, you can gain external access to it by first securing the registry and then exposing the registry. This allows you to log in to the registry from outside the cluster using the route address, and to tag and push images using the route host.

When logging in to the secured and exposed registry, make sure you specify the registry in the login command. For example:

-

```
docker login -e user@company.com -u f83j5h6 -p Ju1PeM47R0B92Lk3AZp-
bWJSck2F7aGCiZ66aFGZrs2 registry.example.com:80
```

3.4.2.2. Exposing a Non-Secure Registry

Instead of securing the registry in order to expose the registry, you can simply expose a non-secure registry for non-production OpenShift Container Platform environments. This allows you to have an external route to the registry without using SSL certificates.

> **Warning**
>
> Only non-production environments should expose a non-secure registry to external access.

To expose a non-secure registry:

1. Expose the registry:

   ```
   # oc expose service docker-registry --hostname=<hostname> -n
   default
   ```

   This creates the following JSON file:

   ```
   apiVersion: v1
   kind: Route
   metadata:
     creationTimestamp: null
     labels:
       docker-registry: default
     name: docker-registry
   spec:
     host: registry.example.com
     port:
       targetPort: "5000"
     to:
       kind: Service
       name: docker-registry
   status: {}
   ```

2. Verify that the route has been created successfully:

   ```
   # oc get route
   NAME              HOST/PORT                   PATH
   SERVICE           LABELS                    INSECURE POLICY
   TLS TERMINATION
   docker-registry   registry.example.com           docker-
   registry   docker-registry=default
   ```

3. Check the health of the registry:

   ```
   $ curl -v http://registry.example.com/healthz
   ```

Expect an HTTP 200/OK message.

After exposing the registry, update your ***/etc/sysconfig/docker*** file by adding the port number to the **OPTIONS** entry. For example:

```
OPTIONS='--selinux-enabled --insecure-registry=172.30.0.0/16 --
insecure-registry registry.example.com:80'
```

Important

The above options should be added on the client from which you are trying to log in.

Also, ensure that Docker is running on the client.

When logging in to the non-secured and exposed registry, make sure you specify the registry in the login command. For example:

```
docker login -e user@company.com -u f83j5h6 -p Ju1PeM47R0B92Lk3AZp-
bWJSck2F7aGCiZ66aFGZrs2 registry.example.com:80
```

3.5. EXTENDED REGISTRY CONFIGURATION

3.5.1. Maintaining the Registry IP Address

OpenShift Container Platform refers to the integrated registry by its service IP address, so if you decide to delete and recreate the **docker-registry** service, you can ensure a completely transparent transition by arranging to re-use the old IP address in the new service. If a new IP address cannot be avoided, you can minimize cluster disruption by rebooting only the masters.

Re-using the Address

To re-use the IP address, you must save the IP address of the old **docker-registry** service prior to deleting it, and arrange to replace the newly assigned IP address with the saved one in the new **docker-registry** service.

1. Make a note of the **clusterIP** for the service:

```
$ oc get svc/docker-registry -o yaml | grep clusterIP:
```

2. Delete the service:

```
$ oc delete svc/docker-registry dc/docker-registry
```

3. Create the registry definition in ***registry.yaml***, replacing **<options>** with, for example, those used in step 3 of the instructions in the Non-Production Use section:

```
$ oadm registry <options> -o yaml > registry.yaml
```

4. Edit ***registry.yaml***, find the **`Service`** there, and change its **`clusterIP`** to the address noted in step 1.

5. Create the registry using the modified ***registry.yaml***:

```
$ oc create -f registry.yaml
```

Rebooting the Masters

If you are unable to re-use the IP address, any operation that uses a pull specification that includes the old IP address will fail. To minimize cluster disruption, you must reboot the masters:

```
# systemctl restart atomic-openshift-master
```

This ensures that the old registry URL, which includes the old IP address, is cleared from the cache.

Note

We recommend against rebooting the entire cluster because that incurs unnecessary downtime for pods and does not actually clear the cache.

3.5.2. Whitelisting Docker Registries

You can specify a whitelist of docker registries, allowing you to curate a set of images and templates that are available for download by OpenShift Container Platform users. This curated set can be placed in one or more docker registries, and then added to the whitelist. When using a whitelist, only the specified registries are accessible within OpenShift Container Platform, and all other registries are denied access by default.

To configure a whitelist:

1. Edit the ***/etc/sysconfig/docker*** file to block all registries:

```
BLOCK_REGISTRY='--block-registry=all'
```

You may need to uncomment the **`BLOCK_REGISTRY`** line.

2. In the same file, add registries to which you want to allow access:

```
ADD_REGISTRY='--add-registry=<registry1> --add-registry=
<registry2>'
```

Example 3.1. Allowing Access to Registries

```
ADD_REGISTRY='--add-registry=registry.access.redhat.com'
```

This example would restrict access to images available on the Red Hat Customer Portal.

Once the whitelist is configured, if a user tries to pull from a docker registry that is not on the whitelist, they will receive an error message stating that this registry is not allowed.

3.5.3. Overriding the Registry Configuration

You can override the integrated registry's default configuration, found by default at ***/config.yml*** in a running registry's container, with your own custom configuration.

Note

Upstream configuration options in this file may also be overridden using environment variables. The middleware section is an exception as there are just a few options that can be overridden using environment variables. Learn how to override specific configuration options.

To enable management of the registry configuration file directly and deploy an updated configuration, mount the configuration file as a secret volume:

1. Deploy the registry.
2. Edit the registry configuration file locally as needed. The initial YAML file deployed on the registry is provided below. Review supported options.

Example 3.2. Registry Configuration File

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /registry
  delete:
    enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
      options:
        acceptschema2: false
        pullthrough: true
        enforcequota: false
```

```
        projectcachettl: 1m
        blobrepositorycachettl: 10m
    storage:
      - name: openshift
```

3. Create a new secret called **registry-config** from your custom registry configuration file you edited locally:

```
$ oc secrets new registry-config config.yml=
</path/to/custom/registry/config.yml>
```

4. Add the **registry-config** secret as a volume to the registry's deployment configuration to mount the custom configuration file at ***/etc/docker/registry/***:

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

5. Update the registry to reference the configuration path from the previous step by adding the following environment variable to the registry's deployment configuration:

```
$ oc set env dc/docker-registry \
    REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yml
```

This may be performed as an iterative process to achieve the desired configuration. For example, during troubleshooting, the configuration may be temporarily updated to put it in **debug** mode.

To update an existing configuration:

Warning

This procedure will overwrite the currently deployed registry configuration.

1. Edit the local registry configuration file, ***config.yml***.
2. Delete the **registry-config** secret:

```
$ oc delete secret registry-config
```

3. Recreate the secret to reference the updated configuration file:

```
$ oc secrets new registry-config config.yml=
</path/to/custom/registry/config.yml>
```

4. Redeploy the registry to read the updated configuration:

```
$ oc rollout latest docker-registry
```

Tip

Maintain configuration files in a source control repository.

3.5.4. Registry Configuration Reference

There are many configuration options available in the upstream docker distribution library. Not all configuration options are supported or enabled. Use this section as a reference when overriding the registry configuration.

Note

Upstream configuration options in this file may also be overridden using environment variables. However, the middleware section may **not** be overridden using environment variables. Learn how to override specific configuration options.

3.5.4.1. Log

Upstream options are supported.

Example:

```
log:
  level: debug
  formatter: text
  fields:
    service: registry
    environment: staging
```

3.5.4.2. Hooks

Mail hooks are not supported.

3.5.4.3. Storage

The following storage drivers are supported:

- Filesystem
- S3. Learn more about CloudFront configuration.
- OpenStack Swift
- Google Cloud Storage (GCS)

General registry storage configuration options are supported.

Example 3.3. General Storage Configuration Options

```
storage:
```

```
    delete:
      enabled: true (1)
    redirect:
      disable: false
    cache:
      blobdescriptor: inmemory
    maintenance:
      uploadpurging:
        enabled: true
        age: 168h
        interval: 24h
        dryrun: false
      readonly:
        enabled: false
```

1

This entry is **mandatory** for image pruning to work properly.

3.5.4.4. Auth

Auth options should not be altered. The **openshift** extension is the only supported option.

```
auth:
  openshift:
    realm: openshift
```

3.5.4.5. Middleware

The **repository** middleware extension allows to configure OpenShift Container Platform middleware responsible for interaction with OpenShift Container Platform and image proxying.

```
middleware:
  registry:
    - name: openshift (1)
  repository:
    - name: openshift (2)
      options:
        acceptschema2: false (3)
        pullthrough: true (4)
        enforcequota: false (5)
        projectcachettl: 1m (6)
        blobrepositorycachettl: 10m (7)
  storage:
    - name: openshift (8)
```

1 2 8

These entries are mandatory. Their presence ensures required components get loaded. These values shouldn't be changed.

3

Allow to store manifest schema v2 during a push to the registry. See below for more details.

4

Let the registry act as a proxy for remote blobs. See below for more details.

5

Prevent blob uploads exceeding size limit defined in targeted project.

6

An expiration timeout for limits cached in the registry. The lower the value, the less time will it take for the limit changes to propagate to the registry. However, the registry will query limits from the server more frequently and, as a consequence, pushes will be slower.

7

An expiration timeout for remembered associations between blob and repository. The higher the value, the higher probability of fast lookup and more efficient registry operation. On the other hand, memory usage will raise as well as a risk of serving image layer to user, who is no longer authorized to access it.

3.5.4.5.1. CloudFront Middleware

The **CloudFront** middleware extension can be added to support AWS, CloudFront CDN storage provider. CloudFront middleware speeds up distribution of image content internationally. The blobs are distributed to several edge locations around the world. The client is always directed to the edge with the lowest latency.

Note

The **CloudFront** middleware extension can be only used with S3 storage. It is utilized only during blob serving. Therefore, only blob downloads can be speeded up, not uploads.

The following is an example of minimal configuration of S3 storage driver with a CloudFront middleware:

```
version: 0.1
log:
```

```
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  s3: 1
    accesskey: BJKMSZBRESWJQXRWMAEQ
    secretkey: 5ah5I91SNXbeoUXXDasFtadRqOdy62JzlnOW1goS
    region: us-east-1
    bucket: docker.myregistry.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
   storage:
    - name: cloudfront 2
      options:
        baseurl: https://jrpbyn0k5k88bi.cloudfront.net/ 3
        privatekey: /etc/docker/cloudfront-ABCEDFGHIJKLMNOPQRST.pem 4
        keypairid: ABCEDFGHIJKLMNOPQRST 5
    - name: openshift
```

1

The S3 storage must be configured the same way regardless of CloudFront middleware.

2

The CloudFront storage middleware needs to be listed before OpenShift middleware.

3

The CloudFront base URL. In the AWS management console, this is listed as **Domain Name** of CloudFront distribution.

4

The location of your AWS private key on the filesystem. This must be not confused with Amazon EC2 key pair. Please refer to AWS documentation on creating CloudFront key pairs for your trusted signers. The file needs to be mounted as a secret secret into the registry pod.

5

The ID of your Cloudfront key pair.

3.5.4.5.2. Overriding Middleware Configuration Options

The **middleware** section cannot be overridden using environment variables. There are a few exceptions, however. For example:

```
middleware:
  repository:
    - name: openshift
      options:
        acceptschema2: false 1
        enforcequota: false 2
        projectcachettl: 1m 3
        blobrepositorycachettl: 10m 4
```

1

A configuration option that can be overridden by the boolean environment variable **`REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ACCEPTSCHEMA2`**, which allows for the ability to accept manifest schema v2 on manifest put requests.

2

A configuration option that can be overridden by the boolean environment variable **`REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA`**, which allows the ability to turn quota enforcement on or off. By default, quota enforcement is off. It overrides OpenShift Container Platform middleware configuration option. Recognized values are **true** and **false**.

3

A configuration option that can be overridden by the environment variable **`REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PROJECTCACHETTL`**, specifying an eviction timeout for project quota objects. It takes a valid time duration string (for example, **2m**). If empty, you get the default timeout. If zero (**0m**), caching is disabled.

4

A configuration option that can be overriden by the environment variable **`REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_BLOBREPOSITORYCACHETTL`**, specifying an eviction timeout for associations between blob and containing repository. The format of the value is the same as in **`projectcachettl`** case.

3.5.4.5.3. Image Pullthrough

If enabled, the registry will attempt to fetch requested blob from a remote registry unless the blob exists locally. The remote candidates are calculated from **DockerImage** entries stored in status of the image stream, a client pulls from. All the unique remote registry references in such entries will be tried in turn until the blob is found. The blob, served this way, will not be stored in the registry.

This feature is on by default. However, it can be disabled using a configuration option.

3.5.4.5.4. Manifest Schema v2 Support

Each image has a manifest describing its blobs, instructions for running it and additional metadata. The manifest is versioned which have different structure and fields as it evolves over time. The same image can be represented by multiple manifest versions. Each version will have different digest though.

The registry currently supports manifest v2 schema 1 (**schema1**) and manifest v2 schema 2 (**schema2**). The former is being obsoleted but will be supported for an extended amount of time.

You should be wary of compatibility issues with various Docker clients:

- Docker clients of version 1.9 or older support only **schema1**. Any manifest this client pulls or pushes will be of this legacy schema.
- Docker clients of version 1.10 support both **schema1** and **schema2**. And by default, it will push the latter to the registry if it supports newer schema.

The registry, storing an image with **schema1** will always return it unchanged to the client. **Schema2** will be transferred unchanged only to newer Docker client. For the older one, it will be converted on-the-fly to **schema1**.

This has significant consequences. For example an image pushed to the registry by a newer Docker client cannot be pulled by the older Docker by its digest. That's because the stored image's manifest is of **schema2** and its digest can be used to pull only this version of manifest.

For this reason, the registry is configured by default not to store **schema2**. This ensures that any docker client will be able to pull from the registry any image pushed there regardless of client's version.

Once you're confident that all the registry clients support **schema2**, you'll be safe to enable its support in the registry. See the middleware configuration reference above for particular option.

3.5.4.6. Reporting

Reporting is unsupported.

3.5.4.7. HTTP

Upstream options are supported. Learn how to alter these settings via environment variables. Only the **tls** section should be altered. For example:

```
http:
  addr: :5000
  tls:
    certificate: /etc/secrets/registry.crt
    key: /etc/secrets/registry.key
```

3.5.4.8. Notifications

Upstream options are supported. The REST API Reference provides more comprehensive integration options.

Example:

```
notifications:
  endpoints:
    - name: registry
      disabled: false
      url: https://url:port/path
      headers:
        Accept:
          - text/plain
      timeout: 500
      threshold: 5
      backoff: 1000
```

3.5.4.9. Redis

Redis is not supported.

3.5.4.10. Health

Upstream options are supported. The registry deployment configuration provides an integrated health check at **/healthz**.

3.5.4.11. Proxy

Proxy configuration should not be enabled. This functionality is provided by the OpenShift Container Platform repository middleware extension, **pullthrough: true**.

3.6. KNOWN ISSUES

3.6.1. Overview

The following are the known issues when deploying or using the integrated registry.

3.6.2. Image Push Errors with Scaled Registry Using Shared NFS Volume

When using a scaled registry with a shared NFS volume, you may see one of the following errors during the push of an image:

- **digest invalid: provided digest did not match uploaded content**
- **blob upload unknown**
- **blob upload invalid**

These errors are returned by an internal registry service when Docker attempts to push the image. Its cause originates in the synchronization of file attributes across nodes. Factors such as NFS client side caching, network latency, and layer size can all contribute to potential errors that might occur when pushing an image using the default round-robin load balancing configuration.

You can perform the following steps to minimize the probability of such a failure:

1. Ensure that the **`sessionAffinity`** of your **docker-registry** service is set to **`ClientIP`**:

   ```
   $ oc get svc/docker-registry --
   template='{{.spec.sessionAffinity}}'
   ```

   This should return **`ClientIP`**, which is the default in recent OpenShift Container Platform versions. If not, change it:

   ```
   $ oc get -o yaml svc/docker-registry | \
         sed 's/\(sessionAffinity:\s*\).*/\1ClientIP/' | \
         oc replace -f -
   ```

2. Ensure that the NFS export line of your registry volume on your NFS server has the **`no_wdelay`** options listed. See Export Settings in the Persistent Storage Using NFS topic for details.

3.6.3. Pull of Internally Managed Image Fails with "not found" Error

This error occurs when the pulled image is pushed to an image stream different from the one it is being pulled from. This is caused by re-tagging a built image into an arbitrary image stream:

```
$ oc tag srcimagestream:latest anyproject/pullimagestream:latest
```

And subsequently pulling from it, using an image reference such as:

```
internal.registry.url:5000/anyproject/pullimagestream:latest
```

During a manual Docker pull, this will produce a similar error:

```
Error: image anyproject/pullimagestream:latest not found
```

To prevent this, avoid the tagging of internally managed images completely, or re-push the built image to the desired namespace manually.

3.6.4. Image Push Fails with "500 Internal Server Error" on S3 Storage

There are problems reported happening when the registry runs on S3 storage back-end. Pushing to a Docker registry occasionally fails with the following error:

```
Received unexpected HTTP status: 500 Internal Server Error
```

To debug this, you need to view the registry logs. In there, look for similar error messages occurring at the time of the failed push:

```
time="2016-03-30T15:01:21.22287816-04:00" level=error msg="unknown
error completing upload: driver.Error{DriverName:\"s3\", Enclosed:
(*url.Error)(0xc20901cea0)}" http.request.method=PUT
...
time="2016-03-30T15:01:21.493067808-04:00" level=error msg="response
completed with error" err.code=UNKNOWN err.detail="s3: Put
https://s3.amazonaws.com/oso-tsi-
docker/registry/docker/registry/v2/blobs/sha256/ab/abe5af443833d60cf672
e2ac57589410dddec060ed725d3e676f1865af63d2e2/data: EOF"
err.message="unknown error" http.request.method=PUT
...
time="2016-04-02T07:01:46.056520049-04:00" level=error msg="error
putting into main store: s3: The request signature we calculated does
not match the signature you provided. Check your key and signing
method." http.request.method=PUT
atest
```

If you see such errors, contact your Amazon S3 support. There may be a problem in your region or with your particular bucket.

3.6.5. Image Pruning Fails

If you encounter the following error when pruning images:

```
BLOB
sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c
error deleting blob
```

And your registry log contains the following information:

```
error deleting blob
\"sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e
6c\": operation unsupported
```

It means that your custom configuration file lacks mandatory entries in the storage section, namely **`storage:delete:enabled`** set to **true**. Add them, re-deploy the registry, and repeat your image pruning operation.

CHAPTER 4. SETTING UP A ROUTER

4.1. ROUTER OVERVIEW

4.1.1. About Routers

The OpenShift Container Platform router is the ingress point for all external traffic destined for services in your OpenShift Container Platform installation. OpenShift Container Platform 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 Container Platform. 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 Container Platform 3.0.2.

4.1.2. Router Service Account

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

Use of labels (e.g., to define router shards) requires **`cluster-reader`** permission.

```
$ oadm policy add-cluster-role-to-user \
    cluster-reader \
    system:serviceaccount:default:router
```

4.2. USING THE DEFAULT HAPROXY ROUTER

4.2.1. Overview

The **`oadm router`** command is provided with the administrator CLI to simplify the tasks of setting up routers in a new installation. If you followed the quick installation, then a default router was automatically created for you. The **`oadm router`** command creates the service and deployment configuration objects. Just about every form of communication between OpenShift Container Platform components is secured by TLS and uses various certificates and authentication methods. Use the **`--service-account`** option to specify the service account the router will 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.

Important

It is recommended to use separate distinct **openshift-router** service account with your router. This can be provided using the **`--service-account`** flag to the **`oadm router`** command.

```
$ oadm router --dry-run --service-account=router 1
```

1

`--service-account` is the name of a service account for the **openshift-router**.

Important

Router pods created using **`oadm router`** have default resource requests that a node must satisfy for the router pod to be deployed. In an effort to increase the reliability of infrastructure components, the default resource requests are used to increase the QoS tier of the router pods above pods without resource requests. The default values represent the observed minimum resources required for a basic router to be deployed and can be edited in the routers deployment configuration and you may want to increase them based on the load of the router.

4.2.2. Creating a Router

The quick installation process automatically creates a default router. If the router does not exist, run the following to create a router:

```
$ oadm router <router_name> --replicas=<number> --service-
account=router
```

You can also use router shards to ensure that the router is filtered to specific namespaces or routes, or set any environment variables after router creation.

4.2.3. Other Basic Router Commands

Checking the Default Router

The default router service account, named **router**, is automatically created during quick and advanced installations. To verify that this account already exists:

```
$ oadm router --dry-run --service-account=router
```

Viewing the Default Router

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

```
$ oadm router -o yaml --service-account=router
```

Deploying the Router to a Labeled Node

To deploy the router to any node(s) that match a specified node label:

```
$ oadm router <router_name> --replicas=<number> --selector=<label> \
    --service-account=router
```

For example, if you want to create a router named **`router`** and have it placed on a node labeled with **`region=infra`**:

```
$ oadm router router --replicas=1 --selector='region=infra' \
  --service-account=router
```

During advanced installation, the **`openshift_hosted_router_selector`** and **`openshift_registry_selector`** Ansible settings are set to **region=infra** by default. The default router and registry will only be automatically deployed if a node exists that matches the **region=infra** label.

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

Using a Different Router Image

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

```
$ oadm router <router_name> -o <format> --images=<image> \
    --service-account=router
```

For example:

```
$ oadm router region-west -o yaml --images=myrepo/somerouter:mytag \
    --service-account=router
```

4.2.4. Filtering Routes to Specific Routers

Using the **`ROUTE_LABELS`** environment variable, you can filter routes so that they are used only by specific routers.

For example, if you have multiple routers, and 100 routes, you can attach labels to the routes so that a portion of them are handled by one router, whereas the rest are handled by another.

1. After creating a router, use the **`ROUTE_LABELS`** environment variable to tag the router:

   ```
   $ oc env dc/<router=name>  ROUTE_LABELS="key=value"
   ```

2. Add the label to the desired routes:

```
oc label route <route=name> key=value
```

3. To verify that the label has been attached to the route, check the route configuration:

```
$ oc describe dc/<route_name>
```

4.2.5. Highly-Available Routers

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

4.2.6. Customizing the Router Service Ports

You can customize the service ports that a template router binds to by setting the environment variables **ROUTER_SERVICE_HTTP_PORT** and **ROUTER_SERVICE_HTTPS_PORT**. This can be done by creating a template router, then editing its deployment configuration.

The following example creates a router deployment with **0** replicas and customizes the router service HTTP and HTTPS ports, then scales it appropriately (to **1** replica).

```
$ oadm router --replicas=0 --ports='10080:10080,10443:10443' 1
$ oc set env dc/router ROUTER_SERVICE_HTTP_PORT=10080  \
                   ROUTER_SERVICE_HTTPS_PORT=10443
$ oc scale dc/router --replicas=1
```

1

Ensures exposed ports are appropriately set for routers that use the container networking mode **--host-network=false**.

Important

If you do customize the template router service ports, you will also need to ensure that the nodes where the router pods run have those custom ports opened in the firewall (either via Ansible or **iptables**, or any other custom method that you use via **firewall-cmd**).

The following is an example using **iptables** to open the custom router service ports.

```
$ iptables -A INPUT -p tcp --dport 10080 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 10443 -j ACCEPT
```

4.2.7. Working With Multiple Routers

An administrator can create multiple routers with the same definition to serve the same set of routes. By having different groups of routers with different namespace or route selectors, they can vary the routes that the router serves.

Multiple routers can be grouped to distribute routing load in the cluster and separate tenants to different routers or shards. Each router or shard in the group handles routes based on the selectors in the router. An administrator can create shards over the whole cluster using **ROUTE_LABELS**. A user can create shards over a namespace (project) by using **NAMESPACE_LABELS**.

4.2.8. Adding a Node Selector to a Deployment Configuration

Making specific routers deploy on specific nodes requires two steps:

1. Add a label to the desired node:

```
$ oc label node 10.254.254.28 "router=first"
```

2. Add a node selector to the router deployment configuration:

```
$ oc edit dc <deploymentConfigName>
```

Add the **template.spec.nodeSelector** field with a key and value corresponding to the label:

```
...
  template:
    metadata:
      creationTimestamp: null
      labels:
        router: router1
    spec:
      nodeSelector:      1
        router: "first"
...
```

1

The key and value are **router** and **first**, respectively, corresponding to the **router=first** label.

4.2.9. Using Router Shards

The access controls are based on the service account that the router is run with.

Using **NAMESPACE_LABELS** and/or **ROUTE_LABELS**, a router can filter out the namespaces and/or routes that it should service. This enables you to partition routes amongst multiple router deployments effectively distributing the set of routes.

Example: A router deployment **finops-router** is run with route selector **NAMESPACE_LABELS="name in (finance, ops)"** and a router deployment **dev-router** is run with route selector **NAMESPACE_LABELS="name=dev"**.

If all routes are in the 3 namespaces **finance**, **ops** or **dev**, then this could effectively distribute our routes across two router deployments.

In the above scenario, sharding becomes a special case of partitioning with no overlapping sets. Routes are divided amongst multiple router shards.

The criteria for route selection governs how the routes are distributed. It is possible to have routes that overlap accross multiple router deployments.

Example: In addition to the **finops-router** and **dev-router** in the example above, we also have an **devops-router** which is run with a route selector **NAMESPACE_LABELS="name in (dev, ops)"**.

The routes in namespaces **dev** or **ops** now are serviced by two different router deployments. This becomes a case where we have partitioned the routes with an overlapping set.

In addition, this enables us to create more complex routing rules ala divert high priority traffic to the dedicated **finops-router** but send the lower priority ones to the **devops-router**.

NAMESPACE_LABELS allows filtering the projects to service and selecting all the routes from those projects. But we may want to partition routes based on other criteria in the routes themselves. The **ROUTE_LABELS** selector allows you to slice-and-dice the routes themselves.

Example: A router deployment **prod-router** is run with route selector **ROUTE_LABELS="mydeployment=prod"** and a router deployment **devtest-router** is run with route selector **ROUTE_LABELS="mydeployment in (dev, test)"**

Example assumes you have all the routes you wish to serviced tagged with a label **"mydeployment=<tag>"**.

4.2.9.1. Creating Router Shards

Router sharding lets you select how routes are distributed among a set of routers.

Router sharding is based on labels; you set labels on the routes in the pool, and express the desired subset of those routes for the router to serve with a selection expression via the **oc set env** command.

First, ensure that service account associated with the router has the **cluster reader** permission.

The rest of this section describes an extended example. Suppose there are 26 routes, named **a** — **z**, in the pool, with various labels:

Possible labels on routes in the pool

```
sla=high        geo=east     hw=modest     dept=finance
sla=medium      geo=west     hw=strong     dept=dev
sla=low                                    dept=ops
```

These labels express the concepts: service level agreement, geographical location, hardware requirements, and department. The routes in the pool can have at most one label from each column. Some routes may have other labels, entirely, or none at all.

Name(s)	SLA	Geo	HW	Dept	Other Labels
a	high	east	modest	finance	type=static
b		west	strong		type=dynamic
c, d, e	low		modest		type=static
g — k	medium		strong	dev	
l — s	high		modest	ops	
t — z		west			type=dynamic

Here is a convenience script ***mkshard*** that ilustrates how **`oadm router`**, **`oc set env`**, and **`oc scale`** work together to make a router shard.

```
#!/bin/bash
# Usage: mkshard ID SELECTION-EXPRESSION
id=$1
sel="$2"
router=router-shard-$id                  1
oadm router $router --replicas=0         2
dc=dc/router-shard-$id                   3
oc set env   $dc ROUTE_LABELS="$sel"     4
oc scale $dc --replicas=3                5
```

1

The created router has name **`router-shard-<id>`**.

2

Specify no scaling for now.

3

The deployment configuration for the router.

4

Set the selection expression using **`oc set env`**. The selection expression is the value of the **`ROUTE_LABELS`** environment variable.

5

Scale it up.

Running ***mkshard*** several times creates several routers:

Router	Selection Expression	Routes
`router-shard-1`	`sla=high`	`a`, `l` — `s`
`router-shard-2`	`geo=west`	`b`, `t` — `z`
`router-shard-3`	`dept=dev`	`g` — `k`

4.2.9.2. Modifying Router Shards

Because a router shard is a construct based on labels, you can modify either the labels (via **`oc label`**) or the selection expression.

This section extends the example started in the Creating Router Shards section, demonstrating how to change the selection expression.

Here is a convenience script ***modshard*** that modifies an existing router to use a new selection expression:

```
#!/bin/bash
# Usage: modshard ID SELECTION-EXPRESSION...
id=$1
shift
router=router-shard-$id        1
dc=dc/$router                  2
oc scale $dc --replicas=0      3
oc set env   $dc "$@"             4
oc scale $dc --replicas=3      5
```

1

The modified router has name **`router-shard-<id>`**.

2

The deployment configuration where the modifications occur.

3

Scale it down.

4

Set the new selection expression using **`oc set env`**. Unlike **`mkshard`** from the Creating Router Shards section, the selection expression specified as the non-**`ID`** arguments to **`modshard`** must include the environment variable name as well as its value.

5

Scale it back up.

Note

In **`modshard`**, the **`oc scale`** commands are not necessary if the deployment strategy for **`router-dhsard-<id>`** is **`Rolling`**.

For example, to expand the department for **`router-shard-3`** to include **`ops`** as well as **`dev`**:

```
$ modshard 3 ROUTE_LABELS='dept in (dev, ops)'
```

The result is that **`router-shard-3`** now selects routes **`g`** — **`s`** (the combined sets of **`g`** — **`k`** and **`l`** — **`s`**).

This example takes into account that there are only three departments in this example scenario, and specifies a department to leave out of the shard, thus achieving the same result as the preceding example:

```
$ modshard 3 ROUTE_LABELS='dept != finanace'
```

This example specifies shows three comma-separated qualities, and results in only route **`b`** being selected:

```
$ modshard 3 ROUTE_LABELS='hw=strong,type=dynamic,geo=west'
```

Similarly to **`ROUTE_LABELS`**, which involve a route's labels, you can select routes based on the labels of the route's namespace labels, with the **`NAMESPACE_LABELS`** environment variable. This example modifies **`router-shard-3`** to serve routes whose namespace has the label **`frequency=weekly`**:

```
$ modshard 3 NAMESPACE_LABELS='frequency=weekly'
```

The last example combines **ROUTE_LABELS** and **NAMESPACE_LABELS** to select routes with label **sla=low** and whose namespace has the label **frequency=weekly**:

```
$ modshard 3 \
    NAMESPACE_LABELS='frequency=weekly' \
    ROUTE_LABELS='sla=low'
```

4.2.9.3. Using Namespace Router Shards

The routes for a project can be handled by a selected router by using **NAMESPACE_LABELS**. The router is given a selector for a **NAMESPACE_LABELS** label and the project that wants to use the router applies the **NAMESPACE_LABELS** label to its namespace.

First, ensure that service account associated with the router has the **cluster reader** permission. This permits the router to read the labels that are applied to the namespaces.

Now create and label the router:

```
$ oadm router ...  --service-account=router
$ oc set env dc/router NAMESPACE_LABELS="router=r1"
```

Because the router has a selector for a namespace, the router will handle routes for that namespace. So, for example:

```
$ oc label namespace default "router=r1"
```

Now create routes in the default namespace, and the route is available in the default router:

```
$ oc create -f route1.yaml
```

Now create a new project (namespace) and create a route, route2.

```
$ oc new-project p1
$ oc create -f route2.yaml
```

And notice the route is not available in your router. Now label namespace p1 with "router=r1"

```
$ oc label namespace p1 "router=r1"
```

Which makes the route available to the router.

Note that removing the label from the namespace won't have immediate effect (as we don't see the updates in the router), so if you redeploy/start a new router pod, you should see the unlabelled effects.

```
$ oc scale dc/router --replicas=0 && oc scale dc/router --replicas=1
```

4.2.10. Customizing the Default Routing Subdomain

You can customize the suffix used as the default routing subdomain for your environment by modifying the master configuration file (the ***/etc/origin/master/master-config.yaml*** file by default). Routes that do not specify a host name would have one generated using this default routing subdomain.

The following example shows how you can set the configured suffix to **v3.openshift.test**:

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

Note

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

With the OpenShift Container Platform master(s) running the above configuration, the generated host name for the example of a route named **no-route-hostname** without a host name added to a namespace **mynamespace** would be:

```
no-route-hostname-mynamespace.v3.openshift.test
```

4.2.11. Forcing Route Host Names to a Custom Routing Subdomain

If an administrator wants to restrict all routes to a specific routing subdomain, they can pass the **`--force-subdomain`** option to the **`oadm router`** command. This forces the router to override any host names specified in a route and generate one based on the template provided to the **`--force-subdomain`** option.

The following example runs a router, which overrides the route host names using a custom subdomain template **`${name}-${namespace}.apps.example.com`**.

```
$ oadm router --force-subdomain='${name}-${namespace}.apps.example.com'
```

4.2.12. 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 Container Platform CA to create the certificate. For example:

```
$ CA=/etc/origin/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 --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*.

4.2.13. 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 --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 using the above certificate and key.

```
$ oc create route edge --service=my-service \
    --hostname=www.example.test \
    --key=example-test.key --cert=example-test.crt
route "my-service" created
```

Look at its definition.

```
$ oc get route/my-service -o yaml
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
```

```
host: www.example.test
to:
  kind: Service
  name: my-service
tls:
  termination: edge
  key: |
    -----BEGIN PRIVATE KEY-----
    [...]
    -----END PRIVATE KEY-----
  certificate: |
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
```

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/
```

4.2.14. Using Wildcard Routes (for a Subdomain)

The HAProxy router has support for wildcard routes, which are enabled by setting the **`ROUTER_ALLOW_WILDCARD_ROUTES`** environment variable to **`true`**. Any routes with a wildcard policy of **`Subdomain`** that pass the router admission checks will be serviced by the HAProxy router. Then, the HAProxy router exposes the associated service (for the route) per the route's wildcard policy.

```
$ oadm router --replicas=0 ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

Using a Secure Wildcard Edge Terminated Route

This example reflects TLS termination occurring on the router before traffic is proxied to the destination. Traffic sent to any hosts in the subdomain **`example.org`** (**`*.example.org`**) is proxied to the exposed service.

The secure edge terminated route specifies the TLS certificate and key information. The TLS certificate is served by the router front end for all hosts that match the subdomain (**`*.example.org`**).

1. Start up a router instance:

   ```
   $ oadm router --replicas=0 --service-account=router
   $ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
   $ oc scale dc/router --replicas=1
   ```

2. Create a private key, certificate signing request (CSR), and certificate for the edge secured route.

   The instructions on how to do this are specific to your certificate authority and provider. For a simple self-signed certificate for a domain named **`*.example.test`**, see this example:

```
# 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=*.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
      -signkey example-test.key -out example-test.crt
```

3. Generate a wildcard route using the above certificate and key:

```
$ cat > route.yaml  <<REOF
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
  host: www.example.test
  wildcardPolicy: Subdomain
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: "$(perl -pe 's/\n/\\n/' example-test.key)"
    certificate: "$(perl -pe 's/\n/\\n/' example-test.cert)"
REOF
$ oc create -f route.yaml
```

   Ensure your DNS entry for **`*.example.test`** points to your router instance(s) and the route to your domain is available.

   This example uses **`curl`** 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:$routerip
https://www.example.test/
# curl -k --resolve abc.example.test:443:$routerip
https://abc.example.test/
# curl -k --resolve anyname.example.test:443:$routerip
https://anyname.example.test/
```

For routers that allow wildcard routes (**`ROUTER_ALLOW_WILDCARD_ROUTES`** set to **`true`**), there are some caveats to the ownership of a subdomain associated with a wildcard route.

Prior to wildcard routes, ownership was based on the claims made for a host name with the namespace with the oldest route winning against any other claimants. For example, route **`r1`** in namespace **`ns1`** with a claim for **`one.example.test`** would win over another route **`r2`** in

namespace **ns2** for the same host name **one.example.test** if route **r1** was older than route **r2**.

In addition, routes in other namespaces were allowed to claim non-overlapping hostnames. For example, route **rone** in namespace **ns1** could claim **www.example.test** and another route **rtwo** in namespace **d2** could claim **c3po.example.test**.

This is still the case if there are *no* wildcard routes claiming that same subdomain (**example.test** in the above example).

However, a wildcard route needs to claim all of the host names within a subdomain (host names of the form ***.example.test**). A wildcard route's claim is allowed or denied based on whether or not the oldest route for that subdomain (**example.test**) is in the same namespace as the wildcard route. The oldest route can be either a regular route or a wildcard route.

For example, if there is already a route **eldest** that exists in the **ns1** namespace that claimed a host named **owner.example.test** and, if at a later point in time, a new wildcard route **wildthing** requesting for routes in that subdomain (**example.test**) is added, the claim by the wildcard route will *only* be allowed if it is the same namespace (**ns1**) as the owning route.

The following examples illustrate various scenarios in which claims for wildcard routes will succeed or fail.

In the example below, a router that allows wildcard routes will allow non-overlapping claims for hosts in the subdomain **example.test** as long as a wildcard route has not claimed a subdomain.

```
$ oadm router ...
$ oc set env dc/router
$ oc project ns1 ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test
$ oc expose service myservice --hostname=bname.example.test

$ oc project ns2
$ oc expose service anotherservice --hostname=second.example.test
$ oc expose service anotherservice --hostname=cname.example.test

$ oc project otherns
$ oc expose service thirdservice --hostname=emmy.example.test
$ oc expose service thirdservice --hostname=webby.example.test
```

In the example below, a router that allows wildcard routes will not allow the claim for **owner.example.test** or **aname.example.test** to succeed since the owning namespace is **ns1**.

```
$ oadm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test

$ oc project ns2
$ oc expose service secondservice --hostname=bname.example.test
$ oc expose service secondservice --hostname=cname.example.test
```

```
$ # Router will not allow this claim with a different path name `/p1`
as
$ # namespace `ns1` has an older route claiming host
`aname.example.test`.
$ oc expose service secondservice --hostname=aname.example.test --
path="/p1"

$ # Router will not allow this claim as namespace `ns1` has an older
route
$ # claiming host name `owner.example.test`.
$ oc expose service secondservice --hostname=owner.example.test

$ oc project otherns

$ # Router will not allow this claim as namespace `ns1` has an older
route
$ # claiming host name `aname.example.test`.
$ oc expose service thirdservice --hostname=aname.example.test
```

In the example below, a router that allows wildcard routes will allow the claim for **`\*.example.test`** to succeed since the owning namespace is **ns1** and the wildcard route belongs to that same namespace.

```
$ oadm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  router will allow this claim.
```

In the example below, a router that allows wildcard routes will not allow the claim for **`\*.example.test`** to succeed since the owning namespace is **ns1** and the wildcard route belongs to another namespace **cyclone**.

```
$ oadm router ...
$ oc set env dc/router
$ oc project ns1 ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Switch to a different namespace/project.
$ oc project cyclone

$ # Reusing the route.yaml from a prior example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain
```

```
$ oc create -f route.yaml   #  router will deny (_NOT_ allow) this
claim.
```

Similarly, once a namespace with a wildcard route claims a subdomain, only routes within that namespace can claim any hosts in that same subdomain.

In the example below, once a route in namespace **ns1** with a wildcard route claims subdomain **example.test**, only routes in the namespace **ns1** are allowed to claim any hosts in that same subdomain.

```
$ oadm router ...
$ oc set env dc/router
$ oc project ns1 ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service otherservice --hostname=other.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to
host
$ #  `other.example.test` due to the wildcard route claiming the
subdomain.

$ # namespace `ns1` is allowed to claim for deux.example.test
$ oc expose service mysecondservice --hostname=deux.example.test

$ # namespace `ns1` is allowed to claim for deux.example.test with path
/p1
$ oc expose service mythirdservice --hostname=deux.example.test --
path="/p1"

$ oc project otherns

$ # namespace `otherns` is not allowed to claim for deux.example.test
$ # with a different path '/otherpath'
$ oc expose service otherservice --hostname=deux.example.test --
path="/otherpath"

$ # namespace `otherns` is not allowed to claim for owner.example.test
$ oc expose service yetanotherservice --hostname=owner.example.test
```

```
$ # namespace `otherns` is not allowed to claim for
unclaimed.example.test
$ oc expose service yetanotherservice --hostname=unclaimed.example.test
```

In the example below, different scenarios are shown, in which the owner routes are deleted and ownership is passed within and across namespaces. While a route claiming host **`eldest.example.test`** in the namespace **`ns1`** exists, wildcard routes in that namespace can claim subdomain **`example.test`**. When the route for host **`eldest.example.test`** is deleted, the next oldest route **`senior.example.test`** would become the oldest route and would not affect any other routes. Once the route for host **`senior.example.test`** is deleted, the next oldest route **`junior.example.test`** becomes the oldest route and block the wildcard route claimant.

```
$ oadm router ...
$ oc set env dc/router
$ oc project ns1 ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=eldest.example.test
$ oc expose service seniorservice --hostname=senior.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service juniorservice --hostname=junior.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to
host
$ #  `junior.example.test` due to the wildcard route claiming the
subdomain.

$ # namespace `ns1` is allowed to claim for dos.example.test
$ oc expose service mysecondservice --hostname=dos.example.test

$ # Delete route for host `eldest.example.test`, the next oldest route
is
$ # the one claiming `senior.example.test`, so route claims are
unaffacted.
$ oc delete route myservice

$ # Delete route for host `senior.example.test`, the next oldest route
is
$ # the one claiming `junior.example.test` in another namespace, so
claims
$ # for a wildcard route would be affected. The route for the host
```

```
$ # `dos.example.test` would be unaffected as there are no other
wildcard
$ # claimants blocking it.
$ oc delete route seniorservice
```

4.2.15. Using the Container Network Stack

The OpenShift Container Platform router runs inside a 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 --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.

Note

On OpenShift Container Platform clusters using multi-tenant network isolation, routers on a non-default namespace with the **`--host-network=false`** option will load all routes in the cluster, but routes across the namespaces will not be reachable due to network isolation. With the **`--host-network=true`** option, routes bypass the container network and it can access any pod in the cluster. If isolation is needed in this case, then do not add routes across the namespaces.

4.2.16. Exposing Router Metrics

Using the **`--metrics-image`** and **`--expose-metrics`** options, you can configure the OpenShift Container Platform router to run a sidecar container that exposes or publishes router metrics for consumption by external metrics collection and aggregation systems (e.g. Prometheus, statsd).

Depending on your router implementation, the image is appropriately set up and the metrics sidecar container is started when the router is deployed. For example, the HAProxy-based router implementation defaults to using the **`prom/haproxy-exporter`** image to run as a sidecar container, which can then be used as a metrics datasource by the Prometheus server.

Note

The **`--metrics-image`** option overrides the defaults for HAProxy-based router implementations and, in the case of custom implementations, enables the image to use for a custom metrics exporter or publisher.

1. Grab the HAProxy Prometheus exporter image from the Docker registry:

```
$ sudo docker pull prom/haproxy-exporter
```

2. Create the OpenShift Container Platform router:

```
$ oadm router --service-account=router --expose-metrics
```

Or, optionally, use the **`--metrics-image`** option to override the HAProxy defaults:

```
$ oadm router --service-account=router --expose-metrics \
    --metrics-image=prom/haproxy-exporter
```

3. Once the haproxy-exporter containers (and your HAProxy router) have started, point Prometheus to the sidecar container on port 9101 on the node where the haproxy-exporter container is running:

```
$ haproxy_exporter_ip="<enter-ip-address-or-hostname>"
$ cat > haproxy-scraper.yml  <<CFGEOF
---
global:
  scrape_interval: "60s"
  scrape_timeout:  "10s"
  # external_labels:
    # source: openshift-router

scrape_configs:
  - job_name:  "haproxy"
    target_groups:
      - targets:
        - "${haproxy_exporter_ip}:9101"
CFGEOF

$ #  And start prometheus as you would normally using the above
config file.
$ echo "  - Example:  prometheus -config.file=haproxy-scraper.yml
"
$ echo "              or you can start it as a container on
{product-title}!!

$ echo "  - Once the prometheus server is up, view the {product-
title} HAProxy "
$ echo "    router metrics at:
http://<ip>:9090/consoles/haproxy.html "
```

4.2.17. Preventing Connection Failures During Restarts

If you connect to the router while the proxy is reloading, there is a small chance that your connection will end up in the wrong network queue and be dropped. The issue is being addressed. In the meantime, it is possible to work around the problem by installing **iptables** rules to prevent connections during the reload window. However, doing so means that the router needs to run with elevated privilege so that it can manipulate **iptables** on the host. It also means that connections that happen during the reload are temporarily ignored and must retransmit their connection start, lengthening the time it takes to connect, but preventing connection failure.

To prevent this, configure the router to use **iptables** by changing the service account, and setting an environment variable on the router.

Use a Privileged SCC

When creating the router, allow it to use the privileged SCC. This gives the router user the ability to create containers with root privileges on the nodes:

```
$ oadm policy add-scc-to-user privileged -z router
```

Patch the Router Deployment Configuration to Create a Privileged Container

You can now create privileged containers. Next, configure the router deployment configuration to use the privilege so that the router can set the iptables rules it needs. This patch changes the router deployment configuration so that the container that is created runs as privileged (and therefore gets correct capabilities) and run as root:

```
$ oc patch dc router -p '{"spec":{"template":{"spec":{"containers":
[{"name":"router","securityContext":
{"privileged":true}}],"securityContext":{"runAsUser": 0}}}}}'
```

Configure the Router to Use iptables

Set the option on the router deployment configuration:

```
$ oc set env dc/router -c router DROP_SYN_DURING_RESTART=true
```

If you used a non-default name for the router, you must change ***dc/router*** accordingly.

4.2.18. ARP Cache Tuning for Large-scale Clusters

In OpenShift Container Platform clusters with large numbers of routes (greater than the value of **net.ipv4.neigh.default.gc_thresh3**, which is **1024** by default), you must increase the default values of sysctl variables to allow more entries in the ARP cache.

The kernel messages would be similar to the following:

```
[ 1738.811139] net_ratelimit: 1045 callbacks suppressed
[ 1743.823136] net_ratelimit: 293 callbacks suppressed
```

When this issue occurs, the **oc** commands might start to fail with the following error:

```
Unable to connect to the server: dial tcp: lookup <hostname> on <ip>:
<port>: write udp <ip>:<port>-><ip>:<port>: write: invalid argument
```

To verify the actual amount of ARP entries for IPv4, run the following:

```
# ip -4 neigh show nud all | wc -l
```

If the number begins to approach the **net.ipv4.neigh.default.gc_thresh3** threshold, increase the values. Run the following on the nodes running the router pods. The following sysctl values are suggested for clusters with large numbers of routes:

```
net.ipv4.neigh.default.gc_thresh1 = 8192
net.ipv4.neigh.default.gc_thresh2 = 32768
net.ipv4.neigh.default.gc_thresh3 = 65536
```

To make these settings permanent across reboots, create a custom tuned profile.

4.2.19. Protecting Against DDoS Attacks

Add **timeout http-request** to the default HAProxy router image to protect the deployment against distributed denial-of-service (DDoS) attacks (for example, slowloris):

```
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s 1
  timeout server 10s
  timeout client 30s
```

1

timeout http-request is set up to 5 seconds. HAProxy gives a client 5 seconds *to send its whole HTTP request. Otherwise, HAProxy shuts the connection with *an error.

Also, when the environment variable **ROUTER_SLOWLORIS_TIMEOUT** is set, it limits the amount of time a client has to send the whole HTTP request. Otherwise, HAProxy will shut down the connection.

Setting the environment variable allows information to be captured as part of the router's deployment configuration and does not require manual modification of the template, whereas manually adding the HAProxy setting requires you to rebuild the router pod and maintain your router template file.

Using annotations implements basic DDoS protections in the HAProxy template router, including the ability to limit the:

- number of concurrent TCP connections
- rate at which a client can request TCP connections
- rate at which HTTP requests can be made

These are enabled on a per route basis because applications can have extremely different traffic patterns.

Table 4.1. HAProxy Template Router Settings

Setting	Description
haproxy.router.openshift.io/rate-limit-connections	Enables the settings be configured (when set to **true**, for example).
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	The number of concurrent TCP connections that can be made by the same IP address on this route.
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	The number of TCP connections that can be opened by a client IP.
haproxy.router.openshift.io/rate-limit-connections.rate-http	The number of HTTP requests that a client IP can make in a 3-second period.

4.3. DEPLOYING A CUSTOMIZED HAPROXY ROUTER

4.3.1. Overview

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 container image, and run a customized router. Alternatively you can use a ConfigMap.

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.

4.3.2. Obtaining the Router Configuration Template

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.

4.3.3. Using a ConfigMap to Replace the Router Configuration Template

You can use ConfigMap to customize the router instance without rebuilding the router image. The ***haproxy-config.template***, ***reload-haproxy***, and other scripts can be modified as well as creating and modifying router environment variables.

1. Copy the ***haproxy-config.template*** that you want to modify as described above. Modify it as desired.

2. Create a ConfigMap:

```
$ oc create configmap customrouter --from-file=haproxy-
config.template
```

   The **`customrouter`** ConfigMap now contains a copy of the modified ***haproxy-config.template*** file.

3. Modify the router deployment configuration to mount the ConfigMap as a file and point the **`TEMPLATE_FILE`** environment variable to it. This can be done via **`oc set env`** and **`oc volume`** commands, or alternatively by editing the router deployment configuration.

   Using oc commands

```
$ oc set env dc/router \
    TEMPLATE_FILE=/var/lib/haproxy/conf/custom/haproxy-
config.template
$ oc volume dc/router --add --overwrite \
    --name=config-volume \
    --mount-path=/var/lib/haproxy/conf/custom \
    --source='{"configMap": { "name": "customrouter"}}'
```

   Editing the Router Deployment Configuration

   Use **`oc edit dc router`** to edit the router deployment configuration with a text editor.

```
...
        - name: STATS_USERNAME
          value: admin
        - name: TEMPLATE_FILE  1
          value: /var/lib/haproxy/conf/custom/haproxy-
config.template
        image: openshift/origin-haproxy-routerp
...
        terminationMessagePath: /dev/termination-log
        volumeMounts: 2
        - mountPath: /var/lib/haproxy/conf/custom
          name: config-volume
      dnsPolicy: ClusterFirst
...
      terminationGracePeriodSeconds: 30
      volumes: 3
      - configMap:
```

```
          name: customrouter
        name: config-volume
  test: false
...
```

1

In the **spec.container.env** field, add the **TEMPLATE_FILE** environment variable to point to the mounted ***haproxy-config.template*** file.

2

Add the **spec.container.volumeMounts** field to create the mount point.

3

Add a new **spec.volumes** field to mention the ConfigMap.

Save the changes and exit the editor. This restarts the router.

4.3.4. 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.

4.3.5. 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 container 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.

4.4. USING THE F5 ROUTER PLUG-IN

4.4. USING THE F5 ROUTER PLUG-IN

4.4.1. Overview

Note

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

The F5 router plug-in is provided as a container image and run as a pod, just like the default HAProxy router.

4.4.2. Prerequisites

When deploying the F5 router plug-in, ensure you meet the following requirements:

1. An F5 host IP and credentials
2. The name of the virtual servers (for both HTTP and HTTPS)
3. The private key to the F5 instance
4. The host-internal IP and VXLAN gateway

4.4.3. Deploying the F5 Router

Important

The F5 router must be run in privileged mode, because route certificates are copied using the **scp** command:

```
$ oadm policy remove-scc-from-user hostnetwork -z router
$ oadm policy add-scc-to-user privileged -z router
```

Deploy the F5 router with the **oadm router** command, but provide additional flags (or environment variables) specifying 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.

Flag	Description
--external-host-username	Specifies the **F5 BIG-IP®** user name (typically **admin**).
--external-host-password	Specifies the **F5 BIG-IP®** password.
--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.
--external-host-partition-path	Specifies the **F5 BIG-IP®** partition path (the default is **/Common**).

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 \
    --service-account=router
```

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 watching routes, endpoints, and nodes 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.

4.4.4. F5 Router Partition Paths

Partition paths allow you to store your OpenShift Container Platform 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 Container Platform-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 ...
```

4.4.5. Setting Up F5 Native Integration

Note

This section reviews how to set up F5 native integration with OpenShift Container Platform. The concepts of F5 appliance and OpenShift Container Platform connection and data flow of F5 native integration are discussed in the F5 Native Integration section of the Routes topic.

As of OpenShift Container Platform version 3.4, using native integration of F5 with OpenShift Container Platform does not require configuring a ramp node for F5 to be able to reach the pods on the overlay network as created by OpenShift SDN.

The F5 controller pod needs to be launched with enough information so that it can successfully directly connect to pods.

1. Create a ghost **`hostsubnet`** on the OpenShift Container Platform cluster:

```
$ cat > f5-hostsubnet.yaml << EOF
{
    "kind": "HostSubnet",
    "apiVersion": "v1",
    "metadata": {
        "name": "openshift-f5-node",
        "annotations": {
        "pod.network.openshift.io/assign-subnet": "true"
        }
    },
```

```
    "host": "openshift-f5-node",
    "hostIP": "10.3.89.213"  # <- internal IP of the F5 appliance
} EOF
$ oc create -f f5-hostsubnet.yaml
```

2. Determine the subnet allocated for the ghost **hostsubnet** just created:

```
$ oc get hostsubnets
NAME                   HOST                   HOST IP
SUBNET
openshift-f5-node      openshift-f5-node      10.3.89.213
10.131.0.0/23
openshift-master-node  openshift-master-node  172.17.0.2
10.129.0.0/23
openshift-node-1       openshift-node-1       172.17.0.3
10.128.0.0/23
openshift-node-2       openshift-node-2       172.17.0.4
10.130.0.0/23
```

3. Check the **SUBNET** for the newly created **hostsubnet**. In this example, **10.131.0.0/23**.

4. Get the entire pod network's CIDR:

```
$ oc get clusternetwork
```

   This value will be something like **10.128.0.0/14**, noting the mask (**14** in this example).

5. To construct the gateway address, pick any IP address from the **hostsubnet** (for example, **10.131.0.5**). Use the mask of the pod network (**14**). The gateway address becomes: **10.131.0.5/14**.

6. Launch the F5 controller pod, following these instructions. However, use the two new additional options for VXLAN native integration:

```
$ #--external-host-internal-ip=10.3.89.213
$ #--external-host-vxlan-gw=10.131.0.5/14
$ oadm router \
    --type=f5-router \
    --external-host=10.3.89.90 \
    --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' \
    --service-account=router \
   --external-host-internal-ip=10.3.89.213 \
   --external-host-vxlan-gw=10.131.0.5/14
```

   The F5 set up is now ready, without the need to set up the ramp node.

CHAPTER 5. UPGRADING A CLUSTER

5.1. OVERVIEW

When new versions of OpenShift Container Platform are released, you can upgrade your existing cluster to apply the latest enhancements and bug fixes. This includes upgrading from previous minor versions, such as release 3.2 to 3.3, and applying asynchronous errata updates within a minor version (3.3.z releases). See the OpenShift Container Platform 3.4 Release Notes to review the latest changes.

Note

Due to the core architectural changes between the major versions, OpenShift Enterprise 2 environments cannot be upgraded to OpenShift Container Platform 3 and require a fresh installation.

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.

5.1.1. In-place or Blue-Green Upgrades

There are two methods for performing OpenShift Container Platform cluster upgrades. You can either do in-place upgrades (automated or manual), or upgrade using a blue-green deployment method.

In-place Upgrades

With in-place upgrades, the cluster upgrade is performed on all hosts in a single, running cluster: first masters and then nodes. Pods are evacuated off of nodes and recreated on other running nodes before a node upgrade begins; this helps reduce downtime of user applications.

If you installed using the quick or advanced installation and the ***~/.config/openshift/installer.cfg.yml*** or inventory file that was used is available, you can perform an automated in-place upgrade. Alternatively, you can upgrade in-place manually.

Blue-green Deployments

The blue-green deployment upgrade method follows a similar flow to the in-place method: masters and etcd servers are still upgraded first, however a parallel environment is created for new nodes instead of upgrading them in-place.

This method allows administrators to switch traffic from the old set of nodes (e.g., the "blue" deployment) to the new set (e.g., the "green" deployment) after the new deployment has been verified. If a problem is detected, it is also then easy to rollback to the old deployment quickly.

5.2. PERFORMING AUTOMATED IN-PLACE CLUSTER UPGRADES

5.2.1. Overview

Important

An etcd performance issue has been discovered on new and upgraded OpenShift Container Platform 3.4 clusters. See the following Knowledgebase Solution for further details:

https://access.redhat.com/solutions/2916381
(**BZ#1415839**)

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 OpenShift cluster upgrade process. If you installed using the quick installation method and a ***~/.config/openshift/installer.cfg.yml*** file is available, you can use the installer to perform the automated upgrade.

The automated upgrade performs the following steps for you:

- Applies the latest configuration.
- Upgrades master and etcd components and restarts services.
- Upgrades node components and restarts 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

Ensure that you have met all prerequisites before proceeding with an upgrade. Failure to do so can result in a failed upgrade.

5.2.2. Preparing for an Automated Upgrade

Important

Before upgrading your cluster to OpenShift Container Platform 3.4, the cluster must be already upgraded to the latest asynchronous release of version 3.3. Cluster upgrades cannot span more than one minor version at a time, so if your cluster is at a version earlier than 3.3, you must first upgrade incrementally (e.g., 3.1 to 3.2, then 3.2 to 3.3).

To prepare for an automated upgrade:

1. If you are upgrading from OpenShift Container Platform 3.3 to 3.4, manually disable the 3.3 channel and enable the 3.4 channel on each master and node host:

```
# subscription-manager repos --disable="rhel-7-server-ose-3.3-
rpms" \
    --enable="rhel-7-server-ose-3.4-rpms"\
```

```
    --enable="rhel-7-server-extras-rpms"
# yum clean all
```

2. For any upgrade path, always ensure that you have the latest version of the **atomic-openshift-utils** package on each RHEL 7 system, which also updates the **openshift-ansible-*** packages:

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

3. Install or update to the following latest available ***-excluder** packages on each RHEL 7 system, which helps ensure your systems stay on the correct versions of **atomic-openshift** and **docker** packages when you are not trying to upgrade, according to the OpenShift Container Platform version:

```
# yum install atomic-openshift-excluder atomic-openshift-docker-
excluder
```

   These packages add entries to the **`exclude`** directive in the host's ***/etc/yum.conf*** file.

4. You must be logged in as a cluster administrative user on the master host for the upgrade to succeed:

```
$ oc login
```

After satisfying these steps, there are two methods for running the automated upgrade:

- Using the installer
- Running the upgrade playbook directly

Choose and follow one of these methods.

5.2.3. Using the Installer to Upgrade

If you installed OpenShift Container Platform using the quick installation method, you should have an installation configuration file located at ***~/.config/openshift/installer.cfg.yml***. The installer requires this file to start an upgrade.

The installer supports upgrading between minor versions of OpenShift Container Platform (one minor version at a time, e.g., 3.3 to 3.4) as well as between asynchronous errata updates within a minor version (e.g., 3.4.z).

If you have an older format installation configuration file in ***~/.config/openshift/installer.cfg.yml*** from an installation of a previous cluster version, the installer will attempt to upgrade the file to the new supported format. If you do not have an installation configuration file of any format, you can create one manually.

To start an upgrade with the quick installer:

1. Satisfy the steps in Preparing for an Automated Upgrade to ensure you are using the latest upgrade playbooks.
2. Run the following command on each host to remove the **atomic-openshift** packages from the list of yum excludes on the host:

```
# atomic-openshift-excluder unexclude
```

3. Run the installer with the **`upgrade`** subcommand:

```
# atomic-openshift-installer upgrade
```

4. Then, follow the on-screen instructions to upgrade to the latest release.

5. Run the following command on each host to add the **atomic-openshift** packages back to the list of yum excludes on the host:

```
# atomic-openshift-excluder exclude
```

6. After all master and node upgrades have completed, a recommendation will be printed to reboot all hosts. After rebooting, if there are no additional features enabled, you can verify the upgrade. Otherwise, the next step depends on what additional features have you previously enabled.

Feature	Next Step
Aggregated Logging	Upgrade the EFK logging stack.
Cluster Metrics	Upgrade cluster metrics.

5.2.4. Running Upgrade Playbooks Directly

You can run automated upgrade playbooks using Ansible directly, similar to the advanced installation method, if you have an inventory file. Playbooks can be run using the **`ansible-playbook`** command.

The same ***v3_4*** upgrade playbooks can be used for either of the following scenarios:

- Upgrading existing OpenShift Container Platform 3.3 clusters to 3.4
- Upgrading existing OpenShift Container Platform 3.4 clusters to the latest asynchronous errata updates

5.2.4.1. Upgrading the Control Plane and Nodes in Separate Phases

An OpenShift Container Platform cluster can be upgraded in one or more phases. You can choose whether to upgrade all hosts in one phase by running a single Ansible playbook, or upgrade the *control plane* (master components) and nodes in multiple phases using separate playbooks.

Note

Instructions on the full upgrade process and when to call these playbooks are described in Upgrading to the Latest OpenShift Container Platform 3.4 Release.

When upgrading in separate phases, the control plane phase includes upgrading:

- etcd
- master components
- Docker on any stand-alone etcd hosts

It does does not include:

- node services running on masters
- Docker running on masters
- node services running on stand-alone nodes

When upgrading only the nodes, the control plane must already be upgraded. The node phase includes upgrading:

- node services running on masters and stand-alone nodes
- Docker running on masters and nodes

5.2.4.2. Customizing Node Upgrades

Whether upgrading in a single or multiple phases, you can customize how the node portion of the upgrade progresses by passing certain Ansible variables to an upgrade playbook using the **`-e`** option.

Note

Instructions on the full upgrade process and when to call these playbooks are described in Upgrading to the Latest OpenShift Container Platform 3.4 Release.

The **`openshift_upgrade_nodes_serial`** variable can be set to an integer or percentage to control how many node hosts are upgraded at the same time. The default is **`1`**, upgrading nodes one at a time.

For example, to upgrade 20 percent of the total number of detected nodes at a time:

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial="20%"
```

The **`openshift_upgrade_nodes_label`** variable allows you to specify that only nodes with a certain label are upgraded. This can also be combined with the **`openshift_upgrade_nodes_serial`** variable.

For example, to only upgrade nodes in the **group1** region, two at a time:

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial="2" \
    -e openshift_upgrade_nodes_label="region=group1"
```

See Manging Nodes for more on node labels.

5.2.4.3. Upgrading to the Latest OpenShift Container Platform 3.4 Release

To upgrade an existing OpenShift Container Platform 3.3 or 3.4 cluster to the latest 3.4 release:

1. Satisfy the steps in Preparing for an Automated Upgrade to ensure you are using the latest upgrade playbooks.
2. Run the following command on each host to remove the **atomic-openshift** packages from the list of yum excludes on the host:

```
# atomic-openshift-excluder unexclude
```

3. Ensure the **`deployment_type`** parameter in your inventory file is set to **`openshift-enterprise`**.
4. If you have multiple masters configured and want to enable rolling, full system restarts of the hosts, you can set the **`openshift_rolling_restart_mode`** parameter in your inventory file to **`system`**. Otherwise, the default value **`services`** performs rolling service restarts on HA masters, but does not reboot the systems. See Configuring Cluster Variables for details.
5. At this point, you can choose to run the upgrade in a single or multiple phases. See Upgrading the Control Plane and Nodes in Separate Phases for more details which components are upgraded in each phase.

   If your inventory file is located somewhere other than the default ***/etc/ansible/hosts***, add the **`-i`** flag to specify its location. If you previously used the **`atomic-openshift-installer`** command to run your installation, you can check ***~/.config/openshift/hosts*** for the last inventory file that was used, if needed.

Note

You can add **`--tags pre_upgrade`** to the following **`ansible-playbook`** commands to run the pre-upgrade checks for the playbook. This is a dry-run option that preforms all pre-upgrade checks without actually upgrading any hosts, and reports any problems found.

Option A) Upgrade control plane and nodes in a single phase.

Run the ***upgrade.yml*** playbook to upgrade the cluster in a single phase using one playbook; the control plane is still upgraded first, then nodes in-place:

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
cluster/upgrades/v3_4/upgrade.yml \
    [-e <customized_node_upgrade_variables>] 1
```

1

See Customizing Node Upgrades for any desired **`<customized_node_upgrade_variables>`**.

Option B) Upgrade the control plane and nodes in separate phases.

a. To upgrade only the control plane, run the ***upgrade_control_plane.yaml*** playbook:

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-
cluster/upgrades/v3_4/upgrade_control_plane.yml
```

b. To upgrade only the nodes, run the ***upgrade_nodes.yaml*** playbook:

```
# ansible-playbook -i </path/to/inventory/file> \
    /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-
cluster/upgrades/v3_4/upgrade_nodes.yml \
    [-e <customized_node_upgrade_variables>] 1
```

1

See Customizing Node Upgrades for any desired **<customized_node_upgrade_variables>**.

If you are upgrading the nodes in groups as described in Customizing Node Upgrades, continue invoking the ***upgrade_nodes.yml*** playbook until all nodes have been successfully upgraded.

6. Run the following command on each host to add the **atomic-openshift** packages back to the list of yum excludes on the host:

```
# atomic-openshift-excluder exclude
```

7. After all master and node upgrades have completed, a recommendation will be printed to reboot all hosts. After rebooting, if there are no additional features enabled, you can verify the upgrade. Otherwise, the next step depends on what additional features have you previously enabled.

Feature	Next Step
Aggregated Logging	Upgrade the EFK logging stack.
Cluster Metrics	Upgrade cluster metrics.

5.2.5. Upgrading the EFK Logging Stack

If you have previously deployed the EFK logging stack and want to upgrade to the latest logging component images, the steps must be performed manually as shown in Manual Upgrades.

5.2.6. Upgrading Cluster Metrics

If you have previously deployed cluster metrics, you must manually update to the latest metric components.

5.2.7. Verifying the Upgrade

To verify the upgrade:

1. First check that all nodes are marked as **Ready**:

```
# oc get nodes
NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

2. Then, verify that you are running the expected versions of the **docker-registry** and **router** images, if deployed. Replace **<tag>** with **v3.4.1.5** for the latest version.

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:<tag>",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:<tag>",
```

3. After upgrading, you can use the diagnostics tool on the master to look for common issues:

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

5.3. PERFORMING MANUAL IN-PLACE CLUSTER UPGRADES

5.3.1. Overview

Important

An etcd performance issue has been discovered on new and upgraded OpenShift Container Platform 3.4 clusters. See the following Knowledgebase Solution for further details:

https://access.redhat.com/solutions/2916381
(**BZ#1415839**)

As an alternative to performing an automated upgrade, 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 now with the entire procedure. Specific releases may require additional steps to be performed at key points before or during the standard upgrade process.

Important

Ensure that you have met all prerequisites before proceeding with an upgrade. Failure to do so can result in a failed upgrade.

5.3.2. Preparing for a Manual Upgrade

Note

Before upgrading your cluster to OpenShift Container Platform 3.4, the cluster must be already upgraded to the latest asynchronous release of version 3.3. Cluster upgrades cannot span more than one minor version at a time, so if your cluster is at a version earlier than 3.3, you must first upgrade incrementally (e.g., 3.1 to 3.2, then 3.2 to 3.3).

To prepare for a manual upgrade, follow these steps:

1. If you are upgrading from OpenShift Container Platform 3.3 to 3.4, manually disable the 3.3 channel and enable the 3.4 channel on each host:

```
# subscription-manager repos --disable="rhel-7-server-ose-3.3-
rpms" \
    --enable="rhel-7-server-ose-3.4-rpms" \
    --enable="rhel-7-server-extras-rpms"
```

   On RHEL 7 systems, also clear the **yum** cache:

```
# yum clean all
```

2. Install or update to the latest available version of the **atomic-openshift-utils** package on each RHEL 7 system, which provides files that will be used in later sections:

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

3. Install or update to the following latest available ***-excluder** packages on each RHEL 7 system, which helps ensure your systems stay on the correct versions of **atomic-openshift** and **docker** packages when you are not trying to upgrade, according to the OpenShift Container Platform version:

```
# yum install atomic-openshift-excluder atomic-openshift-docker-
excluder
```

   These packages add entries to the **`exclude`** directive in the host's ***/etc/yum.conf*** file.

4. Create an **etcd** backup on each master. The **etcd** package is required, even if using embedded etcd, for access to the **`etcdctl`** command to make the backup.

Note

The **etcd** package is installed by default for RHEL Atomic Host 7 systems. If the master is a RHEL 7 system and **etcd** is not already installed, install it now:

```
# yum install etcd
```

To create the backup, run:

```
# ETCD_DATA_DIR=/var/lib/origin/openshift.local.etcd 1
# etcdctl backup \
    --data-dir $ETCD_DATA_DIR \
    --backup-dir $ETCD_DATA_DIR.bak.<date> 2
```

1

This directory is for embedded etcd. For external etcd, use ***/var/lib/etcd*** instead.

2

Use the date of the backup, or some unique identifier, for **`<date>`**. The command will not make a backup if the **`--backup-dir`** location already exists.

5. For any upgrade path, ensure that you are running the latest kernel on each RHEL 7 system:

```
# yum update kernel
```

5.3.3. Upgrading Master Components

Before upgrading any stand-alone nodes, upgrade the master components (which provide the *control plane* for the cluster).

1. Run the following command on each master to remove the **atomic-openshift** packages from the list of yum excludes on the host:

```
# atomic-openshift-excluder unexclude
```

2. Upgrade **etcd** on all master hosts and any external etcd hosts.

    a. For RHEL 7 systems using the RPM-based method:

        i. Upgrade the **etcd** package:

```
# yum update etcd
```

ii. Restart the **etcd** service and review the logs to ensure it restarts successfully:

```
# systemctl restart etcd
# journalctl -r -u etcd
```

b. For RHEL Atomic Host 7 systems and RHEL 7 systems using the containerized method:

i. Pull the latest **rhel7/etcd** image:

```
# docker pull registry.access.redhat.com/rhel7/etcd
```

ii. Restart the **etcd_container** service and review the logs to ensure it restarts successfully:

```
# systemctl restart etcd_container
# journalctl -r -u etcd_container
```

3. On each master host, upgrade the **atomic-openshift** packages or related images.

a. For masters using the RPM-based method on a RHEL 7 system, upgrade all installed **atomic-openshift** packages:

```
# yum upgrade atomic-openshift\*
```

b. For masters using the containerized method on a RHEL 7 or RHEL Atomic Host 7 system, set the **`IMAGE_VERSION`** parameter to the version you are upgrading to in the following files:

- ***/etc/sysconfig/atomic-openshift-master*** (single master clusters only)
- ***/etc/sysconfig/atomic-openshift-master-controllers*** (multi-master clusters only)
- ***/etc/sysconfig/atomic-openshift-master-api*** (multi-master clusters only)
- ***/etc/sysconfig/atomic-openshift-node***
- ***/etc/sysconfig/atomic-openshift-openvswitch***

For example:

```
IMAGE_VERSION=<tag>
```

Replace **`<tag>`** with **`v3.4.1.5`** for the latest version.

4. Restart the master service(s) on each master and review logs to ensure they restart successfully.

For single master clusters:

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

For multi-master clusters:

```
# systemctl restart atomic-openshift-master-controllers
# systemctl restart atomic-openshift-master-api
# journalctl -r -u atomic-openshift-master-controllers
# journalctl -r -u atomic-openshift-master-api
```

5. Because masters also have node components running on them in order to be configured as part of the OpenShift SDN, restart the **atomic-openshift-node** and **openvswitch** services:

```
# systemctl restart atomic-openshift-node
# systemctl restart openvswitch
# journalctl -r -u openvswitch
# journalctl -r -u atomic-openshift-node
```

6. If you are performing a cluster upgrade that requires updating Docker to version 1.12, you must also perform the following steps if you are not already on Docker 1.12:

Important

The node component on masters is set by default to unschedulable status during initial installation, so that pods are not deployed to them. However, it is possible to set them schedulable during the initial installation or manually thereafter. If any of your masters are also configured as a schedulable node, skip the following Docker upgrade steps for those masters and instead run all steps described in Upgrading Nodes when you get to that section for those hosts as well.

   a. Upgrade the **docker** package.

      i. For RHEL 7 systems:

```
# yum update docker
```

      Then, restart the **docker** service and review the logs to ensure it restarts successfully:

```
# systemctl restart docker
# journalctl -r -u docker
```

      ii. For RHEL Atomic Host 7 systems, upgrade to the latest Atomic tree if one is available:

Note

If upgrading to RHEL Atomic Host 7.3.2, this upgrades Docker to version 1.12.

```
# atomic host upgrade
```

b. After the upgrade is completed and prepared for the next boot, reboot the host and ensure the **docker** service starts successfully:

```
# systemctl reboot
# journalctl -r -u docker
```

c. Remove the following file, which is no longer required:

```
# rm /etc/systemd/system/docker.service.d/docker-sdn-
ovs.conf
```

7. Run the following command on each master to add the **atomic-openshift** packages back to the list of yum excludes on the host:

```
# atomic-openshift-excluder exclude
```

Note

During the cluster upgrade, it can sometimes be useful to take a master out of rotation since some DNS client libraries will not properly to the other masters for cluster DNS. In addition to stopping the master and controller services, you can remove the EndPoint from the Kubernetes service's **`subsets.addresses`**.

```
$ oc edit ep/kubernetes -n default
```

When the master is restarted, the Kubernetes service will be automatically updated.

5.3.4. Updating Policy Definitions

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

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

Warning

If you have customized default cluster roles and want to ensure a role reconciliation does not modify those customized roles, annotate them with **`openshift.io/reconcile-protect`** set to **`true`**. Doing so means you are responsible for manually updating those roles with any new or required permissions during upgrades.

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 using the following process:

1. Reconcile the cluster roles:

```
# oadm policy reconcile-cluster-roles \
    --additive-only=true \
    --confirm
```

2. Reconcile the cluster role bindings:

```
# oadm policy reconcile-cluster-role-bindings \
    --exclude-groups=system:authenticated \
    --exclude-groups=system:authenticated:oauth \
    --exclude-groups=system:unauthenticated \
    --exclude-users=system:anonymous \
    --additive-only=true \
    --confirm
```

3. Reconcile security context constraints:

```
# oadm policy reconcile-sccs \
    --additive-only=true \
    --confirm
```

5.3.5. Upgrading Nodes

After upgrading your masters, you can upgrade your nodes. When restarting the **atomic-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.

Note

You can alternatively use the blue-green deployment method at this point to create a parallel environment for new nodes instead of upgrading them in place.

One at at time for each node that is not also a master, you must disable scheduling and evacuate its pods to other nodes, then upgrade packages and restart services.

1. Run the following command on each node to remove the **atomic-openshift** packages from the list of yum excludes on the host:

```
# atomic-openshift-excluder unexclude
```

2. As a user with **cluster-admin** privileges, disable scheduling for the node:

```
# oadm manage-node <node> --schedulable=false
```

3. Evacuate pods on the node to other nodes:

Important

The **`--force`** option deletes any pods that are not backed by a replication controller.

```
# oadm manage-node <node> --evacuate --force
```

4. Upgrade the node component packages or related images.

   a. For nodes using the RPM-based method on a RHEL 7 system, upgrade all installed **atomic-openshift** packages:

```
# yum upgrade atomic-openshift\*
```

   b. For nodes using the containerized method on a RHEL 7 or RHEL Atomic Host 7 system, set the **`IMAGE_VERSION`** parameter in the ***/etc/sysconfig/atomic-openshift-node*** and ***/etc/sysconfig/openvswitch*** files to the version you are upgrading to. For example:

```
IMAGE_VERSION=<tag>
```

   Replace **`<tag>`** with **`v3.4.1.5`** for the latest version.

5. Restart the **atomic-openshift-node** and **openvswitch** services and review the logs to ensure they restart successfully:

```
# systemctl restart atomic-openshift-node
# systemctl restart openvswitch
# journalctl -r -u atomic-openshift-node
# journalctl -r -u openvswitch
```

6. If you are performing a cluster upgrade that requires updating Docker to version 1.12, you must also perform the following steps if you are not already on Docker 1.12:

    a. Upgrade the **docker** package.

        i. For RHEL 7 systems:

        ```
        # yum update docker
        ```

        Then, restart the **docker** service and review the logs to ensure it restarts successfully:

        ```
        # systemctl restart docker
        # journalctl -r -u docker
        ```

        After Docker is restarted, restart the **atomic-openshift-node** service again and review the logs to ensure it restarts successfully:

        ```
        # systemctl restart atomic-openshift-node
        # journalctl -r -u atomic-openshift-node
        ```

        ii. For RHEL Atomic Host 7 systems, upgrade to the latest Atomic tree if one is available:

        Note

        If upgrading to RHEL Atomic Host 7.3.2, this upgrades Docker to version 1.10.

        ```
        # atomic host upgrade
        ```

        After the upgrade is completed and prepared for the next boot, reboot the host and ensure the **docker** service starts successfully:

        ```
        # systemctl reboot
        # journalctl -r -u docker
        ```

    b. Remove the following file, which is no longer required:

    ```
    # rm /etc/systemd/system/docker.service.d/docker-sdn-
    ovs.conf
    ```

7. Re-enable scheduling for the node:

    ```
    # oadm manage-node <node> --schedulable
    ```

8. Run the following command on each node to add the **atomic-openshift** packages back to the list of yum excludes on the host:

    ```
    # atomic-openshift-excluder exclude
    ```

9. Repeat the previous steps on the next node, and continue repeating these steps until all nodes have been upgraded.

10. After all nodes have been upgraded, as a user with **cluster-admin** privileges, verify that all nodes are showing as **Ready**:

```
# oc get nodes
NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

5.3.6. 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.

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:<tag> 1
        imagePullPolicy: IfNotPresent
        ...
```

1

Adjust **<tag>** to match the version you are upgrading to (use **v3.4.1.5** for the latest version).

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

5.3.7. 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. Storage for the Registry details how to configure persistent storage for the registry.

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:<tag> 1
        imagePullPolicy: IfNotPresent
        ...
```

1

Adjust **<tag>** to match the version you are upgrading to (use **v3.4.1.5** for the latest version).

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.

5.3.8. Updating the Default Image Streams and Templates

By default, the quick and advanced installation methods automatically create default image streams, InstantApp 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 the ***/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/*** directory.

Note

Because RHEL Atomic Host 7 cannot use **yum** to update packages, the following steps must take place on a RHEL 7 system.

1. Update the packages that provide the example JSON files. On a subscribed Red Hat Enterprise Linux 7 system where you can run the CLI as a user with **cluster-admin** permissions, install or update to the latest version of the **atomic-openshift-utils** package, which should also update the **openshift-ansible-** packages:

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

   The **openshift-ansible-roles** package provides the latest example JSON files.

2. Update the global **openshift** project by running the following commands. Receiving warnings about items that already exist is expected.

```
# oc create -n openshift -f /usr/share/openshift/examples/image-
streams/image-streams-rhel7.json
# oc create -n openshift -f /usr/share/openshift/examples/image-
streams/dotnet_imagestreams.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. After a manual upgrade, get the latest templates from **openshift-ansible-roles**:

```
rpm -ql openshift-ansible-roles | grep examples | grep v1.4
```

In this example, ***/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.4/image-streams/image-streams-rhel7.json*** is the latest file that you want in the latest **openshift-ansible-roles** package.

/usr/share/openshift/examples/image-streams/image-streams-rhel7.json is not owned by a package, but is updated by Ansible. If you are upgrading outside of Ansible. you need to get the latest .json files on the system where you are running **oc**, which can run anywhere that has access to the master.

4. Install **atomic-openshift-utils** and its dependencies to install the new content into ***/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.4/***.:

```
$ oc create -n openshift -f  /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/image-
streams/image-streams-rhel7.json
$ oc create -n openshift -f  /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/image-
streams/dotnet_imagestreams.json
$ oc replace -n openshift -f  /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/image-
streams/image-streams-rhel7.json
$ oc replace -n openshift -f  /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/image-
streams/dotnet_imagestreams.json
```

5. Update the templates:

```
$ oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/
$ oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/db-
templates/
$ oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/infrastructu
re-templates/
$ oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/xpaas-
templates/
$ oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/xpaas-
streams/
$ oc replace -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/
$ oc replace -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/db-
templates/
$ oc replace -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/infrastructu
re-templates/
$ oc replace -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/xpaas-
templates/
$ oc replace -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/xpaas-
streams/
```

Errors are generated for items that already exist. This is expected behavior:

```
# oc create -n openshift -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/
Error from server: error when creating
"/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/cakephp-mysql.json": templates "cakephp-mysql-example"
already exists
Error from server: error when creating
"/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/cakephp.json": templates "cakephp-example" already
exists
Error from server: error when creating
"/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/dancer-mysql.json": templates "dancer-mysql-example"
already exists
```

```
Error from server: error when creating
"/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/dancer.json": templates "dancer-example" already exists
Error from server: error when creating
"/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.4/quickstart-
templates/django-postgresql.json": templates "django-psql-
example" already exists
```

Now, content can be updated. Without running the automated upgrade playbooks, the content is not updated in ***/usr/share/openshift/***.

5.3.9. 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,v3.1.1.6    16 hours ago
mysql    registry.access.redhat.com/openshift3/mysql-55-rhel7
5.5,latest,v3.1.1.6    16 hours ago
nodejs   registry.access.redhat.com/openshift3/nodejs-010-rhel7
0.10,latest,v3.1.1.6   16 hours ago
...
```

Update each image stream one at a time:

```
# oc import-image -n openshift nodejs
The import completed successfully.

Name:   nodejs
Created:  10 seconds ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2016-07-
05T19:20:30Z
Docker Pull Spec: 172.30.204.22:5000/openshift/nodejs

Tag Spec        Created  PullSpec      Image
latest 4        9 seconds ago registry.access.redhat.com/rhscl/nodejs-
4-rhel7:latest
570ad8ed927fd5c2c9554ef4d9534cef808dfa05df31ec491c0969c3bd372b05
4 registry.access.redhat.com/rhscl/nodejs-4-rhel7:latest  9 seconds ago
<same>
```

```
570ad8ed927fd5c2c9554ef4d9534cef808dfa05df31ec491c0969c3bd372b05
0.10 registry.access.redhat.com/openshift3/nodejs-010-rhel7:latest 9
seconds ago <same>
a1ef33be788a28ec2bdd48a9a5d174ebcfbe11c8e986d2996b77f5bccaaa4774
```

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>`**.

5.3.10. Upgrading the EFK Logging Stack

Use the following to upgrade an already-deployed EFK logging stack.

Note

The following steps apply when upgrading to OpenShift Container Platform 3.4+.

1. Ensure you are working in the project where the EFK stack was previously deployed. For example, if the project is named **logging**:

```
$ oc project logging
```

2. Recreate the deployer templates for service accounts and running the deployer:

```
$ oc apply -n openshift -f \
    /usr/share/ansible/openshift-
ansible/roles/openshift_hosted_templates/files/v1.4/enterprise/lo
gging-deployer.yaml
```

3. Generate any missing service accounts and roles:

```
$ oc process logging-deployer-account-template | oc apply -f -
```

4. Ensure that the cluster role **`oauth-editor`** is assigned to the **logging-deployer** service account:

```
$ oadm policy add-cluster-role-to-user oauth-editor \
       system:serviceaccount:logging:logging-deployer
```

5. Ensure that the cluster role **`rolebinding-reader`** is assigned to the **aggregated-logging-elasticsearch** service account where **`logging`** is the namespace with aggregated logging installed:

```
$ oadm policy add-cluster-role-to-user rolebinding-reader \
     system:serviceaccount:logging:aggregated-logging-
elasticsearch
```

6. If you are upgrading from OpenShift Container Platform 3.3 to 3.4, add the **privileged** SCC to the **aggregated-logging-fluentd** service account:

```
$ oadm policy add-scc-to-user privileged \
    system:serviceaccount:logging:aggregated-logging-fluentd
```

7. In preparation for running the deployer, ensure that you have the configurations for your current deployment in the **logging-deployer** ConfigMap.

Important

Ensure that your image version is the latest version, not the currently installed version.

8. Run the deployer with the parameter in **`upgrade`** mode:

```
$ oc new-app logging-deployer-template -p MODE=upgrade
```

Running the deployer in this mode handles scaling down the components to minimize loss of logs, patching configurations, generating missing secrets and keys, and scaling the components back up to their previous replica count.

Important

Due to the privileges needed to label and unlabel a node for controlling the deployment of Fluentd pods, the deployer does delete the **logging-fluentd** Daemonset and recreates it from the **logging-fluentd-template** template.

5.3.11. Upgrading Cluster Metrics

After upgrading an already-deployed Cluster Metrics install, you must update to a newer version of the metrics components.

- The update process stops all the metrics containers, updates the metrics configuration files, and redeploys the newer components.
- It does not change the metrics route.
- It does not delete the metrics persistent volume claim. Metrics stored to persistent volumes before the update are available after the update completes.

Important

The update deletes all non-persisted metric values and overwrites local changes to the metrics configurations. For example, the number of instances in a replica set is not saved.

To upgrade cluster metrics:

1. If you are upgrading from OpenShift Container Platform 3.3 to 3.4, first add the **`view`** permission to the **hawkular** service account:

```
$ oadm policy add-role-to-user view \
        system:serviceaccount:openshift-infra:hawkular \
        -n openshift-infra
```

2. Then, follow the same steps as when the metrics components were first deployed, using the correct template, except this time, specify the **MODE=refresh** option:

```
$ oc new-app --as=system:serviceaccount:openshift-infra:metrics-
deployer \
    -f metrics-deployer.yaml \
    -p HAWKULAR_METRICS_HOSTNAME=hm.example.com \
    -p MODE=refresh 1
```

1

In the original deployment command, there was no **MODE=refresh**.

Note

During the update, the metrics components do not run. Because of this, they cannot collect data and a gap normally appears in the graphs.

5.3.12. Additional Manual Steps Per Release

Some OpenShift Container Platform releases may have additional instructions specific to that release that must be performed to fully apply the updates across the cluster. This section will be updated over time as new asynchronous updates are released for OpenShift Container Platform 3.4.

See the OpenShift Container Platform 3.4 Release Notes to review the latest release notes.

5.3.12.1. OpenShift Container Platform 3.4.0.40

If you had previously already upgraded to 3.4.0.39 (the GA release of OpenShift Container Platform 3.4), after upgrading to the 3.4.0.40 release you must also then perform a data migration using a data migration tool. See the following Knowledgebase Solution for further details on this tool:

https://access.redhat.com/solutions/2887651

5.3.12.2. OpenShift Container Platform 3.4.1.2

There are no additional manual steps for the upgrade to OpenShift Container Platform 3.4.1.2 that are not already mentioned inline during the standard manual upgrade process.

5.3.12.3. OpenShift Container Platform 3.4.1.5

OpenShift Container Platform 3.4.1.5 delivers the migration tool mentioned in the above OpenShift Container Platform 3.4.0.40 steps. See the following Knowledgebase Solution for instructions on running the script:

https://access.redhat.com/solutions/2887651

5.3.13. Verifying the Upgrade

To verify the upgrade, first check that all nodes are marked as **Ready**:

```
# oc get nodes
NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

Then, verify that you are running the expected versions of the **docker-registry** and **router** images, if deployed. Replace **`<tag>`** with **`v3.4.1.5`** for the latest version.

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:<tag>",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:<tag>",
```

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

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

5.4. BLUE-GREEN DEPLOYMENTS

5.4.1. Overview

Important

An etcd performance issue has been discovered on new and upgraded OpenShift Container Platform 3.4 clusters. See the following Knowledgebase Solution for further details:

https://access.redhat.com/solutions/2916381
(**BZ#1415839**)

Note

This topic serves as an alternative approach for node upgrades to the in-place upgrade method.

The blue-green deployment upgrade method follows a similar flow to the in-place method: masters and etcd servers are still upgraded first, however a parallel environment is created for new nodes instead of upgrading them in-place.

This method allows administrators to switch traffic from the old set of nodes (e.g., the "blue" deployment) to the new set (e.g., the "green" deployment) after the new deployment has been verified. If a problem is detected, it is also then easy to rollback to the old deployment quickly.

While blue-green is a proven and valid strategy for deploying just about any software, there are always trade-offs. Not all environments have the same uptime requirements or the resources to properly perform blue-green deployments. In an OpenShift Container Platform environment, the most suitable candidate for blue-green deployments are the nodes. All user processes run on these systems and even critical pieces of OpenShift Container Platform infrastructure are self-hosted there. Uptime is most important for these workloads and the additional complexity of blue-green deployments can be justified.

The exact implementation of this approach varies based on your requirements. Often the main challenge is having the excess capacity to facilitate such an approach.

Another lesser challenge is that the administrator must temporarily share the Red Hat software entitlements between the blue-green deployments or provide access to the installation content by means of a system such as Red Hat Satellite. This can be accomplished by sharing the consumer ID from the previous host.

5.4.2. Preparing for Upgrade

1. On the old host:

```
# subscription-manager identity | grep system
system identity: 6699375b-06db-48c4-941e-689efd6ce3aa
```

2. On the new host:

```
# subscription-manager register --consumerid=6699375b-06db-48c4-
941e-689efd6ce3aa
```

Important

After a successful deployment, remember to unregister the old host with **`subscription-manager clean`** to prevent the environment from being out of compliance.

3. After the master and etcd servers have been upgraded, you must ensure that your current production nodes are labeled either blue or green. In this example, the current installation will be blue and the new environment will be green. On each production node in your current installation:

```
$ oc label --all nodes color=blue
```

In the case of nodes requiring the uptime guarantees of a blue-green deployment, the **`-l`** flag can be used to match a subset of the environment using a selector.

4. Create the new green environment for any nodes that are to be replaced by adding an equal number of new nodes to the existing cluster. Ansible can apply the **`color=green`** label using the **`openshift_node_labels`** variable for each node.

5. In order to delay workload scheduling until the nodes are healthy, be sure to set the **`openshift_schedulable=false`** variable. After the green nodes are in **Ready** state, they can be made schedulable.

6. Blue nodes are disabled so that no new pods are run on them:

```
# oadm manage-node --schedulable=true --selector=color=green
# oadm manage-node --schedulable=false --selector=color=blue
```

A common practice is to scale the registry and router pods until they are migrated to the green nodes. For these pods, a *canary* deployment approach is commonly used. Scaling them up will make them immediately active on the new nodes. Pointing the deployment configuration to the new image initiates a rolling update. However, because of node anti-affinity, and the fact that the blue nodes are still unschedulable, the deployments to the old nodes will fail. At this point, the registry and router deployments can be scaled down to the original number of pods. At any given point, the original number of pods is still available so no capacity is lost.

5.4.3. Warming the New Nodes

In order for pods to be migrated from the blue environment to the green, the images must be pulled. Network latency and load on the registry can cause delays if there is not sufficient capacity built in to the environment. Often, the best way to minimize impact to the running system is to trigger new pod deployments that will land on the new nodes. Accomplish this by importing new image streams.

A major release of OpenShift Container Platform is the motivation for a blue-green deployment. At that time, new image streams become available for users of Source-to-Image (S2I). Upon import, any builds or deployments configured with **ImageChangeTriggers** are automatically created.

Note

To continue with the upgrade process, update the default image streams and templates and import the latest images.

It is important to realize that this process can trigger a large number of builds. The good news is that the builds are performed on the green nodes and, therefore, do not impact any traffic on the blue deployment.

To monitor build progress across all namespaces (projects) in the cluster:

```
$ oc get events -w --all-namespaces
```

In large environments, builds rarely completely stop. However, you should see a large increase and decrease caused by the administrative import.

Another benefit of triggering the builds is that it does a fairly good job of fetching the majority of the ancillary images to all nodes such as the various build images, the pod infrastructure image, and deployers. Everything else can be moved over using node evacuation and will proceed more quickly as a result.

5.4.4. Node Evacuation

For larger deployments, it is possible to have other labels that help determine how evacuation can be coordinated. The most conservative approach for avoiding downtime is to evacuate one node at

a time. If services are composed of pods using zone anti-affinity, then an entire zone can be evacuated at once. It is important to ensure that the storage volumes used are available in the new zone as this detail can vary among cloud providers.

In OpenShift Enterprise 3.2 and later, a node evacuation is triggered whenever the service is stopped. Achieve manual evacuation and deletion of all blue nodes at once by:

```
# oadm manage-node --selector=color=blue --evacuate
# oc delete node --selector=color=blue
```

5.5. OPERATING SYSTEM UPDATES AND UPGRADES

5.5.1. Impacts

When updating or upgrading your operating system (OS), either changing OS versions or updating the system software, be aware that this can impact the OpenShift Container Platform software running on those machines. In particular, these updates can affect the **`iptables`** rules or **`ovs`** flows that OpenShift Container Platform requires to operate.

5.5.2. Solutions

Reboot the node to ensure that all of the software is running the new versions, especially if a new kernel is installed. A reboot will also ensure that the **`docker`** and OpenShift Container Platform processes have been restarted, which will force them to check that all of the rules in other services are correct.

However, if you do not wish to do a reboot of the node, then it is possible to just restart the services that are affected, or to preserve the **`iptables`** state. Both processes are described in the OpenShift Container Platform IPtables topic. The **`ovs`** flow rules do not need to be saved, but restarting the OpenShift Container Platform node software will fix the flow rules.

CHAPTER 6. DOWNGRADING OPENSHIFT

6.1. OVERVIEW

Following an OpenShift Container Platform upgrade, it may be desirable in extreme cases to downgrade your cluster to a previous version. The following sections outline the required steps for each system in a cluster to perform such a downgrade for the OpenShift Container Platform 3.2 to 3.1 downgrade path.

Warning

These steps are currently only supported for RPM-based installations of OpenShift Container Platform and assumes downtime of the entire cluster.

Important

For the OpenShift Container Platform 3.1 to 3.0 downgrade path, see the OpenShift Enterprise 3.1 documentation, which has modified steps.

6.2. VERIFYING BACKUPS

The Ansible playbook used during the upgrade process should have created a backup of the ***master-config.yaml*** file and the etcd data directory. Ensure these exist on your masters and etcd members:

```
/etc/origin/master/master-config.yaml.<timestamp>
/var/lib/origin/etcd-backup-<timestamp>
```

Also, back up the ***node-config.yaml*** file on each node (including masters, which have the node component on them) with a timestamp:

```
/etc/origin/node/node-config.yaml.<timestamp>
```

If you are using an external etcd cluster (versus the single embedded etcd), the backup is likely created on all etcd members, though only one is required for the recovery process.

The RPM downgrade process in a later step should create ***.rpmsave*** backups of the following files, but it may be a good idea to keep a separate copy regardless:

```
/etc/sysconfig/atomic-openshift-master
/etc/etcd/etcd.conf 1
```

1

Only required if using external etcd.

6.3. SHUTTING DOWN THE CLUSTER

On all masters, nodes, and etcd members (if using an external etcd cluster), ensure the relevant services are stopped.

On the master in a single master cluster:

```
# systemctl stop atomic-openshift-master
```

On each master in a multi-master cluster:

```
# systemctl stop atomic-openshift-master-api
# systemctl stop atomic-openshift-master-controllers
```

On all master and node hosts:

```
# systemctl stop atomic-openshift-node
```

On any external etcd hosts:

```
# systemctl stop etcd
```

6.4. REMOVING RPMS

On all masters, nodes, and etcd members (if using an external etcd cluster), remove the following packages:

```
# yum remove atomic-openshift \
    atomic-openshift-clients \
    atomic-openshift-node \
    atomic-openshift-master \
    openvswitch \
    atomic-openshift-sdn-ovs \
    tuned-profiles-atomic-openshift-node
```

If you are using external etcd, also remove the **etcd** package:

```
# yum remove etcd
```

For embedded etcd, you can leave the **etcd** package installed, as the package is only required so that the **`etcdctl`** command is available to issue operations in later steps.

6.5. DOWNGRADING DOCKER

OpenShift Container Platform 3.2 requires Docker 1.9.1 and also supports Docker 1.10.3, however OpenShift Container Platform 3.1 requires Docker 1.8.2.

Downgrade to Docker 1.8.2 on each host using the following steps:

1. Remove all local containers and images on the host. Any pods backed by a replication controller will be recreated.

   Warning

   The following commands are destructive and should be used with caution.

   Delete all containers:

   ```
   # docker rm $(docker ps -a -q)
   ```

   Delete all images:

   ```
   # docker rmi $(docker images -q)
   ```

2. Use **`yum swap`** (instead of **`yum downgrade`**) to install Docker 1.8.2:

   ```
   # yum swap docker-* docker-*1.8.2
   # sed -i 's/--storage-opt dm.use_deferred_deletion=true//'
   /etc/sysconfig/docker-storage
   # systemctl restart docker
   ```

3. You should now have Docker 1.8.2 installed and running on the host. Verify with the following:

   ```
   # docker version
   Client:
    Version:      1.8.2-el7
    API version:  1.20
    Package Version: docker-1.8.2-10.el7.x86_64
   [...]

   # systemctl status docker
   ● docker.service - Docker Application Container Engine
      Loaded: loaded (/usr/lib/systemd/system/docker.service;
   enabled; vendor preset: disabled)
      Active: active (running) since Mon 2016-06-27 15:44:20 EDT;
   33min ago
   [...]
   ```

6.6. REINSTALLING RPMS

Disable the OpenShift Container Platform 3.3 repositories, and re-enable the 3.2 repositories:

```
# subscription-manager repos \
```

```
    --disable=rhel-7-server-ose-3.3-rpms \
    --enable=rhel-7-server-ose-3.2-rpms
```

On each master, install the following packages:

```
# yum install atomic-openshift \
    atomic-openshift-clients \
    atomic-openshift-node \
    atomic-openshift-master \
    openvswitch \
    atomic-openshift-sdn-ovs \
    tuned-profiles-atomic-openshift-node
```

On each node, install the following packages:

```
# yum install atomic-openshift \
    atomic-openshift-node \
    openvswitch \
    atomic-openshift-sdn-ovs \
    tuned-profiles-atomic-openshift-node
```

If using an external etcd cluster, install the following package on each etcd member:

```
# yum install etcd
```

6.7. RESTORING ETCD

Whether using embedded or external etcd, you must first restore the etcd backup by creating a new, single node etcd cluster. If using external etcd with multiple members, you must then also add any additional etcd members to the cluster one by one.

However, the details of the restoration process differ between embedded and external etcd. See the following section that matches your etcd configuration and follow the relevant steps before continuing to Bringing OpenShift Container Platform Services Back Online.

Follow the Cluster Restore procedure to restore single-member etcd clusters.

6.7.1. Embedded etcd

Restore your etcd backup and configuration:

1. Run the following on the master with the embedded etcd:

```
# ETCD_DIR=/var/lib/origin/openshift.local.etcd
# mv $ETCD_DIR /var/lib/etcd.orig
# cp -Rp /var/lib/origin/etcd-backup-<timestamp>/ $ETCD_DIR
# chcon -R --reference /var/lib/etcd.orig/ $ETCD_DIR
# chown -R etcd:etcd $ETCD_DIR
```

Warning

The **$ETCD_DIR** location differs between external and embedded etcd.

2. Create the new, single node etcd cluster:

```
# etcd -data-dir=/var/lib/origin/openshift.local.etcd \
    -force-new-cluster
```

Verify etcd has started successfully by checking the output from the above command, which should look similar to the following near the end:

```
[...]
2016-06-24 12:14:45.644073 I | etcdserver: starting server...
[version: 2.2.5, cluster version: 2.2]
[...]
2016-06-24 12:14:46.834394 I | etcdserver: published
{Name:default ClientURLs:[http://localhost:2379
http://localhost:4001]} to cluster 5580663a6e0002
```

3. Shut down the process by running the following from a separate terminal:

```
# pkill etcd
```

4. Continue to Bringing OpenShift Container Platform Services Back Online.

6.7.2. External etcd

Choose a system to be the initial etcd member, and restore its etcd backup and configuration:

1. Run the following on the etcd host:

```
# ETCD_DIR=/var/lib/etcd/
# mv $ETCD_DIR /var/lib/etcd.orig
# cp -Rp /var/lib/origin/etcd-backup-<timestamp>/ $ETCD_DIR
# chcon -R --reference /var/lib/etcd.orig/ $ETCD_DIR
# chown -R etcd:etcd $ETCD_DIR
```

Warning

The **$ETCD_DIR** location differs between external and embedded etcd.

2. Restore your ***/etc/etcd/etcd.conf*** file from backup or ***.rpmsave***.

3. Create the new single node cluster using etcd's **`--force-new-cluster`** option. You can do this with a long complex command using the values from ***/etc/etcd/etcd.conf***, or you can temporarily modify the **systemd** unit file and start the service normally.

   To do so, edit the ***/usr/lib/systemd/system/etcd.service*** and add **`--force-new-cluster`**:

   ```
   # sed -i '/ExecStart/s/"$/  --force-new-cluster"/'
   /usr/lib/systemd/system/etcd.service
   # cat /usr/lib/systemd/system/etcd.service  | grep ExecStart

   ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --
   force-new-cluster"
   ```

   Then restart the **etcd** service:

   ```
   # systemctl daemon-reload
   # systemctl start etcd
   ```

4. Verify the **etcd** service started correctly, then re-edit the ***/usr/lib/systemd/system/etcd.service*** file and remove the **`--force-new-cluster`** option:

   ```
   # sed -i '/ExecStart/s/ --force-new-cluster//'
   /usr/lib/systemd/system/etcd.service
   # cat /usr/lib/systemd/system/etcd.service  | grep ExecStart

   ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd"
   ```

5. Restart the **etcd** service, then verify the etcd cluster is running correctly and displays OpenShift Container Platform's configuration:

   ```
   # systemctl daemon-reload
   # systemctl restart etcd
   # etcdctl --cert-file=/etc/etcd/peer.crt \
       --key-file=/etc/etcd/peer.key \
       --ca-file=/etc/etcd/ca.crt \
       --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
       ls /
   ```

6. If you have additional etcd members to add to your cluster, continue to Adding Additional etcd Members. Otherwise, if you only want a single node external etcd, continue to Bringing OpenShift Container Platform Services Back Online.

6.7.2.1. Adding Additional etcd Members

To add additional etcd members to the cluster, you must first adjust the default **localhost** peer in the **`peerURLs`** value for the first member:

1. Get the member ID for the first member using the **`member list`** command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https:/
/172.18.0.75:2379" \
    member list
```

2. Update the value of **peerURLs** using the **etcdctl member update** command by passing the member ID obtained from the previous step:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https:/
/172.18.0.75:2379" \
    member update 511b7fb6cc0001 https://172.18.1.18:2380
```

Alternatively, you can use **curl**:

```
# curl --cacert /etc/etcd/ca.crt \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
    https://172.18.1.18:2379/v2/members/511b7fb6cc0001 \
    -XPUT -H "Content-Type: application/json" \
    -d '{"peerURLs":["https://172.18.1.18:2380"]}'
```

3. Re-run the **member list** command and ensure the peer URLs no longer include **localhost**.

4. Now add each additional member to the cluster, one at a time.

> **Warning**
>
> Each member must be fully added and brought online one at a time. When adding each additional member to the cluster, the **peerURLs** list must be correct for that point in time, so it will grow by one for each member added. The **etcdctl member add** command will output the values that need to be set in the ***etcd.conf*** file as you add each member, as described in the following instructions.

   a. For each member, add it to the cluster using the values that can be found in that system's ***etcd.conf*** file:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --
```

```
peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    member add 10.3.9.222 https://172.16.4.27:2380

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to
cluster

ETCD_NAME="10.3.9.222"
ETCD_INITIAL_CLUSTER="10.3.9.221=https://172.16.4.18:2380,1
0.3.9.222=https://172.16.4.27:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

b. Using the environment variables provided in the output of the above **`etcdctl member add`** command, edit the ***/etc/etcd/etcd.conf*** file on the member system itself and ensure these settings match.

c. Now start etcd on the new member:

```
# rm -rf /var/lib/etcd/member
# systemctl enable etcd
# systemctl start etcd
```

d. Ensure the service starts correctly and the etcd cluster is now healthy:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --
peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    member list

51251b34b80001: name=10.3.9.221
peerURLs=https://172.16.4.18:2380
clientURLs=https://172.16.4.18:2379
d266df286a41a8a4: name=10.3.9.222
peerURLs=https://172.16.4.27:2380
clientURLs=https://172.16.4.27:2379

# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --
peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
    cluster-health

cluster is healthy
member 51251b34b80001 is healthy
member d266df286a41a8a4 is healthy
```

e. Now repeat this process for the next member to add to the cluster.

5. After all additional etcd members have been added, continue to Bringing OpenShift Container Platform Services Back Online.

6.8. BRINGING OPENSHIFT CONTAINER PLATFORM SERVICES BACK ONLINE

On each OpenShift Container Platform master, restore your master and node configuration from backup and enable and restart all relevant services.

On the master in a single master cluster:

```
# cp /etc/sysconfig/atomic-openshift-master.rpmsave
/etc/sysconfig/atomic-openshift-master
# cp /etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp /etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# systemctl enable atomic-openshift-master
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master
# systemctl start atomic-openshift-node
```

On each master in a multi-master cluster:

```
# cp /etc/sysconfig/atomic-openshift-master-api.rpmsave
/etc/sysconfig/atomic-openshift-master-api
# cp /etc/sysconfig/atomic-openshift-master-controllers.rpmsave
/etc/sysconfig/atomic-openshift-master-controllers
# cp /etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp /etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# systemctl enable atomic-openshift-master-api
# systemctl enable atomic-openshift-master-controllers
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master-api
# systemctl start atomic-openshift-master-controllers
# systemctl start atomic-openshift-node
```

On each OpenShift Container Platform node, restore your ***node-config.yaml*** file from backup and enable and restart the **atomic-openshift-node** service:

```
# cp /etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-node
```

Your OpenShift Container Platform cluster should now be back online.

6.9. VERIFYING THE DOWNGRADE

To verify the downgrade, first check that all nodes are marked as **Ready**:

```
# oc get nodes
```

```
NAME                      STATUS                     AGE
master.example.com        Ready,SchedulingDisabled   165d
node1.example.com         Ready                      165d
node2.example.com         Ready                      165d
```

Then, verify that you are running the expected versions of the **docker-registry** and **router** images, if deployed:

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:v3.1.1.6",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.1.1.6",
```

You can use the diagnostics tool on the master to look for common issues and provide suggestions. In OpenShift Container Platform 3.1, the **`oc adm diagnostics`** tool is available as **`openshift ex diagnostics`**:

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

CHAPTER 7. MASTER AND NODE CONFIGURATION

7.1. OVERVIEW

The **`openshift start`** command is used to launch OpenShift Container Platform servers. The command and its subcommands (**`master`** to launch a master server and **`node`** to launch a node server) all take a limited set of arguments that are sufficient for launching servers in a development or experimental environment.

However, these arguments are insufficient to describe and control the full set of configuration and security options that are necessary in a production environment. To provide those options, it is necessary to use the dedicated master and node configuration files.

Master configuration files and node configuration files are fully specified with no default values. Therefore, any empty value indicates that you want to start up with an empty value for that parameter. This makes it easy to reason about exactly what your configuration is, but it also makes it difficult to remember all of the options to specify. To make this easier, the configuration files can be created with the **`--write-config`** option and then used with the **`--config`** option.

7.2. MASTER CONFIGURATION FILES

This section reviews parameters mentioned in the ***master-config.yaml*** file.

You can create a new master configuration file to see the valid options for your installed version of OpenShift Container Platform.

7.2.1. Admission Control Configuration

Table 7.1. Admission Control Configuration Parameters

Parameter Name	Description
`AdmissionConfig`	Contains admission control plug-in configuration.
`APIServerArguments`	Key-value pairs that will be passed directly to the Kube API server that match the API servers' command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in **`KubernetesMasterConfig`**, which may cause invalid configurations.
`ControllerArguments`	Key-value pairs that will be passed directly to the Kube controller manager that match the controller manager's command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in **`KubernetesMasterConfig`**, which may cause invalid configurations.

Parameter Name	Description
DefaultAdmissionConfig	Used to enable or disable various admission plug-ins. When this type is present as the **configuration** object under **pluginConfig** and if the admission plug-in supports it, this will cause an **off by default** admission plug-in to be enabled.
PluginConfig	Allows specifying a configuration file per admission control plug-in.
PluginOrderOverride	A list of admission control plug-in names that will be installed on the master. Order is significant. If empty, a default list of plug-ins is used.
SchedulerArguments	Key-value pairs that will be passed directly to the Kube scheduler that match the scheduler's command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in **KubernetesMasterConfig**, which may cause invalid configurations.

7.2.2. Asset Configuration

Table 7.2. Asset Configuration Parameters

Parameter Name	Description
AssetConfig	Holds the necessary configuration options for serving assets.
DisabledFeatures	A list of features that should not be started. You will likely want to set this as **null**. It is very unlikely that anyone will want to manually disable features and that is not encouraged.
Extensions	Files to serve from the asset server file system under a subcontext.

Parameter Name	Description
`ExtensionDevelopment`	When set to **true**, tells the asset server to reload extension scripts and stylesheets for every request rather than only at startup. It lets you develop extensions without having to restart the server for every change.
`ExtensionProperties`	Key- (string) and value- (string) pairs that will be injected into the console under the global variable **`OPENSHIFT_EXTENSION_PROPERTIES`**.
`ExtensionScripts`	File paths on the asset server files to load as scripts when the web console loads.
`ExtensionStylesheets`	File paths on the asset server files to load as style sheets when the web console loads.
`LoggingPublicURL`	The public endpoint for logging (optional).
`LogoutURL`	An optional, absolute URL to redirect web browsers to after logging out of the web console. If not specified, the built-in logout page is shown.
`MasterPublicURL`	How the web console can access the OpenShift Container Platform server.
`MetricsPublicURL`	The public endpoint for metrics (optional).
`PublicURL`	URL of the the asset server.

7.2.3. Authentication and Authorization Configuration

Table 7.3. Authentication and Authorization Parameters

Parameter Name	Description
`authConfig`	Holds authentication and authorization configuration options.
`AuthenticationCacheSize`	Indicates how many authentication results should be cached. If 0, the default cache size is used.
`AuthorizationCacheTTL`	Indicates how long an authorization result should be cached. It takes a valid time duration string (e.g. "5m"). If empty, you get the default timeout. If zero (e.g. "0m"), caching is disabled.

7.2.4. Controller Configuration

Table 7.4. Controller Configuration Parameters

Parameter Name	Description
`Controllers`	List of the controllers that should be started. If set to **none**, no controllers will start automatically. The default value is * which will start all controllers. When using *, you may exclude controllers by prepending a `-` in front of their name. No other values are recognized at this time.
`ControllerLeaseTTL`	Enables controller election, instructing the master to attempt to acquire a lease before controllers start and renewing it within a number of seconds defined by this value. Setting this value non-negative forces `pauseControllers=true`. This value defaults off (0, or omitted) and controller election can be disabled with -1.
`PauseControllers`	Instructs the master to not automatically start controllers, but instead to wait until a notification to the server is received before launching them.

7.2.5. etcd Configuration

Table 7.5. etcd Configuration Parameters

Parameter Name	Description
`Address`	The advertised host:port for client connections to etcd.
`etcdClientInfo`	Contains information about how to connect to etcd.
`etcdConfig`	Holds the necessary configuration options for connecting with an etcd database.
`etcdStorageConfig`	Contains information about how API resources are stored in etcd. These values are only relevant when etcd is the backing store for the cluster.
`KubernetesStoragePrefix`	The path within etcd that the Kubernetes resources will be rooted under. This value, if changed, will mean existing objects in ***etcd*** will no longer be located. The default value is **kubernetes.io**.
`KubernetesStorageVersion`	The API version that Kubernetes resources in ***etcd*** should be serialized to. This value should **not** be advanced until all clients in the cluster that read from etcd have code that allows them to read the new version.
`OpenShiftStoragePrefix`	The path within etcd that the OpenShift Container Platform resources will be rooted under. This value, if changed, will mean existing objects in etcd will no longer be located. The default value is **openshift.io**.
`OpenShiftStorageVersion`	API version that OS resources in ***etcd*** should be serialized to. This value should **not** be advanced until all clients in the cluster that read from ***etcd*** have code that allows them to read the new version.
`PeerAddress`	The advertised host:port for peer connections to ***etcd***.
`PeerServingInfo`	Describes how to start serving the ***etcd*** peer.
`ServingInfo`	Describes how to start serving the etcd master.
`StorageDir`	The path to the ***etcd*** storage directory.

7.2.6. Grant Configuration

Table 7.6. Grant Configuration Parameters

Parameter Name	Description
`GrantConfig`	Describes how to handle grants.
`GrantHandlerAuto`	Auto-approves client authorization grant requests.
`GrantHandlerDeny`	Auto-denies client authorization grant requests.
`GrantHandlerPrompt`	Prompts the user to approve new client authorization grant requests.
`Method`	Determines the default strategy to use when an OAuth client requests a grant.This method will be used only if the specific OAuth client does not provide a strategy of their own. Valid grant handling methods are: - auto: always approves grant requests, useful for trusted clients - prompt: prompts the end user for approval of grant requests, useful for third-party clients - deny: always denies grant requests, useful for black-listed clients

7.2.7. Image Configuration

Table 7.7. Image Configuration Parameters

Parameter Name	Description
`DisableScheduledImport`	Allows scheduled background import of images to be disabled.
`Format`	The format of the name to be built for the system component.
`ImageConfig`	Holds options that describe how to build image names for system components.

Parameter Name	Description
ImagePolicyConfig	Controls limits and behavior for importing images.
Latest	Determines if the latest tag will be pulled from the registry.
MaxImagesBulkImportedPerRepository	Controls the number of images that are imported when a user does a bulk import of a Docker repository. This number defaults to 5 to prevent users from importing large numbers of images accidentally. Set **-1** for no limit.
MaxScheduledImageImportsPerMinute	The maximum number of scheduled image streams that will be imported in the background per minute. The default value is 60. Set to -1 for unlimited.
ScheduledImageImportMinimumIntervalSeconds	The minimum number of seconds that can elapse between when image streams scheduled for background import are checked against the upstream repository. The default value is 15 minutes.

7.2.8. Kubernetes Master Configuration

Table 7.8. Kubernetes Master Configuration Parameters

Parameter Name	Description
APILevels	A list of API levels that should be enabled on startup, v1 as examples.
DisabledAPIGroupVersions	A map of groups to the versions (or *****) that should be disabled.
KubeletClientInfo	Contains information about how to connect to kubelets.
KubernetesMasterConfig	Holds the necessary configuration options for the Kubernetes master.

Parameter Name	Description
`MasterCount`	The number of expected masters that should be running. This value defaults to 1 and may be set to a positive integer, or if set to -1, indicates this is part of a cluster.
`MasterIP`	The public IP address of Kubernetes resources. If empty, the first result from **`net.InterfaceAddrs`** will be used.
`MasterKubeConfig`	File name for the ***.kubeconfig*** file that describes how to connect this node to the master.
`ServicesNodePortRange`	The range to use for assigning service public ports on a host.
`ServicesSubnet`	The subnet to use for assigning service IPs.
`StaticNodeNames`	The list of nodes that are statically known.

7.2.9. Network Configuration

Table 7.9. Network Configuration Parameters

Parameter Name	Description
`ClusterNetworkCIDR`	The CIDR string to specify the global overlay network's L3 space.
`ExternalIPNetworkCIDRs`	Controls what values are acceptable for the service external IP field. If empty, no **`externalIP`** may be set. It may contain a list of CIDRs which are checked for access. If a CIDR is prefixed with **!**, IPs in that CIDR will be rejected. Rejections will be applied first, then the IP checked against one of the allowed CIDRs. You hould ensure this range does not overlap with your nodes, pods, or service CIDRs for security reasons.
`HostSubnetLength`	The number of bits to allocate to each host's subnet. For example, 8 would mean a /24 network on the host.

Parameter Name	Description
`IngressIPNetworkCIDR`	Controls the range to assign ingress IPs from for services of type **LoadBalancer** on bare metal. If empty, ingress IPs will not be assigned. It may contain a single CIDR that will be allocated from. For security reasons, you should ensure that this range does not overlap with the CIDRs reserved for external IPs, nodes, pods, or services.
`NetworkConfig`	Provides network options for the node.
`NetworkPluginName`	The name of the network plug-in to use.
`ServiceNetwork`	The CIDR string to specify the service networks.

7.2.10. OAuth Authentication Configuration

Table 7.10. OAuth Configuration Parameters

Parameter Name	Description
`AlwaysShowProviderSelection`	Forces the provider selection page to render even when there is only a single provider.
`AssetPublicURL`	Used for building valid client redirect URLs for external access.
`Error`	A path to a file containing a go template used to render error pages during the authentication or grant flow If unspecified, the default error page is used.
`IdentityProviders`	Ordered list of ways for a user to identify themselves.
`Login`	A path to a file containing a go template used to render the login page. If unspecified, the default login page is used.
`MasterCA`	CA for verifying the TLS connection back to the **`MasterURL`**.

Parameter Name	Description
MasterPublicURL	Used for building valid client redirect URLs for external access.
MasterURL	Used for making server-to-server calls to exchange authorization codes for access tokens.
OAuthConfig	Holds the necessary configuration options for OAuth authentication.
OAuthTemplates	Allows for customization of pages like the login page.
ProviderSelection	A path to a file containing a go template used to render the provider selection page. If unspecified, the default provider selection page is used.
SessionConfig	Holds information about configuring sessions.
Templates	Allows you to customize pages like the login page.
TokenConfig	Contains options for authorization and access tokens.

7.2.11. Project Configuration

Table 7.11. Project Configuration Parameters

Parameter Name	Description
DefaultNodeSelector	Holds default project node label selector.
ProjectConfig	Holds information about project creation and defaults.
ProjectRequestMessage	The string presented to a user if they are unable to request a project via the project request API endpoint.

Parameter Name	Description
`ProjectRequestTemplate`	The template to use for creating projects in response to **projectrequest**. It is in the format namespace/template and it is optional. If it is not specified, a default template is used.

7.2.12. Scheduler Configuration

Table 7.12. Scheduler Configuration Parameters

Parameter Name	Description
`SchedulerConfigFile`	Points to a file that describes how to set up the scheduler. If empty, you get the default scheduling rules

7.2.13. Security Allocator Configuration

Table 7.13. Security Allocator Parameters

Parameter Name	Description
`MCSAllocatorRange`	Defines the range of MCS categories that will be assigned to namespaces. The format is **`<prefix>/<numberOfLabels>[,<maxCategory>]`**. The default is **s0/2** and will allocate from c0 to c1023, which means a total of 535k labels are available (1024 choose 2 ~ 535k). If this value is changed after startup, new projects may receive labels that are already allocated to other projects. Prefix may be any valid SELinux set of terms (including user, role, and type), although leaving them as the default will allow the server to set them automatically.
`SecurityAllocator`	Controls the automatic allocation of UIDs and MCS labels to a project. If nil, allocation is disabled.
`UIDAllocatorRange`	Defines the total set of Unix user IDs (UIDs) that will be allocated to projects automatically, and the size of the block each namespace gets. For example, 1000-1999/10 will allocate ten UIDs per namespace, and will be able to allocate up to 100 blocks before running out of space. The default is to allocate from 1 billion to 2 billion in 10k blocks (which is the expected size of the ranges container images will use once user namespaces are started).

7.2.14. Service Account Configuration

Table 7.14. Service Account Configuration Parameters

Parameter Name	Description
`LimitSecretReferences`	Controls whether or not to allow a service account to reference any secret in a namespace without explicitly referencing them.
`ManagedNames`	A list of service account names that will be auto-created in every namespace. If no names are specified, the **`ServiceAccountsController`** will not be started.
`MasterCA`	The CA for verifying the TLS connection back to the master. The service account controller will automatically inject the contents of this file into pods so they can verify connections to the master.
`PrivateKeyFile`	A file containing a PEM-encoded private RSA key, used to sign service account tokens. If no private key is specified, the service account **`TokensController`** will not be started.
`PublicKeyFiles`	A list of files, each containing a PEM-encoded public RSA key. If any file contains a private key, the public portion of the key is used. The list of public keys is used to verify presented service account tokens. Each key is tried in order until the list is exhausted or verification succeeds. If no keys are specified, no service account authentication will be available.
`ServiceAccountConfig`	Holds the necessary configuration options for a service account.

7.2.15. Serving Information Configuration

Table 7.15. Serving Information Configuration Parameters

Parameter Name	Description

Parameter Name	Description
`AllowRecursiveQueries`	Allows the DNS server on the master to answer queries recursively. Note that open resolvers can be used for DNS amplification attacks and the master DNS should not be made accessible to public networks.
`BindAddress`	The **ip:port** to serve on.
`BindNetwork`	Controls limits and behavior for importing images.
`CertFile`	A file containing a PEM-encoded certificate.
`CertInfo`	TLS cert information for serving secure traffic.
`ClientCA`	The certificate bundle for all the signers that you recognize for incoming client certificates.
`dnsConfig`	Holds the necessary configuration options for DNS.
`DNSDomain`	Holds the domain suffix.
`DNSIP`	Holds the IP.
`KeyFile`	A file containing a PEM-encoded private key for the certificate specified by **`CertFile`**.
`MasterClientConnectionOverrides`	Provides overrides to the client connection used to connect to the master.
`MaxRequestsInFlight`	The number of concurrent requests allowed to the server. If zero, no limit.
`NamedCertificates`	A list of certificates to use to secure requests to specific host names.

Parameter Name	Description
`RequestTimeoutSecond`	The number of seconds before requests are timed out. The default is 60 minutes. If -1, there is no limit on requests.
`ServingInfo`	The HTTP serving information for the assets.

7.2.16. Volume Configuration

Table 7.16. Volume Configuration Parameters

Parameter Name	Description
`DynamicProvisioningEnabled`	A boolean to enable or disable dynamic provisioning. Default is **true**.
FSGroup	Can be specified to enable a quota on local storage use per unique FSGroup ID. At present this is only implemented for emptyDir volumes, and if the underlying **`volumeDirectory`** is on an XFS filesystem.
`LocalQuota`	Contains options for controlling local volume quota on the node.
`MasterVolumeConfig`	Contains options for configuring volume plug-ins in the master node.
`NodeVolumeConfig`	Contains options for configuring volumes on the node.
`VolumeConfig`	Contains options for configuring volumes on the node.
`VolumeDirectory`	The directory that volumes are stored under.

7.2.17. Audit Configuration

Audit provides a security-relevant chronological set of records documenting the sequence of activities that have affected system by individual users, administrators, or other components of the system.

Audit works at the API server level, logging all requests coming to the server. Each audit log contains two entries:

1. The request line containing:
    a. A Unique ID allowing to match the response line (see #2)
    b. The source IP of the request
    c. The HTTP method being invoked
    d. The original user invoking the operation
    e. The impersonated user for the operation (**`self`** meaning himself)
    f. The impersonated group for the operation (**`lookup`** meaning user's group)
    g. The namespace of the request or <none>
    h. The URI as requested
2. The response line containing:
    a. The the unique ID from #1
    b. The response code

Example output for user **admin** asking for a list of pods:

```
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" ip="127.0.0.1"
method="GET" user="admin" as="<self>" asgroups="<lookup>"
namespace="default" uri="/api/v1/namespaces/default/pods"
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" response="200"
```

Table 7.17. Audit Configuration Parameters

Parameter Name	Description
`Enabled`	A boolean to enable or disable audit logs. Default is **`false`**.
`AuditFilePath`	File path where the requests should be logged to. If not set, logs are printed to master logs.
`MaximumFileRetentionDays`	Specifies maximum number of days to retain old audit log files based on the time stamp encoded in their filename.
`MaximumRetainedFiles`	Specifies the maximum number of old audit log files to retain.

Parameter Name	Description
`MaximumFileSizeMegabytes`	Specifies maximum size in megabytes of the log file before it gets rotated. Defaults to 100MB.

7.3. NODE CONFIGURATION FILES

The following ***node-config.yaml*** file is a sample node configuration file that was generated with the default values as of writing. You can create a new node configuration file to see the valid options for your installed version of OpenShift Container Platform.

Example 7.1. Sample Node Configuration File

```
allowDisabledDocker: false
apiVersion: v1
authConfig:
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsDomain: cluster.local
dnsIP: 10.0.2.15 1
dockerConfig:
  execHandlerName: native
imageConfig:
  format: openshift/origin-${component}:${version}
  latest: false
iptablesSyncPeriod: 5s
kind: NodeConfig
masterKubeConfig: node.kubeconfig
networkConfig:
  mtu: 1450
  networkPluginName: ""
nodeIP: ""
nodeName: node1.example.com
podManifestConfig: 2
  path: "/path/to/pod-manifest-file" 3
  fileCheckIntervalSeconds: 30 4
proxyArguments:
  proxy-mode:
  - iptables 5
volumeConfig:
  localQuota:
   perFSGroup: null 6
servingInfo:
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  certFile: server.crt
```

```
  clientCA: node-client-ca.crt
  keyFile: server.key
  namedCertificates: null
volumeDirectory: /root/openshift.local.volumes
```

1

Configures an IP address to be prepended to a pod's ***/etc/resolv.conf*** by adding the address here.

2

Allows pods to be placed directly on certain set of nodes, or on all nodes without going through the scheduler. You can then use pods to perform the same administrative tasks and support the same services on each node.

3

Specifies the path for the pod manifest file or directory. If it is a directory, then it is expected to contain one or more manifest files. This is used by the Kubelet to create pods on the node.

4

This is the interval (in seconds) for checking the manifest file for new data. The interval must be a positive value.

5

The service proxy implementation to use.

6

Preliminary support for local emptyDir volume quotas, set this value to a resource quantity representing the desired quota per FSGroup, per node. (i.e. 1Gi, 512Mi, etc) Currently requires that the ***volumeDirectory*** be on an XFS filesystem mounted with the 'gquota' option, and the matching security context contraint's fsGroup type set to 'MustRunAs'.

7.3.1. Pod and Node Configuration

Table 7.18. Pod and Node Configuration Parameters

Parameter Name	Description
NodeConfig	The fully specified configuration starting an OpenShift Container Platform node.
NodeIP	Node may have multiple IPs, so this specifies the IP to use for pod traffic routing. If not specified, network parse/lookup on the **nodeName** is performed and the first non-loopback address is used.
NodeName	The value used to identify this particular node in the cluster. If possible, this should be your fully qualified hostname. If you are describing a set of static nodes to the master, this value must match one of the values in the list.
PodEvictionTimeout	Controls grace period for deleting pods on failed nodes. It takes valid time duration string. If empty, you get the default pod eviction timeout.
ProxyClientInfo	Specifies the client cert/key to use when proxying to pods.

7.3.2. Docker Configuration

Table 7.19. Docker Configuration Parameters

Parameter Name	Description
AllowDisabledDocker	If true, the kubelet will ignore errors from Docker. This means that a node can start on a machine that does not have docker started.
DockerConfig	Holds Docker related configuration options
ExecHandlerName	The handler to use for executing commands in Docker containers.

7.3.3. Parallel Image Pulls with Docker 1.9+

If you are using Docker 1.9+, you may want to consider enabling parallel image pulling, as the default is to pull images one at a time.

Note

There is a potential issue with data corruption prior to Docker 1.9. However, starting with 1.9, the corruption issue is resolved and it is safe to switch to parallel pulls.

```
kubeletArguments:
  serialize-image-pulls:
  - "false" 1
```

1

Change to true to disable parallel pulls. (This is the default config)

7.4. PASSWORDS AND OTHER SENSITIVE DATA

For some authentication configurations, an LDAP **`bindPassword`** or OAuth **`clientSecret`** value is required. Instead of specifying these values directly in the master configuration file, these values may be provided as environment variables, external files, or in encrypted files.

Environment Variable Example

```
...
bindPassword:
  env: BIND_PASSWORD_ENV_VAR_NAME
```

External File Example

```
...
bindPassword:
  file: bindPassword.txt
```

Encrypted External File Example

```
...
bindPassword:
  file: bindPassword.encrypted
  keyFile: bindPassword.key
```

To create the encrypted file and key file for the above example:

```
$ oadm ca encrypt --genkey=bindPassword.key --
out=bindPassword.encrypted
> Data to encrypt: B1ndPass0rd!
```

Warning

Encrypted data is only as secure as the decrypting key. Care should be taken to limit filesystem permissions and access to the key file.

7.5. CREATING NEW CONFIGURATION FILES

When defining an OpenShift Container Platform configuration from scratch, start by creating new configuration files.

For master host configuration files, use the **`openshift start`** command with the **`--write-config`** option to write the configuration files. For node hosts, use the **`oadm create-node-config`** command to write the configuration files.

The following commands write the relevant launch configuration file(s), certificate files, and any other necessary files to the specified **`--write-config`** or **`--node-dir`** directory.

To create configuration files for an all-in-one server (a master and a node on the same host) in the specified directory:

```
$ openshift start --write-config=/openshift.local.config
```

To create a master configuration file and other required files in the specified directory:

```
$ openshift start master --write-config=/openshift.local.config/master
```

To create a node configuration file and other related files in the specified directory:

```
$ oadm create-node-config \
    --node-dir=/openshift.local.config/node-<node_hostname> \
    --node=<node_hostname> \
    --hostnames=<node_hostname>,<ip_address> \
    --certificate-authority="/path/to/ca.crt" \
    --signer-cert="/path/to/ca.crt" \
    --signer-key="/path/to/ca.key"
    --signer-serial="/path/to/ca.serial.txt"
    --node-client-certificate-authority="/path/to/ca.crt"
```

When creating node configuration files, the **`--hostnames`** option accepts a comma-delimited list of every host name or IP address you want server certificates to be valid for.

7.6. LAUNCHING SERVERS USING CONFIGURATION FILES

Once you have modified the master and/or node configuration files to your specifications, you can use them when launching servers by specifying them as an argument. Keep in mind that if you specify a configuration file, none of the other command line options you pass are respected.

To launch an all-in-one server using a master configuration and a node configuration file:

```
$ openshift start --master-
config=/openshift.local.config/master/master-config.yaml --node-
config=/openshift.local.config/node-<node_hostname>/node-config.yaml
```

To launch a master server using a master configuration file:

```
$ openshift start master --
config=/openshift.local.config/master/master-config.yaml
```

To launch a node server using a node configuration file:

```
$ openshift start node --config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

CHAPTER 8. ADDING HOSTS TO AN EXISTING CLUSTER

8.1. OVERVIEW

Depending on how your OpenShift Container Platform cluster was installed, you can add new hosts (either nodes or masters) to your installation by using the install tool for quick installations, or by using the ***scaleup.yml*** playbook for advanced installations.

8.2. ADDING HOSTS USING THE QUICK INSTALLER TOOL

If you used the quick install tool to install your OpenShift Container Platform cluster, you can use the quick install tool to add a new node host to your existing cluster.

Note

Currently, you can not use the quick installer tool to add new master hosts. You must use the advanced installation method to do so.

If you used the installer in either interactive or unattended mode, you can re-run the installation as long as you have an installation configuration file at ***~/.config/openshift/installer.cfg.yml*** (or specify a different location with the **-c** option).

Important

The recommended maximum number of nodes is 300.

To add nodes to your installation:

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

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

2. Run the installer with the **scaleup** subcommand in interactive or unattended mode:

```
# atomic-openshift-installer [-u] [-c </path/to/file>] scaleup
```

3. The installer detects your current environment and allows you to add additional nodes:

```
*** Installation Summary ***

Hosts:
- 100.100.1.1
  - OpenShift master
  - OpenShift node
  - Etcd (Embedded)
  - Storage
```

```
Total OpenShift masters: 1
Total OpenShift nodes: 1


---

We have detected this previously installed OpenShift environment.

This tool will guide you through the process of adding additional
nodes to your cluster.

Are you ready to continue? [y/N]:
```

Choose (y) and follow the on-screen instructions to complete your desired task.

8.3. ADDING HOSTS USING THE ADVANCED INSTALL

If you installed using the advanced install, you can add new hosts to your cluster by running the ***scaleup.yml*** playbook. This playbook queries the master, generates and distributes new certificates for the new hosts, then runs the configuration playbooks on the new hosts only. Before running the ***scaleup.yml*** playbook, complete all prerequisite host preparation steps.

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 when 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/hosts*** (previously located at ***~/.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 a host 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_<host_type>** to the **[OSEv3:children]** section:

   For example, to add a new node host, add **new_nodes**:

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

To add new master hosts, add **new_masters**.

3. Create a **[new_<host_type>]** section much like an existing section, specifying host information for any new hosts you want to add. For example, when adding a new node:

```
[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.

When adding new masters, hosts added to the **[new_masters]** section must also be added to the **[new_nodes]** section with the **openshift_schedulable=false** variable. This ensures the new master host is part of the OpenShift SDN and that pods are not scheduled for placement on them. For example:

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
node[1:3].example.com openshift_node_labels="{'region': 'infra'}"
master[1:2].example.com openshift_schedulable=false

[new_nodes]
master3.example.com openshift_schedulable=false
```

4. Run the ***scaleup.yml*** playbook. If your inventory file is located somewhere other than the default of ***/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-
node/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_<host_type>]** section into their appropriate section (but leave the **[new_<host_type>]** 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, when adding new nodes:

```
[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]
```

CHAPTER 9. LOADING THE DEFAULT IMAGE STREAMS AND TEMPLATES

9.1. OVERVIEW

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

9.2. OFFERINGS BY SUBSCRIPTION TYPE

Depending on the active subscriptions on your Red Hat account, the following sets of image streams and templates are provided and supported by Red Hat. Contact your Red Hat sales representative for further subscription details.

9.2.1. OpenShift Container Platform Subscription

The core set of image streams and templates are provided and supported with an active *OpenShift Container Platform subscription*. This includes the following technologies:

Type	Technology
Languages & Frameworks	.NET Core Node.js Perl PHP Python Ruby
Databases	MongoDB MySQL PostgreSQL
Middleware Services	Red Hat JBoss Web Server (Tomcat) Red Hat Single Sign-on
Other Services	Jenkins

9.2.2. xPaaS Middleware Add-on Subscriptions

Support for xPaaS middleware images are provided by *xPaaS Middleware add-on subscriptions*, which are separate subscriptions for each xPaaS product. If the relevant subscription is active on your account, image streams and templates are provided and supported for the following technologies:

Type	Technology
Middleware Services	Red Hat JBoss A-MQ Red Hat JBoss BPM Suite Intelligent Process Server Red Hat JBoss BRMS Decision Server Red Hat JBoss Data Grid Red Hat JBoss EAP Red Hat JBoss Fuse Integration Services

9.3. BEFORE YOU BEGIN

Before you consider performing the tasks in this topic, confirm if these image streams and templates are already registered in your OpenShift Container Platform cluster by doing one of the following:

- Log into the web console and click **Add to Project**.
- List them for the **openshift** project using the CLI:

```
$ oc get is -n openshift
$ oc get templates -n openshift
```

If the default image streams and templates are ever removed or changed, you can follow this topic to create the default objects yourself. Otherwise, the following instructions are not necessary.

9.4. PREREQUISITES

Before you can create the default image streams and templates:

- The integrated Docker registry service must be deployed in your OpenShift Container Platform installation.
- You must be able to run the **oc create** command with **cluster-admin** privileges, because they operate on the default **openshift**project.
- You must have installed the **atomic-openshift-utils** RPM package. See Software Prerequisites for instructions.
- Define shell variables for the directories containing image streams and templates. This significantly shortens the commands in the following sections. To do this:

```
$ IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.1/image-streams";
\
    XPAASSTREAMDIR="/usr/share/ansible/openshift-
```

```
ansible/roles/openshift_examples/files/examples/v1.1/xpaas-streams";
\
    XPAASTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.1/xpaas-
templates"; \
    DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.1/db-templates";
\
    QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.1/quickstart-
templates"
```

9.5. CREATING IMAGE STREAMS FOR OPENSHIFT CONTAINER PLATFORM IMAGES

If your node hosts are subscribed using Red Hat Subscription Manager and you want to use the core set of image streams that used Red Hat Enterprise Linux (RHEL) 7 based images:

```
$ oc create -f $IMAGESTREAMDIR/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 $IMAGESTREAMDIR/image-streams-centos7.json -n openshift
```

Creating both the CentOS and RHEL sets of image streams is not possible, because they use the same names. 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.

9.6. CREATING IMAGE STREAMS FOR XPAAS MIDDLEWARE IMAGES

The xPaaS Middleware image streams provide images for **JBoss EAP**, **JBoss JWS**, **JBoss A-MQ**, **JBoss Fuse Integration Services**, **Decision Server**, and **JBoss Data Grid**. 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 $XPAASSTREAMDIR/jboss-image-streams.json -n openshift
```

Note

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

9.7. 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 Container Platform installation to have persistent volumes configured.

To create the core set of database templates:

```
$ oc create -f $DBTEMPLATES -n openshift
```

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

9.8. CREATING INSTANT APP AND QUICKSTART TEMPLATES

The Instant App and Quickstart 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.

Using these templates, users are able to easily instantiate full applications using the various language images provided with OpenShift Container Platform. 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 Instant App and Quickstart templates:

```
$ oc create -f $QSTEMPLATES -n openshift
```

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

```
$ oc create -f $XPAASTEMPLATES -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.

9.9. 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 Instant App and Quickstart Templates section in the Developer Guide for these instructions.

CHAPTER 10. CONFIGURING CUSTOM CERTIFICATES

10.1. OVERVIEW

Administrators can configure custom serving certificates for the public host names of the OpenShift Container Platform API and web console. This can be done during an advanced installation or configured after installation.

10.2. CONFIGURING CUSTOM CERTIFICATES WITH ANSIBLE

During advanced installations, custom certificates can be configured using the **openshift_master_named_certificates** and **openshift_master_overwrite_named_certificates** parameters, which are configurable in the inventory file. More details are available about configuring custom certificates with Ansible.

Example 10.1. Example Custom Certificate Configuration with Ansible

```
# Configure custom named certificates
# NOTE: openshift_master_named_certificates is cached on masters and
is an
# additive fact, meaning that each run with a different set of
certificates
# will add the newly provided certificates to the cached set of
certificates.
#
# An optional CA may be specified for each named certificate. CAs
will
# be added to the OpenShift CA bundle which allows for the named
# certificate to be served for internal cluster communication.
#
# If you would like openshift_master_named_certificates to be
overwritten with
# the provided value, specify
openshift_master_overwrite_named_certificates.
openshift_master_overwrite_named_certificates=true
#
# Provide local certificate paths which will be deployed to masters
openshift_master_named_certificates=[{"certfile":
"/path/on/host/to/custom1.crt", "keyfile":
"/path/on/host/to/custom1.key", "cafile": "/path/on/host/to/custom-
ca1.crt"}]
#
# Detected names may be overridden by specifying the "names" key
#openshift_master_named_certificates=[{"certfile":
"/path/on/host/to/custom1.crt", "keyfile":
"/path/on/host/to/custom1.key", "names": ["public-master-host.com"],
"cafile": "/path/on/host/to/custom-ca1.crt"}]
```

10.3. CONFIGURING CUSTOM CERTIFICATES

In the master configuration file you can list the **namedCertificates** section in the **assetConfig.servingInfo** section so the custom certificate serves up for the web console, and in the **servingInfo** section so the custom certificate serves up for the CLI and other API calls. Multiple certificates can be configured this way and each certificate may be associated with multiple host names or wildcards.

A default certificate must be configured in the **servingInfo.certFile** and **servingInfo.keyFile** configuration sections in addition to **namedCertificates**.

Note

The **namedCertificates** section should only be configured for the host name associated with the **masterPublicURL**, **assetConfig.publicURL**, and **oauthConfig.assetPublicURL** settings. Using a custom serving certificate for the host name associated with the **masterURL** will result in TLS errors as infrastructure components will attempt to contact the master API using the internal **masterURL** host.

Example 10.2. Custom Certificates Configuration

```
servingInfo:
  ...
  namedCertificates:
  - certFile: custom.crt
    keyFile: custom.key
    names:
    - "customhost.com"
    - "api.customhost.com"
    - "console.customhost.com"
  - certFile: wildcard.crt
    keyFile: wildcard.key
    names:
    - "*.wildcardhost.com"
  ...
```

Relative paths are resolved relative to the master configuration file. Restart the server to pick up the configuration changes.

CHAPTER 11. REDEPLOYING CERTIFICATES

11.1. OVERVIEW

This topic reviews how to back up and redeploy cluster certificates using the **`ansible-playbook`** command. This method fixes common certificate errors. Possible use cases include:

- The installer detected the wrong host names and the issue was identified too late.
- The certificates are expired and you need to update them.
- You have a new CA and would like to create certificates using it instead

11.2. RUNNING THE CERTIFICATE REDEPLOY PLAYBOOK

Running the certificate redeploy playbook will redeploy OpenShift Container Platform certificates that exist on systems (master, node, etcd):

```
$ ansible-playbook -i <inventory> playbooks/byo/openshift-
cluster/redeploy-certificates.yml
```

> **Warning**
>
> This playbook must be run with an inventory that is representative of the cluster. The inventory must specify or override all host names and IP addresses set via **`openshift_hostname`**, **`openshift_public_hostname`**, **`openshift_ip`**, **`openshift_public_ip`**, **`openshift_master_cluster_hostname`**, or **`openshift_master_cluster_public_hostname`** such that they match the current cluster configuration.

By default, the redeploy playbook does *not* redeploy the OpenShift Container Platform CA. New certificates are created using the original OpenShift Container Platform CA.

To redeploy all certificates including the OpenShift Container Platform CA, specify **`openshift_certificates_redeploy_ca=true`**.

For example:

```
$ ansible-playbook -i <inventory> playbooks/byo/openshift-
cluster/redeploy-certificates.yml \
--extra-vars "openshift_certificates_redeploy_ca=true"
```

This also adds a custom CA certificate:

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing clusters.
# This option may only be specified when creating a new cluster or
# when redeploying cluster certificates with the redeploy-certificates
```

```
# playbook.
#openshift_master_ca_certificate={'certfile': '/path/to/ca.crt',
'keyfile': '/path/to/ca.key'}
```

All pods using service accounts to communicate with the OpenShift Container Platform API must be redeployed when the OpenShift Container Platform CA is replaced so the certificate redeploy playbook will serially evacuate all nodes in the cluster when this variable is set.

CHAPTER 12. CONFIGURING AUTHENTICATION AND USER AGENT

12.1. OVERVIEW

The OpenShift Container Platform master includes a built-in OAuth server. Developers and administrators obtain OAuth access tokens to authenticate themselves to the API.

As an administrator, you can configure OAuth using the master configuration file to specify an identity provider. This can be done during an advanced installation or configured after installation.

If you installed OpenShift Container Platform using the Quick Installation or Advanced Installation method, the Deny All identity provider is used by default, which denies access for all user names and passwords. To allow access, you must choose a different identity provider and configure the master configuration file appropriately (located at ***/etc/origin/master/master-config.yaml*** by default).

When running a master without a configuration file, the Allow All identity provider is used by default, which allows any non-empty user name and password to log in. This is useful for testing purposes. To use other identity providers, or to modify any token, grant, or session options, you must run the master from a configuration file.

Note

Roles need to be assigned to administer the setup with an external user.

12.2. CONFIGURING IDENTITY PROVIDERS WITH ANSIBLE

For initial advanced installations, the Deny All identity provider is configured by default, though it can be overridden during installation using the **`openshift_master_identity_providers`** parameter, which is configurable in the inventory file. Session options in the OAuth configuration are also configurable in the inventory file.

Example 12.1. Example Identity Provider Configuration with Ansible

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'user1': '<pre-hashed password>',
'user2': '<pre-hashed password>'
# or
#openshift_master_htpasswd_file=<path to local pre-generated htpasswd
file>

# Allow all auth
#openshift_master_identity_providers=[{'name': 'allow_all', 'login':
'true', 'challenge': 'true', 'kind':
```

```
'AllowAllPasswordIdentityProvider'}]

# LDAP auth
#openshift_master_identity_providers=[{'name': 'my_ldap_provider',
'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email':
['mail'], 'name': ['cn'], 'preferredUsername': ['uid']}, 'bindDN':
'', 'bindPassword': '', 'ca': '', 'insecure': 'false', 'url':
'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]
# Configuring the ldap ca certificate
#openshift_master_ldap_ca=<ca text>
# or
#openshift_master_ldap_ca_file=<path to local ca file to use>

# Available variables for configuring certificates for other identity
providers:
#openshift_master_openid_ca
#openshift_master_openid_ca_file
#openshift_master_request_header_ca
#openshift_master_request_header_ca_file
```

12.3. IDENTITY PROVIDERS

You can configure the master host for authentication using your desired identity provider by modifying the master configuration file. The following sections detail the identity providers supported by OpenShift Container Platform.

There are four parameters common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
challenge	When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header. Not supported by all identity providers. To prevent cross-site request forgery (CSRF) attacks against browser clients Basic authentication challenges are only sent if a **X-CSRF-Token** header is present on the request. Clients that expect to receive Basic **WWW-Authenticate** challenges should set this header to a non-empty value.

Parameter	Description
login	When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider. Not supported by all identity providers. If you want users to be sent to a branded page before being redirected to the identity provider's login, then set **oauthConfig → alwaysShowProviderSelection: true** in the master configuration file. This provider selection page can be customized.
mappingMethod	Defines how new identities are mapped to users when they login. See Mapping Identities to Users for more information.

Note

When adding or changing identity providers, you can map identities from the new provider to existing users by setting the **mappingMethod** parameter to **add**.

12.3.1. Mapping Identities to Users

Setting the **mappingMethod** parameter in a master configuration file determines how identities are mapped to users:

```
...
oauthConfig:
  identityProviders:
  - name: htpasswd_auth
    challenge: true
    login: false
    mappingMethod: "claim"
...
```

When set to the default **claim** value, OAuth will fail if the identity is mapped to a previously-existing user name. The following table outlines the use cases for the available **mappingMethod** parameter values:

Parameter	Description
claim	The default value. Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity.

Parameter	Description
lookup	Looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. This allows cluster administrators to set up identities and users manually, or using an external process.
generate	Provisions a user with the identity's preferred user name. If a user with the preferred user name is already mapped to an existing identity, a unique user name is generated. For example, **myuser2**. This method should not be used in combination with external processes that require exact matches between OpenShift Container Platform user names and identity provider user names, such as LDAP group sync.
add	Provisions a user with the identity's preferred user name. If a user with that user name already exists, the identity is mapped to the existing user, adding to any existing identity mappings for the user. Required when multiple identity providers are configured that identify the same set of users and map to the same user names.

12.3.2. Allow All

Set **AllowAllPasswordIdentityProvider** in the **identityProviders** stanza to allow any non-empty user name and password to log in. This is the default identity provider when running OpenShift Container Platform without a master configuration file.

Example 12.2. Master Configuration Using AllowAllPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_allow_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider
```

1 This provider name is prefixed to provider user names to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

12.3.3. Deny All

Set **DenyAllPasswordIdentityProvider** in the **`identityProviders`** stanza to deny access for all user names and passwords.

Example 12.3. Master Configuration Using DenyAllPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_deny_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: DenyAllPasswordIdentityProvider
```

1

This provider name is prefixed to provider user names to form an identity name.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

> Controls how mappings are established between this provider's identities and user objects, as described above.

12.3.4. HTPasswd

Set **HTPasswdPasswordIdentityProvider** in the **identityProviders** stanza to validate user names and passwords against a flat file generated using **htpasswd**.

Note

The **htpasswd** utility is in the **httpd-tools** package:

```
# yum install httpd-tools
```

OpenShift Container Platform supports the Bcrypt, SHA-1, and MD5 cryptographic hash functions, and MD5 is the default for **htpasswd**. Plaintext, encrypted text, and other hash functions are not currently supported.

The flat file is reread if its modification time changes, without requiring a server restart.

To create the file, run:

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```

To add or update a login to the file, run:

```
$ htpasswd </path/to/users.htpasswd> <user_name>
```

To remove a login from the file, run:

```
$ htpasswd -D </path/to/users.htpasswd> <user_name>
```

Example 12.4. Master Configuration Using HTPasswdPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /path/to/users.htpasswd 5
```

1

This provider name is prefixed to provider user names to form an identity name.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

File generated using **`htpasswd`**.

12.3.5. Keystone

Set **KeystonePasswordIdentityProvider** in the **`identityProviders`** stanza to validate user names and passwords against an OpenStack Keystone v3 server. This enables shared authentication with an OpenStack server configured to store users in an internal Keystone database.

Example 12.5. Master Configuration Using KeystonePasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_keystone_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: KeystonePasswordIdentityProvider
      domainName: default 5
```

```
      url: http://keystone.example.com:5000 6
      ca: ca.pem 7
      certFile: keystone.pem 8
      keyFile: keystonekey.pem 9
```

1

This provider name is prefixed to provider user names to form an identity name.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

Keystone domain name. In Keystone, usernames are domain-specific. Only a single domain is supported.

6

The URL to use to connect to the Keystone server (required).

7

Optional: Certificate bundle to use to validate server certificates for the configured URL.

8

Optional: Client certificate to present when making requests to the configured URL.

9

Key for the client certificate. Required if **`certFile`** is specified.

12.3.6. LDAP Authentication

Set **LDAPPasswordIdentityProvider** in the **`identityProviders`** stanza to validate user names and passwords against an LDAPv3 server, using simple bind authentication.

During authentication, the LDAP directory is searched for an entry that matches the provided user name. If a single unique match is found, a simple bind is attempted using the distinguished name (DN) of the entry plus the provided password.

These are the steps taken:

1. Generate a search filter by combining the attribute and filter in the configured **`url`** with the user-provided user name.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny access.
3. Attempt to bind to the LDAP server using the DN of the entry retrieved from the search, and the user-provided password.
4. If the bind is unsuccessful, deny access.
5. If the bind is successful, build an identity using the configured attributes as the identity, email address, display name, and preferred user name.

The configured **`url`** is an RFC 2255 URL, which specifies the LDAP host and search parameters to use. The syntax of the URL is:

```
ldap://host:port/basedn?attribute?scope?filter
```

For the above example:

URL Component	Description
`ldap`	For regular LDAP, use the string **`ldap`**. For secure LDAP (LDAPS), use **`ldaps`** instead.
`host:port`	The name and port of the LDAP server. Defaults to **`localhost:389`** for ldap and **`localhost:636`** for LDAPS.
`basedn`	The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but it could also specify a subtree in the directory.

URL Component	Description
attribute	The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use **uid**. It is recommended to choose an attribute that will be unique across all entries in the subtree you will be using.
scope	The scope of the search. Can be either either **one** or **sub**. If the scope is not provided, the default is to use a scope of **sub**.
filter	A valid LDAP search filter. If not provided, defaults to **(objectClass=*)**

When doing searches, the attribute, filter, and provided user name are combined to create a search filter that looks like:

```
(&(<filter>)(<attribute>=<username>))
```

For example, consider a URL of:

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

When a client attempts to connect using a user name of **bob**, the resulting search filter will be **(&(enabled=true)(cn=bob))**.

If the LDAP directory requires authentication to search, specify a **bindDN** and **bindPassword** to use to perform the entry search.

Example 12.6. Master Configuration Using LDAPPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
        id: 5
        - dn
        email: 6
        - mail
        name: 7
```

```
    - cn
    preferredUsername: 8
    - uid
  bindDN: "" 9
  bindPassword: "" 10
  ca: my-ldap-ca-bundle.crt 11
  insecure: false 12
  url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13
```

1

This provider name is prefixed to the returned user ID to form an identity name.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

List of attributes to use as the identity. First non-empty attribute is used. At least one attribute is required. If none of the listed attribute have a value, authentication fails.

6

List of attributes to use as the email address. First non-empty attribute is used.

7

List of attributes to use as the display name. First non-empty attribute is used.

8

List of attributes to use as the preferred user name when provisioning a user for this

identity. First non-empty attribute is used.

9

Optional DN to use to bind during the search phase.

10

Optional password to use to bind during the search phase. This value may also be provided in an environment variable, external file, or encrypted file.

11

Certificate bundle to use to validate server certificates for the configured URL. If empty, system trusted roots are used. Only applies if **insecure: false**.

12

When **true**, no TLS connection is made to the server. When **false**, **`ldaps://`** URLs connect using TLS, and **`ldap://`** URLs are upgraded to TLS.

13

An RFC 2255 URL which specifies the LDAP host and search parameters to use, as described above.

12.3.7. Basic Authentication (Remote)

Set **BasicAuthPasswordIdentityProvider** in the **`identityProviders`** stanza to validate user names and passwords against a remote server using a server-to-server Basic authentication request. User names and passwords are validated against a remote URL that is protected by Basic authentication and returns JSON.

A **`401`** response indicates failed authentication.

A non-**`200`** status, or the presence of a non-empty "error" key, indicates an error:

```
{"error":"Error message"}
```

A **`200`** status with a **sub** (subject) key indicates success:

```
{"sub":"userid"} 1
```

1

The subject must be unique to the authenticated user and must not be able to be modified.

A successful response may optionally provide additional data, such as:

- A display name using the **name** key. For example:

```
{"sub":"userid", "name": "User Name", ...}
```

- An email address using the **email** key. For example:

```
{"sub":"userid", "email":"user@example.com", ...}
```

- A preferred user name using the **preferred_username** key. This is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity. For example:

```
{"sub":"014fbff9a07c", "preferred_username":"bob", ...}
```

Example 12.7. Master Configuration Using BasicAuthPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_remote_basic_auth_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://www.example.com/remote-idp 5
      ca: /path/to/ca.file 6
      certFile: /path/to/client.crt 7
      keyFile: /path/to/client.key 8
```

1

This provider name is prefixed to the returned user ID to form an identity name.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

URL accepting credentials in Basic authentication headers.

6

Optional: Certificate bundle to use to validate server certificates for the configured URL.

7

Optional: Client certificate to present when making requests to the configured URL.

8

Key for the client certificate. Required if **`certFile`** is specified.

12.3.8. Request Header

Set **RequestHeaderIdentityProvider** in the **`identityProviders`** stanza to identify users from request header values, such as **`X-Remote-User`**. It is typically used in combination with an authenticating proxy, which sets the request header value. This is similar to how the remote user plug-in in OpenShift Enterprise 2 allowed administrators to provide Kerberos, LDAP, and many other forms of enterprise authentication.

For users to authenticate using this identity provider, they must access *https://<master>/oauth/authorize* via an authenticating proxy. You can either proxy the entire master API server so that all access goes through the proxy, or you can configure the OAuth server to redirect unauthenticated requests to the proxy.

To redirect unauthenticated requests from clients expecting login flows:

1. Set the **`login`** parameter to **true**.
2. Set the **`provider.loginURL`** parameter to the proxy URL to send those clients to.

To redirect unauthenticated requests from clients expecting **`WWW-Authenticate`** challenges:

1. Set the **`challenge`** parameter to **true**.
2. Set the **`provider.challengeURL`** parameter to the proxy URL to send those clients to.

The **`provider.challengeURL`** and **`provider.loginURL`** parameters can include the following tokens in the query portion of the URL:

- **`${url}`** is replaced with the current URL, escaped to be safe in a query parameter.

  For example: **`https://www.example.com/sso-login?then=${url}`**

- **`${query}`** is replaced with the current query string, unescaped.

  For example: **`https://www.example.com/auth-proxy/oauth/authorize?${query}`**

Warning

If you expect unauthenticated requests to reach the OAuth server, a **`clientCA`** parameter should be set for this identity provider, so that incoming requests are checked for a valid client certificate before the request's headers are checked for a user name. Otherwise, any direct request to the OAuth server can impersonate any identity from this provider, merely by setting a request header.

Example 12.8. Master Configuration Using RequestHeaderIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_request_header_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://www.example.com/challenging-
proxy/oauth/authorize?${query}" (5)
      loginURL: "https://www.example.com/login-
proxy/oauth/authorize?${query}" (6)
      clientCA: /path/to/client-ca.file (7)
      clientCommonNames: (8)
      - my-auth-proxy
      headers: (9)
      - X-Remote-User
      - SSO-User
      emailHeaders: (10)
      - X-Remote-User-Email
      nameHeaders: (11)
      - X-Remote-User-Display-Name
      preferredUsernameHeaders: (12)
      - X-Remote-User-Login
```

1

This provider name is prefixed to the user name in the request header to form an identity name.

2

RequestHeaderIdentityProvider can only respond to clients that request **`WWW-Authenticate`** challenges by redirecting to a configured **`challengeURL`**. The configured URL should respond with a **`WWW-Authenticate`** challenge.

3

RequestHeaderIdentityProvider can only respond to clients requesting a login flow by redirecting to a configured **`loginURL`**. The configured URL should respond with a login flow.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

Optional: URL to redirect unauthenticated */oauth/authorize* requests to, for clients which expect interactive logins. *${url}* is replaced with the current URL, escaped to be safe in a query parameter. *${query}* is replaced with the current query string.

6

Optional: URL to redirect unauthenticated */oauth/authorize* requests to, for clients which expect **`WWW-Authenticate`** challenges. *${url}* is replaced with the current URL, escaped to be safe in a query parameter. *${query}* is replaced with the current query string.

7

Optional: PEM-encoded certificate bundle. If set, a valid client certificate must be presented and validated against the certificate authorities in the specified file before the request headers are checked for user names.

8

Optional: list of common names (**`cn`**). If set, a valid client certificate with a Common Name (**`cn`**) in the specified list must be presented before the request headers are checked for user names. If empty, any Common Name is allowed. Can only be used in combination with **`clientCA`**.

9

Header names to check, in order, for the user identity. The first header containing a value is used as the identity. Required, case-insensitive.

10

Header names to check, in order, for an email address. The first header containing a value is used as the email address. Optional, case-insensitive.

11

Header names to check, in order, for a display name. The first header containing a value is used as the display name. Optional, case-insensitive.

12

Header names to check, in order, for a preferred user name, if different than the immutable identity determined from the headers specified in **`headers`**. The first header containing a value is used as the preferred user name when provisioning. Optional, case-insensitive.

Example 12.9. Apache Authentication Using RequestHeaderIdentityProvider

This example configures an authentication proxy on the same host as the master. Having the proxy and master on the same host is merely a convenience and may not be suitable for your environment. For example, if you were already running a router on the master, port 443 would not be available.

It is also important to note that while this reference configuration uses Apache's **mod_auth_form**, it is by no means required and other proxies can easily be used if the following requirements are met:

1. Block the **`X-Remote-User`** header from client requests to prevent spoofing.
2. Enforce client certificate authentication in the **RequestHeaderIdentityProvider** configuration.
3. Require the **`X-Csrf-Token`** header be set for all authentication request using the challenge flow.
4. Only the */oauth/authorize* endpoint should be proxied, and redirects should not be rewritten to allow the backend server to send the client to the correct location.

Installing the Prerequisites

The **mod_auth_form** module is shipped as part of the **mod_session** package that is found in the Optional channel:

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl
```

Generate a CA for validating requests that submit the trusted header. This CA should be used as the file name for **clientCA** in the master's identity provider configuration.

```
# oadm ca create-signer-cert \
  --cert='/etc/origin/master/proxyca.crt' \
  --key='/etc/origin/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/origin/master/proxyca.serial.txt'
```

Generate a client certificate for the proxy. This can be done using any x509 certificate tooling. For convenience, the **oadm** CLI can be used:

```
# oadm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy' 1

# pushd /etc/origin/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt 2
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

1

The user name can be anything, however it is useful to give it a descriptive name as it will appear in logs.

2

When running the authentication proxy on a different host name than the master, it is important to generate a certificate that matches the host name instead of using the default master certificate as shown above. The value for **masterPublicURL** in the ***/etc/origin/master/master-config.yaml*** file must be included in the **X509v3 Subject Alternative Name** in the certificate that is specified for **SSLCertificateFile**. If a new certificate needs to be created, the **oadm ca create-server-cert** command can be used.

Configuring Apache

Unlike OpenShift Enterprise 2, this proxy does not need to reside on the same host as the master. It uses a client certificate to connect to the master, which is configured to trust the **X-Remote-User** header.

Configure Apache per the following:

```
LoadModule auth_form_module modules/mod_auth_form.so
LoadModule session_module modules/mod_session.so
LoadModule request_module modules/mod_request.so

# Nothing needs to be served over HTTP.  This virtual host simply
redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN
and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
  ServerName www.example.com

  DocumentRoot /var/www/html
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It's critical to enforce client certificates on the Master.
Otherwise
  # requests could spoof the X-Remote-User header by accessing the
Master's
  # /oauth/authorize endpoint directly.
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

  # Send all requests to the console
  RewriteEngine              On
  RewriteRule     ^/console(.*)$     https://%
{HTTP_HOST}:8443/console$1 [R,L]

  # In order to using the challenging-proxy an X-Csrf-Token must be
present.
  RewriteCond %{REQUEST_URI} ^/challenging-proxy
  RewriteCond %{HTTP:X-Csrf-Token} ^$ [NC]
  RewriteRule ^.* - [F,L]

  <Location /challenging-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize
    AuthType basic
  </Location>

  <Location /login-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize
```

```
    # mod_auth_form providers are implemented by mod_authn_dbm,
mod_authn_file,
    # mod_authn_dbd, mod_authnz_ldap and mod_authn_socache.
    AuthFormProvider file
    AuthType form
    AuthName openshift
    ErrorDocument 401 /login.html
  </Location>

  <ProxyMatch /oauth/authorize>
    AuthUserFile /etc/origin/master/htpasswd
    AuthName openshift
    Require valid-user
    RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

    # For ldap:
    # AuthBasicProvider ldap
    # AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-
domain,dc=com?uid?sub?(objectClass=*)"

    # It's possible to remove the mod_auth_form usage and replace it
with
    # something like mod_auth_kerb, mod_auth_gsspai or even
mod_auth_mellon.
    # The former would be able to support both the login and
challenge flows
    # from the Master.  Mellon would likely only support the login
flow.

    # For Kerberos
    # yum install mod_auth_gssapi
    # AuthType GSSAPI
    # GssapiCredStore keytab:/etc/httpd.keytab
  </ProxyMatch>

</VirtualHost>

RequestHeader unset X-Remote-User
```

Additional mod_auth_form Requirements

A sample login page is available from the openshift_extras repository. This file should be placed in the **`DocumentRoot`** location (***/var/www/html*** by default).

Creating Users

At this point, you can create the users in the system Apache is using to store accounts information. In this example, file-backed authentication is used:

```
# yum -y install httpd-tools
# touch /etc/origin/master/htpasswd
# htpasswd /etc/origin/master/htpasswd <user_name>
```

Configuring the Master

The **`identityProviders`** stanza in the ***/etc/origin/master/master-config.yaml*** file must be updated as well:

```
  identityProviders:
  - name: requestheader
    challenge: true
    login: true
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://[MASTER]/challenging-
proxy/oauth/authorize?${query}"
      loginURL: "https://[MASTER]/login-proxy/oauth/authorize?
${query}"
      clientCA: /etc/origin/master/proxyca.crt
      headers:
      - X-Remote-User
```

Restarting Services

Finally, restart the following services:

```
# systemctl restart httpd
# systemctl restart atomic-openshift-master
```

Verifying the Configuration

1. Test by bypassing the proxy. You should be able to request a token if you supply the correct client certificate and header:

   ```
   # curl -L -k -H "X-Remote-User: joe" \
      --cert /etc/pki/tls/certs/authproxy.pem \
      https://[MASTER]:8443/oauth/token/request
   ```

2. If you do not supply the client certificate, the request should be denied:

   ```
   # curl -L -k -H "X-Remote-User: joe" \
      https://[MASTER]:8443/oauth/token/request
   ```

3. This should show a redirect to the configured **`challengeURL`** (with additional query parameters):

   ```
   # curl -k -v -H 'X-Csrf-Token: 1' \
      '<masterPublicURL>/oauth/authorize?client_id=openshift-
   challenging-client&response_type=token'
   ```

4. This should show a 401 response with a **`WWW-Authenticate`** basic challenge:

   ```
   #  curl -k -v -H 'X-Csrf-Token: 1' \
       '<redirected challengeURL from step 3 +query>'
   ```

5. This should show a redirect with an access token:

   ```
   #  curl -k -v -u <your_user>:<your_password> \
       -H 'X-Csrf-Token: 1' '<redirected_challengeURL_from_step_3
   +query>'
   ```

12.3.9. GitHub

Set **GitHubIdentityProvider** in the **`identityProviders`** stanza to use GitHub as an identity provider, using the OAuth integration.

Note

Using GitHub as an identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Warning

Using GitHub as an identity provider allows any GitHub user to authenticate to your server. You can limit authentication to members of specific GitHub organizations with the **`organizations`** configuration attribute, as shown below.

Example 12.10. Master Configuration Using GitHubIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: github 1
    challenge: false 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      clientID: ... 5
      clientSecret: ... 6
      organizations: 7
      - myorganization1
      - myorganization2
```

1 This provider name is prefixed to the GitHub numeric user ID to form an identity name. It is also used to build the callback URL.

2 **GitHubIdentityProvider** cannot be used to send **`WWW-Authenticate`** challenges.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to GitHub to log in.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

The client ID of a registered GitHub OAuth application. The application must be configured with a callback URL of **`<master>/oauth2callback/<identityProviderName>`**.

6

The client secret issued by GitHub. This value may also be provided in an environment variable, external file, or encrypted file.

7

Optional list of organizations. If specified, only GitHub users that are members of at least one of the listed organizations will be allowed to log in. If the GitHub OAuth application configured in **clientID** is not owned by the organization, an organization owner must grant third-party access in order to use this option. This can be done during the first GitHub login by the organization's administrator, or from the GitHub organization settings.

12.3.10. GitLab

Set **GitLabIdentityProvider** in the **`identityProviders`** stanza to use GitLab.com or any other GitLab instance as an identity provider, using the OAuth integration. The OAuth provider feature requires GitLab version 7.7.0 or higher.

Note

Using GitLab as an identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Example 12.11. Master Configuration Using GitLabIdentityProvider

```
oauthConfig:
  ...
```

```
identityProviders:
- name: gitlab (1)
  challenge: true (2)
  login: true (3)
  mappingMethod: claim (4)
  provider:
    apiVersion: v1
    kind: GitLabIdentityProvider
    url: ... (5)
    clientID: ... (6)
    clientSecret: ... (7)
    ca: ... (8)
```

1

This provider name is prefixed to the GitLab numeric user ID to form an identity name. It is also used to build the callback URL.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider. This uses the Resource Owner Password Credentials grant flow to obtain an access token from GitLab.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to GitLab to log in.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

The host URL of a GitLab OAuth provider. This could either be **`https://gitlab.com/`** or any other self hosted instance of GitLab.

6

The client ID of a registered GitLab OAuth application. The application must be configured with a callback URL of **`<master>/oauth2callback/<identityProviderName>`**.

7

The client secret issued by GitLab. This value may also be provided in an environment variable, external file, or encrypted file.

8

CA is an optional trusted certificate authority bundle to use when making requests to the GitLab instance. If empty, the default system roots are used.

12.3.11. Google

Set **GoogleIdentityProvider** in the **`identityProviders`** stanza to use Google as an identity provider, using Google's OpenID Connect integration.

Note

Using Google as an identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Warning

Using Google as an identity provider allows any Google user to authenticate to your server. You can limit authentication to members of a specific hosted domain with the **`hostedDomain`** configuration attribute, as shown below.

Example 12.12. Master Configuration Using GoogleIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: google 1
    challenge: false 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: GoogleIdentityProvider
      clientID: ... 5
      clientSecret: ... 6
      hostedDomain: "" 7
```

1

This provider name is prefixed to the Google numeric user ID to form an identity name. It is also used to build the redirect URL.

2

GoogleIdentityProvider cannot be used to send **`WWW-Authenticate`** challenges.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to Google to log in.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

The client ID of a registered Google project. The project must be configured with a redirect URI of **`<master>/oauth2callback/<identityProviderName>`**.

6

The client secret issued by Google. This value may also be provided in an environment variable, external file, or encrypted file.

7

Optional hosted domain to restrict sign-in accounts to. If empty, any Google account is allowed to authenticate.

12.3.12. OpenID Connect

Set **OpenIDIdentityProvider** in the **`identityProviders`** stanza to integrate with an OpenID Connect identity provider using an Authorization Code Flow.

Note

ID Token and **UserInfo** decryptions are not supported.

By default, the **openid** scope is requested. If required, extra scopes can be specified in the **`extraScopes`** field.

Claims are read from the JWT **`id_token`** returned from the OpenID identity provider and, if specified, from the JSON returned by the **`UserInfo`** URL.

At least one claim must be configured to use as the user's identity. The standard identity claim is **`sub`**.

You can also indicate which claims to use as the user's preferred user name, display name, and email address. If multiple claims are specified, the first one with a non-empty value is used. The standard claims are:

`sub`	The user identity.
`preferred_username`	The preferred user name when provisioning a user.
`email`	Email address.
`name`	Display name.

Note

Using an OpenID Connect identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Example 12.13. Standard Master Configuration Using OpenIDIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ... (5)
      clientSecret: ... (6)
      claims:
        id:
        - sub (7)
        preferredUsername:
```

```
      - preferred_username
      name:
      - name
      email:
      - email
    urls:
      authorize: https://myidp.example.com/oauth2/authorize 8
      token: https://myidp.example.com/oauth2/token 9
```

1

This provider name is prefixed to the value of the identity claim to form an identity name. It is also used to build the redirect URL.

2

When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider. This requires the OpenID provider to support the Resource Owner Password Credentials grant flow.

3

When **true**, unauthenticated token requests from web clients (like the web console) are redirected to the authorize URL to log in.

4

Controls how mappings are established between this provider's identities and user objects, as described above.

5

The client ID of a client registered with the OpenID provider. The client must be allowed to redirect to **`<master>/oauth2callback/<identityProviderName>`**.

6

The client secret. This value may also be provided in an environment variable, external file, or encrypted file.

7

Use the value of the **`sub`** claim in the returned **`id_token`** as the user's identity.

8

Authorization Endpoint described in the OpenID spec. Must use **`https`**.

9

Token Endpoint described in the OpenID spec. Must use **`https`**.

A custom certificate bundle, extra scopes, extra authorization request parameters, and **`userInfo`** URL can also be specified:

Example 12.14. Full Master Configuration Using OpenIDIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ...
      clientSecret: ...
      ca: my-openid-ca-bundle.crt 1
      extraScopes: 2
      - email
      - profile
      extraAuthorizeParameters: 3
        include_granted_scopes: "true"
      claims:
        id: 4
        - custom_id_claim
        - sub
        preferredUsername: 5
        - preferred_username
        - email
        name: 6
        - nickname
        - given_name
        - name
        email: 7
        - custom_email_claim
        - email
      urls:
        authorize: https://myidp.example.com/oauth2/authorize
        token: https://myidp.example.com/oauth2/token
        userInfo: https://myidp.example.com/oauth2/userinfo 8
```

1

Certificate bundle to use to validate server certificates for the configured URLs. If empty, system trusted roots are used.

2

Optional list of scopes to request, in addition to the **openid** scope, during the authorization token request.

3

Optional map of extra parameters to add to the authorization token request.

4

List of claims to use as the identity. First non-empty claim is used. At least one claim is required. If none of the listed claims have a value, authentication fails.

5

List of claims to use as the preferred user name when provisioning a user for this identity. First non-empty claim is used.

6

List of claims to use as the display name. First non-empty claim is used.

7

List of claims to use as the email address. First non-empty claim is used.

8

UserInfo Endpoint described in the OpenID spec. Must use **`https`**.

12.4. TOKEN OPTIONS

The OAuth server generates two kinds of tokens:

Access tokens	Longer-lived tokens that grant access to the API.

Authorize codes	Short-lived tokens whose only use is to be exchanged for an access token.

Use the **tokenConfig** stanza to set token options:

Example 12.15. Master Configuration Token Options

```
oauthConfig:
  ...
  tokenConfig:
    accessTokenMaxAgeSeconds: 86400 1
    authorizeTokenMaxAgeSeconds: 300 2
```

1 Set **accessTokenMaxAgeSeconds** to control the lifetime of access tokens. The default lifetime is 24 hours.

2 Set **authorizeTokenMaxAgeSeconds** to control the lifetime of authorize codes. The default lifetime is five minutes.

12.5. GRANT OPTIONS

When the OAuth server receives token requests for a client to which the user has not previously granted permission, the action that the OAuth server takes is dependent on the OAuth client's grant strategy.

When the OAuth client requesting token does not provide its own grant strategy, the server-wide default strategy is used. To configure the default strategy, set the **method** value in the **grantConfig** stanza. Valid values for **method** are:

auto	Auto-approve the grant and retry the request.
prompt	Prompt the user to approve or deny the grant.
deny	Auto-deny the grant and return a failure error to the client.

Example 12.16. Master Configuration Grant Options

```
oauthConfig:
  ...
  grantConfig:
    method: auto
```

12.6. SESSION OPTIONS

The OAuth server uses a signed and encrypted cookie-based session during login and redirect flows.

Use the **`sessionConfig`** stanza to set session options:

Example 12.17. Master Configuration Session Options

```
oauthConfig:
  ...
  sessionConfig:
    sessionMaxAgeSeconds: 300 (1)
    sessionName: ssn (2)
    sessionSecretsFile: "..." (3)
```

(1) Controls the maximum age of a session; sessions auto-expire once a token request is complete. If auto-grant is not enabled, sessions must last as long as the user is expected to take to approve or reject a client authorization request.

(2) Name of the cookie used to store the session.

(3) File name containing serialized **`SessionSecrets`** object. If empty, a random signing and encryption secret is generated at each server start.

If no **`sessionSecretsFile`** is specified, a random signing and encryption secret is generated at each start of the master server. This means that any logins in progress will have their sessions invalidated if the master is restarted. It also means that if multiple masters are configured, they will not be able to decode sessions generated by one of the other masters.

To specify the signing and encryption secret to use, specify a **sessionSecretsFile**. This allows you separate secret values from the configuration file and keep the configuration file distributable, for example for debugging purposes.

Multiple secrets can be specified in the **sessionSecretsFile** to enable rotation. New sessions are signed and encrypted using the first secret in the list. Existing sessions are decrypted and authenticated by each secret until one succeeds.

Example 12.18. Session Secret Configuration:

```
apiVersion: v1
kind: SessionSecrets
secrets: 1
- authentication: "..." 2
  encryption: "..." 3
- authentication: "..."
  encryption: "..."
...
```

1

List of secrets used to authenticate and encrypt cookie sessions. At least one secret must be specified. Each secret must set an authentication and encryption secret.

2

Signing secret, used to authenticate sessions using HMAC. Recommended to use a secret with 32 or 64 bytes.

3

Encrypting secret, used to encrypt sessions. Must be 16, 24, or 32 characters long, to select AES-128, AES-192, or AES-256.

12.7. PREVENTING CLI VERSION MISMATCH WITH USER AGENT

OpenShift Container Platform implements a user agent that can be used to prevent an application developer's CLI accessing the OpenShift Container Platform API.

User agents for the OpenShift Container Platform CLI are constructed from a set of values within OpenShift Container Platform:

```
<command>/<version> (<platform>/<architecture>) <client>/<git_commit>
```

So, for example, when:

- <command> = **oc**

- <version> = The client version. For example, **v3.3.0**. This can change depending on if the request is made against the Kubernetes API at **/api**, or the OpenShift Container Platform API at **/oapi**
- <platform> = **linux**
- <architecture> = **amd64**
- <client> = **openshift**, or **kubernetes** depending on if the request is made against the Kubernetes API at **/api**, or the OpenShift Container Platform API at **/oapi**.
- <git_commit> = The Git commit of the client version (for example, **f034127**)

the user agent will be:

```
oc/v3.3.0 (linux/amd64) openshift/f034127
```

As an OpenShift Container Platform administrator, you can prevent clients from accessing the API with the **userAgentMatching** configuration setting of a master configuration. So, if a client is using a particular library or binary, they will be prevented from accessing the API.

The following user agent example denies the Kubernetes 1.2 client binary, OpenShift Origin 1.1.3 binary, and the POST and PUT **httpVerbs**:

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to
https://example.org to update it."
    deniedClients:
    - regex: '\w+/v(?:(?:1\.1\.1)|(?:1\.0\.1)) \(.+/.+\)
openshift/\w{7}'
    - regex: '\w+/v(?:1\.1\.3) \(.+/.+\) openshift/\w{7}'
      httpVerbs:
      - POST
      - PUT
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
    requiredClients: null
```

Administrators can also deny clients that do not exactly match the expected clients:

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to
https://example.org to update it."
    deniedClients: []
    requiredClients:
    - regex: '\w+/v1\.1\.3 \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
```

Note

When the client’s user agent mismatches the configuration, errors occur. To ensure that mutating requests match, enforce a whitelist. Rules are mapped to specific verbs, so you can ban mutating requests while allowing non-mutating requests.

CHAPTER 13. SYNCING GROUPS WITH LDAP

13.1. OVERVIEW

As an OpenShift Container Platform administrator, you can use groups to manage users, change their permissions, and enhance collaboration. Your organization may have already created user groups and stored them in an LDAP server. OpenShift Container Platform can sync those LDAP records with internal OpenShift Container Platform records, enabling you to manage your groups in one place. OpenShift Container Platform currently supports group sync with LDAP servers using three common schemas for defining group membership: RFC 2307, Active Directory, and augmented Active Directory.

Note

You must have **cluster-admin** privileges to sync groups.

13.2. CONFIGURING LDAP SYNC

Before you can run LDAP sync, you need a sync configuration file. This file contains LDAP client configuration details:

- Configuration for connecting to your LDAP server.
- Sync configuration options that are dependent on the schema used in your LDAP server.

A sync configuration file can also contain an administrator-defined list of name mappings that maps OpenShift Container Platform Group names to groups in your LDAP server.

13.2.1. LDAP Client Configuration

Example 13.1. LDAP Client Configuration

```
url: ldap://10.0.0.0:389 1
bindDN: cn=admin,dc=example,dc=com 2
bindPassword: password 3
insecure: true 4
ca: my-ldap-ca-bundle.crt 5
```

1

The connection protocol, IP address of the LDAP server hosting your database, and the port to connect to, formatted as **`scheme://host:port`**.

2

Optional distinguished name (DN) to use as the Bind DN. OpenShift Container Platform uses this if elevated privilege is required to retrieve entries for the sync operation.

3

Optional password to use to bind. OpenShift Container Platform uses this if elevated privilege is necessary to retrieve entries for the sync operation. This value may also be provided in an environment variable, external file, or encrypted file.

4

When **`true`**, no TLS connection is made to the server. When **`false`**, secure LDAP (**`ldaps://`**) URLs connect using TLS, and insecure LDAP (**`ldap://`**) URLs are upgraded to TLS.

5

The certificate bundle to use for validating server certificates for the configured URL. If empty, OpenShift Container Platform uses system-trusted roots. This only applies if **`insecure`** is set to **`false`**.

13.2.2. LDAP Query Definition

Sync configurations consist of LDAP query definitions for the entries that are required for synchronization. The specific definition of an LDAP query depends on the schema used to store membership information in the LDAP server.

Example 13.2. LDAP Query Definition

```
baseDN: ou=users,dc=example,dc=com 1
scope: sub 2
derefAliases: never 3
timeout: 0 4
filter: (objectClass=inetOrgPerson) 5
pageSize: 0 6
```

1

The distinguished name (DN) of the branch of the directory where all searches will start from. It is required that you specify the top of your directory tree, but you can also specify a subtree in the directory.

2

The scope of the search. Valid values are **base**, **one**, or **sub**. If this is left undefined, then a scope of **sub** is assumed. Descriptions of the scope options can be found in the table below.

3

The behavior of the search with respect to aliases in the LDAP tree. Valid values are **never**, **search**, **base**, or **always**. If this is left undefined, then the default is to **always** dereference aliases. Descriptions of the dereferencing behaviors can be found in the table below.

4

The time limit allowed for the search by the client, in seconds. A value of 0 imposes no client-side limit.

5

A valid LDAP search filter. If this is left undefined, then the default is **(objectClass=*)**.

6

The optional maximum size of response pages from the server, measured in LDAP entries. If set to 0, no size restrictions will be made on pages of responses. Setting paging sizes is necessary when queries return more entries than the client or server allow by default.

Table 13.1. LDAP Search Scope Options

LDAP Search Scope	Description
base	Only consider the object specified by the base DN given for the query.
one	Consider all of the objects on the same level in the tree as the base DN for the query.
sub	Consider the entire subtree rooted at the base DN given for the query.

Table 13.2. LDAP Dereferencing Behaviors

Dereferencing Behavior	Description
never	Never dereference any aliases found in the LDAP tree.
search	Only dereference aliases found while searching.
base	Only dereference aliases while finding the base object.
always	Always dereference all aliases found in the LDAP tree.

13.2.3. User-Defined Name Mapping

A user-defined name mapping explicitly maps the names of OpenShift Container Platform Groups to unique identifiers that find groups on your LDAP server. The mapping uses normal YAML syntax. A user-defined mapping can contain an entry for every group in your LDAP server or only a subset of those groups. If there are groups on the LDAP server that do not have a user-defined name mapping, the default behavior during sync is to use the attribute specified as the Group's name.

Example 13.3. User-Defined Name Mapping

```
groupUIDNameMapping:
  "cn=group1,ou=groups,dc=example,dc=com": firstgroup
  "cn=group2,ou=groups,dc=example,dc=com": secondgroup
  "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

13.3. RUNNING LDAP SYNC

Once you have created a sync configuration file, then sync can begin. OpenShift Container Platform allows administrators to perform a number of different sync types with the same server.

Note

By default, all group synchronization or pruning operations are dry-run, so you must set the **--confirm** flag on the **sync-groups** command in order to make changes to OpenShift Container Platform Group records.

To sync all groups from the LDAP server with OpenShift Container Platform:

```
$ oadm groups sync --sync-config=config.yaml --confirm
```

To sync all Groups already in OpenShift Container Platform that correspond to groups in the LDAP server specified in the configuration file:

```
$ oadm groups sync --type=openshift --sync-config=config.yaml --confirm
```

To sync a subset of LDAP groups with OpenShift Container Platform, you can use whitelist files, blacklist files, or both:

Note

Any combination of blacklist files, whitelist files, or whitelist literals will work; whitelist literals can be included directly in the command itself. This applies to groups found on LDAP servers, as well as Groups already present in OpenShift Container Platform. Your files must contain one unique group identifier per line.

```
$ oadm groups sync --whitelist=<whitelist_file> \
                   --sync-config=config.yaml    \
                   --confirm
$ oadm groups sync --blacklist=<blacklist_file> \
                   --sync-config=config.yaml    \
                   --confirm
$ oadm groups sync <group_unique_identifier>    \
                   --sync-config=config.yaml    \
                   --confirm
$ oadm groups sync <group_unique_identifier>    \
                   --whitelist=<whitelist_file> \
                   --blacklist=<blacklist_file> \
                   --sync-config=config.yaml    \
                   --confirm
$ oadm groups sync --type=openshift             \
                   --whitelist=<whitelist_file> \
                   --sync-config=config.yaml    \
                   --confirm
```

13.4. RUNNING A GROUP PRUNING JOB

An administrator can also choose to remove groups from OpenShift Container Platform records if the records on the LDAP server that created them are no longer present. The prune job will accept the same sync configuration file and white- or black-lists as used for the sync job.

For example, if groups had previously been synchronized from LDAP using some ***config.yaml*** file, and some of those groups no longer existed on the LDAP server, the following command would determine which Groups in OpenShift Container Platform corresponded to the deleted groups in LDAP and then remove them from OpenShift Container Platform:

```
$ oadm groups prune --sync-config=config.yaml --confirm
```

13.5. SYNC EXAMPLES

This section contains examples for the RFC 2307, Active Directory, and augmented Active Directory schemas. All of the following examples synchronize a group named **admins** that has two members: **Jane** and **Jim**. Each example explains:

- How the group and users are added to the LDAP server.
- What the LDAP sync configuration file looks like.
- What the resulting Group record in OpenShift Container Platform will be after synchronization.

13.5.1. RFC 2307

In the RFC 2307 schema, both users (Jane and Jim) and groups exist on the LDAP server as first-class entries, and group membership is stored in attributes on the group. The following snippet of `ldif` defines the users and group for this schema:

Example 13.4. LDAP Entries Using RFC 2307 Schema: *rfc2307.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (1)
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com (2)
member: cn=Jim,ou=users,dc=example,dc=com
```

1

The group is a first-class entry in the LDAP server.

2

Members of a group are listed with an identifying reference as attributes on the group.

To sync this group, you must first create the configuration file. The RFC 2307 schema requires you to provide an LDAP query definition for both user and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform records.

For clarity, the Group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of a Group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships:

Note

If using user-defined name mappings, your configuration file will differ.

Example 13.5. LDAP Sync Configuration Using RFC 2307 Schema: *rfc2307_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389 1
insecure: true 2
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn 3
    groupNameAttributes: [ cn ] 4
    groupMembershipAttributes: [ member ] 5
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userUIDAttribute: dn 6
    userNameAttributes: [ mail ] 7
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfScopeErrors: false
```

1

The IP address and host of the LDAP server where this group's record is stored.

2

When **`true`**, no TLS connection is made to the server. When **`false`**, secure LDAP (**`ldaps://`**) URLs connect using TLS, and insecure LDAP (**`ldap://`**) URLs are upgraded to TLS.

3

The attribute that uniquely identifies a group on the LDAP server. You cannot specify **`groupsQuery`** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

4

The attribute to use as the name of the Group.

5

The attribute on the group that stores the membership information.

6

The attribute that uniquely identifies a user on the LDAP server. You cannot specify **`usersQuery`** filters when using DN for userUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

7

The attribute to use as the name of the user in the OpenShift Container Platform Group record.

To run sync with the ***rfc2307_config.yaml*** file:

```
$ oadm groups sync --sync-config=rfc2307_config.yaml --confirm
```

OpenShift Container Platform creates the following Group record as a result of the above sync operation:

Example 13.6. OpenShift Container Platform Group Created Using *rfc2307_config.yaml*

```
apiVersion: v1
kind: Group
metadata:
  annotations:
```

```
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 (1)
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com (2)
    openshift.io/ldap.url: LDAP_SERVER_IP:389 (3)
  creationTimestamp:
  name: admins (4)
users: (5)
- jane.smith@example.com
- jim.adams@example.com
```

1 The last time this Group was synchronized with the LDAP server, in ISO 6801 format.

2 The unique identifier for the group on the LDAP server.

3 The IP address and host of the LDAP server where this Group's record is stored.

4 The name of the Group as specified by the sync file.

5 The users that are members of the Group, named as specified by the sync file.

13.5.1.1. RFC2307 with User-Defined Name Mappings

When syncing groups with user-defined name mappings, the configuration file changes to contain these mappings as shown below.

Example 13.7. LDAP Sync Configuration Using RFC 2307 Schema With User-Defined Name Mappings: *rfc2307_config_user_defined.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators (1)
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
```

```
    derefAliases: never
    pageSize: 0
groupUIDAttribute: dn 2
groupNameAttributes: [ cn ] 3
groupMembershipAttributes: [ member ]
usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
userUIDAttribute: dn 4
userNameAttributes: [ mail ]
tolerateMemberNotFoundErrors: false
tolerateMemberOutOfScopeErrors: false
```

1

The user-defined name mapping.

2

The unique identifier attribute that is used for the keys in the user-defined name mapping. You cannot specify **`groupsQuery`** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

3

The attribute to name OpenShift Container Platform Groups with if their unique identifier is not in the user-defined name mapping.

4

The attribute that uniquely identifies a user on the LDAP server. You cannot specify **`usersQuery`** filters when using DN for userUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

To run sync with the ***rfc2307_config_user_defined.yaml*** file:

```
$ oadm groups sync --sync-config=rfc2307_config_user_defined.yaml --
confirm
```

OpenShift Container Platform creates the following Group record as a result of the above sync operation:

Example 13.8. OpenShift Container Platform Group Created Using *rfc2307_config_user_defined.yaml*

```
apiVersion: v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: Administrators (1)
users:
- jane.smith@example.com
- jim.adams@example.com
```

1

The name of the Group as specified by the user-defined name mapping.

13.5.2. RFC 2307 with User-Defined Error Tolerances

By default, if the groups being synced contain members whose entries are outside of the scope defined in the member query, the group sync fails with an error:

```
Error determining LDAP group membership for "<group>": membership
lookup for user "<user>" in group "<group>" failed because of "search
for entry with dn="<user-dn>" would search outside of the base dn
specified (dn="<base-dn>")".
```

This often indicates a mis-configured **baseDN** in the **usersQuery** field. However, in cases where the **baseDN** intentionally does not contain some of the members of the group, setting **tolerateMemberOutOfScopeErrors: true** allows the group sync to continue. Out of scope members will be ignored.

Similarly, when the group sync process fails to locate a member for a group, it fails outright with errors:

```
Error determining LDAP group membership for "<group>": membership
lookup for user "<user>" in group "<group>" failed because of "search
for entry with base dn="<user-dn>" refers to a non-existent entry".

Error determining LDAP group membership for "<group>": membership
lookup for user "<user>" in group "<group>" failed because of "search
for entry with base dn="<user-dn>" and filter "<filter>" did not return
any results".
```

This often indicates a mis-configured **usersQuery** field. However, in cases where the group contains member entries that are known to be missing, setting **tolerateMemberNotFoundErrors: true** allows the group sync to continue. Problematic members will be ignored.

Warning

Enabling error tolerances for the LDAP group sync causes the sync process to ignore problematic member entries. If the LDAP group sync is not configured correctly, this could result in synced OpenShift Container Platform groups missing members.

Example 13.9. LDAP Entries Using RFC 2307 Schema With Problematic Group Membership: *rfc2307_problematic_users.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
member: cn=INVALID,ou=users,dc=example,dc=com 1
member: cn=Jim,ou=OUTOFSCOPE,dc=example,dc=com 2
```

1

A member that does not exist on the LDAP server.

2

A member that may exist, but is not under the **baseDN** in the user query for the sync job.

In order to tolerate the errors in the above example, the following additions to your sync configuration file must be made:

Example 13.10. LDAP Sync Configuration Using RFC 2307 Schema Tolerating Errors: *rfc2307_config_tolerating.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
insecure: true
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
    groupMembershipAttributes: [ member ]
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
    userUIDAttribute: dn 1
    userNameAttributes: [ mail ]
    tolerateMemberNotFoundErrors: true 2
    tolerateMemberOutOfScopeErrors: true 3
```

2

When **true**, the sync job tolerates groups for which some members were not found, and members whose LDAP entries are not found are ignored. The default behavior for the sync job is to fail if a member of a group is not found.

3

When **true**, the sync job tolerates groups for which some members are outside the user scope given in the **usersQuery** base DN, and members outside the member query scope are ignored. The default behavior for the sync job is to fail if a member of a group is out of scope.

1

The attribute that uniquely identifies a user on the LDAP server. You cannot specify **`usersQuery`** filters when using DN for userUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

To run sync with the ***rfc2307_config_tolerating.yaml*** file:

```
$ oadm groups sync --sync-config=rfc2307_config_tolerating.yaml --
confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

Example 13.11. OpenShift Container Platform Group Created Using *rfc2307_config.yaml*

```
apiVersion: v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: admins
users: 1
- jane.smith@example.com
- jim.adams@example.com
```

1

The users that are members of the group, as specified by the sync file. Members for which lookup encountered tolerated errors are absent.

13.5.3. Active Directory

In the Active Directory schema, both users (Jane and Jim) exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of **`ldif`** defines the users and group for this schema:

Example 13.12. LDAP Entries Using Active Directory Schema: *active_directory.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
```

```
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
testMemberOf: admins (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
testMemberOf: admins
```

1

The user's group memberships are listed as attributes on the user, and the group does not exist as an entry on the server. The **`testMemberOf`** attribute cannot be a literal attribute on the user; it can be created during search and returned to the client, but not committed to the database.

To sync this group, you must first create the configuration file. The Active Directory schema requires you to provide an LDAP query definition for user entries, as well as the attributes to represent them with in the internal OpenShift Container Platform Group records.

For clarity, the Group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of a Group by their e-mail, but define the name of the Group by the name of the group on the LDAP server. The following configuration file creates these relationships:

Example 13.13. LDAP Sync Configuration Using Active Directory Schema: *active_directory_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
insecure: true
activeDirectory:
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        filter: (objectclass=inetOrgPerson)
        pageSize: 0
    userNameAttributes: [ mail ] (1)
    groupMembershipAttributes: [ testMemberOf ] (2)
```

1

The attribute to use as the name of the user in the OpenShift Container Platform Group record.

2

The attribute on the user that stores the membership information.

To run sync with the ***active_directory_config.yaml*** file:

```
$ oadm groups sync --sync-config=active_directory_config.yaml --confirm
```

OpenShift Container Platform creates the following Group record as a result of the above sync operation:

Example 13.14. OpenShift Container Platform Group Created Using *active_directory_config.yaml*

```
apiVersion: v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: admins 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1

The last time this Group was synchronized with the LDAP server, in ISO 6801 format.

2

The unique identifier for the group on the LDAP server.

3

The IP address and host of the LDAP server where this Group's record is stored.

4

The name of the group as listed in the LDAP server.

5

The users that are members of the Group, named as specified by the sync file.

13.5.4. Augmented Active Directory

In the augmented Active Directory schema, both users (Jane and Jim) and groups exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of **`ldif`** defines the users and group for this schema:

Example 13.15. LDAP Entries Using Augmented Active Directory Schema: *augmented_active_directory.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
testMemberOf: cn=admins,ou=groups,dc=example,dc=com 1

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
testMemberOf: cn=admins,ou=groups,dc=example,dc=com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com 2
objectClass: groupOfNames
cn: admins
```

```
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
```

1

The user's group memberships are listed as attributes on the user.

2

The group is a first-class entry on the LDAP server.

To sync this group, you must first create the configuration file. The augmented Active Directory schema requires you to provide an LDAP query definition for both user entries and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform Group records.

For clarity, the Group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of a Group by their e-mail, and use the name of the Group as the common name. The following configuration file creates these relationships.

Example 13.16. LDAP Sync Configuration Using Augmented Active Directory Schema: *augmented_active_directory_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
insecure: true
augmentedActiveDirectory:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn 1
    groupNameAttributes: [ cn ] 2
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userNameAttributes: [ mail ] 3
    groupMembershipAttributes: [ testMemberOf ] 4
```

1

The attribute that uniquely identifies a group on the LDAP server. You cannot specify **`groupsQuery`** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the whitelist / blacklist method.

2

The attribute to use as the name of the Group.

3

The attribute to use as the name of the user in the OpenShift Container Platform Group record.

4

The attribute on the user that stores the membership information.

To run sync with the ***augmented_active_directory_config.yaml*** file:

```
$ oadm groups sync --sync-config=augmented_active_directory_config.yaml
--confirm
```

OpenShift Container Platform creates the following Group record as a result of the above sync operation:

Example 13.17. OpenShift Group Created Using *augmented_active_directory_config.yaml*

```
apiVersion: v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1

The last time this Group was synchronized with the LDAP server, in ISO 6801 format.

2

The unique identifier for the group on the LDAP server.

3

The IP address and host of the LDAP server where this Group's record is stored.

4

The name of the Group as specified by the sync file.

5

The users that are members of the Group, named as specified by the sync file.

CHAPTER 14. ADVANCED LDAP CONFIGURATION

14.1. OVERVIEW

OpenShift Container Platform Advanced Lightweight Directory Access Protocol (LDAP) Configuration covers the following topics:

- Setting up SSSD for LDAP Failover
- Configuring Form-Based Authentication
- Configuring Extended LDAP Attributes

14.2. SETTING UP SSSD FOR LDAP FAILOVER

14.2.1. Overview

OpenShift Container Platform provides an authentication provider for use with Lightweight Directory Access Protocol (LDAP) setups, but it can only connect to a single LDAP server. This can be problematic if that LDAP server becomes unavailable. System Security Services Daemon (SSSD) can be used to solve the issue.

Originally designed to manage local and remote authentication to the host operating system, SSSD can now be configured to provide identity, authentication, and authorization services to web services like OpenShift Container Platform. SSSD provides advantages over the built-in LDAP provider, including the ability to connect to any number of failover LDAP servers, as well as the ability to cache authentication attempts in case it can no longer reach any of those servers.

The setup for this configuration is advanced and requires a separate authentication server (also called an **authenticating proxy**) for OpenShift Container Platform to communicate with. This topic describes how to do this setup on a dedicated physical or virtual machine (VM), but the concepts are also applicable to a setup in a container.

14.2.2. Prerequisites for Authenticating Proxy Setup

1. Before starting setup, you need to know the following information about your LDAP server.
   - Whether the directory server is powered by FreeIPA, Active Directory, or another LDAP solution.
   - The Uniform Resource Identifier (URI) for the LDAP server (for example, ldap.example.com).
   - The location of the CA certificate for the LDAP server.
   - Whether the LDAP server corresponds to RFC 2307 or RFC2307bis for user groups.
2. Prepare the VMs:
   - ***proxy.example.com***: A VM to use as the authenticating proxy. This machine must have at least SSSD 1.12.0 available, which means a fairly recent operating system. This topic uses a Red Hat Enterprise Linux 7.2 server for its examples.
   - ***openshift.example.com***: A VM to use to run OpenShift Container Platform.

Note

These VMs can be configured to run on the same system, but for the examples used in this topic they are kept separate.

14.2.3. Phase 1: Certificate Generation

1. To ensure that communication between the authenticating proxy and OpenShift Container Platform is trustworthy, create a set of Transport Layer Security (TLS) certificates to use during the other phases of this setup. In the OpenShift Container Platform system, start by using the auto-generated certificates created as part of running:

```
# openshift start \
    --public-master=https://openshift.example.com:8443 \
    --write-config=/etc/origin/
```

Among other things, this generates a ***/etc/origin/master/ca.{cert|key}***. Use this signing certificate to generate keys to use on the authenticating proxy.

```
# mkdir -p /etc/origin/proxy/
# oadm ca create-server-cert \
    --cert='/etc/origin/proxy/proxy.example.com.crt' \
    --key='/etc/origin/proxy/proxy.example.com.key' \
    --hostnames=proxy.example.com \
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key='/etc/origin/master/ca.key' \
    --signer-serial='/etc/origin/master/ca.serial.txt'
```

Important

Ensure that any host names and interface IP addresses that need to access the proxy are listed. Otherwise, the HTTPS connection will fail.

2. Create a new CA to sign this client certificate:

```
# oadm ca create-signer-cert \
  --cert='/etc/origin/proxy/proxyca.crt' \
  --key='/etc/origin/proxy/proxyca.key' \
  --name='openshift-proxy-signer@UNIQUESTRING' \ 1
  --serial='/etc/origin/proxy/proxyca.serial.txt'
```

1

Make **`UNIQUESTRING`** something unique.

3. Generate the API client certificate that the authenticating proxy will use to prove its identity to OpenShift Container Platform.

```
# oadm create-api-client-config \
    --certificate-authority='/etc/origin/proxy/proxyca.crt' \
    --client-dir='/etc/origin/proxy' \
    --signer-cert='/etc/origin/proxy/proxyca.crt' \
    --signer-key='/etc/origin/proxy/proxyca.key' \
    --signer-serial='/etc/origin/proxy/proxyca.serial.txt' \
    --user='system:proxy'
```

This prevents malicious users from impersonating the proxy and sending fake identities.

4. Copy the certificate and key information to the appropriate file for future steps:

```
# cat /etc/origin/proxy/system\:proxy.crt \
      /etc/origin/proxy/system\:proxy.key \
      > /etc/origin/proxy/authproxy.pem
```

14.2.4. Phase 2: Authenticating Proxy Setup

This section guides you through the steps to authenticate the proxy setup.

14.2.4.1. Step 1: Copy Certificates

From ***openshift.example.com***, securely copy the necessary certificates to the proxy machine:

```
# scp /etc/origin/master/ca.crt \
      root@proxy.example.com:/etc/pki/CA/certs/

# scp /etc/origin/proxy/proxy.example.com.crt \
      /etc/origin/proxy/authproxy.pem \
      root@proxy.example.com:/etc/pki/tls/certs/

# scp /etc/origin/proxy/proxy.example.com.key \
      root@proxy.example.com:/etc/pki/tls/private/
```

14.2.4.2. Step 2: SSSD Configuration

1. Install a new VM with an operating system that includes 1.12.0 or later so that you can use the **mod_identity_lookup** module. The examples in this topic use a Red Hat Enterprise Linux 7.2 Server.

2. Install all of the necessary dependencies:

```
# yum install -y sssd \
                 sssd-dbus \
                 realmd \
                 httpd \
                 mod_session \
                 mod_ssl \
                 mod_lookup_identity \
                 mod_authnz_pam
```

This gives you the needed SSSD and the web server components.

3. Edit the /etc/httpd/conf.modules.d/55-authnz_pam.conf file and remove the comment from the following:

```
LoadModule authnz_pam_module modules/mod_authnz_pam.so
```

4. Set up SSSD to authenticate this VM against the LDAP server. If the LDAP server is a FreeIPA or Active Directory environment, then **realmd** can be used to join this machine to the domain.

```
# realm join ldap.example.com
```

For more advanced case, see the System-Level Authentication Guide

If you want to use SSSD to manage failover situations for LDAP, this can be configured by adding additional entries in ***/etc/sssd/sssd.conf*** on the **ldap_uri** line. Systems enrolled with FreeIPA can automatically handle failover using DNS SRV records.

5. Restart SSSD to ensure that all of the changes are applied properly:

```
$ systemctl restart sssd.service
```

6. Test that the user information can be retrieved properly:

```
$ getent passwd <username>
username:*:12345:12345:Example User:/home/username:/usr/bin/bash
```

7. Attempt to log into the VM as an LDAP user and confirm that the authentication is properly set up. This can be done via the local console or a remote service such as SSH.

Note

If you do not want LDAP users to be able to log into this machine, it is recommended to modify ***/etc/pam.d/system-auth*** and ***/etc/pam.d/password-auth*** to remove the lines containing **pam_sss.so**.

14.2.4.3. Step 3: Apache Configuration

You need to set up Apache to communicate with SSSD. Create a PAM stack file for use with Apache. To do so:

1. Create the ***/etc/pam.d/openshift*** file and add the following contents:

```
auth required pam_sss.so
account required pam_sss.so
```

This configuration enables PAM (the pluggable authentication module) to use **pam_sss.so** to determine authentication and access control when an authentication request is issued for the **openshift** stack.

2. Configure the Apache ***httpd.conf***. The steps in this section focus on setting up the challenge authentication, which is useful for logging in with **`oc login`** and similar automated tools.

Note

Configuring Form-Based Authentication explains how to set up a graphical login using SSSD as well, but it requires the rest of this setup as a prerequisite.

3. Create the new file ***openshift-proxy.conf*** in ***/etc/httpd/conf.d*** (substituting the correct host names where indicated):

```
LoadModule request_module modules/mod_request.so
LoadModule lookup_identity_module modules/mod_lookup_identity.so
# Nothing needs to be served over HTTP.  This virtual host simply
redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the
CN and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in
/etc/pki/tls/certs/proxy.example.com.crt
  ServerName proxy.example.com

  DocumentRoot /var/www/html
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/proxy.example.com.crt
  SSLCertificateKeyFile
/etc/pki/tls/private/proxy.example.com.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  # Send logs to a specific location to make them easier to find
  ErrorLog logs/proxy_error_log
  TransferLog logs/proxy_access_log
  LogLevel warn
  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It's critical to enforce client certificates on the Master.
Otherwise
  # requests could spoof the X-Remote-User header by accessing
the Master's
  # /oauth/authorize endpoint directly.
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem
```

```
  # Send all requests to the console
  RewriteEngine              On
  RewriteRule     ^/console(.*)$     https://%
{HTTP_HOST}:8443/console$1 [R,L]

  # In order to using the challenging-proxy an X-Csrf-Token must
be present.
  RewriteCond %{REQUEST_URI} ^/challenging-proxy
  RewriteCond %{HTTP:X-Csrf-Token} ^$ [NC]
  RewriteRule ^.* - [F,L]

  <Location /challenging-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://openshift.example.com:8443/oauth/authorize
    AuthType Basic
    AuthBasicProvider PAM
    AuthPAMService openshift
    Require valid-user
  </Location>

  <ProxyMatch /oauth/authorize>
    AuthName openshift
    RequestHeader set X-Remote-User %{REMOTE_USER}s
env=REMOTE_USER
  </ProxyMatch>
</VirtualHost>

RequestHeader unset X-Remote-User
```

Note

Configuring Form-Based Authentication explains how to add the **login-proxy** block to support form authentication.

4. Set a boolean to tell SELinux that it is acceptable for Apache to contact the PAM subsystem:

```
# setsebool -P allow_httpd_mod_auth_pam on
```

5. Start up Apache:

```
# systemctl start httpd.service
```

14.2.5. Phase 3: OpenShift Container Platform Configuration

This section describes how to set up an OpenShift Container Platform server from scratch in an "all in one" configuration. Master and Node Configuration provides more information on alternate configurations.

Modify the default configuration to use the new identity provider just created. To do so:

1. Modify the ***/etc/origin/master/master-config.yaml*** file.
2. Scan through it and locate the **identityProviders** section and replace it with:

```
  identityProviders:
  - name: any_provider_name
    challenge: true
    login: false
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://proxy.example.com/challenging-
proxy/oauth/authorize?${query}"
      clientCA: /etc/origin/proxy/proxyca.crt
      headers:
      - X-Remote-User
```

Note

Configuring Form-Based Authentication explains how to add the login URL to support web logins.

Configuring Extended LDAP Attributes explains how to add the email and full-name attributes. Note that the full-name attributes are only stored to the database on the first login.

3. Start OpenShift Container Platform with the updated configuration:

```
# openshift start \
    --public-master=https://openshift.example.com:8443 \
    --master-config=/etc/origin/master/master-config.yaml \
    --node-config=/etc/origin/node-node1.example.com/node-
config.yaml
```

4. Test logins:

```
oc login https://openshift.example.com:8443
```

It should now be possible to log in with only valid LDAP credentials.

14.3. CONFIGURING FORM-BASED AUTHENTICATION

14.3.1. Overview

This topic builds upon Setting up SSSD for LDAP Failover and describes how to set up form-based authentication for signing into the OpenShift Container Platform web console.

14.3.2. Prepare a Login Page

The OpenShift Container Platform upstream repositories have a template for forms. Copy that to your authenticating proxy on ***proxy.example.com***:

```
# curl -o /var/www/html/login.html \
    https://raw.githubusercontent.com/openshift/openshift-
extras/master/misc/form_auth/login.html
```

Modify this .html file to change the logo icon and "Welcome" content for your environment.

14.3.3. Install Another Apache Module

To intercept form-based authentication, install an Apache module:

```
 # yum -y install mod_intercept_form_submit
```

14.3.4. Apache Configuration

1. Modify ***/etc/httpd/conf.modules.d/55-intercept_form_submit.conf*** and uncomment the **LoadModule** line.
2. Add a new section to your ***openshift-proxy.conf*** file inside the **`<VirtualHost *:443>`** block.

```
 <Location /login-proxy/oauth/authorize>
  # Insert your backend server name/ip here.
  ProxyPass https://openshift.example.com:8443/oauth/authorize

  InterceptFormPAMService openshift
  InterceptFormLogin httpd_username
  InterceptFormPassword httpd_password

  RewriteCond %{REQUEST_METHOD} GET
  RewriteRule ^.*$ /login.html [L]
</Location>
```

   This tells Apache to listen for POST requests on the ***/login-proxy/oauth/authorize*** and to pass the user name and password over to the **openshift** PAM service.

3. Restart the service and move back over to the OpenShift Container Platform configuration.

14.3.5. OpenShift Container Platform Configuration

1. In the ***master-config.yaml*** file, update the **identityProviders** section:

```
identityProviders:
- name: any_provider_name
  challenge: true
  login: true (1)
  mappingMethod: claim
  provider:
```

```
    apiVersion: v1
    kind: RequestHeaderIdentityProvider
    challengeURL: "https://proxy.example.com/challenging-
proxy/oauth/authorize?${query}"
    loginURL: "https://proxy.example.com/login-
proxy/oauth/authorize?${query}" 2
    clientCA: /etc/origin/master/proxy/proxyca.crt
    headers:
    - X-Remote-User
```

1

Note that **login** is set to **true**, not **false**.

2

Newly added line.

2. Restart OpenShift Container Platform with the updated configuration.

Note

You should be able to browse to ***https://openshift.example.com:8443*** and use your LDAP credentials to sign in via the login form.

14.4. CONFIGURING EXTENDED LDAP ATTRIBUTES

14.4.1. Overview

This topic builds upon Setting up SSSD for LDAP Failover and Configuring Form-Based Authentication and focuses on configuring extended Lightweight Directory Access Protocol (LDAP) attributes.

14.4.2. Prerequisites

- SSSD 1.12.0 or later. This is available on Red Hat Enterprise Linux 7.0 and later.
- mod_lookup_identity 0.9.4 or later.
  - The required version is not yet available on any version of Red Hat Enterprise Linux. However, compatible packages (RPMs) are available from upstream until they arrive in Red Hat Enterprise Linux.

14.4.3. Configuring SSSD

You need to ask System Security Services Daemon (SSSD) to look up attributes in LDAP that it normally does not care about for simple system-login use-cases. In the case of OpenShift Container Platform, there is only one such attribute: email. So, you need to:

1. Modify the **[domain/DOMAINNAME]** section of ***/etc/sssd/sssd.conf*** on the authenticating proxy and add this attribute:

```
[domain/example.com]
...
ldap_user_extra_attrs = mail
```

2. Tell SSSD that it is acceptable for this attribute to be retrieved by Apache. Add the following two lines to the **[ifp]** section of ***/etc/sssd/sssd.conf***:

```
[ifp]
user_attributes = +mail
allowed_uids = apache, root
```

3. Restart SSSD:

```
# systemctl restart sssd.service
```

4. Test this configuration.

14.4.4. Configuring Apache

Now that SSSD is set up and successfully serving extended attributes, configure the web server to ask for them and to insert them in the correct places.

1. Enable the module to be loaded by Apache. To do so, modify ***/etc/httpd/conf.modules.d/55-lookup_identity.conf*** and uncomment the line:

```
LoadModule lookup_identity_module modules/mod_lookup_identity.so
```

2. Set an SELinux boolean so that SElinux allows Apache to connect to SSSD over D-BUS:

```
# setsebool -P httpd_dbus_sssd on
```

3. Edit ***/etc/httpd/conf.d/openshift-proxy.conf*** and add the following lines inside the **<ProxyMatch /oauth/authorize>** section:

```
<ProxyMatch /oauth/authorize>
  AuthName openshift

  LookupOutput Headers (1)
  LookupUserAttr mail X-Remote-User-Email (2)
  LookupUserGECOS X-Remote-User-Display-Name (3)

  RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER
</ProxyMatch>
```

1 2 3

Added line.

4. Restart Apache to pick up the changes:

```
# systemctl restart httpd.service
```

14.4.5. Configuring OpenShift Container Platform

Tell OpenShift Container Platform where to find these new attributes during login. To do so:

1. Edit the ***/etc/origin/master/master-config.yaml*** file and add the following lines to the **identityProviders** section:

```
identityProviders:
 - name: sssd
 challenge: true
 login: true
 mappingMethod: claim
 provider:
   apiVersion: v1
   kind: RequestHeaderIdentityProvider
   challengeURL: "https://proxy.example.com/challenging-
proxy/oauth/authorize?${query}"
   loginURL: "https://proxy.example.com/login-
proxy/oauth/authorize?${query}"
   clientCA:
/home/example/workspace/openshift/configs/openshift.example.com/p
roxy/proxyca.crt
   headers:
   - X-Remote-User
   emailHeaders: 1
   - X-Remote-User-Email 2
   nameHeaders: 3
   - X-Remote-User-Display-Name 4
```

1 2 3 4

Added line.

2. Launch OpenShift Container Platform with this updated configuration and log in to the web as a new user.

   You should see their full name appear in the upper-right of the screen. You can also verify with **`oc get identities -o yaml`** that both email addresses and full names are available.

14.4.6. Debugging Notes

Currently, OpenShift Container Platform only saves these attributes to the user at the time of the first login and does not update them again after that. So, while you are testing (and only while testing), run **`oc delete users,identities --all`** to clear the identities out so you can log in again.

CHAPTER 15. CONFIGURING THE SDN

15.1. OVERVIEW

The OpenShift SDN enables communication between pods across the OpenShift Container Platform cluster, establishing a *pod network*. Two SDN plug-ins are currently available (**ovs-subnet** and **ovs-multitenant**), which provide different methods for configuring the pod network.

15.2. CONFIGURING THE POD NETWORK WITH ANSIBLE

For initial advanced installations, the **ovs-subnet** plug-in is installed and configured by default, though it can be overridden during installation using the **`os_sdn_network_plugin_name`** parameter, which is configurable in the Ansible inventory file.

Example 15.1. Example SDN Configuration with Ansible

```
# Configure the multi-tenant SDN plugin (default is
'redhat/openshift-ovs-subnet')
# os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'

# Disable the OpenShift SDN plugin
# openshift_use_openshift_sdn=False

# Configure SDN cluster network CIDR block. 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.
# Can not be changed after deployment.
#osm_cluster_network_cidr=10.1.0.0/16

# default subdomain to use for exposed routes
#openshift_master_default_subdomain=apps.test.example.com

# Configure SDN cluster network and kubernetes service CIDR blocks.
These
# network blocks should be private and should not conflict with
network blocks
# in your infrastructure that pods may require access to. Can not be
changed
# after deployment.
#osm_cluster_network_cidr=10.1.0.0/16
#openshift_portal_net=172.30.0.0/16

# Configure number of bits to allocate to each host's subnet e.g. 8
# would mean a /24 network on the host.
#osm_host_subnet_length=8

# This variable specifies the service proxy implementation to use:
# either iptables for the pure-iptables version (the default),
# or userspace for the userspace proxy.
#openshift_node_proxy_mode=iptables
```

For initial quick installations, the **ovs-subnet** plug-in is installed and configured by default as well, and can be reconfigured post-installation using the **networkConfig** stanza of the ***master-config.yaml*** file.

15.3. CONFIGURING THE POD NETWORK ON MASTERS

Cluster administrators can control pod network settings on masters by modifying parameters in the **networkConfig** section of the master configuration file (located at ***/etc/origin/master/master-config.yaml*** by default):

```
networkConfig:
  clusterNetworkCIDR: 10.128.0.0/14 (1)
  hostSubnetLength: 9 (2)
  networkPluginName: "redhat/openshift-ovs-subnet" (3)
  serviceNetworkCIDR: 172.30.0.0/16 (4)
```

(1) Cluster network for node IP allocation

(2) Number of bits for pod IP allocation within a node

(3) Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in or **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in

(4) Service IP allocation for the cluster

Important

The **serviceNetworkCIDR** and **hostSubnetLength** values cannot be changed after the cluster is first created, and **clusterNetworkCIDR** can only be changed to be a larger network that still contains the original network. For example, given the default value of **10.128.0.0/14**, you could change **clusterNetworkCIDR** to **10.128.0.0/9** (i.e., the entire upper half of net 10) but not to **10.64.0.0/16**, because that does not overlap the original value.

15.4. CONFIGURING THE POD NETWORK ON NODES

Cluster administrators can control pod network settings on nodes by modifying parameters in the **networkConfig** section of the node configuration file (located at ***/etc/origin/node/node-config.yaml*** by default):

```
networkConfig:
  mtu: 1450 1
  networkPluginName: "redhat/openshift-ovs-subnet" 2
```

1

Maximum transmission unit (MTU) for the pod overlay network

2

Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in or **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in

15.5. MIGRATING BETWEEN SDN PLUG-INS

If you are already using one SDN plug-in and want to switch to another:

1. Change the **networkPluginName** parameter on all masters and nodes in their configuration files.
2. Restart the **atomic-openshift-master** service on masters and the **atomic-openshift-node** service on nodes.
3. If you are switching from an OpenShift SDN plug-in to a third-party plug-in, then clean up OpenShift SDN-specific artifacts:

```
$ oc delete clusternetwork --all
$ oc delete hostsubnets --all
$ oc delete netnamespaces --all
```

When switching from the **ovs-subnet** to the **ovs-multitenant** OpenShift SDN plug-in, all the existing projects in the cluster will be fully isolated (assigned unique VNIDs). Cluster administrators can choose to modify the project networks using the administrator CLI.

Check VNIDs by running:

```
$ oc get netnamespace
```

15.6. EXTERNAL ACCESS TO THE CLUSTER NETWORK

If a host that is external to OpenShift Container Platform requires access to the cluster network, you have two options:

1. Configure the host as an OpenShift Container Platform node but mark it unschedulable so that the master does not schedule containers on it.

2. Create a tunnel between your host and a host that is on the cluster network.

Both options are presented as part of a practical use-case in the documentation for configuring routing from an edge load-balancer to containers within OpenShift SDN.

15.7. USING FLANNEL

As an alternative to the default SDN, OpenShift Container Platform also provides Ansible playbooks for installing **flannel**-based networking. This is useful if running OpenShift Container Platform within a cloud provider platform, such as OpenStack, and you want to avoid using dual Open vSwtich SDN on both platforms.

To enable **flannel** within your OpenShift Container Platform cluster, set the following variables in your Ansible inventory file before running the installation.

```
openshift_use_openshift_sdn=false
openshift_use_flannel=true
```

Setting the **`openshift_use_openshift_sdn`** variable to false disables the default SDN and setting the **`openshift_use_flannel`** variable to true enables **flannel** in place.

CHAPTER 16. CONFIGURING NUAGE SDN

16.1. OVERVIEW

16.1.1. Overview of OpenShift Container Platform workflows

Integrating VSP with the OpenShift Container Platform application workflow allows business applications to be quickly turned up and updated by removing the network lag faced by DevOps teams. VSP supports different workflows with OpenShift Container Platform in order to accommodate scenarios where users can choose ease-of-use or complete control using policy-based automation.

Developer Workflow

This workflow is used in developer environments and requires little input from the developer in setting up the networking. In this workflow, **nuage-openshift-monitor** is responsible for creating the VSP constructs (Zone, Subnets, etc.) needed to provide appropriate policies and networking for pods created in an OpenShift Container Platform project. When a project is created, a default zone and default subnet for that project are created by **nuage-openshift-monitor**. When the default subnet created for a given project gets depleted, **nuage-openshift-monitor** dynamically creates additional subnets.

Note

A separate VSP Zone is created for each OpenShift Container Platform project ensuring isolation amongst the projects.

Operations Workflow

This workflow is used by operations teams rolling out applications.

In this workflow, the network and security policies are first configured on the VSD in accordance with the rules set by the organization to deploy applications. Administrative users can potentially create multiple zones and subnets and map them to the same project using labels. While spinning up the pods, the user can use the Nuage Labels to specify what network a pod needs to attach to and what network policies need to be applied to it. This allows for deployments where inter- and intra-project traffic can be controlled in a fine-grained manner. For example, inter-project communication is enabled on a project by project basis. This may be used to connect projects to common services that are deployed in a shared project.

16.2. INSTALLATION

The VSP integration with OpenShift Container Platform works for both virtual machines (VMs) and bare metal OpenShift Container Platform installations.

16.2.1. Installation for single master

In the Ansible nodes file, specify the following parameters in order to set up Nuage VSP as the network plug-in:

```
# Nuage specific parameters
```

```
 openshift_use_openshift_sdn=False
 openshift_use_nuage=True
 os_sdn_network_plugin_name='nuage/vsp-openshift'
 openshift_node_proxy_mode='userspace'
 nuage_openshift_monitor_rest_server_port=9443

 # VSP related parameters
 vsd_api_url=https://192.168.103.200:8443
 vsp_version=v4_0
 enterprise=nuage
 domain=openshift
 vsc_active_ip=192.168.103.201
 vsc_standby_ip=192.168.103.202
 uplink_interface=eth0

 # rpm locations
 nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_6
4/nuage-openshift-monitor-4.0.X.1830.el7.centos.x86_64.rpm
 vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-
openvswitch-4.0.X.225.el7.x86_64.rpm
 plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-
openshift-4.0.X1830.el7.centos.x86_64.rpm

 # Optional parameters
 nuage_interface_mtu=1460
 nuage_master_adminusername=admin
 nuage_master_adminuserpasswd=admin
 nuage_master_cspadminpasswd=admin
 nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor

 # Required for brownfield install (where a {product-title} cluster
exists without Nuage as the networking plugin)
 nuage_dockker_bridge=lbr0
```

16.2.2. Installation for multiple masters (HA)

An environment with High Availability (HA) can be configured with multiple masters and multiple nodes.

Nuage VSP integration in multi-master mode only supports the native HA configuration method described in this section. This can be combined with any load balancing solution, the default being HAProxy. The inventory file contains three master hosts, the nodes, an etcd server, and a host that functions as the HAProxy to balance the master API on all master hosts. The HAProxy host is defined in the [lb] section of the inventory file enabling Ansible to automatically install and configure HAProxy as the load balancing solution.

In the Ansible nodes file, the following parameters need to be specified in order to setup Nuage VSP as the network plug-in:

```
 # Create and OSEv3 group that contains masters, nodes, load-balancers,
and etcd hosts
 masters
 nodes
 etcd
 lb
```

```
# Nuage specific parameters
openshift_use_openshift_sdn=False
openshift_use_nuage=True
os_sdn_network_plugin_name='nuage/vsp-openshift'
openshift_node_proxy_mode='userspace'

# VSP related parameters
vsd_api_url=https://192.168.103.200:8443
vsp_version=v4_0
enterprise=nuage
domain=openshift
vsc_active_ip=192.168.103.201
vsc_standby_ip=192.168.103.202
uplink_interface=eth0

# rpm locations
nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openshift-monitor-4.0.X.1830.el7.centos.x86_64.rpm
vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openvswitch-4.0.X.225.el7.x86_64.rpm
plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-openshift-4.0.X1830.el7.centos.x86_64.rpm

# Required for Nuage Monitor REST server and HA
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.nuageopenshift.com
openshift_master_cluster_public_hostname=lb.nuageopenshift.com
nuage_openshift_monitor_rest_server_port=9443

# Optional parameters
nuage_interface_mtu=1460
nuage_master_adminusername='admin's user-name'
nuage_master_adminuserpasswd='admin's password'
nuage_master_cspadminpasswd='csp admin password'
nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor

# Required for brownfield install (where a {product-title} cluster exists without Nuage as the networking plugin)
nuage_dockker_bridge=lbr0

# Specify master hosts
[masters]
fqdn_of_master_1
fqdn_of_master_2
fqdn_of_master_3

# Specify load balancer host
[lb]
fqdn_of_load_balancer
```

CHAPTER 17. CONFIGURING FOR AWS

17.1. OVERVIEW

OpenShift Container Platform can be configured to access an AWS EC2 infrastructure, including using AWS volumes as persistent storage for application data. After AWS is configured properly, some additional configurations will need to be completed on the OpenShift Container Platform hosts.

17.2. CONFIGURING AWS VARIABLES

To set the required AWS variables, create a ***/etc/aws/aws.conf*** file with the following contents on all of your OpenShift Container Platform hosts, both masters and nodes:

```
[Global]
Zone = us-east-1c (1)
```

1

This is the Availability Zone of your AWS Instance and where your EBS Volume resides; this information is obtained from the AWS Managment Console.

17.3. CONFIGURING OPENSHIFT CONTAINER PLATFORM MASTERS FOR AWS

You can set the AWS configuration on your OpenShift Container Platform master hosts in two ways:

- using Ansible and the advanced installation tool
- manually, by modifying the ***master-config.yaml*** file

17.3.1. Configuring OpenShift Container Platform for AWS with Ansible

During advanced installations, AWS can be configured using the **`openshift_cloudprovider_aws_access_key`**, **`openshift_cloudprovider_aws_secret_key`**, and **`openshift_cloudprovider_kind`** parameters, which are configurable in the inventory file.

Example 17.1. Example AWS Configuration with Ansible

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_aws_access_key="{{
lookup('env','AWS_ACCESS_KEY_ID') }}"
#openshift_cloudprovider_aws_secret_key="{{
```

```
lookup('env','AWS_SECRET_ACCESS_KEY') }}"
#
# AWS
#openshift_cloudprovider_kind=aws
# Note: IAM profiles may be used instead of storing API credentials
on disk.
#openshift_cloudprovider_aws_access_key=aws_access_key_id
#openshift_cloudprovider_aws_secret_key=aws_secret_access_key
```

Note

When Ansible configures AWS, the following files are created for you:

- ***/etc/aws/aws.conf***
- ***/etc/origin/master/master-config.yaml***
- ***/etc/origin/node/node-config.yaml***
- ***/etc/sysconfig/atomic-openshift-master***
- ***/etc/sysconfig/atomic-openshift-node***

17.3.2. Manually Configuring OpenShift Container Platform Masters for AWS

Edit or create the master configuration file on all masters (***/etc/origin/master/master-config.yaml*** by default) and update the contents of the **`apiServerArguments`** and **`controllerArguments`** sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/aws/aws.conf"
  controllerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/aws/aws.conf"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***aws.conf*** should be in ***/etc/origin/*** instead of ***/etc/***.

17.3.3. Manually Configuring OpenShift Container Platform Nodes for AWS

Edit or create the node configuration file on all nodes (***/etc/origin/node/node-config.yaml*** by default) and update the contents of the **`kubeletArguments`** section:

```
kubeletArguments:
  cloud-provider:
    - "aws"
  cloud-config:
    - "/etc/aws/aws.conf"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***aws.conf*** should be in ***/etc/origin/*** instead of ***/etc/***.

17.4. SETTING KEY VALUE ACCESS PAIRS

Make sure the following environment variables are set in the ***/etc/sysconfig/atomic-openshift-master*** file on masters and the ***/etc/sysconfig/atomic-openshift-node*** file on nodes:

```
AWS_ACCESS_KEY_ID=<key_ID>
AWS_SECRET_ACCESS_KEY=<secret_key>
```

Note

Access keys are obtained when setting up your AWS IAM user.

17.5. APPLYING CONFIGURATION CHANGES

Start or restart OpenShift Container Platform services on all master and node hosts to apply your configuration changes:

```
# systemctl restart atomic-openshift-master
# systemctl restart atomic-openshift-node
```

Switching from not using a cloud provider to using a cloud provider produces an error message. Adding the cloud provider tries to delete the node because the node switches from using the **hostname** as the **`externalID`** (which would have been the case when no cloud provider was being used) to using the AWS **`instance-id`** (which is what the AWS cloud provider specifies). To resolve this issue:

1. Log in to the CLI as a cluster administrator.
2. Delete the nodes:

   ```
   $ oc delete node <node_name>
   ```

3. On each node host, restart the **atomic-openshift-node** service.

4. Add back any labels on each node that you previously had.

CHAPTER 18. CONFIGURING FOR OPENSTACK

18.1. OVERVIEW

When deployed on OpenStack, OpenShift Container Platform can be configured to access OpenStack infrastructure, including using OpenStack Cinder volumes as persistent storage for application data.

18.2. CONFIGURING OPENSTACK VARIABLES

To set the required OpenStack variables, create a ***/etc/cloud.conf*** file with the following contents on all of your OpenShift Container Platform hosts, both masters and nodes:

```
[Global]
auth-url = <OS_AUTH_URL>
username = <OS_USERNAME>
password = <password>
tenant-id = <OS_TENANT_ID>
region = <OS_REGION_NAME>

[LoadBalancer]
subnet-id = <UUID of the load balancer subnet>
```

Consult your OpenStack administrators for values of the **OS**_ variables, which are commonly used in OpenStack configuration.

18.3. CONFIGURING OPENSHIFT CONTAINER PLATFORM MASTERS FOR OPENSTACK

You can set an OpenStack configuration on your OpenShift Container Platform master and node hosts in two different ways:

- Using Ansible and the advanced installation tool
- Manually, by modifying the ***master-config.yaml*** and ***node-config.yaml*** files.

18.3.1. Configuring OpenShift Container Platform for OpenStack with Ansible

During advanced installations, OpenStack can be configured using the following parameters, which are configurable in the inventory file:

- **openshift_cloudprovider_kind**
- **openshift_cloudprovider_openstack_auth_url**
- **openshift_cloudprovider_openstack_username**
- **openshift_cloudprovider_openstack_password**
- **openshift_cloudprovider_openstack_tenant_id**
- **openshift_cloudprovider_openstack_tenant_name**

- **openshift_cloudprovider_openstack_region**
- **openshift_cloudprovider_openstack_lb_subnet_id**

Example 18.1. Example OpenStack Configuration with Ansible

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_openstack_username="{{
lookup('env','USERNAME') }}"
#openshift_cloudprovider_openstack_password="{{
lookup('env','PASSWORD') }}"
#
# Openstack
#openshift_cloudprovider_kind=openstack
#openshift_cloudprovider_openstack_auth_url=http://openstack.example.
com:35357/v2.0/
#openshift_cloudprovider_openstack_username=username
#openshift_cloudprovider_openstack_password=password
#openshift_cloudprovider_openstack_tenant_id=tenant_id
#openshift_cloudprovider_openstack_tenant_name=tenant_name
#openshift_cloudprovider_openstack_region=region
#openshift_cloudprovider_openstack_lb_subnet_id=subnet_id
```

18.3.2. Manually Configuring OpenShift Container Platform Masters for OpenStack

Edit or create the master configuration file on all masters (***/etc/origin/master/master-config.yaml*** by default) and update the contents of the **apiServerArguments** and **controllerArguments** sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "openstack"
    cloud-config:
      - "/etc/cloud.conf"
  controllerArguments:
    cloud-provider:
      - "openstack"
    cloud-config:
      - "/etc/cloud.conf"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***cloud.conf*** should be in ***/etc/origin/*** instead of ***/etc/***.

18.3.3. Manually Configuring OpenShift Container Platform Nodes for OpenStack

Edit or create the node configuration file on all nodes (***/etc/origin/node/node-config.yaml*** by default) and update the contents of the **`kubeletArguments`** and **`nodeName`** sections:

```
nodeName:
  <instance_name> 1

kubeletArguments:
  cloud-provider:
    - "openstack"
  cloud-config:
    - "/etc/cloud.conf"
```

1

Name of the OpenStack instance where the node runs (i.e., name of the virtual machine)

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***cloud.conf*** should be in ***/etc/origin/*** instead of ***/etc/***.

CHAPTER 19. CONFIGURING FOR GCE

19.1. OVERVIEW

OpenShift Container Platform can be configured to access an GCE infrastructure, including using GCE volumes as persistent storage for application data. After GCE is configured properly, some additional configurations will need to be completed on the OpenShift Container Platform hosts.

19.2. CONFIGURING MASTERS

Edit or create the master configuration file on all masters (***/etc/origin/master/master-config.yaml*** by default) and update the contents of the **`apiServerArguments`** and **`controllerArguments`** sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "gce"
  controllerArguments:
    cloud-provider:
      - "gce"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***master-config.yaml*** should be in ***/etc/origin/master*** instead of ***/etc/***.

19.3. CONFIGURING NODES

Edit or create the node configuration file on all nodes (***/etc/origin/node/node-config.yaml*** by default) and update the contents of the **`kubeletArguments`** section:

```
kubeletArguments:
  cloud-provider:
    - "gce"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***node-config.yaml*** should be in ***/etc/origin/node*** instead of ***/etc/***.

Then, start or restart the OpenShift Container Platform services on the master and all nodes.

CHAPTER 20. CONFIGURING FOR AZURE

20.1. OVERVIEW

OpenShift Container Platform can be configured to access an Azure infrastructure, including using Azure disk as persistent storage for application data. After Azure is configured properly, some additional configurations need to be completed on the OpenShift Container Platform hosts.

20.2. CONFIGURING MASTERS

Edit or create the master configuration file on all masters (***/etc/origin/master/master-config.yaml*** by default) and update the contents of the **`apiServerArguments`** and **`controllerArguments`** sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "azure"
      cloud-config:
        - "/etc/azure/azure.conf"
  controllerArguments:
    cloud-provider:
      - "azure"
      cloud-config:
        - "/etc/azure/azure.conf"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***master-config.yaml*** should be in ***/etc/origin/master*** instead of ***/etc/***.

20.3. CONFIGURING NODES

1. Edit or create the node configuration file on all nodes (***/etc/origin/node/node-config.yaml*** by default) and update the contents of the **`kubeletArguments`** section:

```
kubeletArguments:
  cloud-provider:
    - "azure"
    cloud-config:
      - "/etc/azure/azure.conf"
```

Important

When triggering a containerized installation, only the directories of ***/etc/origin*** and ***/var/lib/origin*** are mounted to the master and node container. Therefore, ***node-config.yaml*** should be in ***/etc/origin/node*** instead of ***/etc/***.

2. Start or restart the OpenShift Container Platform services on the master and all nodes.

CHAPTER 21. CONFIGURING PERSISTENT STORAGE

21.1. OVERVIEW

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

These topics show how to configure persistent volumes in OpenShift Container Platform using the following supported volume plug-ins:

- NFS
- GlusterFS
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel
- Dynamic Provisioning and Creating Storage Classes
- Volume Security
- Selector-Label Volume Binding

21.2. PERSISTENT STORAGE USING NFS

21.2.1. Overview

OpenShift Container Platform clusters can be provisioned with persistent storage using NFS. Persistent volumes (PVs) and persistent volume claims (PVCs) provide a convenient method for sharing a volume across a project. While the NFS-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

This topic covers the specifics of using the NFS persistent storage type. Some familiarity with OpenShift Container Platform and NFS is beneficial. See the Persistent Storage concept topic for details on the OpenShift Container Platform persistent volume (PV) framework in general.

21.2.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. To provision NFS volumes, a list of NFS servers and export paths are all that is required.

You must first create an object definition for the PV:

Example 21.1. PV Object Definition Using NFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Recycle 7
```

1

The name of the volume. This is the PV identity in various **oc <command> pod** commands.

2

The amount of storage allocated to this volume.

3

Though this appears to be related to controlling access to the volume, it is actually used similarly to labels and used to match a PVC to a PV. Currently, no access rules are enforced based on the **accessModes**.

4

The volume type being used, in this case the **nfs** plug-in.

5

The path that is exported by the NFS server.

6

The host name or IP address of the NFS server.

7

The reclaim policy for the PV. This defines what happens to a volume when released from its claim. Valid options are **Retain** (default) and **Recycle**. See Reclaiming Resources.

Note

Each NFS volume must be mountable by all schedulable nodes in the cluster.

Save the definition to a file, for example ***nfs-pv.yaml***, and create the PV:

```
$ oc create -f nfs-pv.yaml
persistentvolume "pv0001" created
```

Verify that the PV was created:

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES   STATUS
CLAIM     REASON    AGE
pv0001                   <none>    5368709120   RWO
Available                          31s
```

The next step can be to create a persistent volume claim (PVC) which will bind to the new PV:

Example 21.2. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce 1
  resources:
    requests:
      storage: 1Gi 2
```

1

As mentioned above for PVs, the **accessModes** do not enforce security, but rather act as labels to match a PV to a PVC.

2

This claim will look for PVs offering **1Gi** or greater capacity.

Save the definition to a file, for example ***nfs-claim.yaml***, and create the PVC:

```
# oc create -f nfs-claim.yaml
```

21.2.3. Enforcing Disk Quotas

You can use disk partitions to enforce disk quotas and size constraints. Each partition can be its own export. Each export is one PV. OpenShift Container Platform enforces unique names for PVs, but the uniqueness of the NFS volume's server and path is up to the administrator.

Enforcing quotas in this way allows the developer to request persistent storage by a specific amount (for example, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

21.2.4. NFS Volume Security

This section covers NFS volume security, including matching permissions and SELinux considerations. The reader is expected to understand the basics of POSIX permissions, process UIDs, supplemental groups, and SELinux.

Note

See the full Volume Security topic before implementing NFS volumes.

Developers request NFS storage by referencing, in the **`volumes`** section of their pod definition, either a PVC by name or the NFS volume plug-in directly.

The ***/etc/exports*** file on the NFS server contains the accessible NFS directories. The target NFS directory has POSIX owner and group IDs. The OpenShift Container Platform NFS plug-in mounts the container's NFS directory with the same POSIX ownership and permissions found on the exported NFS directory. However, the container is not run with its effective UID equal to the owner of the NFS mount, which is the desired behavior.

As an example, if the target NFS directory appears on the NFS server as:

```
# ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs

# id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

Then the container must match SELinux labels, and either run with a UID of **65534** (**nfsnobody** owner) or with **5555** in its supplemental groups in order to access the directory.

Note

The owner ID of 65534 is used as an example. Even though NFS's **root_squash** maps **root** (0) to **nfsnobody** (65534), NFS exports can have arbitrary owner IDs. Owner 65534 is not required for NFS exports.

21.2.4.1. Group IDs

The recommended way to handle NFS access (assuming it is not an option to change permissions on the NFS export) is to use supplemental groups. Supplemental groups in OpenShift Container Platform are used for shared storage, of which NFS is an example. In contrast, block storage, such as Ceph RBD or iSCSI, use the **fsGroup** SCC strategy and the **fsGroup** value in the pod's **`securityContext`**.

Note

It is generally preferable to use supplemental group IDs to gain access to persistent storage versus using user IDs. Supplemental groups are covered further in the full Volume Security topic.

Because the group ID on the example target NFS directory shown above is 5555, the pod can define that group ID using **`supplementalGroups`** under the pod-level **`securityContext`** definition. For example:

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1

`securityContext` must be defined at the pod level, not under a specific container.

2

An array of GIDs defined for the pod. In this case, there is one element in the array; additional GIDs would be comma-separated.

Assuming there are no custom SCCs that might satisfy the pod's requirements, the pod will likely match the **restricted** SCC. This SCC has the **`supplementalGroups`** strategy set to **RunAsAny**, meaning that any supplied group ID will be accepted without range checking.

As a result, the above pod will pass admissions and will be launched. However, if group ID range checking is desired, a custom SCC, as described in pod security and custom SCCs, is the preferred solution. A custom SCC can be created such that minimum and maximum group IDs are defined, group ID range checking is enforced, and a group ID of 5555 is allowed.

21.2.4.2. User IDs

User IDs can be defined in the container image or in the pod definition. The full Volume Security topic covers controlling storage access based on user IDs, and should be read prior to setting up NFS persistent storage.

Note

It is generally preferable to use supplemental group IDs to gain access to persistent storage versus using user IDs.

In the example target NFS directory shown above, the container needs its UID set to 65534 (ignoring group IDs for the moment), so the following can be added to the pod definition:

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

1

Pods contain a **`securtityContext`** specific to each container (shown here) and a pod-level **`securityContext`** which applies to all containers defined in the pod.

2

65534 is the **nfsnobody** user.

Assuming the **default** project and the **restricted** SCC, the pod's requested user ID of 65534 will, unfortunately, not be allowed, and therefore the pod will fail. The pod fails because of the following:

- It requests 65534 as its user ID.
- All SCCs available to the pod are examined to see which SCC will allow a user ID of 65534 (actually, all policies of the SCCs are checked but the focus here is on user ID).
- Because all available SCCs use **MustRunAsRange** for their **`runAsUser`** strategy, UID range checking is required.
- 65534 is not included in the SCC or project's user ID range.

It is generally considered a good practice not to modify the predefined SCCs. The preferred way to fix this situation is to create a custom SCC, as described in the full Volume Security topic. A custom SCC can be created such that minimum and maximum user IDs are defined, UID range checking is still enforced, and the UID of 65534 will be allowed.

21.2.4.3. SELinux

Note

See the full Volume Security topic for information on controlling storage access in conjunction with using SELinux.

By default, SELinux does not allow writing from a pod to a remote NFS server. The NFS volume mounts correctly, but is read-only.

To enable writing to NFS volumes with SELinux enforcing on each node, run:

```
# setsebool -P virt_use_nfs 1
# setsebool -P virt_sandbox_use_nfs 1
```

The **-P** option above makes the bool persistent between reboots.

The **virt_use_nfs** boolean is defined by the ***docker-selinux*** package. If an error is seen indicating that this bool is not defined, ensure this package has been installed.

21.2.4.4. Export Settings

In order to enable arbitrary container users to read and write the volume, each exported volume on the NFS server should conform to the following conditions:

- Each export must be:

  ```
  /<example_fs> *(rw,root_squash,no_wdelay)
  ```

  The **no_wdelay** option prevents the server from delaying writes, which greatly improves read-after-write consistency.

- The firewall must be configured to allow traffic to the mount point. For NFSv4, the default port is 2049 (**nfs**). For NFSv3, there are three ports to configure: 2049 (**nfs**), 20048 (**mountd**), and 111 (**portmapper**).

  NFSv4

  ```
  # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
  ```

  NFSv3

  ```
  # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
  # iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
  # iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
  ```

- The NFS export and directory must be set up so that it is accessible by the target pods. Either set the export to be owned by the container's primary UID, or supply the pod group access using **supplementalGroups**, as shown in Group IDs above. See the full Volume Security topic for additional pod security information as well.

21.2.5. Reclaiming Resources

NFS implements the OpenShift Container Platform **Recyclable** plug-in interface. Automatic processes handle reclamation tasks based on policies set on each persistent volume.

By default, persistent volumes are set to **Retain**. NFS volumes which are set to **Recycle** are scrubbed (i.e., `rm -rf` is run on the volume) after being released from their claim (i.e, after the user's `PersistentVolumeClaim` bound to the volume is deleted). Once recycled, the NFS volume can be bound to a new claim.

21.2.6. Automation

Clusters can be provisioned with persistent storage using NFS in the following ways:

- Enforce storage quotas using disk partitions.
- Enforce security by restricting volumes to the project that has a claim to them.
- Configure reclamation of discarded resources for each PV.

They are many ways that you can use scripts to automate the above tasks. You can use an example Ansible playbook to help you get started.

21.2.7. Additional Configuration and Troubleshooting

Depending on what version of NFS is being used and how it is configured, there may be additional configuration steps needed for proper export and security mapping. The following are some that may apply:

NFSv4 mount incorrectly shows all files with ownership of **nobody:nobody**	- Could be attributed to the ID mapping settings (/etc/idmapd.conf) on your NFS - See this Red Hat Solution.
Disabling ID mapping on NFSv4	- On both the NFS client and server, run: `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

21.3. PERSISTENT STORAGE USING GLUSTERFS

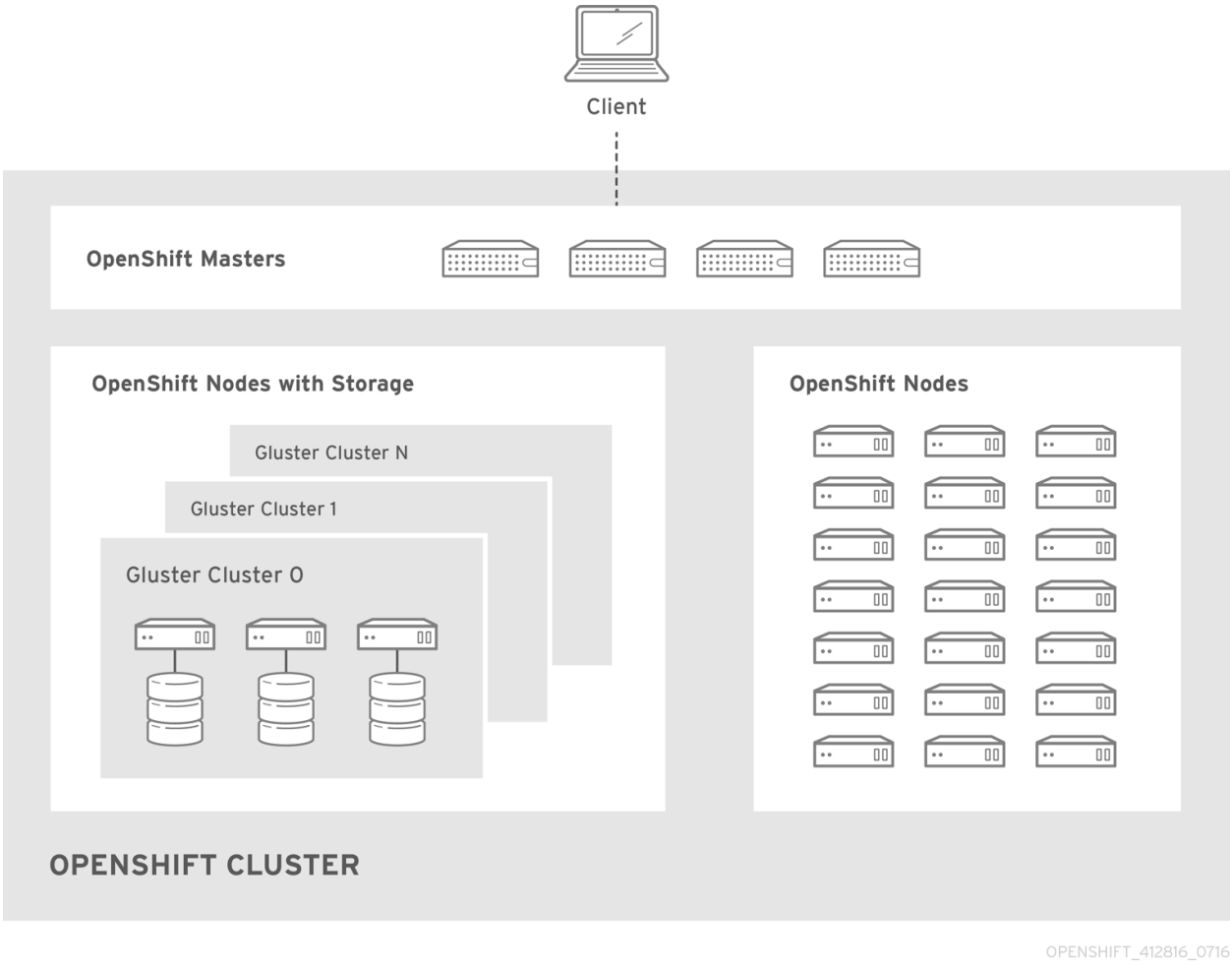
21.3.1. Overview

You can configure your OpenShift Container Platform cluster to use Red Hat Gluster Storage as persistent storage for containerized applications. There are two deployment solutions available when using Red Hat Gluster Storage, using either a containerized or dedicated storage cluster. This topic focuses mainly on the the persistent volume plug-in solution using a dedicated Red Hat Gluster Storage cluster.

21.3.1.1. Containerized Red Hat Gluster Storage

Starting with the Red Hat Gluster Storage 3.1 update 3 release, you can deploy containerized Red Hat Gluster Storage directly on OpenShift Container Platform. Containerized Red Hat Gluster Storage converged with OpenShift Container Platform addresses the use case where containerized applications require both shared file storage and the flexibility of a converged infrastructure with compute and storage instances being scheduled and run from the same set of hardware.

Figure 21.1. Architecture - Red Hat Gluster Storage Container Converged with OpenShift



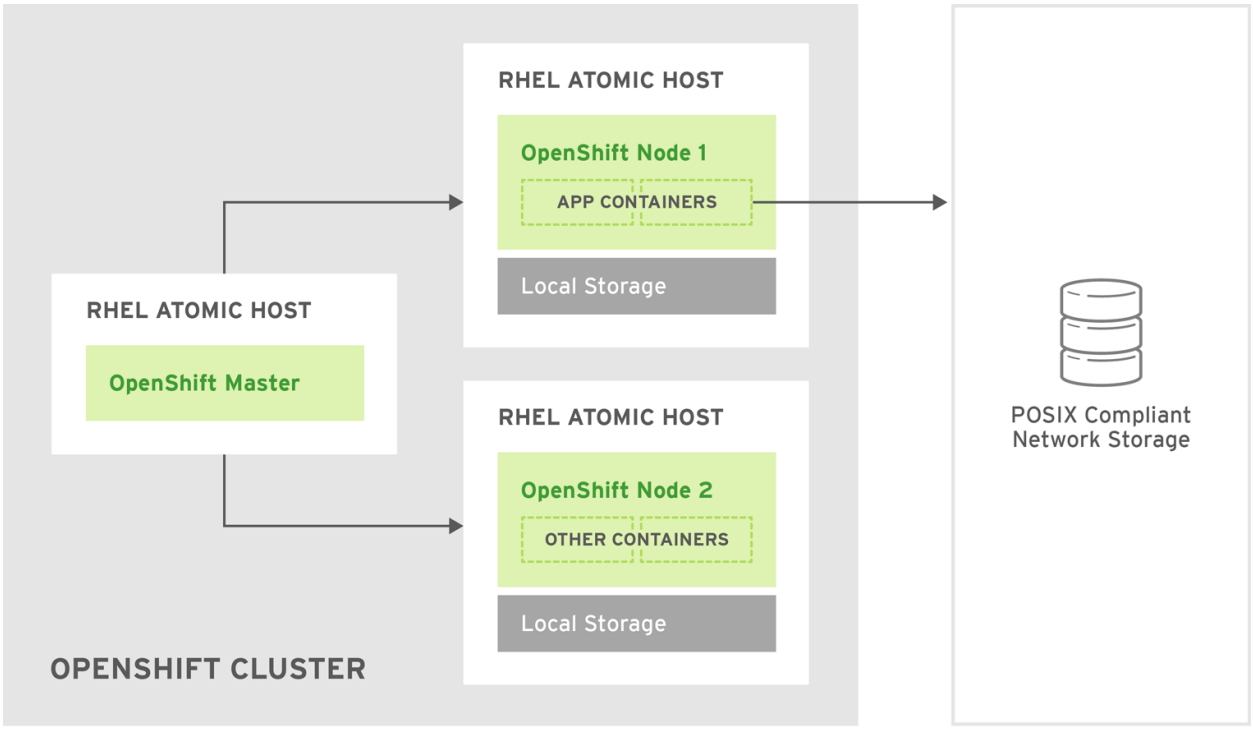
Step-by-step instructions for this containerized solution are provided separately in the following documentation:

Deployment Guide for Containerized Red Hat Gluster Storage

21.3.1.2. Dedicated Storage Cluster

If you have a dedicated Red Hat Gluster Storage cluster available in your environment, you can configure OpenShift Container Platform's Gluster volume plug-in. The dedicated storage cluster delivers persistent Red Hat Gluster Storage file storage for containerized applications over the network. The applications access storage served out from the storage clusters through common storage protocols.

Figure 21.2. Architecture - Dedicated Red Hat Gluster Storage Cluster Using the OpenShift Container Platform Volume Plug-in



This solution is a conventional deployment where containerized compute applications run on an OpenShift Container Platform cluster. The remaining sections in this topic provide the step-by-step instructions for the dedicated Red Hat Gluster Storage solution.

This topic presumes some familiarity with OpenShift Container Platform and GlusterFS; see the Red Hat Gluster Storage 3 Administration Guide for more on GlusterFS. See the Persistent Storage topic for details on the OpenShift Container Platform PV framework in general.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.3.2. Support Requirements

The following requirements must be met to create a supported integration of Red Hat Gluster Storage and OpenShift Container Platform.

21.3.2.1. Supported Operating Systems

The following table lists the supported versions of OpenShift Container Platform with Red Hat Gluster Storage Server.

Red Hat Gluster Storage	OpenShift Container Platform
3.1.3	3.1 or later

21.3.2.2. Environment Requirements

The environment requirements for OpenShift Container Platform and Red Hat Gluster Storage are described in this section.

Red Hat Gluster Storage

- All installations of Red Hat Gluster Storage must have valid subscriptions to Red Hat Network channels and Subscription Management repositories.
- Red Hat Gluster Storage installations must adhere to the requirements laid out in the Red Hat Gluster Storage Installation Guide.
- Red Hat Gluster Storage installations must be completely up to date with the latest patches and upgrades. Refer to the Red Hat Gluster Storage 3.1 Installation Guide to upgrade to the latest version.
- The versions of OpenShift Container Platform and Red Hat Gluster Storage integrated must be compatible, according to the information in Supported Operating Systems.
- A fully-qualified domain name (FQDN) must be set for each hypervisor and Red Hat Gluster Storage server node. Ensure that correct DNS records exist, and that the FQDN is resolvable via both forward and reverse DNS lookup.

Red Hat OpenShift Enterprise

- All installations of OpenShift Container Platform must have valid subscriptions to Red Hat Network channels and Subscription Management repositories.
- OpenShift Container Platform installations must adhere to the requirements laid out in the Installation and Configuration documentation.
- The OpenShift Container Platform cluster must be up and running.
- A user with **cluster-admin** permissions must be created.
- All OpenShift Container Platform nodes on RHEL systems must have the **glusterfs-fuse** RPM installed, which should match the version of Red Hat Gluster Storage server running in the containers. For more information on installing **glusterfs-fuse**, see Native Client in the Red Hat Gluster Storage Administration Guide.

21.3.3. Provisioning

To provision GlusterFS volumes the following are required:

- An existing storage device in your underlying infrastructure.
- A distinct list of servers (IP addresses) in the Gluster cluster, to be defined as endpoints.
- A service, to persist the endpoints (optional).
- An existing Gluster volume to be referenced in the persistent volume object.
- **glusterfs-fuse** installed on each schedulable OpenShift Container Platform node in your cluster:

```
# yum install glusterfs-fuse
```

Note

Persistent volumes (PVs) and persistent volume claims (PVCs) can share volumes across a single project. While the GlusterFS-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

21.3.3.1. Creating Gluster Endpoints

An endpoints definition defines the GlusterFS cluster as **`EndPoints`** and includes the IP addresses of your Gluster servers. The port value can be any numeric value within the accepted range of ports. Optionally, you can create a service that persists the endpoints.

1. Define the following service:

Example 21.3. Gluster Service Definition

```
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster 1
spec:
  ports:
  - port: 1
```

1

This name must be defined in the endpoints definition. If using a service, then the endpoints name must match the service name.

2. Save the service definition to a file, for example ***gluster-service.yaml***, then create the service:

```
$ oc create -f gluster-service.yaml
```

3. Verify that the service was created:

```
# oc get services
NAME                      CLUSTER_IP       EXTERNAL_IP
PORT(S)    SELECTOR        AGE
glusterfs-cluster         172.30.205.34    <none>        1/TCP
<none>          44s
```

4. Define the Gluster endpoints:

Example 21.4. Gluster Endpoints Definition

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster 1
subsets:
  - addresses:
      - ip: 192.168.122.221 2
    ports:
      - port: 1
  - addresses:
      - ip: 192.168.122.222 3
    ports:
      - port: 1 4
```

1 This name must match the service name from step 1.

2 3 The **ip** values must be the actual IP addresses of a Gluster server, not fully-qualified host names.

4 The port number is ignored.

5. Save the endpoints definition to a file, for example ***gluster-endpoints.yaml***, then create the endpoints:

```
$ oc create -f gluster-endpoints.yaml
endpoints "glusterfs-cluster" created
```

6. Verify that the endpoints were created:

```
$ oc get endpoints
NAME                ENDPOINTS                                  AGE
docker-registry     10.1.0.3:5000                              4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1        11s
kubernetes          172.16.35.3:8443                           4d
```

21.3.3.2. Creating the Persistent Volume

Note

GlusterFS does not support the 'Recycle' recycling policy.

1. Next, define the PV in an object definition before creating it in OpenShift Container Platform:

Example 21.5. Persistent Volume Object Definition Using GlusterFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes: 3
    - ReadWriteMany
  glusterfs: 4
    endpoints: glusterfs-cluster 5
    path: myVol1 6
    readOnly: false
  persistentVolumeReclaimPolicy: Retain 7
```

1

The name of the volume. This is how it is identified via persistent volume claims or from pods.

2

The amount of storage allocated to this volume.

3

accessModes are used as labels to match a PV and a PVC. They currently do not define any form of access control.

4

The volume type being used, in this case the **glusterfs** plug-in.

5

The endpoints name that defines the Gluster cluster created in Creating Gluster Endpoints.

6

The Gluster volume that will be accessed, as shown in the **`gluster volume status`** command.

7

The Recycle policy is currently not supported with glusterfs

Note

Endpoints are name-spaced. Each project accessing the Gluster volume needs its own endpoints.

1. Save the definition to a file, for example ***gluster-pv.yaml***, and create the persistent volume:

```
# oc create -f gluster-pv.yaml
```

2. Verify that the persistent volume was created:

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
gluster-default-volume   <none>    2147483648   RWX
Available                          2s
```

21.3.3.3. Creating the Persistent Volume Claim

Developers request GlusterFS storage by referencing either a PVC or the Gluster volume plug-in directly in the **`volumes`** section of a pod spec. A PVC exists only in the user's project and can only be referenced by pods within that project. Any attempt to access a PV across a project causes the pod to fail.

1. Create a PVC that will bind to the new PV:

Example 21.6. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
```

```
accessModes:
- ReadWriteMany 1
resources:
   requests:
     storage: 1Gi 2
```

1 **accessModes** do not enforce security, but rather act as labels to match a PV to a PVC.

2 This claim will look for PVs offering **1Gi** or greater capacity.

2. Save the definition to a file, for example ***gluster-claim.yaml***, and create the PVC:

```
# oc create -f gluster-claim.yaml
```

Note

PVs and PVCs make sharing a volume across a project simpler. The gluster-specific information contained in the PV definition can also be defined directly in a pod specification.

21.3.4. Gluster Volume Security

This section covers Gluster volume security, including matching permissions and SELinux considerations. Understanding the basics of POSIX permissions, process UIDs, supplemental groups, and SELinux is presumed.

Note

See the full Volume Security topic before implementing Gluster volumes.

As an example, assume that the target Gluster volume, **HadoopVol** is mounted under ***/mnt/glusterfs/***, with the following POSIX permissions and SELinux labels:

```
# ls -lZ /mnt/glusterfs/
drwxrwx---. yarn hadoop system_u:object_r:fusefs_t:s0    HadoopVol

# id yarn
uid=592(yarn) gid=590(hadoop) groups=590(hadoop)
```

In order to access the **HadoopVol** volume, containers must match the SELinux label, and run with a

UID of 592 or 590 in their supplemental groups. The OpenShift Container Platform GlusterFS plug-in mounts the volume in the container with the same POSIX ownership and permissions found on the target gluster mount, namely the owner will be **592** and group ID will be **590**. However, the container is not run with its effective UID equal to **592**, nor with its GID equal to **590**, which is the desired behavior. Instead, a container's UID and supplemental groups are determined by Security Context Constraints (SCCs) and the project defaults.

21.3.4.1. Group IDs

Configure Gluster volume access by using supplemental groups, assuming it is not an option to change permissions on the Gluster mount. Supplemental groups in OpenShift Container Platform are used for shared storage, such as GlusterFS. In contrast, block storage, such as Ceph RBD or iSCSI, use the **fsGroup** SCC strategy and the **fsGroup** value in the pod's **`securityContext`**.

Note

Use supplemental group IDs instead of user IDs to gain access to persistent storage. Supplemental groups are covered further in the full Volume Security topic.

The group ID on the target Gluster mount example above is 590. Therefore, a pod can define that group ID using **`supplementalGroups`** under the pod-level **`securityContext`** definition. For example:

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [590] 2
```

1

`securityContext` must be defined at the pod level, not under a specific container.

2

An array of GIDs defined at the pod level.

Assuming there are no custom SCCs that satisfy the pod's requirements, the pod matches the **restricted** SCC. This SCC has the **`supplementalGroups`** strategy set to **RunAsAny**, meaning that any supplied group IDs are accepted without range checking.

As a result, the above pod will pass admissions and can be launched. However, if group ID range checking is desired, use a custom SCC, as described in pod security and custom SCCs. A custom SCC can be created to define minimum and maximum group IDs, enforce group ID range checking, and allow a group ID of **590**.

21.3.4.2. User IDs

User IDs can be defined in the container image or in the pod definition. The full Volume Security topic covers controlling storage access based on user IDs, and should be read prior to setting up NFS persistent storage.

Note

Use supplemental group IDs instead of user IDs to gain access to persistent storage.

In the target Gluster mount example above, the container needs a UID set to **592**, so the following can be added to the pod definition:

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 592 2
```

1

Pods contain a **`securtityContext`** specific to each container and a pod-level **`securityContext`**, which applies to all containers defined in the pod.

2

The UID defined on the Gluster mount.

With the **default** project and the **restricted** SCC, a pod's requested user ID of **592** will not be allowed, and the pod will fail. This is because:

- The pod requests **592** as its user ID.
- All SCCs available to the pod are examined to see which SCC will allow a user ID of **592**.
- Because all available SCCs use **MustRunAsRange** for their **`runAsUser`** strategy, UID range checking is required.
- **592** is not included in the SCC or project's user ID range.

Do not modify the predefined SCCs. Instead, create a custom SCC so that minimum and maximum user IDs are defined, UID range checking is still enforced, and the UID of **592** will be allowed.

21.3.4.3. SELinux

Note

See the full Volume Security topic for information on controlling storage access in conjunction with using SELinux.

By default, SELinux does not allow writing from a pod to a remote Gluster server.

To enable writing to GlusterFS volumes with SELinux enforcing on each node, run:

```
$ sudo setsebool -P virt_sandbox_use_fusefs on
```

Note

The **`virt_sandbox_use_fusefs`** boolean is defined by the **docker-selinux** package. If you get an error saying it is not defined, please ensure that this package is installed.

The **`-P`** option makes the bool persistent between reboots.

21.4. PERSISTENT STORAGE USING OPENSTACK CINDER

21.4.1. Overview

You can provision your OpenShift Container Platform cluster with persistent storage using OpenStack Cinder. Some familiarity with Kubernetes and OpenStack is assumed.

Important

Before creating persistent volumes using Cinder, OpenShift Container Platform must first be properly configured for OpenStack.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. OpenStack Cinder volumes can be provisioned dynamically. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims, however, are specific to a project or namespace and can be requested by users.

For a detailed example, see the guide for WordPress and MySQL using persistent volumes.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.4.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for OpenStack, all that is required for Cinder is a Cinder volume ID and the**`PersistentVolume`** API.

21.4.2.1. Creating the Persistent Volume

Note

Cinder does not support the 'Recycle' recycling policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 21.7. Persistent Volume Object Definition Using Cinder

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  cinder: 3
    fsType: "ext3" 4
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" 5
```

1 The name of the volume. This will be how it is identified via persistent volume claims or from pods.

2 The amount of storage allocated to this volume.

3 This defines the volume type being used, in this case the **cinder** plug-in.

4 File system type to mount.

5 This is the Cinder volume that will be used.

Important

Changing the value of the **`fstype`** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

Save your definition to a file, for example ***cinder-pv.yaml***, and create the persistent volume:

```
# oc create -f cinder-pv.yaml
persistentvolume "pv0001" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

Users can then request storage using persistent volume claims, which can now utilize your new persistent volume.

Important

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

21.4.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **`fsType`** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted Cinder volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

21.5. PERSISTENT STORAGE USING CEPH RADOS BLOCK DEVICE (RBD)

21.5.1. Overview

OpenShift Container Platform clusters can be provisioned with persistent storage using Ceph RBD.

Persistent volumes (PVs) and persistent volume claims (PVCs) can share volumes across a single project. While the Ceph RBD-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

This topic presumes some familiarity with OpenShift Container Platform and Ceph RBD. See the Persistent Storage concept topic for details on the OpenShift Container Platform persistent volume (PV) framework in general.

Note

Project and *namespace* are used interchangeably throughout this document. See Projects and Users for details on the relationship.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.5.2. Provisioning

To provision Ceph volumes, the following are required:

- An existing storage device in your underlying infrastructure.
- The Ceph key to be used in an OpenShift Container Platform secret object.
- The Ceph image name.
- The file system type on top of the block storage (e.g., ext4).
- **ceph-common** installed on each schedulable OpenShift Container Platform node in your cluster:

```
# yum install ceph-common
```

21.5.2.1. Creating the Ceph Secret

Define the authorization key in a secret configuration, which is then converted to base64 for use by OpenShift Container Platform.

Note

In order to use Ceph storage to back a persistent volume, the secret must be created in the same project as the PVC and pod. The secret cannot simply be in the default project.

1. Run **`ceph auth get-key`** on a Ceph MON node to display the key value for the **`client.admin`** user:

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ==
```

2. Save the secret definition to a file, for example ***ceph-secret.yaml***, then create the secret:

```
$ oc create -f ceph-secret.yaml
```

3. Verify that the secret was created:

```
# oc get secret ceph-secret
NAME          TYPE      DATA      AGE
ceph-secret   Opaque    1         23d
```

21.5.2.2. Creating the Persistent Volume

Note

Ceph RBD does not support the 'Recycle' recycling policy.

Developers request Ceph RBD storage by referencing either a PVC, or the Gluster volume plug-in directly in the **`volumes`** section of a pod specification. A PVC exists only in the user's namespace and can be referenced only by pods within that same namespace. Any attempt to access a PV from a different namespace causes the pod to fail.

1. Define the PV in an object definition before creating it in OpenShift Container Platform:

Example 21.8. Persistent Volume Object Definition Using Ceph RBD

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes:
    - ReadWriteOnce 3
  rbd: 4
    monitors: 5
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret 6
    fsType: ext4 7
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1

The name of the PV that is referenced in pod definitions or displayed in various **oc** volume commands.

2

The amount of storage allocated to this volume.

3

`accessModes` are used as labels to match a PV and a PVC. They currently do not define any form of access control. All block storage is defined to be single user (non-shared storage).

4

The volume type being used, in this case the **rbd** plug-in.

5

An array of Ceph monitor IP addresses and ports.

6

The Ceph secret used to create a secure connection from OpenShift Container Platform to the Ceph server.

7

The file system type mounted on the Ceph RBD block device.

Important

Changing the value of the **`fstype`** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

2. Save your definition to a file, for example ***ceph-pv.yaml***, and create the PV:

```
# oc create -f ceph-pv.yaml
```

3. Verify that the persistent volume was created:

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
ceph-pv                  <none>    2147483648   RWO
Available                          2s
```

4. Create a PVC that will bind to the new PV:

Example 21.9. PVC Object Definition

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

1

The **accessModes** do not enforce access right, but instead act as labels to match a PV to a PVC.

2

This claim looks for PVs offering **2Gi** or greater capacity.

5. Save the definition to a file, for example ***ceph-claim.yaml***, and create the PVC:

```
# oc create -f ceph-claim.yaml
```

21.5.3. Ceph Volume Security

Note

See the full Volume Security topic before implementing Ceph RBD volumes.

A significant difference between shared volumes (NFS and GlusterFS) and block volumes (Ceph RBD, iSCSI, and most cloud storage), is that the user and group IDs defined in the pod definition or container image are applied to the target physical storage. This is referred to as managing ownership of the block device. For example, if the Ceph RBD mount has its owner set to **123** and its

group ID set to **567**, and if the pod defines its **runAsUser** set to **222** and its **fsGroup** to be **7777**, then the Ceph RBD physical mount's ownership will be changed to **222:7777**.

Note

Even if the user and group IDs are not defined in the pod specification, the resulting pod may have defaults defined for these IDs based on its matching SCC, or its project. See the full Volume Security topic which covers storage aspects of SCCs and defaults in greater detail.

A pod defines the group ownership of a Ceph RBD volume using the **fsGroup** stanza under the pod's **securityContext** definition:

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777 2
```

1

The **securityContext** must be defined at the pod level, not under a specific container.

2

All containers in the pod will have the same fsGroup ID.

21.6. PERSISTENT STORAGE USING AWS ELASTIC BLOCK STORE

21.6.1. Overview

OpenShift Container Platform supports AWS Elastic Block Store volumes (EBS). You can provision your OpenShift Container Platform cluster with persistent storage using AWS EC2. Some familiarity with Kubernetes and AWS is assumed.

Important

Before creating persistent volumes using AWS, OpenShift Container Platform must first be properly configured for AWS ElasticBlockStore.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. AWS Elastic Block Store volumes can be provisioned dynamically. Persistent volumes are not bound to a single project or namespace; they can be shared across the

OpenShift Container Platform cluster. Persistent volume claims, however, are specific to a project or namespace and can be requested by users.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.6.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift is configured for AWS Elastic Block Store, all that is required for OpenShift and AWS is an AWS EBS volume ID and the **`PersistentVolume`** API.

21.6.2.1. Creating the Persistent Volume

Note

AWS does not support the 'Recycle' recycling policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 21.10. Persistent Volume Object Definition Using AWS

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  awsElasticBlockStore: 3
    fsType: "ext4" 4
    volumeID: "vol-f37a03aa" 5
```

1

The name of the volume. This will be how it is identified via persistent volume claims or from pods.

2

The amount of storage allocated to this volume.

3

This defines the volume type being used, in this case the **awsElasticBlockStore** plug-in.

4

File system type to mount.

5

This is the AWS volume that will be used.

Important

Changing the value of the **`fstype`** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

Save your definition to a file, for example ***aws-pv.yaml***, and create the persistent volume:

```
# oc create -f aws-pv.yaml
persistentvolume "pv0001" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

Users can then request storage using persistent volume claims, which can now utilize your new persistent volume.

Important

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

21.6.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **`fsType`** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted AWS volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

21.7. PERSISTENT STORAGE USING GCE PERSISTENT DISK

21.7.1. Overview

OpenShift Container Platform supports GCE Persistent Disk volumes (gcePD). You can provision your OpenShift Container Platform cluster with persistent storage using GCE. Some familiarity with Kubernetes and GCE is assumed.

Important

Before creating persistent volumes using GCE, OpenShift Container Platform must first be properly configured for GCE Persistent Disk.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. GCE Persistent Disk volumes can be provisioned dynamically. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims, however, are specific to a project or namespace and can be requested by users.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.7.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for GCE PersistentDisk, all that is required for OpenShift Container Platform and GCE is an GCE Persistent Disk volume ID and the **`PersistentVolume`** API.

21.7.2.1. Creating the Persistent Volume

Note

GCE does not support the 'Recycle' recycling policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 21.11. Persistent Volume Object Definition Using GCE

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  gcePersistentDisk: 3
    fsType: "ext4" 4
    pdName: "pd-disk-1" 5
```

1

The name of the volume. This will be how it is identified via persistent volume claims or from pods.

2

The amount of storage allocated to this volume.

3

This defines the volume type being used, in this case the **gcePersistentDisk** plug-in.

4

File system type to mount.

5

This is the GCE Persistent Disk volume that will be used.

Important

Changing the value of the **`fstype`** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

Save your definition to a file, for example ***gce-pv.yaml***, and create the persistent volume:

```
# oc create -f gce-pv.yaml
```

```
persistentvolume "pv0001" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

Users can then request storage using persistent volume claims, which can now utilize your new persistent volume.

Important

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

21.7.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **`fsType`** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted GCE volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

21.7.2.3. Multi-zone Configuration

In multi-zone configurations, You must specify **`failure-domain.beta.kubernetes.io/region`** and **`failure-domain.beta.kubernetes.io/zone`** PV labels to match the zone where GCE volume exists.

Example 21.12. Persistent Volume Object With Failure Domain

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001"
  labels:
    failure-domain.beta.kubernetes.io/region: "us-central1" 1
    failure-domain.beta.kubernetes.io/zone: "us-central1-a" 2
spec:
  capacity:
    storage: "5Gi"
  accessModes:
```

```
    - "ReadWriteOnce"
  gcePersistentDisk:
    fsType: "ext4"
    pdName: "pd-disk-1"
```

1

The region in which the volume exists.

2

The zone in which the volume exists.

21.8. PERSISTENT STORAGE USING ISCSI

21.8.1. Overview

You can provision your OpenShift Container Platform cluster with persistent storage using iSCSI. Some familiarity with Kubernetes and iSCSI is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.8.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. All that is required for iSCSI is iSCSI target portal, valid iSCSI IQN, valid LUN number, and filesystem type, and the **`PersistentVolume`** API.

Note

iSCSI does not support the 'Recycle' recycling policy.

Example 21.13. Persistent Volume Object Definition

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
```

```
capacity:
  storage: 1Gi
accessModes:
  - ReadWriteOnce
iscsi:
   targetPortal: 10.16.154.81
   iqn: iqn.2014-12.example.server:storage.target00
   lun: 0
   fsType: 'ext4'
   readOnly: false
```

21.8.2.1. Enforcing Disk Quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is one persistent volume. Kubernetes enforces unique names for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (e.g, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

21.8.2.2. iSCSI Volume Security

Users request storage with a **`PersistentVolumeClaim`**. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each iSCSI LUN must be accessible by all nodes in the cluster.

21.9. PERSISTENT STORAGE USING FIBRE CHANNEL

21.9.1. Overview

You can provision your OpenShift Container Platform cluster with persistent storage using Fibre Channel. Some familiarity with Kubernetes and Fibre Channel is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Important

High-availability of storage in the infrastructure is left to the underlying storage provider.

21.9.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. All that is required for Fibre Channel persistent storage is the targetWWNs (array of Fibre Channel target's World Wide Names), a valid LUN number, and filesystem type, and the **`PersistentVolume`** API. Note, the number of LUNs must correspond to the number of Persistent Volumes that are created. In the example below, we have LUN as 2, therefore we have created two Persistent Volume definitions.

Note

Fiber Channel does not support the 'Recycle' recycling policy.

Example 21.14. Persistent Volumes Object Definition

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5']
    lun: 2
    fsType: ext4
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0002
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadOnlyMany
  fc:
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5']
    lun: 2
    fsType: ext4
```

Important

Changing the value of the **`fstype`** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

21.9.2.1. Enforcing Disk Quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is one persistent volume. Kubernetes enforces unique names for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (e.g, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

21.9.2.2. Fibre Channel Volume Security

Users request storage with a **`PersistentVolumeClaim`**. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each Fibre Channel LUN must be accessible by all nodes in the cluster.

21.10. PERSISTENT STORAGE USING AZURE DISK

21.10.1. Overview

OpenShift Container Platform supports Azure Disk volumes. You can provision your OpenShift Container Platform cluster with persistent storage using Azure. Some familiarity with Kubernetes and Azure is assumed.

Important

Before creating persistent volumes using Azure, OpenShift Container Platform must first be properly configured for Azure Disk.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims, however, are specific to a project or namespace and can be requested by users.

Important

High availability of storage in the infrastructure is left to the underlying storage provider.

21.10.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for Azure Disk, all that is required for OpenShift Container Platform and Azure is an Azure Disk Name and Disk URI and the **`PersistentVolume`** API.

21.10.2.1. Creating the Persistent Volume

Note

Azure does not support the **Recycle** recycling policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 21.15. Persistent Volume Object Definition Using Azure

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteOnce"
  azureDisk: (3)
    diskName: test2.vhd (4)
    diskURI: https://someacount.blob.core.windows.net/vhds/test2.vhd
(5)
    cachingMode: readwrite (6)
    fsType: ext4 (7)
    readOnly: false (8)
```

(1)

The name of the volume. This will be how it is identified via persistent volume claims or from pods.

(2)

The amount of storage allocated to this volume.

(3)

This defines the volume type being used (**azureDisk** plug-in, in this example).

(4)

The name of the data disk in the blob storage.

(5)

The URI the the data disk in the blob storage.

(6)

Host caching mode: None, Read Only, or Read Write.

(7)

File system type to mount (for example, **`ext4`**, **`xfs`**, and so on).

8

Defaults to **`false`** (read/write). **`ReadOnly`** here will force the **`ReadOnly`** setting in **`VolumeMounts`**.

Important

Changing the value of the **`fsType`** parameter after the volume is formatted and provisioned can result in data loss and pod failure.

1. Save your definition to a file, for example ***azure-pv.yaml***, and create the persistent volume:

```
# oc create -f azure-pv.yaml
persistentvolume "pv0001" created
```

2. Verify that the persistent volume was created:

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

Now you can request storage using persistent volume claims, which can now use your new persistent volume.

Important

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

21.10.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **`fsType`** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows unformatted Azure volumes to be used as persistent volumes because OpenShift Container Platform formats them before the first use.

21.11. DYNAMIC PROVISIONING AND CREATING STORAGE CLASSES

21.11.1. Overview

The *StorageClass* resource object describes and classifies storage that can be requested, as well

as provides a means for passing parameters for **dynamically provisioned storage** on demand. *StorageClass* objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. Cluster Administrators (**`cluster-admin`**) or Storage Administrators (**`storage-admin`**) define and create the *StorageClass* objects that users can request without needing any intimate knowledge about the underlying storage volume sources.

The OpenShift Container Platform persistent volume framework enables this functionality and allows administrators to provision a cluster with persistent storage. The framework also gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Many storage types are available for use as persistent volumes in OpenShift Container Platform. While all of them can be statically provisioned by an administrator, some types of storage are created dynamically using the built-in provider and plug-in APIs.

21.11.2. Available Dynamically Provisioned Plug-ins

OpenShift Container Platform provides the following *provisioner plug-ins*, which have generic implementations for dynamic provisioning that use the cluster's configured provider's API to create new storage resources:

Storage Type	Provisioner Plug-in Name	Required Configuration	Notes
OpenStack Cinder	**`kubernetes.io/cinder`**	Configuring for OpenStack	
AWS Elastic Block Store (EBS)	**`kubernetes.io/aws-ebs`**	Configuring for AWS	For dynamic provisioning when using multiple clusters in different zones, tag each node with **`Key=KubernetesCluster,Value=clusterid`**.
GCE Persistent Disk (gcePD)	**`kubernetes.io/gce-pd`**	Configuring for GCE	In multi-zone configurations, it is advisable to run one Openshift cluster per GCE project to avoid PVs from getting created in zones where no node from current cluster exists.

Storage Type	Provisioner Plug-in Name	Required Configuration	Notes
GlusterFS	**kubernetes.io/glusterfs**	Container Native Storage with GlusterFS	Container Native Storage (CNS) utilizes Heketi to manage Gluster Storage.
Ceph RBD	**kubernetes.io/rbd**	Configuring for OpenStack	

Important

Any chosen provisioner plug-in also requires configuration for the relevant cloud, host, or third-party provider as per the relevant documentation.

21.11.3. Defining a StorageClass

StorageClass objects are currently a globally scoped object and need to be created by **cluster-admin** or **storage-admin** users. There are currently five plug-ins that are supported. The following sections describe the basic object definition for a *StorageClass* and specific examples for each of the supported plug-in types.

21.11.3.1. Basic StorageClass Object Definition

Example 21.16. StorageClass Basic Object Definition

```
kind: StorageClass 1
apiVersion: storage.k8s.io/v1beta1 2
metadata:
  name: foo 3
  annotations: 4
     ...
provisioner: kubernetes.io/plug-in-type 5
parameters: 6
  param1: value
  ...
  paramN: value
```

1

(required) The API object type.

2

(required) The current apiVersion.

3

(required) The name of the StorageClass.

4

(optional) Annotations for the StorageClass

5

(required) The type of provisioner associated with this storage class.

6

(optional) The parameters required for the specific provisioner, this will change from plug-in to plug-in.

21.11.3.2. StorageClass Annotations

To set a *StorageClass* as the cluster-wide default:

```
storageclass.beta.kubernetes.io/is-default-class: "true"
```

This enables any Persistent Volume Claim (PVC) that does not specify a specific volume to automatically be provisioned through the *default* StorageClass

To set a *StorageClass* description:

```
kubernetes.io/description: My StorgeClass Description
```

21.11.3.3. OpenStack Cinder Object Defintion

Example 21.17. cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: gold
```

```
provisioner: kubernetes.io/cinder
parameters:
  type: fast  (1)
  availability: nova  (2)
```

(1) VolumeType created in Cinder. Default is empty.

(2) Availability Zone. Default is empty.

21.11.3.4. AWS ElasticBlockStore (EBS) Object Defintion

Example 21.18. aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1  (1)
  zone: us-east-1d  (2)
  iopsPerGB: "10"  (3)
  encrypted: true  (4)
  kmsKeyId: keyvalue  (5)
```

(1) Select from **io1**, **gp2**, **sc1**, **st1**. The default is **gp2**. See AWS docs for valid ARN value.

(2) AWS zone. If not specified, the zone is randomly selected from zones where OpenShift Container Platform cluster has a node.

(3)

Only for io1 volumes. I/O operations per second per GiB. The AWS volume plug-in multiplies this with the size of the requested volume to compute IOPS of the volume. The value cap is 20,000 IOPS, which is the maximum supported by AWS. See AWS documentation for further details.

4

Denotes whether to encrypt the EBS volume. Valid values are **`true`** or **`false`**.

5

(optional) The full Amazon Resource Name (ARN) of the key to use when encrypting the volume. If none is supplied but encrypted is true, AWS generates a key. See AWS docs for valid ARN value.

21.11.3.5. GCE PersistentDisk (gcePD) Object Defintion

Example 21.19. gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard  1
  zone: us-central1-a  2
```

1

Select either **`pd-standard`** or **`pd-ssd`**. The default is **`pd-ssd`**.

2

GCE zone. If not specified, the zone is randomly chosen from zones in the same region as **`controller-manager`**.

21.11.3.6. GlusterFS Object Defintion

Example 21.20. glusterfs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
```

```
metadata:
  name: slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://127.0.0.1:8081" 1
  restuser: "admin" 2
  secretName: "heketi-secret" 3
  secretNamespace: "default" 4
  gidMin: "40000" 5
  gidMax: "50000" 6
```

1

Gluster REST service/Heketi service URL that provisions Gluster volumes on demand. The general format should be **`{http/https}://{IPaddress}:{Port}`**. This is a mandatory parameter for the GlusterFS dynamic provisioner. If the Heketi service is exposed as a routable service in the OpenShift Container Platform, it will have a resolvable fully qualified domain name and Heketi service URL. For additional information and configuration, See Container-Native Storage for OpenShift Container Platform.

2

Gluster REST service/Heketi user who has access to create volumes in the Gluster Trusted Pool.

3

Identification of a Secret instance that contains a user password to use when talking to the Gluster REST service. Optional; an empty password will be used when both **`secretNamespace`** and **`secretName`** are omitted. The provided secret must be of type **`"kubernetes.io/glusterfs"`**.

4

The namespace of mentioned **`secretName`**. Optional; an empty password will be used when both **`secretNamespace`** and **`secretName`** are omitted. The provided secret must be of type **`"kubernetes.io/glusterfs"`**.

5

Optional. The minimum value of GID range for the storage class.

6

Optional. The maximum value of GID range for the storage class.

When the **gidMin** and **gidMax** values are not specified, the volume is provisioned with a value between 2000 and 2147483647, which are defaults for **gidMin** and **gidMax** respectively. If specified, a unique value (GID) in this range (**gidMin-gidMax**) is used for dynamically provisioned volumes. The GID of the provisioned volume will be set to this value. It is required to run Heketi version 3 or later to make use of this feature. This GID is released from the pool when the subjected volume is deleted. The GID pool is per storage class, if 2 or more storage classes have GID ranges that overlap there will be duplicate GIDs dispatched by the provisioner.

When the persistent volumes are dynamically provisioned, the Gluster plug-in automatically creates an endpoint and a headless service of the name **gluster-dynamic-<claimname>**. When the persistent volume claim is deleted, this dynamic endpoint and service is deleted automatically.

Example of a Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  # base64 encoded password. E.g.: echo -n "mypassword" | base64
  key: bXlwYXNzd29yZA==
type: kubernetes.io/glusterfs
```

21.11.3.7. Ceph RBD Object Definition

Example 21.21. ceph-storageclass.yaml

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.16.153.105:6789  1
  adminId: kube  2
  adminSecretName: ceph-secret  3
  adminSecretNamespace: kube-system  4
  pool: kube  5
  userId: kube  6
  userSecretName: ceph-secret-user  7
```

1

Ceph monitors, comma delimited. It is required.

2

Ceph client ID that is capable of creating images in the pool. Default is "admin".

3

Secret Name for **`adminId`**. It is required. The provided secret must have type "kubernetes.io/rbd".

4

The namespace for **`adminSecret`**. Default is "default".

5

Ceph RBD pool. Default is "rbd".

6

Ceph client ID that is used to map the RBD image. Default is the same as **`adminId`**.

7

The name of Ceph Secret for **`userId`** to map RBD image. It must exist in the same namespace as PVCs. It is required.

21.11.4. Additional Information and Examples

- Examples and uses of StorageClasses for Dynamic Provisioning
- Examples and uses of StorageClasses without Dynamic Provisioning

21.12. VOLUME SECURITY

21.12.1. Overview

This topic provides a general guide on pod security as it relates to volume security. For information on pod-level security in general, see Managing Security Context Constraints (SCC) and the Security Context Constraint concept topic. For information on the OpenShift Container Platform persistent volume (PV) framework in general, see the Persistent Storage concept topic.

Accessing persistent storage requires coordination between the cluster and/or storage administrator and the end developer. The cluster administrator creates PVs, which abstract the underlying physical storage. The developer creates pods and, optionally, PVCs, which bind to PVs, based on matching criteria, such as capacity.

Multiple persistent volume claims (PVCs) within the same project can bind to the same PV. However, once a PVC binds to a PV, that PV cannot be bound by a claim outside of the first claim's project. If the underlying storage needs to be accessed by multiple projects, then each project needs its own PV, which can point to the same physical storage. In this sense, a bound PV is tied to a

project. For a detailed PV and PVC example, see the guide for WordPress and MySQL using NFS.

For the cluster administrator, granting pods access to PVs involves:

- knowing the group ID and/or user ID assigned to the actual storage,
- understanding SELinux considerations, and
- ensuring that these IDs are allowed in the range of legal IDs defined for the project and/or the SCC that matches the requirements of the pod.

Group IDs, the user ID, and SELinux values are defined in the **`SecurityContext`** section in a pod definition. Group IDs are global to the pod and apply to all containers defined in the pod. User IDs can also be global, or specific to each container. Four sections control access to volumes:

- **`supplementalGroups`**
- **`fsGroup`**
- **`runAsUser`**
- **`seLinuxOptions`**

21.12.2. SCCs, Defaults, and Allowed Ranges

SCCs influence whether or not a pod is given a default user ID, **`fsGroup`** ID, supplemental group ID, and SELinux label. They also influence whether or not IDs supplied in the pod definition (or in the image) will be validated against a range of allowable IDs. If validation is required and fails, then the pod will also fail.

SCCs define strategies, such as **`runAsUser`**, **`supplementalGroups`**, and **`fsGroup`**. These strategies help decide whether the pod is authorized. Strategy values set to **RunAsAny** are essentially stating that the pod can do what it wants regarding that strategy. Authorization is skipped for that strategy and no OpenShift Container Platform default is produced based on that strategy. Therefore, IDs and SELinux labels in the resulting container are based on container defaults instead of OpenShift Container Platform policies.

For a quick summary of **RunAsAny**:

- Any ID defined in the pod definition (or image) is allowed.
- Absence of an ID in the pod definition (and in the image) results in the container assigning an ID, which is **root** (0) for Docker.
- No SELinux labels are defined, so Docker will assign a unique label.

For these reasons, SCCs with **RunAsAny** for ID-related strategies should be protected so that ordinary developers do not have access to the SCC. On the other hand, SCC strategies set to **MustRunAs** or **MustRunAsRange** trigger ID validation (for ID-related strategies), and cause default values to be supplied by OpenShift Container Platform to the container when those values are not supplied directly in the pod definition or image.

Caution

Allowing access to SCCs with a **RunAsAny** **`FSGroup`** strategy can also prevent users from accessing their block devices. Pods need to specify an **`fsGroup`** in order to take over their block devices. Normally, this is done when the SCC **`FSGroup`** strategy is set to **MustRunAs**. If a user's pod is assigned an SCC with a **RunAsAny** **`FSGroup`** strategy, then the user may face **permission denied** errors until they discover that they need to specify an **`fsGroup`** themselves.

SCCs may define the range of allowed IDs (user or groups). If range checking is required (for example, using **MustRunAs**) and the allowable range is not defined in the SCC, then the project determines the ID range. Therefore, projects support ranges of allowable ID. However, unlike SCCs, projects do not define strategies, such as **`runAsUser`**.

Allowable ranges are helpful not only because they define the boundaries for container IDs, but also because the minimum value in the range becomes the default value for the ID in question. For example, if the SCC ID strategy value is **MustRunAs**, the minimum value of an ID range is **100**, and the ID is absent from the pod definition, then 100 is provided as the default for this ID.

As part of pod admission, the SCCs available to a pod are examined (roughly, in priority order followed by most restrictive) to best match the requests of the pod. Setting a SCC's strategy type to **RunAsAny** is less restrictive, whereas a type of **MustRunAs** is more restrictive. All of these strategies are evaluated. To see which SCC was assigned to a pod, use the **`oc get pod`** command:

```
# oc get pod <pod_name> -o yaml
...
metadata:
  annotations:
    openshift.io/scc: nfs-scc 1
  name: nfs-pod1 2
  namespace: default 3
...
```

1

Name of the SCC that the pod used (in this case, a custom SCC).

2

Name of the pod.

3

Name of the project. "Namespace" is interchangeable with "project" in OpenShift Container Platform. See Projects and Users for details.

It may not be immediately obvious which SCC was matched by a pod, so the command above can be very useful in understanding the UID, supplemental groups, and SELinux relabeling in a live container.

Any SCC with a strategy set to **RunAsAny** allows specific values for that strategy to be defined in the pod definition (and/or image). When this applies to the user ID (**`runAsUser`**) it is prudent to restrict access to the SCC to prevent a container from being able to run as root.

Because pods often match the **restricted** SCC, it is worth knowing the security this entails. The **restricted** SCC has the following characteristics:

- User IDs are constrained due to the **`runAsUser`** strategy being set to **MustRunAsRange**. This forces user ID validation.
- Because a range of allowable user IDs is not defined in the SCC (see **`oc export scc restricted`** for more details), the project's **`openshift.io/sa.scc.uid-range`** range will be used for range checking and for a default ID, if needed.
- A default user ID is produced when a user ID is not specified in the pod definition and the matching SCC's **`runAsUser`** is set to **MustRunAsRange**.
- An SELinux label is required (**`seLinuxContext`** set to **MustRunAs**), which uses the project's default MCS label.
- **`fsGroup`** IDs are constrained to a single value due to the **`FSGroup`** strategy being set to **MustRunAs**, which dictates that the value to use is the minimum value of the first range specified.
- Because a range of allowable **`fsGroup`** IDs is not defined in the SCC, the minimum value of the project's **`openshift.io/sa.scc.supplemental-groups`** range (or the same range used for user IDs) will be used for validation and for a default ID, if needed.
- A default **`fsGroup`** ID is produced when a **`fsGroup`** ID is not specified in the pod and the matching SCC's **`FSGroup`** is set to **MustRunAs**.
- Arbitrary supplemental group IDs are allowed because no range checking is required. This is a result of the **`supplementalGroups`** strategy being set to **RunAsAny**.
- Default supplemental groups are not produced for the running pod due to **RunAsAny** for the two group strategies above. Therefore, if no groups are defined in the pod definition (or in the image), the container(s) will have no supplemental groups predefined.

The following shows the **default** project and a custom SCC (**my-custom-scc**), which summarizes the interactions of the SCC and the project:

```
$ oc get project default -o yaml (1)
...
metadata:
  annotations: (2)
    openshift.io/sa.scc.mcs: s0:c1,c0 (3)
    openshift.io/sa.scc.supplemental-groups: 1000000000/10000 (4)
    openshift.io/sa.scc.uid-range: 1000000000/10000 (5)

$ oc get scc my-custom-scc -o yaml
...
fsGroup:
  type: MustRunAs (6)
  ranges:
  - min: 5000
```

```
    max: 6000
runAsUser:
  type: MustRunAsRange 7
  uidRangeMin: 1000100000
  uidRangeMax: 1000100999
seLinuxContext: 8
  type: MustRunAs
  SELinuxOptions: 9
    user: <selinux-user-name>
    role: ...
    type: ...
    level: ...
supplementalGroups:
  type: MustRunAs 10
  ranges:
  - min: 5000
    max: 6000
```

1

default is the name of the project.

2

Default values are only produced when the corresponding SCC strategy is not **RunAsAny**.

3

SELinux default when not defined in the pod definition or in the SCC.

4

Range of allowable group IDs. ID validation only occurs when the SCC strategy is **RunAsAny**. There can be more than one range specified, separated by commas. See below for supported formats.

5

Same as **<4>** but for user IDs. Also, only a single range of user IDs is supported.

6 10

MustRunAs enforces group ID range checking and provides the container's groups default. Based on this SCC definition, the default is 5000 (the minimum ID value). If the range was omitted from the SCC, then the default would be 1000000000 (derived from the project). The other supported type, **RunAsAny**, does not perform range checking, thus allowing any group ID, and produces no default groups.

7

MustRunAsRange enforces user ID range checking and provides a UID default. Based on this SCC, the default UID is 1000100000 (the minimum value). If the minimum and maximum range were omitted from the SCC, the default user ID would be 1000000000 (derived from the project). **MustRunAsNonRoot** and **RunAsAny** are the other supported types. The range of allowed IDs can be defined to include any user IDs required for the target storage.

8

When set to **MustRunAs**, the container is created with the SCC's SELinux options, or the MCS default defined in the project. A type of **RunAsAny** indicates that SELinux context is not required, and, if not defined in the pod, is not set in the container.

9

The SELinux user name, role name, type, and labels can be defined here.

Two formats are supported for allowed ranges:

1. **`M/N`**, where **`M`** is the starting ID and **`N`** is the count, so the range becomes **`M`** through (and including) **`M+N-1`**.
2. **`M-N`**, where **`M`** is again the starting ID and **`N`** is the ending ID. The default group ID is the starting ID in the first range, which is **`1000000000`** in this project. If the SCC did not define a minimum group ID, then the project's default ID is applied.

21.12.3. Supplemental Groups

Note

Read SCCs, Defaults, and Allowed Ranges before working with supplemental groups.

Tip

It is generally preferable to use group IDs (supplemental or fsGroup) to gain access to persistent storage versus using user IDs.

Supplemental groups are regular Linux groups. When a process runs in Linux, it has a UID, a GID, and one or more supplemental groups. These attributes can be set for a container's main process. The **`supplementalGroups`** IDs are typically used for controlling access to shared storage, such as NFS and GlusterFS, whereas fsGroup is used for controlling access to block storage, such as Ceph RBD and iSCSI.

The OpenShift Container Platform shared storage plug-ins mount volumes such that the POSIX permissions on the mount match the permissions on the target storage. For example, if the target storage's owner ID is **1234** and its group ID is **5678**, then the mount on the host node and in the container will have those same IDs. Therefore, the container's main process must match one or

both of those IDs in order to access the volume.

For example, consider the following NFS export.

On an OpenShift Container Platform node:

Note

`showmount` requires access to the ports used by **`rpcbind`** and **`rpc.mount`** on the NFS server

```
# showmount -e <nfs-server-ip-or-hostname>
Export list for f21-nfs.vm:
/opt/nfs  *
```

On the NFS server:

```
# cat /etc/exports
/opt/nfs *(rw,sync,root_squash)
...

# ls -lZ /opt/nfs -d
drwx------. 1000100001 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

The ***/opt/nfs/*** export is accessible by UID **1000100001** and the group **5555**. In general, containers should not run as root. So, in this NFS example, containers which are not run as UID **1000100001** and are not members the group **5555** will not have access to the NFS export.

Often, the SCC matching the pod does not allow a specific user ID to be specified, thus using supplemental groups is a more flexible way to grant storage access to a pod. For example, to grant NFS access to the export above, the group **5555** can be defined in the pod definition:

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: ...
    volumeMounts:
    - name: nfs 1
      mountPath: /usr/share/... 2
  securityContext: 3
    supplementalGroups: [5555] 4
  volumes:
  - name: nfs 5
    nfs:
      server: <nfs_server_ip_or_host>
      path: /opt/nfs 6
```

1

Name of the volume mount. Must match the name in the **`volumes`** section.

2

NFS export path as seen in the container.

3

Pod global security context. Applies to all containers inside the pod. Each container can also define its **`securityContext`**, however group IDs are global to the pod and cannot be defined for individual containers.

4

Supplemental groups, which is an array of IDs, is set to 5555. This grants group access to the export.

5

Name of the volume. Must match the name in the **`volumeMounts`** section.

6

Actual NFS export path on the NFS server.

All containers in the above pod (assuming the matching SCC or project allows the group **5555**) will be members of the group **5555** and have access to the volume, regardless of the container's user ID. However, the assumption above is critical. Sometimes, the SCC does not define a range of allowable group IDs but instead requires group ID validation (a result of **`supplementalGroups`** set to **MustRunAs**). Note that this is **not** the case for the **restricted** SCC. The project will not likely allow a group ID of **5555**, unless the project has been customized to access this NFS export. So, in this scenario, the above pod will fail because its group ID of **5555** is not within the SCC's or the project's range of allowed group IDs.

Supplemental Groups and Custom SCCs

To remedy the situation in the previous example, a custom SCC can be created such that:

- a minimum and max group ID are defined,
- ID range checking is enforced, and
- the group ID of **5555** is allowed.

It is often better to create a new SCC rather than modifying a predefined SCC, or changing the range of allowed IDs in the predefined projects.

The easiest way to create a new SCC is to export an existing SCC and customize the YAML file to meet the requirements of the new SCC. For example:

1. Use the **restricted** SCC as a template for the new SCC:

```
$ oc export scc restricted > new-scc.yaml
```

2. Edit the ***new-scc.yaml*** file to your desired specifications.
3. Create the new SCC:

```
$ oc create -f new-scc.yaml
```

Note

The **`oc edit scc`** command can be used to modify an instantiated SCC.

Here is a fragment of a new SCC named **nfs-scc**:

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false (1)
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc (2)
priority: 9 (3)
...
supplementalGroups:
  type: MustRunAs (4)
  ranges:
  -  min: 5000 (5)
     max: 6000
...
```

1

The **`allow`** booleans are the same as for the **restricted** SCC.

2

Name of the new SCC.

3

Numerically larger numbers have greater priority. Nil or omitted is the lowest priority. Higher priority SCCs sort before lower priority SCCs and thus have a better chance of matching a new pod.

4

`supplementalGroups` is a strategy and it is set to **MustRunAs**, which means group ID checking is required.

5

Multiple ranges are supported. The allowed group ID range here is 5000 through 5999, with the default supplemental group being 5000.

When the same pod shown earlier runs against this new SCC (assuming, of course, the pod matches the new SCC), it will start because the group **5555**, supplied in the pod definition, is now allowed by the custom SCC.

21.12.4. fsGroup

Note

Read SCCs, Defaults, and Allowed Ranges before working with supplemental groups.

Tip

It is generally preferable to use group IDs (supplemental or **`fsGroup`**) to gain access to persistent storage versus using user IDs.

`fsGroup` defines a pod's "file system group" ID, which is added to the container's supplemental groups. The **`supplementalGroups`** ID applies to shared storage, whereas the **`fsGroup`** ID is used for block storage.

Block storage, such as Ceph RBD, iSCSI, and various cloud storage, is typically dedicated to a single pod which has requested the block storage volume, either directly or using a PVC. Unlike shared storage, block storage is taken over by a pod, meaning that user and group IDs supplied in the pod definition (or image) are applied to the actual, physical block device. Typically, block storage is not shared.

A **`fsGroup`** definition is shown below in the following pod definition fragment:

```
kind: Pod
...
spec:
  containers:
  - name: ...
  securityContext: 1
    fsGroup: 5555 2
  ...
```

1

As with **supplementalGroups**, **fsGroup** must be defined globally to the pod, not per container.

2

5555 will become the group ID for the volume's group permissions and for all new files created in the volume.

As with **supplementalGroups**, all containers in the above pod (assuming the matching SCC or project allows the group **5555**) will be members of the group **5555**, and will have access to the block volume, regardless of the container's user ID. If the pod matches the **restricted** SCC, whose **fsGroup** strategy is **MustRunAs**, then the pod will fail to run. However, if the SCC has its**fsGroup** strategy set to **RunAsAny**, then any **fsGroup** ID (including **5555**) will be accepted. Note that if the SCC has its **fsGroup** strategy set to **RunAsAny** and no **fsGroup** ID is specified, the "taking over" of the block storage does not occur and permissions may be denied to the pod.

fsGroups and Custom SCCs

To remedy the situation in the previous example, a custom SCC can be created such that:

- a minimum and maximum group ID are defined,
- ID range checking is enforced, and
- the group ID of **5555** is allowed.

It is better to create new SCCs versus modifying a predefined SCC, or changing the range of allowed IDs in the predefined projects.

Consider the following fragment of a new SCC definition:

```
# oc export scc new-scc
...
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs (1)
  ranges: (2)
  - max: 6000
    min: 5000 (3)
...
```

1

MustRunAs triggers group ID range checking, whereas **RunAsAny** does not require range checking.

2

The range of allowed group IDs is 5000 through, and including, 5999. Multiple ranges are supported but not used. The allowed group ID range here is 5000 through 5999, with the default **fsGroup** being 5000.

3

The minimum value (or the entire range) can be omitted from the SCC, and thus range checking and generating a default value will defer to the project's **`openshift.io/sa.scc.supplemental-groups`** range. **`fsGroup`** and **`supplementalGroups`** use the same group field in the project; there is not a separate range for **`fsGroup`**.

When the pod shown above runs against this new SCC (assuming, of course, the pod matches the new SCC), it will start because the group **5555**, supplied in the pod definition, is allowed by the custom SCC. Additionally, the pod will "take over" the block device, so when the block storage is viewed by a process outside of the pod, it will actually have **5555** as its group ID.

A list of volumes supporting block ownership include:

- AWS Elastic Block Store
- OpenStack Cinder
- Ceph RBD
- GCE Persistent Disk
- iSCSI
- emptyDir
- gitRepo

Note

This list is potentially incomplete.

21.12.5. User IDs

Note

Read SCCs, Defaults, and Allowed Ranges before working with supplemental groups.

Tip

It is generally preferable to use group IDs (supplemental or fsGroup) to gain access to persistent storage versus using user IDs.

User IDs can be defined in the container image or in the pod definition. In the pod definition, a single user ID can be defined globally to all containers, or specific to individual containers (or both). A user ID is supplied as shown in the pod definition fragment below:

```
spec:
  containers:
  - name: ...
```

```
    securityContext:
      runAsUser: 1000100001
```

ID 1000100001 in the above is container-specific and matches the owner ID on the export. If the NFS export's owner ID was **54321**, then that number would be used in the pod definition. Specifying **`securityContext`** outside of the container definition makes the ID global to all containers in the pod.

Similar to group IDs, user IDs may be validated according to policies set in the SCC and/or project. If the SCC's **`runAsUser`** strategy is set to **RunAsAny**, then any user ID defined in the pod definition or in the image is allowed.

Warning

This means even a UID of **0** (root) is allowed.

If, instead, the **`runAsUser`** strategy is set to **MustRunAsRange**, then a supplied user ID will be validated against a range of allowed IDs. If the pod supplies no user ID, then the default ID is set to the minimum value of the range of allowable user IDs.

Returning to the earlier NFS example, the container needs its UID set to **1000100001**, which is shown in the pod fragment above. Assuming the **default** project and the **restricted** SCC, the pod's requested user ID of 1000100001 will not be allowed, and therefore the pod will fail. The pod fails because:

- it requests **1000100001** as its user ID,
- all available SCCs use **MustRunAsRange** for their **`runAsUser`** strategy, so UID range checking is required, and
- **1000100001** is not included in the SCC or in the project's user ID range.

To remedy this situation, a new SCC can be created with the appropriate user ID range. A new project could also be created with the appropriate user ID range defined. There are also other, less-preferred options:

- The **restricted** SCC could be modified to include **1000100001** within its minimum and maximum user ID range. This is not recommended as you should avoid modifying the predefined SCCs if possible.
- The **restricted** SCC could be modified to use **RunAsAny** for the **`runAsUser`** value, thus eliminating ID range checking. This is *strongly* not recommended, as containers could run as root.
- The **default** project's UID range could be changed to allow a user ID of **1000100001**. This is not generally advisable because only a single range of user IDs can be specified, and thus other pods may not run if the range is altered.

User IDs and Custom SCCs

It is good practice to avoid modifying the predefined SCCs if possible. The preferred approach is to create a custom SCC that better fits an organization's security needs, or create a new project that supports the desired user IDs.

To remedy the situation in the previous example, a custom SCC can be created such that:

- a minimum and maximum user ID is defined,
- UID range checking is still enforced, and
- the UID of **1000100001** is allowed.

For example:

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false 1
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc 2
priority: 9 3
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsRange 4
  uidRangeMax: 1000100001 5
  uidRangeMin: 1000100001
...
```

1

The **allowXX** bools are the same as for the **restricted** SCC.

2

The name of this new SCC is **nfs-scc**.

3

Numerically larger numbers have greater priority. Nil or omitted is the lowest priority. Higher priority SCCs sort before lower priority SCCs, and thus have a better chance of matching a new pod.

4

The **runAsUser** strategy is set to **MustRunAsRange**, which means UID range checking is enforced.

5

The UID range is 1000100001 through 1000100001 (a range of one value).

Now, with **runAsUser: 1000100001** shown in the previous pod definition fragment, the pod matches the new **nfs-scc** and is able to run with a UID of 1000100001.

21.12.6. SELinux Options

All predefined SCCs, except for the **privileged** SCC, set the **`seLinuxContext`** to **MustRunAs**. So the SCCs most likely to match a pod's requirements will force the pod to use an SELinux policy. The SELinux policy used by the pod can be defined in the pod itself, in the image, in the SCC, or in the project (which provides the default).

SELinux labels can be defined in a pod's **`securityContext.seLinuxOptions`** section, and supports **`user`**, **`role`**, **`type`**, and **`level`**:

Note

Level and MCS label are used interchangeably in this topic.

```
...
 securityContext: 1
    seLinuxOptions:
      level: "s0:c123,c456" 2
...
```

1

`level` can be defined globally for the entire pod, or individually for each container.

2

SELinux level label.

Here are fragments from an SCC and from the **default** project:

```
$ oc export scc scc-name
...
seLinuxContext:
  type: MustRunAs 1

# oc export project default
...
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c1,c0 2
...
```

1

MustRunAs causes volume relabeling.

2

If the label is not provided in the pod or in the SCC, then the default comes from the project.

All predefined SCCs, except for the **privileged** SCC, set the **`seLinuxContext`** to **MustRunAs**. This forces pods to use MCS labels, which can be defined in the pod definition, the image, or provided as a default.

The SCC determines whether or not to require an SELinux label and can provide a default label. If the **`seLinuxContext`** strategy is set to **MustRunAs** and the pod (or image) does not define a label, OpenShift Container Platform defaults to a label chosen from the SCC itself or from the project.

If **`seLinuxContext`** is set to **RunAsAny**, then no default labels are provided, and the container determines the final label. In the case of Docker, the container will use a unique MCS label, which will not likely match the labeling on existing storage mounts. Volumes which support SELinux management will be relabeled so that they are accessible by the specified label and, depending on how exclusionary the label is, only that label.

This means two things for unprivileged containers:

- The volume will be given a **`type`** which is accessible by unprivileged containers. This **`type`** is usually **svirt_sandbox_file_t**.
- If a **`level`** is specified, the volume will be labeled with the given MCS label.

For a volume to be accessible by a pod, the pod must have both categories of the volume. So a pod with **s0:c1,c2** will be able to access a volume with **s0:c1,c2**. A volume with **s0** will be accessible by all pods.

If pods fail authorization, or if the storage mount is failing due to permissions errors, then there is a possibility that SELinux enforcement is interfering. One way to check for this is to run:

```
# ausearch -m avc --start recent
```

This examines the log file for AVC (Access Vector Cache) errors.

21.13. SELECTOR-LABEL VOLUME BINDING

21.13.1. Overview

This guide provides the steps necessary to enable binding of persistent volume claims (PVCs) to persistent volumes (PVs) via **selector** and **label** attributes. By implementing selectors and labels, regular users are able to target provisioned storage by identifiers defined by a cluster administrator.

21.13.2. Motivation

In cases of statically provisioned storage, developers seeking persistent storage are required to know a handful identifying attributes of a PV in order to deploy and bind a PVC. This creates several problematic situations. Regular users might have to contact a cluster administrator to either deploy the PVC or provide the PV values. PV attributes alone do not convey the intended use of the storage volumes, nor do they provide methods by which volumes can be grouped.

Selector and label attributes can be used to abstract away PV details from the user while providing cluster administrators a way of identifying volumes by a descriptive and customizable tag. Through the selector-label method of binding, users are only required to know which labels are defined by the administrator.

Note

The selector-label feature is currently only available for *statically* provisioned storage and is currently not implemented for storage provisioned dynamically.

21.13.3. Deployment

This section reviews how to define and deploy PVCs.

21.13.3.1. Prerequisites

1. A running OpenShift Container Platform 3.3+ cluster
2. A volume provided by a supported storage provider
3. A user with a cluster-admin role binding

21.13.3.2. Define the Persistent Volume and Claim

1. As the **cluser-admin** user, define the PV. For this example, we will be using a GlusterFS volume. See the appropriate storage provider for your provider's configuration.

Example 21.22. Persistent Volume with Labels

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: 1
    volume-type: ssd
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

1

A PVC whose selectors match *all* of a PV's labels will be bound, assuming a PV is available.

2. Define the PVC:

Example 21.23. Persistent Volume Claim with Selectors

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
  selector: 1
    matchLabels: 2
      volume-type: ssd
      aws-availability-zone: us-east-1
```

1

Begin **selectors** section.

2

List all labels by which the user is requesting storage. Must match *all* labels of targeted PV.

21.13.3.3. Deploy the Persistent Volume and Claim

As the **cluster-admin** user, create the persistent volume:

Example 21.24. Create the Persistent Volume

```
# oc create -f gluster-pv.yaml
persistentVolume "gluster-volume" created

# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
gluster-volume            map[]    2147483648   RWX
Available                       2s
```

Once the PV is created, any user whose selectors match *all* its labels can create their PVC.

Example 21.25. Create the Persistent Volume Claim

```
# oc create -f gluster-pvc.yaml
persistentVolumeClaim "gluster-claim" created
# oc get pvc
NAME          LABELS    STATUS    VOLUME
gluster-claim           Bound     gluster-volume
```

CHAPTER 22. PERSISTENT STORAGE EXAMPLES

22.1. OVERVIEW

The following sections provide detailed, comprehensive instructions on setting up and configuring common storage use cases. These examples cover both the administration of persistent volumes and their security, and how to claim against the volumes as a user of the system.

- Sharing an NFS PV Across Two Pods
- Ceph-RBD Block Storage Volume
- Shared Storage Using a GlusterFS Volume
- Dynamic Provisioning Storage Using GlusterFS
- Mounting a PV to Privileged Pods
- Backing Docker Registry with GlusterFS Storage
- Binding Persistent Volumes by Labels

22.2. SHARING AN NFS MOUNT ACROSS TWO PERSISTENT VOLUME CLAIMS

22.2.1. Overview

The following use case describes how a cluster administrator wanting to leverage shared storage for use by two separate containers would configure the solution. This example highlights the use of NFS, but can easily be adapted to other shared storage types, such as GlusterFS. In addition, this example will show configuration of pod security as it relates to shared storage.

Persistent Storage Using NFS provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and using NFS as persistent storage. This topic shows and end-to-end example of using an existing NFS cluster and OpenShift Container Platform persistent store, and assumes an existing NFS server and exports exist in your OpenShift Container Platform infrastructure.

Note

All **oc** commands are executed on the OpenShift Container Platform master host.

22.2.2. Creating the Persistent Volume

Before creating the PV object in OpenShift Container Platform, the persistent volume (PV) file is defined:

Example 22.1. Persistent Volume Object Definition Using NFS

```
apiVersion: v1
kind: PersistentVolume
```

```
metadata:
  name: nfs-pv 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
    - ReadWriteMany 3
  persistentVolumeReclaimPolicy: Retain 4
  nfs: 5
    path: /opt/nfs 6
    server: nfs.f22 7
    readOnly: false
```

1

The name of the PV, which is referenced in pod definitions or displayed in various **oc** volume commands.

2

The amount of storage allocated to this volume.

3

accessModes are used as labels to match a PV and a PVC. They currently do not define any form of access control.

4

A volume reclaim policy of **retain** indicates to preserve the volume after the pods.

5

This defines the volume type being used, in this case the **NFS** plug-in.

6

This is the NFS mount path.

7

This is the NFS server. This can also be specified by IP address.

Save the PV definition to a file, for example ***nfs-pv.yaml***, and create the persistent volume:

```
# oc create -f nfs-pv.yaml
persistentvolume "nfs-pv" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME         LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
nfs-pv       <none>    1Gi        RWX           Available
37s
```

22.2.3. Creating the Persistent Volume Claim

A persistent volume claim (PVC) specifies the desired access mode and storage capacity. Currently, based on only these two attributes, a PVC is bound to a single PV. Once a PV is bound to a PVC, that PV is essentially tied to the PVC's project and cannot be bound to by another PVC. There is a one-to-one mapping of PVs and PVCs. However, multiple pods in the same project can use the same PVC. This is the use case we are highlighting in this example.

Example 22.2. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc  1
spec:
  accessModes:
  - ReadWriteMany      2
  resources:
     requests:
       storage: 1Gi    3
```

1 The claim name is referenced by the pod under its **`volumes`** section.

2 As mentioned above for PVs, the **`accessModes`** do not enforce access right, but rather act as labels to match a PV to a PVC.

3 This claim will look for PVs offering **1Gi** or greater capacity.

Save the PVC definition to a file, for example ***nfs-pvc.yaml***, and create the PVC:

```
# oc create -f nfs-pvc.yaml
persistentvolumeclaim "nfs-pvc" created
```

Verify that the PVC was created and bound to the expected PV:

```
# oc get pvc
NAME            LABELS    STATUS    VOLUME       CAPACITY
ACCESSMODES   AGE
nfs-pvc         <none>    Bound     nfs-pv       1Gi        RWX
24s
                                    1
```

1

The claim, **nfs-pvc**, was bound to the **nfs-pv** PV.

22.2.4. Ensuring NFS Volume Access

Access is necessary to a node in the NFS server. On this node, examine the NFS export mount:

```
[root@nfs nfs]# ls -lZ /opt/nfs/
total 8
-rw-r--r--. 1 root 100003  system_u:object_r:usr_t:s0     10 Oct 12
23:27 test2b
              1
                    2
```

1

the owner has ID 0.

2

the group has ID 100003.

In order to access the NFS mount, the container must match the SELinux label, and either run with a UID of 0, or with 100003 in its supplemental groups range. Gain access to the volume by matching the NFS mount's groups, which will be defined in the pod definition below.

By default, SELinux does not allow writing from a pod to a remote NFS server. To enable writing to NFS volumes with SELinux enforcing on each node, run:

```
# setsebool -P virt_sandbox_use_nfs on
# setsebool -P virt_use_nfs on
```

Note

The **virt_sandbox_use_nfs** boolean is defined by the **docker-selinux** package. If you get an error saying it is not defined, ensure that this package is installed.

22.2.5. Creating the Pod

A pod definition file or a template file can be used to define a pod. Below is a pod specification that creates a single container and mounts the NFS volume for read-write access:

Example 22.3. Pod Object Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-nfs-pod 1
  labels:
    name: nginx-nfs-pod
spec:
  containers:
    - name: nginx-nfs-pod
      image: fedora/nginx 2
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol 3
          mountPath: /usr/share/nginx/html 4
  securityContext:
      supplementalGroups: [100003] 5
      privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: nfs-pvc 6
```

1

The name of this pod as displayed by **oc get pod**.

2

The image run by this pod.

3

The name of the volume. This name must be the same in both the **`containers`** and **`volumes`** sections.

4

The mount path as seen in the container.

5

The group ID to be assigned to the container.

6

The PVC that was created in the previous step.

Save the pod definition to a file, for example ***nfs.yaml***, and create the pod:

```
# oc create -f nfs.yaml
pod "nginx-nfs-pod" created
```

Verify that the pod was created:

```
# oc get pods
NAME                READY     STATUS    RESTARTS   AGE
nginx-nfs-pod       1/1       Running   0          4s
```

More details are shown in the **`oc describe pod`** command:

```
[root@ose70 nfs]# oc describe pod nginx-nfs-pod
Name:    nginx-nfs-pod
Namespace:   default (1)
Image(s):   fedora/nginx
Node:    ose70.rh7/192.168.234.148 (2)
Start Time:   Mon, 21 Mar 2016 09:59:47 -0400
Labels:    name=nginx-nfs-pod
Status:    Running
Reason:
Message:
IP:    10.1.0.4
Replication Controllers: <none>
Containers:
  nginx-nfs-pod:
    Container ID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc
6f
    Image:  fedora/nginx
    Image ID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a0
```

```
0a
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 09:59:49 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
  nfsvol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName: nfs-pvc 3
    ReadOnly: false
  default-token-a06zb:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-a06zb
Events: 4
  FirstSeen LastSeen Count From   SubobjectPath    Reason  Message
  _________ ________ _____ ____   _____________    ______  _______

  4m  4m  1 {scheduler }       Scheduled Successfully assigned nginx-
nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Pulled  Container image "openshift3/ose-pod:v3.1.0.4" already present
on machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Created  Created with docker id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Started  Started with docker id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} spec.containers{nginx-nfs-pod}  Pulled
Container image "fedora/nginx" already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{nginx-nfs-pod}  Created
Created with docker id a3292104d6c2
  4m  4m  1 {kubelet ose70.rh7} spec.containers{nginx-nfs-pod}  Started
Started with docker id a3292104d6c2
```

1

The project (namespace) name.

2

The IP address of the OpenShift Container Platform node running the pod.

3

The PVC name used by the pod.

4

The list of events resulting in the pod being launched and the NFS volume being mounted. The container will not start correctly if the volume cannot mount.

There is more internal information, including the SCC used to authorize the pod, the pod's user and group IDs, the SELinux label, and more, shown in the **`oc get pod <name> -o yaml`** command:

```
[root@ose70 nfs]# oc get pod nginx-nfs-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/scc: restricted (1)
  creationTimestamp: 2016-03-21T13:59:47Z
  labels:
    name: nginx-nfs-pod
  name: nginx-nfs-pod
  namespace: default (2)
  resourceVersion: "2814411"
  selflink: /api/v1/namespaces/default/pods/nginx-nfs-pod
  uid: 2c22d2ea-ef6d-11e5-adc7-000c2900f1e3
spec:
  containers:
  - image: fedora/nginx
    imagePullPolicy: IfNotPresent
    name: nginx-nfs-pod
    ports:
    - containerPort: 80
      name: web
      protocol: TCP
    resources: {}
    securityContext:
      privileged: false
    terminationMessagePath: /dev/termination-log
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: nfsvol
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-a06zb
      readOnly: true
  dnsPolicy: ClusterFirst
  host: ose70.rh7
  imagePullSecrets:
  - name: default-dockercfg-xvdew
  nodeName: ose70.rh7
  restartPolicy: Always
  securityContext:
    supplementalGroups:
    - 100003 (3)
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes:
```

```
  - name: nfsvol
    persistentVolumeClaim:
      claimName: nfs-pvc 4
  - name: default-token-a06zb
    secret:
      secretName: default-token-a06zb
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-03-21T13:59:49Z
    status: "True"
    type: Ready
  containerStatuses:
  - containerID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc
6f
    image: fedora/nginx
    imageID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a0
0a
    lastState: {}
    name: nginx-nfs-pod
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-03-21T13:59:49Z
  hostIP: 192.168.234.148
  phase: Running
  podIP: 10.1.0.4
  startTime: 2016-03-21T13:59:47Z
```

1

The SCC used by the pod.

2

The project (namespace) name.

3

The supplemental group ID for the pod (all containers).

4

The PVC name used by the pod.

22.2.6. Creating an Additional Pod to Reference the Same PVC

This pod definition, created in the same namespace, uses a different container. However, we can use the same backing storage by specifying the claim name in the volumes section below:

Example 22.4. Pod Object Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-nfs-pod 1
  labels:
    name: busybox-nfs-pod
spec:
  containers:
  - name: busybox-nfs-pod
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: nfsvol-2 3
      mountPath: /usr/share/busybox  4
      readOnly: false
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
  - name: nfsvol-2
    persistentVolumeClaim:
      claimName: nfs-pvc 6
```

1

The name of this pod as displayed by **`oc get pod`**.

2

The image run by this pod.

3

The name of the volume. This name must be the same in both the **`containers`** and **`volumes`** sections.

4

The mount path as seen in the container.

5

The group ID to be assigned to the container.

6

The PVC that was created earlier and is also being used by a different container.

Save the pod definition to a file, for example ***nfs-2.yaml***, and create the pod:

```
# oc create -f nfs-2.yaml
pod "busybox-nfs-pod" created
```

Verify that the pod was created:

```
# oc get pods
NAME                READY     STATUS    RESTARTS   AGE
busybox-nfs-pod     1/1       Running   0          3s
```

More details are shown in the **`oc describe pod`** command:

```
[root@ose70 nfs]# oc describe pod busybox-nfs-pod
Name:    busybox-nfs-pod
Namespace:   default
Image(s):   busybox
Node:    ose70.rh7/192.168.234.148
Start Time:   Mon, 21 Mar 2016 10:19:46 -0400
Labels:    name=busybox-nfs-pod
Status:    Running
Reason:
Message:
IP:    10.1.0.5
Replication Controllers: <none>
Containers:
  busybox-nfs-pod:
    Container ID:
docker://346d432e5a4824ebf5a47fceb4247e0568ecc64eadcc160e9bab481aecfb05
94
    Image:  busybox
    Image ID:
docker://17583c7dd0dae6244203b8029733bdb7d17fccbb2b5d93e2b24cf48b8bfd06
e2
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 10:19:48 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
```

```
Volumes:
  nfsvol-2:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName: nfs-pvc
    ReadOnly: false
  default-token-32d2z:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-32d2z
Events:
  FirstSeen LastSeen Count From   SubobjectPath    Reason  Message
  ————————— ———————— ————— ————   ———————————————    ——————  ———————

  4m  4m  1 {scheduler }       Scheduled Successfully assigned
busybox-nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Pulled  Container image "openshift3/ose-pod:v3.1.0.4" already present
on machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Created  Created with docker id 249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Started  Started with docker id 249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Pulled
Container image "busybox" already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod}
Created  Created with docker id 346d432e5a48
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod}
Started  Started with docker id 346d432e5a48
```

As you can see, both containers are using the same storage claim that is attached to the same NFS mount on the back end.

22.3. COMPLETE EXAMPLE USING CEPH RBD

22.3.1. Overview

This topic provides an end-to-end example of using an existing Ceph cluster as an OpenShift Container Platform persistent store. It is assumed that a working Ceph cluster is already set up. If not, consult the Overview of Red Hat Ceph Storage.

Persistent Storage Using Ceph Rados Block Device provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and using Ceph RBD as persistent storage.

Note

All **oc ...** commands are executed on the OpenShift Container Platform master host.

22.3.2. Installing the ceph-common Package

The **ceph-common** library must be installed on **all schedulable** OpenShift Container Platform nodes:

Note

The OpenShift Container Platform all-in-one host is not often used to run pod workloads and, thus, is not included as a schedulable node.

```
# yum install -y ceph-common
```

22.3.3. Creating the Ceph Secret

The **ceph auth get-key** command is run on a Ceph **MON** node to display the key value for the **client.admin** user:

Example 22.5. Ceph Secret Definition

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
```

1

This base64 key is generated on one of the Ceph MON nodes using the **ceph auth get-key client.admin | base64** command, then copying the output and pasting it as the secret key's value.

Save the secret definition to a file, for example ***ceph-secret.yaml***, then create the secret:

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

Verify that the secret was created:

```
# oc get secret ceph-secret
NAME          TYPE      DATA      AGE
ceph-secret   Opaque    1         23d
```

22.3.4. Creating the Persistent Volume

Next, before creating the PV object in OpenShift Container Platform, define the persistent volume file:

Example 22.6. Persistent Volume Object Definition Using Ceph RBD

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv      1
spec:
  capacity:
    storage: 2Gi     2
  accessModes:
    - ReadWriteOnce  3
  rbd:               4
    monitors:        5
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret  6
    fsType: ext4         7
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

1

The name of the PV, which is referenced in pod definitions or displayed in various **oc** volume commands.

2

The amount of storage allocated to this volume.

3

accessModes are used as labels to match a PV and a PVC. They currently do not define any form of access control. All block storage is defined to be single user (non-shared storage).

4

This defines the volume type being used. In this case, the **rbd** plug-in is defined.

5

This is an array of Ceph monitor IP addresses and ports.

6

This is the Ceph secret, defined above. It is used to create a secure connection from

This is the Ceph secret, defined above. It is used to create a secure connection from OpenShift Container Platform to the Ceph server.

7

This is the file system type mounted on the Ceph RBD block device.

Save the PV definition to a file, for example ***ceph-pv.yaml***, and create the persistent volume:

```
# oc create -f ceph-pv.yaml
persistentvolume "ceph-pv" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES   STATUS
CLAIM     REASON    AGE
ceph-pv                  <none>    2147483648   RWO
Available                       2s
```

22.3.5. Creating the Persistent Volume Claim

A persistent volume claim (PVC) specifies the desired access mode and storage capacity. Currently, based on only these two attributes, a PVC is bound to a single PV. Once a PV is bound to a PVC, that PV is essentially tied to the PVC's project and cannot be bound to by another PVC. There is a one-to-one mapping of PVs and PVCs. However, multiple pods in the same project can use the same PVC.

Example 22.7. PVC Object Definition

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes:     1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

1

As mentioned above for PVs, the **accessModes** do not enforce access right, but rather act as labels to match a PV to a PVC.

2

This claim will look for PVs offering **2Gi** or greater capacity.

Save the PVC definition to a file, for example ***ceph-claim.yaml***, and create the PVC:

```
# oc create -f ceph-claim.yaml
persistentvolumeclaim "ceph-claim" created

#and verify the PVC was created and bound to the expected PV:
# oc get pvc
NAME         LABELS    STATUS    VOLUME    CAPACITY   ACCESSMODES
AGE
ceph-claim   <none>    Bound     ceph-pv   1Gi        RWX
21s
                                  1
```

1

the claim was bound to the **ceph-pv** PV.

22.3.6. Creating the Pod

A pod definition file or a template file can be used to define a pod. Below is a pod specification that creates a single container and mounts the Ceph RBD volume for read-write access:

Example 22.8. Pod Object Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1           1
spec:
  containers:
  - name: ceph-busybox
    image: busybox          2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1       3
      mountPath: /usr/share/busybox 4
      readOnly: false
  volumes:
  - name: ceph-vol1         5
    persistentVolumeClaim:
      claimName: ceph-claim 6
```

1

The name of this pod as displayed by **`oc get pod`**.

2

The image run by this pod. In this case, we are telling **busybox** to sleep.

3 5

The name of the volume. This name must be the same in both the **`containers`** and **`volumes`** sections.

4

The mount path as seen in the container.

6

The PVC that is bound to the Ceph RBD cluster.

Save the pod definition to a file, for example ***ceph-pod1.yaml***, and create the pod:

```
# oc create -f ceph-pod1.yaml
pod "ceph-pod1" created

#verify pod was created
# oc get pod
NAME        READY     STATUS    RESTARTS   AGE
ceph-pod1   1/1       Running   0          2m
                      1
```

1

After a minute or so, the pod will be in the **Running** state.

22.3.7. Defining Group and Owner IDs (Optional)

When using block storage, such as Ceph RBD, the physical block storage is **managed** by the pod. The group ID defined in the pod becomes the group ID of **both** the Ceph RBD mount inside the container, and the group ID of the actual storage itself. Thus, it is usually unnecessary to define a group ID in the pod specifiation. However, if a group ID is desired, it can be defined using **`fsGroup`**, as shown in the following pod definition fragment:

Example 22.9. Group ID Pod Definition

```
...
```

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777  2
...
```

1

`securityContext` must be defined at the pod level, not under a specific container.

2

All containers in the pod will have the same **`fsGroup`** ID.

22.4. COMPLETE EXAMPLE USING GLUSTERFS

22.4.1. Overview

This topic provides an end-to-end example of how to use an existing Gluster cluster as an OpenShift Container Platform persistent store. It is assumed that a working Gluster cluster is already set up. If not, consult the Red Hat Gluster Storage Administration Guide.

Persistent Storage Using GlusterFS provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and using GlusterFS as persistent storage.

For an end-to-end example of how to dynamically provision GlusterFS volumes, see Complete Example of Dynamic Provisioning Using GlusterFS. The persistent volume (PV) and endpoints are both created dynamically by GlusterFS.

Note

All **oc ...** commands are executed on the OpenShift Container Platform master host.

22.4.2. Installing the glusterfs-fuse Package

The **glusterfs-fuse** library must be installed on all **schedulable** OpenShift Container Platform nodes:

```
# yum install -y glusterfs-fuse
```

Note

The OpenShift Container Platform all-in-one host is often not used to run pod workloads and, thus, is not included as a schedulable node.

22.4.3. Creating the Gluster Endpoints and Gluster Service for Persistence

The named endpoints define each node in the Gluster-trusted storage pool:

Example 22.10. GlusterFS Endpoint Definition

```
apiVersion: v1
kind: Endpoints
metadata:
  name: gluster-cluster 1
subsets:
- addresses:              2
  - ip: 192.168.122.21
  ports:                  3
  - port: 1
    protocol: TCP
- addresses:
  - ip: 192.168.122.22
  ports:
  - port: 1
    protocol: TCP
```

1 The endpoints name. If using a service, then the endpoints name must match the service name.

2 An array of IP addresses for each node in the Gluster pool. Currently, host names are not supported.

3 The port numbers are ignored, but must be legal port numbers. The value **1** is commonly used.

Save the endpoints definition to a file, for example ***gluster-endpoints.yaml***, then create the endpoints object:

```
# oc create -f gluster-endpoints.yaml
```

```
endpoints "gluster-cluster" created
```

Verify that the endpoints were created:

```
# oc get endpoints gluster-cluster
NAME                ENDPOINTS                                 AGE
gluster-cluster     192.168.122.21:1,192.168.122.22:1   1m
```

Note

To persist the Gluster endpoints, you also need to create a service.

Note

Endpoints are name-spaced. Each project accessing the Gluster volume needs its own endpoints.

Example 22.11. GlusterFS Service Definition

```
apiVersion: v1
kind: Service
metadata:
  name: gluster-cluster 1
spec:
  ports:
  - port: 1 2
```

1

The name of the service. If using a service, then the endpoints name must match the service name.

2

The port should match the same port used in the endpoints.

Save the service definition to a file, for example ***gluster-service.yaml***, then create the endpoints object:

```
# oc create -f gluster-service.yaml
endpoints "gluster-cluster" created
```

Verify that the service was created:

```
# oc get service gluster-cluster
NAME                CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
gluster-cluster     10.0.0.130   <none>        1/TCP      9s
```

22.4.4. Creating the Persistent Volume

Next, before creating the PV object, define the persistent volume in OpenShift Container Platform:

Example 22.12. Persistent Volume Object Definition Using GlusterFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-pv    1
spec:
  capacity:
    storage: 1Gi      2
  accessModes:
  - ReadWriteMany     3
  glusterfs:          4
    endpoints: gluster-cluster 5
    path: /HadoopVol  6
    readOnly: false
  persistentVolumeReclaimPolicy: Retain 7
```

1

The name of the PV, which is referenced in pod definitions or displayed in various **oc** volume commands.

2

The amount of storage allocated to this volume.

3

accessModes are used as labels to match a PV and a PVC. They currently do not define any form of access control.

4

This defines the volume type being used. In this case, the **glusterfs** plug-in is defined.

5

This references the endpoints named above.

6

This is the Gluster volume name, preceded by `/`.

7

A volume reclaim policy of **retain** indicates that the volume will be preserved after the pods accessing it terminate. Accepted values include Retain, Delete, and Recycle.

Save the PV definition to a file, for example ***gluster-pv.yaml***, and create the persistent volume:

```
# oc create -f gluster-pv.yaml
persistentvolume "gluster-pv" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME         LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-pv   <none>    1Gi        RWX           Available
37s
```

22.4.5. Creating the Persistent Volume Claim

A persistent volume claim (PVC) specifies the desired access mode and storage capacity. Currently, based on only these two attributes, a PVC is bound to a single PV. Once a PV is bound to a PVC, that PV is essentially tied to the PVC's project and cannot be bound to by another PVC. There is a one-to-one mapping of PVs and PVCs. However, multiple pods in the same project can use the same PVC.

Example 22.13. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim   1
spec:
  accessModes:
  - ReadWriteMany       2
  resources:
     requests:
       storage: 1Gi     3
```

1

The claim name is referenced by the pod under its **volumes** section.

2

As mentioned above for PVs, the **accessModes** do not enforce access rights, but rather act as labels to match a PV to a PVC.

3

This claim will look for PVs offering **1Gi** or greater capacity.

Save the PVC definition to a file, for example ***gluster-claim.yaml***, and create the PVC:

```
# oc create -f gluster-claim.yaml
persistentvolumeclaim "gluster-claim" created
```

Verify the PVC was created and bound to the expected PV:

```
# oc get pvc
NAME            LABELS    STATUS    VOLUME       CAPACITY
ACCESSMODES   AGE
gluster-claim   <none>    Bound     gluster-pv   1Gi        RWX
24s
                                    1
```

1

The claim was bound to the **gluster-pv** PV.

22.4.6. Defining GlusterFS Volume Access

Access is necessary to a node in the Gluster-trusted storage pool. On this node, examine the **glusterfs-fuse** mount:

```
# ls -lZ /mnt/glusterfs/
drwxrwx---. yarn hadoop system_u:object_r:fusefs_t:s0    HadoopVol

# id yarn
uid=592(yarn) gid=590(hadoop) groups=590(hadoop)
    1
                2
```

1

The owner has ID 592.

2

The group has ID 590.

In order to access the **HadoopVol** volume, the container must match the SELinux label, and either run with a UID of 592, or with 590 in its supplemental groups. It is recommended to gain access to the volume by matching the Gluster mount's groups, which is defined in the pod definition below.

By default, SELinux does not allow writing from a pod to a remote Gluster server. To enable writing to GlusterFS volumes with SELinux enforcing on each node, run:

```
# setsebool -P virt_sandbox_use_fusefs on
```

Note

The **`virt_sandbox_use_fusefs`** boolean is defined by the **docker-selinux** package. If you get an error saying it is not defined, ensure that this package is installed.

22.4.7. Creating the Pod using NGINX Web Server image

A pod definition file or a template file can be used to define a pod. Below is a pod specification that creates a single container and mounts the Gluster volume for read-write access:

Note

The NGINX image may require to run in privileged mode to create the mount and run properly. An easy way to accomplish this is to simply add your user to the **privileged** Security Context Constraint (SCC):

```
$ oadm policy add-scc-to-user privileged myuser
```

Then, add the **privileged: true** to the containers **`securityContext:`** section of the YAML file (as seen in the example below).

Managing Security Context Constraints provides additional information regarding SCCs.

Example 22.14. Pod Object Definition using NGINX image

```
apiVersion: v1
kind: Pod
metadata:
  name: gluster-pod1
  labels:
    name: gluster-pod1    1
spec:
  containers:
  - name: gluster-pod1
    image: nginx          2
```

```
    ports:
    - name: web
      containerPort: 80
    securityContext:
      privileged: true
    volumeMounts:
    - name: gluster-vol1 3
      mountPath: /usr/share/nginx/html 4
      readOnly: false
  securityContext:
    supplementalGroups: [590]       5
  volumes:
  - name: gluster-vol1    6
    persistentVolumeClaim:
      claimName: gluster-claim      7
```

1

The name of this pod as displayed by **`oc get pod`**.

2

The image run by this pod. In this case, we are using a standard NGINX image.

3 6

The name of the volume. This name must be the same in both the **`containers`** and **`volumes`** sections.

4

The mount path as seen in the container.

5

The **`SupplementalGroup`** ID (Linux Groups) to be assigned at the pod level and as discussed this should match the POSIX permissions on the Gluster volume.

7

The PVC that is bound to the Gluster cluster.

Save the pod definition to a file, for example ***gluster-pod1.yaml***, and create the pod:

```
# oc create -f gluster-pod1.yaml
pod "gluster-pod1" created
```

Verify the pod was created:

```
# oc get pod
NAME           READY     STATUS    RESTARTS   AGE
gluster-pod1   1/1       Running   0          31s

                         1
```

1

After a minute or so, the pod will be in the **Running** state.

More details are shown in the **`oc describe pod`** command:

```
# oc describe pod gluster-pod1
Name:   gluster-pod1
Namespace:  default  1
Security Policy: privileged
Node:   ose1.rhs/192.168.122.251
Start Time:  Wed, 24 Aug 2016 12:37:45 -0400
Labels:   name=gluster-pod1
Status:   Running
IP:   172.17.0.2  2
Controllers:  <none>
Containers:
  gluster-pod1:
    Container ID:
docker://e67ed01729e1dc7369c5112d07531a27a7a02a7eb942f17d1c5fce32d8c31a
2d
    Image:  nginx
    Image ID:
docker://sha256:4efb2fcdb1ab05fb03c9435234343c1cc65289eeb016be86193e88d
3a5d84f6b
    Port:  80/TCP
    State:  Running
      Started:  Wed, 24 Aug 2016 12:37:52 -0400
    Ready:  True
    Restart Count: 0
    Volume Mounts:
      /usr/share/nginx/html/test from glustervol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-
1n70u (ro)
    Environment Variables: <none>
Conditions:
  Type  Status
  Initialized  True
  Ready  True
  PodScheduled  True
```

```
Volumes:
  glustervol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName: gluster-claim  3
    ReadOnly: false
  default-token-1n70u:
    Type: Secret (a volume populated by a Secret)
    SecretName: default-token-1n70u
QoS Tier: BestEffort
Events:    4
  FirstSeen LastSeen Count From   SubobjectPath   Type  Reason  Message
  --------- -------- ----- ----   -------------   -------- ------  -
------
  10s  10s  1 {default-scheduler }     Normal  Scheduled Successfully
assigned gluster-pod1 to ose1.rhs
  9s  9s  1 {kubelet ose1.rhs} spec.containers{gluster-pod1} Normal
Pulling  pulling image "nginx"
  4s  4s  1 {kubelet ose1.rhs} spec.containers{gluster-pod1} Normal
Pulled  Successfully pulled image "nginx"
  3s  3s  1 {kubelet ose1.rhs} spec.containers{gluster-pod1} Normal
Created  Created container with docker id e67ed01729e1
  3s  3s  1 {kubelet ose1.rhs} spec.containers{gluster-pod1} Normal
Started  Started container with docker id e67ed01729e1
```

1

The project (namespace) name.

2

The IP address of the OpenShift Container Platform node running the pod.

3

The PVC name used by the pod.

4

The list of events resulting in the pod being launched and the Gluster volume being mounted.

There is more internal information, including the SCC used to authorize the pod, the pod's user and group IDs, the SELinux label, and more shown in the **`oc get pod <name> -o yaml`** command:

```
# oc get pod gluster-pod1 -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
```

```
    openshift.io/scc: privileged  (1)
  creationTimestamp: 2016-08-24T16:37:45Z
  labels:
    name: gluster-pod1
  name: gluster-pod1
  namespace: default  (2)
  resourceVersion: "482"
  selfLink: /api/v1/namespaces/default/pods/gluster-pod1
  uid: 15afda77-6a19-11e6-aadb-525400f7256d
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: gluster-pod1
    ports:
    - containerPort: 80
      name: web
      protocol: TCP
    resources: {}
    securityContext:
      privileged: true  (3)
    terminationMessagePath: /dev/termination-log
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: glustervol
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-1n70u
      readOnly: true
  dnsPolicy: ClusterFirst
  host: ose1.rhs
  imagePullSecrets:
  - name: default-dockercfg-20xg9
  nodeName: ose1.rhs
  restartPolicy: Always
  securityContext:
    supplementalGroups:
    - 590  (4)
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes:
  - name: glustervol
    persistentVolumeClaim:
      claimName: gluster-claim  (5)
  - name: default-token-1n70u
    secret:
      secretName: default-token-1n70u
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-08-24T16:37:45Z
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: 2016-08-24T16:37:53Z
```

```
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: 2016-08-24T16:37:45Z
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID:
docker://e67ed01729e1dc7369c5112d07531a27a7a02a7eb942f17d1c5fce32d8c31a
2d
    image: nginx
    imageID:
docker://sha256:4efb2fcdb1ab05fb03c9435234343c1cc65289eeb016be86193e88d
3a5d84f6b
    lastState: {}
    name: gluster-pod1
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-08-24T16:37:52Z
  hostIP: 192.168.122.251
  phase: Running
  podIP: 172.17.0.2
  startTime: 2016-08-24T16:37:45Z
```

1

The SCC used by the pod.

2

The project (namespace) name.

3

The security context level requested, in this case privileged

4

The supplemental group ID for the pod (all containers).

5

The PVC name used by the pod.

22.5. COMPLETE EXAMPLE OF DYNAMIC PROVISIONING USING GLUSTERFS

Note

This example assumes a working OpenShift Container Platform installed and functioning along with Heketi and GlusterFS.

All **oc** commands are executed on the OpenShift Container Platform master host.

22.5.1. Overview

This topic provides an end-to-end example of how to dynamically provision GlusterFS volumes. In this example, a simple NGINX HelloWorld application is deployed using the Red Hat Container Native Storage (CNS) solution. CNS hyper-converges GlusterFS storage by containerizing it into the OpenShift Container Platform cluster.

The Red Hat Gluster Storage Administration Guide can also provide additional information about GlusterFS.

To get started, follow the gluster-kubernetes quickstart guide for an easy Vagrant-based installation and deployment of a working OpenShift Container Platform cluster with Heketi and GlusterFS containers.

22.5.2. Verify the Environment and Gather Needed Information

Note

At this point, there should be a working OpenShift Container Platform cluster deployed, and a working Heketi server with GlusterFS.

1. Verify and view the cluster environment, including nodes and pods:

```
$ oc get nodes,pods
NAME      STATUS    AGE
master    Ready     22h
node0     Ready     22h
node1     Ready     22h
node2     Ready     22h
NAME                                  READY     STATUS
RESTARTS   AGE           1/1       Running             0
1d
glusterfs-node0-2509304327-vpce1   1/1       Running   0
1d        192.168.10.100   node0
glusterfs-node1-3290690057-hhq92   1/1       Running   0
1d        192.168.10.101   node1 (1)
glusterfs-node2-4072075787-okzjv   1/1       Running   0
1d        192.168.10.102   node2
heketi-3017632314-yyngh            1/1       Running   0
1d        10.42.0.0        node0 (2)
```

(1)

Example of GlusterFS storage pods running. There are three in this example.

2

Heketi server pod.

2. If not already set in the environment, export the **HEKETI_CLI_SERVER**:

```
$ export HEKETI_CLI_SERVER=$(oc describe svc/heketi | grep
"Endpoints:" | awk '{print "http://"$2}')
```

3. Identify the Heketi REST URL and server IP address:

```
$ echo $HEKETI_CLI_SERVER
http://10.42.0.0:8080
```

4. Identify the Gluster endpoints that are needed to pass in as a parameter into the storage class, which is used in a later step (**heketi-storage-endpoints**).

```
$ oc get endpoints
NAME                      ENDPOINTS
AGE
heketi                    10.42.0.0:8080
22h
heketi-storage-endpoints
192.168.10.100:1,192.168.10.101:1,192.168.10.102:1   22h 1
kubernetes                192.168.10.90:6443
23h
```

1

The defined GlusterFS endpoints. In this example, they are called **heketi-storage-endpoints**.

Note

By default, **user_authorization** is disabled. If enabled, you may need to find the rest user and rest user secret key. (This is not applicable for this example, as any values will work).

22.5.3. Create a Storage Class for Your GlusterFS Dynamic Provisioner

Storage classes manage and enable persistent storage in OpenShift Container Platform. Below is an example of a *Storage class* requesting 5GB of on-demand storage to be used with your *HelloWorld* application.

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
```

```
  name: gluster-heketi  1
provisioner: kubernetes.io/glusterfs  2
 parameters:
  endpoint: "heketi-storage-endpoints"  3
  resturl: "http://10.42.0.0:8080"  4
  restuser: "joe"  5
  restuserkey: "My Secret Life"  6
```

1

Name of the storage class.

2

The provisioner.

3

The GlusterFS-defined endpoint (**`oc get endpoints`**).

4

Heketi REST URL, taken from Step 1 above (**`echo $HEKETI_CLI_SERVER`**).

5

Rest username. This can be any value since authorization is turned off.

6

Rest user key. This can be any value.

1. Create the Storage Class YAML file, save it, then submit it to OpenShift Container Platform:

```
$ oc create -f gluster-storage-class.yaml
storageclass "gluster-heketi" created
```

2. View the storage class:

```
$ oc get storageclass
NAME              TYPE
gluster-heketi    kubernetes.io/glusterfs
```

22.5.4. Create a PVC to Request Storage for Your Application

1. Create a persistent volume claim (PVC) requesting 5GB of storage.

   During that time, the Dynamic Provisioning Framework and Heketi will automatically provision a new GlusterFS volume and generate the persistent volume (PV) object:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster1
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-heketi  1
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 5Gi  2
```

1

   The Kubernetes storage class annotation and the name of the storage class.

2

   The amount of storage requested.

1. Create the PVC YAML file, save it, then submit it to OpenShift Container Platform:

```
$ oc create -f gluster-pvc.yaml
persistentvolumeclaim "gluster1" created
```

2. View the PVC:

```
$ oc get pvc
NAME       STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
gluster1   Bound     pvc-7d37c7bd-bb5b-11e6-b81e-525400d87180
5Gi        RWO           14h
```

   Notice that the PVC is bound to a dynamically created volume.

3. View the persistent volume (PV):

```
$ oc get pv
NAME                                       CAPACITY
ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM
REASON    AGE
pvc-7d37c7bd-bb5b-11e6-b81e-525400d87180   5Gi        RWO
Delete          Bound     default/gluster1             14h
```

23.5.5. Create a NGINX Pod That Uses the PVC

22.5.5. Create a NGINX Pod That Uses the PVC

At this point, you have a dynamically created GlusterFS volume, bound to a PVC. Now, you can use this claim in a pod. Create a simple NGINX pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    name: nginx-pod
spec:
  containers:
  - name: nginx-pod
    image: gcr.io/google_containers/nginx-slim:0.8
    ports:
    - name: web
      containerPort: 80
    securityContext:
      privileged: true
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 1
```

1

The name of the PVC created in the previous step.

1. Create the Pod YAML file, save it, then submit it to OpenShift Container Platform:

```
$ oc create -f nginx-pod.yaml
pod "gluster-pod1" created
```

2. View the pod:

```
$ oc get pods -o wide
NAME                                READY     STATUS    RESTARTS
AGE       IP               NODE
nginx-pod                           1/1       Running   0
9m        10.38.0.0        node1
glusterfs-node0-2509304327-vpce1    1/1       Running   0
1d        192.168.10.100   node0
glusterfs-node1-3290690057-hhq92    1/1       Running   0
1d        192.168.10.101   node1
glusterfs-node2-4072075787-okzjv    1/1       Running   0
1d        192.168.10.102   node2
heketi-3017632314-yyngh             1/1       Running   0
1d        10.42.0.0        node0
```

Note

This may take a few minutes, as the the pod may need to download the image if it does not already exist.

3. **`oc exec`** into the container and create an ***index.html*** file in the **`mountPath`** definition of the pod:

```
$ oc exec -ti nginx-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello World from GlusterFS!!!' > index.html
$ ls
index.html
$ exit
```

4. Using the **`curl`** command from the master node, **`curl`** the URL of the pod:

```
$ curl http://10.38.0.0
Hello World from GlusterFS!!!
```

5. Check your Gluster pod to ensure that the ***index.html*** file was written. Choose any of the Gluster pods:

```
$ oc exec -ti glusterfs-node1-3290690057-hhq92 /bin/sh
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_
1e730a5462c352835055018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512
,swidth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_
d8c06e606ff4cc29ccb9d018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512
,swidth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_
d8c06e606ff4cc29ccb9d018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello World from GlusterFS!!!
```

22.6. MOUNTING VOLUMES ON PRIVILEGED PODS

22.6.1. Overview

Persistent volumes can be mounted to pods with the **privileged** security context constraint (SCC) attached.

Note

While this topic uses GlusterFS as a sample use-case for mounting volumes onto privileged pods, it can be adapted to use any supported storage plug-in.

22.6.2. Prerequisites

- An existing Gluster volume.
- **glusterfs-fuse** installed on all hosts.
- Definitions for GlusterFS:
  - Endpoints and services: ***gluster-endpoints-service.yaml*** and ***gluster-endpoints.yaml***
  - Persistent volumes: ***gluster-pv.yaml***
  - Persistent volume claims: ***gluster-pvc.yaml***
  - Privileged pods: ***gluster-nginx-pod.yaml***
- A user with the **cluster-admin** role binding. For this guide, that user is called **admin**.

22.6.3. Creating the Persistent Volume

Creating the **PersistentVolume** makes the storage accessible to users, regardless of projects.

1. As the admin, create the service, endpoint object, and persistent volume:

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
```

2. Verify that the objects were created:

```
$ oc get svc
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)
SELECTOR   AGE
gluster-cluster   172.30.151.58   <none>        1/TCP     <none>
24s
```

```
$ oc get ep
NAME              ENDPOINTS                               AGE
gluster-cluster   192.168.59.102:1,192.168.59.103:1   2m
```

```
$ oc get pv
NAME                     LABELS    CAPACITY   ACCESSMODES
STATUS      CLAIM     REASON    AGE
gluster-default-volume   <none>    2Gi        RWX
Available                       2d
```

22.6.4. Creating a Regular User

Adding a regular user to the **privileged** SCC (or to a group given access to the SCC) allows them to run **privileged** pods:

1. As the admin, add a user to the SCC:

```
$ oadm policy add-scc-to-user privileged <username>
```

1. Log in as the regular user:

```
$ oc login -u <username> -p <password>
```

1. Then, create a new project:

```
$ oc new-project <project_name>
```

22.6.5. Creating the Persistent Volume Claim

1. As a regular user, create the **PersistentVolumeClaim** to access the volume:

```
$ oc create -f gluster-pvc.yaml -n <project_name>
```

2. Define your pod to access the claim:

Example 22.15. Pod Definition

```
apiVersion: v1
id: gluster-nginx-pvc
kind: Pod
metadata:
  name: gluster-nginx-priv
spec:
  containers:
    - name: gluster-nginx-priv
      image: fedora/nginx
      volumeMounts:
        - mountPath: /mnt/gluster  1
          name: gluster-volume-claim
      securityContext:
        privileged: true
  volumes:
    - name: gluster-volume-claim
      persistentVolumeClaim:
        claimName: gluster-claim  2
```

1

Volume mount within the pod.

2

The **gluster-claim** must reflect the name of the **PersistentVolume**.

3. Upon pod creation, the mount directory is created and the volume is attached to that mount point.

   As regular user, create a pod from the definition:

```
$ oc create -f gluster-nginx-pod.yaml
```

4. Verify that the pod created successfully:

```
$ oc get pods
NAME                 READY     STATUS    RESTARTS   AGE
gluster-nginx-pod    1/1       Running   0          36m
```

   It can take several minutes for the pod to create.

22.6.6. Verifying the Setup

22.6.6.1. Checking the Pod SCC

1. Export the pod configuration:

```
$ oc export pod <pod_name>
```

2. Examine the output. Check that **`openshift.io/scc`** has the value of **`privileged`**:

Example 22.16. Export Snippet

```
metadata:
  annotations:
    openshift.io/scc: privileged
```

22.6.6.2. Verifying the Mount

1. Access the pod and check that the volume is mounted:

```
$ oc rsh <pod_name>
[root@gluster-nginx-pvc /]# mount
```

2. Examine the output for the Gluster volume:

Example 22.17. Volume Mount

```
192.168.59.102:gv0 on /mnt/gluster type fuse.gluster
(rw,relatime,user_id=0,group_id=0,default_permissions,allow_oth
er,max_read=131072)
```

22.7. BACKING DOCKER REGISTRY WITH GLUSTERFS STORAGE

22.7.1. Overview

This topic reviews how to attach a GlusterFS persistent volume to the Docker Registry.

It is assumed that the Docker registry service has already been started and the Gluster volume has been created.

22.7.2. Prerequisites

- The docker-registry was deployed **without** configuring storage.
- A Gluster volume exists and **glusterfs-fuse** is installed on schedulable nodes.
- Definitions written for GlusterFS endpoints and service, persistent volume (PV), and persistent volume claim (PVC).
  - For this guide, these will be:
    - ***gluster-endpoints-service.yaml***
    - ***gluster-endpoints.yaml***
    - ***gluster-pv.yaml***
    - ***gluster-pvc.yaml***
- A user with the **cluster-admin** role binding.
  - For this guide, that user is **admin**.

Note

All **oc** commands are executed on the master node as the **admin** user.

22.7.3. Create the Gluster Persistent Volume

First, make the Gluster volume available to the registry.

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
$ oc create -f gluster-pvc.yaml
```

Check to make sure the PV and PVC were created and bound successfully. The expected output should resemble the following. Note that the PVC status is **Bound**, indicating that it has bound to the PV.

```
$ oc get pv
NAME         LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-pv   <none>    1Gi        RWX           Available
37s
$ oc get pvc
NAME            LABELS    STATUS    VOLUME       CAPACITY
ACCESSMODES   AGE
gluster-claim   <none>    Bound     gluster-pv   1Gi        RWX
24s
```

Note

If either the PVC or PV failed to create or the PVC failed to bind, refer back to the GlusterFS Persistent Storage guide. **Do not** proceed until they initialize and the PVC status is **Bound**.

22.7.4. Attach the PVC to the Docker Registry

Before moving forward, ensure that the **docker-registry** service is running.

```
$ oc get svc
NAME              CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR                  AGE
docker-registry   172.30.167.194   <none>        5000/TCP
docker-registry=default   18m
```

Note

If either the **docker-registry** service or its associated pod is not running, refer back to the **docker-registry** setup instructions for troubleshooting before continuing.

Then, attach the PVC:

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

Deploying a Docker Registry provides more information on using the Docker registry.

22.7.5. Known Issues

22.7.5.1. Pod Cannot Resolve the Volume Host

In non-production cases where the **dnsmasq** server is located on the same node as the OpenShift Container Platform master service, pods might not resolve to the host machines when mounting the volume, causing errors in the **docker-registry-1-deploy** pod. This can happen when

dnsmasq.service fails to start because of a collision with OpenShift Container Platform DNS on port 53. To run the DNS server on the master host, some configurations needs to be changed.

In ***/etc/dnsmasq.conf***, add:

```
# Reverse DNS record for master
host-record=master.example.com,<master-IP>
# Wildcard DNS for OpenShift Applications - Points to Router
address=/apps.example.com/<master-IP>
# Forward .local queries to SkyDNS
server=/local/127.0.0.1#8053
# Forward reverse queries for service network to SkyDNS.
# This is for default OpenShift SDN - change as needed.
server=/17.30.172.in-addr.arpa/127.0.0.1#8053
```

With these settings, **dnsmasq** will pull from the ***/etc/hosts*** file on the master node.

Add the appropriate host names and IPs for all necessary hosts.

In ***master-config.yaml***, change **`bindAddress`** to:

```
dnsConfig:
 bindAddress: 127.0.0.1:8053
```

When pods are created, they receive a copy of ***/etc/resolv.conf***, which typically contains only the master DNS server so they can resolve external DNS requests. To enable internal DNS resolution, insert the **dnsmasq** server at the top of the server list. This way, **dnsmasq** will attempt to resolve requests internally first.

In ***/etc/resolv.conf*** all scheduled nodes:

```
nameserver 192.168.1.100  1
nameserver 192.168.1.1    2
```

1

Add the internal DNS server.

2

Pre-existing external DNS server.

Once the configurations are changed, restart the OpenShift Container Platform master and **dnsmasq** services.

```
$ systemctl restart atomic-openshift-master
$ systemctl restart dnsmasq
```

22.8. BINDING PERSISTENT VOLUMES BY LABELS

22.8.1. Overview

This topic provides an end-to-end example for binding persistent volume claims (PVCs) to persistent volumes (PVs), by defining labels in the PV and matching selectors in the PVC. This feature is available for all storage options. It is assumed that a OpenShift Container Platform cluster contains persistent storage resources which are available for binding by PVCs.

A Note on Labels and Selectors

Labels are an OpenShift Container Platform feature that support user-defined tags (key-value pairs) as part of an object's specification. Their primary purpose is to enable the arbitrary grouping of objects by defining identical labels among them. These labels can then be targeted by selectors to match all objects with specified label values. It is this functionality we will take advantage of to enable our PVC to bind to our PV. For a more in-depth look at labels, see Pods and Services.

Note

For this example, we will be using modified GlusterFS PV and PVC specifications. However, implementation of selectors and labels is generic across for all storage options. See the relevant storage option for your volume provider to learn more about its unique configuration.

22.8.1.1. Assumptions

It is assumed that you have:

- An existing OpenShift Container Platform cluster with at least one **master** and one **node**
- At least one supported storage volume
- A user with **cluster-admin** privileges

22.8.2. Defining Specifications

Note

These specifications are tailored to **GlusterFS**. Consult the relevant storage option for your volume provider to learn more about its unique configuration.

22.8.2.1. Persistent Volume with Labels

Example 22.18. glusterfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: 1
    storage-tier: gold
    aws-availability-zone: us-east-1
```

```
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster 2
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1

Use labels to identify common attributes or characteristics shared among volumes. In this case, we defined the Gluster volume to have a custom attribute (key) named **storage-tier** with a value of **gold** assigned. A claim will be able to select a PV with **`storage-tier=gold`** to match this PV.

2

Endpoints define the Gluster trusted pool and are discussed below.

22.8.2.2. Persistent Volume Claim with Selectors

A claim with a **selector** stanza (see example below) attempts to match existing, unclaimed, and non-prebound PVs. The existence of a PVC selector ignores a PV's capacity. However, **accessModes** are still considered in the matching criteria.

It is important to note that a claim must match **all** of the key-value pairs included in its **selector** stanza. If no PV matches the claim, then the PVC will remain unbound (Pending). A PV can subsequently be created and the claim will automatically check for a label match.

Example 22.19. glusterfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
  selector: 1
    matchLabels:
      storage-tier: gold
      aws-availability-zone: us-east-1
```

1

The **selector** stanza defines all labels necessary in a PV in order to match this claim.

22.8.2.3. Volume Endpoints

To attach the PV to the Gluster volume, endpoints should be configured before creating our objects.

Example 22.20. glusterfs-ep.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
      - ip: 192.168.122.221
    ports:
      - port: 1
  - addresses:
      - ip: 192.168.122.222
    ports:
      - port: 1
```

22.8.2.4. Deploy the PV, PVC, and Endpoints

For this example, run the **oc** commands as a **cluster-admin** privileged user. In a production environment, cluster clients might be expected to define and create the PVC.

```
# oc create -f glusterfs-ep.yaml
endpoints "glusterfs-cluster" created
# oc create -f glusterfs-pv.yaml
persistentvolume "gluster-volume" created
# oc create -f glusterfs-pvc.yaml
persistentvolume "gluster-volume" created
```

Lastly, confirm that the PV and PVC bound successfully.

```
# oc get pv,pvc
NAME             CAPACITY   ACCESSMODES      STATUS     CLAIM
REASON    AGE
gluster-volume    2Gi        RWX              Bound      gfs-
trial/gluster-claim            7s
NAME             STATUS     VOLUME           CAPACITY   ACCESSMODES
AGE
gluster-claim     Bound      gluster-volume   2Gi        RWX
7s
```

Note

PVCs are local to a project, whereas PVs are a cluster-wide, global resource. Developers and non-administrator users may not have access to see all (or any) of the available PVs.

22.9. USING STORAGE CLASSES FOR DYNAMIC PROVISIONING

22.9.1. Overview

In these examples we will walk through a few scenarios of various configuratons of StorageClasses and Dynamic Provisioning using Google Cloud Platform Compute Engine (GCE). These examples assume some familiarity with Kubernetes, GCE and Persistent Disks and OpenShift Container Platform is installed and properly configured to use GCE.

- Basic Dynamic Provisioning
- Defaulting Cluster Dynamic Provisioning Behavior

22.9.2. Scenario 1: Basic Dynamic Provisioning with Two Types of *StorageClasses*

StorageClasses can be used to differentiate and delineate storage levels and usages. In this case, the **`cluster-admin`** or **`storage-admin`** sets up two distinct classes of storage in GCE.

- **`slow`**: Cheap, efficient, and optimized for sequential data operations (slower reading and writing)
- **`fast`**: Optimized for higher rates of random IOPS and sustained throughput (faster reading and writing)

By creating these *StorageClasses*, the **`cluster-admin`** or **`storage-admin`** allows users to create claims requesting a particular level or service of *StorageClass*.

Example 22.21. StorageClass Slow Object Definitions

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow 1
provisioner: kubernetes.io/gce-pd 2
parameters:
  type: pd-standard 3
  zone: us-east1-d  4
```

1

Name of the *StorageClass*.

2

The provisioner plug-in to be used. This is a required field for *StorageClasses*.

3

PD type. This example uses **pd-standard**, which has a slightly lower cost, rate of sustained IOPS, and throughput versus **pd-ssd**, which carries more sustained IOPS and throughput.

4

The zone is required.

Example 22.22. StorageClass Fast Object Definition

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-east1-d
```

As a **cluster-admin** or **storage-admin**, save both definitions as YAML files. For example, **slow-gce.yaml** and **fast-gce.yaml**. Then create the *StorageClasses*.

```
# oc create -f slow-gce.yaml
storageclass "slow" created

# oc create -f fast-gce.yaml
storageclass "fast" created

# oc get storageclass
NAME       TYPE
fast       kubernetes.io/gce-pd
slow       kubernetes.io/gce-pd
```

Important

cluster-admin or **storage-admin** users are responsible for relaying the correct *StorageClass* name to the correct users, groups, and projects.

As a regular user, create a new project:

```
# oc new-project rh-eng
```

Create the claim YAML definition, save it to a file (**pvc-fast.yaml**):

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering
 annotations:
   volume.beta.kubernetes.io/storage-class: fast
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 10Gi
```

Add the claim with the **oc create** command:

```
# oc create -f pvc-fast.yaml
persistentvolumeclaim "pvc-engineering" created
```

Check to see if your claim is bound:

```
# oc get pvc
NAME              STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering   Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           2m
```

Important

Since this claim was created and bound in the *rh-eng* project, it can be shared by any user in the same project.

As a **cluster-admin** or **storage-admin** user, view the recent dynamically provisioned Persistent Volume (PV).

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS    CLAIM                        REASON    AGE
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound     rh-eng/pvc-engineering2              5m
```

Important

Notice the RECLAIMPOLICY is *Delete* by default for all dynamically provisioned volumes. This means the volume only lasts as long as the claim still exists in the system. If you delete the claim, the volume is also deleted and all data on the volume is lost.

Finally, check the GCE console. The new disk has been created and is ready for use.

```
kubernetes-dynamic-pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  SSD
persistent disk  10 GB  us-east1-d
```

Pods can now reference the persistent volume claim and start using the volume.

22.9.3. Scenario 2: How to enable Default *StorageClass* behavior for a Cluster

In this example, a **`cluster-admin`** or **`storage-admin`** enables a *default* storage class for all other users and projects that do not implicitly specify a *StorageClass* annotation in their claim. This is useful for a **`cluster-admin`** or **`storage-admin`** to provide easy management of a storage volume without having to set up or communicate specialized *StorageClasses* across the cluster.

This example builds upon Section 22.9.2, “Scenario 1: Basic Dynamic Provisioning with Two Types of *StorageClasses*”. The **`cluster-admin`** or **`storage-admin`** will create another *StorageClass* for designation as the *defaultStorageClass*.

Example 22.23. Default StorageClass Object Definition

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: generic 1
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true" 2
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zone: us-east1-d
```

1 Name of the *StorageClass*, which needs to be unique in the cluster.

2 Annotation that marks this *StorageClass* as the default class. You must use **`"true"`** quoted in this version of the API. Without this annotation, OpenShift Container Platform considers this not the *default StorageClass*.

As a **`cluster-admin`** or **`storage-admin`** save the definition to a YAML file (**`generic-gce.yaml`**), then create the *StorageClasses*:

```
# oc create -f generic-gce.yaml
storageclass "generic" created
```

```
# oc get storageclass
NAME       TYPE
generic    kubernetes.io/gce-pd
fast       kubernetes.io/gce-pd
slow       kubernetes.io/gce-pd
```

As a regular user, create a new claim definition without any *StorageClass* annotation and save it to a file (**`generic-pvc.yaml`**).

Example 22.24. *default* Storage Claim Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering2
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 5Gi
```

Execute it and check the claim is bound:

```
# oc create -f generic-pvc.yaml
persistentvolumeclaim "pvc-engineering2" created

3s
# oc get pvc
NAME               STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering    Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           41m
pvc-engineering2   Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004
5Gi        RWX           7s  (1)
```

(1)

`pvc-engineering2` is bound to a dynamically provisioned Volume by *default*.

As a **`cluster-admin`** or **`storage-admin`**, view the Persistent Volumes defined so far:

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS    CLAIM                      REASON    AGE
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound     rh-eng/pvc-engineering2              5m  (1)
```

```
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound     mytest/gce-dyn-claim1                21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound     rh-eng/pvc-engineering               46m  2
```

1

This PV was bound to our *default* dynamic volume from the *default StorageClass*.

2

This PV was bound to our first PVC from Section 22.9.2, “Scenario 1: Basic Dynamic Provisioning with Two Types of *StorageClasses*” with our *fast StorageClass*.

Create a manually provisioned disk using GCE (not dynamically provisioned). Then create a Persistent Volume that connects to the new GCE disk (**`pv-manual-gce.yaml`**).

Example 22.25. Manual PV Object Defition

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-newly-created-gce-PD
   fsType: ext4
```

Execute the object definition file:

```
# oc create -f pv-manual-gce.yaml
```

Now view the PVs again. Notice that a **`pv-manual-gce`** volume is *Available*.

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS      CLAIM                      REASON    AGE
pv-manual-gce                              35Gi       RWX
Retain          Available                                       4s
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound       rh-eng/pvc-engineering2            12m
```

```
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound       mytest/gce-dyn-claim1               21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound       rh-eng/pvc-engineering              53m
```

Now create another claim identical to the **`generic-pvc.yaml`** PVC definition but change the name and do not set an annotation.

Example 22.26. Claim Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering3
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 15Gi
```

Because *default StorageClass* is enabled in this instance, the manually created PV does not satisfy the claim request. The user receives a new dynamically provisioned Persistent Volume.

```
# oc get pvc
NAME                STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering     Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           1h
pvc-engineering2    Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004
5Gi        RWX           19m
pvc-engineering3    Bound     pvc-6fa8e73b-8c00-11e6-9962-42010af00004
15Gi       RWX           6s
```

Important

Since the *default StorageClass* is enabled on this system, for the manually created Persistent Volume to get bound by the above claim and not have a new dynamic provisioned volume be bound, the PV would need to have been created in the *default StorageClass*.

Since the *default StorageClass* is enabled on this system, you would need to create the PV in the *default StorageClass* for the manually created Persistent Volume to get bound to the above claim and not have a new dynamic provisioned volume bound to the claim.

To fix this, the **`cluster-admin`** or **`storage-admin`** user simply needs to create another GCE disk or delete the first manual PV and use a PV object definition that assigns a *StorageClass* annotation (**`pv-manual-gce2.yaml`**) if necessary:

Example 22.27. Manual PV Spec with *default* StorageClass annotation

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce2
 annotations:
   volume.beta.kubernetes.io/storage-class: generic 1
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-newly-created-gce-PD
   fsType: ext4
```

1 The annotation for previously created *generic StorageClass*.

Execute the object definition file:

```
# oc create -f pv-manual-gce2.yaml
```

List the PVs:

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS      CLAIM                      REASON    AGE
pv-manual-gce                              35Gi       RWX
Retain          Available                                        4s
1
pv-manual-gce2                             35Gi       RWX
Retain          Bound       rh-eng/pvc-engineering3              4s
2
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound       rh-eng/pvc-engineering2              12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound       mytest/gce-dyn-claim1                21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound       rh-eng/pvc-engineering               53m
```

1 The original manual PV, still unbound and Available. This is because it was not created in the *default StorageClass*.

the default StorageClass.

2

The second PVC (other than the name) is bound to the Available manually created PV **pv-manual-gce2**.

Important

Notice that all dynamically provisioned volumes by default have a *RECLAIMPOLICY* of *Delete*. Once the PVC dynamically bound to the PV is deleted, the GCE volume is deleted and all data is lost. However, the manually created PV has a default *RECLAIMPOLICY* of *Retain*.

22.10. USING STORAGE CLASSES FOR EXISTING LEGACY STORAGE

22.10.1. Overview

In this example, a legacy data volume exists and a **cluster-admin** or **storage-admin** needs to make it available for consumption in a particular project. Using *StorageClasses* decreases the likelihood of other users and projects gaining access to this volume from a claim because the claim would have to have an exact matching value for the *StorageClass* annotation. This example also disables dynamic provisioning. This example assumes:

- Some familiarity with OpenShift Container Platform, GCE, and Persistent Disks
- OpenShift Container Platform is properly configured to use GCE.

22.10.1.1. Scenario 1: Link StorageClass to existing Persistent Volume with Legacy Data

As a **cluster-admin** or **storage-admin**, define and create the *StorageClass* for historical financial data.

Example 22.28. StorageClass finance-history Object Definitions

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: finance-history 1
provisioner: no-provisioning 2
parameters: 3
```

1

Name of the StorageClass.

2

This is a required field, but since there is to be no dynamic provisioning, a value must be put here as long as it is not an actual provisioner plug-in type.

3

Parameters can simply be left blank, since these are only used for the dynamic provisioner.

Save the definitions to a YAML file (**`finance-history-storageclass.yaml`**) and create the *StorageClass*.

```
# oc create -f finance-history-storageclass.yaml
storageclass "finance-history" created


# oc get storageclass
NAME              TYPE
finance-history   no-provisioning
```

Important

`cluster-admin` or **`storage-admin`** users are responsible for relaying the correct *StorageClass* name to the correct users, groups, and projects.

The *StorageClass* exists. A **`cluster-admin`** or **`storage-admin`** can create the Persistent Volume (PV) for use with the *StorageClass*. Create a manually provisioned disk using GCE (not dynamically provisioned) and a Persistent Volume that connects to the new GCE disk (**`gce-pv.yaml`**).

Example 22.29. Finance History PV Object

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-finance-history
 annotations:
   volume.beta.kubernetes.io/storage-class: finance-history 1
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
```

```
    readOnly: false
    pdName: the-existing-PD-volume-name-that-contains-the-valuable-data
  2
    fsType: ext4
```

1

The *StorageClass* annotation, that must match exactly.

2

The name of the GCE disk that already exists and contains the legacy data.

As a **`cluster-admin`** or **`storage-admin`**, create and view the PV.

```
# oc create -f gce-pv.yaml
persistentvolume "pv-finance-history" created

# oc get pv
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                       REASON    AGE
pv-finance-history   35Gi       RWX           Retain
Available                                        2d
```

Notice you have a **`pv-finance-history`** Available and ready for consumption.

As a user, create a Persistent Volume Claim (PVC) as a YAML file and specify the correct *StorageClass* annotation:

Example 22.30. Claim for finance-history Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-finance-history
 annotations:
   volume.beta.kubernetes.io/storage-class: finance-history  1
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 20Gi
```

1

The *StorageClass* annotation, that must match exactly or the claim will go unbound until it is deleted or another *StorageClass* is created that matches the annotation.

it is deleted or another *StorageClass* is created that matches the annotation.

Create and view the PVC and PV to see if it is bound.

```
# oc create -f pvc-finance-history.yaml
persistentvolumeclaim "pvc-finance-history" created

# oc get pvc
NAME                  STATUS    VOLUME               CAPACITY
ACCESSMODES   AGE
pvc-finance-history   Bound     pv-finance-history   35Gi       RWX
9m


# oc get pv  (cluster/storage-admin)
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                          REASON    AGE
pv-finance-history   35Gi       RWX           Retain          Bound
default/pvc-finance-history              5m
```

Important

You can use *StorageClasses* in the same cluster for both legacy data (no dynamic provisioning) and with dynamic provisioning.

CHAPTER 23. WORKING WITH HTTP PROXIES

23.1. OVERVIEW

Production environments can deny direct access to the Internet and instead have an HTTP or HTTPS proxy available. Configuring OpenShift Container Platform to use these proxies can be as simple as setting standard environment variables in configuration or JSON files. This can be done during an advanced installation or configured after installation.

The proxy configuration must be the same on each host in the cluster. Therefore, when setting up the proxy or modifying it, you must update the files on each OpenShift Container Platform host to the same values. Then, you must restart OpenShift Container Platform services on each host in the cluster.

The **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables are found in each host's ***/etc/sysconfig/atomic-openshift-master*** file (for single master configuration), ***/etc/sysconfig/atomic-openshift-master-api***, or ***/etc/sysconfig/atomic-openshift-master-controllers*** files (for multi-master configuration) and ***/etc/sysconfig/atomic-openshift-node***.

23.2. CONFIGURING NO_PROXY

The **NO_PROXY** environment variable lists all of the OpenShift Container Platform components and all IP addresses that are managed by OpenShift Container Platform.

NO_PROXY accepts a comma-separated list of hosts, IP addresses, or IP ranges in CIDR format:

For master hosts

- Node host name
- Master IP or host name

For node hosts

- Master IP or host name

For the Docker service

- Registry service IP and host name

NO_PROXY also includes the SDN network and service IP addresses as found in the ***master-config.yaml*** file.

/etc/origin/master/master-config.yaml

```
networkConfig:
  clusterNetworkCIDR: 10.1.0.0/16
  serviceNetworkCIDR: 172.30.0.0/16
```

OpenShift Container Platform does not accept * as a wildcard attached to a domain suffix. For example, this works:

```
NO_PROXY=.example.com
```

However, this does not:

```
NO_PROXY=*.example.com
```

The only wildcard **NO_PROXY** accepts is a single ***** character, which matches all hosts, and effectively disables the proxy.

Each name in this list is matched as either a domain which contains the host name as a suffix, or the host name itself.

For instance, **example.com** would match **example.com**, **example.com:80**, and **www.example.com**.

23.3. CONFIGURING HOSTS FOR PROXIES

1. Edit the proxy environment variables in the OpenShift Container Platform control files. Ensure all of the files in the cluster are correct.

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,10.1.0.0/16,172.30.0.0/16
1
```

1

Supports host names and CIDRs. Must include the SDN network and service IP ranges **10.1.0.0/16,172.30.0.0/16** by default.

2. Restart the master or node host as appropriate:

```
# systemctl restart atomic-openshift-master
# systemctl restart atomic-openshift-node
```

For multi-master installations:

```
# systemctl restart atomic-openshift-master-controllers
# systemctl restart atomic-openshift-master-api
```

23.4. CONFIGURING HOSTS FOR PROXIES USING ANSIBLE

During advanced installations, the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables can be configured using the **openshift_no_proxy**, **openshift_http_proxy**, and **openshift_https_proxy** parameters, which are configurable in the inventory file.

Example 23.1. Example Proxy Configuration with Ansible

```
# Global Proxy Configuration
# These options configure HTTP_PROXY, HTTPS_PROXY, and NOPROXY
```

```
environment
# variables for docker and master services.
openshift_http_proxy=http://<user>:<password>@<ip_addr>:<port>
openshift_https_proxy=https://<user>:<password>@<ip_addr>:<port>
openshift_no_proxy='.hosts.example.com,some-host.com'
#
# Most environments do not require a proxy between OpenShift
masters, nodes, and
# etcd hosts. So automatically add those host names to the
openshift_no_proxy list.
# If all of your hosts share a common domain you may wish to disable
this and
# specify that domain above.
# openshift_generate_no_proxy_hosts=True
```

Note

There are additional proxy settings that can be configured for builds using Ansible parameters. For example:

The **openshift_builddefaults_git_http_proxy** and **openshift_builddefaults_git_https_proxy** parameters allow you to use a proxy for Git cloning

The **openshift_builddefaults_http_proxy** and **openshift_builddefaults_https_proxy** parameters can make environment variables available to the Docker build strategy and Custom build strategy processes.

23.5. PROXYING DOCKER PULL

OpenShift Container Platform node hosts need to perform push and pull operations to Docker registries. If you have a registry that does not need a proxy for nodes to access, include the **NO_PROXY** parameter with:

- the registry's host name,
- the registry service's IP address, and
- the service name.

This blacklists that registry, leaving the external HTTP proxy as the only option.

1. Retrieve the registry service's IP address **docker_registy_ip** by running:

```
$ oc describe svc/docker-registry -n default

Name:   docker-registry
Namespace:  default
Labels:   docker-registry=default
Selector:  docker-registry=default
Type:   ClusterIP
IP:   172.30.163.183 (1)
```

```
Port:   5000-tcp 5000/TCP
Endpoints:  10.1.0.40:5000
Session Affinity: ClientIP
No events.
```

1

Registry service IP.

2. Edit the */etc/sysconfig/docker* file and add the **NO_PROXY** variables in shell format, replacing **<docker_registry_ip>** with the IP address from the previous step.

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,<docker_registry_ip>,docker-
registry.default.svc.cluster.local
```

3. Restart the Docker service:

```
# systemctl restart docker
```

23.6. CONFIGURING S2I BUILDS FOR PROXIES

S2I builds fetch dependencies from various locations. You can use a ***.s2i/environment*** file to specify simple shell variables and OpenShift Container Platform will react accordingly when seeing build images.

The following are the supported proxy environment variables with example values:

```
HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com
```

23.7. CONFIGURING DEFAULT TEMPLATES FOR PROXIES

The example templates available in OpenShift Container Platform by default do not include settings for HTTP proxies. For existing applications based on these templates, modify the **source** section of the application's build configuration and add proxy settings:

```
...
source:
  type: Git
  git:
    uri: https://github.com/openshift/ruby-hello-world
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
...
```

This is similar to the process for using proxies for Git cloning.

23.8. SETTING PROXY ENVIRONMENT VARIABLES IN PODS

You can set the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables in the **templates.spec.containers** stanza in a deployment configuration to pass proxy connection information. The same can be done for configuring a Pod's proxy at runtime:

```
...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://<user>:<password>@<ip_addr>:<port>"
...
```

You can also use the **oc set env** command to update an existing deployment configuration with a new environment variable:

```
$ oc set env dc/frontend HTTP_PROXY=http://<user>:<password>@<ip_addr>:
<port>
```

If you have a ConfigChange trigger set up in your OpenShift Container Platform instance, the changes happen automatically. Otherwise, manually redeploy your application for the changes to take effect.

23.9. GIT REPOSITORY ACCESS

If your Git repository can only be accessed using a proxy, you can define the proxy to use in the **source** section of the **BuildConfig**. You can configure both a HTTP and HTTPS proxy to use. Both fields are optional. Domains for which no proxying should be performed can also be specified via the **NoProxy** field.

Note

Your source URI must use the HTTP or HTTPS protocol for this to work.

```
source:
  type: Git
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```

Cluster administrators can also configure a global proxy for Git cloning using Ansible.

CHAPTER 24. CONFIGURING GLOBAL BUILD DEFAULTS AND OVERRIDES

24.1. OVERVIEW

Developers can define settings in specific build configurations within their projects, such as configuring a proxy for Git cloning. Rather than requiring developers to define certain settings in each of their build configurations, cluster administrators can use admission control plug-ins to configure global build defaults and overrides that automatically use these settings in any build.

The settings from these plug-ins are not set in the build configurations or builds themselves, but rather are only used during the build process. This allows administrators to change the global configuration at any time, and any builds that are re-run from existing build configurations or builds will get the new settings.

The **`BuildDefaults`** admission control plug-in allows administrators to set global defaults for settings such as the Git HTTP and HTTPS proxy, as well as default environment variables. These defaults do not overwrite values that have been configured for a specific build. However, if those values are not present on the build definition, they are set to the default value.

The **`BuildOverrides`** admission control plug-in allows administrators to override a setting in a build, regardless of the value stored in the build. It currently supports overriding the **`forcePull`** flag on a build strategy to enforce always refreshing the local image during a build by pulling the image from the registry. This ensures that a user can only build with an image that they are allowed to pull.

24.2. SETTING GLOBAL BUILD DEFAULTS

You can set global build defaults two ways:

- using Ansible and the advanced installation tool
- manually by modifying the ***master-config.yaml*** file

24.2.1. Configuring Global Build Defaults with Ansible

During advanced installations, the **`BuildDefaults`** plug-in can be configured using the following parameters, which are configurable in the inventory file:

- **`openshift_builddefaults_http_proxy`**
- **`openshift_builddefaults_https_proxy`**
- **`openshift_builddefaults_no_proxy`**
- **`openshift_builddefaults_git_http_proxy`**
- **`openshift_builddefaults_git_https_proxy`**

> **Example 24.1. Example Build Defaults Configuration with Ansible**
>
> ```
> # These options configure the BuildDefaults admission controller
> which injects
> ```

```
# environment variables into Builds. These values will default to
the global proxy
# config values. You only need to set these if they differ from the
global settings
# above. See BuildDefaults
# documentation at
https://docs.openshift.org/latest/admin_guide/build_defaults_override
s.html
#openshift_builddefaults_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_no_proxy=build_defaults
openshift_builddefaults_git_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_https_proxy=https://USER:PASSWORD@HOST:PO
RT
# Or you may optionally define your own serialized as json
#openshift_builddefaults_json='{"BuildDefaults":{"configuration":
{"apiVersion":"v1","env":
[{"name":"HTTP_PROXY","value":"http://proxy.example.com.redhat.com:31
28"},{"name":"NO_PROXY","value":"ose3-
master.example.com"}],"gitHTTPProxy":"http://proxy.example.com:3128",
"kind":"BuildDefaultsConfig"}}}'
```

Note

There are additional proxy settings that can be configured for builds using Ansible parameters. For example: - The **`openshift_builddefaults_git_http_proxy`** and **`openshift_builddefaults_git_https_proxy`** parameters allow you to use a proxy for git cloning - The **`openshift_builddefaults_http_proxy`** and **`openshift_builddefaults_https_proxy`** parameters can make environment variables available to the Docker build strategy and Custom build strategy processes.

24.2.2. Manually Setting Global Build Defaults

To configure the **`BuildDefaults`** plug-in, add a configuration for it in the ***/etc/origin/master/master-config.yaml*** file on masters:

```
admissionConfig:
  pluginConfig:
    BuildDefaults:
      configuration:
        apiVersion: v1
        kind: BuildDefaultsConfig
        gitHTTPProxy: http://my.proxy:8080 (1)
        gitHTTPSProxy: https://my.proxy:8443 (2)
        gitNoProxy: somedomain.com, otherdomain.com (3)
        env:
        - name: HTTP_PROXY (4)
          value: http://my.proxy:8080
        - name: HTTPS_PROXY (5)
          value: https://my.proxy:8443
```

```
      - name: BUILD_LOGLEVEL 6
        value: 4
      - name: CUSTOM_VAR 7
        value: custom_value
    imageLabels:
    - name: url 8
      value: https://containers.example.org
    - name: vendor
      value: ExampleCorp Ltd.
    nodeSelector: 9
      key1: value1
      key2: value2
    annotations: 10
      key1: value1
      key2: value2
    resources: 11
      requests:
        cpu: "100m"
        memory: "256Mi"
      limits:
        cpu: "100m"
        memory: "256Mi"
```

1

Sets the HTTP proxy to use when cloning source code from a Git repository.

2

Sets the HTTPS proxy to use when cloning source code from a Git repository.

3

Sets the list of domains for which proxying should not be performed.

4

Default environment variable that sets the HTTP proxy to use during the build. This can be used for downloading dependencies during the assemble and build phases.

5

Default environment variable that sets the HTTPS proxy to use during the build. This can be used for downloading dependencies during the assemble and build phases.

6

Default environment variable that sets the build log level during the build.

7

Additional default environment variable that will be added to every build.

8

Labels to be applied to every image built. These can be overridden in **`BuildConfig`**.

9

Build pods will only run on nodes with the **`key1=value2`** and **`key2=value2`** labels. Users can define a different set of **`nodeSelectors`** for their builds, causing these values to be ignored.

10

Build pods will have these annotations added to them.

11

Sets the default resources to the build pod if the **`BuildConfig`** does not have related resource defined.

Restart the master service for the changes to take effect:

```
# systemctl restart atomic-openshift-master
```

24.3. SETTING GLOBAL BUILD OVERRIDES

To configure the **`BuildOverrides`** plug-in, add a configuration for it in the ***/etc/origin/master/master-config.yaml*** file on masters:

```
admissionConfig:
  pluginConfig:
    BuildOverrides:
      configuration:
        apiVersion: v1
        kind: BuildOverridesConfig
        forcePull: true 1
        imageLabels:
        - name: distribution-scope 2
          value: private
        nodeSelector: 3
          key1: value1
```

```
      key2: value2
    annotations: 4
      key1: value1
      key2: value2
```

1

Force all builds to pull their builder image and any source images before starting the build.

2

Additional labels to be applied to every image built. Labels defined here take precedence over labels defined in **`BuildConfig`**.

3

Build pods will only run on nodes with the **`key1=value2`** and **`key2=value2`** labels. Users can define additional key/value labels to further constrain the set of nodes a build runs on, but the **node** must have at least these labels.

4

Build pods will have these annotations added to them.

Restart the master service for the changes to take effect:

```
# systemctl restart atomic-openshift-master
```

CHAPTER 25. CONFIGURING PIPELINE EXECUTION

25.1. OVERVIEW

The first time a user creates a build configuration using the Pipeline build strategy, OpenShift Container Platform looks for a template named **`jenkins-ephemeral`** in the **`openshift`** namespace and instantiates it within the user's project. The **`jenkins-ephemeral`** template that ships with OpenShift Container Platform creates, upon instantiation:

- a deployment configuration for Jenkins using the official OpenShift Container Platform Jenkins image
- a service and route for accessing the Jenkins deployment
- a new Jenkins service account
- rolebindings to grant the service account edit access to the project

Cluster administrators can control what is created by either modifying the content of the built-in template, or by editing the cluster configuration to direct the cluster to a different template location.

To modify the content of the default template:

```
$ oc edit template jenkins-ephemeral -n openshift
```

To use a different template, such as the **`jenkins-persistent`** template which uses persistent storage for Jenkins, add the following to your master configuration file:

```
jenkinsPipelineConfig:
  autoProvisionEnabled: true 1
  templateNamespace: openshift 2
  templateName: jenkins-pipeline 3
  serviceName: jenkins-pipeline-svc 4
  parameters: 5
    key1: value1
    key2: value2
```

1

Defaults to **`true`** if unspecified. If **`false`**, then no template will be instantiated.

2

Namespace containing the template to be instantiated.

3

Name of the template to be instantiated.

4

Name of the service to be created by the template upon instantiation.

5

Optional values to pass to the template during instantiation.

When a Pipeline build configuration is created, OpenShift Container Platform looks for a Service matching **`serviceName`**. This means **`serviceName`** must be chosen such that it is unique in the project. If no Service is found, OpenShift Container Platform instantiates the **`jenkinsPipelineConfig`** template. If this is not desirable (if you would like to use a Jenkins server external to OpenShift Container Platform, for example), there are a few things you can do, depending on who you are.

- If you are a cluster administrator, simply set **`autoProvisionEnabled`** to **`false`**. This will disable autoprovisioning across the cluster.
- If you are an unpriviledged user, a Service must be created for OpenShift Container Platform to use. The service name must match the cluster configuration value of **`serviceName`** in the **`jenkinsPipelineConfig`**. The default value is **`jenkins`**. If you are disabling autoprovisioning because you are running a Jenkins server outside your project, it is recommended that you point this new service to your existing Jenkins server. See: Integrating External Services

The latter option could also be used to disable autoprovisioning in select projects only.

CHAPTER 26. CONFIGURING ROUTING

26.1. OVERVIEW

After installing OpenShift Container Platform and deploying a router, you can modify the router to suit your needs using the examples in this topic.

26.2. CONFIGURING ROUTE TIMEOUTS

You can configure the default timeouts for an existing route when you have services in need of a low timeout, as required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

Using the **`oc annotate`** command, add the timeout to the route:

```
# oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout>
<time_unit>
```

For example, to set a route named **`myroute`** to a timeout of two seconds:

```
# oc annotate route myroute --overwrite
haproxy.router.opensfhit.io/timeout=2s
```

Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

26.3. CONFIGURING NATIVE CONTAINER ROUTING

This section describes how to set up container networking using existing switches and routers and the kernel networking stack in Linux. The setup requires that the network administrator or a script modifies the router or routers when new nodes are added to the cluster.

Note

The procedure outlined in this topic can be adapted to any type of router.

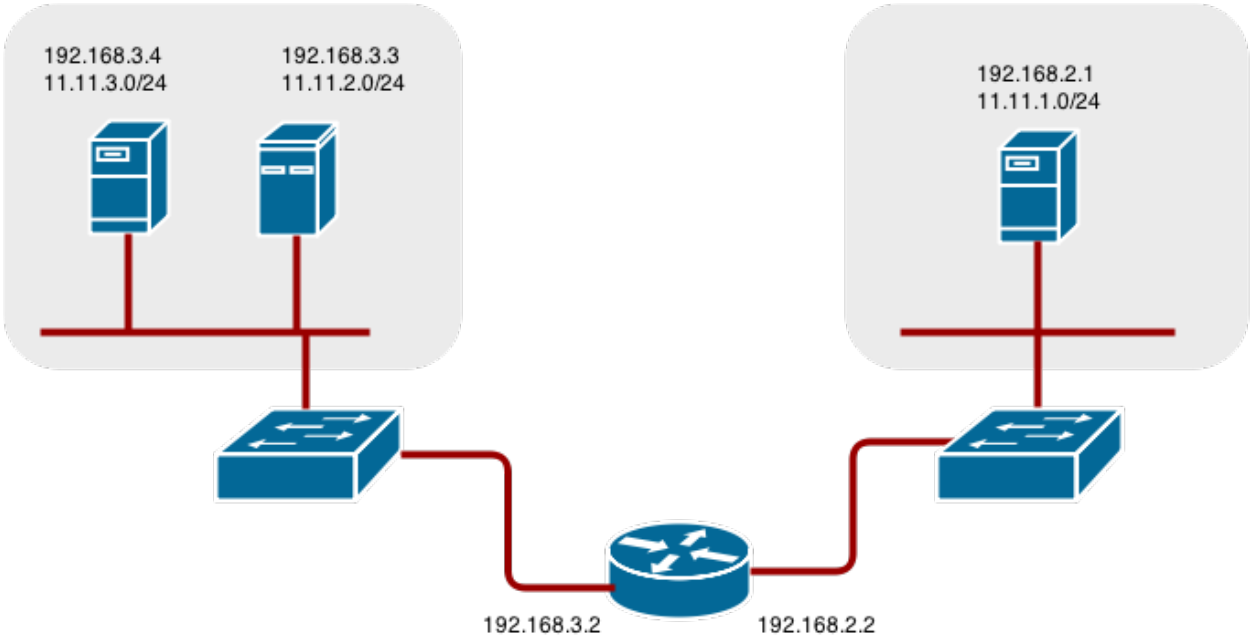
26.3.1. Network Overview

The following describes a general network setup:

- 11.11.0.0/16 is the container network.
- The 11.11.x.0/24 subnet is reserved for each node and assigned to the Docker Linux bridge.
- Each node has a route to the router for reaching anything in the 11.11.0.0/16 range, except the local subnet.
- The router has routes for each node, so it can be directed to the right node.

- Existing nodes do not need any changes when new nodes are added, unless the network topology is modified.
- IP forwarding is enabled on each node.

The following diagram shows the container networking setup described in this topic. It uses one Linux node with two network interface cards serving as a router, two switches, and three nodes connected to these switches.



Node setup

1. Assign an unused 11.11.x.0/24 subnet IP address to the Linux bridge on the node:

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

1. Modify the Docker startup script to use the new bridge. By default, the startup script is the **`/etc/sysconfig/docker`** file:

```
# docker -d -b lbr0 --other-options
```

2. Add a route to the router for the 11.11.0.0/16 network:

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

3. Enable IP forwarding on the node:

```
# sysctl -w net.ipv4.ip_forward=1
```

Router setup

The following procedure assumes a Linux box with multiple NICs is used as a router. Modify the steps as required to use the syntax for a particular router:

1. Enable IP forwarding on the router:

```
# sysctl -w net.ipv4.ip_forward=1
```

2. Add a route for each node added to the cluster:

```
# ip route add <node_subnet> via <node_ip_address> dev
<interface through which node is L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

CHAPTER 27. ROUTING FROM EDGE LOAD BALANCERS

27.1. OVERVIEW

Pods inside of an OpenShift Container Platform cluster are only reachable via their IP addresses on the cluster network. An edge load balancer can be used to accept traffic from outside networks and proxy the traffic to pods inside the OpenShift Container Platform cluster. In cases where the load balancer is not part of the cluster network, routing becomes a hurdle as the internal cluster network is not accessible to the edge load balancer.

To solve this problem where the OpenShift Container Platform cluster is using OpenShift SDN as the cluster networking solution, there are two ways to achieve network access to the pods.

27.2. INCLUDING THE LOAD BALANCER IN THE SDN

If possible, run an OpenShift Container Platform node instance on the load balancer itself that uses OpenShift SDN as the networking plug-in. This way, the edge machine gets its own Open vSwitch bridge that the SDN automatically configures to provide access to the pods and nodes that reside in the cluster. The *routing table* is dynamically configured by the SDN as pods are created and deleted, and thus the routing software is able to reach the pods.

Mark the load balancer machine as an unschedulable node so that no pods end up on the load balancer itself:

```
$ oadm manage-node <load_balancer_hostname> --schedulable=false
```

If the load balancer comes packaged as a container, then it is even easier to integrate with OpenShift Container Platform: Simply run the load balancer as a pod with the host port exposed. The pre-packaged HAProxy router in OpenShift Container Platform runs in precisely this fashion.

27.3. ESTABLISHING A TUNNEL USING A RAMP NODE

In some cases, the previous solution is not possible. For example, an **F5 BIG-IP®** host cannot run an OpenShift Container Platform node instance or the OpenShift Container Platform SDN because **F5®** uses a custom, incompatible Linux kernel and distribution.

Instead, to enable **F5 BIG-IP®** to reach pods, you can choose an existing node within the cluster network as a *ramp node* and establish a tunnel between the **F5 BIG-IP®** host and the designated ramp node. Because it is otherwise an ordinary OpenShift Container Platform node, the ramp node has the necessary configuration to route traffic to any pod on any node in the cluster network. The ramp node thus assumes the role of a gateway through which the **F5 BIG-IP®** host has access to the entire cluster network.

Following is an example of establishing an **ipip** tunnel between an **F5 BIG-IP®** host and a designated ramp node.

On the F5 BIG-IP® host:

1. Set the following variables:

```
# F5_IP=10.3.89.66 1
# RAMP_IP=10.3.89.89 2
# TUNNEL_IP1=10.3.91.216 3
# CLUSTER_NETWORK=10.128.0.0/14 4
```

1 2

The **F5_IP** and **RAMP_IP** variables refer to the **F5 BIG-IP®** host's and the ramp node's IP addresses, respectively, on a shared, internal network.

3

An arbitrary, non-conflicting IP address for the **F5®** host's end of the **ipip** tunnel.

4

The overlay network CIDR that the OpenShift SDN uses to assign addresses to pods.

2. Delete any old route, self, tunnel and SNAT pool:

```
# tmsh delete net route $CLUSTER_NETWORK || true
# tmsh delete net self SDN || true
# tmsh delete net tunnels tunnel SDN || true
# tmsh delete ltm snatpool SDN_snatpool || true
```

3. Create the new tunnel, self, route and SNAT pool and use the SNAT pool in the virtual servers:

```
# tmsh create net tunnels tunnel SDN \
    \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsh create net self SDN \{ address \
    ${TUNNEL_IP1}/24 allow-service all vlan SDN \}
# tmsh create net route $CLUSTER_NETWORK interface SDN
# tmsh create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1
}
# tmsh modify ltm virtual  ose-vserver source-address-translation
{ type snat pool SDN_snatpool }
# tmsh modify ltm virtual  https-ose-vserver source-address-
translation { type snat pool SDN_snatpool }
```

On the ramp node:

1. Set the following variables:

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217 1
# CLUSTER_NETWORK=10.128.0.0/1 2
```

1

A second, arbitrary IP address for the ramp node's end of the **ipip** tunnel.

2

The overlay network CIDR that the OpenShift SDN uses to assign addresses to pods.

2. Delete any old tunnel:

```
# ip tunnel del tun1 || true
```

3. Create the **ipip** tunnel on the ramp node, using a suitable L2-connected interface (e.g., **eth0**):

```
# ip tunnel add tun1 mode ipip \
    remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

4. SNAT the tunnel IP with an unused IP from the SDN subnet:

```
# source /run/openshift-sdn/config.env
# tap1=$(ip -o -4 addr list tun0 | awk '{print $4}' | cut -d/ -f1
| head -n 1)
# subaddr=$(echo ${OPENSHIFT_SDN_TAP1_ADDR:-"$tap1"} | cut -d "."
-f 1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

5. Assign this **RAMP_SDN_IP** as an additional address to **tun0** (the local SDN's gateway):

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

6. Modify the OVS rules for SNAT:

```
# ipflowopts="cookie=0x999,ip"
# arpflowopts="cookie=0x999, table=0, arp"
#
# ovs-ofctl -O OpenFlow13 add-flow br0 \
```

```
"${ipflowopts},nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN
_IP},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_
IP1},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, priority=200, in_port=2,
arp_spa=${RAMP_SDN_IP}, arp_tpa=${CLUSTER_NETWORK},
actions=goto_table:5"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "arp, table=5, priority=300, arp_tpa=${RAMP_SDN_IP},
actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"ip,table=5,priority=300,nw_dst=${RAMP_SDN_IP},actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0
"${ipflowopts},nw_dst=${TUNNEL_IP1},actions=output:2"
```

7. Optionally, if you do not plan on configuring the ramp node to be highly available, mark the ramp node as unschedulable. Skip this step if you do plan to follow the next section and plan on creating a highly available ramp node.

```
$ oadm manage-node <ramp_node_hostname> --schedulable=false
```

Note

The F5 router plug-in integrates with F5 BIG-IP®.

27.3.1. Configuring a Highly-Available Ramp Node

You can use OpenShift Container Platform's **ipfailover** feature, which uses **keepalived** internally, to make the ramp node highly available from **F5 BIG-IP®**'s point of view. To do so, first bring up two nodes, for example called **ramp-node-1** and **ramp-node-2**, on the same L2 subnet.

Then, choose some unassigned IP address from within the same subnet to use for your virtual IP, or *VIP*. This will be set as the **`RAMP_IP`** variable with which you will configure your tunnel on **F5 BIG-IP®**.

For example, suppose you are using the **10.20.30.0/24** subnet for your ramp nodes, and you have assigned **10.20.30.2** to **ramp-node-1** and **10.20.30.3** to **ramp-node-2**. For your VIP, choose some unassigned address from the same **10.20.30.0/24** subnet, for example **10.20.30.4**. Then, to configure **ipfailover**, mark both nodes with a label, such as **f5rampnode**:

```
$ oc label node ramp-node-1 f5rampnode=true
$ oc label node ramp-node-2 f5rampnode=true
```

Similar to instructions from the **ipfailover** documentation, you must now create a service account and add it to the **privileged** SCC. First, create the **f5ipfailover** service account:

```
$ oc create serviceaccount f5ipfailover -n default
```

Next, you can add the **f5ipfailover** service to the **privileged** SCC. To add the **f5ipfailover** in the **default** namespace to the **privileged** SCC, run:

```
$ oadm policy add-scc-to-user privileged
system:serviceaccount:default:f5ipfailover
```

Finally, configure **ipfailover** using your chosen VIP (the **RAMP_IP** variable) and the **f5ipfailover** service account, assigning the VIP to your two nodes using the **f5rampnode** label you set earlier:

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 (1)
# oadm ipfailover <name-tag> \
    --virtual-ips=$RAMP_IP \
    --interface=$IFNAME \
    --watch-port=0 \
    --replicas=2 \
    --service-account=f5ipfailover  \
    --selector='f5rampnode=true'
```

1

The interface where **RAMP_IP** should be configured.

With the above setup, the VIP (the **RAMP_IP** variable) is automatically re-assigned when the ramp node host that currently has it assigned fails.

CHAPTER 28. AGGREGATING CONTAINER LOGS

28.1. OVERVIEW

As an OpenShift Container Platform cluster administrator, you can deploy the EFK stack to aggregate logs for a range of OpenShift Container Platform services. Application developers can view the logs of the projects for which they have view access. The EFK stack aggregates logs from hosts and applications, whether coming from multiple containers or even deleted pods.

The EFK stack is a modified version of the ELK stack and is comprised of:

- Elasticsearch: An object store where all logs are stored.
- Fluentd: Gathers logs from nodes and feeds them to Elasticsearch.
- Kibana: A web UI for Elasticsearch.

Once deployed in a cluster, the stack aggregates logs from all nodes and projects into Elasticsearch, and provides a Kibana UI to view any logs. Cluster administrators can view all logs, but application developers can only view logs for projects they have permission to view. The stack components communicate securely.

Note

Managing Docker Container Logs discusses the use of **`json-file`** logging driver options to manage container logs and prevent filling node disks.

28.2. PRE-DEPLOYMENT CONFIGURATION

1. Ensure that you have deployed a router for the cluster.
2. Ensure that you have the necessary storage for Elasticsearch. Note that each Elasticsearch replica requires its own storage volume. See Elasticsearch for more information.
3. Ansible-based installs should create the **logging-deployer-template** template in the **openshift** project. Otherwise you can create it with the following command:

```
$ oc apply -n openshift -f \
    /usr/share/ansible/openshift-
ansible/roles/openshift_hosted_templates/files/v1.4/enterprise/lo
gging-deployer.yaml
```

4. Create a new project. Once implemented in a single project, the EFK stack collects logs for every project within your OpenShift Container Platform cluster. The examples in this topic use **logging** as an example project:

```
$ oadm new-project logging --node-selector=""
$ oc project logging
```

Note

Specifying an empty node selector on the project is recommended, as Fluentd should be deployed throughout the cluster and any selector would restrict where it is deployed. To control component placement, specify node selectors per component to be applied to their deployment configurations.

5. Create the logging service accounts and custom roles:

```
$ oc new-app logging-deployer-account-template
```

Note

If you deployed logging previously, for example in a different project, then it is normal for the cluster roles to fail to be created because they already exist.

6. Enable the deployer service account to create an OAuthClient (normally a cluster administrator privilege) for Kibana to use later when authenticating against the master.

```
$ oadm policy add-cluster-role-to-user oauth-editor \
       system:serviceaccount:logging:logging-deployer 1
```

1

Use the project you created earlier (for example, **logging**) when specifying this service account.

7. Enable the Fluentd service account to mount and read system logs by adding it to the **privileged** security context, and also enable it to read pod metadata by giving it the **cluster-reader** role:

```
$ oadm policy add-scc-to-user privileged  \
    system:serviceaccount:logging:aggregated-logging-fluentd 1
$ oadm policy add-cluster-role-to-user cluster-reader \
    system:serviceaccount:logging:aggregated-logging-fluentd 2
```

1 2

Use the project you created earlier (for example, **logging**) when specifying this service account.

8. Enable the Elasticsearch service account to get cluster role bindings so that it can verify a user's roles and allow access to operations logs:

```
$ oadm policy add-cluster-role-to-user rolebinding-reader \
     system:serviceaccount:logging:aggregated-logging-
elasticsearch 1
```

1

Use the project you created earlier (for example, **logging**) when specifying this service account.

9. Ensure that port 9300 is open. By default the Elasticsearch service uses port 9300 for TCP communication between nodes in a cluster.

28.3. SPECIFYING DEPLOYER PARAMETERS

Parameters for the EFK deployment may be specified in the form of a ConfigMap, a secret, or template parameters (which are passed to the deployer in environment variables). The deployer looks for each value first in a **logging-deployer** ConfigMap, then a **logging-deployer** secret, then as an environment variable. Any or all may be omitted if not needed. If you are specifying values within a ConfigMap, the values will not be reflected in the output from **oc new-app**, but will still precede the corresponding template values within the deployer pod.

The available parameters are outlined below. Typically, you should at least specify the host name at which Kibana should be exposed to client browsers, and also the master URL where client browsers will be directed to for authenticating to OpenShift Container Platform.

1. Create a ConfigMap to provide most deployer parameters. An invocation supplying the most important parameters might be:

```
$ oc create configmap logging-deployer \
   --from-literal kibana-hostname=kibana.example.com \
   --from-literal public-master-
url=https://master.example.com:8443 \
   --from-literal es-cluster-size=3 \
   --from-literal es-instance-ram=8G
```

2. Edit the ConfigMap YAML file after creating it:

```
$ oc edit configmap logging-deployer
```

Other parameters are available. Read the ElasticSearch section before choosing ElasticSearch parameters for the deployer, and the Fluentd section for some possible parameters:

Parameter	Description
kibana-hostname	The external host name for web clients to reach Kibana.

Parameter	Description
public-master-url	The external URL for the master; used for OAuth purposes.
es-cluster-size (default: 1)	The number of instances of Elasticsearch to deploy. Redundancy requires at least three, and more can be used for scaling.
es-instance-ram (default: 8G)	Amount of RAM to reserve per Elasticsearch instance. The default is 8G (for 8GB), and it must be at least 512M. Possible suffixes are G,g,M,m.
es-pvc-prefix (default: **logging-es-**)	Prefix for the names of persistent volume claims to be used as storage for Elasticsearch instances; a number will be appended per instance (for example, **logging-es-1**). If they do not already exist, they will be created with size ***es-pvc-size***.
es-pvc-size	Size of the persistent volume claim to create per ElasticSearch instance, 100G, for example. If omitted, no PVCs are created and ephemeral volumes are used instead.
es-pvc-dynamic	Set to **`true`** to have created persistent volume claims annotated so that their backing storage can be dynamically provisioned (if that is available for your cluster).
storage-group	Number of a supplemental group ID for access to Elasticsearch storage volumes; backing volumes should allow access by this group ID (defaults to 65534).
fluentd-nodeselector (default: **logging-infra-fluentd=true**)	A node selector that specifies which nodes are eligible targets for deploying Fluentd instances. All nodes where Fluentd should run (typically, all) must have this label before Fluentd will be able to run and collect logs.
es-nodeselector	A node selector that specifies which nodes are eligible targets for deploying Elasticsearch instances. This can be used to place these instances on nodes reserved and/or optimized for running them. For example, the selector could be **`node-type=infrastructure`**. At least one active node must have this label before Elasticsearch will deploy.

Parameter	Description
kibana-nodeselector	A node selector that specifies which nodes are eligible targets for deploying Kibana instances.
curator-nodeselector	A node selector that specifies which nodes are eligible targets for deploying Curator instances.
enable-ops-cluster	If set to `true`, configures a second Elasticsearch cluster and Kibana for operations logs. Fluentd splits logs between the main cluster and a cluster reserved for operations logs (which consists of ***/var/log/messages*** on nodes and the logs from the projects **default**, **openshift**, and **openshift-infra**). This means a second Elasticsearch and Kibana are deployed. The deployments are distinguishable by the **-ops** included in their names and have parallel deployment options listed below.
kibana-ops-hostname, es-ops-instance-ram, es-ops-pvc-size, es-ops-pvc-prefix, es-ops-cluster-size, es-ops-nodeselector, kibana-ops-nodeselector, curator-ops-nodeselector	Parallel parameters for the ops log cluster.
image-pull-secret	Specify the name of an existing pull secret to be used for pulling component images from an authenticated registry.

3. Create a secret to provide security-related files to the deployer. Providing the secret is optional, and the objects will be randomly generated if not supplied.

   You can supply the following files when creating a new secret, for example:

```
$ oc create secret generic logging-deployer \
   --from-file kibana.crt=/path/to/cert \
   --from-file kibana.key=/path/to/key
```

File Name	Description
kibana.crt	A browser-facing certificate for the Kibana server.

File Name	Description
kibana.key	A key to be used with the Kibana certificate.
kibana-ops.crt	A browser-facing certificate for the Ops Kibana server.
kibana-ops.key	A key to be used with the Ops Kibana certificate.
server-tls.json	JSON TLS options to override the Kibana server defaults. Refer to Node.JS docs for available options.
ca.crt	A certificate for a CA that will be used to sign all certificates generated by the deployer.
ca.key	A matching CA key.

28.4. DEPLOYING THE EFK STACK

The EFK stack is deployed using a template to create a deployer pod that reads the deployment parameters and manages the deployment.

Run the deployer, optionally specifying parameters (described in the table below), for example:

Without template parameters:

```
$ oc new-app logging-deployer-template
```

With parameters:

```
$ oc new-app logging-deployer-template \
             --param IMAGE_VERSION=<tag> \
             --param MODE=install
```

Replace **`<tag>`** with **`v3.4.1`** for the latest version.

Parameter Name	Description
IMAGE_PREFIX	The prefix for logging component images. For example, setting the prefix to **registry.access.redhat.com/openshift3/** creates **registry.access.redhat.com/openshift3/logging-deployer:latest**.

Parameter Name	Description
IMAGE_VERSION	The version for logging component images. For example, setting the version to **v3.3** creates **registry.access.redhat.com/openshift3/logging-deployer:v3.3**.
MODE (default: **install**)	Mode to run the deployer in; one of **`install`**, **`uninstall`**, **`reinstall`**, **`upgrade`**, **`migrate`**, **`start`**, **`stop`**.

Running the deployer creates a deployer pod and prints its name. Wait until the pod is running. This can take up to a few minutes for OpenShift Container Platform to retrieve the deployer image from the registry. Watch its process with:

```
$ oc get pod/<pod_name> -w
```

It will eventually enter **Running** status and end in **Complete** status. If takes too long to start, retrieve more details about the pod and any associated events with:

```
$ oc describe pod/<pod_name>
```

Check the logs if the deployment does not complete successfully:

```
$ oc logs -f <pod_name>
```

Once deployment completes successfully, you may need to label the nodes for Fluentd to deploy on, and may have other adjustments to make to the deployed components. These tasks are described in the next section.

28.5. UNDERSTANDING AND ADJUSTING THE DEPLOYMENT

This section describes adjustments that you can make to deployed components.

28.5.1. Ops Cluster

Note

The logs for the **default**, **openshift**, and **openshift-infra** projects are automatically aggregated and grouped into the **.operations** item in the Kibana interface.

The project where you have deployed the EFK stack (**logging**, as documented here) is *not* aggregated into **.operations** and is found under its ID.

If you set **`enable-ops-cluster`** to **true** for the deployer, Fluentd is configured to split logs between the main ElasticSearch cluster and another cluster reserved for operations logs (which are defined as node system logs and the projects **default**, **openshift**, and **openshift-infra**). Therefore, a separate Elasticsearch cluster, a separate Kibana, and a separate Curator are deployed to index,

access, and manage operations logs. These deployments are set apart with names that include **-ops**. Keep these separate deployments in mind if you enabled this option. Most of the following discussion also applies to the operations cluster if present, just with the names changed to include **-ops**.

28.5.2. Elasticsearch

A highly-available environment requires at least three replicas of Elasticsearch; each on a different host. Elasticsearch replicas require their own storage, but an OpenShift Container Platform deployment configuration shares storage volumes between all its pods. So, when scaled up, the EFK deployer ensures each replica of Elasticsearch has its own deployment configuration.

It is possible to scale your cluster up after creation by adding more deployments from a template; however, scaling up (or down) requires the correct procedure and an awareness of clustering parameters (to be described in a separate section). It is best to indicate the desired scale at first deployment.

Refer to Elastic's documentation for considerations involved in choosing storage and network location as directed below.

Viewing all Elasticsearch Deployments

To view all current Elasticsearch deployments:

```
$ oc get dc --selector logging-infra=elasticsearch
```

Node Selector

Because Elasticsearch can use a lot of resources, all members of a cluster should have low latency network connections to each other and to any remote storage. Ensure this by directing the instances to dedicated nodes, or a dedicated region within your cluster, using a node selector.

To configure a node selector, specify the **es-nodeselector** configuration option at deployment. This applies to all Elasticsearch deployments; if you need to individualize the node selectors, you must manually edit each deployment configuration after deployment.

Persistent Elasticsearch Storage

By default, the deployer creates an ephemeral deployment in which all of a pod's data is lost upon restart. For production usage, specify a persistent storage volume for each Elasticsearch deployment configuration. You can create the necessary persistent volume claims before deploying or have them created for you. The PVCs must be named based on the **es-pvc-prefix** setting, which defaults to **logging-es-**; each PVC name will have a sequence number added to it, so **logging-es-1**, **logging-es-2**, and so on. If a PVC needed for the deployment exists already, it is used; if not, and **es-pvc-size** has been specified, it is created with a request for that size.

Warning

Using NFS storage as a volume or a persistent volume (or via NAS such as Gluster) is not supported for Elasticsearch storage, as Lucene relies on file system behavior that NFS does not supply. Data corruption and other problems can occur. If NFS storage is a requirement, you can allocate a large file on a volume to serve as a storage device and mount it locally on one host. For example, if your NFS storage volume is mounted at ***/nfs/storage***:

```
$ truncate -s 1T /nfs/storage/elasticsearch-1
$ mkfs.xfs /nfs/storage/elasticsearch-1
$ mount -o loop /nfs/storage/elasticsearch-1 /usr/local/es-
storage
$ chown 1000:1000 /usr/local/es-storage
```

Then, use ***/usr/local/es-storage*** as a host-mount as described below. Use a different backing file as storage for each Elasticsearch replica.

This loopback must be maintained manually outside of OpenShift Container Platform, on the node. You must not maintain it from inside a container.

It is possible to use a local disk volume (if available) on each node host as storage for an Elasticsearch replica. Doing so requires some preparation as follows.

1. The relevant service account must be given the privilege to mount and edit a local volume:

```
$ oadm policy add-scc-to-user privileged  \
       system:serviceaccount:logging:aggregated-logging-
elasticsearch 1
```

1

Use the project you created earlier (for example, **logging**) when specifying this service account.

2. Each Elasticsearch replica definition must be patched to claim that privilege, for example:

```
$ for dc in $(oc get deploymentconfig --selector logging-
infra=elasticsearch -o name); do
    oc scale $dc --replicas=0
    oc patch $dc \
       -p '{"spec":{"template":{"spec":{"containers":
[{"name":"elasticsearch","securityContext":{"privileged":
true}}]}}}}'
  done
```

3. The Elasticsearch replicas must be located on the correct nodes to use the local storage, and should not move around even if those nodes are taken down for a period of time. This requires giving each Elasticsearch replica a node selector that is unique to a node where an administrator has allocated storage for it. To configure a node selector, edit each

Elasticsearch deployment configuration and add or edit the **nodeSelector** section to specify a unique label that you have applied for each desired node:

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1" 1
```

1

This label should uniquely identify a replica with a single node that bears that label, in this case **`logging-es-node=1`**. Use the **`oc label`** command to apply labels to nodes as needed.

To automate applying the node selector you can instead use the **`oc patch`** command:

```
$ oc patch dc/logging-es-<suffix> \
   -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-
node":"1"}}}}}'
```

4. Once these steps are taken, a local host mount can be applied to each replica as in this example (where we assume storage is mounted at the same path on each node):

```
$ for dc in $(oc get deploymentconfig --selector logging-
infra=elasticsearch -o name); do
    oc set volume $dc \
          --add --overwrite --name=elasticsearch-storage \
          --type=hostPath --path=/usr/local/es-storage
    oc rollout latest $dc
    oc scale $dc --replicas=1
  done
```

Changing the Scale of Elasticsearch

If you need to scale up the number of Elasticsearch instances your cluster uses, it is not as simple as scaling up an Elasticsearch deployment configuration. This is due to the nature of persistent volumes and how Elasticsearch is configured to store its data and recover the cluster. Instead, scaling up requires creating a deployment configuration for each Elasticsearch cluster node.

By far the simplest way to change the scale of Elasticsearch is to reinstall the whole deployment. Assuming you have supplied persistent storage for the deployment, this should not be very disruptive. Simply re-run the deployer with the updated **`es-cluster-size`** configuration value and the **`MODE=reinstall`** template parameter. For example:

```
$ oc edit configmap logging-deployer
  [change es-cluster-size value to 5]
$ oc new-app logging-deployer-template --param MODE=reinstall
```

If you previously deployed using template parameters rather than a ConfigMap, this would be a good time to create a ConfigMap instead for future deployer execution.

If you do not wish to reinstall, for instance because you have made customizations that you would like to preserve, then it is possible to add new Elasticsearch deployment configurations to the cluster using a template supplied by the deployer. This requires a more complicated procedure however.

During installation, the deployer creates templates with the Elasticsearch configurations provided to it: **`logging-es-template`** (and **`logging-es-ops-template`** if the deployer was run with **`ENABLE_OPS_CLUSTER=true`**). You can use these for scaling, but you need to adjust the size-related parameters in the templates:

Parameter	Description
`NODE_QUORUM`	The quorum required to elect a new master. Should be more than half the intended cluster size.
`RECOVER_AFTER_NODES`	When restarting the cluster, require this many nodes to be present before starting recovery. Defaults to one less than the cluster size to allow for one missing node.
`RECOVER_EXPECTED_NODES`	When restarting the cluster, wait for this number of nodes to be present before starting recovery. By default, the same as the cluster size.

The node quorum and recovery settings in the template were set based on the **`es-[ops-]cluster-size`** value initially provided to the deployer. Since the cluster size is changing, those values need to be overridden.

1. The existing deployment configurations for that cluster also need to have the three environment variable values above updated. To edit each of the configurations for the cluster in series, you may use the following command:

   ```
   $ oc edit $(oc get dc -l component=es[-ops] -o name)
   ```

   Edit the environment variables supplied so that the next time they restart, they will begin with the correct values. For example, for a cluster of size 5, you would set **`NODE_QUORUM`** to **`3`**, **`RECOVER_AFTER_NODES`** to **`4`**, and **`RECOVER_EXPECTED_NODES`** to **`5`**.

2. Create additional deployment configurations by running the following command against the Elasticsearch cluster you want to to scale up for (**`logging-es-template`** or **`logging-es-ops-template`**), overriding the parameters as above.

   ```
   $ oc new-app logging-es[-ops]-template \
        --param NODE_QUORUM=3 \
        --param RECOVER_AFTER_NODES=4 \
        --param RECOVER_EXPECTED_NODES=5
   ```

   These deployments will be named differently, but all will have the **`logging-es`** prefix.

3. Each new deployment configuration is created without a persistent volume. If you want to attach a persistent volume to it, after creation you can use the **`oc set volume`** command to do so, for example:

```
$ oc volume dc/logging-es-<suffix> \
          --add --overwrite --name=elasticsearch-storage \
          --type=persistentVolumeClaim --claim-name=<your_pvc>
```

4. After the intended number of deployment configurations are created, scale up each new one to deploy it:

```
$ oc scale --replicas=1 dc/logging-es-<suffix>
```

Allowing cluster-reader to view operations logs

By default, only **`cluster-admin`** users are granted access in Elasticsearch and Kibana to view operations logs. To allow **`cluster-reader`** users to also view these logs, update the value of **`openshift.operations.allow_cluster_reader`** in the Elasticsearch configmap to **`true`**:

```
$ oc edit configmap/logging-elasticsearch
```

Please note that changes to the configmap might not appear until after redeploying the pods.

28.5.3. Fluentd

Fluentd is deployed as a DaemonSet that deploys replicas according to a node label selector (which you can specify with the deployer parameter **`fluentd-nodeselector`**; the default is **`logging-infra-fluentd`**).

Once you have ElasticSearch running as desired, label the nodes intended for Fluentd deployment to feed their logs into ES. The example below would label a node named **`node.example.com`** using the default Fluentd node selector:

```
$ oc label node/node.example.com logging-infra-fluentd=true
```

Alternatively, you can label all nodes with:

```
$ oc label node --all logging-infra-fluentd=true
```

Note

Labeling nodes requires cluster administrator capability.

Having Fluentd Use the Systemd Journal as the Log Source

By default, Fluentd reads from ***/var/log/messages*** and ***/var/log/containers/<container>.log*** for system logs and container logs, respectively. You can instead use the systemd journal as the log source. There are three deployer configuration parameters available in the deployer ConfigMap:

Parameter	Description
`use-journal`	The default is empty, which tells the deployer to have Fluentd check which log driver Docker is using. If Docker is using `--log-driver=journald`, Fluentd reads from the systemd journal, otherwise, it assumes docker is using the `json-file` log driver and reads from the ***/var/log*** file sources. You can specify the `use-journal` option as `true` or `false` to be explicit about which log source to use. Using the systemd journal requires `docker-1.10` or later, and Docker must be configured to use `--log-driver=journald`.
`journal-source`	The default is empty, so that when using the systemd journal, Fluentd first looks for ***/var/log/journal***, and if that is not available, uses ***/run/log/journal*** as the journal source. You can specify `journal-source` with an explicit journal path. For example, if you want Fluentd to always read logs from the transient in-memory journal, set `journal-source`=***/run/log/journal***.
`journal-read-from-head`	If this setting is **false**, Fluentd starts reading from the end of the journal, ignoring historical logs. If this setting is **true**, Fluentd starts reading logs from the beginning of the journal.

Note

As of OpenShift Container Platform 3.3, Fluentd no longer reads historical log files when using the JSON file log driver. In situations where clusters have a large number of log files and are older than the EFK deployment, this avoids delays when pushing the most recent logs into Elasticsearch. Curator deleting logs are migrated soon after they are added to Elasticsearch.

Note

It may require several minutes, or hours, depending on the size of your journal, before any new log entries are available in Elasticsearch, when using `journal-read-from-head=true`.

Having Fluentd Send Logs to Another Elasticsearch

Note

The use of **ES_COPY** is being deprecated. To configure FluentD to send a copy of its logs to an external aggregator, use Fluentd Secure Forward instead.

You can configure Fluentd to send a copy of each log message to both the Elasticsearch instance

included with OpenShift Container Platform aggregated logging, *and* to an external Elasticsearch instance. For example, if you already have an Elasticsearch instance set up for auditing purposes, or data warehousing, you can send a copy of each log message to that Elasticsearch.

This feature is controlled via environment variables on Fluentd, which can be modified as described below.

If its environment variable **ES_COPY** is **true**, Fluentd sends a copy of the logs to another Elasticsearch. The names for the copy variables are just like the current **ES_HOST**, **OPS_HOST**, and other variables, except that they add **_COPY**: **ES_COPY_HOST**, **OPS_COPY_HOST**, and so on. There are some additional parameters added:

- **ES_COPY_SCHEME**, **OPS_COPY_SCHEME** - can use either **http** or **https** - defaults to **https**
- **ES_COPY_USERNAME**, **OPS_COPY_USERNAME** - user name to use to authenticate to Elasticsearch using username/password auth
- **ES_COPY_PASSWORD**, **OPS_COPY_PASSWORD** - password to use to authenticate to Elasticsearch using username/password auth

Note

Sending logs directly to an AWS Elasticsearch instance is not supported. Use Fluentd Secure Forward to direct logs to an instance of Fluentd that you control and that is configured with the **fluent-plugin-aws-elasticsearch-service** plug-in.

To set the parameters:

1. Edit the template for the Fluentd daemonset:

   ```
   $ oc edit -n logging template logging-fluentd-template
   ```

   Add or edit the environment variable **ES_COPY** to have the value **"true"** (with the quotes), and add or edit the COPY variables listed above.

2. Recreate the Fluentd daemonset from the template:

   ```
   $ oc delete daemonset logging-fluentd
   $ oc new-app logging-fluentd-template
   ```

Configuring Fluentd to Send Logs to an External Log Aggregator

You can configure Fluentd to send a copy of its logs to an external log aggregator, and not the default Elasticsearch, using the **secure-forward** plug-in. From there, you can further process log records after the locally hosted Fluentd has processed them.

The deployer provides a **secure-forward.conf** section in the Fluentd configmap for configuring the external aggregator:

```
<store>
@type secure_forward
self_hostname pod-${HOSTNAME}
shared_key thisisasharedkey
secure yes
```

```
enable_strict_verification yes
ca_cert_path /etc/fluent/keys/your_ca_cert
ca_private_key_path /etc/fluent/keys/your_private_key
ca_private_key_passphrase passphrase
<server>
  host ose1.example.com
  port 24284
</server>
<server>
  host ose2.example.com
  port 24284
  standby
</server>
<server>
  host ose3.example.com
  port 24284
  standby
</server>
</store>
```

This can be updated using the **`oc edit`** command:

```
$ oc edit configmap/logging-fluentd
```

Certificates to be used in **`secure-forward.conf`** can be added to the existing secret that is mounted on the Fluentd pods. The **`your_ca_cert`** and **`your_private_key`** values must match what is specified in **`secure-forward.conf`** in **`configmap/logging-fluentd`**:

```
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_ca_cert','value':'$(base64
/path/to/your_ca_cert.pem)'}]"
$ oc patch secrets/logging-fluentd --type=json \
  --patch "
[{'op':'add','path':'/data/your_private_key','value':'$(base64
/path/to/your_private_key.pem)'}]"
```

Note

Avoid using secret names such as 'cert', 'key', and 'ca' so that the values do not conflict with the keys generated by the Deployer pod for Fluentd to talk to the OpenShift Container Platform hosted Elasticsearch.

When configuring the external aggregator, it must be able to accept messages securely from Fluentd.

If the external aggregator is another Fluentd process, it must have the **`fluent-plugin-secure-forward`** plug-in installed and make use of the input plug-in it provides:

```
<source>
  @type secure_forward

  self_hostname ${HOSTNAME}
  bind 0.0.0.0
```

```
  port 24284

  shared_key thisisasharedkey

  secure yes
  cert_path        /path/for/certificate/cert.pem
  private_key_path /path/for/certificate/key.pem
  private_key_passphrase secret_foo_bar_baz
</source>
```

Further explanation of how to set up the **`fluent-plugin-secure-forward`** plug-in can be found here.

Throttling logs in Fluentd

For projects that are especially verbose, an administrator can throttle down the rate at which the logs are read in by Fluentd before being processed.

Warning

Throttling can contribute to log aggregation falling behind for the configured projects; log entries can be lost if a pod is deleted before Fluentd catches up.

Note

Throttling does not work when using the systemd journal as the log source. The throttling implementation depends on being able to throttle the reading of the individual log files for each project. When reading from the journal, there is only a single log source, no log files, so no file-based throttling is available. There is not a method of restricting the log entries that are read into the Fluentd process.

To tell Fluentd which projects it should be restricting, edit the throttle configuration in its ConfigMap after deployment:

```
$ oc edit configmap/logging-fluentd
```

The format of the ***throttle-config.yaml*** key is a YAML file that contains project names and the desired rate at which logs are read in on each node. The default is 1000 lines at a time per node. For example:

```
logging:
  read_lines_limit: 500

test-project:
  read_lines_limit: 10

.operations:
  read_lines_limit: 100
```

28.5.4. Kibana

To access the Kibana console from the OpenShift Container Platform web console, add the **`loggingPublicURL`** parameter in the ***/etc/origin/master/master-config.yaml*** file, with the URL of the Kibana console (the **`kibana-hostname`** parameter). The value must be an HTTPS URL:

```
...
assetConfig:
  ...
  loggingPublicURL: "https://kibana.example.com"
...
```

Setting the **`loggingPublicURL`** parameter creates a **View Archive** button on the OpenShift Container Platform web console under the **Browse** → **Pods** → **<pod_name>** → **Logs** tab. This links to the Kibana console.

You can scale the Kibana deployment as usual for redundancy:

```
$ oc scale dc/logging-kibana --replicas=2
```

You can see the user interface by visiting the site specified at the **`KIBANA_HOSTNAME`** variable.

See the Kibana documentation for more information on Kibana.

28.5.5. Curator

Curator allows administrators to configure scheduled Elasticsearch maintenance operations to be performed automatically on a per-project basis. It is scheduled to perform actions daily based on its configuration. Only one Curator pod is recommended per Elasticsearch cluster. Curator is configured via a YAML configuration file with the following structure:

```
$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE

$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE
 ...
```

The available parameters are:

Variable Name	Description
`$PROJECT_NAME`	The actual name of a project, such as **myapp-devel**. For OpenShift Container Platform **operations** logs, use the name **`.operations`** as the project name.
`$ACTION`	The action to take, currently only **`delete`** is allowed.

Variable Name	Description
$UNIT	One of **days**, **weeks**, or **months**.
$VALUE	An integer for the number of units.
.defaults	Use **.defaults** as the **$PROJECT_NAME** to set the defaults for projects that are not specified.
runhour	(Number) the hour of the day in 24-hour format at which to run the Curator jobs. For use with **.defaults**.
runminute	(Number) the minute of the hour at which to run the Curator jobs. For use with **.defaults**.

For example, to configure Curator to:

- delete indices in the **myapp-dev** project older than **1 day**
- delete indices in the **myapp-qe** project older than **1 week**
- delete **operations** logs older than **8 weeks**
- delete all other projects indices after they are **30 days** old
- run the Curator jobs at midnight every day

Use:

```
myapp-dev:
 delete:
   days: 1

myapp-qe:
  delete:
    weeks: 1

.operations:
  delete:
    weeks: 8

.defaults:
  delete:
    days: 30
  runhour: 0
  runminute: 0
```

Important

When you use **`month`** as the **`$UNIT`** for an operation, Curator starts counting at the first day of the current month, not the current day of the current month. For example, if today is April 15, and you want to delete indices that are 2 months older than today (delete: months: 2), Curator does not delete indices that are dated older than February 15; it deletes indices older than February 1. That is, it goes back to the first day of the current month, then goes back two whole months from that date. If you want to be exact with Curator, it is best to use days (for example, **`delete: days: 30`**).

28.5.5.1. Creating the Curator Configuration

The deployer provides a ConfigMap from which Curator reads its configuration. You may edit or replace this ConfigMap to reconfigure Curator. Currently the **`logging-curator`** ConfigMap is used to configure both your ops and non-ops Curator instances. Any **`.operations`** configurations will be in the same location as your application logs configurations.

1. To edit the provided ConfigMap to configure your Curator instances:

```
$ oc edit configmap/logging-curator
```

2. To replace the provided ConfigMap instead:

```
$ create /path/to/mycuratorconfig.yaml
$ oc create configmap logging-curator -o yaml \
  --from-file=config.yaml=/path/to/mycuratorconfig.yaml | \
  oc replace -f -
```

3. After you make your changes, redeploy Curator:

```
$ oc rollout latest dc/logging-curator
$ oc rollout latest dc/logging-curator-ops
```

28.6. CLEANUP

Remove everything generated during the deployment while leaving other project contents intact:

```
$ oc new-app logging-deployer-template --param MODE=uninstall
```

28.7. UPGRADING

To upgrade the EFK logging stack, see Manual Upgrades.

28.8. TROUBLESHOOTING KIBANA

Using the Kibana console with OpenShift Container Platform can cause problems that are easily solved, but are not accompanied with useful error messages. Check the following troubleshooting sections if you are experiencing any problems when deploying Kibana on OpenShift Container Platform:

Login Loop

The OAuth2 proxy on the Kibana console must share a secret with the master host's OAuth2 server. If the secret is not identical on both servers, it can cause a login loop where you are continuously redirected back to the Kibana login page.

To fix this issue, delete the current OAuthClient, and create a new one, using the same template as before:

```
$ oc delete oauthclient/kibana-proxy
$ oc new-app logging-support-template
```

Cryptic Error When Viewing the Console

When attempting to visit the Kibana console, you may receive a browser error instead:

```
{"error":"invalid_request","error_description":"The request is missing
a required parameter,
 includes an invalid parameter value, includes a parameter more than
once, or is otherwise malformed."}
```

This can be caused by a mismatch between the OAuth2 client and server. The return address for the client must be in a whitelist so the server can securely redirect back after logging in.

Fix this issue by replacing the OAuthClient entry:

```
$ oc delete oauthclient/kibana-proxy
$ oc new-app logging-support-template
```

If the problem persists, check that you are accessing Kibana at a URL listed in the OAuth client. This issue can be caused by accessing the URL at a forwarded port, such as 1443 instead of the standard 443 HTTPS port. You can adjust the server whitelist by editing the OAuth client:

```
$ oc edit oauthclient/kibana-proxy
```

503 Error When Viewing the Console

If you receive a proxy error when viewing the Kibana console, it could be caused by one of two issues.

First, Kibana may not be recognizing pods. If Elasticsearch is slow in starting up, Kibana may timeout trying to reach it. Check whether the relevant service has any endpoints:

```
$ oc describe service logging-kibana
Name:                   logging-kibana
[...]
Endpoints:              <none>
```

If any Kibana pods are live, endpoints will be listed. If they are not, check the state of the Kibana pods and deployment. You may need to scale the deployment down and back up again.

The second possible issue may be caused if the route for accessing the Kibana service is masked.

This can happen if you perform a test deployment in one project, then deploy in a different project without completely removing the first deployment. When multiple routes are sent to the same destination, the default router will only route to the first created. Check the problematic route to see if it is defined in multiple places:

```
$ oc get route  --all-namespaces --selector logging-infra=support
```

F-5 Load Balancer and X-Forwarded-For Enabled

If you are attempting to use a F-5 load balancer in front of Kibana with **`X-Forwarded-For`** enabled, this can cause an issue in which the Elasticsearch **`Searchguard`** plug-in is unable to correctly accept connections from Kibana.

Example Kibana Error Message

```
Kibana: Unknown error while connecting to Elasticsearch

Error: Unknown error while connecting to Elasticsearch
Error: UnknownHostException[No trusted proxies]
```

To configure Searchguard to ignore the extra header:

1. Scale down all Fluentd pods.
2. Scale down Elasticsearch after the Fluentd pods have terminated.
3. Add **`searchguard.http.xforwardedfor.header: DUMMY`** to the Elasticsearch configuration section.

   ```
   $ oc edit configmap/logging-elasticsearch 1
   ```

   1

   This approach requires that Elasticsearch's configurations are within a ConfigMap.
4. Scale Elasticsearch back up.
5. Scale up all Fluentd pods.

28.9. SENDING LOGS TO AN EXTERNAL ELASTICSEARCH INSTANCE

Fluentd sends logs to the value of the **`ES_HOST`**, **`ES_PORT`**, **`OPS_HOST`**, and **`OPS_PORT`** environment variables of the Elasticsearch deployment configuration. The application logs are directed to the **`ES_HOST`** destination, and operations logs to **`OPS_HOST`**.

Note

Sending logs directly to an AWS Elasticsearch instance is not supported. Use Fluentd Secure Forward to direct logs to an instance of Fluentd that you control and that is configured with the **`fluent-plugin-aws-elasticsearch-service`** plug-in.

To direct logs to a specific Elasticsearch instance, edit the deployment configuration and replace the value of the above variables with the desired instance:

```
$ oc edit dc/<deployment_configuration>
```

For an external Elasticsearch instance to contain both application and operations logs, you can set **`ES_HOST`** and **`OPS_HOST`** to the same destination, while ensuring that **`ES_PORT`** and **`OPS_PORT`** also have the same value.

If your externally hosted Elasticsearch instance does not use TLS, update the **`_CLIENT_CERT`**, **`_CLIENT_KEY`**, and **`_CA`** variables to be empty. If it does use TLS, but not mutual TLS, update the **`_CLIENT_CERT`** and **`_CLIENT_KEY`** variables to be empty and patch or recreate the **logging-fluentd** secret with the appropriate **`_CA`** value for communicating with your Elasticsearch instance. If it uses Mutual TLS as the provided Elasticsearch instance does, patch or recreate the **logging-fluentd** secret with your client key, client cert, and CA.

Since Fluentd is deployed by a DaemonSet, update the **logging-fluentd-template** template, delete your current DaemonSet, and recreate it with **`oc new-app logging-fluentd-template`** after seeing all previous Fluentd pods have terminated.

Note

If you are not using the provided Kibana and Elasticsearch images, you will not have the same multi-tenant capabilities and your data will not be restricted by user access to a particular project.

28.10. PERFORMING ADMINISTRATIVE ELASTICSEARCH OPERATIONS

As of the Deployer version 3.2.0, an administrator certificate, key, and CA that can be used to communicate with and perform administrative operations on Elasticsearch are provided within the **logging-elasticsearch** secret.

Note

To confirm whether or not your EFK installation provides these, run:

```
$ oc describe secret logging-elasticsearch
```

If they are not available, refer to Manual Upgrades to ensure you are on the latest version first.

1. Connect to an Elasticsearch pod that is in the cluster on which you are attempting to perform maintenance.

2. To find a pod in a cluster use either:

```
$ oc get pods -l component=es -o name | head -1
$ oc get pods -l component=es-ops -o name | head -1
```

3. Connect to a pod:

```
$ oc rsh <your_Elasticsearch_pod>
```

4. Once connected to an Elasticsearch container, you can use the certificates mounted from the secret to communicate with Elasticsearch per its Indices APIs documentation.

   Fluentd sends its logs to Elasticsearch using the index format **project.{project_name}.{project_uuid}.YYYY.MM.DD** where YYYY.MM.DD is the date of the log record.

   For example, to delete all logs for the **logging** project with uuid **3b3594fa-2ccd-11e6-acb7-0eb6b35eaee3** from June 15, 2016, we can run:

```
$ curl --key /etc/elasticsearch/secret/admin-key \
  --cert /etc/elasticsearch/secret/admin-cert \
  --cacert /etc/elasticsearch/secret/admin-ca -XDELETE \
  "https://localhost:9200/project.logging.3b3594fa-2ccd-11e6-
acb7-0eb6b35eaee3.2016.06.15"
```

CHAPTER 29. AGGREGATE LOGGING SIZING GUIDELINES

29.1. OVERVIEW

The Elasticsearch, Fluentd, and Kibana (EFK) stack aggregates logs from nodes and applications running inside your OpenShift Container Platform installation. Once deployed it uses Fluentd to aggregate event logs from all nodes, projects, and pods into Elasticsearch (ES). It also provides a centralized Kibana web UI where users and administrators can create rich visualizations and dashboards with the aggregated data.

Fluentd bulk uploads logs to an index, in JSON format, then Elasticsearch routes your search requests to the appropriate shards.

29.2. INSTALLATION

The general procedure for installing an aggregate logging stack in OpenShift Container Platform is described in Aggregating Container Logs. There are some important things to keep in mind while going through the installation guide:

In order for the logging pods to spread evenly across your cluster, an empty node selector should be used.

```
$ oadm new-project logging --node-selector=""
```

In conjunction with node labeling, which is done later, this controls pod placement across the logging project. You can now create the logging project.

```
$ oc project logging
```

A local **openshift-ansible** template is used by the deployer.

```
$ oc create -f
${OPENSHIFT_ANSIBLE_REPO}/roles/openshift_hosted_templates/files/${VERS
ION}/enterprise/logging-deployer.yaml
```

Elasticsearch (ES) should be deployed with a cluster size of at least three for resiliency to node failures. This is specified by passing the **ES_CLUSTER_SIZE** parameter to the installer.

```
$ oc new-app logging-deployer-template \
                        --param ES_CLUSTER_SIZE=3 \
                        --param PUBLIC_MASTER_URL=$PUBLIC_MASTER_URL \
                        --param KIBANA_HOSTNAME=$KIBANA_URL
```

Refer to Deploying the EFK Stack for a full list of parameters.

If you do not have an existing Kibana installation, you can use **kibana.example.com** as a value to **KIBANA_HOSTNAME**.

As a last step, ensure Fluentd pod spreading through labeling.

```
$ oc label nodes --all logging-infra-fluentd=true
```

This operation requires the **cluster-admin** default role.

Installation can take some time depending on whether the images were already retrieved from the registry or not, and on the size of your cluster.

Inside the **logging** namespace, you can check your deployment with **`oc get all`**.

```
$ oc get all

NAME                          REVISION                 REPLICAS
TRIGGERED BY
logging-curator               1                        1
logging-es-6cvk237t           1                        1
logging-es-e5x4t4ai           1                        1
logging-es-xmwvnorv           1                        1
logging-kibana                1                        1
NAME                          DESIRED                  CURRENT
AGE
logging-curator-1             1                        1
3d
logging-es-6cvk237t-1         1                        1
3d
logging-es-e5x4t4ai-1         1                        1
3d
logging-es-xmwvnorv-1         1                        1
3d
logging-kibana-1              1                        1
3d
NAME                          HOST/PORT                PATH
SERVICE              TERMINATION   LABELS
logging-kibana                kibana.example.com
logging-kibana       reencrypt     component=support,logging-
infra=support,provider=openshift
logging-kibana-ops            kibana-ops.example.com
logging-kibana-ops   reencrypt     component=support,logging-
infra=support,provider=openshift
NAME                          CLUSTER-IP               EXTERNAL-IP
PORT(S)              AGE
logging-es                    172.24.155.177           <none>
9200/TCP             3d
logging-es-cluster            None                     <none>
9300/TCP             3d
logging-es-ops                172.27.197.57            <none>
9200/TCP             3d
logging-es-ops-cluster        None                     <none>
9300/TCP             3d
logging-kibana                172.27.224.55            <none>
443/TCP              3d
logging-kibana-ops            172.25.117.77            <none>
443/TCP              3d
NAME                          READY                    STATUS
RESTARTS             AGE
logging-curator-1-6s7wy       1/1                      Running
0                    3d
logging-deployer-un6ut        0/1                      Completed
0                    3d
logging-es-6cvk237t-1-cnpw3   1/1                      Running
0                    3d
```

```
logging-es-e5x4t4ai-1-v933h   1/1                       Running
0                   3d
logging-es-xmwvnorv-1-adr5x   1/1                       Running
0                   3d
logging-fluentd-156xn         1/1                       Running
0                   3d
logging-fluentd-40biz         1/1                       Running
0                   3d
logging-fluentd-8k847         1/1                       Running
0                   3d
```

You should end up with a similar setup to the below.

```
$ oc get pods -o wide

NAME                          READY     STATUS      RESTARTS   AGE
NODE
logging-curator-1-6s7wy       1/1       Running     0          3d
ip-172-31-24-239.us-west-2.compute.internal
logging-deployer-un6ut        0/1       Completed   0          3d
ip-172-31-6-152.us-west-2.compute.internal
logging-es-6cvk237t-1-cnpw3   1/1       Running     0          3d
ip-172-31-24-238.us-west-2.compute.internal
logging-es-e5x4t4ai-1-v933h   1/1       Running     0          3d
ip-172-31-24-235.us-west-2.compute.internal
logging-es-xmwvnorv-1-adr5x   1/1       Running     0          3d
ip-172-31-24-233.us-west-2.compute.internal
logging-fluentd-156xn         1/1       Running     0          3d
ip-172-31-24-241.us-west-2.compute.internal
logging-fluentd-40biz         1/1       Running     0          3d
ip-172-31-24-236.us-west-2.compute.internal
logging-fluentd-8k847         1/1       Running     0          3d
ip-172-31-24-237.us-west-2.compute.internal
logging-fluentd-9a3qx         1/1       Running     0          3d
ip-172-31-24-231.us-west-2.compute.internal
logging-fluentd-abvgj         1/1       Running     0          3d
ip-172-31-24-228.us-west-2.compute.internal
logging-fluentd-bh74n         1/1       Running     0          3d
ip-172-31-24-238.us-west-2.compute.internal
...
...
```

By default the amount of RAM allocated to each ES instance is 8GB. **`ES_INSTANCE_RAM`** is the parameter used in the **openshift-ansible**template. Keep in mind that **half** of this value will be passed to the individual elasticsearch pods java processes heap size.

Learn more about installing EFK.

29.2.1. Large Clusters

At 100 nodes or more, it is recommended to pre-pull the logging images first and to set **`ImagePullPolicy: IfNotPresent`** in the ***logging-deployer.yaml*** file. After deploying the logging infrastructure pods (Elasticsearch, Kibana and Curator), node labeling should be done in steps of 20 nodes at a time. For example:

Using a simple loop:

```
$ while read node; do oc label nodes $node logging-infra-fluentd=true;
done < 20_fluentd.lst
```

The below also works:

```
$ oc label nodes 10.10.0.{100..119} logging-infra-fluentd=true
```

Labeling nodes in groups paces the DaemonSets used by OpenShift logging, helping to avoid contention on shared resources such as the image registry.

Note

Check for the occurence of any "CrashLoopBackOff | ImagePullFailed | Error" issues. **`oc logs <pod>`**, **`oc describe pod <pod>`** and **`oc get event`** are helpful diagnostic commands.

29.3. SYSTEMD-JOURNALD AND RSYSLOG

Rate-limiting

In Red Hat Enterprise Linux (RHEL) 7 the **systemd-journald.socket** unit creates ***/dev/log*** during the boot process, and then passes input to **systemd-journald.service**. Every **syslog()** call goes to the journal.

Rsyslog uses the **imjournal** module as a default input mode for journal files. Refer to Interaction of rsyslog and journal for detailed information about this topic.

A simple test harness was developed, which uses logger across the cluster nodes to make entries of different sizes at different rates in the system log. During testing simulations under a default Red Hat Enterprise Linux (RHEL) 7 installation with **`systemd-219-19.el7.x86_64`** at certain logging rates (approximately 40 log lines per second), we encountered the default rate limit of **`rsyslogd`**. After adjusting these limits, entries stopped being written to journald due to local journal file corruption. This issue is resolved in later versions of systemd.

Scaling up

As you scale up your project, the default logging environment might need some adjustments. After updating to **systemd-219-22.el7.x86_64**, we added:

```
$IMUXSockRateLimitInterval 0
$IMJournalRatelimitInterval 0
```

to ***/etc/rsyslog.conf*** and:

```
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=5s
```

to ***/etc/systemd/journald.conf***.

Now, restart the services.

```
$ systemctl restart systemd-journald.service
$ systemctl restart rsyslog.service
```

These settings account for the bursty nature of uploading in bulk.

After removing the rate limit, you may see increased CPU utilization on the system logging daemons as it processes any messages that would have previously been throttled.

Rsyslog is configured (see **ratelimit.interval**, **ratelimit.burst**) to rate-limit entries read from the journal at 10,000 messages in 300 seconds. A good rule of thumb is to ensure that the rsyslog rate-limits account for the systemd-journald rate-limits.

29.4. SCALING UP EFK LOGGING

If you do not indicate the desired scale at first deployment, the least disruptive way of adjusting your cluster is by re-running the deployer with the updated **`ES_CLUSTER_SIZE`** value and using the **`MODE=reinstall`** template parameter. Refer to the Performing Administrative Elasticsearch Operations section for more in-depth information.

```
$ oc edit configmap logging-deployer
  [change es-cluster-size value to 5]

$ oc new-app logging-deployer-template --param MODE=reinstall
```

29.5. STORAGE CONSIDERATIONS

An Elasticsearch index is a collection of shards and its corresponding replica shards. This is how ES implements high availability internally, therefore there is little need to use hardware based mirroring RAID variants. RAID 0 can still be used to increase overall disk performance.

Every search request needs to hit a copy of every shard in the index. Each ES instance requires its own individual storage, but an OpenShift Container Platform deployment can only provide volumes shared by all of its pods, which again means that Elasticsearch shouldn't be implemented with a single node.

A persistent volume should be added to each Elasticsearch deployment configuration so that we have one volume per replica shard. On OpenShift Container Platform this is often achieved through Persistent Volume Claims

- 1 volume per shard
- 1 volume per replica shard

The PVCs must be named based on the **es-pvc-prefix** setting. Refer to Persistent Elasticsearch Storage for more details.

Below are capacity planning guidelines for OpenShift Container Platform aggregate logging.
Example scenario

Assumptions:

1. Which application: Apache
2. Bytes per line: 256

3. Lines per second load on application: 1
4. Raw text data → JSON

Baseline (256 characters per minute → 15KB/min)

Logging Infra Pods	Storage Throughput
3 es 1 kibana 1 curator 1 fluentd	6 pods total: 90000 x 86400 = 7,7 GB/day
3 es 1 kibana 1 curator 11 fluentd	16 pods total: 225000 x 86400 = 24,0 GB/day
3 es 1 kibana 1 curator 20 fluentd	25 pods total: 225000 x 86400 = 32,4 GB/day

Calculating total logging throughput and disk space required for your logging environment requires knowledge of your application. For example, if one of your applications on average logs 10 lines-per-second, each 256 bytes-per-line, calculate per-application throughput and disk space as follows:

```
 (bytes-per-line * (lines-per-second) = 2560 bytes per app per second
 (2560) * (number-of-pods-per-node,100) = 256,000 bytes per second per
node
 256k * (number-of-nodes) = total logging throughput per cluster
```

Fluentd ships any logs from ***/var/log/messages*** and ***/var/lib/docker/containers/*** to Elasticsearch. Learn more.

Local SSD drives are recommended in order to achieve the best performance. In Red Hat Enterprise Linux (RHEL) 7, the deadline IO scheduler is the default for all block devices except SATA disks. For SATA disks, the default IO scheduler is **cfq**.

Sizing storage for ES is greatly dependent on how you optimize your indices. Therefore, consider how much data you need in advance and that you are aggregating application log data.

CHAPTER 30. ENABLING CLUSTER METRICS

30.1. OVERVIEW

The kubelet exposes metrics that can be collected and stored in back-ends by Heapster.

As an OpenShift Container Platform administrator, you can view a cluster's metrics from all containers and components in one user interface. These metrics are also used by horizontal pod autoscalers in order to determine when and how to scale.

This topic describes using Hawkular Metrics as a metrics engine which stores the data persistently in a Cassandra database. When this is configured, CPU, memory and network-based metrics are viewable from the OpenShift Container Platform web console and are available for use by horizontal pod autoscalers.

Heapster retrieves a list of all nodes from the master server, then contacts each node individually through the **`/stats`** endpoint. From there, Heapster scrapes the metrics for CPU, memory and network usage, then exports them into Hawkular Metrics.

Browsing individual pods in the web console displays separate sparkline charts for memory and CPU. The time range displayed is selectable, and these charts automatically update every 30 seconds. If there are multiple containers on the pod, then you can select a specific container to display its metrics.

If resource limits are defined for your project, then you can also see a donut chart for each pod. The donut chart displays usage against the resource limit. For example: **`145 Available of 200 MiB`**, with the donut chart showing **`55 MiB Used`**.

30.2. BEFORE YOU BEGIN

The components for cluster metrics must be deployed to the **openshift-infra** project. This allows horizontal pod autoscalers to discover the Heapster service and use it to retrieve metrics that can be used for autoscaling.

All of the following commands in this topic must be executed under the **openshift-infra** project. To switch to the **openshift-infra** project:

```
$ oc project openshift-infra
```

To enable cluster metrics, you must next configure the following:

- Service Accounts
- Metrics Data Storage
- Metrics Deployer

30.3. SERVICE ACCOUNTS

You must configure service accounts for:

- Metrics Deployer
- Heapster

30.3.1. Metrics Deployer Service Account

The Metrics Deployer will be discussed in a later step, but you must first set up a service account for it:

1. Create a **metrics-deployer** service account:

```
$ oc create -f - <<API
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metrics-deployer
secrets:
- name: metrics-deployer
API
```

2. Before it can deploy components, the **metrics-deployer** service account must also be granted the **edit** permission for the **openshift-infra** project:

```
$ oadm policy add-role-to-user \
    edit system:serviceaccount:openshift-infra:metrics-deployer
```

30.3.2. Heapster Service Account

The Heapster component requires access to the master server to list all available nodes and access the **/stats** endpoint for each node. Before it can do this, the Heapster service account requires the **cluster-reader** permission:

```
$ oadm policy add-cluster-role-to-user \
    cluster-reader system:serviceaccount:openshift-infra:heapster
```

Note

The Heapster service account is created automatically during the Deploying the Metrics Components step.

30.4. METRICS DATA STORAGE

You can store the metrics data to either persistent storage or to a temporary pod volume.

30.4.1. Persistent Storage

Running OpenShift Container Platform cluster metrics with persistent storage means that your metrics will be stored to a persistent volume and be able to survive a pod being restarted or recreated. This is ideal if you require your metrics data to be guarded from data loss. For production environments it is highly recommended to configure persistent storage for your metrics pods.

The size of the persisted volume can be specified with the **CASSANDRA_PV_SIZE**template parameter. By default it is set to 10 GB, which may or may not be sufficient for the size of the cluster you are using. If you require more space, for instance 100 GB, you could specify it with something like this:

```
$ oc process -f metrics-deployer.yaml \
    -v HAWKULAR_METRICS_HOSTNAME=hawkular-metrics.example.com \
    -v CASSANDRA_PV_SIZE=100Gi \
    | oc create -f -
```

The size requirement of the Cassandra storage is dependent on the number of pods. It is the administrator's responsibility to ensure that the size requirements are sufficient for their setup and to monitor usage to ensure that the disk does not become full.

If you would like to use dynamically provisioned persistent volumes you must set the **DYNAMICALLY_PROVISION_STORAGE**template option to **true** for the Metrics Deployer.

30.4.2. Capacity Planning for Cluster Metrics

After the metrics deployer runs, the output of **oc get pods** should resemble the following:

```
 # oc get pods -n openshift-infra
 NAME                                READY              STATUS
RESTARTS          AGE
 hawkular-cassandra-1-l5y4g          1/1                Running     0
17h
 hawkular-metrics-1t9so              1/1                Running     0
17h
 heapster-febru                      1/1                Running     0
17h
```

OpenShift Container Platform metrics are stored using the Cassandra database, which is deployed with settings of **MAX_HEAP_SIZE=512M** and **NEW_HEAP_SIZE=100M**. These values should cover most OpenShift Container Platform metrics installations, but you can modify them in the Cassandra Dockerfile prior to deploying cluster metrics.

By default, metrics data is stored for 7 days. You can configure this with the **METRIC_DURATION** parameter in the ***metrics.yaml*** configuration file. After seven days, Cassandra begins to purge the oldest metrics data. Metrics data for deleted pods and projects is not automatically purged; it is only removed once the data is over seven days old.

Warning

If the Cassandra persisted volume runs out of sufficient space, then data loss will occur.

For cluster metrics to work with persistent storage, ensure that the persistent volume has the **ReadWriteOnce** access mode. If this mode is not active, then the persistent volume claim cannot locate the persistent volume, and Cassandra fails to start.

To use persistent storage with the metric components, ensure that a persistent volume of sufficient size is available. The creation of persistent volume claims is handled by the Metrics Deployer.

OpenShift Container Platform metrics also supports dynamically-provisioned persistent volumes. To use this feature with OpenShift Container Platform metrics, it is necessary to add an additional flag to the metrics-deployer: **DYNAMICALLY_PROVISION_STORAGE=true**. You can use EBS, GCE, and Cinder storage back-ends to dynamically provision persistent volumes.

In tests performed with with 210 and 990 OpenShift Container Platform nodes, where 10500 pods and 11000 pods were monitored respectively, the Cassandra database grew at the speed shown in the table below:

Table 30.1. Cassandra Database storage requirements based on number of nodes/pods in the cluster

Number of Nodes	Number of Pods	Cassandra Storage growth speed	Cassandra storage growth per day	Cassandra storage growth per week
210	10500	500 MB per hour	15 GB	75 GB
990	11000	1 GB per hour	30 GB	210 GB

In the above calculation, approximately 20% of the expected size was added as overhead to ensure that the storage requirements do not exceed calculated value.

If the **`METRICS_DURATION`** and **`METRICS_RESOLUTION`** values are kept at the default (**`7`** days and **`15`** seconds respectively), it is safe to plan Cassandra storage size requrements for week, as in the values above.

> **Warning**
>
> Because OpenShift Container Platform metrics uses the Cassandra database as a datastore for metrics data, if **`USE_PERSISTANT_STORAGE=true`** is set during the metrics set up process, **`PV`** will be on top in the network storage, with NFS as the default. However, using network storage in combination with Cassandra is not recommended, as per the Cassandra documentation.

Known Issues and Limitations

Testing found that the **`heapster`** metrics component is capable of handling up to 12000 pods. If the amount of pods exceed that number, Heapster begins to fall behind in metrics processing, resulting in the possibility of metrics graphs no longer appearing. Work is ongoing to increase the number of pods that Heapster can gather metrics on, as well as upstream development of alternate metrics-gathering solutions.

30.4.3. Non-Persistent Storage

Running OpenShift Container Platform cluster metrics with non-persistent storage means that any stored metrics will be deleted when the pod is deleted. While it is much easier to run cluster metrics with non-persistent data, running with non-persistent data does come with the risk of permanent data loss. However, metrics can still survive a container being restarted.

In order to use non-persistent storage, you must set the **`USE_PERSISTENT_STORAGE`**template option to **`false`** for the Metrics Deployer.

Note

When using non-persistent storage, metrics data will be written to ***/var/lib/origin/openshift.local.volumes/pods*** on the node where the Cassandra pod is running. Ensure ***/var*** has enough free space to accommodate metrics storage.

30.5. METRICS DEPLOYER

The Metrics Deployer deploys and configures all of the metrics components. You can configure it by passing in information from secrets and by passing parameters to the Metrics Deployer's template.

30.5.1. Using Secrets

The Metrics Deployer will auto-generate self-signed certificates for use between its components and will generate a re-encrypting route to expose the Hawkular Metrics service. This route is what allows the web console to access the Hawkular Metrics service.

In order for the browser running the web console to trust the connection through this route, it must trust the route's certificate. This can be accomplished by providing your own certificates signed by a trusted Certificate Authority. The Metric Deployer's secret allows you to pass your own certificates which it will then use when creating the route.

If you do not wish to provide your own certificates, the router's default certificate will be used instead.

30.5.1.1. Providing Your Own Certificates

To provide your own certificate which will be used by the re-encrypting route, you can pass these values as secrets to the Metrics Deployer.

The **`hawkular-metrics.pem`** value needs to contain the certificate in its ***.pem*** format. You may also need to provide the certificate for the Certificate Authority which signed this ***pem*** file via the **`hawkular-metrics-ca.cert`** secret.

```
$ oc secrets new metrics-deployer \
    hawkular-metrics.pem=/home/openshift/metrics/hm.pem \
    hawkular-metrics-ca.cert=/home/openshift/metrics/hm-ca.cert
```

When these secrets are provided, the deployer uses these values to specify the **`key`**, **`certificate`** and **`caCertificate`** values for the re-encrypting route it generated.

For more information, please see the re-encryption route documentation.

30.5.1.2. Using the Router's Default Certificate

If the **`hawkular-metrics.pem`** value is not specified, the re-encrypting route will use the router's default certificate, which may not be trusted by browsers.

A secret named **metrics-deployer** will still be required in this situation. This can be considered a "dummy" secret since the secret it specifies is not actually used by the component.

To create a "dummy" secret that does not specify a certificate value:

```
$ oc secrets new metrics-deployer nothing=/dev/null
```

30.5.1.3. Deployer Secret Options

The following table contains more advanced configuration options, detailing all the secrets which can be used by the deployer:

Secret Name	Description
hawkular-metrics.pem	The ***pem*** file to use for the Hawkular Metrics certificate used for the re-encrypting route. This certificate **must** contain the host name used by the route (e.g., **`HAWKULAR_METRICS_HOSTNAME`**). The default router's certificate is used for the route if this option is unspecified.
hawkular-metrics-ca.cert	The certificate for the CA used to sign the ***hawkular-metrics.pem***. This is optional if the ***hawkular-metrics.pem*** does not contain the CA certificate directly.
heapster.cert	The certificate for Heapster to use. This is auto-generated if unspecified.
heapster.key	The key to use with the Heapster certificate. This is ignored if ***heapster.cert*** is not specified
heapster_client_ca.cert	The certificate that generates ***heapster.cert***. This is required if ***heapster.cert*** is specified. Otherwise, the main CA for the OpenShift Container Platform installation is used. In order for horizontal pod autoscaling to function properly, this should not be overridden.
heapster_allowed_users	A file containing a comma-separated list of CN to accept from certificates signed with the specified CA. By default, this is set to allow the OpenShift Container Platform service proxy to connect. If you override this, make sure to add **`system:master-proxy`** to the list in order to allow horizontal pod autoscaling to function properly.

30.5.2. Modifying the Deployer Template

The OpenShift Container Platform installer uses a template to deploy the metrics components. The default template can be found at the following path:

```
/usr/share/ansible/openshift-
ansible/roles/openshift_hosted_templates/files/v1.4/enterprise/metrics-
deployer.yaml
```

In case you need to make any changes to this file, copy it to another directory with the file name ***metrics-deployer.yaml*** and refer to the new location when using it in the following sections.

30.5.2.1. Deployer Template Parameters

The deployer template parameter options and their defaults are listed in the default ***metrics-deployer.yaml*** file. If required, you can override these values when creating the Metrics Deployer.

Table 30.2. Template Parameters

Parameter	Description
METRIC_DURATION	The number of days metrics should be stored.
CASSANDRA_PV_SIZE	The persistent volume size for each of the Cassandra nodes.
CASSANDRA_NODES	The number of initial Cassandra nodes to deploy.
USE_PERSISTENT_STORAGE	Set to **true** for persistent storage; set to **false** to use non-persistent storage.
DYNAMICALLY_PROVISION_STORAGE	Set to **true** to allow for dynamically provisioned storage.
REDEPLOY	If set to **true**, the deployer will try to delete all the existing components before trying to redeploy.
HAWKULAR_METRICS_HOSTNAME	External host name where clients can reach Hawkular Metrics. This is the FQDN of the machine running the router pod.
MASTER_URL	Internal URL for the master, for authentication retrieval.
IMAGE_VERSION	Specify version for metrics components. For example, for **openshift/origin-metrics-deployer:latest**, set version to **latest**.

Parameter	Description
IMAGE_PREFIX	Specify prefix for metrics components. For example, for **openshift/origin-metrics-deployer:latest**, set prefix to **openshift/origin-**.
MODE	Can be set to: - **preflight** to perform validation before a deployment. - **deploy** to perform an initial deployment. - **refresh** to delete and redeploy all components but to keep persisted data and routes. - **redeploy** to delete and redeploy everything (losing all data in the process). - **validate** to re-run validations after a deployment.
IGNORE_PREFLIGHT	Can be set to **true** to disable the preflight checks. This allows the deployer to continue even if the preflight check has failed.
USER_WRITE_ACCESS	Sets whether user accounts should be able to write metrics. Defaults to **false** so that only Heapster can write metrics and not individual users. It is recommended to disable user write access; if enabled, any user will be able to write metrics to the system which can affect performance and can cause Cassandra disk usage to unpredictably increase.

The only required parameter is **HAWKULAR_METRICS_HOSTNAME**. This value is required when creating the deployer, because it specifies the host name for the Hawkular Metrics route. This value should correspond to a fully qualified domain name. You must know the value of **HAWKULAR_METRICS_HOSTNAME** when configuring the console for metrics access.

If you are using persistent storage with Cassandra, it is the administrator's responsibility to set a sufficient disk size for the cluster using the **CASSANDRA_PV_SIZE** parameter. It is also the administrator's responsibility to monitor disk usage to make sure that it does not become full.

Warning

Data loss will result if the Cassandra persisted volume runs out of sufficient space.

All of the other parameters are optional and allow for greater customization. For instance, if you

have a custom install in which the Kubernetes master is not available under *https://kubernetes.default.svc:443* you can specify the value to use instead with the **MASTER_URL** parameter. To deploy a specific version of the metrics components, use the **IMAGE_VERSION** parameter.

Warning

It is highly recommended to not use **latest** for the **IMAGE_VERSION**. The **latest** version corresponds to the very latest version available and can cause issues if it brings in a newer version not meant to function on the version of OpenShift Container Platform you are currently running.

30.6. DEPLOYING THE METRIC COMPONENTS

Because deploying and configuring all the metric components is handled by the Metrics Deployer, you can simply deploy everything in one step.

The following examples show you how to deploy metrics with and without persistent storage using the default template parameters. Optionally, you can specify any of the template parameters when calling these commands.

Example 30.1. Deploying with Persistent Storage

The following command sets the Hawkular Metrics route to use **hawkular-metrics.example.com** and is deployed using persistent storage.

You must have a persistent volume of sufficient size available.

```
$ oc new-app --as=system:serviceaccount:openshift-infra:metrics-
deployer \
    -f metrics-deployer.yaml \
    -p HAWKULAR_METRICS_HOSTNAME=hawkular-metrics.example.com
```

Example 30.2. Deploying without Persistent Storage

The following command sets the Hawkular Metrics route to use **hawkular-metrics.example.com** and deploy without persistent storage.

```
$ oc new-app --as=system:serviceaccount:openshift-infra:metrics-
deployer \
    -f metrics-deployer.yaml \
    -p HAWKULAR_METRICS_HOSTNAME=hawkular-metrics.example.com \
    -p USE_PERSISTENT_STORAGE=false
```

Warning

Because this is being deployed without persistent storage, metric data loss can occur.

30.6.1. Metrics Deployer Validations

The metrics deployer does some validation both before and after deployment. If the pre-flight validation fails, the environment for deployment is considered unsuitable and the deployment is aborted. However, you can add **`IGNORE_PREFLIGHT=true`** to the deployer parameters if you believe the validation has erred.

If post-deployment validation fails, the deployer finishes in an **Error** state, which indicates that you should check the deployer logs for issues that may require addressing. For example, the validation may detect that the external **hawkular-metrics** route is not actually in use, because the route was already created somewhere else. The validation output at the end of a deployment should explain as clearly as possible any issues it finds and what you can do to address them.

Once you have addressed deployment validation issues, you can re-run just the validation by running the deployer again with the **`MODE=validate`** parameter added, for example:

```
$ oc new-app --as=system:serviceaccount:openshift-infra:metrics-
deployer \
    -f metrics-deployer.yaml \
    -p HAWKULAR_METRICS_HOSTNAME=hawkular-metrics.example.com \
    -p MODE=validate
```

There is also a diagnostic for metrics:

```
$ oadm diagnostics MetricsApiProxy
```

30.7. SETTING THE METRICS PUBLIC URL

The OpenShift Container Platform web console uses the data coming from the Hawkular Metrics service to display its graphs. The URL for accessing the Hawkular Metrics service must be configured via the **`metricsPublicURL`** option in the master configuration file (***/etc/origin/master/master-config.yaml***). This URL corresponds to the route created with the **`HAWKULAR_METRICS_HOSTNAME`** template parameter during the deployment of the metrics components.

Note

You must be able to resolve the **`HAWKULAR_METRICS_HOSTNAME`** from the browser accessing the console.

For example, if your **`HAWKULAR_METRICS_HOSTNAME`** corresponds to **`hawkular-metrics.example.com`**, then you must make the following change in the ***master-config.yaml*** file:

```
  assetConfig:
    ...
```

```
    metricsPublicURL: "https://hawkular-
metrics.example.com/hawkular/metrics"
```

Once you have updated and saved the ***master-config.yaml*** file, you must restart your OpenShift Container Platform instance.

When your OpenShift Container Platform server is back up and running, metrics will be displayed on the pod overview pages.

Caution

If you are using self-signed certificates, remember that the Hawkular Metrics service is hosted under a different host name and uses different certificates than the console. You may need to explicitly open a browser tab to the value specified in **`metricsPublicURL`** and accept that certificate.

To avoid this issue, use certificates which are configured to be acceptable by your browser.

30.8. ACCESSING HAWKULAR METRICS DIRECTLY

To access and manage metrics more directly, use the Hawkular Metrics API.

Note

When accessing Hawkular Metrics via the API, you will only be able to perform reads. Writing metrics has been disabled by default. If you want for individual users to also be able to write metrics, you must set the **`USER_WRITE_ACCESS`**deployer template parameter to **true**.

However, it is recommended to use the default configuration and only have metrics enter the system via Heapster. If write access is enabled, any user will be able to write metrics to the system, which can affect performance and cause Cassandra disk usage to unpredictably increase.

The Hawkular Metrics documentation covers how to use the API, but there are a few differences when dealing with the version of Hawkular Metrics configured for use on OpenShift Container Platform:

30.8.1. OpenShift Container Platform Projects and Hawkular Tenants

Hawkular Metrics is a multi-tenanted application. It is configured so that a project in OpenShift Container Platform corresponds to a tenant in Hawkular Metrics.

As such, when accessing metrics for a project named **MyProject** you must set the **Hawkular-Tenant** header to **MyProject**.

There is also a special tenant named **_system** which contains system level metrics. This requires either a **cluster-reader** or **cluster-admin** level privileges to access.

30.8.2. Authorization

The Hawkular Metrics service will authenticate the user against OpenShift Container Platform to

determine if the user has access to the project it is trying to access.

Hawkular Metrics accepts a bearer token from the client and verifies that token with the OpenShift Container Platform server using a **SubjectAccessReview**. If the user has proper read privileges for the project, they are allowed to read the metrics for that project. For the **_system** tenant, the user requesting to read from this tenant must have **cluster-reader** permission.

When accessing the Hawkular Metrics API, you must pass a bearer token in the **Authorization** header.

30.9. SCALING OPENSHIFT CONTAINER PLATFORM METRICS PODS

One set of metrics pods (Cassandra/Hawkular/Heapster) is able to monitor at least 10,000 pods.

Caution

Pay attention to system load on nodes where OpenShift Container Platform metrics pods run. Use that information to determine if it is necessary to scale out a number of OpenShift Container Platform metrics pods and spread the load across multiple OpenShift Container Platform nodes. Scaling OpenShift Container Platform metrics heapster pods is not recommended.

30.9.1. Prerequisites

If persistent storage was used to deploy OpenShift Container Platform metrics, then you must create a persistent volume (PV) for the new Cassandra pod to use before you can scale out the number of OpenShift Container Platform metrics Cassandra pods. However, if Cassandra was deployed with dynamically provisioned PVs, then this step is not necessary.

30.9.2. Scaling the Cassandra Components

The Cassandra nodes use persistent storage, therefore scaling up or down is not possible with replication controllers.

Scaling a Cassandra cluster requires you to use the **`hawkular-cassandra-node`** template. By default, the Cassandra cluster is a single-node cluster.

To scale out the number of OpenShift Container Platform metrics hawkular pods to two replicas, run:

```
# oc scale -n openshift-infra --replicas=2 rc hawkular-metrics
```

Note

If you add a new node to a Cassandra cluster, the data stored in the cluster rebalances across the cluster. The same thing happens If you remove a node from the Cluster.

30.10. HORIZONTAL POD AUTOSCALING

OpenShift Container Platform version 3.3 does not provide Horizontal Pod Autoscaling (HPA) support for metrics pods and scaling metrics pods.

30.11. CLEANUP

You can remove everything deloyed by the metrics deployer by performing the following steps:

```
$ oc delete all,sa,templates,secrets,pvc --selector="metrics-infra"
```

To remove the deployer components, perform the following steps:

```
$ oc delete sa,secret metrics-deployer
```

CHAPTER 31. CUSTOMIZING THE WEB CONSOLE

31.1. OVERVIEW

Administrators can customize the web console using extensions, which let you run scripts and load custom stylesheets when the web console loads. Extension scripts allow you to override the default behavior of the web console and customize it for your needs.

For example, extension scripts can be used to add your own company's branding or to add company-specific capabilities. A common use case for this is rebranding or white-labelling for different environments. You can use the same extension code, but provide settings that change the web console. You can change the look and feel of nearly any aspect of the user interface in this way.

31.2. LOADING EXTENSION SCRIPTS AND STYLESHEETS

To add scripts and stylesheets, edit the master configuration file. The scripts and stylesheet files must exist on the Asset Server and are added with the following options:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/script1.js
    - /path/to/script2.js
    - ...
  extensionStylesheets:
    - /path/to/stylesheet1.css
    - /path/to/stylesheet2.css
    - ...
```

Relative paths are resolved relative to the master configuration file. To pick up configuration changes, restart the server.

Custom scripts and stylesheets are read once at server start time. To make developing extensions easier, you can reload scripts and stylesheets on every request by enabling development mode with the following setting:

```
assetConfig:
  ...
  extensionDevelopment: true
```

When set, the web console reloads any changes to existing extension script or stylesheet files when you refresh the page in your browser. You still must restart the server when adding new extension stylesheets or scripts, however. This setting is only recommended for testing changes and not for production.

The examples in the following sections show common ways you can customize the web console.

Note

Additional extension examples are available in the OpenShift Origin repository on GitHub.

31.2.1. Setting Extension Properties

If you have a specific extension, but want to use different text in it for each of the environments, you can define the environment in the ***master-config.yaml*** file, and use the same extension script across environments. Pass settings from the ***master-config.yaml*** file to be used by the extension using the **`extensionProperties`** mechanism:

```
assetConfig:
  extensionDevelopment: true
  extensionProperties:
    doc_url: https://docs.openshift.com
    key1: value1
    key2: value2
  extensionScripts:
```

This results in a global variable that can be accessed by the extension, as if the following code was executed:

```
window.OPENSHIFT_EXTENSION_PROPERTIES = {
  doc_url: "https://docs.openshift.com",
  key1: "value1",
  key2: "value2",
}
```

31.2.2. Customizing the Logo

The following style changes the logo in the web console header:

```
#header-logo {
  background-image: url("https://www.example.com/images/logo.png");
  width: 190px;
  height: 20px;
}
```

Replace the **example.com** URL with a URL to an actual image, and adjust the width and height. The ideal height is **20px**.

Save the style to a file (for example, ***logo.css***) and add it to the master configuration file:

```
assetConfig:
  ...
  extensionStylesheets:
    - /path/to/logo.css
```

31.2.3. Changing Links to Documentation

Links to external documentation are shown in various sections of the web console. The following example changes the URL for two given links to the documentation:

```
window.OPENSHIFT_CONSTANTS.HELP['get_started_cli']      =
"https://example.com/doc1.html";
window.OPENSHIFT_CONSTANTS.HELP['basic_cli_operations'] =
"https://example.com/doc2.html";
```

Save this script to a file (for example, ***help-links.js***) and add it to the master configuration file:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/help-links.js
```

31.2.4. Adding or Changing Links to Download the CLI

The **About** page in the web console provides download links for the command line interface (CLI) tools. These links can be configured by providing both the link text and URL, so that you can choose to point them directly to file packages, or to an external page that points to the actual packages.

For example, to point directly to packages that can be downloaded, where the link text is the package platform:

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Linux (32 bits)": "https://<cdn>/openshift-client-tools-linux-
32bit.tar.gz",
  "Linux (64 bits)": "https://<cdn>/openshift-client-tools-linux-
64bit.tar.gz",
  "Windows":         "https://<cdn>/openshift-client-tools-
windows.zip",
  "Mac OS X":        "https://<cdn>/openshift-client-tools-mac.zip"
};
```

Alternatively, to point to a page that links the actual download packages, with the **Latest Release** link text:

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Latest Release": "https://<cdn>/openshift-client-tools/latest.html"
};
```

Save this script to a file (for example, ***cli-links.js***) and add it to the master configuration file:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/cli-links.js
```

31.2.5. Customizing the About Page

To provide a custom **About** page for the web console:

1. Write an extension that looks like:

   ```
   angular
   ```

```
.module('aboutPageExtension', ['openshiftConsole'])
.config(function($routeProvider) {
  $routeProvider
    .when('/about', {
      templateUrl: 'extensions/about/about.html',
      controller: 'AboutController'
    });
  }
);
```

2. Save the script to a file (for example, ***about/about.js***).

3. Write a customized template.

    a. Start from the version of ***about.html*** from the OpenShift Container Platform release you are using. Within the template, there are two angular scope variables available: **`version.master.openshift`** and **`version.master.kubernetes`**.

    b. Save the custom template to a file (for example, ***about/about.html***).

    c. Modify the master configuration file:

```
assetConfig:
  ...
  extensionScripts:
    - about/about.js
  ...
  extensions:
    - name: about
      sourceDirectory: /path/to/about
```

31.2.6. Configuring Navigation Menus

31.2.6.1. Top Navigation Dropdown Menus

The top navigation bar of the web console contains the help icon and the user dropdown menus. You can add additional menu items to these using the angular-extension-registry.

The available extension points are:

- **`nav-help-dropdown`** - the help icon dropdown menu, visible at desktop screen widths
- **`nav-user-dropdown`** - the user dropdown menu, visible at desktop screen widths
- **`nav-dropdown-mobile`** - the single menu for top navigation items at mobile screen widths

The following example extends the **`nav-help-dropdown`** menu, with a name of **`<myExtensionModule>`**:

Note

`<myExtensionModule>` is a placeholder name. Each dropdown menu extension must be unique enough so that it does not clash with any future angular modules.

```
angular
  .module('<myExtensionModule>', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      extensionRegistry
        .add('nav-help-dropdown', function() {
          return [
            {
              type: 'dom',
              node: '<li><a href="http://www.example.com/report"
target="_blank">Report a Bug</a></li>'
            }, {
              type: 'dom',
              node: '<li class="divider"></li>'  // If you want a
horizontal divider to appear in the menu
            }, {
              type: 'dom',
              node: '<li><a href="http://www.example.com/status"
target="_blank">System Status</a></li>'
            }
          ];
        });
    }
  ]);

hawtioPluginLoader.addModule('<myExtensionModule>');
```

31.2.6.2. Project Left Navigation

When navigating within a project, a menu appears on the left with primary and secondary navigation. This menu structure is defined as a constant and can be overridden or modified.

Note

Significant customizations to the project navigation may affect the user experience and should be done with careful consideration. You may need to update this customization in future upgrades if you modify existing navigation items.

1. Create the configuration scripts within a file (for example, ***navigation.js***):

```
// Append a new primary nav item.  This is a simple direct
navigation item
// with no secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.push({
  label: "Dashboard",           // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "/dashboard"            // Where to go when this nav
item is clicked.
                                // Relative URLs are pre-pended
with the path
                                // '/project/<project-name>'
});
```

```
// Splice a primary nav item to a specific spot in the list.
This primary item has
// a secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.splice(2, 0, { //
Insert at the third spot
  label: "Git",
  iconClass: "fa fa-code",
  secondaryNavSections: [       // Instead of an href, a sub-
menu can be defined
    {
      items: [
        {
          label: "Branches",
          href: "/git/branches",
          prefixes: [
            "/git/branches/"     // Defines prefix URL patterns
that will cause
                                 // this nav item to show the
active state, so
                                 // tertiary or lower pages
show the right context
          ]
        }
      ]
    },
    {
      header: "Collaboration",   // Sections within a sub-menu
can have an optional header
      items: [
        {
          label: "Pull Requests",
          href: "/git/pull-requests",
          prefixes: [
            "/git/pull-requests/"
          ]
        }
      ]
    }
  ]
});

// Add a primary item to the top of the list.  This primary item
is shown conditionally.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.unshift({
  label: "Getting Started",
  iconClass: "pficon pficon-screen",
  href: "/getting-started",
  prefixes: [                   // Primary nav items can also
specify prefixes to trigger
    "/getting-started/"         // active state
  ],
  isValid: function() {         // Primary or secondary items
can define an isValid
    return isNewUser;           // function. If present it will
be called to test whether
```

```
                                    // the item should be shown, it
should return a boolean
  }
});

// Modify an existing menu item
var applicationsMenu =
_.find(window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION, { label:
'Applications' });
applicationsMenu.secondaryNavSections.push({ // Add a new
secondary nav section to the Applications menu
  // my secondary nav section
});
```

2. Save the file and add it to the master configuration at ***/etc/origin/master/master-config.yml***:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/navigation.js
```

3. Restart the master host:

```
# systemctl restart atomic-openshift-master
```

31.2.7. Configuring Catalog Categories

Catalog categories organize the display of builder images and templates on the **Add to Project** page on the OpenShift Container Platform web console. A builder image or template is grouped in a category if it includes a tag with the same name of the category or category alias. Categories only display if one or more builder images or templates with matching tags are present in the catalog.

Note

Significant customizations to the catalog categories may affect the user experience and should be done with careful consideration. You may need to update this customization in future upgrades if you modify existing category items.

1. Create the following configuration scripts within a file (for example, ***catalog-categories.js***):

```
// Add Go to the Languages category
var category =
_.find(window.OPENSHIFT_CONSTANTS.CATALOG_CATEGORIES,
{ id: 'languages' });
category.items.splice(2,0,{ // Insert at the third spot
  // Required.  Must be unique
  id: "go",
  // Required
  label: "Go",
  // Optional.  If specified, defines a unique icon for this item
  iconClass: "font-icon icon-go-gopher",
```

```
  // Optional.  If specified, enables matching other tag values
to this category
  // item
  categoryAliases: [
    "golang"
  ]
});

// Add a Featured category section at the top of the catalog
window.OPENSHIFT_CONSTANTS.CATALOG_CATEGORIES.unshift({
  // Required.  Must be unique
  id: "featured",
  // Required
  label: "Featured",
  // Optional.  If specified, each item in the category will
utilize this icon
  // as a default
  iconClassDefault: "fa fa-code",
  items: [
    {
      // Required.  Must be unique
      id: "go",
      // Required
      label: "Go",
      // Optional.  If specified, defines a unique icon for this
item
      iconClass: "font-icon icon-go-gopher",
      // Optional.  If specified, enables matching other tag
values to this
      // category item
      categoryAliases: [
        "golang"
      ],
      // Optional.  If specified, will display below the item
label
      description: "An open source programming language developed
at Google in
      2007 by Robert Griesemer, Rob Pike, and Ken Thompson."
    },
    {
      // Required.  Must be unique
      id: "jenkins",
      // Required
      label: "Jenkins",
      // Optional.  If specified, defines a unique icon for this
item
      iconClass: "font-icon icon-jenkins",
      // Optional.  If specified, will display below the item
label
      description: "An open source continuous integration tool
written in Java."
    }
  ]
});
```

2. Save the file and add it to the master configuration at ***/etc/origin/master/master-config.yml***:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/catalog-categories.js
```

3. Restart the master host:

```
# systemctl restart atomic-openshift-master
```

31.2.8. Enabling Features in Technology Preview

Sometimes features are available in Technology Preview. By default, these features are disabled and hidden in the web console.

Currently, there are no web console features in Technology Preview.

To enable a Technology Preview feature:

1. Save this script to a file (for example, ***tech-preview.js***):

```
window.OPENSHIFT_CONSTANTS.ENABLE_TECH_PREVIEW_FEATURE.
<feature_name> = true;
```

2. Add it to the master configuration file:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/tech-preview.js
```

31.3. SERVING STATIC FILES

You can serve other files from the Asset Server as well. For example, you might want to make the CLI executable available for download from the web console or add images to use in a custom stylesheet.

Add the directory with the files you want using the following configuration option:

```
assetConfig:
  ...
  extensions:
    - name: images
      sourceDirectory: /path/to/my_images
```

The files under the ***/path/to/my_images*** directory will be available under the URL */<context>/extensions/images* in the web console.

To reference these files from a stylesheet, you should generally use a relative path. For example:

```
#header-logo {
  background-image: url("../extensions/images/my-logo.png");
}
```

31.3.1. Enabling HTML5 Mode

The web console has a special mode for supporting certain static web applications that use the HTML5 history API:

```
assetConfig:
  ...
  extensions:
    - name: my_extension
      sourceDirectory: /path/to/myExtension
      html5Mode: true
```

Setting **`html5Mode`** to **true** enables two behaviors:

1. Any request for a non-existent file under ***/<context>/extensions/my_extension/*** instead serves ***/path/to/myExtension/index.html*** rather than a "404 Not Found" page.
2. The element **`<base href="/">`** will be rewritten in ***/path/to/myExtension/index.html*** to use the actual base depending on the asset configuration; only this exact string is rewritten.

This is needed for JavaScript frameworks such as AngularJS that require **`base`** to be set in ***index.html***.

31.4. CUSTOMIZING THE LOGIN PAGE

You can also change the login page, and the login provider selection page for the web console. Run the following commands to create templates you can modify:

```
$ oadm create-login-template > login-template.html
$ oadm create-provider-selection-template > provider-selection-
template.html
```

Edit the file to change the styles or add content, but be careful not to remove any required parameters inside the curly brackets.

To use your custom login page or provider selection page, set the following options in the master configuration file:

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.html
    providerSelection: /path/to/provider-selection-template.html
```

Relative paths are resolved relative to the master configuration file. You must restart the server after changing this configuration.

When there are multiple login providers configured or when the **alwaysShowProviderSelection** option in the ***master-config.yaml*** file is set to **true**, each time a user's token to OpenShift Container Platform expires, the user is presented with this custom page before they can proceed with other tasks.

31.4.1. Example Usage

Custom login pages can be used to create Terms of Service information. They can also be helpful if you use a third-party login provider, like GitHub or Google, to show users a branded page that they trust and expect before being redirected to the authentication provider.

31.5. CUSTOMIZING THE OAUTH ERROR PAGE

When errors occur during authentication, you can change the page shown.

1. Run the following command to create a template you can modify:

```
$ oadm create-error-template > error-template.html
```

2. Edit the file to change the styles or add content.

   You can use the **Error** and **ErrorCode** variables in the template. To use your custom error page, set the following option in the master configuration file:

```
oauthConfig:
  ...
  templates:
    error: /path/to/error-template.html
```

   Relative paths are resolved relative to the master configuration file.

3. You must restart the server after changing this configuration.

31.6. CHANGING THE LOGOUT URL

You can change the location a console user is sent to when logging out of the console by modifying the **logoutURL** parameter in the ***/etc/origin/master/master-config.yaml*** file:

```
...
assetConfig:
  logoutURL: "http://www.example.com"
...
```

This can be useful when authenticating with Request Header and OAuth or OpenID identity providers, which require visiting an external URL to destroy single sign-on sessions.

31.7. CONFIGURING WEB CONSOLE CUSTOMIZATIONS WITH ANSIBLE

During advanced installations, many modifications to the web console can be configured using the following parameters, which are configurable in the inventory file:

- **openshift_master_logout_url**
- **openshift_master_extension_scripts**
- **openshift_master_extension_stylesheets**
- **openshift_master_extensions**
- **openshift_master_oauth_template**
- **openshift_master_metrics_public_url**
- **openshift_master_logging_public_url**

Example Web Console Customization with Ansible

```
# Configure logoutURL in the master config for console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console
_customization.html#changing-the-logout-url
#openshift_master_logout_url=http://example.com

# Configure extensionScripts in the master config for console
customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console
_customization.html#loading-custom-scripts-and-stylesheets
#openshift_master_extension_scripts=
['/path/on/host/to/script1.js','/path/on/host/to/script2.js']

# Configure extensionStylesheets in the master config for console
customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console
_customization.html#loading-custom-scripts-and-stylesheets
#openshift_master_extension_stylesheets=
['/path/on/host/to/stylesheet1.css','/path/on/host/to/stylesheet2.css']

# Configure extensions in the master config for console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console
_customization.html#serving-static-files
#openshift_master_extensions=[{'name': 'images', 'sourceDirectory':
'/path/to/my_images'}]

# Configure extensions in the master config for console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console
_customization.html#serving-static-files
#openshift_master_oauth_template=/path/on/host/to/login-template.html

# Configure metricsPublicURL in the master config for cluster metrics.
Ansible is also able to configure metrics for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/cluster_met
rics.html
```

```
#openshift_master_metrics_public_url=https://hawkular-
metrics.example.com/hawkular/metrics

# Configure loggingPublicURL in the master config for aggregate
logging. Ansible is also able to install logging for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/aggregate_l
ogging.html
#openshift_master_logging_public_url=https://kibana.example.com
```

CHAPTER 32. REVISION HISTORY: INSTALLATION AND CONFIGURATION

32.1. THU FEB 16 2017

Affected Topic	Description of Change
Aggregating Container Logs	Fixed example in the Configuring Fluentd to Send Logs to an External Log Aggregator section.
Aggregating Container Logs	Added a version variable and <tag> to code block example in Deploying the EFK Stack section to display the correct current version to use.
Setting up a Router → Using the Default HAProxy Router	Added the Filtering Routes to Specific Routers section.
Setting up a Router → Using the F5 Router Plug-in	Arranged the topic and added a Prerequisites section.
Installing a Cluster→ Disconnected Isntallation	Added the cluster image to the Synching Images section.
Setting up the Registry → Deploying a Registry on Existing Clusters	Added additional URL instructions to the Deploying the Registry Console section.
Setting up the Registry → Securing and Exposing the Registry	Added a step for adding the public route host name in the **`--hostnames`** flag.
Configuring Persistent Storage → Volume Security	Added details about **RunAsAny** FSGroup and block device permissions.
Aggregating Container Logs	Fixed the options for creating a configuration file in the Creating New Configuration Files section.

32.2. THU FEB 09 2017

Affected Topic	Description of Change
Upgrading a Cluster → Manual In-Place Upgrades	
Upgrading a Cluster → Automated In-Place Upgrades	Added an Important admonition about an etcd performance issue.
Upgrading a Cluster → Blue-Green Deployments	

32.3. MON FEB 06 2017

Affected Topic	Description of Change
Upgrading a Cluster → Manual Upgrades	
Setting up the Registry → Deploying a Registry on Existing Clusters	Updated file paths to the logging and metrics deployer templates to their new OpenShift Container Platform 3.4 locations.
Aggregating Container Logs	
Enabling Cluster Metrics	
Installing a Cluster → Host Preparation	Added steps on using `yum-config-manager` to the host registration steps.
Installing a Cluster → Advanced Installation	Added the Configuring a Registry Location section.

Affected Topic	Description of Change
Setting up the Registry → Deploying a Registry on Existing Clusters	Added Important box about shutting down Cockpit to the Non-Production Use section.
	Arranged the Securing the Registry Console section to include information on the certificate.
Setting up a Router → Using the Default HAProxy Router	Arranged the topic to create the Creating a Router section, and added a paragraph on router options on creation.
Setting up a Router → Using the F5 Router Plug-in	Added a new Setting Up F5 Native Integration section.
Configuring Nuage SDN	Added the Configuring Nuage SDN topic.
Persistent Storage Examples → Complete Example Using GlusterFS	Added a link to the Complete Example of Dynamic Provisioning Using GlusterFS.
	Clarified that, if using a service, the endpoints name must match the service name.
Enabling Cluster Metrics	Updated Cluster Metrics sizing recommendations for the new version of OpenShift Container Platform.
Customizing the Web Console	Removed information about Pipelines being a feature in Technology Preview.

32.4. MON JAN 30 2017

Affected Topic	Description of Change
Setting up the Registry → Securing and Exposing the Registry	Removed references to the deprecated `--api-version` flag.

Affected Topic	Description of Change
Configuring Custom Certificates	Clarified custom certificate configuration locations in the Configuring Custom Certificates section.

32.5. WED JAN 25 2017

Affected Topic	Description of Change
Working with HTTP Proxies	Added step to Proxying Docker Pull for finding the registry service IP.
Setting up a Router → Using the F5 Router Plug-in	Removed references to the deprecated `--credentials` option.
Installing a Cluster → Prerequisites	Added information about required ports for Aggregated Logging.
Configuring Global Build Defaults and Overrides	Added notes to explain additional values in the ***/etc/origin/master/master-config.yaml*** file in the Manually Setting Global Build Defaults section.
Aggregate Logging Sizing Guidelines	Updated scale testing guidelines.
Customizing the Web Console	Added information about setting extension properties.
Configuring Persistent Storage → Dynamic Provisioning and Creating Storage Classes	Added additional details to the ***glusterfs-storageclass.yaml*** file example in the GlusterFS Object Definition section.
Master and Node Configuration	Updated the Audit Configuration section description and added Audit Configuration Parameters.

32.6. WED JAN 18 2017

OpenShift Container Platform 3.4 initial release.

Affected Topic	Description of Change
Installing a Cluster → Prerequisites	Added sizing guidelines for etcd service nodes within the Minimum Hardware Requirements table.
Setting up the Registry → Securing and Exposing the Registry	Added a NOTE box about mounting secrets to service accounts.
Setting up the Registry → Extended Registry Configuration	Updated with new **`oc rollout`** commands.
Setting up a Router → Using the Default HAProxy Router	Added the ARP Cache Tuning for Large-scale Clusters section.
	Added information about security caveats and ownership claims of host names and subdomains.
	Added a new Using Wildcard Routes (for a Subdomain) section.
Adding Hosts to an Existing Cluster	Updated quick installer instructions to use the new **`scaleup`** command.
Configuring Authentication and User Agent	Clarified the difference between **`/api`** and **`/oapi`** in the User Agent section.
Configuring for Azure	New topic on how OpenShift Container Platform can be configured to access an Azure infrastructure, including using Azure disk as persistent storage] for application data.
Configuring Persistent Storage → Persistent Storage Using Azure Disk	New topic on how to provision your OpenShift Container Platform cluster with persistent storage using Azure.

Affected Topic	Description of Change
Persistent Storage Examples → Complete Example of Dynamic Provisioning Using GlusterFS	New topic providing an end-to-end example of how to dynamically provision GlusterFS volumes.
Configuring Pipeline Execution	Added information about setting up an external Jenkins server.
Configuring Global Build Defaults and Overrides	Added examples for node selectors in build configurations.
Configuring Persistent Storage → Volume Security	Removed **`nfsnobody`** references.
Configuring Global Build Defaults and Overrides	Added image label example to build file examples.
Configuring Pipeline Execution	Noted that for **`jenkinsPipelineConfig`**, the **`autoProvisionEnabled`** value defaults to **`true`** if unspecified.
Aggregating Container Logs	Updated with new **`oc rollout`** commands.
	Added clarification regarding ConfigMaps and output of **`oc new-app`**.
Customizing the Web Console	Added the Configuring Catalog Categories section.