OpenShift Container Platform 4.20

etcd

Providing redundancy with etcd

Last Updated: 2025-12-17

OpenShift Container Platform 4.20 etcd

Providing redundancy with etcd

Legal Notice

Abstract

This document provides instructions for using etcd, which ensures a reliable approach to cluster configuration and resiliency in OpenShift Container Platform.

Table of Contents

CHAPTER 1. OVERVIEW OF ETCD

etcd (pronounced et-see-dee) is a consistent, distributed key-value store that stores small amounts of data across a cluster of machines that can fit entirely in memory. As the core component of many projects, etcd is also the primary data store for Kubernetes, which is the standard system for container orchestration.

By using etcd, you can benefit in several ways:

- Support consistent uptime for your cloud-native applications, and keep them working even if individual servers fail
- Store and replicate all cluster states for Kubernetes
- Distribute configuration data to offer redundancy and resiliency for the configuration of nodes

IMPORTANT

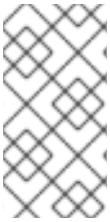
The default etcd configuration optimizes container orchestration. Use it as designed for the best results.

1.1. HOW ETCD WORKS

To ensure a reliable approach to cluster configuration and management, etcd uses the etcd Operator. The Operator simplifies the use of etcd on a Kubernetes container platform such as OpenShift Container Platform.

Additionally, you can use the etcd Operator to deploy and manage the etcd cluster for the OpenShift Container Platform control plane. The etcd Operator manages the cluster state in the following ways:

- Observes the cluster state by using the Kubernetes API
- Analyzes differences between the current state and the required state
- Corrects the differences through the etcd cluster management APIs, the Kubernetes API, or both



NOTE

etcd holds the cluster state, which is constantly updated. This state is continuously persisted, which leads to a high number of small changes at high frequency. As a result, it is critical to back up the etcd cluster member with fast, low-latency I/O. For more information about best practices for etcd, see "Recommended etcd practices".

Additional resources

- Recommended etcd practices

1.2. UNDERSTANDING ETCD PERFORMANCE

As a consistent distributed key-value store operating as a cluster of replicated nodes, etcd follows the Raft algorithm by electing one node as the leader and the others as followers. The leader maintains the current state of the system current state and ensures that the followers are up-to-date.

The leader node is responsible for log replication. It handles incoming write transactions from the client and writes a Raft log entry that it then broadcasts to the followers.

When an etcd client such as **kube-apiserver** connects to an etcd member that is requesting an action that requires a quorum, such as writing a value, if the etcd member is a follower, it returns a message indicating that the transaction needs to go to the leader.

When the etcd client requests an action from the leader that requires a quorum, such as writing a value, the leader maintains the client connection open while it writes the local Raft log, broadcasts the log to the followers, and waits for the majority of the followers to acknowledge to have committed the log without failures. The leader sends the acknowledgment to the etcd client and closes the session. If failure notifications are received from the followers and a consensus is not met, the leader returns the error message to the client and closes the session.

OpenShift Container Platform timer conditions for etcd

OpenShift Container Platform maintains etcd timers that are optimized for each platform. OpenShift Container Platform has prescribed validated values that are optimized for each platform provider. The default **etcd timers** parameters with **platform=none** or **platform=metal** values are as follows:

```
- name: ETCD_ELECTION_TIMEOUT 1
  value: "1000"
  ...
- name: ETCD_HEARTBEAT_INTERVAL 2
  value: "100"
```

1 This timeout is how long a follower node waits without hearing a heartbeat before it attempts to become the leader.

2 The frequency that the leader notifies followers that it is still the leader.

These parameters do not provide all of the information for the control plane or for etcd. An etcd cluster is sensitive to disk latencies. Because etcd must persist proposals to its log, disk activity from other processes might cause long **fsync** latencies. The consequence is that etcd might miss heartbeats, causing request timeouts and temporary leader loss. During a leader loss and reelection, the Kubernetes API cannot process any request that causes a service-affecting event and instability of the cluster.

Effects of disk latency on etcd

An etcd cluster is sensitive to disk latencies. To understand the disk latency that etcd experiences by etcd in your control plane environment, run the Flexible I/O Tester (fio) tests or suite, to check etcd disk performance in OpenShift Container Platform.

IMPORTANT

Use only the **fio** test to measure disk latency at a specific point in time. This test does not account for long-term disk behavior and other disk workloads that occur with etcd in a production environment.

Ensure that the final report classifies the disk as appropriate for etcd, as shown in the following example:

```
...
99th percentile of fsync is 5865472 ns
99th percentile of the fsync is within the suggested threshold: - 20 ms, the disk can be used to host
etcd
```

When a high latency disk is used, a message states that the disk is not suggested for etcd, as shown in the following example:

```
...
99th percentile of fsync is 15865472 ns
99th percentile of the fsync is greater than the suggested value which is 20 ms, faster disks are suggested to host etcd for better performance
```

When your cluster deployments span many data centers that are using disks for etcd that do not meet the suggested latency, service-affecting failures can occur. In addition, the network latency that the control plane can sustain is dramatically reduced.

Effects of network latency and jitter on etcd

Use the tools that are described in the maximum transmission unit (MTU) discovery and validation section to obtain the average and maximum network latency.

The value of the heartbeat interval should be approximately the maximum of the average round-trip time (RTT) between members, normally around 1.5 times the round-trip time. With the OpenShift Container Platform default heartbeat interval of 100 ms, the suggested RTT between control plane nodes is less than 33 ms, with a maximum of less than 66 ms (66 ms x 1.5 = 99 ms). Any network latency that is larger might cause service-affecting events and cluster instability.

The network latency is determined by factors that include the technology of the transport networks, such as copper, fiber, wireless, or satellite, the number and quality of the network devices in the transport network, and other factors.

Consider network latency with network jitter for exact calculations. *Network jitter* is the variance in network latency or the variation in the delay of received packets. In efficient network conditions, the jitter should be zero. Network jitter affects the network latency calculations for etcd because the actual network latency over time will be the RTT plus or minus Jitter.

For example, a network with a maximum latency of 80 ms and jitter of 30 ms will experience latencies of 110 ms, which means etcd will miss heartbeats. This condition results in request timeouts and temporary leader loss. During a leader loss and re-election, the Kubernetes API cannot process any request that causes a service-affecting event and instability of the cluster.

Effects of consensus latency on etcd

The procedure can run only on an active cluster. The disk or network test should be completed while you plan a cluster deployment. That test validates and monitors cluster health after a deployment.

By using the **etcdctl** CLI, you can watch the latency for reaching consensus as experienced by etcd. You must identify one of the etcd pods and then retrieve the endpoint health.

etcd peer round trip time impacts on performance

The etcd peer round trip time is not the same as the network round trip time. This calculation is an end-to-end test metric about how quickly replication can occur among members.

The etcd peer round trip time is the metric that shows the latency of etcd to finish replicating a client request among all the etcd members. The OpenShift Container Platform console provides dashboards to visualize the various etcd metrics. In the console, click **Observe → Dashboards**. From the dropdown list, select **etcd**.

A plot that summarizes the etcd peer round trip time is near the end of the etcd **Dashboard** page.

Effects of database size on etcd

The etcd database size has a direct impact on the time to complete the etcd defragmentation process. OpenShift Container Platform automatically runs the etcd defragmentation on one etcd member at a time when it detects at least 45% fragmentation. During the defragmentation process, the etcd member cannot process any requests. On small etcd databases, the defragmentation process happens in less than a second. With larger etcd databases, the disk latency directly impacts the fragmentation time, causing additional latency, as operations are blocked while defragmentation happens.

The size of the etcd database is a factor to consider when network partitions isolate a control plane node for a period of time, and the control plane needs to sync after communication is re-established.

Minimal options exist for controlling the size of the etcd database, because it depends on the Operators and applications in the system. When you consider the latency range where the system operates, account for the effects of synchronization or defragmentation per size of the etcd database.

The magnitude of the effects is specific to the deployment. The time to complete a defragmentation will cause degradation in the transaction rate, as the etcd member cannot accept updates during the defragmentation process. Similarly, the time for the etcd re-synchronization for large databases with high change rate affects the transaction rate and transaction latency on the system. Consider the following two examples for the type of impacts to plan for.

The first example of the effect of etcd defragmentation based on database size is that writing an etcd database of 1 GB to a slow 7200 RPMs disk at 80 Mb per second takes about 1 minute and 40 seconds. In such a scenario, the defragmentation process takes at least this long, to complete the defragmentation.

The second example of the effect of database size on etcd synchronization is that if there is a change of 10% of the etcd database during disconnection of one of the control plane nodes, the sync needs to transfer at least 100 MB. Transferring 100 MB over a 1 Gbps link takes 800 ms. On clusters with regular transactions with the Kubernetes API, the larger the etcd database size, the more network instabilities will cause control plane instabilities.

In OpenShift Container Platform, the etcd dashboard has a plot that reports the size of the etcd database. Alternatively, you can obtain the database size from the CLI by using the **etcdctl** tool.

```
# oc get pods -n openshift-etcd -l app=etcd
```

Example output

```
NAME     READY   STATUS    RESTARTS   AGE
etcd-m0   4/4     Running   4          22h
etcd-m1   4/4     Running   4          22h
etcd-m2   4/4     Running   4          22h
```

```
# oc exec -t etcd-m0 -- etcdctl endpoint status -w simple | cut -d, -f 1,3,4
```

Example output

```
https://198.18.111.12:2379, 3.5.6, 1.1 GB
https://198.18.111.13:2379, 3.5.6, 1.1 GB
https://198.18.111.14:2379, 3.5.6, 1.1 GB
```

Effects of the Kubernetes API transaction rate on etcd

When you are using a stretched control plane, the Kebernetes API transaction rate depends on the characteristics of the particular deployment. It depends on the combination of the etcd disk latency, the etcd round trip time, and the size of objects that are written to the API. As a result, when you use stretched control planes, the cluster administrators need to test the environment to determine the sustained transaction rate that is possible for their environment. The **kube-burner** tool can be used for this purpose.

Determining Kubernetes API transaction rate for your environment

You cannot determine the transaction rate of the Kubernetes API without measuring it. One of the tools that is used for load testing the control plane is **kube-burner**. The binary provides a OpenShift Container Platform wrapper for testing OpenShift Container Platform clusters. It is used to test cluster or node density. For testing the control plane, **kube-burner ocp** has three workload profiles: **cluster-density**, **cluster-density-v2**, and **cluster-density-ms**. Each workload profile creates a series of resources designed to load the control.

CHAPTER 2. RECOMMENDED ETCD PRACTICES

The following documentation provides information about recommended performance and scalability practices for etcd.

2.1. STORAGE PRACTICES FOR ETCD

Because etcd writes data to disk and persists proposals on disk, its performance depends on disk performance. Although etcd is not particularly I/O intensive, it requires a low latency block device for optimal performance and stability. Because the consensus protocol for etcd depends on persistently storing metadata to a log (WAL), etcd is sensitive to disk-write latency. Slow disks and disk activity from other processes can cause long fsync latencies.

Those latencies can cause etcd to miss heartbeats, not commit new proposals to the disk on time, and ultimately experience request timeouts and temporary leader loss. High write latencies also lead to an OpenShift API slowness, which affects cluster performance. Because of these reasons, avoid colocating other workloads on the control-plane nodes that are I/O sensitive or intensive and share the same underlying I/O infrastructure.

Run etcd on a block device that can write at least 50 IOPS of 8KB sequentially, including fdatasync, in under 10ms. For heavy loaded clusters, sequential 500 IOPS of 8000 bytes (2 ms) are recommended. To measure those numbers, you can use a benchmarking tool, such as the **fio** command.

To achieve such performance, run etcd on machines that are backed by SSD or NVMe disks with low latency and high throughput. Consider single-level cell (SLC) solid-state drives (SSDs), which provide 1 bit per memory cell, are durable and reliable, and are ideal for write-intensive workloads.

NOTE

The load on etcd arises from static factors, such as the number of nodes and pods, and dynamic factors, including changes in endpoints due to pod autoscaling, pod restarts, job executions, and other workload-related events. To accurately size your etcd setup, you must analyze the specific requirements of your workload. Consider the number of nodes, pods, and other relevant factors that impact the load on etcd.

The following hard drive practices provide optimal etcd performance:

- Use dedicated etcd drives. Avoid drives that communicate over the network, such as iSCSI. Do not place log files or other heavy workloads on etcd drives.
- Prefer drives with low latency to support fast read and write operations.
- Prefer high-bandwidth writes for faster compactions and defragmentation.
- Prefer high-bandwidth reads for faster recovery from failures.
- Use solid state drives as a minimum selection. Prefer NVMe drives for production environments.
- Use server-grade hardware for increased reliability.
- Avoid NAS or SAN setups and spinning drives. Ceph Rados Block Device (RBD) and other types of network-attached storage can result in unpredictable network latency. To provide fast storage to etcd nodes at scale, use PCI passthrough to pass NVM devices directly to the nodes.

- Always benchmark by using utilities such as **fio**. You can use such utilities to continuously monitor the cluster performance as it increases.
- Avoid using the Network File System (NFS) protocol or other network based file systems.

Some key metrics to monitor on a deployed OpenShift Container Platform cluster are p99 of etcd disk write ahead log duration and the number of etcd leader changes. Use Prometheus to track these metrics.

NOTE

The etcd member database sizes can vary in a cluster during normal operations. This difference does not affect cluster upgrades, even if the leader size is different from the other members.

2.2. CLUSTER LATENCY REQUIREMENTS FOR ETCD

Two important constraints should be addressed to provide a low-latency, high-availability network for etcd:

- network I/O latency
- disk I/O latency

etcd uses the Raft consensus algorithm, and every change should replicate to a majority of the cluster members before it commits. This process is highly sensitive to network and disk performance. The minimum time for an etcd request is the Round-Trip Time (RTT) between members, plus the time required for data to write to permanent storage.

To achieve high availability, etcd should detect and recover from a leader failure quickly. This depends on two key tuning parameters:

Heartbeat Interval

The frequency that the leader sends a heartbeat to followers. This value should be close to the average RTT between members.

Election Timeout

The time a follower waits without hearing a heartbeat before it attempts to become the new leader. This should be at least 10 times the RTT value to account for network variance.

In a healthy cluster, the round-trip time between members should be less than 50 ms to ensure stability and avoid frequent leader elections. This is why etcd clusters are often deployed within a single data center or availability zone to minimize physical distance and network latency.

To support a low-latency, high-availability network, especially during the leader election process, an arbiter site should be located where it provides an RTT latency of less than 10 ms. The arbiter component of a network maintains consistency and availability in a distributed system.

Additional resources

- Setting tuning parameters for etcd

2.3. VALIDATING THE HARDWARE FOR ETCD

To validate the hardware for etcd before or after you create the OpenShift Container Platform cluster, you can use fio.

Prerequisites

- Container runtimes such as Podman or Docker are installed on the machine that you are testing.
- Data is written to the **/var/lib/etcd** path.

Procedure

- Run fio and analyze the results:
  - If you use Podman, run this command:

```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

  - If you use Docker, run this command:

```
$ sudo docker run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

The output reports whether the disk is fast enough to host etcd by comparing the 99th percentile of the fsync metric captured from the run to see if it is less than 10 ms. A few of the most important etcd metrics that might affected by I/O performance are as follows:

- **etcd_disk_wal_fsync_duration_seconds_bucket** metric reports the etcd's WAL fsync duration
- **etcd_disk_backend_commit_duration_seconds_bucket** metric reports the etcd backend commit latency duration
- **etcd_server_leader_changes_seen_total** metric reports the leader changes

Because etcd replicates the requests among all the members, its performance strongly depends on network input/output (I/O) latency. High network latencies result in etcd heartbeats taking longer than the election timeout, which results in leader elections that are disruptive to the cluster. A key metric to monitor on a deployed OpenShift Container Platform cluster is the 99th percentile of etcd network peer latency on each etcd cluster member. Use Prometheus to track the metric.

The **histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[2m]))** metric reports the round trip time for etcd to finish replicating the client requests between the members. Ensure that it is less than 50 ms.

Additional resources

- How to use **fio** to check etcd disk performance in OpenShift Container Platform
- etcd performance troubleshooting guide for OpenShift Container Platform

CHAPTER 3. ENSURING RELIABLE ETCD PERFORMANCE AND SCALABILITY

To ensure optimal performance with etcd, it's important to understand the conditions that affect performance, including node scaling, leader election, log replication, tuning, latency, network jitter, peer round trip time, database size, and Kubernetes API transaction rates.

3.1. LEADER ELECTION AND LOG REPLICATION OF ETCD

etcd is a consistent, distributed key-value store that operates as a cluster of replicated nodes. Following the Raft algorithm, etcd operates by electing one node as the leader and the others as followers. The leader maintains the system's current state and ensures that the followers are up-to-date.

The leader node is responsible for log replication. It handles incoming write transactions from the client and writes a Raft log entry that it then broadcasts to the followers.

When an etcd client such as **kube-apiserver** connects to an etcd member that is requesting an action that requires a quorum, such as writing a value, if the etcd member is a follower, it returns a message indicating the transaction should be sent to the leader.

When the etcd client requests an action that requires a quorum from the leader, the leader keeps the client connection open while it writes the local Raft log, broadcasts the log to the followers, and waits for the majority of the followers to acknowledge to have committed the log without failures. Only then does the leader send the acknowledgment to the etcd client and close the session. If failure notifications are received from the followers and the majority fails to reach a consensus, the leader returns the error message to the client and closes the session.

Additional resources

- The etcd learner design
- Failure modes

3.2. NODE SCALING FOR ETCD

In general, clusters must have 3 control plane nodes. However, if your cluster is installed on a bare metal platform, it can have up to 5 control plane nodes. If an existing bare-metal cluster has fewer than 5 control plane nodes, you can scale the cluster up as a postinstallation task.

For example, to scale from 3 to 4 control plane nodes after installation, you can add a host and install it as a control plane node. Then, the etcd Operator scales accordingly to account for the additional control plane node.

Scaling a cluster to 4 or 5 control plane nodes is available only on bare metal platforms.

For more information about how to scale control plane nodes by using the Assisted Installer, see "Adding hosts with the API" and "Replacing a control plane node in a healthy cluster".

NOTE

While adding control plane nodes can increase reliability and availability, it can decrease throughput and increase latency, affecting performance.

The following table shows failure tolerance for clusters of different sizes:

Table 3.1. Failure tolerances by cluster size

Cluster size	Majority	Failure tolerance
1 node	1	0
3 nodes	2	1
4 nodes	3	1
5 nodes	3	2

For more information about recovering from quorum loss, see "Restoring to a previous cluster state".

Additional resources

- Adding hosts with the API
- Replacing a control plane node in a healthy cluster
- Expanding the cluster
- Restoring to a previous cluster state

3.3. EFFECTS OF DISK LATENCY ON ETCD

An etcd cluster is sensitive to disk latencies. To understand the disk latency that is experienced by etcd in your control plane environment, run the **fio** tests or suite.

Make sure that the final report classifies the disk as appropriate for etcd, as shown in the following example:

```
...
99th percentile of fsync is 5865472 ns
99th percentile of the fsync is within the recommended threshold: - 20 ms, the disk can be used to host etcd
```

When a high latency disk is used, a message states that the disk is not recommended for etcd, as shown in the following example:

```
...
99th percentile of fsync is 15865472 ns
99th percentile of the fsync is greater than the recommended value which is 20 ms, faster disks are recommended to host etcd for better performance
```

When you use cluster deployments that span multiple data centers that are using disks for etcd that do not meet the recommended latency, it increases the chances of service-affecting failures and dramatically reduces the network latency that the control plane can sustain.

3.4. MONITORING CONSENSUS LATENCY FOR ETCD

By using the **etcdctl** CLI, you can monitor the latency for reaching consensus as experienced by etcd. You must identify one of the etcd pods and then retrieve the endpoint health.

This procedure, which validates and monitors cluster health, can be run only on an active cluster.

Prerequisites

- During planning for cluster deployment, you completed the disk and network tests.

Procedure

1. Enter the following command:

```
# oc get pods -n openshift-etcd -l app=etcd
```

Example output

```
NAME     READY   STATUS    RESTARTS   AGE
etcd-m0  4/4     Running   4          8h
etcd-m1  4/4     Running   4          8h
etcd-m2  4/4     Running   4          8h
```

2. Enter the following command. To better understand the etcd latency for consensus, you can run this command on a precise watch cycle for a few minutes to observe that the numbers remain below the ~66 ms threshold. The closer the consensus time is to 100 ms, the more likely the cluster will experience service-affecting events and instability.

```
# oc exec -ti etcd-m0 -- etcdctl endpoint health -w table
```

Example output

```
+---------------------------+--------+-------------+-------+
|         ENDPOINT          | HEALTH |    TOOK     | ERROR |
+---------------------------+--------+-------------+-------+
| https://198.18.111.12:2379 |   true |  3.798349ms |       |
| https://198.18.111.14:2379 |   true |  7.389608ms |       |
| https://198.18.111.13:2379 |   true |  6.263117ms |       |
+---------------------------+--------+-------------+-------+
```

3. Enter the following command:

```
# oc exec -ti etcd-m0 -- watch -dp -c etcdctl endpoint health -w table
```

Example output

```
+---------------------------+--------+-------------+-------+
|         ENDPOINT          | HEALTH |    TOOK     | ERROR |
+---------------------------+--------+-------------+-------+
| https://198.18.111.12:2379 |   true |  9.533405ms |       |
```

```
| https://198.18.111.13:2379 |   true |  4.628054ms |       |
| https://198.18.111.14:2379 |   true |  5.803378ms |       |
+----------------------------+--------+-------------+-------+
```

3.5. MOVING ETCD TO A DIFFERENT DISK

You can move etcd from a shared disk to a separate disk to prevent or resolve performance issues.

The Machine Config Operator (MCO) is responsible for mounting a secondary disk for OpenShift Container Platform 4.20 container storage.

NOTE

This encoded script only supports device names for the following device types:

SCSI or SATA

/dev/sd*

Virtual device

/dev/vd*

NVMe

/dev/nvme*[0-9]*n*

Limitations

- When the new disk is attached to the cluster, the etcd database is part of the root mount. It is not part of the secondary disk or the intended disk when the primary node is recreated. As a result, the primary node will not create a separate **/var/lib/etcd** mount.

Prerequisites

- You have a backup of your cluster's etcd data.
- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster with **cluster-admin** privileges.
- Add additional disks before uploading the machine configuration.
- The **MachineConfigPool** must match **metadata.labels[machineconfiguration.openshift.io/role]**. This applies to a controller, worker, or a custom pool.

NOTE

This procedure does not move parts of the root file system, such as **/var/**, to another disk or partition on an installed node.

IMPORTANT

This procedure is not supported when using control plane machine sets.

Procedure

1. Attach the new disk to the cluster and verify that the disk is detected in the node by running the **lsblk** command in a debug shell:

```
$ oc debug node/<node_name>
```

```
# lsblk
```

Note the device name of the new disk reported by the **lsblk** command.

2. Create the following script and name it **etcd-find-secondary-device.sh**:

```
#!/bin/bash
set -uo pipefail

for device in <device_type_glob>; do 1
/usr/sbin/blkid "${device}" &> /dev/null
 if [ $? == 2  ]; then
    echo "secondary device found ${device}"
    echo "creating filesystem for etcd mount"
    mkfs.xfs -L var-lib-etcd -f "${device}" &> /dev/null
    udevadm settle
    touch /etc/var-lib-etcd-mount
    exit
 fi
done
echo "Couldn't find secondary block device!" >&2
exit 77
```

1 Replace **<device_type_glob>** with a shell glob for your block device type. For SCSI or SATA drives, use **/dev/sd***; for virtual drives, use **/dev/vd***; for NVMe drives, use **/dev/nvme*[0-9]*n***.

3. Create a base64-encoded string from the **etcd-find-secondary-device.sh** script and note its contents:

```
$ base64 -w0 etcd-find-secondary-device.sh
```

4. Create a **MachineConfig** YAML file named **etcd-mc.yml** with contents such as the following:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:
  config:
    ignition:
      version: 3.5.0
    storage:
      files:
```

```
      - path: /etc/find-secondary-device
        mode: 0755
        contents:
          source: data:text/plain;charset=utf-8;base64,
<encoded_etcd_find_secondary_device_script> (1)
    systemd:
      units:
      - name: find-secondary-device.service
        enabled: true
        contents: |
          [Unit]
          Description=Find secondary device
          DefaultDependencies=false
          After=systemd-udev-settle.service
          Before=local-fs-pre.target
          ConditionPathExists=!/etc/var-lib-etcd-mount

          [Service]
          RemainAfterExit=yes
          ExecStart=/etc/find-secondary-device

          RestartForceExitStatus=77

          [Install]
          WantedBy=multi-user.target
      - name: var-lib-etcd.mount
        enabled: true
        contents: |
          [Unit]
          Before=local-fs.target

          [Mount]
          What=/dev/disk/by-label/var-lib-etcd
          Where=/var/lib/etcd
          Type=xfs
          TimeoutSec=120s

          [Install]
          RequiredBy=local-fs.target
      - name: sync-var-lib-etcd-to-etcd.service
        enabled: true
        contents: |
          [Unit]
          Description=Sync etcd data if new mount is empty
          DefaultDependencies=no
          After=var-lib-etcd.mount var.mount
          Before=crio.service

          [Service]
          Type=oneshot
          RemainAfterExit=yes
          ExecCondition=/usr/bin/test ! -d /var/lib/etcd/member
          ExecStart=/usr/sbin/setsebool -P rsync_full_access 1
          ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
          ExecStart=/usr/sbin/semanage fcontext -a -t container_var_lib_t '/var/lib/etcd(/.*)?'
          ExecStart=/usr/sbin/setsebool -P rsync_full_access 0
```

```
      TimeoutSec=0

      [Install]
      WantedBy=multi-user.target graphical.target
  - name: restorecon-var-lib-etcd.service
    enabled: true
    contents: |
      [Unit]
      Description=Restore recursive SELinux security contexts
      DefaultDependencies=no
      After=var-lib-etcd.mount
      Before=crio.service

      [Service]
      Type=oneshot
      RemainAfterExit=yes
      ExecStart=/sbin/restorecon -R /var/lib/etcd/
      TimeoutSec=0

      [Install]
      WantedBy=multi-user.target graphical.target
```

1 Replace **<encoded_etcd_find_secondary_device_script>** with the encoded script contents that you noted.

5. Apply the created **MachineConfig** YAML file:

```
$ oc create -f etcd-mc.yml
```

Verification steps

- Run the **grep /var/lib/etcd /proc/mounts** command in a debug shell for the node to ensure that the disk is mounted:

```
$ oc debug node/<node_name>
```

```
# grep -w "/var/lib/etcd" /proc/mounts
```

Example output

```
/dev/sdb /var/lib/etcd xfs rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,noquota
0 0
```

Additional resources

- Red Hat Enterprise Linux CoreOS (RHCOS)

3.6. DEFRAGMENTING ETCD DATA

For large and dense clusters, etcd can suffer from poor performance if the keyspace grows too large and exceeds the space quota. Periodically maintain and defragment etcd to free up space in the data store. Monitor Prometheus for etcd metrics and defragment it when required; otherwise, etcd can raise

a cluster-wide alarm that puts the cluster into a maintenance mode that accepts only key reads and deletes.

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit
- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction
- **etcd_mvcc_db_total_size_in_bytes**, which shows the database size, including free space waiting for defragmentation

Defragment etcd data to reclaim disk space after events that cause disk fragmentation, such as etcd history compaction.

History compaction is performed automatically every five minutes and leaves gaps in the back-end database. This fragmented space is available for use by etcd, but is not available to the host file system. You must defragment etcd to make this space available to the host file system.

Defragmentation occurs automatically, but you can also trigger it manually.

NOTE

Automatic defragmentation is good for most cases, because the etcd operator uses cluster information to determine the most efficient operation for the user.

3.6.1. Automatic defragmentation

The etcd Operator automatically defragments disks. No manual intervention is needed.

Verify that the defragmentation process is successful by viewing one of these logs:

- etcd logs
- cluster-etcd-operator pod
- operator status error log

WARNING

Automatic defragmentation can cause leader election failure in various OpenShift core components, such as the Kubernetes controller manager, which triggers a restart of the failing component. The restart is harmless and either triggers failover to the next running instance or the component resumes work again after the restart.

Example log output for successful defragmentation

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

Example log output for unsuccessful defragmentation

```
failed defrag on member: <member_name>, memberID: <member_id>: <error_message>
```

3.6.2. Manual defragmentation

A Prometheus alert indicates when you need to use manual defragmentation. The alert is displayed in two cases:

- When etcd uses more than 50% of its available space for more than 10 minutes
- When etcd is actively using less than 50% of its total database size for more than 10 minutes

You can also determine whether defragmentation is needed by checking the etcd database size in MB that will be freed by defragmentation with the PromQL expression:
(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024

WARNING

Defragmenting etcd is a blocking action. The etcd member will not respond until defragmentation is complete. For this reason, wait at least one minute between defragmentation actions on each of the pods to allow the cluster to recover.

Follow this procedure to defragment etcd data on each etcd member.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Determine which etcd member is the leader, because the leader should be defragmented last.
   a. Get the list of etcd pods:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

   Example output

```
etcd-ip-10-0-159-225.example.redhat.com        3/3     Running     0          175m     10.0.159.225   ip-10-0-159-225.example.redhat.com   <none>           <none>
etcd-ip-10-0-191-37.example.redhat.com         3/3     Running     0          173m     10.0.191.37    ip-10-0-191-37.example.redhat.com    <none>           <none>
etcd-ip-10-0-199-170.example.redhat.com        3/3     Running     0          176m     10.0.199.170   ip-10-0-199-170.example.redhat.com   <none>           <none>
```

   b. Choose a pod and run the following command to determine which etcd member is the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint
status --cluster -w table
```

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|        ENDPOINT           |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |              91624 |        |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

Based on the **IS LEADER** column of this output, the **https://10.0.199.170:2379** endpoint is the leader. Matching this endpoint with the output of the previous step, the pod name of the leader is **etcd-ip-10-0-199-170.example.redhat.com**.

2. Defragment an etcd member.

   a. Connect to the running etcd container, passing in the name of a pod that is *not* the leader:

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
   ```

   b. Unset the **ETCDCTL_ENDPOINTS** environment variable:

   ```
   sh-4.4# unset ETCDCTL_ENDPOINTS
   ```

   c. Defragment the etcd member:

   ```
   sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
   ```

   Example output

   ```
   Finished defragmenting etcd member[https://localhost:2379]
   ```

   If a timeout error occurs, increase the value for **--command-timeout** until the command succeeds.

   d. Verify that the database size was reduced:

   ```
   sh-4.4# etcdctl endpoint status -w table --cluster
   ```

   Example output

```
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|        ENDPOINT          |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |             91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |   41 MB |     false |      false |
7 |      91624 |             91624 |        | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |             91624 |        |
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

This example shows that the database size for this etcd member is now 41 MB as opposed to the starting size of 104 MB.

e. Repeat these steps to connect to each of the other etcd members and defragment them. Always defragment the leader last.
Wait at least one minute between defragmentation actions to allow the etcd pod to recover. Until the etcd pod recovers, the etcd member will not respond.

3. If any **NOSPACE** alarms were triggered due to the space quota being exceeded, clear them.

a. Check if there are any **NOSPACE** alarms:

```
sh-4.4# etcdctl alarm list
```

Example output

```
memberID:12345678912345678912 alarm:NOSPACE
```

b. Clear the alarms:

```
sh-4.4# etcdctl alarm disarm
```

3.7. SETTING TUNING PARAMETERS FOR ETCD

You can set the control plane hardware speed to **"Standard"**, **"Slower"**, or the default, which is **""**.

The default setting allows the system to decide which speed to use. This value enables upgrades from versions where this feature does not exist, as the system can select values from previous versions.

By selecting one of the other values, you are overriding the default. If you see many leader elections due to timeouts or missed heartbeats and your system is set to **""** or **"Standard"**, set the hardware speed to **"Slower"** to make the system more tolerant to the increased latency.

3.7.1. Changing hardware speed tolerance

To change the hardware speed tolerance for etcd, complete the following steps.

Procedure

1. Check to see what the current value is by entering the following command:

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

Example output

```
Control Plane Hardware Speed:  <VALUE>
```

NOTE

If the output is empty, the field has not been set and should be considered as the default ("").

2. Change the value by entering the following command. Replace **<value>** with one of the valid values: **""**, **"Standard"**, or **"Slower"**:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"controlPlaneHardwareSpeed": "<value>"}}'
```

The following table indicates the heartbeat interval and leader election timeout for each profile. These values are subject to change.

Profile	ETCD_HEARTBEAT_INTERVAL	ETCD_LEADER_ELECTION_TIMEOUT
""	Varies depending on platform	Varies depending on platform
Standard	100	1000
Slower	500	2500

3. Review the output:

Example output

```
etcd.operator.openshift.io/cluster patched
```

If you enter any value besides the valid values, error output is displayed. For example, if you entered **"Faster"** as the value, the output is as follows:

Example output

```
The Etcd "cluster" is invalid: spec.controlPlaneHardwareSpeed: Unsupported value: "Faster":
supported values: "", "Standard", "Slower"
```

4. Verify that the value was changed by entering the following command:

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

Example output

```
Control Plane Hardware Speed:  ""
```

5. Wait for etcd pods to roll out:

```
$ oc get pods -n openshift-etcd -w
```

The following output shows the expected entries for master-0. Before you continue, wait until all masters show a status of **4/4 Running**.

Example output

```
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     Pending             0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     Pending             0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     ContainerCreating   0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     ContainerCreating   0          1s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        1/1     Running             0          2s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     Completed           0          34s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     Completed           0          36s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0        0/1     Completed           0          36s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Running             0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               0/4     Pending             0          0s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               0/4     Init:1/3            0          1s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               0/4     Init:2/3            0          2s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               0/4     PodInitializing     0          3s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               3/4     Running             0          4s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0         1/1     Running             0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               3/4     Running             0          20s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0               4/4     Running             0          20s
```

6. Enter the following command to review to the values:

```
$ oc describe -n openshift-etcd pod/<ETCD_PODNAME> | grep -e HEARTBEAT_INTERVAL
-e ELECTION_TIMEOUT
```

NOTE

These values might not have changed from the default.

Additional resources

- Understanding feature gates

3.8. OPENSHIFT CONTAINER PLATFORM TIMER TUNABLES FOR ETCD

OpenShift Container Platform maintains etcd timers that are optimized for each platform. OpenShift Container Platform has prescribed validated values that are optimized for each platform provider. The default etcd timers with **platform=none** or **platform=metal** are as follows:

```
- name: ETCD_ELECTION_TIMEOUT
```

```
  value: "1000"
  ...
- name: ETCD_HEARTBEAT_INTERVAL
  value: "100"
```

From an etcd perspective, the two key values are election timeout and heartbeat interval:

Heartbeat interval

The frequency with which the leader notifies followers that it is still the leader.

Election timeout

This timeout is how long a follower node will go without hearing a heartbeat before it attempts to become leader itself.

These values do not provide the whole story for the control plane or even etcd. An etcd cluster is sensitive to disk latencies. Because etcd must persist proposals to its log, disk activity from other processes might cause long fsync latencies. The consequence is that etcd might miss heartbeats, causing request timeouts and temporary leader loss. During a leader loss and reelection, the Kubernetes API cannot process any request that causes a service-affecting event and instability of the cluster.

3.9. DETERMINING THE SIZE OF THE ETCD DATABASE AND UNDERSTANDING ITS EFFECTS

The size of the etcd database has a direct impact on the time to complete the etcd defragmentation process. OpenShift Container Platform automatically runs the etcd defragmentation on one etcd member at a time when it detects at least 45% fragmentation. During the defragmentation process, the etcd member cannot process any requests. On small etcd databases, the defragmentation process happens in less than a second. With larger etcd databases, the disk latency directly impacts the fragmentation time, causing additional latency, as operations are blocked while defragmentation happens.

The size of the etcd database is a factor to consider when network partitions isolate a control plane node for a period and the control plane needs to resync after communication is re-established.

Minimal options exist for controlling the size of the etcd database, as it depends on the operators and applications in the system. When you consider the latency range under which the system will operate, account for the effects of synchronization or defragmentation per size of the etcd database.

The magnitude of the effects is specific to the deployment. The time to complete a defragmentation will cause degradation in the transaction rate, as the etcd member cannot accept updates during the defragmentation process. Similarly, the time for the etcd re-synchronization for large databases with high change rate affects the transaction rate and transaction latency on the system.

Consider the following two examples for the type of impacts to plan for.

Example of the effect of etcd defragementation based on database size

Writing an etcd database of 1 GB to a slow 7200 RPMs disk at 80 Mbit/s takes about 1 minute and 40 seconds. In such a scenario, the defragmentation process takes at least this long, if not longer, to complete the defragmentation.

Example of the effect of database size on etcd synchronization

If there is a change of 10% of the etcd database during the disconnection of one of the control plane nodes, the resync needs to transfer at least 100 MB. Transferring 100 MB over a 1 Gbps link takes 800 ms. On clusters with regular transactions with the Kubernetes API, the larger the etcd database size, the more network instabilities will cause control plane instabilities.

You can determine the size of an etcd database by using the OpenShift Container Platform console or by running commands in the **etcdctl** tool.

Procedure

- To find the database size in the OpenShift Container Platform console, go to the **etcd** dashboard to view a plot that reports the size of the etcd database.
- To find the database size by using the etcdctl tool, you can enter two commands:
  a. Enter the following command to list the pods:

```
# oc get pods -n openshift-etcd -l app=etcd
```

Example output

```
NAME     READY   STATUS    RESTARTS   AGE
etcd-m0  4/4     Running   4          22h
etcd-m1  4/4     Running   4          22h
etcd-m2  4/4     Running   4          22h
```

  b. Enter the following command and view the database size in the output:

```
# oc exec -t etcd-m0 -- etcdctl endpoint status -w simple | cut -d, -f 1,3,4
```

Example output

```
https://198.18.111.12:2379, 3.5.6, 1.1 GB
https://198.18.111.13:2379, 3.5.6, 1.1 GB
https://198.18.111.14:2379, 3.5.6, 1.1 GB
```

3.10. INCREASING THE DATABASE SIZE FOR ETCD

You can set the disk quota in gibibytes (GiB) for each etcd instance. If you set a disk quota for your etcd instance, you can specify integer values from 8 to 32. The default value is 8. You can specify only increasing values.

You might want to increase the disk quota if you encounter a **low space** alert. This alert indicates that the cluster is too large to fit in etcd despite automatic compaction and defragmentation. If you see this alert, you need to increase the disk quota immediately because after etcd runs out of space, writes fail.

Another scenario where you might want to increase the disk quota is if you encounter an **excessive database growth** alert. This alert is a warning that the database might grow too large in the next four hours. In this scenario, consider increasing the disk quota so that you do not eventually encounter a **low space** alert and possible write fails.

If you increase the disk quota, the disk space that you specify is not immediately reserved. Instead, etcd can grow to that size if needed. Ensure that etcd is running on a dedicated disk that is larger than the value that you specify for the disk quota.

For large etcd databases, the control plane nodes must have additional memory and storage. Because you must account for the API server cache, the minimum memory required is at least three times the configured size of the etcd database.

IMPORTANT

Increasing the database size for etcd 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 .

3.10.1. Changing the etcd database size

To change the database size for etcd, complete the following steps.

Procedure

1. Check the current value of the disk quota for each etcd instance by entering the following command:

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

Example output

```
Backend Quota Gi B: <value>
```

2. Change the value of the disk quota by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": <value>}}'
```

Example output

```
etcd.operator.openshift.io/cluster patched
```

Verification

1. Verify that the new value for the disk quota is set by entering the following command:

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

The etcd Operator automatically rolls out the etcd instances with the new values.

2. Verify that the etcd pods are up and running by entering the following command:

```
$ oc get pods -n openshift-etcd
```

The following output shows the expected entries.

Example output

```
NAME                                READY   STATUS     RESTARTS   AGE
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-0          4/4     Running    0          39m
```

```
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-1                 4/4     Running     0          37m
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-2                 4/4     Running     0          41m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-0           1/1     Running     0          51m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-1           1/1     Running     0          49m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-2           1/1     Running     0          54m
installer-5-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          51m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-0          0/1     Completed   0          46m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          44m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-2          0/1     Completed   0          49m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-0          0/1     Completed   0          40m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          38m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-2          0/1     Completed   0          42m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-0    0/1     Completed   0          43m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-1    0/1     Completed   0          43m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-2    0/1     Completed   0          43m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-0    0/1     Completed   0          42m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-1    0/1     Completed   0          42m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-2    0/1     Completed   0          42m
```

3. Verify that the disk quota value is updated for the etcd pod by entering the following command:

```
$ oc describe -n openshift-etcd pod/<etcd_podname> | grep
"ETCD_QUOTA_BACKEND_BYTES"
```

The value might not have changed from the default value of **8**.

Example output

```
ETCD_QUOTA_BACKEND_BYTES:                8589934592
```

NOTE

While the value that you set is an integer in GiB, the value shown in the output is converted to bytes.

3.10.2. Troubleshooting

If you encounter issues when you try to increase the database size for etcd, the following troubleshooting steps might help.

3.10.2.1. Value is too small

If the value that you specify is less than **8**, you see the following error message:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 5}}'
```

Example error message

```
The Etcd "cluster" is invalid:
* spec.backendQuotaGiB: Invalid value: 5: spec.backendQuotaGiB in body should be greater than or
equal to 8
* spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased
```

To resolve this issue, specify an integer between **8** and **32**.

3.10.2.2. Value is too large

If the value that you specify is greater than **32**, you see the following error message:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 64}}'
```

Example error message

```
The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: 64: spec.backendQuotaGiB in body should be less than or equal to 32
```

To resolve this issue, specify an integer between **8** and **32**.

3.10.2.3. Value is decreasing

If the value is set to a valid value between **8** and **32**, you cannot decrease the value. Otherwise, you see an error message.

1. Check to see the current value by entering the following command:

   ```
   $ oc describe etcd/cluster | grep "Backend Quota"
   ```

   Example output

   ```
   Backend Quota Gi B: 10
   ```

2. Decrease the disk quota value by entering the following command:

   ```
   $ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 8}}'
   ```

   Example error message

   ```
   The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased
   ```

3. To resolve this issue, specify an integer greater than **10**.

3.11. MEASURING NETWORK JITTER BETWEEN CONTROL PLANE NODES

The value of the heartbeat interval should be around the maximum of the average round-trip time (RTT) between members, normally around 1.5 times the round-trip time. With the OpenShift Container Platform default heartbeat interval of 100 ms, the recommended RTT between control plane nodes is less than approximately 33 ms with a maximum of less than 66 ms (66 ms multiplied by 1.5 equals 99 ms). For more information, see "Setting tuning parameters for etcd". Any network latency that is higher might cause service-affecting events and cluster instability.

The network latency is influenced by many factors, including but not limited to the following factors:

- The technology of the transport networks, such as copper, fiber, wireless, or satellite
- The number and quality of the network devices in the transport network

A good evaluation reference is the comparison of the network latency in the organization with the commercial latencies that are published by telecommunications providers, such as monthly IP latency statistics.

Consider network latency with network jitter for more accurate calculations. *Network jitter* is the variance in network latency or, more specifically, the variation in the delay of received packets. On ideal network conditions, the jitter is as close to zero as possible. Network jitter affects the network latency calculations for etcd because the actual network latency over time will be the RTT plus or minus jitter. For example, a network with a maximum latency of 80 ms and jitter of 30 ms will experience latencies of 110 ms, which means etcd is missing heartbeats, causing request timeouts and temporary leader loss. During a leader loss and reelection, the Kubernetes API cannot process any request that causes a service-affecting event and instability of the cluster.

It's important to measure the network jitter among all control plane nodes. To do so, you can use the **iPerf3** tool in UDP mode.

Prerequisite

- You built your own iPerf image. For more information, see the following Red Hat Knowledgebase articles
  - Testing Network Bandwidth in OpenShift using iPerf Container
  - How to run iPerf network performance test in OpenShift 4

Procedure

1. Connect to one of the control plane nodes and run the iPerf container as iPerf server in host network mode. When you are running in server mode, the tool accepts TCP and UDP tests. Enter the following command, being careful to replace **<iperf_image>** with your iPerf image:

   ```
   # podman run -ti --rm --net host <iperf_image> iperf3 -s
   ```

2. Connect to another control plane node and run the iPerf in UDP client mode by entering the following command:

   ```
   # podman run -ti --rm --net host <iperf_image> iperf3 -u -c <node_iperf_server> -t 300
   ```

   The default test runs for 10 seconds, and at the end, the client output shows the average jitter from the client perspective.

3. Run the debug node mode by entering the following command:

   ```
   # oc debug node/m1
   ```

   #### Example output

   ```
   Starting pod/m1-debug ...
   To use host binaries, run `chroot /host`
   Pod IP: 198.18.111.13
   If you don't see a command prompt, try pressing enter.
   ```

4. Enter the following commands:

```
sh-4.4# chroot /host
```

```
sh-4.4# podman run -ti --rm --net host <iperf_image> iperf3 -u -c m0
```

Example output

```
Connecting to host m0, port 5201
[  5] local 198.18.111.13 port 60878 connected to 198.18.111.12 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   3.00-4.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   4.00-5.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   5.00-6.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   6.00-7.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  5]   0.00-10.04  sec  1.25 MBytes  1.05 Mbits/sec  1.074 ms  0/906 (0%)  receiver

iperf Done.
```

5. On the iPerf server, the output shows the jitter on every second interval. The average is shown at the end. For the purpose of this test, you want to identify the maximum jitter that is experienced during the test, ignoring the output of the first second as it might contain an invalid measurement. Enter the following command:

```
# oc debug node/m0
```

Example output

```
Starting pod/m0-debug ...
To use host binaries, run `chroot /host`
Pod IP: 198.18.111.12
If you don't see a command prompt, try pressing enter.
```

6. Enter the following commands:

```
sh-4.4# chroot /host
```

```
sh-4.4# podman run -ti --rm --net host <iperf_image> iperf3 -s
```

Example output

```
-----------------------------------------------------------
```

```
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 198.18.111.13, port 44136
[  5] local 198.18.111.12 port 5201 connected to 198.18.111.13 port 60878
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec   124 KBytes  1.02 Mbits/sec  4.763 ms  0/88 (0%)
[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  4.735 ms  0/90 (0%)
[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  0.568 ms  0/91 (0%)
[  5]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  2.443 ms  0/90 (0%)
[  5]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  1.372 ms  0/91 (0%)
[  5]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec  2.769 ms  0/90 (0%)
[  5]   6.00-7.00   sec   129 KBytes  1.05 Mbits/sec  2.393 ms  0/91 (0%)
[  5]   7.00-8.00   sec   127 KBytes  1.04 Mbits/sec  0.883 ms  0/90 (0%)
[  5]   8.00-9.00   sec   129 KBytes  1.05 Mbits/sec  0.594 ms  0/91 (0%)
[  5]   9.00-10.00  sec   127 KBytes  1.04 Mbits/sec  0.953 ms  0/90 (0%)
[  5]  10.00-10.04  sec  5.66 KBytes  1.30 Mbits/sec  1.074 ms  0/4 (0%)
- - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.04  sec  1.25 MBytes  1.05 Mbits/sec  1.074 ms  0/906 (0%)  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

7. Add the calculated jitter as a penalty to the network latency. For example, if the network latency is 80 ms and the jitter is 30 ms, consider an effective network latency of 110 ms for the purposes of the control plane. In this example, that value goes above the 100 ms threshold, and the system will miss heartbeats.

8. When you calculate the network latency for etcd, use the effective network latency, which is the sum of the following equation:
   RTT + jitter

   You might be able to use the average jitter value to calculate the penalty, but the cluster can sporadically miss heartbeats if the etcd heartbeat timer is lower than the sum of the following equation:

   RTT + max(jitter)

   Instead, consider using the 99th percentile or max jitter value for a more resilient deployment:

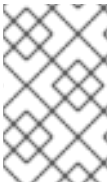
   Effective Network Latency = RTT + max(jitter)

3.12. HOW ETCD PEER ROUND TRIP TIME AFFECTS PERFORMANCE

The etcd peer round trip time is an end-to-end test metric on how quickly something can be replicated among members. It shows the latency of etcd to finish replicating a client request among all the etcd members. The etcd peer round trip time is not the same thing as the network round trip time.

You can monitor various etcd metrics on dashboards in the OpenShift Container Platform console. In the console, click **Observe → Dashboards** and from the dropdown list, select **etcd**.

Near the end of the **etcd** dashboard, you can find a plot that summarizes the etcd peer round trip time.



NOTE

These etcd metrics are collected by the OpenShift metrics system in Prometheus. You can access them from the CLI by following the Red Hat Knowledgebase solution, How to query from the command line Prometheus statistics.

```
# Get token to connect to Prometheus
SECRET=$(oc get secret -n openshift-user-workload-monitoring | grep  prometheus-user-workload-token | head -n 1 | awk '{print $1 }')
export TOKEN=$(oc get secret $SECRET -n openshift-user-workload-monitoring -o json | jq -r '.data.token' | base64 -d)
export THANOS_QUERIER_HOST=$(oc get route thanos-querier -n openshift-monitoring -o json | jq -r '.spec.host')
```

Queries must be URL-encoded. The following example shows how to retrieve the metrics that are reporting the round trip time (in seconds) for etcd to finish replicating the client requests among the members:

```
# prometheus query
query="histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[5m]))"

# urlencoded query
encoded_query=$(printf "%s" $query |jq -sRr @uri)

# querying the OpenShift metrics service
curl -s -X GET -k -H "Authorization: Bearer $TOKEN"
"https://$THANOS_QUERIER_HOST/api/v1/query?query=$encoded_query" | jq '.data.result[] |
.metric.pod,.value[1]'

"etcd-m2"
"0.09318400000000004"   # example ~93ms
"etcd-m0"
"0.050688"              # example ~51ms
"etcd-m1"
"0.050688"              # example ~51ms
```

The following metrics are also relevant to understanding etcd performance:

etcd_disk_wal_fsync_duration_seconds_bucket

Reports the etcd WAL fsync duration.

etcd_disk_backend_commit_duration_seconds_bucket

Reports the etcd backend commit latency duration.

etcd_server_leader_changes_seen_total

Reports the leader changes.

3.13. DETERMINING KUBERNETES API TRANSACTION RATE FOR YOUR ENVIRONMENT

When you are using stretched control planes, the Kubernetes API transaction rate depends on the characteristics of the particular deployment. Specifically, it depends on the following combined factors:

- The etcd disk latency

- The etcd round trip time
- The size of objects that are being written to the API

As a result, when you use stretched control planes, cluster administrators must test the environment to determine the sustained transaction rate that is possible for the environment. The **kube-burner** tool is useful for that purpose. The binary includes a wrapper for testing OpenShift clusters: **kube-burner-ocp**. You can use **kube-burner-ocp** to test cluster or node density. To test the control plane, **kube-burner-ocp** has three workload profiles: cluster-density, cluster-density-v2, and cluster-density-ms. Each workload profile creates a series of resources that are designed to load the control plane. For more information about each profile, see the **kube-burner-ocp** workload documentation.

Procedure

1. Enter a command to create and delete resources. The following example shows a command that creates and deletes resources within 20 minutes:

```
# kube-burner ocp cluster-density-ms --churn-duration 20m --churn-delay 0s --iterations 10 --timeout 30m
```

2. The OpenShift Container Platform console provides a dashboard with all the relevant API performance information. To access API performance information, click **Observe → Dashboards**, and from the **Dashboards** menu, click **API Performance**.

3. During the run, observe the API performance dashboard in the OpenShift Container Platform console by clicking **Observe → Dashboards**, and from the **Dashboards** menu, click **API Performance**.
   On the dashboard, notice how the control plane responds during load and the 99th percentile transaction rate it can achieve for the execution of various verbs and request rates by read and write. Use this information and the knowledge of your organization's workload to determine the load that the organization can put in the clusters for the specific stretched control plane deployment.

Additional resources

- kube-burner-ocp documentation

CHAPTER 4. BACKING UP AND RESTORING ETCD DATA

4.1. BACKING UP AND RESTORING ETCD DATA

As the key-value store for OpenShift Container Platform, etcd persists the state of all resource objects.

Back up the etcd data for your cluster regularly and store it in a secure location, ideally outside the OpenShift Container Platform environment. Do not take an etcd backup before the first certificate rotation completes, which occurs 24 hours after installation, otherwise the backup will contain expired certificates. It is also recommended to take etcd backups during non-peak usage hours because the etcd snapshot has a high I/O cost.

Be sure to take an etcd backup before you update your cluster. Taking a backup before you update is important because when you restore your cluster, you must use an etcd backup that was taken from the same z-stream release. For example, an OpenShift Container Platform 4.17.5 cluster must use an etcd backup that was taken from 4.17.5.

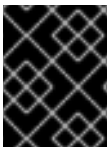
IMPORTANT

Back up your cluster's etcd data by performing a single invocation of the backup script on a control plane host. Do not take a backup for each control plane host.

After you have an etcd backup, you can restore to a previous cluster state .

4.1.1. Backing up etcd data

Follow these steps to back up etcd data by creating an etcd snapshot and backing up the resources for the static pods. This backup can be saved and used at a later time if you need to restore etcd.



IMPORTANT

Only save a backup from a single control plane host. Do not take a backup from each control plane host in the cluster.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have checked whether the cluster-wide proxy is enabled.

  TIP

  You can check whether the proxy is enabled by reviewing the output of **oc get proxy cluster -o yaml**. The proxy is enabled if the **httpProxy**, **httpsProxy**, and **noProxy** fields have values set.

Procedure

1. Start a debug session as root for a control plane node:

   ```
   $ oc debug --as-root node/<node_name>
   ```

2. Change your root directory to /**host** in the debug shell:

```
sh-4.4# chroot /host
```

3. If the cluster-wide proxy is enabled, export the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables by running the following commands:

```
$ export HTTP_PROXY=http://<your_proxy.example.com>:8080
```

```
$ export HTTPS_PROXY=https://<your_proxy.example.com>:8080
```

```
$ export NO_PROXY=<example.com>
```

4. Run the **cluster-backup.sh** script in the debug shell and pass in the location to save the backup to.

TIP

The **cluster-backup.sh** script is maintained as a component of the etcd Cluster Operator and is a wrapper around the **etcdctl snapshot save** command.

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

Example script output

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

In this example, two files are created in the **/home/core/assets/backup/** directory on the control plane host:

- **snapshot_<datetimestamp>.db**: This file is the etcd snapshot. The **cluster-backup.sh** script confirms its validity.
- **static_kuberesources_<datetimestamp>.tar.gz**: This file contains the resources for the static pods. If etcd encryption is enabled, it also contains the encryption keys for the etcd snapshot.

NOTE

If etcd encryption is enabled, it is recommended to store this second file separately from the etcd snapshot for security reasons. However, this file is required to restore from the etcd snapshot.

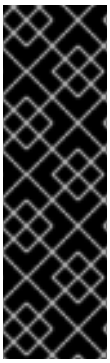
Keep in mind that etcd encryption only encrypts values, not keys. This means that resource types, namespaces, and object names are unencrypted.

Additional resources

- Recovering an unhealthy etcd cluster

4.1.2. Creating automated etcd backups

The automated backup feature for etcd supports both recurring and single backups. Recurring backups create a cron job that starts a single backup each time the job triggers.



IMPORTANT

Automating etcd backups 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 .

Follow these steps to enable automated backups for etcd.

WARNING

Enabling the **TechPreviewNoUpgrade** feature set on your cluster prevents minor version updates. The **TechPreviewNoUpgrade** feature set cannot be disabled. Do not enable this feature set on production clusters.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a **FeatureGate** custom resource (CR) file named **enable-tech-preview-no-upgrade.yaml** with the following contents:

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
spec:
  featureSet: TechPreviewNoUpgrade
```

2. Apply the CR and enable automated backups:

```
$ oc apply -f enable-tech-preview-no-upgrade.yaml
```

3. It takes time to enable the related APIs. Verify the creation of the custom resource definition (CRD) by running the following command:

```
$ oc get crd | grep backup
```

Example output

```
backups.config.openshift.io 2023-10-25T13:32:43Z
etcdbackups.operator.openshift.io 2023-10-25T13:32:04Z
```

4.1.2.1. Creating a single automated etcd backup

Follow these steps to create a single etcd backup by creating and applying a custom resource (CR).

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift CLI (**oc**).

Procedure

- If dynamically-provisioned storage is available, complete the following steps to create a single automated etcd backup:
  a. Create a persistent volume claim (PVC) named **etcd-backup-pvc.yaml** with contents such as the following example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```
      storage: 200Gi 1
    volumeMode: Filesystem
```

1 1 The amount of storage available to the PVC. Adjust this value for your requirements.

b. Apply the PVC by running the following command:

```
$ oc apply -f etcd-backup-pvc.yaml
```

c. Verify the creation of the PVC by running the following command:

```
$ oc get pvc
```

Example output

```
NAME              STATUS    VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
etcd-backup-pvc   Bound                                      51s
```

NOTE

Dynamic PVCs stay in the **Pending** state until they are mounted.

d. Create a CR file named **etcd-single-backup.yaml** with contents such as the following example:

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc 1
```

1 The name of the PVC to save the backup to. Adjust this value according to your environment.

e. Apply the CR to start a single backup:

```
$ oc apply -f etcd-single-backup.yaml
```

- If dynamically-provisioned storage is not available, complete the following steps to create a single automated etcd backup:

a. Create a **StorageClass** CR file named **etcd-backup-local-storage.yaml** with the following contents:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: etcd-backup-local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

b. Apply the **StorageClass** CR by running the following command:

```
$ oc apply -f etcd-backup-local-storage.yaml
```

c. Create a PV named **etcd-backup-pv-fs.yaml** with contents such as the following example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-backup-pv-fs
spec:
  capacity:
    storage: 100Gi (1)
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: etcd-backup-local-storage
  local:
    path: /mnt
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - <example_master_node> (2)
```

(1) The amount of storage available to the PV. Adjust this value for your requirements.

(2) Replace this value with the node to attach this PV to.

d. Verify the creation of the PV by running the following command:

```
$ oc get pv
```

Example output

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS              REASON   AGE
etcd-backup-pv-fs      100Gi      RWO            Retain           Available           etcd-backup-
local-storage           10s
```

e. Create a PVC named **etcd-backup-pvc.yaml** with contents such as the following example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
```

```
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi 1
```

1 The amount of storage available to the PVC. Adjust this value for your requirements.

f. Apply the PVC by running the following command:

```
$ oc apply -f etcd-backup-pvc.yaml
```

g. Create a CR file named **etcd-single-backup.yaml** with contents such as the following example:

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc 1
```

1 The name of the persistent volume claim (PVC) to save the backup to. Adjust this value according to your environment.

h. Apply the CR to start a single backup:

```
$ oc apply -f etcd-single-backup.yaml
```

4.1.2.2. Creating recurring automated etcd backups

Follow these steps to create automated recurring backups of etcd.

Use dynamically-provisioned storage to keep the created etcd backup data in a safe, external location if possible. If dynamically-provisioned storage is not available, consider storing the backup data on an NFS share to make backup recovery more accessible.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift CLI (**oc**).

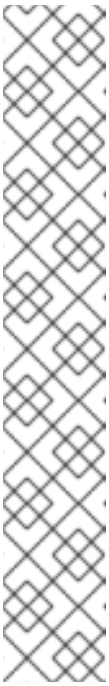
Procedure

1. If dynamically-provisioned storage is available, complete the following steps to create automated recurring backups:

a. Create a persistent volume claim (PVC) named **etcd-backup-pvc.yaml** with contents such as the following example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 200Gi (1)
  volumeMode: Filesystem
  storageClassName: etcd-backup-local-storage
```

(1) The amount of storage available to the PVC. Adjust this value for your requirements.



NOTE

Each of the following providers require changes to the **accessModes** and **storageClassName** keys:

Provider	**accessModes** value	**storageClassName** value
AWS with the **versioned-installer-efc_operator-ci** profile	**- ReadWriteMany**	**efs-sc**
Google Cloud	**- ReadWriteMany**	**filestore-csi**
Microsoft Azure	**- ReadWriteMany**	**azurefile-csi**

b. Apply the PVC by running the following command:

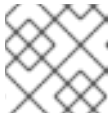
```
$ oc apply -f etcd-backup-pvc.yaml
```

c. Verify the creation of the PVC by running the following command:

```
$ oc get pvc
```

Example output

```
NAME              STATUS    VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
etcd-backup-pvc   Bound                                      51s
```



NOTE

Dynamic PVCs stay in the **Pending** state until they are mounted.

2. If dynamically-provisioned storage is unavailable, create a local storage PVC by completing the following steps:

WARNING

If you delete or otherwise lose access to the node that contains the stored backup data, you can lose data.

a. Create a **StorageClass** CR file named **etcd-backup-local-storage.yaml** with the following contents:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: etcd-backup-local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

b. Apply the **StorageClass** CR by running the following command:

```
$ oc apply -f etcd-backup-local-storage.yaml
```

c. Create a PV named **etcd-backup-pv-fs.yaml** from the applied **StorageClass** with contents such as the following example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-backup-pv-fs
spec:
  capacity:
    storage: 100Gi (1)
  volumeMode: Filesystem
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: etcd-backup-local-storage
  local:
    path: /mnt/
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
```

```
      operator: In
      values:
      - <example_master_node> 2
```

1 The amount of storage available to the PV. Adjust this value for your requirements.

2 Replace this value with the master node to attach this PV to.

TIP

Run the following command to list the available nodes:

```
$ oc get nodes
```

d. Verify the creation of the PV by running the following command:

```
$ oc get pv
```

Example output

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS            REASON   AGE
etcd-backup-pv-fs   100Gi      RWX            Delete           Available           etcd-backup-
local-storage           10s
```

e. Create a PVC named **etcd-backup-pvc.yaml** with contents such as the following example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi 1
  storageClassName: etcd-backup-local-storage
```

1 The amount of storage available to the PVC. Adjust this value for your requirements.

f. Apply the PVC by running the following command:

```
$ oc apply -f etcd-backup-pvc.yaml
```

3. Create a custom resource definition (CRD) file named **etcd-recurring-backups.yaml**. The contents of the created CRD define the schedule and retention type of automated backups.

- For the default retention type of **RetentionNumber** with 15 retained backups, use contents such as the following example:

```
apiVersion: config.openshift.io/v1alpha1
kind: Backup
metadata:
  name: etcd-recurring-backup
spec:
  etcd:
    schedule: "20 4 * * *" 1
    timeZone: "UTC"
    pvcName: etcd-backup-pvc
```

1 The **CronTab** schedule for recurring backups. Adjust this value for your needs.

- To use retention based on the maximum number of backups, add the following key-value pairs to the **etcd** key:

```
spec:
  etcd:
    retentionPolicy:
      retentionType: RetentionNumber 1
      retentionNumber:
        maxNumberOfBackups: 5 2
```

1 The retention type. Defaults to **RetentionNumber** if unspecified.

2 The maximum number of backups to retain. Adjust this value for your needs. Defaults to 15 backups if unspecified.

WARNING

A known issue causes the number of retained backups to be one greater than the configured value.

- For retention based on the file size of backups, use the following:

```
spec:
  etcd:
    retentionPolicy:
      retentionType: RetentionSize
      retentionSize:
        maxSizeOfBackupsGb: 20 1
```

1 The maximum file size of the retained backups in gigabytes. Adjust this value for your needs. Defaults to 10 GB if unspecified.

WARNING

A known issue causes the maximum size of retained backups to be up to 10 GB greater than the configured value.

4. Create the cron job defined by the CRD by running the following command:

```
$ oc create -f etcd-recurring-backup.yaml
```

5. To find the created cron job, run the following command:

```
$ oc get cronjob -n openshift-etcd
```

4.2. REPLACING AN UNHEALTHY ETCD MEMBER

The process to replace a single unhealthy etcd member depends on whether the etcd member is unhealthy because the machine is not running or the node is not ready, or because the etcd pod is crashlooping.

NOTE

If you have lost the majority of your control plane hosts, follow the disaster recovery procedure to restore to a previous cluster state instead of this procedure.

If the control plane certificates are not valid on the member being replaced, then you must follow the procedure to recover from expired control plane certificates instead of this procedure.

If a control plane node is lost and a new one is created, the etcd cluster Operator handles generating the new TLS certificates and adding the node as an etcd member.

4.2.1. Identifying an unhealthy etcd member

You can identify if your cluster has an unhealthy etcd member.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have taken an etcd backup. For more information, see "Backing up etcd data".

Procedure

1. Check the status of the **EtcdMembersAvailable** status condition using the following command:

```
$ oc get etcd -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="EtcdMembersAvailable")]}{.message}{"\n"}{end}'
```

2. Review the output:

```
2 of 3 members are available, ip-10-0-131-183.ec2.internal is unhealthy
```

This example output shows that the **ip-10-0-131-183.ec2.internal** etcd member is unhealthy.

4.2.2. Determining the state of the unhealthy etcd member

The steps to replace an unhealthy etcd member depend on which of the following states your etcd member is in:

- The machine is not running or the node is not ready
- The etcd pod is crashlooping

This procedure determines which state your etcd member is in. This enables you to know which procedure to follow to replace the unhealthy etcd member.

NOTE

If you are aware that the machine is not running or the node is not ready, but you expect it to return to a healthy state soon, then you do not need to perform a procedure to replace the etcd member. The etcd cluster Operator will automatically sync when the machine or node returns to a healthy state.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have identified an unhealthy etcd member.

Procedure

1. Determine if the **machine is not running**:

```
$ oc get machines -A -ojsonpath='{range .items[*]}{@.status.nodeRef.name}{"\t"}
{@.status.providerStatus.instanceState}{"\n"}' | grep -v running
```

Example output

```
ip-10-0-131-183.ec2.internal  stopped 1
```

1 This output lists the node and the status of the node's machine. If the status is anything other than **running**, then the **machine is not running**.

If the **machine is not running**, then follow the *Replacing an unhealthy etcd member whose machine is not running or whose node is not ready* procedure.

2. Determine if the **node is not ready**.
If either of the following scenarios are true, then the **node is not ready**.

- If the machine is running, then check whether the node is unreachable:

```
$ oc get nodes -o jsonpath='{range .items[*]}{"\n"}{.metadata.name}{"\t"}{range
.spec.taints[*]}{.key}{" "}' | grep unreachable
```

Example output

```
ip-10-0-131-183.ec2.internal node-role.kubernetes.io/master
node.kubernetes.io/unreachable node.kubernetes.io/unreachable 1
```

1 If the node is listed with an **unreachable** taint, then the **node is not ready**.

- If the node is still reachable, then check whether the node is listed as **NotReady**:

```
$ oc get nodes -l node-role.kubernetes.io/master | grep "NotReady"
```

Example output

```
ip-10-0-131-183.ec2.internal   NotReady   master   122m   v1.33.4 1
```

1 If the node is listed as **NotReady**, then the **node is not ready**.

If the **node is not ready**, then follow the *Replacing an unhealthy etcd member whose machine is not running or whose node is not ready* procedure.

3. Determine if the **etcd pod is crashlooping**
   If the machine is running and the node is ready, then check whether the etcd pod is crashlooping.

   a. Verify that all control plane nodes are listed as **Ready**:

```
$ oc get nodes -l node-role.kubernetes.io/master
```

Example output

```
NAME                         STATUS   ROLES    AGE     VERSION
ip-10-0-131-183.ec2.internal   Ready    master   6h13m   v1.33.4
ip-10-0-164-97.ec2.internal    Ready    master   6h13m   v1.33.4
ip-10-0-154-204.ec2.internal   Ready    master   6h13m   v1.33.4
```

   b. Check whether the status of an etcd pod is either **Error** or **CrashloopBackoff**:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
etcd-ip-10-0-131-183.ec2.internal          2/3     Error       7          6h9m 1
etcd-ip-10-0-164-97.ec2.internal           3/3     Running     0          6h6m
etcd-ip-10-0-154-204.ec2.internal          3/3     Running     0          6h6m
```

1 Since this status of this pod is **Error**, then the **etcd pod is crashlooping**

If the **etcd pod is crashlooping** then follow the *Replacing an unhealthy etcd member whose etcd pod is crashlooping* procedure.

4.2.3. Replacing the unhealthy etcd member

Depending on the state of your unhealthy etcd member, use one of the following procedures:

- Replacing an unhealthy etcd member whose machine is not running or whose node is not ready
- Installing a primary control plane node on an unhealthy cluster
- Replacing an unhealthy etcd member whose etcd pod is crashlooping
- Replacing an unhealthy stopped baremetal etcd member

4.2.3.1. Replacing an unhealthy etcd member whose machine is not running or whose node is not ready

This procedure details the steps to replace an etcd member that is unhealthy either because the machine is not running or because the node is not ready.

NOTE

If your cluster uses a control plane machine set, see "Recovering a degraded etcd Operator" in "Troubleshooting the control plane machine set" for an etcd recovery procedure.

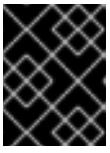
Prerequisites

- You have identified the unhealthy etcd member.
- You have verified that either the machine is not running or the node is not ready.

IMPORTANT

You must wait if you power off other control plane nodes. The control plane nodes must remain powered off until the replacement of an unhealthy etcd member is complete.

- You have access to the cluster as a user with the **cluster-admin** role.
- You have taken an etcd backup.



IMPORTANT

Before you perform this procedure, take an etcd backup so that you can restore your cluster if you experience any issues.

Procedure

1. Remove the unhealthy member.
   a. Choose a pod that is not on the affected node:

In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
etcd-ip-10-0-131-183.ec2.internal             3/3     Running     0          123m
etcd-ip-10-0-164-97.ec2.internal              3/3     Running     0          123m
etcd-ip-10-0-154-204.ec2.internal             3/3     Running     0          124m
```

b. Connect to the running etcd container, passing in the name of a pod that is not on the affected node:
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
```

c. View the member list:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+------------------+---------+-----------------------------+---------------------------+---------------
-----------+
|        ID        | STATUS  |            NAME             |        PEER ADDRS         |      CLIENT
ADDRS       |
+------------------+---------+-----------------------------+---------------------------+---------------
-----------+
| 6fc1e7c9db35841d | started | ip-10-0-131-183.ec2.internal | https://10.0.131.183:2380 |
https://10.0.131.183:2379 |
| 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+------------------+---------+-----------------------------+---------------------------+---------------
-----------+
```

Take note of the ID and the name of the unhealthy etcd member because these values are needed later in the procedure. The **$ etcdctl endpoint health** command will list the removed member until the procedure of replacement is finished and a new member is added.

d. Remove the unhealthy etcd member by providing the ID to the **etcdctl member remove** command:

```
sh-4.2# etcdctl member remove 6fc1e7c9db35841d
```

Example output

```
Member 6fc1e7c9db35841d removed from cluster ead669ce1fbfb346
```

e. View the member list again and verify that the member was removed:

```
sh-4.2# etcdctl member list -w table
```

Example output

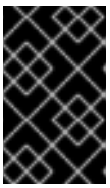
```
+------------------+---------+-----------------------------+--------------------------+----------------
----------+
|        ID        | STATUS  |            NAME             |        PEER ADDRS        |      CLIENT
ADDRS       |
+------------------+---------+-----------------------------+--------------------------+----------------
----------+
| 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+------------------+---------+-----------------------------+--------------------------+----------------
----------+
```

You can now exit the node shell.

2. Turn off the quorum guard by entering the following command:

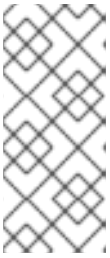
```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

This command ensures that you can successfully re-create secrets and roll out the static pods.



IMPORTANT

After you turn off the quorum guard, the cluster might be unreachable for a short time while the remaining etcd instances reboot to reflect the configuration change.



NOTE

etcd cannot tolerate any additional member failure when running with two members. Restarting either remaining member breaks the quorum and causes downtime in your cluster. The quorum guard protects etcd from restarts due to configuration changes that could cause downtime, so it must be disabled to complete this procedure.

3. Delete the affected node by running the following command:

```
$ oc delete node <node_name>
```

Example command

```
$ oc delete node ip-10-0-131-183.ec2.internal
```

4. Remove the old secrets for the unhealthy etcd member that was removed.

a. List the secrets for the unhealthy etcd member that was removed.

```
$ oc get secrets -n openshift-etcd | grep ip-10-0-131-183.ec2.internal 1
```

1 Pass in the name of the unhealthy etcd member that you took note of earlier in this procedure.

There is a peer, serving, and metrics secret as shown in the following output:

Example output

```
etcd-peer-ip-10-0-131-183.ec2.internal             kubernetes.io/tls                     2      47m
etcd-serving-ip-10-0-131-183.ec2.internal          kubernetes.io/tls                     2      47m
etcd-serving-metrics-ip-10-0-131-183.ec2.internal  kubernetes.io/tls                     2
47m
```

b. Delete the secrets for the unhealthy etcd member that was removed.

i. Delete the peer secret:

```
$ oc delete secret -n openshift-etcd etcd-peer-ip-10-0-131-183.ec2.internal
```

ii. Delete the serving secret:

```
$ oc delete secret -n openshift-etcd etcd-serving-ip-10-0-131-183.ec2.internal
```

iii. Delete the metrics secret:

```
$ oc delete secret -n openshift-etcd etcd-serving-metrics-ip-10-0-131-
183.ec2.internal
```

5. Check whether a control plane machine set exists by entering the following command:

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

- If the control plane machine set exists, delete and re-create the control plane machine. After this machine is re-created, a new revision is forced and etcd scales up automatically. For more information, see "Replacing an unhealthy etcd member whose machine is not running or whose node is not ready".
  If you are running installer-provisioned infrastructure, or you used the Machine API to create your machines, follow these steps. Otherwise, you must create the new control plane by using the same method that was used to originally create it.

a. Obtain the machine for the unhealthy member.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                              PHASE    TYPE        REGION      ZONE         AGE
NODE                          PROVIDERID                          STATE
clustername-8qw5l-master-0        Running  m4.xlarge   us-east-1   us-east-1a
```

```
3h37m  ip-10-0-131-183.ec2.internal  aws:///us-east-1a/i-0ec2782f8287dfb7e
stopped (1)
clustername-8qw5l-master-1                 Running  m4.xlarge  us-east-1  us-east-1b
3h37m  ip-10-0-154-204.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2                 Running  m4.xlarge  us-east-1  us-east-1c
3h37m  ip-10-0-164-97.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1  us-
east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-
east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

(1) This is the control plane machine for the unhealthy node, **ip-10-0-131-183.ec2.internal**.

b. Delete the machine of the unhealthy member:

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 (1)
```

(1) Specify the name of the control plane machine for the unhealthy node.

A new machine is automatically provisioned after deleting the machine of the unhealthy member.

c. Verify that a new machine was created:

```
$ oc get machines -n openshift-machine-api -o wide
```

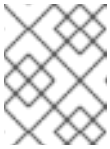
Example output

```
NAME                                 PHASE          TYPE        REGION      ZONE
AGE  NODE                          PROVIDERID                          STATE
clustername-8qw5l-master-1           Running        m4.xlarge   us-east-1   us-east-
1b   3h37m  ip-10-0-154-204.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2           Running        m4.xlarge   us-east-1   us-east-
1c   3h37m  ip-10-0-164-97.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-master-3           Provisioning   m4.xlarge   us-east-1   us-east-
1a   85s    ip-10-0-133-53.ec2.internal  aws:///us-east-1a/i-015b0888fe17bc2c8
running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd  Running      m4.large    us-east-1
us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running      m4.large    us-east-1   us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
```

```
clustername-8qw5l-worker-us-east-1c-pkg26  Running      m4.large  us-east-1
us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

1 The new machine, **clustername-8qw5l-master-3** is being created and is ready once the phase changes from **Provisioning** to **Running**.

It might take a few minutes for the new machine to be created. The etcd cluster Operator automatically syncs when the machine or node returns to a healthy state.



NOTE

Verify the subnet IDs that you are using for your machine sets to ensure that they end up in the correct availability zone.

- If the control plane machine set does not exist, delete and re-create the control plane machine. After this machine is re-created, a new revision is forced and etcd scales up automatically.
  If you are running installer-provisioned infrastructure, or you used the Machine API to create your machines, follow these steps. Otherwise, you must create the new control plane by using the same method that was used to originally create it.

  a. Obtain the machine for the unhealthy member.
  In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                                PHASE     TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-0          Running   m4.xlarge   us-east-1   us-east-1a
3h37m   ip-10-0-131-183.ec2.internal   aws:///us-east-1a/i-0ec2782f8287dfb7e
stopped 1
clustername-8qw5l-master-1          Running   m4.xlarge   us-east-1   us-east-1b
3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2          Running   m4.xlarge   us-east-1   us-east-1c
3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1  us-
east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-
east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

1 This is the control plane machine for the unhealthy node, **ip-10-0-131-183.ec2.internal**.

b. Save the machine configuration to a file on your file system:

```
$ oc get machine clustername-8qw5l-master-0 \ 1
    -n openshift-machine-api \
    -o yaml \
    > new-master-machine.yaml
```

1 Specify the name of the control plane machine for the unhealthy node.

c. Edit the **new-master-machine.yaml** file that was created in the previous step to assign a new name and remove unnecessary fields.

i. Remove the entire **status** section:

```
status:
  addresses:
  - address: 10.0.131.183
    type: InternalIP
  - address: ip-10-0-131-183.ec2.internal
    type: InternalDNS
  - address: ip-10-0-131-183.ec2.internal
    type: Hostname
  lastUpdated: "2020-04-20T17:44:29Z"
  nodeRef:
    kind: Node
    name: ip-10-0-131-183.ec2.internal
    uid: acca4411-af0d-4387-b73e-52b2484295ad
  phase: Running
  providerStatus:
    apiVersion: awsproviderconfig.openshift.io/v1beta1
    conditions:
    - lastProbeTime: "2020-04-20T16:53:50Z"
      lastTransitionTime: "2020-04-20T16:53:50Z"
      message: machine successfully created
      reason: MachineCreationSucceeded
      status: "True"
      type: MachineCreation
    instanceId: i-0fdb85790d76d0c3f
    instanceState: stopped
    kind: AWSMachineProviderStatus
```

ii. Change the **metadata.name** field to a new name.
Keep the same base name as the old machine and change the ending number to the next available number. In this example, **clustername-8qw5l-master-0** is changed to **clustername-8qw5l-master-3**.

For example:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  name: clustername-8qw5l-master-3
  ...
```

iii. Remove the **spec.providerID** field:

```
providerID: aws:///us-east-1a/i-0fdb85790d76d0c3f
```

d. Delete the machine of the unhealthy member:

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 1
```

1 Specify the name of the control plane machine for the unhealthy node.

e. Verify that the machine was deleted:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                                PHASE   TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1          Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2          Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-worker-us-east-1a-wbtgd   Running m4.large    us-east-1   us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running m4.large    us-east-1   us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26   Running m4.large    us-east-1   us-east-1c   3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-06861c00007751b0a   running
```

f. Create the new machine by using the **new-master-machine.yaml** file:

```
$ oc apply -f new-master-machine.yaml
```

g. Verify that the new machine was created:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                                PHASE   TYPE        REGION      ZONE         AGE     NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1          Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2          Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
```

```
clustername-8qw5l-master-3                 Provisioning  m4.xlarge  us-east-1  us-east-
1a  85s   ip-10-0-133-53.ec2.internal   aws:///us-east-1a/i-015b0888fe17bc2c8
running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd  Running       m4.large   us-east-1
us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running       m4.large   us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running       m4.large   us-east-1
us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

(1) The new machine, **clustername-8qw5l-master-3** is being created and is ready once the phase changes from **Provisioning** to **Running**.

It might take a few minutes for the new machine to be created. The etcd cluster Operator automatically syncs when the machine or node returns to a healthy state.

6. Turn the quorum guard back on by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

7. You can verify that the **unsupportedConfigOverrides** section is removed from the object by entering this command:

```
$ oc get etcd/cluster -oyaml
```

8. If you are using single-node OpenShift, restart the node. Otherwise, you might experience the following error in the etcd cluster Operator:

Example output

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-
0": the object has been modified; please apply your changes to the latest version and try
again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been
modified; please apply your changes to the latest version and try again, Operation cannot be
fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply
your changes to the latest version and try again]
```

Verification

1. Verify that all etcd pods are running properly.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
etcd-ip-10-0-133-53.ec2.internal          3/3     Running     0          7m49s
etcd-ip-10-0-164-97.ec2.internal          3/3     Running     0          123m
etcd-ip-10-0-154-204.ec2.internal         3/3     Running     0          124m
```

If the output from the previous command only lists two pods, you can manually force an etcd redeployment. In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$( date --rfc-3339=ns )"'"}}' --type=merge 1
```

1 The **forceRedeploymentReason** value must be unique, which is why a timestamp is appended.

2. Verify that there are exactly three etcd members.

   a. Connect to the running etcd container, passing in the name of a pod that was not on the affected node:
   In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
   ```

   b. View the member list:

   ```
   sh-4.2# etcdctl member list -w table
   ```

   Example output

   ```
   +------------------+---------+-----------------------------+---------------------------+---------------------------+
   |        ID        | STATUS  |             NAME            |        PEER ADDRS         |        CLIENT ADDRS       |
   +------------------+---------+-----------------------------+---------------------------+---------------------------+
   | 5eb0d6b8ca24730c | started |  ip-10-0-133-53.ec2.internal |  https://10.0.133.53:2380 | https://10.0.133.53:2379 |
   | 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 | https://10.0.164.97:2379 |
   | ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 | https://10.0.154.204:2379 |
   +------------------+---------+-----------------------------+---------------------------+---------------------------+
   ```

   If the output from the previous command lists more than three etcd members, you must carefully remove the unwanted member.

   WARNING

   Be sure to remove the correct etcd member; removing a good etcd member might lead to quorum loss.

Additional resources

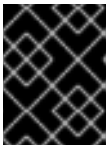
- Recovering a degraded etcd Operator
- Installing a primary control plane node on an unhealthy cluster

4.2.3.2. Replacing an unhealthy etcd member whose etcd pod is crashlooping

This procedure details the steps to replace an etcd member that is unhealthy because the etcd pod is crashlooping.

Prerequisites

- You have identified the unhealthy etcd member.
- You have verified that the etcd pod is crashlooping.
- You have access to the cluster as a user with the **cluster-admin** role.
- You have taken an etcd backup.



IMPORTANT

It is important to take an etcd backup before performing this procedure so that your cluster can be restored if you encounter any issues.

Procedure

1. Stop the crashlooping etcd pod.
   a. Debug the node that is crashlooping.
   In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

   ```
   $ oc debug node/ip-10-0-131-183.ec2.internal 1
   ```

   1 Replace this with the name of the unhealthy node.

   b. Change your root directory to /**host**:

   ```
   sh-4.2# chroot /host
   ```

   c. Move the existing etcd pod file out of the kubelet manifest directory:

   ```
   sh-4.2# mkdir /var/lib/etcd-backup
   ```

   ```
   sh-4.2# mv /etc/kubernetes/manifests/etcd-pod.yaml /var/lib/etcd-backup/
   ```

   d. Move the etcd data directory to a different location:

   ```
   sh-4.2# mv /var/lib/etcd/ /tmp
   ```

You can now exit the node shell.

2. Remove the unhealthy member.

   a. Choose a pod that is *not* on the affected node.
   In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

   ```
   $ oc -n openshift-etcd get pods -l k8s-app=etcd
   ```

   Example output

   ```
   etcd-ip-10-0-131-183.ec2.internal             2/3     Error       7          6h9m
   etcd-ip-10-0-164-97.ec2.internal              3/3     Running     0          6h6m
   etcd-ip-10-0-154-204.ec2.internal             3/3     Running     0          6h6m
   ```

   b. Connect to the running etcd container, passing in the name of a pod that is not on the affected node.
   In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
   ```

   c. View the member list:

   ```
   sh-4.2# etcdctl member list -w table
   ```

   Example output

   ```
   +------------------+---------+-----------------------------+---------------------------+---------------
   -----------+
   |        ID        | STATUS  |            NAME             |        PEER ADDRS         |      CLIENT
   ADDRS       |
   +------------------+---------+-----------------------------+---------------------------+---------------
   -----------+
   | 62bcf33650a7170a | started | ip-10-0-131-183.ec2.internal | https://10.0.131.183:2380 |
   https://10.0.131.183:2379 |
   | b78e2856655bc2eb | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
   https://10.0.164.97:2379 |
   | d022e10b498760d5 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380
   | https://10.0.154.204:2379 |
   +------------------+---------+-----------------------------+---------------------------+---------------
   -----------+
   ```

   Take note of the ID and the name of the unhealthy etcd member, because these values are needed later in the procedure.

   d. Remove the unhealthy etcd member by providing the ID to the **etcdctl member remove** command:

   ```
   sh-4.2# etcdctl member remove 62bcf33650a7170a
   ```

   Example output

```
Member 62bcf33650a7170a removed from cluster ead669ce1fbfb346
```

e. View the member list again and verify that the member was removed:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+------------------+---------+----------------------------+--------------------------+----------------
----------+
|        ID        | STATUS  |            NAME            |       PEER ADDRS         |      CLIENT
ADDRS       |
+------------------+---------+----------------------------+--------------------------+----------------
----------+
| b78e2856655bc2eb | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| d022e10b498760d5 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380
| https://10.0.154.204:2379 |
+------------------+---------+----------------------------+--------------------------+----------------
----------+
```

You can now exit the node shell.

3. Turn off the quorum guard by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

This command ensures that you can successfully re-create secrets and roll out the static pods.

4. Remove the old secrets for the unhealthy etcd member that was removed.

a. List the secrets for the unhealthy etcd member that was removed.

```
$ oc get secrets -n openshift-etcd | grep ip-10-0-131-183.ec2.internal 1
```

1 Pass in the name of the unhealthy etcd member that you took note of earlier in this procedure.

There is a peer, serving, and metrics secret as shown in the following output:

Example output

```
etcd-peer-ip-10-0-131-183.ec2.internal             kubernetes.io/tls                     2      47m
etcd-serving-ip-10-0-131-183.ec2.internal          kubernetes.io/tls                     2      47m
etcd-serving-metrics-ip-10-0-131-183.ec2.internal   kubernetes.io/tls                     2
47m
```

b. Delete the secrets for the unhealthy etcd member that was removed.

i. Delete the peer secret:

```
$ oc delete secret -n openshift-etcd etcd-peer-ip-10-0-131-183.ec2.internal
```

ii. Delete the serving secret:

```
$ oc delete secret -n openshift-etcd etcd-serving-ip-10-0-131-183.ec2.internal
```

iii. Delete the metrics secret:

```
$ oc delete secret -n openshift-etcd etcd-serving-metrics-ip-10-0-131-183.ec2.internal
```

5. Force etcd redeployment.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "single-master-recovery-'"$( date --rfc-3339=ns )"'"}}' --type=merge (1)
```

(1) The **forceRedeploymentReason** value must be unique, which is why a timestamp is appended.

When the etcd cluster Operator performs a redeployment, it ensures that all control plane nodes have a functioning etcd pod.

6. Turn the quorum guard back on by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

7. You can verify that the **unsupportedConfigOverrides** section is removed from the object by entering this command:

```
$ oc get etcd/cluster -oyaml
```

8. If you are using single-node OpenShift, restart the node. Otherwise, you might encounter the following error in the etcd cluster Operator:

Example output

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply your changes to the latest version and try again]
```

Verification

- Verify that the new member is available and healthy.

  a. Connect to the running etcd container again.
  In a terminal that has access to the cluster as a cluster-admin user, run the following command:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
```

b. Verify that all members are healthy:

```
sh-4.2# etcdctl endpoint health
```

Example output

```
https://10.0.131.183:2379 is healthy: successfully committed proposal: took =
16.671434ms
https://10.0.154.204:2379 is healthy: successfully committed proposal: took =
16.698331ms
https://10.0.164.97:2379 is healthy: successfully committed proposal: took =
16.621645ms
```

4.2.3.3. Replacing an unhealthy bare metal etcd member whose machine is not running or whose node is not ready

This procedure details the steps to replace a bare metal etcd member that is unhealthy either because the machine is not running or because the node is not ready.

If you are running installer-provisioned infrastructure or you used the Machine API to create your machines, follow these steps. Otherwise you must create the new control plane node using the same method that was used to originally create it.

Prerequisites

- You have identified the unhealthy bare metal etcd member.
- 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 have taken an etcd backup.

IMPORTANT

You must take an etcd backup before performing this procedure so that your cluster can be restored if you encounter any issues.

Procedure

1. Verify and remove the unhealthy member.

   a. Choose a pod that is *not* on the affected node:
   In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

Example output

```
etcd-openshift-control-plane-0  5/5  Running  11  3h56m  192.168.10.9  openshift-
control-plane-0  <none>          <none>
etcd-openshift-control-plane-1  5/5  Running  0   3h54m  192.168.10.10  openshift-
```

```
control-plane-1   <none>          <none>
etcd-openshift-control-plane-2  5/5  Running  0  3h58m  192.168.10.11  openshift-
control-plane-2   <none>          <none>
```

b. Connect to the running etcd container, passing in the name of a pod that is not on the affected node:
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc rsh -n openshift-etcd etcd-openshift-control-plane-0
```

c. View the member list:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+------------------+---------+--------------------+--------------------------+--------------------------
-+---------------------+
| ID               | STATUS  | NAME               | PEER ADDRS               | CLIENT
ADDRS                | IS LEARNER |
+------------------+---------+--------------------+--------------------------+--------------------------
-+---------------------+
| 7a8197040a5126c8 | started | openshift-control-plane-2 | https://192.168.10.11:2380/ |
https://192.168.10.11:2379/ | false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380/ |
https://192.168.10.10:2379/ | false |
| cc3830a72fc357f9 | started | openshift-control-plane-0 | https://192.168.10.9:2380/ |
https://192.168.10.9:2379/   | false |
+------------------+---------+--------------------+--------------------------+--------------------------
-+---------------------+
```

Take note of the ID and the name of the unhealthy etcd member, because these values are required later in the procedure. The **etcdctl endpoint health** command will list the removed member until the replacement procedure is completed and the new member is added.

d. Remove the unhealthy etcd member by providing the ID to the **etcdctl member remove** command:

WARNING

Be sure to remove the correct etcd member; removing a good etcd member might lead to quorum loss.

```
sh-4.2# etcdctl member remove 7a8197040a5126c8
```

Example output

```
Member 7a8197040a5126c8 removed from cluster b23536c33f2cdd1b
```

e. View the member list again and verify that the member was removed:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+------------------+---------+--------------------+--------------------------+--------------------------
-+------------------------+
| ID               | STATUS  | NAME                    | PEER ADDRS               | CLIENT
ADDRS               | IS LEARNER |
+------------------+---------+--------------------+--------------------------+--------------------------
-+------------------------+
| cc3830a72fc357f9 | started | openshift-control-plane-2 | https://192.168.10.11:2380/ |
https://192.168.10.11:2379/ | false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380/ |
https://192.168.10.10:2379/ | false |
+------------------+---------+--------------------+--------------------------+--------------------------
-+------------------------+
```

You can now exit the node shell.

IMPORTANT

After you remove the member, the cluster might be unreachable for a short time while the remaining etcd instances reboot.

2. Turn off the quorum guard by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

This command ensures that you can successfully re-create secrets and roll out the static pods.

3. Remove the old secrets for the unhealthy etcd member that was removed by running the following commands.

a. List the secrets for the unhealthy etcd member that was removed.

```
$ oc get secrets -n openshift-etcd | grep openshift-control-plane-2
```

Pass in the name of the unhealthy etcd member that you took note of earlier in this procedure.

There is a peer, serving, and metrics secret as shown in the following output:

```
etcd-peer-openshift-control-plane-2              kubernetes.io/tls   2   134m
etcd-serving-metrics-openshift-control-plane-2  kubernetes.io/tls   2   134m
etcd-serving-openshift-control-plane-2          kubernetes.io/tls   2   134m
```

b. Delete the secrets for the unhealthy etcd member that was removed.

i. Delete the peer secret:

```
$ oc delete secret etcd-peer-openshift-control-plane-2 -n openshift-etcd

secret "etcd-peer-openshift-control-plane-2" deleted
```

ii. Delete the serving secret:

```
$ oc delete secret etcd-serving-metrics-openshift-control-plane-2 -n openshift-etcd

secret "etcd-serving-metrics-openshift-control-plane-2" deleted
```

iii. Delete the metrics secret:

```
$ oc delete secret etcd-serving-openshift-control-plane-2 -n openshift-etcd

secret "etcd-serving-openshift-control-plane-2" deleted
```

4. Obtain the machine for the unhealthy member.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                              PHASE     TYPE   REGION   ZONE   AGE     NODE
PROVIDERID                                                              STATE
examplecluster-control-plane-0    Running                          3h11m   openshift-control-plane-0
baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-
0aa41222964e   externally provisioned  (1)
examplecluster-control-plane-1    Running                          3h11m   openshift-control-plane-1
baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
03b20280a3e1   externally provisioned
examplecluster-control-plane-2    Running                          3h11m   openshift-control-plane-2
baremetalhost:///openshift-machine-api/openshift-control-plane-2/3354bdac-61d8-410f-be5b-
6a395b056135   externally provisioned
examplecluster-compute-0          Running                          165m    openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f         provisioned
examplecluster-compute-1          Running                          165m    openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9         provisioned
```

(1) This is the control plane machine for the unhealthy node, **examplecluster-control-plane-2**.

5. Ensure that the Bare Metal Operator is available by running the following command:

```
$ oc get clusteroperator baremetal
```

Example output

```
NAME        VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
baremetal   4.20.0    True        False         False      3d15h
```

6. Remove the old **BareMetalHost** object by running the following command:

```
$ oc delete bmh openshift-control-plane-2 -n openshift-machine-api
```

Example output

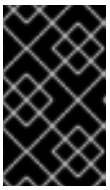
```
baremetalhost.metal3.io "openshift-control-plane-2" deleted
```

7. Delete the machine of the unhealthy member by running the following command:

```
$ oc delete machine -n openshift-machine-api examplecluster-control-plane-2
```

After you remove the **BareMetalHost** and **Machine** objects, then the **Machine** controller automatically deletes the **Node** object.

If deletion of the machine is delayed for any reason or the command is obstructed and delayed, you can force deletion by removing the machine object finalizer field.



IMPORTANT

Do not interrupt machine deletion by pressing **Ctrl+c**. You must allow the command to proceed to completion. Open a new terminal window to edit and delete the finalizer fields.

A new machine is automatically provisioned after deleting the machine of the unhealthy member.

a. Edit the machine configuration by running the following command:

```
$ oc edit machine -n openshift-machine-api examplecluster-control-plane-2
```

b. Delete the following fields in the **Machine** custom resource, and then save the updated file:

```
finalizers:
- machine.machine.openshift.io
```

Example output

```
machine.machine.openshift.io/examplecluster-control-plane-2 edited
```

8. Verify that the machine was deleted by running the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                        PHASE     TYPE   REGION   ZONE   AGE     NODE
PROVIDERID                                                        STATE
```

```
examplecluster-control-plane-0    Running                 3h11m   openshift-control-plane-0
baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-
0aa41222964e   externally provisioned
examplecluster-control-plane-1    Running                 3h11m   openshift-control-plane-1
baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
03b20280a3e1   externally provisioned
examplecluster-compute-0          Running                 165m    openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f         provisioned
examplecluster-compute-1          Running                 165m    openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9         provisioned
```

9. Verify that the node has been deleted by running the following command:

```
$ oc get nodes

NAME                    STATUS ROLES    AGE    VERSION
openshift-control-plane-0 Ready master 3h24m v1.33.4
openshift-control-plane-1 Ready master 3h24m v1.33.4
openshift-compute-0       Ready worker 176m v1.33.4
openshift-compute-1       Ready worker 176m v1.33.4
```

10. Create the new **BareMetalHost** object and the secret to store the BMC credentials:

```
$ cat <<EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: openshift-control-plane-2-bmc-secret
  namespace: openshift-machine-api
data:
  password: <password>
  username: <username>
type: Opaque
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: openshift-control-plane-2
  namespace: openshift-machine-api
spec:
  automatedCleaningMode: disabled
  bmc:
    address: redfish://10.46.61.18:443/redfish/v1/Systems/1
    credentialsName: openshift-control-plane-2-bmc-secret
    disableCertificateVerification: true
  bootMACAddress: 48:df:37:b0:8a:a0
  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
```

NOTE

The username and password can be found from the other bare metal host's secrets. The protocol to use in **bmc:address** can be taken from other bmh objects.

IMPORTANT

If you reuse the **BareMetalHost** object definition from an existing control plane host, do not leave the **externallyProvisioned** field set to **true**.

Existing control plane **BareMetalHost** objects may have the **externallyProvisioned** flag set to **true** if they were provisioned by the OpenShift Container Platform installation program.

After the inspection is complete, the **BareMetalHost** object is created and available to be provisioned.

11. Verify the creation process using available **BareMetalHost** objects:

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

NAME                 STATE                  CONSUMER                      ONLINE ERROR   AGE
openshift-control-plane-0 externally provisioned examplecluster-control-plane-0 true
4h48m
openshift-control-plane-1 externally provisioned examplecluster-control-plane-1 true
4h48m
openshift-control-plane-2 available          examplecluster-control-plane-3 true        47m
openshift-compute-0       provisioned        examplecluster-compute-0       true        4h48m
openshift-compute-1       provisioned        examplecluster-compute-1       true        4h48m
```

a. Verify that a new machine has been created:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                              PHASE     TYPE   REGION   ZONE   AGE     NODE
PROVIDERID                                                                 STATE
examplecluster-control-plane-0        Running                          3h11m   openshift-control-
plane-0   baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-
3ff2-41c5-b099-0aa41222964e   externally provisioned (1)
examplecluster-control-plane-1        Running                          3h11m   openshift-control-
plane-1   baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-
329c-475e-8d81-03b20280a3e1   externally provisioned
examplecluster-control-plane-2        Running                          3h11m   openshift-control-
plane-2   baremetalhost:///openshift-machine-api/openshift-control-plane-2/3354bdac-
61d8-410f-be5b-6a395b056135   externally provisioned
examplecluster-compute-0              Running                          165m    openshift-compute-
```

```
0          baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-
4bb3-80ec-13a31858241f          provisioned
examplecluster-compute-1          Running                    165m    openshift-compute-
1          baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-
4241-91dc-e7ea72ab13b9          provisioned
```

1 The new machine, **clustername-8qw5l-master-3** is being created and is ready after the phase changes from **Provisioning** to **Running**.

It should take a few minutes for the new machine to be created. The etcd cluster Operator will automatically sync when the machine or node returns to a healthy state.

b. Verify that the bare metal host becomes provisioned and no error reported by running the following command:

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

Example output

```
$ oc get bmh -n openshift-machine-api
NAME                 STATE                CONSUMER                  ONLINE ERROR AGE
openshift-control-plane-0 externally provisioned examplecluster-control-plane-0 true
4h48m
openshift-control-plane-1 externally provisioned examplecluster-control-plane-1 true
4h48m
openshift-control-plane-2 provisioned          examplecluster-control-plane-3 true
47m
openshift-compute-0       provisioned          examplecluster-compute-0       true
4h48m
openshift-compute-1       provisioned          examplecluster-compute-1       true
4h48m
```

c. Verify that the new node is added and in a ready state by running this command:

```
$ oc get nodes
```

Example output

```
$ oc get nodes
NAME                 STATUS ROLES    AGE    VERSION
openshift-control-plane-0 Ready master 4h26m v1.33.4
openshift-control-plane-1 Ready master 4h26m v1.33.4
openshift-control-plane-2 Ready master 12m    v1.33.4
openshift-compute-0       Ready worker 3h58m v1.33.4
openshift-compute-1       Ready worker 3h58m v1.33.4
```

12. Turn the quorum guard back on by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

13. You can verify that the **unsupportedConfigOverrides** section is removed from the object by entering this command:

```
$ oc get etcd/cluster -oyaml
```

14. If you are using single-node OpenShift, restart the node. Otherwise, you might encounter the following error in the etcd cluster Operator:

Example output

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply your changes to the latest version and try again]
```

Verification

1. Verify that all etcd pods are running properly.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
etcd-openshift-control-plane-0     5/5     Running     0     105m
etcd-openshift-control-plane-1     5/5     Running     0     107m
etcd-openshift-control-plane-2     5/5     Running     0     103m
```

If the output from the previous command only lists two pods, you can manually force an etcd redeployment. In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$( date --rfc-3339=ns )"'"}}' --type=merge 1
```

1 The **forceRedeploymentReason** value must be unique, which is why a timestamp is appended.

To verify there are exactly three etcd members, connect to the running etcd container, passing in the name of a pod that was not on the affected node. In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc rsh -n openshift-etcd etcd-openshift-control-plane-0
```

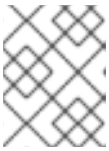
2. View the member list:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+------------------+---------+--------------------+--------------------------+--------------------------+---
--------------+
```

```
|        ID        | STATUS |       NAME       |       PEER ADDRS       |      CLIENT ADDRS
|   IS LEARNER    |
+------------------+---------+------------------+------------------------+------------------------+---
--------------+
| 7a8197040a5126c8 | started | openshift-control-plane-2 | https://192.168.10.11:2380 |
https://192.168.10.11:2379 |    false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380 |
https://192.168.10.10:2379 |    false |
| cc3830a72fc357f9 | started | openshift-control-plane-0 | https://192.168.10.9:2380 |
https://192.168.10.9:2379 |     false |
+------------------+---------+------------------+------------------------+------------------------+---
--------------+
```



NOTE

If the output from the previous command lists more than three etcd members, you must carefully remove the unwanted member.

3. Verify that all etcd members are healthy by running the following command:

```
# etcdctl endpoint health --cluster
```

Example output

```
https://192.168.10.10:2379 is healthy: successfully committed proposal: took = 8.973065ms
https://192.168.10.9:2379 is healthy: successfully committed proposal: took = 11.559829ms
https://192.168.10.11:2379 is healthy: successfully committed proposal: took = 11.665203ms
```

4. Validate that all nodes are at the latest revision by running the following command:

```
$ oc get etcd -o=jsonpath='{range.items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

```
AllNodesAtLatestRevision
```

Additional resources

- Quorum protection with machine lifecycle hooks

4.3. DISASTER RECOVERY

The disaster recovery documentation provides information for administrators on how to recover from several disaster situations that might occur with their OpenShift Container Platform cluster. As an administrator, you might need to follow one or more of the following procedures to return your cluster to a working state.

IMPORTANT

Disaster recovery requires you to have at least one healthy control plane host.

4.3.1. Quorum restoration

You can use the **quorum-restore.sh** script to restore etcd quorum on clusters that are offline due to quorum loss. When quorum is lost, the OpenShift Container Platform API becomes read-only. After quorum is restored, the OpenShift Container Platform API returns to read/write mode.

4.3.1.1. Restoring etcd quorum for high availability clusters

You can use the **quorum-restore.sh** script to restore etcd quorum on clusters that are offline due to quorum loss. When quorum is lost, the OpenShift Container Platform API becomes read-only. After quorum is restored, the OpenShift Container Platform API returns to read/write mode.

The **quorum-restore.sh** script instantly brings back a new single-member etcd cluster based on its local data directory and marks all other members as invalid by retiring the previous cluster identifier. No prior backup is required to restore the control plane from.

For high availability (HA) clusters, a three-node HA cluster requires you to shut down etcd on two hosts to avoid a cluster split. On four-node and five-node HA clusters, you must shut down three hosts. Quorum requires a simple majority of nodes. The minimum number of nodes required for quorum on a three-node HA cluster is two. On four-node and five-node HA clusters, the minimum number of nodes required for quorum is three. If you start a new cluster from backup on your recovery host, the other etcd members might still be able to form quorum and continue service.

WARNING

You might experience data loss if the host that runs the restoration does not have all data replicated to it.

IMPORTANT

Quorum restoration should not be used to decrease the number of nodes outside of the restoration process. Decreasing the number of nodes results in an unsupported cluster configuration.

Prerequisites

- You have SSH access to the node used to restore quorum.

Procedure

1. Select a control plane host to use as the recovery host. You run the restore operation on this host.
   a. List the running etcd pods by running the following command:

      ```
      $ oc get pods -n openshift-etcd -l app=etcd --field-selector="status.phase==Running"
      ```

   b. Choose a pod and run the following command to obtain its IP address:

      ```
      $ oc exec -n openshift-etcd <etcd-pod> -c etcdctl -- etcdctl endpoint status -w table
      ```

      Note the IP address of a member that is not a learner and has the highest Raft index.

c. Run the following command and note the node name that corresponds to the IP address of the chosen etcd member:

```
$ oc get nodes -o jsonpath='{range .items[*]}[{.metadata.name},{.status.addresses[?(@.type=="InternalIP")].address}]{end}'
```

2. Using SSH, connect to the chosen recovery node and run the following command to restore etcd quorum:

```
$ sudo -E /usr/local/bin/quorum-restore.sh
```

After a few minutes, the nodes that went down are automatically synchronized with the node that the recovery script was run on. Any remaining online nodes automatically rejoin the new etcd cluster created by the **quorum-restore.sh** script. This process takes a few minutes.

3. Exit the SSH session.

4. Return to a three-node configuration if any nodes are offline. Repeat the following steps for each node that is offline to delete and re-create them. After the machines are re-created, a new revision is forced and etcd automatically scales up.

- If you use a user-provisioned bare-metal installation, you can re-create a control plane machine by using the same method that you used to originally create it. For more information, see "Installing a user-provisioned cluster on bare metal".

WARNING

Do not delete and re-create the machine for the recovery host.

- If you are running installer-provisioned infrastructure, or you used the Machine API to create your machines, follow these steps:

WARNING

Do not delete and re-create the machine for the recovery host.

For bare-metal installations on installer-provisioned infrastructure, control plane machines are not re-created. For more information, see "Replacing a bare-metal control plane node".

a. Obtain the machine for one of the offline nodes.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

-

Example output

```
NAME                                PHASE    TYPE       REGION    ZONE       AGE
NODE                                PROVIDERID                          STATE
clustername-8qw5l-master-0          Running  m4.xlarge  us-east-1 us-east-1a
3h37m  ip-10-0-131-183.ec2.internal  aws:///us-east-1a/i-0ec2782f8287dfb7e
stopped (1)
clustername-8qw5l-master-1          Running  m4.xlarge  us-east-1 us-east-1b
3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2          Running  m4.xlarge  us-east-1 us-east-1c
3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1  us-
east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-
east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

(1) This is the control plane machine for the offline node, **ip-10-0-131-183.ec2.internal**.

b. Delete the machine of the offline node by running:

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 (1)
```

(1) Specify the name of the control plane machine for the offline node.

A new machine is automatically provisioned after deleting the machine of the offline node.

5. Verify that a new machine has been created by running:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                                PHASE         TYPE       REGION    ZONE       AGE
NODE                                PROVIDERID                          STATE
clustername-8qw5l-master-1          Running       m4.xlarge  us-east-1 us-east-1b
3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-096c349b700a19631  running
clustername-8qw5l-master-2          Running       m4.xlarge  us-east-1 us-east-1c
3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba  running
clustername-8qw5l-master-3          Provisioning  m4.xlarge  us-east-1 us-east-1a  85s
ip-10-0-173-171.ec2.internal  aws:///us-east-1a/i-015b0888fe17bc2c8  running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1  us-east-1a
3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1  us-east-1b
```

```
3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-east-1c
3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-06861c00007751b0a  running
```

1 The new machine, **clustername-8qw5l-master-3** is being created and is ready after the phase changes from **Provisioning** to **Running**.

It might take a few minutes for the new machine to be created. The etcd cluster Operator will automatically synchronize when the machine or node returns to a healthy state.

a. Repeat these steps for each node that is offline.

6. Wait until the control plane recovers by running the following command:

```
$ oc adm wait-for-stable-cluster
```

NOTE

It can take up to 15 minutes for the control plane to recover.

Troubleshooting

- If you see no progress rolling out the etcd static pods, you can force redeployment from the etcd cluster Operator by running the following command:

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$(date --rfc-3339=ns )""}}' --type=merge
```

Additional resources

- Installing a user-provisioned cluster on bare metal
- Replacing a bare-metal control plane node

NOTE

If you have a majority of your control plane nodes still available and have an etcd quorum, replace a single unhealthy etcd member.

4.3.2. Restoring to a previous cluster state

To restore the cluster to a previous state, you must have previously backed up the **etcd** data by creating a snapshot. You will use this snapshot to restore the cluster state. For more information, see "Backing up etcd data".

If applicable, you might also need to recover from expired control plane certificates .

WARNING

Restoring to a previous cluster state is a destructive and destablizing action to take on a running cluster. This procedure should only be used as a last resort.

Before performing a restore, see "About restoring to a previous cluster state" for more information on the impact to the cluster.

4.3.2.1. About restoring to a previous cluster state

To restore the cluster to a previous state, you must have previously backed up the **etcd** data by creating a snapshot. You will use this snapshot to restore the cluster state. For more information, see "Backing up etcd data".

You can use an etcd backup to restore your cluster to a previous state. This can be used to recover from the following situations:

- The cluster has lost the majority of control plane hosts (quorum loss).
- An administrator has deleted something critical and must restore to recover the cluster.

WARNING

Restoring to a previous cluster state is a destructive and destablizing action to take on a running cluster. This should only be used as a last resort.

If you are able to retrieve data using the Kubernetes API server, then etcd is available and you should not restore using an etcd backup.

Restoring etcd effectively takes a cluster back in time and all clients will experience a conflicting, parallel history. This can impact the behavior of watching components like kubelets, Kubernetes controller managers, persistent volume controllers, and OpenShift Container Platform Operators, including the network Operator.

It can cause Operator churn when the content in etcd does not match the actual content on disk, causing Operators for the Kubernetes API server, Kubernetes controller manager, Kubernetes scheduler, and etcd to get stuck when files on disk conflict with content in etcd. This can require manual actions to resolve the issues.

In extreme cases, the cluster can lose track of persistent volumes, delete critical workloads that no longer exist, reimage machines, and rewrite CA bundles with expired certificates.

4.3.2.2. Restoring to a previous cluster state for a single node

You can use a saved etcd backup to restore a previous cluster state on a single node.

IMPORTANT

When you restore your cluster, you must use an etcd backup that was taken from the same z-stream release. For example, an OpenShift Container Platform 4.20.2 cluster must use an etcd backup that was taken from 4.20.2.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role through a certificate-based **kubeconfig** file, like the one that was used during installation.
- You have SSH access to control plane hosts.
- A backup directory containing both the etcd snapshot and the resources for the static pods, which were from the same backup. The file names in the directory must be in the following formats: **snapshot_<datetimestamp>.db** and **static_kuberesources_<datetimestamp>.tar.gz**.

Procedure

1. Use SSH to connect to the single node and copy the etcd backup to the /**home/core** directory by running the following command:

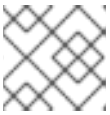
```
$ cp <etcd_backup_directory> /home/core
```

2. Run the following command in the single node to restore the cluster from a previous backup:

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/<etcd_backup_directory>
```

3. Exit the SSH session.
4. Monitor the recovery progress of the control plane by running the following command:

```
$ oc adm wait-for-stable-cluster
```



NOTE

It can take up to 15 minutes for the control plane to recover.

4.3.2.3. Restoring to a previous cluster state for more than one node

You can use a saved etcd backup to restore a previous cluster state or restore a cluster that has lost the majority of control plane hosts.

For high availability (HA) clusters, a three-node HA cluster requires you to shut down etcd on two hosts to avoid a cluster split. On four-node and five-node HA clusters, you must shut down three hosts. Quorum requires a simple majority of nodes. The minimum number of nodes required for quorum on a three-node HA cluster is two. On four-node and five-node HA clusters, the minimum number of nodes required for quorum is three. If you start a new cluster from backup on your recovery host, the other etcd members might still be able to form quorum and continue service.

NOTE

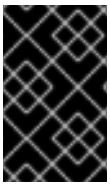
If your cluster uses a control plane machine set, see "Recovering a degraded etcd Operator" in "Troubleshooting the control plane machine set" for an etcd recovery procedure. For OpenShift Container Platform on a single node, see "Restoring to a previous cluster state for a single node".

IMPORTANT

When you restore your cluster, you must use an etcd backup that was taken from the same z-stream release. For example, an OpenShift Container Platform 4.20.2 cluster must use an etcd backup that was taken from 4.20.2.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role through a certificate-based **kubeconfig** file, like the one that was used during installation.
- A healthy control plane host to use as the recovery host.
- You have SSH access to control plane hosts.
- A backup directory containing both the **etcd** snapshot and the resources for the static pods, which were from the same backup. The file names in the directory must be in the following formats: **snapshot_<datetimestamp>.db** and **static_kuberesources_<datetimestamp>.tar.gz**.
- Nodes must be accessible or bootable.

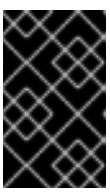


IMPORTANT

For non-recovery control plane nodes, it is not required to establish SSH connectivity or to stop the static pods. You can delete and re-create other non-recovery, control plane machines, one by one.

Procedure

1. Select a control plane host to use as the recovery host. This is the host that you run the restore operation on.
2. Establish SSH connectivity to each of the control plane nodes, including the recovery host. **kube-apiserver** becomes inaccessible after the restore process starts, so you cannot access the control plane nodes. For this reason, it is recommended to establish SSH connectivity to each control plane host in a separate terminal.

   IMPORTANT

   If you do not complete this step, you will not be able to access the control plane hosts to complete the restore procedure, and you will be unable to recover your cluster from this state.

3. Using SSH, connect to each control plane node and run the following command to disable etcd:

   ```
   $ sudo -E /usr/local/bin/disable-etcd.sh
   ```

4. Copy the etcd backup directory to the recovery control plane host.
   This procedure assumes that you copied the **backup** directory containing the etcd snapshot and the resources for the static pods to the **/home/core/** directory of your recovery control plane host.

5. Use SSH to connect to the recovery host and restore the cluster from a previous backup by running the following command:

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/<etcd-backup-directory>
```

6. Exit the SSH session.

7. Once the API responds, turn off the etcd Operator quorum guard by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

8. Monitor the recovery progress of the control plane by running the following command:

```
$ oc adm wait-for-stable-cluster
```

NOTE

It can take up to 15 minutes for the control plane to recover.

9. Once recovered, enable the quorum guard by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

Troubleshooting

If you see no progress rolling out the etcd static pods, you can force redeployment from the **cluster-etcd-operator** by running the following command:

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$(date --rfc-3339=ns
)"'"}}' --type=merge
```

Additional resources

- Recovering a degraded etcd Operator

4.3.2.4. Restoring a cluster manually from an etcd backup

The restore procedure described in the section "Restoring to a previous cluster state":

- Requires the complete recreation of 2 control plane nodes, which might be a complex procedure for clusters installed with the UPI installation method, since an UPI installation does not create any **Machine** or **ControlPlaneMachineset** for the control plane nodes.

- Uses the script /usr/local/bin/cluster-restore.sh, which starts a new single-member etcd cluster and then scales it to three members.

In contrast, this procedure:

- Does not require recreating any control plane nodes.
- Directly starts a three-member etcd cluster.

If the cluster uses a **MachineSet** for the control plane, it is suggested to use the "Restoring to a previous cluster state" for a simpler etcd recovery procedure.

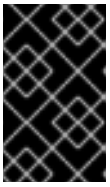
When you restore your cluster, you must use an etcd backup that was taken from the same z-stream release. For example, an OpenShift Container Platform 4.7.2 cluster must use an etcd backup that was taken from 4.7.2.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role; for example, the **kubeadmin** user.
- SSH access to *all* control plane hosts, with a host user allowed to become **root**; for example, the default **core** host user.
- A backup directory containing both a previous etcd snapshot and the resources for the static pods from the same backup. The file names in the directory must be in the following formats: **snapshot_<datetimestamp>.db** and **static_kuberesources_<datetimestamp>.tar.gz**.

Procedure

1. Use SSH to connect to each of the control plane nodes.
   The Kubernetes API server becomes inaccessible after the restore process starts, so you cannot access the control plane nodes. For this reason, it is recommended to use a SSH connection for each control plane host you are accessing in a separate terminal.

   IMPORTANT

   If you do not complete this step, you will not be able to access the control plane hosts to complete the restore procedure, and you will be unable to recover your cluster from this state.

2. Copy the etcd backup directory to each control plane host.
   This procedure assumes that you copied the **backup** directory containing the etcd snapshot and the resources for the static pods to the **/home/core/assets** directory of each control plane host. You might need to create such **assets** folder if it does not exist yet.

3. Stop the static pods on all the control plane nodes; one host at a time.

   a. Move the existing Kubernetes API Server static pod manifest out of the kubelet manifest directory.

   ```
   $ mkdir -p /root/manifests-backup
   ```

   ```
   $ mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /root/manifests-backup/
   ```

   b. Verify that the Kubernetes API Server containers have stopped with the command:

   ```
   $ crictl ps | grep kube-apiserver | grep -E -v "operator|guard"
   ```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

c. If the Kubernetes API Server containers are still running, terminate them manually with the following command:

```
$ crictl stop <container_id>
```

d. Repeat the same steps for **kube-controller-manager-pod.yaml**, **kube-scheduler-pod.yaml** and **finally etcd-pod.yaml**.

i. Stop the **kube-controller-manager** pod with the following command:

```
$ mv /etc/kubernetes/manifests/kube-controller-manager-pod.yaml /root/manifests-backup/
```

ii. Check if the containers are stopped using the following command:

```
$ crictl ps | grep kube-controller-manager | grep -E -v "operator|guard"
```

iii. Stop the **kube-scheduler** pod using the following command:

```
$ mv /etc/kubernetes/manifests/kube-scheduler-pod.yaml /root/manifests-backup/
```

iv. Check if the containers are stopped using the following command:

```
$ crictl ps | grep kube-scheduler | grep -E -v "operator|guard"
```

v. Stop the **etcd** pod using the following command:

```
$ mv /etc/kubernetes/manifests/etcd-pod.yaml /root/manifests-backup/
```

vi. Check if the containers are stopped using the following command:

```
$ crictl ps | grep etcd | grep -E -v "operator|guard"
```

4. On each control plane host, save the current **etcd** data, by moving it into the **backup** folder:

```
$ mkdir /home/core/assets/old-member-data
```

```
$ mv /var/lib/etcd/member /home/core/assets/old-member-data
```

This data will be useful in case the **etcd** backup restore does not work and the **etcd** cluster must be restored to the current state.

5. Find the correct etcd parameters for each control plane host.

a. The value for **<ETCD_NAME>** is unique for the each control plane host, and it is equal to the value of the **ETCD_NAME** variable in the manifest **/etc/kubernetes/static-pod-resources/etcd-certs/configmaps/restore-etcd-pod/pod.yaml** file in the specific control plane host. It can be found with the command:

```
RESTORE_ETCD_POD_YAML="/etc/kubernetes/static-pod-resources/etcd-certs/configmaps/restore-etcd-pod/pod.yaml"
cat $RESTORE_ETCD_POD_YAML | \
  grep -A 1 $(cat $RESTORE_ETCD_POD_YAML | grep 'export ETCD_NAME' | grep -Eo 'NODE_.+_ETCD_NAME') | \
  grep -Po '(?<=value: ").+(?=")'
```

b. The value for **<UUID>** can be generated in a control plane host with the command:

```
$ uuidgen
```

NOTE

The value for **<UUID>** must be generated only once. After generating **UUID** on one control plane host, do not generate it again on the others. The same **UUID** will be used in the next steps on all control plane hosts.

c. The value for **ETCD_NODE_PEER_URL** should be set like the following example:

```
https://<IP_CURRENT_HOST>:2380
```

The correct IP can be found from the **<ETCD_NAME>** of the specific control plane host, with the command:

```
$ echo <ETCD_NAME> | \
  sed -E 's/[.-]/_/g' | \
  xargs -I {} grep {} /etc/kubernetes/static-pod-resources/etcd-certs/configmaps/etcd-scripts/etcd.env | \
  grep "IP" | grep -Po '(?<=").+(?=")'
```

d. The value for **<ETCD_INITIAL_CLUSTER>** should be set like the following, where **<ETCD_NAME_n>** is the **<ETCD_NAME>** of each control plane host.

NOTE

The port used must be 2380 and not 2379. The port 2379 is used for etcd database management and is configured directly in etcd start command in container.

Example output

```
<ETCD_NAME_0>=<ETCD_NODE_PEER_URL_0>,<ETCD_NAME_1>=<ETCD_NODE_PEER_URL_1>,<ETCD_NAME_2>=<ETCD_NODE_PEER_URL_2>
```
(1)

(1) Specifies the **ETCD_NODE_PEER_URL** values from each control plane host.

The **<ETCD_INITIAL_CLUSTER>** value remains same across all control plane hosts. The same value is required in the next steps on every control plane host.

6. Regenerate the etcd database from the backup.

Such operation must be executed on each control plane host.

a. Copy the **etcd** backup to **/var/lib/etcd** directory with the command:

```
$ cp /home/core/assets/backup/<snapshot_yyyy-mm-dd_hhmmss>.db /var/lib/etcd
```

b. Identify the correct **etcdctl** image before proceeding. Use the following command to retrieve the image from the backup of the pod manifest:

```
$ jq -r '.spec.containers[]|select(.name=="etcdctl")|.image' /root/manifests-backup/etcd-pod.yaml
```

```
$ podman run --rm -it --entrypoint="/bin/bash" -v /var/lib/etcd:/var/lib/etcd:z <image-hash>
```

c. Check that the version of the **etcdctl** tool is the version of the **etcd** server where the backup was created:

```
$ etcdctl version
```

d. Run the following command to regenerate the **etcd** database, using the correct values for the current host:

```
$ ETCDCTL_API=3 /usr/bin/etcdctl snapshot restore /var/lib/etcd/<snapshot_yyyy-mm-dd_hhmmss>.db \
  --name "<ETCD_NAME>" \
  --initial-cluster="<ETCD_INITIAL_CLUSTER>" \
  --initial-cluster-token "openshift-etcd-<UUID>" \
  --initial-advertise-peer-urls "<ETCD_NODE_PEER_URL>" \
  --data-dir="/var/lib/etcd/restore-<UUID>" \
  --skip-hash-check=true
```

NOTE

The quotes are mandatory when regenerating the **etcd** database.

7. Record the values printed in the **added member** logs; for example:

Example output

```
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "56cd73b614699e7", "added-peer-peer-urls": ["https://10.0.91.5:2380"], "added-peer-is-learner": false}
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "1f63d01b31bb9a9e", "added-peer-peer-urls": ["https://10.0.90.221:2380"], "added-peer-is-learner": false}
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "fdc2725b3b70127c", "added-peer-peer-urls": ["https://10.0.94.214:2380"], "added-peer-is-learner": false}
```

a. Exit from the container.

b. Repeat these steps on the other control plane hosts, checking that the values printed in the **added member** logs are the same for all control plane hosts.

8. Move the regenerated **etcd** database to the default location.
Such operation must be executed on each control plane host.

a. Move the regenerated database (the **member** folder created by the previous **etcdctl snapshot restore** command) to the default etcd location **/var/lib/etcd**:

```
$ mv /var/lib/etcd/restore-<UUID>/member /var/lib/etcd
```

b. Restore the SELinux context for **/var/lib/etcd/member** folder on **/var/lib/etcd** directory:

```
$ restorecon -vR /var/lib/etcd/
```

c. Remove the leftover files and directories:

```
$ rm -rf /var/lib/etcd/restore-<UUID>
```

```
$ rm /var/lib/etcd/<snapshot_yyyy-mm-dd_hhmmss>.db
```

IMPORTANT

When you are finished the **/var/lib/etcd** directory must contain only the folder **member**.

d. Repeat these steps on the other control plane hosts.

9. Restart the etcd cluster.

a. The following steps must be executed on all control plane hosts, but **one host at a time**.

b. Move the **etcd** static pod manifest back to the kubelet manifest directory, in order to make kubelet start the related containers :

```
$ mv /root/manifests-backup/etcd-pod.yaml /etc/kubernetes/manifests
```

c. Verify that all the **etcd** containers have started:

```
$ crictl ps | grep etcd | grep -v operator
```

Example output

```
38c814767ad983
f79db5a8799fd2c08960ad9ee22f784b9fbe23babe008e8a3bf68323f004c840
28 seconds ago       Running             etcd-health-monitor                 2
fe4b9c3d6483c
e1646b15207c6
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago   Running             etcd-metrics                        0
fe4b9c3d6483c
08ba29b1f58a7
```

```
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago   Running             etcd                                    0
fe4b9c3d6483c
2ddc9eda16f53
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago   Running             etcdctl
```

If the output of this command is empty, wait a few minutes and check again.

10. Check the status of the **etcd** cluster.

    a. On any of the control plane hosts, check the status of the **etcd** cluster with the following command:

    ```
    $ crictl exec -it $(crictl ps | grep etcdctl | awk '{print $1}') etcdctl endpoint status -w table
    ```

    Example output

    ```
    +--------------------------+------------------+---------+---------+-----------+------------+-----------
    +------------+-------------------+--------+
    |        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
    RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
    +--------------------------+------------------+---------+---------+-----------+------------+-----------
    +------------+-------------------+--------+
    | https://10.0.89.133:2379 | 682e4a83a0cec6c0 |   3.5.0 |   67 MB |      true |      false |
    2 |        218 |               218 |        |
    |  https://10.0.92.74:2379 | 450bcf6999538512 |   3.5.0 |   67 MB |     false |      false |
    2 |        218 |               218 |        |
    | https://10.0.93.129:2379 | 358efa9c1d91c3d6 |   3.5.0 |   67 MB |     false |      false |
    2 |        218 |               218 |        |
    +--------------------------+------------------+---------+---------+-----------+------------+-----------
    +------------+-------------------+--------+
    ```

11. Restart the other static pods.
    The following steps must be executed on all control plane hosts, but one host at a time.

    a. Move the Kubernetes API Server static pod manifest back to the kubelet manifest directory to make kubelet start the related containers with the command:

    ```
    $ mv /root/manifests-backup/kube-apiserver-pod.yaml /etc/kubernetes/manifests
    ```

    b. Verify that all the Kubernetes API Server containers have started:

    ```
    $ crictl ps | grep kube-apiserver | grep -v operator
    ```

    NOTE

    if the output of the following command is empty, wait a few minutes and check again.

    c. Repeat the same steps for **kube-controller-manager-pod.yaml** and **kube-scheduler-pod.yaml** files.

        i. Restart the kubelets in all nodes using the following command:

```
$ systemctl restart kubelet
```

ii. Start the remaining control plane pods using the following command:

```
$ mv /root/manifests-backup/kube-* /etc/kubernetes/manifests/
```

iii. Check if the **kube-apiserver**, **kube-scheduler** and **kube-controller-manager** pods start correctly:

```
$ crictl ps | grep -E 'kube-(apiserver|scheduler|controller-manager)' | grep -v -E
'operator|guard'
```

iv. Wipe the OVN databases using the following commands:

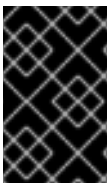
```
for NODE in  $(oc get node -o name | sed 's:node/::g')
do
  oc debug node/${NODE} -- chroot /host /bin/bash -c  'rm -f /var/lib/ovn-
ic/etc/ovn*.db && systemctl restart ovs-vswitchd ovsdb-server'
  oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-node --field-
selector=spec.nodeName=${NODE} --wait
  oc -n openshift-ovn-kubernetes wait pod -l app=ovnkube-node --field-
selector=spec.nodeName=${NODE} --for condition=ContainersReady --
timeout=600s
done
```

Additional resources

- Backing up etcd data
- Installing a user-provisioned cluster on bare metal
- Accessing hosts on Amazon Web Services in an installer-provisioned infrastructure cluster
- Replacing a bare-metal control plane node

4.3.2.5. Issues and workarounds for restoring a persistent storage state

If your OpenShift Container Platform cluster uses persistent storage of any form, a state of the cluster is typically stored outside etcd. When you restore from an etcd backup, the status of the workloads in OpenShift Container Platform is also restored. However, if the etcd snapshot is old, the status might be invalid or outdated.



IMPORTANT

The contents of persistent volumes (PVs) are never part of the etcd snapshot. When you restore an OpenShift Container Platform cluster from an etcd snapshot, non-critical workloads might gain access to critical data, or vice-versa.

The following are some example scenarios that produce an out-of-date status:

- MySQL database is running in a pod backed up by a PV object. Restoring OpenShift Container Platform from an etcd snapshot does not bring back the volume on the storage provider, and

does not produce a running MySQL pod, despite the pod repeatedly attempting to start. You must manually restore this pod by restoring the volume on the storage provider, and then editing the PV to point to the new volume.

- Pod P1 is using volume A, which is attached to node X. If the etcd snapshot is taken while another pod uses the same volume on node Y, then when the etcd restore is performed, pod P1 might not be able to start correctly due to the volume still being attached to node Y. OpenShift Container Platform is not aware of the attachment, and does not automatically detach it. When this occurs, the volume must be manually detached from node Y so that the volume can attach on node X, and then pod P1 can start.

- Cloud provider or storage provider credentials were updated after the etcd snapshot was taken. This causes any CSI drivers or Operators that depend on the those credentials to not work. You might have to manually update the credentials required by those drivers or Operators.

- A device is removed or renamed from OpenShift Container Platform nodes after the etcd snapshot is taken. The Local Storage Operator creates symlinks for each PV that it manages from **/dev/disk/by-id** or **/dev** directories. This situation might cause the local PVs to refer to devices that no longer exist.
  To fix this problem, an administrator must:

  1. Manually remove the PVs with invalid devices.

  2. Remove symlinks from respective nodes.

  3. Delete **LocalVolume** or **LocalVolumeSet** objects (see *Storage → Configuring persistent storage → Persistent storage using local volumes → Deleting the Local Storage Operator Resources*).

4.3.3. Recovering from expired control plane certificates

The cluster can automatically recover from expired control plane certificates.

However, you must manually approve the pending **node-bootstrapper** certificate signing requests (CSRs) to recover kubelet certificates. For user-provisioned installations, you might also need to approve pending kubelet serving CSRs.

Use the following steps to approve the pending CSRs:

Procedure

1. Get the list of current CSRs:

```
$ oc get csr
```

Example output

```
NAME        AGE    SIGNERNAME                                    REQUESTOR
CONDITION
csr-2s94x   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
Pending 1
csr-4bd6t   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
Pending
csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending 2
```

```
csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
...
```

1. A pending kubelet service CSR (for user-provisioned installations).
2. A pending **node-bootstrapper** CSR.

2. Review the details of a CSR to verify that it is valid:

```
$ oc describe csr <csr_name> 1
```

1. **<csr_name>** is the name of a CSR from the list of current CSRs.

3. Approve each valid **node-bootstrapper** CSR:

```
$ oc adm certificate approve <csr_name>
```

4. For user-provisioned installations, approve each valid kubelet serving CSR:

```
$ oc adm certificate approve <csr_name>
```

4.3.4. Testing restore procedures

Testing the restore procedure is important to ensure that your automation and workload handle the new cluster state gracefully. Due to the complex nature of etcd quorum and the etcd Operator attempting to mend automatically, it is often difficult to correctly bring your cluster into a broken enough state that it can be restored.

WARNING

You **must** have SSH access to the cluster. Your cluster might be entirely lost without SSH access.

Prerequisites

- You have SSH access to control plane hosts.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Use SSH to connect to each of your nonrecovery nodes and run the following commands to disable etcd and the **kubelet** service:

   a. Disable etcd by running the following command:

```
$ sudo /usr/local/bin/disable-etcd.sh
```

b. Delete variable data for etcd by running the following command:

```
$ sudo rm -rf /var/lib/etcd
```

c. Disable the **kubelet** service by running the following command:

```
$ sudo systemctl disable kubelet.service
```

2. Exit every SSH session.
3. Run the following command to ensure that your nonrecovery nodes are in a **NOT READY** state:

```
$ oc get nodes
```

4. Follow the steps in "Restoring to a previous cluster state" to restore your cluster.
5. After you restore the cluster and the API responds, use SSH to connect to each nonrecovery node and enable the **kubelet** service:

```
$ sudo systemctl enable kubelet.service
```

6. Exit every SSH session.
7. Run the following command to observe your nodes coming back into the **READY** state:

```
$ oc get nodes
```

8. Run the following command to verify that etcd is available:

```
$ oc get pods -n openshift-etcd
```

Additional resources

- Restoring to a previous cluster state

CHAPTER 5. ENABLING ETCD ENCRYPTION

5.1. ABOUT ETCD ENCRYPTION

By default, etcd data is not encrypted in OpenShift Container Platform. You can enable etcd encryption for your cluster to provide an additional layer of data security. For example, it can help protect the loss of sensitive data if an etcd backup is exposed to the incorrect parties.

When you enable etcd encryption, the following OpenShift API server and Kubernetes API server resources are encrypted:

- Secrets
- Config maps
- Routes
- OAuth access tokens
- OAuth authorize tokens

When you enable etcd encryption, encryption keys are created. You must have these keys to restore from an etcd backup.

NOTE

Etcd encryption only encrypts values, not keys. Resource types, namespaces, and object names are unencrypted.

If etcd encryption is enabled during a backup, the ***static_kuberesources_<datetimestamp>.tar.gz*** file contains the encryption keys for the etcd snapshot. For security reasons, store this file separately from the etcd snapshot. However, this file is required to restore a previous state of etcd from the respective etcd snapshot.

5.2. SUPPORTED ENCRYPTION TYPES

The following encryption types are supported for encrypting etcd data in OpenShift Container Platform:

AES-CBC

Uses AES-CBC with PKCS#7 padding and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

AES-GCM

Uses AES-GCM with a random nonce and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

5.3. ENABLING ETCD ENCRYPTION

You can enable etcd encryption to encrypt sensitive resources in your cluster.

WARNING

Do not back up etcd resources until the initial encryption process is completed. If the encryption process is not completed, the backup might be only partially encrypted.

After you enable etcd encryption, several changes can occur:

- The etcd encryption might affect the memory consumption of a few resources.
- You might notice a transient affect on backup performance because the leader must serve the backup.
- A disk I/O can affect the node that receives the backup state.

You can encrypt the etcd database in either AES-GCM or AES-CBC encryption.

NOTE

To migrate your etcd database from one encryption type to the other, you can modify the API server's **spec.encryption.type** field. Migration of the etcd data to the new encryption type occurs automatically.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

```
$ oc edit apiserver
```

2. Set the **spec.encryption.type** field to **aesgcm** or **aescbc**:

```
spec:
  encryption:
    type: aesgcm 1
```

1 Set to **aesgcm** for AES-GCM encryption or **aescbc** for AES-CBC encryption.

3. Save the file to apply the changes.
The encryption process starts. It can take 20 minutes or longer for this process to complete, depending on the size of the etcd database.

4. Verify that etcd encryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its

resources were successfully encrypted:

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its resources were successfully encrypted:

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully encrypted:

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

5.4. DISABLING ETCD ENCRYPTION

You can disable encryption of etcd data in your cluster.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

```
$ oc edit apiserver
```

2. Set the **encryption** field type to **identity**:

```
spec:
  encryption:
    type: identity (1)
```

(1) The **identity** type is the default value and means that no encryption is performed.

3. Save the file to apply the changes.
The decryption process starts. It can take 20 minutes or longer for this process to complete, depending on the size of your cluster.

4. Verify that etcd decryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its resources were successfully decrypted:

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

   b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its resources were successfully decrypted:

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

   c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully decrypted:

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

CHAPTER 6. GUIDANCE FOR CLUSTERS THAT SPAN DATA CENTERS

Red Hat strongly recommends a deployment model where OpenShift Container Platform clusters are deployed within a data center, but also acknowledges that there can be scenarios where a provider can use a deployment model where a cluster can span across data centers. This document outlines considerations when exploring the use of cluster deployments that span many data centers and describes important metrics that affect the supportability of such deployments. The design of such deployments should adhere to these guidelines for the product to function optimally and ensure the highest quality of support with the appropriate product support subscriptions.

WARNING

A cluster deployment that spans many data centers extends the cluster as a single failure domain across locations and should not be considered a replacement for a disaster recovery plan.

Clusters with cluster deployments that span many data centers are bound by standard Red Hat OpenShift Container Platform support guidance. See the Red Hat OpenShift Container Platform Lifecycle and Red Hat Production Support Scope of Coverage for more information.

It is not recommended to deploy an OpenShift Container Platform cluster that spans many sites. If you need to be in many data centers or regions, deploy one cluster per region or site and use tools such as Red Hat Advanced Cluster Management for Kubernetes (ACM) to manage these clusters and deployments.

Some OpenShift Container Platform platforms have specific support for many data center deployments. Check the platform-specific product documentation and release notes for details. Other platforms can span data centers, depending on the quality of the network connectivity between nodes. For more information, see Understanding etcd and the tunables/conditions affecting performance .

When implementing a cluster deployment that spans many data centers, you should strive to implement the practices detailed in Red Hat OpenShift Container Platform High Availability, and Recommended Practices. An alternative to multisite deployments is to deploy one OpenShift Container Platform cluster per site, managed by ACM.

6.1. DEPLOYMENT CAVEATS FOR SPANNED CLUSTERS

The guidance provided in this documentation focuses on general aspects of a cluster deployment that spans data centers. Some caveats to remember:

- Although the designs for deployments that span data centers are not bound by any special support requirements, these clusters do have additional inherent complexities that can require additional consideration or support involvement (time to identify, remediate and resolve issues) when compared to a standard single-site cluster.
- Applications might not work well or not work at all in clusters with high Kube API latency or low transaction rates.

- Layered products, such as storage providers, have lower latency requirements. In those cases, the latency limits are dictated by the architectures that are supported by the layered product.
- The failure scenarios are amplified with stretched control planes, and the way they are affected is specific to the deployment. Because of this, before using a deployment that spans data centers on a production environment, the organization should test and document the behavior of the cluster during disruptions such as:
    - When there is a network partition leaving one, two, or all control plane nodes isolated
    - When there are MTU mismatches on the transport network among the control plane nodes
    - When there is a sustained spike in latency as a Day 2 event towards one or more of the control plane nodes
    - When there is a considerable change in jitter due to network congestion, misconfiguration, or lack of QoS, an intermediate network device causing packet errors, and others
- Clusters deployed across many sites, network infrastructures, storage infrastructures, or other components inherently have a higher number of points of failure. Network disruptions or splits become a larger threat to such clusters especially, putting the nodes at risk of losing contact with each other. These multisite clusters must be designed with the potential for such failures in mind. Organizations deploying multisite clusters should extensively test failure scenarios, and should consider whether the cluster has protection from all points of failure. Consult with Red Hat Support for assistance in considering the important aspects of a resilient High Availability cluster design.
- In some cases, GEO awareness is a requirement or issue that must be solved to minimize latency, so a proper implementation of a Global Service Load Balancing (GSLB) method must be available.

6.2. INFRASTRUCTURE AS A SERVICE (IAAS) AND CLOUD PROVIDER CONSIDERATIONS

This guidance applies to any infrastructure provider for which OpenShift Container Platform control plane nodes are supported by the user-provisioned infrastructure installer (platform=none) or the agent-based installer (platform=metal) using the "User Managed Network" option. Installer-provisioned infrastructure installers are not covered by these guidelines, however, where possible, installer-provisioned infrastructure deployments will span zones or availability zones on cloud or IaaS providers if possible by following these or similar guidelines. This means infrastructure provider-specific integrations will not be available (for example, integration with Cloud provider services such as storage services and load balancers). Provider-specific services might still be used as external services.

Using different infrastructure platform providers for control plane nodes is discouraged (for example, mixing nodes across IaaS, cloud, and bare metal as control plane nodes). Consider the following guidelines when such combinations are needed:

- The minimum effective MTU across the infrastructure should be the maximum MTU used for the deployment. Using a lower MTU is acceptable. See *Understanding and Validating MTU setting with OpenShift Container Platform 4.x* for more information.
- The combined disk and network latency and jitter must maintain an etcd peer round trip time of less than 100ms. This is not the same as the network round trip time.
- Layered products might have lower latency requirements. In those cases, the latency limits are dictated by the requirements of the architecture supported by the layered product. For

example, OpenShift Container Platform cluster deployments that span data centers with Red Hat OpenShift Data Foundation must have a latency requirement of less than 10ms RTT. For those cases, follow the specific product guidance.

- For guidance on cluster deployments that span data centers using OpenShift Data Foundation as the storage provider, see *Configuring OpenShift Data Foundation Disaster Recovery for OpenShift Workloads*.

Additional resources

- Understanding and Validating MTU setting with OpenShift Container Platform 4.x
- Configuring OpenShift Data Foundation Disaster Recovery for OpenShift Workloads

6.3. SITE RECOMMENDATIONS

Assuming each site gets one control plane member, you theoretically define three sites, which is what Red Hat recommends. This allows for one data center to go into an inactive state and the cluster still maintains quorum and operational consistency.

When this assumption is not met, attention should be given to the desired and actual fault tolerance state of the cluster, as it will often outline or dictate the operational capabilities (uptime and stability) of the deployment.

6.4. REQUIREMENTS FOR ETCD, NETWORKING, AND STORAGE

Consider the following requirements for clusters that span data centers.

6.4.1. etcd requirements

There is a large list of factors and considerations that go into planning an etcd cluster deployment. When planning an OpenShift Container Platform cluster that spans data centers, you need to plan for situations that will likely stress or push etcd to the edge of its operational limits.

See *Understanding etcd and the tunables/conditions affecting performance* for more details on how to maintain operational capabilities and reduce service-affecting events and instability of the cluster.

6.4.2. Network requirements

The chosen network topology must yield direct IP connectivity between nodes. The minimum effective MTU across the infrastructure should be the maximum MTU used for the deployment. Using a lower MTU is acceptable.

For more information, see *Understanding and Validating MTU setting with OpenShift Container Platform 4.x*. The latency needs are ultimately defined by the services that use the network. See the sections related to etcd and storage for more details on requirements.

In addition to the base networking requirements, you need to think about how applications will be accessed. A top-level Global Service Load Balancing (GSLB) method will be needed outside of OpenShift Container Platform to enable external traffic to connect to the OpenShift Container Platform control plane services and ingress controllers.

6.4.3. Storage requirements

When considering a cluster deployment that spans data centers, special consideration needs to be given to the selected storage integration to ensure that it also meets multisite requirements, as it pertains to accessibility from all sites, fault tolerance, high availability, and so on.

An object storage solution should be used for the registry, and this storage solution needs to be in addition to any PV storage integration used for application volumes or workloads. This object storage solution should also have the same special considerations given to accessibility from all sites, fault tolerance, high availability, and so on.

Because disk I/O is a critical factor in the health of etcd database, it is required that they are deployed on a high speed, low latency media. See etcd guidance on *etcd peer round trip time* and *etcd database size* for more details on the exact requirements to meet.

Additional resources

- Understanding etcd and the tunables/conditions affecting performance
- Understanding and Validating MTU setting with OpenShift Container Platform 4.x
- etcd peer round trip time
- etcd database size

6.5. WORKLOAD PLACEMENT CONSIDERATIONS

With multisite clusters, administrators and developers must take special considerations into account to ensure that critical workloads are scheduled or placed based on the proper hardware or hosts within the topology of the cluster. This ensures that the applications and services are Highly Available and Fault Tolerant based on the topology of the cluster's deployment.

Without considering this, it is possible for OpenShift Container Platform to schedule workloads on hosts within the cluster so that a Single Point of Failure (SPoF) is created for OpenShift Container Platform infrastructure services and other application services if there is a data center outage.