OpenShift Container Platform 3.11

Upgrading Clusters

OpenShift Container Platform 3.11 Upgrading Clusters

Last Updated: 2022-02-24

OpenShift Container Platform 3.11 Upgrading Clusters

OpenShift Container Platform 3.11 Upgrading Clusters

Legal Notice

Abstract

Upgrade your OpenShift Container Platform 3.11 cluster with this guide

Table of Contents

CHAPTER 1. UPGRADE METHODS AND STRATEGIES

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 3.10 to 3.11, and applying asynchronous errata updates within a minor version (3.11.z releases). See the OpenShift Container Platform 3.11 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.

The OpenShift Container Platform upgrade process uses Ansible playbooks to automate the tasks needed to upgrade your cluster. You must use the inventory file that you used during initial installation or during the last time that the upgrade was successful to run the upgrade playbook. Using this method allows you to choose between either upgrade strategy: in-place upgrades or blue-green deployments.

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

Before upgrading, ensure that all OpenShift Container Platform services are running well. In the event of a control plane upgrade failure, check the versions of your masters to ensure that all versions are the same. If your masters are all the same version, re-run the upgrade. If they differ, downgrade the masters to match the lower versioned master, then re-run the upgrade.

1.1. UPGRADE STRATEGIES

There are two strategies you can take for performing the OpenShift Container Platform cluster upgrade: in-place upgrades or blue-green deployments.

1.1.1. 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.

1.1.2. 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.

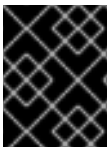
This method allows administrators to switch traffic from the old set of nodes (e.g., the "blue" deployment) to the new set (e.g., the "green" deployment) after the new deployment has been verified. If a problem is detected, it is also then easy to rollback to the old deployment quickly.

CHAPTER 2. PERFORMING AUTOMATED IN-PLACE CLUSTER UPGRADES

If you installed using the standard cluster installation process, and the inventory file that was used is available, you can use upgrade playbooks to automate the cluster upgrade process.

To upgrade OpenShift Container Platform, you run Ansible playbooks with the same inventory file that you used during installation. You run the same ***v3_11*** upgrade playbooks to:

- Upgrade existing OpenShift Container Platform version 3.10 clusters to version 3.11.
- Upgrade OpenShift Container Platform version 3.11 clusters to the latest asynchronous errata update.



IMPORTANT

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

2.1. UPGRADE WORKFLOW

The 3.10 to 3.11 control plane upgrade performs the following steps for you:

- Back up all etcd data for recovery purposes.
- Update the API and controllers from 3.10 to 3.11.
- Update internal data structures to 3.11.
- Update the default router, if one exists, from 3.10 to 3.11.
- Update the default registry, if one exists, from 3.10 to 3.11.
- Update the default image streams and InstantApp templates.

The 3.10 to 3.11 node upgrade performs a rolling update of nodes, which:

- Marks a subset of nodes unschedulable and drains them of pods.
- Updates node components from 3.10 to 3.11.
- Returns those nodes to service.

2.2. PREREQUISITES

Before you upgrade your cluster:

- Review the OpenShift Container Platform 3.11 Release Notes . The release notes contain important notices about changes to OpenShift Container Platform and its function.
- If you are completing a large-scale upgrade, which involves at least 10 worker nodes and thousands of projects and pods, review Special considerations for large-scale upgrades to prevent upgrade failures.

- If you are completing a disconnected cluster update, you must update your image registry with new image versions or the cluster update will fail. For example, when updating from **3.11.153** to **3.11.157**, make sure the **v3.11.157** image tags are present.
- Upgrade the cluster to the latest asynchronous release of version 3.10. If your cluster is at a version earlier than 3.10, you must first upgrade incrementally. For example, upgrade from 3.7 to 3.9 (the 3.8 version was skipped), and then from 3.9 to 3.10.
- Run the Environment health checks to verify the cluster's health. In this process, you confirm that the nodes are in the **Ready** state and running the expected starting version and that there are no diagnostic errors or warnings.
- Ensure that your cluster meets the current prerequisites. If it does not, your upgrade might fail.
- The day before the upgrade, validate OpenShift Container Platform storage migration to ensure potential issues are resolved before the outage window. The following command updates the stored version of API objects for etcd storage:

```
$ oc adm migrate storage --include=* --loglevel=2 --config
/etc/origin/master/admin.kubeconfig
```

2.3. PREPARING FOR AN UPGRADE

After you satisfy the prerequisites, prepare for an automated upgrade:

1. Pull the latest subscription data from Red Hat Subscription Manager:

```
# subscription-manager refresh
```

2. If you are upgrading from OpenShift Container Platform 3.10 to 3.11:
   a. Back up the files that you need if you must downgrade to OpenShift Container Platform 3.10:
      i. On master hosts, back up the following files:

```
/etc/origin/master/master-config.yaml
/etc/origin/master/master.env
/etc/origin/master/scheduler.json
```

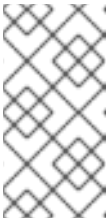
      ii. On node hosts, including masters, back up the following files:

```
/etc/origin/node/node-config.yaml
```

      iii. On etcd hosts, including masters that have etcd co-located on them, back up the following file:

```
/etc/etcd/etcd.conf
```

   b. The upgrade process creates a backup of all etcd data for recovery purposes, but ensure that you have a recent etcd backup at ***/backup/etcd-xxxxxx/backup.db*** before continuing. Manual etcd backup steps are described in the Day Two Operations Guide.



NOTE

When you upgrade OpenShift Container Platform, your etcd configuration does not change. Whether you run etcd as static pods on master hosts or as a separate service on master hosts or separate hosts does not change after you upgrade.

c. Manually disable the 3.10 repository and enable the 3.11 repository on each master and node host. You must also enable the **rhel-7-server-ansible-2.9-rpms** repository, if it is not already enabled:

- For cloud installations and on-premise installations on x86_64 servers, run the following command:

```
# subscription-manager repos \
    --disable="rhel-7-server-ose-3.10-rpms" \
    --disable="rhel-7-server-ansible-2.4-rpms" \
    --enable="rhel-7-server-ose-3.11-rpms" \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ansible-2.9-rpms"
# yum clean all
```

- For on-premise installations on IBM POWER8 servers, run the following command:

```
# subscription-manager repos \
    --disable="rhel-7-for-power-le-ose-3.10-rpms" \
    --enable="rhel-7-for-power-le-rpms" \
    --enable="rhel-7-for-power-le-extras-rpms" \
    --enable="rhel-7-for-power-le-optional-rpms" \
    --enable="rhel-7-server-ansible-2.9-for-power-le-rpms" \
    --enable="rhel-7-server-for-power-le-rhscl-rpms" \
    --enable="rhel-7-for-power-le-ose-3.11-rpms"
# yum clean all
```

- For on-premise installations on IBM POWER9 servers, run the following command:

```
# subscription-manager repos \
    --disable="rhel-7-for-power-le-ose-3.10-rpms" \
    --enable="rhel-7-for-power-9-rpms" \
    --enable="rhel-7-for-power-9-extras-rpms" \
    --enable="rhel-7-for-power-9-optional-rpms" \
    --enable="rhel-7-server-ansible-2.9-for-power-9-rpms" \
    --enable="rhel-7-server-for-power-9-rhscl-rpms" \
    --enable="rhel-7-for-power-9-ose-3.11-rpms"
# yum clean all
```

d. Ensure that you have the latest version of the **openshift-ansible** package on the host you run the upgrade playbooks on:

```
# yum update -y openshift-ansible
```

e. Prepare for the Cluster Monitoring Operator. In version 3.11, the Cluster Monitoring Operator is installed on an infrastructure node by default. If your cluster does not use infrastructure nodes:

- Add an infrastructure node to your cluster.
- Disable the Cluster Monitoring Operator by adding **openshift_cluster_monitoring_operator_install=false** to your inventory file.
- Specify which node to install the Cluster Monitoring Operator on by marking it with the **openshift_cluster_monitoring_operator_node_selector**.

f. If you use the standard OpenShift Container Platform registry, prepare for the change from **registry.access.redhat.com** to **registry.redhat.io**. Complete the configuration steps in Accessing and Configuring the Red Hat Registry .

3. Review and update your inventory file.

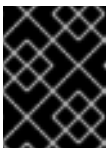
a. Ensure that any manual configuration changes you made to your master or node configuration files since your last Ansible playbook run, whether that was initial installation or your most recent cluster upgrade, are in the inventory file. For any variables that are relevant to the manual changes you made, apply the equivalent appropriate changes to your inventory files before running the upgrade. Otherwise, your manual changes might be overwritten by default values during the upgrade, which could cause pods to not run properly or other cluster stability issues.

b. By default, the installer checks to see if your certificates will expire within a year and fails if they will expire within that time. To change the number of days that your certificate is valid, specify a new value for the **openshift_certificate_expiry_warning_days** parameter. For example, to ensure that your certificates are valid for 180 days, specify **openshift_certificate_expiry_warning_days=180**.

c. To skip checking if your certificates will expire, set **openshift_certificate_expiry_fail_on_warn=False**.

d. If you made any changes to **admissionConfig** settings in your master configuration files, review the **openshift_master_admission_plugin_config** variable in Configuring Your Inventory File. Failure to do so might cause pods to get stuck in **Pending** state if you had **ClusterResourceOverride** settings manually configured previously, as described in Configuring Masters for Overcommitment.

e. If you used the **openshift_hostname** parameter in versions of OpenShift Container Platform before 3.10, ensure that the **openshift_kubelet_name_override** parameter is still in your inventory file and set to the value of **openshift_hostname** that you used in previous versions.



IMPORTANT

You must not remove the **openshift_kubelet_name_override** parameter from your inventory file after you upgrade.

f. If you manually manage the cluster's */etc/origin/master/htpasswd* file, add **openshift_master_manage_htpasswd=false** to your inventory file to prevent the upgrade process from overwriting the *htpasswd* file.

2.3.1. Updating policy definitions

During a cluster upgrade, and on every restart of any master, the default cluster roles are automatically reconciled to restore any missing permissions.

1. If you customized default cluster roles and want to ensure a role reconciliation does not modify them, protect each role from reconciliation:

```
$ oc annotate clusterrole.rbac <role_name> --overwrite
rbac.authorization.kubernetes.io/autoupdate=false
```

WARNING

You must manually update the roles that contain this setting to include any new or required permissions after upgrading.

2. Generate a default bootstrap policy template file:

```
$ oc adm create-bootstrap-policy-file --filename=policy.json
```

NOTE

The contents of the file vary based on the OpenShift Container Platform version, but the file contains only the default policies.

3. Update the ***policy.json*** file to include any cluster role customizations.

4. Use the policy file to automatically reconcile roles and role bindings that are not reconcile protected:

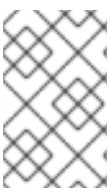
```
$ oc auth reconcile -f policy.json
```

5. Reconcile security context constraints:

```
# oc adm policy reconcile-sccs \
    --additive-only=true \
    --confirm
```

2.3.2. Upgrade phases

You can upgrade the OpenShift Container Platform cluster in one or more phases. You can choose to upgrade all hosts in one phase by running a single Ansible playbook or upgrade the *control plane*, or master components, and nodes in multiple phases using separate playbooks.



NOTE

If your OpenShift Container Platform cluster uses GlusterFS pods, you must perform the upgrade in multiple phases. See Special Considerations When Using Containerized GlusterFS for details on how to upgrade with GlusterFS.

When upgrading in separate phases, the control plane phase includes upgrading:

- Master components
- Node services running on masters
- Docker or CRI-O running on masters
- Docker or CRI-O running on any stand-alone etcd hosts

If you upgrade only the nodes, you must first upgrade the control plane. The node phase includes upgrading:

- Node services running on stand-alone nodes
- Docker or CRI-O running on stand-alone nodes

Nodes that run master components are upgraded only during the control plane upgrade phase. This ensures that the node services and container engines on masters are not upgraded twice, once during the control plane phase and again during the node phase.

2.3.3. Node upgrade parameters

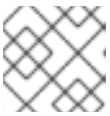
Whether you upgrade in a single or multiple phases, you can customize how the node portion of the upgrade progresses by passing certain Ansible variables to an upgrade playbook using the **-e** option.

- Set the **openshift_upgrade_nodes_serial** variable to an integer or percentage to control how many node hosts are upgraded at the same time. The default is **1**, which upgrades one node at a time.
  For example, to upgrade 20 percent of the total number of detected nodes at a time, run:

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial="20%"
```

- Set the **openshift_upgrade_nodes_label** to specify that only nodes with a certain label are upgraded.
  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"
```



NOTE

See Managing Nodes for more information about node labels.

- Set the **openshift_upgrade_nodes_max_fail_percentage** variable to specify how many nodes can fail in each batch of upgrades. If the percentage of failed nodes exceeds your value, the playbook stops the upgrade process.
- Set the **openshift_upgrade_nodes_drain_timeout** variable to specify the length of time to wait before marking a node as failed.

In this example, 10 nodes are upgraded at a time, the upgrade stops if more than 20 percent of the nodes fail, and a node is marked as failed if it takes more than 600 seconds to drain the node:

```
$ ansible-playbook -i <path/to/inventory/file> \
    </path/to/upgrade/playbook> \
    -e openshift_upgrade_nodes_serial=10 \
    -e openshift_upgrade_nodes_max_fail_percentage=20 \
    -e openshift_upgrade_nodes_drain_timeout=600
```

2.3.4. Ansible hooks for upgrades

When upgrading OpenShift Container Platform, you can execute custom tasks during specific operations through a system called *hooks*. Hooks allow cluster administrators to provide files defining tasks to execute before or after specific areas during upgrades. You can use hooks to validate or modify custom infrastructure when upgrading OpenShift Container Platform.

Because when a hook fails, the operation fail, design hooks that are idempotent, or can run multiple times and provide the same results.

2.3.4.1. Limitations

- Hooks have no defined or versioned interface. They can use internal **openshift-ansible** variables, but there is no guarantee these variables will remain in future releases. In the future, hooks might be versioned, giving you advance warning that your hook needs to be updated to work with the latest **openshift-ansible**.
- Hooks have no error handling, so an error in a hook halts the upgrade process. If you get an error, you must address the problem and then start the upgrade again.
- You can run node upgrade hooks on only nodes, not masters. To run the hooks on masters, you must specify a master hook for those nodes.

2.3.4.2. Using hooks

You define hooks in the ***hosts*** inventory file under the **OSEv3:vars** section.

Each hook must point to a YAML file that defines Ansible tasks. This file is used as an *include*, meaning that the file cannot be a playbook, but is a set of tasks. Best practice suggests using absolute paths to the hook file to avoid any ambiguity.

Example hook definitions in an inventory file

```
[OSEv3:vars]
openshift_master_upgrade_pre_hook=/usr/share/custom/pre_master.yml
openshift_master_upgrade_hook=/usr/share/custom/master.yml
openshift_master_upgrade_post_hook=/usr/share/custom/post_master.yml

openshift_node_upgrade_pre_hook=/usr/share/custom/pre_node.yml
openshift_node_upgrade_hook=/usr/share/custom/node.yml
openshift_node_upgrade_post_hook=/usr/share/custom/post_node.yml
```

Example *pre_master.yml* task

```
---
# Trivial example forcing an operator to ack the start of an upgrade
# file=/usr/share/custom/pre_master.yml

- name: note the start of a master upgrade
  debug:
    msg: "Master upgrade of {{ inventory_hostname }} is about to start"

- name: require an operator agree to start an upgrade
  pause:
    prompt: "Hit enter to start the master upgrade"
```

2.3.4.3. Available upgrade hooks

Table 2.1. Master Upgrade Hooks

Hook name	Description
openshift_master_upgrade_pre_hook	• Runs *before* each master is upgraded. • This hook runs against *each master* in serial. • If a task must run against a different host, the task must use **delegate_to** or **local_action**.
openshift_master_upgrade_hook	• Runs *after* each master is upgraded but *before* its service or system restart. • This hook runs against *each master* in serial. • If a task must run against a different host, the task must use **delegate_to** or **local_action**.
openshift_master_upgrade_post_hook	• Runs *after* each master is upgraded and its service or system restarts. • This hook runs against *each master* in serial. • If a task must run against a different host, the task must use **delegate_to** or **local_action**.

Table 2.2. Node upgrade hooks

Hook name	Description

Hook name	Description
openshift_node_upgrade_pre_hook	• Runs *before* each node is upgraded. • This hook runs against *each node* in serial. • If a task must run against a different host, the task must use **delegate_to** or **local_action**.
openshift_node_upgrade_hook	• Runs *after* each node is upgraded but*before* it is marked schedulable again. • This hook runs against *each node* in serial. • If a task must run against a different host, they task must use **delegate_to** or **local_action**.
openshift_node_upgrade_post_hook	• Runs *after* each node is upgraded. It is the *last* node upgrade action. • This hook runs against *each node* in serial. • If a task must run against a different host, the task must use **delegate_to** or **local_action**.

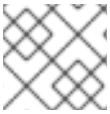
2.3.5. Special considerations for upgrading OpenShift Container Platform

If your OpenShift Container Platform cluster uses a mixed environment or gcePD storage, you need to take more steps before you upgrade it.

Before you upgrade a mixed environment, such as one with Red Hat Enterprise Linux (RHEL) and RHEL Atomic Host, set values in the inventory file for both the **openshift_pkg_version** and **openshift_image_tag** parameters. Setting these values ensures that all nodes in your cluster run the same version of OpenShift Container Platform. While this is a best practice for major updates, such as from OpenShift Container Platform 2 to OpenShift Container Platform 3, setting these values are mandatory for minor version upgrades.

For example, to upgrade from OpenShift Container Platform 3.9 to OpenShift Container Platform 3.10, set the following parameters and values:

```
openshift_pkg_version=-3.10.16
openshift_image_tag=v3.10.16
```



NOTE

These parameters can also be present in other, non-mixed, environments.

2.3.5.1. Special considerations for large-scale upgrades

For large-scale cluster upgrades, which involve at least 10 worker nodes and thousands of projects and pods, the API object storage migration should be performed prior to running the upgrade playbooks, and then again after the upgrade has successfully completed. Otherwise, the upgrade process will fail.

Refer to the **Running the pre- and post- API server model object migration outside of the upgrade window** section of the Recommendations for large-scale OpenShift upgrades for further guidance.

2.3.5.2. Special considerations when using gcePD

Because the default gcePD storage provider uses an RWO (Read-Write Only) access mode, you cannot perform a rolling upgrade on the registry or scale the registry to multiple pods. Therefore, when upgrading OpenShift Container Platform, you must specify the following environment variables in your Ansible inventory file:

```
[OSEv3:vars]

openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_gcs_bucket=bucket01
openshift_hosted_registry_storage_gcs_keyfile=test.key
openshift_hosted_registry_storage_gcs_rootdirectory=/registry
```

2.4. UPGRADING TO THE LATEST OPENSHIFT CONTAINER PLATFORM RELEASE

To upgrade an existing OpenShift Container Platform 3.10 or 3.11 cluster to the latest 3.11 release:

1. Prepare for an upgrade to ensure you use the latest upgrade playbooks.
2. Ensure the **openshift_deployment_type** parameter in your inventory file is set to **openshift-enterprise**.
3. To enable rolling, full system restarts of the hosts, set the **openshift_rolling_restart_mode** parameter in your inventory file to **system**. Otherwise, the service is restarted on masters, but the systems do not reboot.

   NOTE

   The **openshift_rolling_restart_mode** only works for master hosts.

   See Configuring Cluster Variables for details.
4. If you modified the **oreg_url** parameter to change the cluster image registry location, you must run the **imageconfig** playbook to update the image location:

   ```
   $ cd /usr/share/ansible/openshift-ansible
   $ ansible-playbook -i </path/to/inventory/file> \
       playbooks/openshift-node/imageconfig.yml
   ```
5. Upgrade your nodes.
   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.

- To upgrade control plane and nodes in a single phase, run the ***upgrade.yml*** playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i </path/to/inventory/file> \
    playbooks/byo/openshift-cluster/upgrades/v3_11/upgrade.yml
```

- To upgrade the control plane and nodes in separate phases:

  a. Upgrade the control plane by running the ***upgrade_control_plane.yaml*** playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i </path/to/inventory/file> \
    playbooks/byo/openshift-cluster/upgrades/v3_11/upgrade_control_plane.yml
```

  b. Upgrade the nodes by running the ***upgrade_nodes.yaml*** playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i </path/to/inventory/file> \
    playbooks/byo/openshift-cluster/upgrades/v3_11/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 running the ***upgrade_nodes.yml*** playbook until all nodes are upgraded.

6. If you did not enable automated reboot of the master hosts by using **openshift_rolling_restart variable=system** in step 3 of this procedure, you can manually reboot all master hosts together with all node hosts after the upgrade has completed. Rebooting the hosts is optional.

7. If you use aggregated logging, upgrade the EFK logging stack .

8. If you use cluster metrics, upgrade cluster metrics.

9. Verify the upgrade.

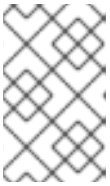
2.5. UPGRADING OPENSHIFT CONTAINER PLATFORM WHEN USING CONTAINERIZED GLUSTERFS

When upgrading OpenShift Container Platform, you must upgrade the set of nodes where GlusterFS pods run. However, because these pods run as part of a daemonset, you cannot use **drain** or **unschedule** commands to terminate and evacuate the GlusterFS pods. To avoid data availability and cluster corruption, you must also upgrade nodes that host GlusterFS pods one at a time to ensure that the upgrade process completes on a node that runs GlusterFS before the upgrade starts on the next node.

To upgrade OpenShift Container Platform if you use containerized GlusterFS:

1. Upgrade the control plane (the master nodes and etcd nodes).

2. Upgrade standard **infra** nodes (router, registry, logging, and metrics).



NOTE

If any of the nodes in those groups are running GlusterFS, perform step 4 of this procedure at the same time. GlusterFS nodes must be upgraded along with other nodes in their class (**app** versus **infra**), one at a time.

3. Upgrade standard nodes running application containers.

NOTE

If any of the nodes in those groups are running GlusterFS, perform step 4 of this procedure at the same time. GlusterFS nodes must be upgraded along with other nodes in their class (**app** versus **infra**), one at a time.

4. Upgrade the OpenShift Container Platform nodes running GlusterFS one at a time.

   a. Add the following parameters in the inventory file at */etc/ansible/hosts*:

```
openshift_hosted_registry_storage_kind=glusterfs
openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/rhgs-volmanager-rhel7:<your_cns_vesion> (1)
openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-server-rhel7:<your_cns_vesion> (2)
openshift_storage_glusterfs_block_image=registry.access.redhat.com/rhgs3/rhgs-gluster-block-prov-rhel7:<your_cns_vesion> (3)
openshift_storage_glusterfs_s3_image=registry.access.redhat.com/rhgs3/rhgs-s3-server-rhel7:<your_cns_vesion> (4)
```

   (1) (2) (3) (4) Specify the CNS version, such as **v3.9** or **v3.11.3**.

   b. Update the image tags from **latest** to **<your_cns_version>**, such as **v3.9** or **v3.11.3**, on the following resources, if they are present in your configuration:

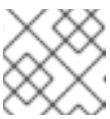
```
$ oc edit -n <glusterfs_namespace> ds glusterfs-<name>
$ oc edit -n <glusterfs_namespace> dc heketi-<name>
$ oc edit -n <glusterfs_namespace> dc glusterblock-<name>-provisioner-dc
$ oc edit -n <glusterfs_namespace> dc gluster-<name>-<account>-s3
```

   c. Add a label to the node you want to upgrade so that only one node is upgraded at a time:

```
$ oc label node <node_name> type=upgrade
```

   d. Do not terminate the GlusterFS pod you want to restart.

   e. To run the upgrade playbook on a single GlusterFS node, use **-e openshift_upgrade_nodes_label="type=upgrade"**.



NOTE

The GlusterFS pod should not be terminated.

   f. Wait for the GlusterFS pod to respawn and appear.

g. Run basic health checks after each pod restart to ensure they pass.

h. **oc rsh** into the pod and verify all volumes are healed:

```
$ oc rsh <GlusterFS_pod_name>
$ for vol in `gluster volume list`; do gluster volume heal $vol info; done
```

Ensure all of the volumes are healed and there are no outstanding tasks. The **heal info** command lists all pending entries for a given volume's heal process. A volume is considered healed when **Number of entries** for that volume is **0**.

i. Remove the upgrade label and go to the next GlusterFS node.

```
$ oc label node <node_name> type-
```

2.6. UPGRADING OPTIONAL COMPONENTS

If you installed an EFK logging stack or cluster metrics, you must separately upgrade the component.

2.6.1. Upgrading the EFK Logging Stack

To upgrade an existing EFK logging stack deployment, you review your parameters and run the ***openshift-logging/config.yml*** playbook.

NOTE

The upgrade can replace your **logging-fluentd** and **logging-curator** ConfigMaps. If you want to retain your current **logging-fluentd** and **logging-curator** ConfigMaps, set the **openshift_logging_fluentd_replace_configmap** and **openshift_logging_curator_replace_configmap** parameters to **no** in the inventory host file.

The EFK upgrade also upgrades Elasticsearch from version 2 to version 5. For important information on changes in Elasticsearch 5, you should review the Elasticsearch breaking changes.

It is important to note that Elasticsearch 5 has some significant changes to the index structures. Previously, Elasticsearch permitted a dot character, **.**, in field names. In version 5, Elasticsearch interprets any dot in an Elasticsearch field name as nested structure. If you have a field with a dot, the string after the dot is interpreted as the type of field, leading to mapping conflicts during the upgrade.

To help identify potential conflicts, OpenShift Container Platform provides a script that examines your Elasticsearch fields to determine if any fields contain a dot in the name.

For example, the following fields were allowed in Elasticsearch 2:

```
{
  "field": 123  // "field" is of type number
}

// Any dot in field name is treated as any other valid character in the field name.
// It is just part of the field name.
{
  "field.name": "Bob"  // "field.name" is of type String
```

In Elasticsearch 5 and higher the **field** string would become the field and the **name** string would become a type for the field:

```
{
  "field": 123  // "field" is of type number
}

// Any dot in field name is always interpreted as nested structure.
{
  "field" : {  // "field" is of type Object
    "name": "Bob"  // "name" is of type String
  }
}
```

Upgrading in this case would result in the **field** field having two different types, which is not permitted.

If you need to keep these conflicting indices, you need to reindex the data and change the documents to get rid of conflicting data structure. For more information, see Upgrading fields with dots to 5.x .

2.6.1.1. Determining if fields have dots in field names

You can run the following script to determine if your indices contain any fields with a dot in the name.

NOTE

The following command uses the **jq** JSON processor to get directly at the necessary data. Red Hat Enterprise Linux (RHEL), depending on version, might not provide a package for **jq**. You might need to install this from external sources, or unsupported locations.

```
oc exec -c elasticsearch -n $LOGGING_NS $pod -- es_util --query='_mapping?
pretty&filter_path=**.mappings.*.properties' \
  | jq '.[].mappings[].properties | keys' \
  | jq .[] \
  | egrep -e "\."
```

The upgrade path depends on whether the indices have fields with dots or do not have fields with dots .

2.6.1.2. Upgrading if fields have dots in field names

If the script above indicates your indices contain fields with a dot in the name, use the following steps to correct this issue and upgrade.

To upgrade your EFK stack:

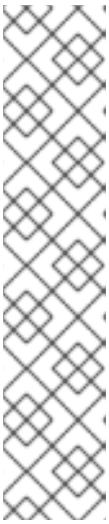
1. Review how to specify logging Ansible variables and update your Ansible inventory file to at least set the following required variable in the **[OSEv3:vars]** section:

```
[OSEv3:vars]

openshift_logging_install_logging=true (1)
```

(1) Enables the ability to upgrade the logging stack.

2. Update any other **openshift_logging_*** variables that you want to override the default values for, as described in Specifying Logging Ansible Variables .
   You can set the **openshift_logging_elasticsearch_replace_configmap** parameter to **true** to replace your **logging-elasticsearch** ConfigMap with the current default values. In some cases, using an older ConfigMap can cause the upgrade to fail. The default is set to **false**. For more information, see the parameter in specify logging Ansible variables .



NOTE

If your Fluentd **DeploymentConfig** object and **DaemonSet** object for the EFK components are already set with:

```
image: <image_name>:<vX.Y>
imagePullPolicy: IfNotPresent
```

The latest version **<image_name>** might not be pulled if there is already one with the same **<image_name:vX.Y>** stored locally on the node where the pod is being re-deployed. If your image version is v3.11, and you want to upgrade to the latest version using the playbook, set the **openshift_image_tag=v3.11.<Z>** or **oreg_url=registry.access.redhat.com/openshift3/ose-${component}:v3.11.<Z>** Ansible parameter.

3. Dechedule your Fluentd pods to stop data ingestion and ensure the cluster state does not change.
   For example, you can change the node selector in Fluentd pods to one that does not match any nodes.

```
oc patch daemonset logging-fluentd -p '{"spec": {"template": {"spec": {"nodeSelector": {"non-existing": "true"}}}}}'
```

4. Perform an Elasticsearch Index flush on all relevant indices. The flush process persists all logs from memory to disk, which prevents log loss when Elasticsearch is shutdown during the upgrade.

5. Perform an online or offline backup:

   - Perform an online backup of specific Elasticsearch indices the entire cluster.
   - Perform an offline backup:

     a. Scale down all Elasticsearch DeploymentConfig to **0**:

```
$ oc scale dc <name> -n openshift-logging --replicas=0
```

     b. Back up external persistent volumes using the appropriate method for your organization.

6. For any file name with a dot character, you need to take one of the following actions before upgrading:

   - Deleting the indices. This is the better approach to avoid mapping conflicts during the upgrade.

- Reindexing the data and changing the documents to get rid of conflicting data structure. This method retain the data. For information on potential mapping conflicts see Mappging changes in the Elasticsearch documentation.

7. Repeat the on-line or offline backup.

8. Run the ***openshift-logging/config.yml*** playbook according to the deploying the EFK stack instructions to complete the logging upgrade. You run the installation playbook for the new OpenShift Container Platform version to upgrade the logging deployment.

2.6.1.3. Upgrading if fields do not have dots

If the script above indicates your indices do not contain fields with a dot in the name, use the following steps to upgrade.

1. Review how to specify logging Ansible variables and update your Ansible inventory file to at least set the following required variable in the **[OSEv3:vars]** section:

```
[OSEv3:vars]

openshift_logging_install_logging=true (1)
```

(1) Enables the ability to upgrade the logging stack.

2. Update any other **openshift_logging_*** variables that you want to override the default values for, as described in Specifying Logging Ansible Variables .
You can set the **openshift_logging_elasticsearch_replace_configmap** parameter to **true** to replace your **logging-elasticsearch** ConfigMap with the current default values. In some cases, using an older ConfigMap can cause the upgrade to fail. The default is set to **false**. For more information, see the parameter in specify logging Ansible variables .

NOTE

If your Fluentd **DeploymentConfig** object and **DaemonSet** object for the EFK components are already set with:

```
image: <image_name>:<vX.Y>
imagePullPolicy: IfNotPresent
```

The latest version <image_name> might not be pulled if there is already one with the same **<image_name:vX.Y>** stored locally on the node where the pod is being re-deployed. If your image version is v3.11, and you want to upgrade to the latest version using the playbook, set the **openshift_image_tag=v3.11.<Z>** or **oreg_url=registry.access.redhat.com/openshift3/ose-${component}:v3.11.<Z>** Ansible parameter.

3. Optionally, dechedule your Fluentd pods and scale down your Elasticsearch pods to stop data ingestion and ensure the cluster state does not change.
For example, you can change the node selector in Fluentd pods to one that does not match any nodes.

```
oc patch daemonset logging-fluentd -p '{"spec": {"template": {"spec": {"nodeSelector": {"non-existing": "true"}}}}}'
```

4. Optionally, perform and online or offline backup:
   - Perform an online backup of specific Elasticsearch indices the entire cluster.
   - Perform an offline backup:
     a. Scale down all Elasticsearch DeploymentConfigs to **0**:

     ```
     $ oc scale dc <name> -n openshift-logging --replicas=0
     ```

     b. Back up external persistent volumes using the appropriate method for your organization.
5. Run the ***openshift-logging/config.yml*** playbook according to the deploying the EFK stack instructions to complete the logging upgrade. You run the installation playbook for the new OpenShift Container Platform version to upgrade the logging deployment.
6. Optionally, use the Elasticsearch restore module to restore your Elasticsearch indices from the snapshot.

2.6.2. Upgrading cluster metrics

To upgrade an existing cluster metrics deployment, you review your parameters and run the ***openshift-metrics/config.yml*** playbook.

1. Review how to specify metrics Ansible variables and update your Ansible inventory file to at least set the following required variable in the **[OSEv3:vars]** section:

```
[OSEv3:vars]

openshift_metrics_install_metrics=true (1)
openshift_metrics_hawkular_hostname=<fqdn> (2)
openshift_metrics_cassandra_storage_type=(emptydir|pv|dynamic) (3)
```

(1) Enables the ability to upgrade the metrics deployment.

(2) Used for the Hawkular Metrics route. Specify a fully qualified domain name.

(3) Specify the same type as the previous deployment.

2. Update any other **openshift_metrics_*** variables that you want to override the default values for, as described in Specifying Metrics Ansible Variables .
3. Run the ***openshift-metrics/config.yml*** playbook according to the deploying the metrics deployment instructions to complete the metrics upgrade. You run the installation playbook for the new OpenShift Container Platform version to upgrade the logging deployment.

2.7. VERIFYING THE UPGRADE

Ensure that:

- The cluster is healthy.
- The master, node, and etcd services or static pods are running well.

- The OpenShift Container Platform, **docker-registry**, and router versions are correct.
- The original applications are still available, and new application can be created.
- Running **oc adm diagnostics** produces no errors.

To verify the upgrade:

1. Check that all nodes are marked as **Ready**:

```
# oc get nodes
NAME                    STATUS    ROLES       AGE       VERSION
master1.example.com     Ready     master      47d       v1.11.0+d4cacc0
master2.example.com     Ready     master      47d       v1.11.0+d4cacc0
master3.example.com     Ready     master      47d       v1.11.0+d4cacc0
infra-node1.example.com Ready     infra       47d       v1.11.0+d4cacc0
infra-node2.example.com Ready     infra       47d       v1.11.0+d4cacc0
node1.example.com       Ready     compute     47d       v1.11.0+d4cacc0
node2.example.com       Ready     compute     47d       v1.11.0+d4cacc0
```

2. Verify that the static pods for the control plane are running:

```
# oc get pods -n kube-system
NAME                                READY     STATUS    RESTARTS   AGE
master-api-master1.example.com          1/1       Running   4          1h
master-controllers-master1.example.com  1/1       Running   3          1h
master-etcd-master1.example.com         1/1       Running   6          5d
[...]
```

3. 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:v3.11.634",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.11.634",
```

4. 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.
```

CHAPTER 3. PERFORMING BLUE-GREEN CLUSTER UPGRADES

NOTE

This topic serves as an alternative approach for node host 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 node hosts instead of upgrading them in-place.

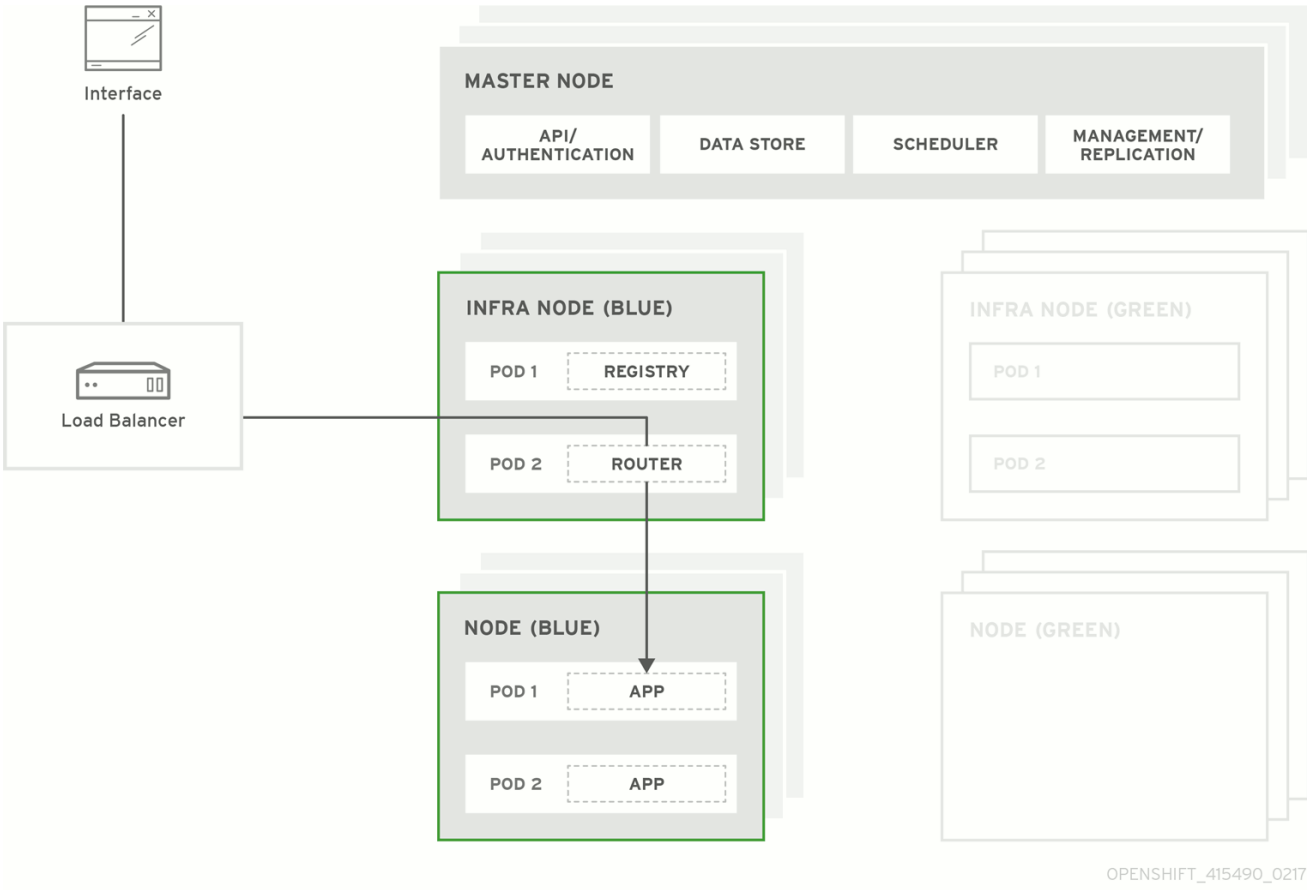
This method allows administrators to switch traffic from the old set of node hosts (e.g., the *blue* deployment) to the new set (e.g., the *green* deployment) after the new deployment has been verified. If a problem is detected, it is also then easy to rollback to the old deployment quickly.

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 node hosts. All user processes run on these systems and even critical pieces of OpenShift Container Platform infrastructure are self-hosted on these resources. 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.

Figure 3.1. Blue-green deployment



3.1. PREPARING FOR A BLUE-GREEN UPGRADE

After you have upgraded your master and etcd hosts using method described for In-place Upgrades, use the following sections to prepare your environment for a blue-green upgrade of the remaining node hosts.

3.1.1. Sharing software entitlements

Administrators 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 node host:

1. On each old node host that will be upgraded, note its **system identity** value, which is the consumer ID:

```
# subscription-manager identity | grep system
system identity: 6699375b-06db-48c4-941e-689efd6ce3aa
```

2. On each new RHEL 7 or RHEL Atomic Host 7 system that will replace an old node host, register using the respective consumer ID from the previous step:

```
# subscription-manager register --consumerid=6699375b-06db-48c4-941e-689efd6ce3aa
```

3.1.2. Labeling blue nodes

You must ensure that your current node hosts in production are labeled either **blue** or **green**. In this example, the current production environment is **blue**, and the new environment is **green**.

1. Get the current list of node names known to the cluster:

```
$ oc get nodes
```

2. Label all non-master node hosts (compute nodes) and dedicated infrastructure nodes in your current production environment with **color=blue**:

```
$ oc label node --selector=node-role.kubernetes.io/compute=true color=blue

$ oc label node --selector=node-role.kubernetes.io/infra=true color=blue
```

In the previous command, the **--selector** flag is used to match a subset of the cluster using the relevant node labels, and all matches are labeled with **color=blue**.

3.1.3. Creating and labeling green nodes

Create the green environment by adding an equal number of new node hosts to the existing cluster:

1. Add the new node hosts using the procedure as described in Adding Hosts to an Existing Cluster. When updating your inventory file with the **[new_nodes]** group in that procedure, ensure these variables are set:

   - In order to delay workload scheduling until the nodes are deemed healthy, which you verify in later steps, set the **openshift_schedulable=false** variable for each new node host to ensure they are unschedulable initially.

2. After the new nodes deploy, apply the **color=green** label to each new node:

```
$ oc label node <node_name> color=green
```

3.1.4. Verifying green nodes

Verify that your new green nodes are in a healthy state:

1. Verify that new nodes are detected in the cluster and are in **Ready,SchedulingDisabled** state:

```
$ oc get nodes

NAME              STATUS                     ROLES     AGE
node4.example.com   Ready,SchedulingDisabled    compute    1d
```

2. Verify that the green nodes have proper labels:

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

NAME              STATUS                     ROLES     AGE   LABELS
node4.example.com   Ready,SchedulingDisabled    compute    1d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,color=green,kubernetes.io/hostna
me=m01.example.com,node-role.kubernetes.io/compute=true
```

3. Perform a diagnostic check for the cluster:

```
$ oc adm diagnostics
```

```
[Note] Determining if client configuration exists for client/cluster diagnostics
Info:  Successfully read a client config file at '/root/.kube/config'
Info:  Using context for cluster-admin access: 'default/m01-example-com:8443/system:admin'
[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/m01-example-com:8443/system:admin]
      Description: Validate client config context is complete and has connectivity
...
       [Note] Running diagnostic: CheckExternalNetwork
           Description: Check that external network is accessible within a pod

      [Note] Running diagnostic: CheckNodeNetwork
           Description: Check that pods in the cluster can access its own node.

      [Note] Running diagnostic: CheckPodNetwork
           Description: Check pod to pod communication in the cluster. In case of ovs-subnet
network plugin, all pods
should be able to communicate with each other and in case of multitenant network plugin,
pods in non-global projects
should be isolated and pods in global projects should be able to access any pod in the cluster
and vice versa.

      [Note] Running diagnostic: CheckServiceNetwork
           Description: Check pod to service communication in the cluster. In case of ovs-
subnet network plugin, all
pods should be able to communicate with all services and in case of multitenant network
plugin, services in non-global
projects should be isolated and pods in global projects should be able to access any service
in the cluster.
...
```

3.2. PREPARING THE GREEN NODES

To migrate pods from the blue environment to the green, you must pull the required container images.

Network latency and load on the registry can cause delays if the environment does not have sufficient capacity. You can minimize impact to the running system by importing new image streams to trigger new pod deployments to the new nodes.

Major releases of OpenShift Container Platform, and sometimes asynchronous errata updates, introduce new image streams for builder images for users of Source-to-Image (S2I). Upon import, any builds or deployments configured with image change triggers are automatically created.

Another benefit of triggering the builds is that it fetches the majority of the ancillary images to all node hosts, such as the various builder images, the pod infrastructure image, and deployers. The green nodes are then prepared for the expected load increase, and the remaining images more quickly migrated during node evacuation.

When you are ready to continue with the upgrade process, follow these steps to warm the green nodes:

1. Set the green nodes to schedulable so that new pods are deployed to them:

```
$ oc adm manage-node --schedulable=true --selector=color=green
```

2. Set the blue nodes to unschedulable so that no new pods run on them:

```
$ oc adm manage-node --schedulable=false --selector=color=blue
```

3. Update the node selectors for the registry and router deployment configurations to use the **node-role.kubernetes.io/infra=true** label. This change starts new deployments that place the registry and router pods on your new infrastructure nodes.

   a. Edit the **docker-registry** deployment configuration:

   ```
   $ oc edit -n default dc/docker-registry
   ```

   b. Update the **nodeSelector** parameter to use the following value, with **"true"** in quotation marks, and save your changes:

   ```
   nodeSelector:
     node-role.kubernetes.io/infra: "true"
   ```

   c. Edit the **router** deployment configuration:

   ```
   $ oc edit -n default dc/router
   ```

   d. Update the **nodeSelector** parameter to use the following value, with **"true"** in quotation marks, and save your changes:

   ```
   nodeSelector:
     node-role.kubernetes.io/infra: "true"
   ```

   e. Verify that the **docker-registry** and **router** pods are running and in ready state on the new infrastructure nodes:

   ```
   $ oc get pods -n default -o wide

   NAME                      READY     STATUS    RESTARTS   AGE       IP              NODE
   docker-registry-2-b7xbn   1/1       Running   0          18m       10.128.0.188    infra-
   node3.example.com
   router-2-mvq6p            1/1       Running   0          6m        192.168.122.184   infra-
   node4.example.com
   ```

4. Update the default image streams and templates.

5. Import the latest images. This process can trigger a large number of builds, but the builds are performed on the green nodes and, therefore, do not impact any traffic on the blue deployment.

6. 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 image import.

3.3. EVACUATING AND DECOMMISSIONING BLUE NODES

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 host at a time.

If services are composed of pods using zone anti-affinity, you can evacuate an entire zone at one time. You must ensure that the storage volumes used are available in the new zone. Follow the directions in your cloud provider's documentation.

A node host evacuation is triggered whenever the node service is stopped. Node labeling is very important and can cause issues if nodes are mislabeled or commands are run on nodes with generalized labels. Exercise caution if master hosts are also labeled with **color=blue**.

When you are ready to continue with the upgrade process, follow these steps.

1. Evacuate and delete all blue nodes with the following commands:

```
$ oc adm manage-node --selector=color=blue --evacuate
$ oc delete node --selector=color=blue
```

2. After the blue node hosts no longer contain pods and have been removed from OpenShift Container Platform, they are safe to power off. As a safety precaution, confirm that there are no issues with the upgrade before you power off the hosts.

   a. Unregister each old host:

   ```
   # subscription-manager clean
   ```

   b. Back up any useful scripts or required files that are stored on the hosts.

   c. After you are comfortable that the upgrade succeeded, remove these hosts.

CHAPTER 4. UPDATING OPERATING SYSTEMS

Updating the operating system (OS) on a host, by either upgrading across major releases or updating the system software for a minor release, 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.

4.1. UPDATING THE OPERATING SYSTEM ON A HOST

To safely upgrade the OS on a host:

1. Drain the node in preparation for maintenance:

```
$ oc adm drain <node_name> --force --delete-local-data --ignore-daemonsets
```

2. In order to protect sensitive packages that do not need to be updated, apply the exclude rules to the host:

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

A reboot ensures that the host is running the newest versions and means that the **container engine** and OpenShift Container Platform processes have been restarted, which forces them to check that all of the rules in other services are correct.

```
# yum update
# reboot
```

However, instead of rebooting a node host, you can restart the services that are affected or 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 fixes the flow rules.

3. Configure the host to be schedulable again:

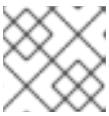
```
$ oc adm uncordon <node_name>
```

4.1.1. Upgrading Nodes Running OpenShift Container Storage

If using OpenShift Container Storage, upgrade the OpenShift Container Platform nodes running OpenShift Container Storage one at a time.

1. To begin, recall the project in which OpenShift Container Storage was deployed.
2. Confirm the node and pod selectors configured on the service's daemonset.

```
$ oc get daemonset -n <project_name> -o wide
```



NOTE

Use **-o wide** to include the pod selector in the output.

These selectors are found under **NODE-SELECTOR** and **SELECTOR**, respectively. The example commands below will use **glusterfs=storage-host** and **glusterfs=storage-pod**, respectively.

3. Given the daemonset's node selector, confirm which hosts have the label, and hence are running pods from the daemonset:

```
$ oc get nodes --selector=glusterfs=storage-host
```

Chose a node which will have its operating system upgraded.

4. Remove the daemonset label from the node:

```
$ oc label node <node_name> glusterfs-
```

This will cause the OpenShift Container Storage pod to terminate on that node.

The node can now have its OS upgraded as described above.

1. To restart an OpenShift Container Storage pod on the node, relabel the node with the daemonset label:

```
$ oc label node <node_name> glusterfs=storage-host
```

2. Wait for the OpenShift Container Storage pod to respawn and appear.

3. Given the daemonset's pod selector, determine the name of the newly spawned pod by searching for a pod running on the node whose OS you upgraded:

```
$ oc get pod -n <project_name> --selector=glusterfs=storage-pod -o wide
```

NOTE

Use **-o wide** to include which host the pod is running on in the output.

4. **oc rsh** into the gluster pod to check the volume heal:

```
$ oc rsh <pod_name>
$ for vol in `gluster volume list`; do gluster volume heal $vol info; done
$ exit
```

Ensure all of the volumes are healed and there are no outstanding tasks. The **heal info** command lists all pending entries for a given volume's heal process. A volume is considered healed when **Number of entries** for that volume is **0**. Use **gluster volume status <volume_name>** for additional details about the volume. The **Online** state should be marked **Y** for all bricks.

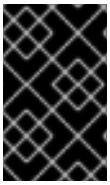
CHAPTER 5. DOWNGRADING A CLUSTER

After an OpenShift Container Platform upgrade, you might need to downgrade your cluster to an earlier version. You can downgrade from OpenShift Container Platform version 3.11 to version 3.10.

WARNING

In the initial release of OpenShift Container Platform version 3.11, downgrading does not completely restore your cluster to version 3.10. Do not downgrade.

If you need to downgrade, contact Red Hat support so they can help you determine the best course of action.



IMPORTANT

Downgrading a cluster to version 3.10 is supported for only RPM-based installations of OpenShift Container Platform, and you must take your entire cluster offline to downgrade.

5.1. VERIFYING BACKUPS

1. Ensure that a backup of the ***master-config.yaml*** file, ***scheduler.json*** file, and the etcd data directory exist on your masters:

```
/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master.env
/etc/origin/master/scheduler.json
/var/lib/etcd/openshift-backup-xxxx
```

   You save these files during the upgrade process.

2. Locate the copies of the following files that you created when you prepared for an upgrade. On node and master hosts:

```
/etc/origin/node/node-config.yaml
```

   On etcd hosts, including masters that have etcd co-located on them:

```
/etc/etcd/etcd.conf
```

5.2. SHUTTING DOWN THE CLUSTER

1. On all master and node hosts, stop the master and node services by removing the pod definition and rebooting the host:

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
# reboot
```

5.3. REMOVING RPMS AND STATIC PODS

1. On all masters, nodes, and etcd members (if using a dedicated etcd cluster), remove the following packages:

```
# yum remove atomic-openshift \
  atomic-openshift-excluder \
  atomic-openshift-hyperkube \
  atomic-openshift-node \
  atomic-openshift-docker-excluder \
  atomic-openshift-clients
```

2. Verify the packages were removed successfully:

```
# rpm -qa | grep atomic-openshift
```

3. On control plane hosts (master and etcd hosts), move the static pod definitions:

```
# mkdir /etc/origin/node/pods-backup
# mv /etc/origin/node/pods/* /etc/origin/node/pods-backup/
```

4. Reboot each host:

```
# reboot
```

5.4. REINSTALLING RPMS

1. Disable the OpenShift Container Platform 3.11 repositories, and re-enable the 3.10 repositories:

```
# subscription-manager repos \
    --disable=rhel-7-server-ose-3.11-rpms \
    --enable=rhel-7-server-ose-3.10-rpms
```

2. On each master and node host, install the following packages:

```
# yum install atomic-openshift \
    atomic-openshift-node \
    atomic-openshift-docker-excluder \
    atomic-openshift-excluder \
    atomic-openshift-clients \
    atomic-openshift-hyperkube
```

3. On each host, verify the packages were installed successfully:

```
# rpm -qa | grep atomic-openshift
```

5.5. BRINGING OPENSHIFT CONTAINER PLATFORM SERVICES BACK ONLINE

After you finish your changes, bring OpenShift Container Platform back online.

Procedure

1. On each OpenShift Container Platform master, restore your master and node configuration from backup and enable and restart all relevant services:

```
# cp ${MYBACKUPDIR}/etc/origin/node/pods/* /etc/origin/node/pods/
# cp ${MYBACKUPDIR}/etc/origin/master/master.env /etc/origin/master/master.env
# cp ${MYBACKUPDIR}/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/master/scheduler.json.<timestamp>
/etc/origin/master/scheduler.json
# master-restart api
# master-restart controllers
```

2. On each OpenShift Container Platform node, update the node configuration maps as needed, 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
```

5.6. VERIFYING THE DOWNGRADE

To verify the downgrade:

1. 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. Verify the successful downgrade of the registry and router, if deployed:

   a. Verify you are running the **v3.10** versions of the **docker-registry** and **router** images:

```
# oc get -n default dc/docker-registry -o json | grep \"image\"
    "image": "openshift3/ose-docker-registry:v3.10",
# oc get -n default dc/router -o json | grep \"image\"
    "image": "openshift3/ose-haproxy-router:v3.10",
```

   b. Verify that **docker-registry** and **router** pods are running and in ready state:

```
# oc get pods -n default

NAME                    READY     STATUS    RESTARTS   AGE
docker-registry-2-b7xbn   1/1       Running   0          18m
router-2-mvq6p            1/1       Running   0          6m
```

3. Use the diagnostics tool on the master to look for common issues and provide suggestions:

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