OpenShift Container Platform 4.20

Installing a Two Node OpenShift Cluster

Installing OpenShift Container Platform on a single node

Last Updated: 2026-01-20

OpenShift Container Platform 4.20 Installing a Two Node OpenShift Cluster

Installing OpenShift Container Platform on a single node

Legal Notice

Abstract

This document describes how to install OpenShift Container Platform on a single node.

Table of Contents

CHAPTER 1. TWO-NODE WITH ARBITER

An OpenShift Container Platform cluster with two control plane nodes and one local arbiter node is a compact, cost-effective OpenShift Container Platform topology. The arbiter node stores the full etcd data, maintaining an etcd quorum and preventing split brain. The arbiter node does not run the additional control plane components **kube-apiserver** and **kube-controller-manager**, nor does it run workloads.

To install a cluster with two control plane nodes and one local arbiter node, assign an arbiter role to at least one of the nodes and set the control plane node count for the cluster to 2. Although OpenShift Container Platform does not currently impose a limit on the number of arbiter nodes, the typical deployment includes only one to minimize the use of hardware resources.

After installation, you can add additional arbiter nodes to a cluster with two control plane nodes and one local arbiter node but not to a standard multi-node cluster. It is also not possible to convert between a two-node OpenShift cluster with a local node arbiter and standard topology.

You can install a cluster with two control plane nodes and one local arbiter node by using one of the following methods:

- Installing on bare metal: Configuring a local arbiter node
- Installing with the Agent-based Installer: Configuring a local arbiter node

CHAPTER 2. TWO-NODE WITH FENCING

2.1. PREPARING TO INSTALL A TWO-NODE OPENSHIFT CLUSTER WITH FENCING

IMPORTANT

Two-node OpenShift cluster with fencing is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

A two-node OpenShift cluster with fencing provides high availability (HA) with a reduced hardware footprint. This configuration is designed for distributed or edge environments where deploying a full three-node control plane cluster is not practical.

A two-node cluster does not include compute nodes. The two control plane machines run user workloads in addition to managing the cluster.

Fencing is managed by Pacemaker, which can isolate an unresponsive node by using the Baseboard Management Console (BMC) of the node. After the unresponsive node is fenced, the remaining node can safely continue operating the cluster without the risk of resource corruption.

NOTE

You can deploy a two-node OpenShift cluster with fencing by using either the user-provisioned infrastructure method or the installer-provisioned infrastructure method.

The two-node OpenShift cluster with fencing requires the following hosts:

Table 2.1. Minimum required hosts

Hosts	Description
Two control plane machines	The control plane machines run the Kubernetes and OpenShift Container Platform services that form the control plane.
One temporary bootstrap machine	You need a bootstrap machine to deploy the OpenShift Container Platform cluster on the control plane machines. You can remove the bootstrap machine after you install the cluster.

The bootstrap and control plane machines must use Red Hat Enterprise Linux CoreOS (RHCOS) as the operating system. For instructions on installing RHCOS and starting the bootstrap process, see Installing RHCOS and starting the OpenShift Container Platform bootstrap process

NOTE

The requirement to use RHCOS applies only to user-provisioned infrastructure deployments. For installer-provisioned infrastructure deployments, the bootstrap and control plane machines are provisioned automatically by the installation program, and you do not need to manually install RHCOS.

2.1.1. Minimum resource requirements for installing the two-node OpenShift cluster with fencing

Each cluster machine must meet the following minimum requirements:

Table 2.2. Minimum resource requirements

Machine	Operating System	CPU [1]	RAM	Storage	Input/Output Per Second (IOPS) [1]
Bootstrap	RHCOS	4	16 GB	120 GB	300
Control plane	RHCOS	4	16 GB	120 GB	300

1. One CPU is equivalent to one physical core when simultaneous multithreading (SMT), or Hyper-Threading, is not enabled. When enabled, use the following formula to calculate the corresponding ratio: (threads per core × cores) × sockets = CPUs.

2. OpenShift Container Platform and Kubernetes are sensitive to disk performance, and faster storage is recommended, particularly for etcd on the control plane nodes. Note that on many cloud platforms, storage size and IOPS scale together, so you might need to over-allocate storage volume to obtain sufficient performance.

2.1.2. User-provisioned DNS requirements

In OpenShift Container Platform deployments, you must ensure that cluster components meet certain DNS name resolution criteria for internal communication, certificate validation, and automated node discovery purposes.

The following is a list of required cluster components:

- The Kubernetes API
- The OpenShift Container Platform application wildcard
- The bootstrap and control plane machines

Reverse DNS resolution is also required for the Kubernetes API, the bootstrap machine, and the control plane machines.

DNS A/AAAA or CNAME records are used for name resolution and PTR records are used for reverse name resolution. The reverse records are important because Red Hat Enterprise Linux CoreOS (RHCOS) uses the reverse records to set the hostnames for all the nodes, unless the hostnames are provided by DHCP. Additionally, the reverse records are used to generate the certificate signing requests (CSR) that OpenShift Container Platform needs to operate.

NOTE

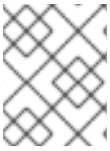
It is recommended to use a DHCP server to provide the hostnames to each cluster node. See the *DHCP recommendations for user-provisioned infrastructure* section for more information.

The following DNS records are required for a user-provisioned OpenShift Container Platform cluster and they must be in place before installation. In each record, **<cluster_name>** is the cluster name and **<base_domain>** is the base domain that you specify in the **install-config.yaml** file. A complete DNS record takes the form: **<component>.<cluster_name>.<base_domain>.**.

Table 2.3. Required DNS records

Component	Record	Description
Kubernetes API	**api.<cluster_name>.<base_domain>.**	A DNS A/AAAA or CNAME record, and a DNS PTR record, to identify the API load balancer. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster.
	api-int.<cluster_name>.<base_domain>.	A DNS A/AAAA or CNAME record, and a DNS PTR record, to internally identify the API load balancer. These records must be resolvable from all the nodes within the cluster.  IMPORTANT The API server must be able to resolve the worker nodes by the hostnames that are recorded in Kubernetes. If the API server cannot resolve the node names, then proxied API calls can fail, and you cannot retrieve logs from pods.
Routes	***.apps.<cluster_name>.<base_domain>.**	A wildcard DNS A/AAAA or CNAME record that refers to the application ingress load balancer. The application ingress load balancer targets the machines that run the Ingress Controller pods. By default, the Ingress Controller pods run on compute nodes. In cluster topologies without dedicated compute nodes, such as two-node or three-node clusters, the control plane nodes also carry the worker label, so the Ingress pods are scheduled on the control plane nodes. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster. For example, **console-openshift-console.apps.<cluster_name>.<base_domain>** is used as a wildcard route to the OpenShift Container Platform console.
Bootstrap machine	**bootstrap.<cluster_name>.<base_domain>.**	A DNS A/AAAA or CNAME record, and a DNS PTR record, to identify the bootstrap machine. These records must be resolvable by the nodes within the cluster.

Component	Record	Description
Control plane machines	**<control_plane><n>.<cluster_name>.<base_domain>.**	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the control plane nodes. These records must be resolvable by the nodes within the cluster.



NOTE

In OpenShift Container Platform 4.4 and later, you do not need to specify etcd host and SRV records in your DNS configuration.

TIP

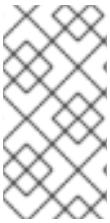
You can use the **dig** command to verify name and reverse name resolution. See the section on *Validating DNS resolution for user-provisioned infrastructure* for detailed validation steps.

2.1.2.1. Example DNS configuration for user-provisioned clusters

Reference the example DNS configurations to understand how A and PTR record configuration samples meet the DNS requirements for deploying OpenShift Container Platform on user-provisioned infrastructure.

The DNS configuration examples provided here are for reference only and are not meant to provide advice for choosing one DNS solution over another.

In the examples, the cluster name is **ocp4** and the base domain is **example.com**.



NOTE

In a two-node cluster with fencing, the control plane machines are also schedulable worker nodes. The DNS configuration must therefore include only the two control plane nodes. If you later add compute machines, provide corresponding A and PTR records for them as in a standard user-provisioned installation.

The following example is a BIND zone file that shows sample DNS A records for name resolution in a user-provisioned cluster.

NOTE

In the example, the same load balancer is used for the Kubernetes API and application ingress traffic. In production scenarios, you can deploy the API and application ingress load balancers separately so that you can scale the load balancer infrastructure for each in isolation.

```
$TTL 1W
@ IN SOA ns1.example.com. root (
  2019070700 ; serial
  3H  ; refresh (3 hours)
  30M  ; retry (30 minutes)
  2W  ; expiry (2 weeks)
```

```
    1W )  ; minimum (1 week)
  IN NS ns1.example.com.
  IN MX 10 smtp.example.com.
 ;
 ;
 ns1.example.com.  IN A 192.168.1.5
 smtp.example.com.  IN A 192.168.1.5
 ;
 helper.example.com.  IN A 192.168.1.5
 helper.ocp4.example.com. IN A 192.168.1.5
 ;
 api.ocp4.example.com.  IN A 192.168.1.5
 api-int.ocp4.example.com. IN A 192.168.1.5
 ;
 *.apps.ocp4.example.com. IN A 192.168.1.5
 ;
 bootstrap.ocp4.example.com. IN A 192.168.1.96
 ;
 control-plane0.ocp4.example.com. IN A 192.168.1.97
 control-plane1.ocp4.example.com. IN A 192.168.1.98
 ;
 ;
 ;EOF
```

where:

api.ocp4.example.com.

Provides name resolution for the Kubernetes API. The record refers to the IP address of the API load balancer.

api-int.ocp4.example.com.

Provides name resolution for the Kubernetes API. The record refers to the IP address of the API load balancer and is used for internal cluster communications.

***.apps.ocp4.example.com.**

Provides name resolution for the wildcard routes. The record refers to the IP address of the application ingress load balancer. The application ingress load balancer targets the machines that run the Ingress Controller pods.

bootstrap.ocp4.example.com

Provides name resolution for the bootstrap machine.

control-plane0.ocp4.example.com

Provides name resolution for the control plane machines.

The following example BIND zone file shows sample PTR records for reverse name resolution in a user-provisioned cluster:

```
$TTL 1W
@ IN SOA ns1.example.com. root (
   2019070700 ; serial
   3H  ; refresh (3 hours)
   30M  ; retry (30 minutes)
   2W  ; expiry (2 weeks)
   1W )  ; minimum (1 week)
 IN NS ns1.example.com.
```

```
;
5.1.168.192.in-addr.arpa. IN PTR api.ocp4.example.com.
5.1.168.192.in-addr.arpa. IN PTR api-int.ocp4.example.com.
;
96.1.168.192.in-addr.arpa. IN PTR bootstrap.ocp4.example.com.
;
97.1.168.192.in-addr.arpa. IN PTR control-plane0.ocp4.example.com.
98.1.168.192.in-addr.arpa. IN PTR control-plane1.ocp4.example.com.
;
;
;EOF
```

where:

api.ocp4.example.com.

Provides reverse DNS resolution for the Kubernetes API. The PTR record refers to the record name of the API load balancer.

api-int.ocp4.example.com.

Provides reverse DNS resolution for the Kubernetes API. The PTR record refers to the record name of the API load balancer and is used for internal cluster communications.

bootstrap.ocp4.example.com.

Provides reverse DNS resolution for the bootstrap machine.

control-plane0.ocp4.example.com.

Provides rebootstrap.ocp4.example.com.verse DNS resolution for the control plane machines.

NOTE

A PTR record is not required for the OpenShift Container Platform application wildcard.

2.1.3. Installer-provisioned DNS requirements

Clients access the OpenShift Container Platform cluster nodes over the **baremetal** network. A network administrator must configure a subdomain or subzone where the canonical name extension is the cluster name.

```
<cluster_name>.<base_domain>
```

For example:

```
test-cluster.example.com
```

OpenShift Container Platform includes functionality that uses cluster membership information to generate A/AAAA records. This resolves the node names to their IP addresses. After the nodes are registered with the API, the cluster can disperse node information without using CoreDNS-mDNS. This eliminates the network traffic associated with multicast DNS.

CoreDNS requires both TCP and UDP connections to the upstream DNS server to function correctly. Ensure the upstream DNS server can receive both TCP and UDP connections from OpenShift Container Platform cluster nodes.

In OpenShift Container Platform deployments, DNS name resolution is required for the following components:

- The Kubernetes API
- The OpenShift Container Platform application wildcard ingress API

A/AAAA records are used for name resolution and PTR records are used for reverse name resolution. Red Hat Enterprise Linux CoreOS (RHCOS) uses the reverse records or DHCP to set the hostnames for all the nodes.

Installer-provisioned installation includes functionality that uses cluster membership information to generate A/AAAA records. This resolves the node names to their IP addresses. In each record, **<cluster_name>** is the cluster name and **<base_domain>** is the base domain that you specify in the **install-config.yaml** file. A complete DNS record takes the form: **<component>.<cluster_name>.<base_domain>.**.

Table 2.4. Required DNS records

Component	Record	Description
Kubernetes API	**api.<cluster_name>.<base_domain>.**	An A/AAAA record and a PTR record identify the API load balancer. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster.
Routes	***.apps.<cluster_name>.<base_domain>.**	The wildcard A/AAAA record refers to the application ingress load balancer. The application ingress load balancer targets the nodes that run the Ingress Controller pods. The Ingress Controller pods run on the worker nodes by default. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster. For example, **console-openshift-console.apps.<cluster_name>.<base_domain>** is used as a wildcard route to the OpenShift Container Platform console.

TIP

You can use the **dig** command to verify DNS resolution.

2.1.4. Configuring an Ingress load balancer for a two-node cluster with fencing

You must configure an external Ingress load balancer (LB) before you install a two-node OpenShift cluster with fencing. The Ingress LB forwards external application traffic to the Ingress Controller pods that run on the control plane nodes. Both nodes can actively receive traffic.

Prerequisites

- You have two control plane nodes with fencing enabled.
- You have network connectivity from the load balancer to both control plane nodes.
- You created DNS records for **api.<cluster_name>.<base_domain>** and ***.apps.<cluster_name>.<base_domain>**.

- You have an external load balancer that supports health checks on endpoints.

Procedure

1. Configure the load balancer to forward traffic for the following ports:
   - **6443**: Kubernetes API server
   - **80** and **443**: Application ingress
     You must forward traffic to both control plane nodes.
2. Configure health checks on the load balancer. You must monitor the backend endpoints so that the load balancer only sends traffic to nodes that respond.
3. Configure the load balancer to forward traffic to both control plane nodes. The following example shows how to configure two control plane nodes:

```
frontend api_frontend
    bind *:6443
    mode tcp
    default_backend api_backend

backend api_backend
    mode tcp
    balance roundrobin
    server cp0 <cp0_ip>:6443 check
    server cp1 <cp1_ip>:6443 check

frontend ingress_frontend
    bind *:80
    bind *:443
    mode tcp
    default_backend ingress_backend

backend ingress_backend
    mode tcp
    balance roundrobin
    server cp0 <cp0_ip>:80 check
    server cp1 <cp1_ip>:80 check
    server cp0 <cp0_ip>:443 check
    server cp1 <cp1_ip>:443 check
```

4. Verify the load balancer configuration:
   a. From an external client, run the following command:

```
$ curl -k https://api.<cluster_name>.<base_domain>:6443/version
```

   b. From an external client, access an application route by running the following command:

```
$ curl https://<app>.<cluster_name>.<base_domain>
```

You can shut down a control plane node and verify that the load balancer stops sending traffic to that node while the other node continues to serve requests.

2.1.5. Creating a manifest object for a customized br-ex bridge

You must create a manifest object to modify the cluster's network configuration after installation. The manifest configures the br-ex bridge, which manages external network connectivity for the cluster.

For instructions on creating this manifest, "Creating a manifest file for a customized br-ex bridge".

2.1.6. Additional resources

- Creating a manifest file for a customized br-ex bridge
- Configuring and managing high availability clusters in RHEL .

2.2. INSTALLING A TWO-NODE OPENSHIFT CLUSTER WITH FENCING

IMPORTANT

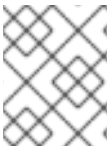
Two-node OpenShift cluster with fencing is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can deploy a two-node OpenShift cluster with fencing by using either the installer-provisioned infrastructure or the user-provisioned infrastructure installation method. The following examples provide sample **install-config.yaml** configurations for both methods.

2.2.1. Sample install-config.yaml for a two-node installer-provisioned infrastructure cluster with fencing

You can use the following **install-config.yaml** configuration as a template for deploying a two-node OpenShift cluster with fencing by using the installer-provisioned infrastructure method:



NOTE

Do an etcd backup before proceeding to ensure that you can restore the cluster if any issues occur.

Sample install-config.yaml configuration

```
apiVersion: v1
baseDomain: example.com
compute:
- name: worker
  replicas: 0
controlPlane:
  name: master
  replicas: 2
  fencing:
```

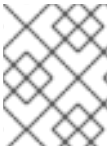
```
    credentials:
      - hostname: <control_0_hostname>
        address: https://<redfish-api-url>
        username: <username>
        password: <password>
        certificateVerification: Disabled
      - hostname: <control_1_hostname>
        address: https://<redfish-api-url>
        username: <username>
        password: <password>
        certificateVerification: Enabled
metadata:
  name: <cluster_name>
featureSet: TechPreviewNoUpgrade
platform:
  baremetal:
    apiVIPs:
      - <api_ip>
    ingressVIPs:
      - <wildcard_ip>
    hosts:
      - name: <control_0_hostname>
        role: master
        bmc:
          address: <bmc_address>
          username: <bmc_username>
          password: <bmc_password>
        bootMACAddress: <boot_mac>
      - name: <control_1_hostname>
        role: master
        bmc:
          address: <bmc_address>
          username: <bmc_username>
          password: <bmc_password>
        bootMACAddress: <boot_mac>
pullSecret: '<pull_secret>'
sshKey: '<ssh_public_key>'
```

- **compute.replicas**: Set this field to **0** because a two-node fencing cluster does not include worker nodes.
- **controlPlane.replicas**: Set this field to **2** for a two-node fencing deployment.
- **fencing.credentials.hostname**: Provide the Baseboard Management Console (BMC) credentials for each control plane node. These credentials are required for node fencing and prevent split-brain scenarios.
- **fencing.credentials.certificateVerification**: Set this field to **Disabled** if your Redfish URL uses self-signed certificates, which is common for internally-hosted endpoints. Set this field to **Enabled** for URLs with valid CA-signed certificates.
- **metadata.name**: The cluster name is used as a prefix for hostnames and DNS records.
- **featureSet**: Set this field to **TechPreviewNoUpgrade** to enable two-node OpenShift cluster deployments.

- **platform.baremetal.apiVIPs** and **platform.baremetal.ingressVIPs** : Virtual IPs for the API and Ingress endpoints. Ensure they are reachable by all nodes and external clients.
- **pullSecret**: Contains credentials required to pull container images for the cluster components.
- **sshKey**: The SSH public key for accessing cluster nodes after installation.

2.2.2. Sample install-config.yaml for a two-node user-provisioned infrastructure cluster with fencing

You can use the following **install-config.yaml** configuration as a template for deploying a two-node OpenShift cluster with fencing by using the user-provisioned infrastructure method:



NOTE

Do an etcd backup before proceeding to ensure that you can restore the cluster if any issues occur.

Sample install-config.yaml configuration

```
apiVersion: v1
baseDomain: example.com
compute:
- name: worker
  replicas: 0
controlPlane:
  name: master
  replicas: 2
  fencing:
    credentials:
      - hostname: <control_0_hostname>
        address: https://<redfish-api-url>
        username: <username>
        password: <password>
      - hostname: <control_1_hostname>
        address: https://<redfish-api-url>
        username: <username>
        password: <password>
metadata:
  name: <cluster_name>
featureSet: TechPreviewNoUpgrade
platform:
  none: {}
pullSecret: '<pull_secret>'
sshKey: '<ssh_public_key>'
```

- **compute.replicas**: Set this field to **0** because a two-node fencing cluster does not include worker nodes.
- **controlPlane.replicas**: Set this field to **2** for a two-node fencing deployment.
- **fencing.credentials.hostname**: Provide BMC credentials for each control plane node.
- **metadata.name**: Cluster name is used as a prefix for hostnames and DNS records.

- **featureSet**: Enables two-node OpenShift cluster deployments.
- **platform.none** Set the platform to **none** for user-provisioned infrastructure deployments. Bare-metal hosts are pre-provisioned outside of the installation program.
- **pullSecret**: Contains credentials required to pull container images for the cluster components.
- **sshKey**: The SSH public key for accessing cluster nodes after installation.

2.3. POST-INSTALLATION TROUBLESHOOTING AND RECOVERY

IMPORTANT

Two-node OpenShift cluster with fencing is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Use the following sections help you with recovering from issues in a two-node OpenShift cluster with fencing.

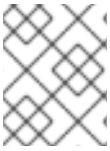
2.3.1. Manually recovering from a disruption event when automated recovery is unavailable

You might need to perform manual recovery steps if a disruption event prevents fencing from functioning correctly. In this case, you can run commands directly on the control plane nodes to recover the cluster. There are four main recovery scenarios, which should be attempted in the following order:

1. Update fencing secrets: Refresh the Baseboard Management Console (BMC) credentials if they are incorrect or outdated.
2. Recover from a single-node failure: Restore functionality when only one control plane node is down.
3. Recover from a complete node failure: Restore functionality when both control plane nodes are down.
4. Replace a control plane node that cannot be recovered: Replace the node to restore cluster functionality.

Prerequisites

- You have administrative access to the control plane nodes.
- You can connect to the nodes by using SSH.



NOTE

Do an etcd backup before proceeding to ensure that you can restore the cluster if any issues occur.

Procedure

1. Update the fencing secrets:
   a. If the Cluster API is unavilable, update fencing secret by running the following command on one of the cluster nodes:

      ```
      $ sudo pcs stonith update <node_name>_redfish username=<user_name> password=
      <password>
      ```

      After the Cluster API recovers, or the Cluster API is already available, update fencing secret in the cluster to ensure it stays in sync, as described in the following step.

   b. Edit the username and password for the existing fencing secret for the control plane node by running the following commads:

      ```
      $ oc project openshift-etcd
      ```

      ```
      $ oc edit secret <node_name>-fencing
      ```

      If the cluster recovers after updating the fencing secrets, no further action is required. If the issue persists, proceed to the next step.

2. Recover from a single-node failure:
   a. Gather initial diagnostics by running the following command:

      ```
      $ sudo pcs status --full
      ```

      This command provides a detailed view of the current cluster and resource states. You can use the output to identify issues with fencing or etcd startup.

   b. Run the following additional diagnostic commands, if necessary:
      Reset the resources on your cluster and instruct Pacemaker to attempt to start them fresh by running the following command:

      ```
      $ sudo pcs resource cleanup
      ```

      Review all Pacemaker activity on the node by running the following command:

      ```
      $ sudo journalctl -u pacemaker
      ```

      Diagnose etcd resource startup issues by running the following command:

      ```
      $ sudo journalctl -u pacemaker | grep podman-etcd
      ```

   c. View the fencing configuration for the node by running the following command:

      ```
      $ sudo pcs stonith config <node_name>_redfish
      ```

If fencing is required but is not functioning, ensure that the Redfish fencing endpoint is accessible and verify that the credentials are correct.

d. If etcd is not starting despite fencing being operational, restore etcd from a backup by running the following commands:

```
$ sudo cp -r /var/lib/etcd-backup/* /var/lib/etcd/
```

```
$ sudo chown -R etcd:etcd /var/lib/etcd
```

If the recovery is successful, no further action is required. If the issue persists, proceed to the next step.

3. Recover from a complete node failure:

a. Power on both control plane nodes.
Pacemaker starts automatically and begins the recovery operation when it detects both nodes are online. If the recovery does not start as expected, use the diagnostic commands described in the previous step to investigate the issue.

b. Reset the resources on your cluster and instruct Pacemaker to attempt to start them fresh by running the following command:

```
$ sudo pcs resource cleanup
```

c. Check resource start order by running the following command:

```
$ sudo pcs status --full
```

d. Inspect the pacemaker service journal if kubelet fails by running the following commands:

```
$ sudo journalctl -u pacemaker
```

```
$ sudo journalctl -u kubelet
```

e. Handle out-of-sync etcd.
If one node has a more up-to-date etcd, Pacemaker attempts to fence the lagging node and start it as a learner. If this process stalls, verify the Redfish fencing endpoint and credentials by running the following command:

```
$ sudo pcs stonith config
```

If the recovery is successful, no further action is required. If the issue persists, perform manual recovery as described in the next step.

4. If you need to manually recover from an event when one of the nodes is not recoverable, follow the procedure in "Replacing control plane nodes in a two-node OpenShift cluster".
When a cluster loses a single node, it enters the degraded mode. In this state, Pacemaker automatically unblocks quorum and allows the cluster to temporarily operate on the remaining node.

If both nodes fail, you must restart both nodes to reestablish quorum so that Pacemaker can resume normal cluster operations.

If only one of the two nodes can be restarted, follow the node replacement procedure to manually reestablish quorum on the surviving node.

If manual recovery is still required and it fails, collect a must-gather and SOS report, and file a bug.

Verification

For information about verifying that both control plane nodes and etcd are operating correctly, see "Verifying etcd health in a two-node OpenShift cluster with fencing".

2.3.2. Additional resources

- Restoring etcd from a backup .
- Verifying etcd health in a two-node OpenShift cluster with fencing

2.3.3. Replacing control plane nodes in a two-node OpenShift cluster with fencing

You can replace a failed control plane node in a two-node OpenShift cluster. The replacement node must use the same host name and IP address as the failed node.

Prerequisites

- You have a functioning survivor control plane node.
- You have verified that either the machine is not running or the node is not ready.
- You have access to the cluster as a user with the **cluster-admin** role.
- You know the host name and IP address of the failed node.

NOTE

Do an etcd backup before proceeding to ensure that you can restore the cluster if any issues occur.

Procedure

1. Check the quorum state by running the following command:

```
$ sudo pcs quorum status
```

Example output

```
Quorum information
------------------
Date:             Fri Oct  3 14:15:31 2025
Quorum provider:  corosync_votequorum
Nodes:            2
Node ID:          1
Ring ID:          1.16
Quorate:          Yes

Votequorum information
```

```
----------------------
Expected votes:   2
Highest expected: 2
Total votes:      2
Quorum:           1
Flags:            2Node Quorate WaitForAll

Membership information
----------------------
    Nodeid      Votes    Qdevice Name
         1          1         NR master-0 (local)
         2          1         NR master-1
```

a. If quorum is lost and one control plane node is still running, restore quorum manually on the survivor node by running the following command:

```
$ sudo pcs quorum unblock
```

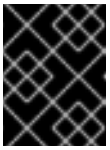
b. If only one node failed, verify that etcd is running on the survivor node by running the following command:

```
$ sudo pcs resource status etcd
```

c. If etcd is not running, restart etcd by running the following command:

```
$ sudo pcs resource cleanup etcd
```

If etcd still does not start, force it manually on the survivor node, skipping fencing:



IMPORTANT

Before running this commands, ensure that the node being replaced is inaccessible. Otherwise, you risk etcd corruption.

```
$ sudo pcs resource debug-stop etcd
```

```
$ sudo OCF_RESKEY_CRM_meta_notify_start_resource='etcd' pcs resource debug-start etcd
```

After recovery, etcd must be running successfully on the survivor node.

2. Delete etcd secrets for the failed node by running the following commands:

```
$ oc project openshift-etcd
```

```
$ oc delete secret etcd-peer-<node_name>
```

```
$ oc delete secret etcd-serving-<node_name>
```

```
$ oc delete secret etcd-serving-metrics-<node_name>
```

NOTE

To replace the failed node, you must delete its etcd secrets first. When etcd is running, it might take some time for the API server to respond to these commands.

3. Delete resources for the failed node:

   a. If you have the **BareMetalHost** (BMH) objects, list them to identify the host you are replacing by running the following command:

   ```
   $ oc get bmh -n openshift-machine-api
   ```

   b. Delete the BMH object for the failed node by running the following command:

   ```
   $ oc delete bmh/<bmh_name> -n openshift-machine-api
   ```

   c. List the **Machine** objects to identify the object that maps to the node that you are replacing by running the following command:

   ```
   $ oc get machines.machine.openshift.io -n openshift-machine-api
   ```

   d. Get the label with the machine hash value from the **Machine** object by running the following command:

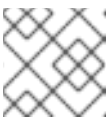
   ```
   $ oc get machines.machine.openshift.io/<machine_name> -n openshift-machine-api \
     -o jsonpath='Machine hash label: {.metadata.labels.machine\.openshift\.io/cluster-api-cluster}{"\n"}'
   ```

   Replace **<machine_name>** with the name of a **Machine** object in your cluster. For example, **ostest-bfs7w-ctrlplane-0**.

   You need this label to provision a new **Machine** object.

   e. Delete the **Machine** object for the failed node by running the following command:

   ```
   $ oc delete machines.machine.openshift.io/<machine_name>-<failed nodename> -n openshift-machine-api
   ```

   NOTE

   The node object is deleted automatically after deleting the **Machine** object.

4. Recreate the failed host by using the same name and IP address:

IMPORTANT

You must perform this step only if you are using installer-provisioned infrastructure or the Machine API to create the original node. For information about replacing a failed bare-metal control plane node, see "Replacing an unhealthy etcd member on bare metal".

a. Remove the BMH and **Machine** objects. The machine controller automatically deletes the node object.

b. Provision a new machine by using the following sample configuration:

Example Machine object configuration

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  annotations:
    metal3.io/BareMetalHost: openshift-machine-api/{bmh_name}
  finalizers:
  - machine.machine.openshift.io
  labels:
    machine.openshift.io/cluster-api-cluster: {machine_hash_label}
    machine.openshift.io/cluster-api-machine-role: master
    machine.openshift.io/cluster-api-machine-type: master
  name: {machine_name}
  namespace: openshift-machine-api
spec:
  authoritativeAPI: MachineAPI
  metadata: {}
  providerSpec:
    value:
      apiVersion: baremetal.cluster.k8s.io/v1alpha1
      customDeploy:
        method: install_coreos
      hostSelector: {}
      image:
        checksum: ""
        url: ""
      kind: BareMetalMachineProviderSpec
      metadata:
        creationTimestamp: null
      userData:
        name: master-user-data-managed
```

- **metadata.annotations.metal3.io/BareMetalHost**: Replace **{bmh_name}** with the name of the BMH object that is associated with the host that you are replacing.
- **labels.machine.openshift.io/cluster-api-cluster**: Replace **{machine_hash_label}** with the label that you fetched from the machine you deleted.
- **metadata.name**: Replace **{machine_name}** with the name of the machine you deleted.

c. Create the new BMH object and the secret to store the BMC credentials by running the following command:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openshift-machine-api
```

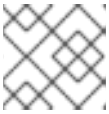
```
  data:
    password: <password>
    username: <username>
  type: Opaque
  ---
  apiVersion: metal3.io/v1alpha1
  kind: BareMetalHost
  metadata:
    name: {bmh_name}
    namespace: openshift-machine-api
  spec:
    automatedCleaningMode: disabled
    bmc:
      address: <redfish_url>/{uuid}
      credentialsName: <name>
      disableCertificateVerification: true
    bootMACAddress: {boot_mac_address}
    bootMode: UEFI
    externallyProvisioned: false
    online: true
    rootDeviceHints:
      deviceName: /dev/disk/by-id/scsi-<serial_number>
    userData:
      name: master-user-data-managed
      namespace: openshift-machine-api
  EOF
```

- **metadata.name**: Specify the name of the secret.
- **metadata.name**: Replace **{bmh_name}** with the name of the BMH object that you deleted.
- **bmc.address**: Replace **{uuid}** with the UUID of the node that you created.
- **bmc.credentialsName**: Replace **name** with the name of the secret that you created.
- **bootMACAddress**: Specify the MAC address of the provisioning network interface. This is the MAC address the node uses to identify itself when communicating with Ironic during provisioning.

5. Verify that the new node has reached the **Provisioned** state by running the following command:

```
$ oc get bmh -o wide
```

The value of the **STATUS** column in the output of this command must be **Provisioned**.



NOTE

The provisioning process can take 10 to 20 minutes to complete.

6. Verify that both control plane nodes are in the **Ready** state by running the following command:

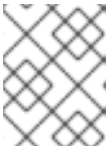
```
$ oc get nodes
```

The value of the **STATUS** column in the output of this command must be **Ready** for both nodes.

7. Apply the **detached** annotation to the BMH object to prevent the Machine API from managing it by running the following command:

```
$ oc annotate bmh <bmh_name> -n openshift-machine-api
baremetalhost.metal3.io/detached=" --overwrite
```

8. Rejoin the replacement node to the pacemaker cluster by running the following command:



NOTE

Run the following command on the survivor control plane node, not the node being replaced.

```
$ sudo pcs cluster node remove <node_name>
```

```
$ sudo pcs cluster node add <node_name> addr=<node_ip> --start --enable
```

9. Delete stale jobs for the failed node by running the following command:

```
$ oc project openshift-etcd
```

```
$ oc delete job tnf-auth-job-<node_name>
```

```
$ oc delete job tnf-after-setup-job-<node_name>
```

Verification

For information about verifying that both control plane nodes and etcd are operating correctly, see "Verifying etcd health in a two-node OpenShift cluster with fencing".

2.3.4. Additional resources

- Restoring etcd from a backup .

2.3.5. Verifying etcd health in a two-node OpenShift cluster with fencing

After completing node recovery or maintenance procedures, verify that both control plane nodes and etcd are operating correctly.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges.
- You can access at least one control plane node through SSH.

Procedure

1. Check the overall node status by running the following command:

```
$ oc get nodes
```

This command verifies that both control plane nodes are in the **Ready** state, indicating that they can receive workloads for scheduling.

2. Verify the status of the **cluster-etcd-operator** by running the following command:

```
$ oc describe co/etcd
```

The **cluster-etcd-operator** manages and reports on the health of your etcd setup. Reviewing its status helps you identify any ongoing issues or degraded conditions.

3. Review the etcd member list by running the following command:

```
$ oc rsh -n openshift-etcd <etcd_pod> etcdctl member list -w table
```

This command shows the current etcd members and their roles. Look for any nodes marked as **learner**, which indicates that they are in the process of becoming voting members.

4. Review the Pacemaker resource status by running the following command on either control plane node:

```
$ sudo pcs status --full
```

This command provides a detailed overview of all resources managed by Pacemaker. You must ensure that the following conditions are met:

- Both nodes are online.
- The **kubelet** and **etcd** resources are running.
- Fencing is correctly configured for both nodes.