OpenShift Container Platform 4.9

Backup and restore

Backing up and restoring your OpenShift Container Platform cluster

Last Updated: 2021-11-23

OpenShift Container Platform 4.9 Backup and restore

Backing up and restoring your OpenShift Container Platform cluster

Legal Notice

Abstract

This document provides instructions for backing up your cluster's data and for recovering from various disaster scenarios.

Table of Contents

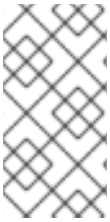
CHAPTER 1. BACKUP AND RESTORE

1.1. OVERVIEW OF BACKUP AND RESTORE OPERATIONS IN OPENSHIFT CONTAINER PLATFORM

As a cluster administrator, you might need to stop an OpenShift Container Platform cluster for a period and restart it later. Some reasons for restarting a cluster are that you need to perform maintenance on a cluster or want to reduce resource costs. In OpenShift Container Platform, you can perform a graceful shutdown of a cluster so that you can easily restart the cluster later.

You must back up etcd data before shutting down a cluster; etcd is the key-value store for OpenShift Container Platform, which persists the state of all resource objects. An etcd backup plays a crucial role in disaster recovery. In OpenShift Container Platform, you can also replace an unhealthy etcd member.

When you want to get your cluster running again, restart the cluster gracefully.



NOTE

A cluster's certificates expire one year after the installation date. You can shut down a cluster and expect it to restart gracefully while the certificates are still valid. Although the cluster automatically retrieves the expired control plane certificates, you must still approve the certificate signing requests (CSRs) .

You might run into several situations where OpenShift Container Platform does not work as expected, such as:

- You have a cluster that is not functional after the restart because of unexpected conditions, such as node failure, or network connectivity issues.
- You have deleted something critical in the cluster by mistake.
- You have lost the majority of your control plane hosts, leading to etcd quorum loss.

You can always recover from a disaster situation by restoring your cluster to its previous state using the saved etcd snapshots.

CHAPTER 2. SHUTTING DOWN THE CLUSTER GRACEFULLY

This document describes the process to gracefully shut down your cluster. You might need to temporarily shut down your cluster for maintenance reasons, or to save on resource costs.

2.1. PREREQUISITES

- Take an etcd backup prior to shutting down the cluster.

2.2. SHUTTING DOWN THE CLUSTER

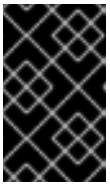
You can shut down your cluster in a graceful manner so that it can be restarted at a later date.

NOTE

You can shut down a cluster until a year from the installation date and expect it to restart gracefully. After a year from the installation date, the cluster certificates expire.

Prerequisites

- 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 when restarting the cluster.

Procedure

1. If you are shutting the cluster down for an extended period, determine the date on which certificates expire.

```
$ oc -n openshift-kube-apiserver-operator get secret kube-apiserver-to-kubelet-signer -o jsonpath='{.metadata.annotations.auth\.openshift\.io/certificate-not-after}'
```

Example output

```
2022-08-05T14:37:50Zuser@user:~ $ 1
```

1 To ensure that the cluster can restart gracefully, plan to restart it on or before the specified date. As the cluster restarts, the process might require you to manually approve the pending certificate signing requests (CSRs) to recover kubelet certificates.

2. Shut down all of the nodes in the cluster. You can do this from your cloud provider's web console, or run the following loop:

```
$ for node in $(oc get nodes -o jsonpath='{.items[*].metadata.name}'); do oc debug node/${node} -- chroot /host shutdown -h 1; done
```

Example output

```
Starting pod/ip-10-0-130-169us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`
Shutdown scheduled for Mon 2021-09-13 09:36:17 UTC, use 'shutdown -c' to cancel.

Removing debug pod ...
Starting pod/ip-10-0-150-116us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`
Shutdown scheduled for Mon 2021-09-13 09:36:29 UTC, use 'shutdown -c' to cancel.
```

Shutting down the nodes using one of these methods allows pods to terminate gracefully, which reduces the chance for data corruption.

NOTE

It is not necessary to drain control plane nodes of the standard pods that ship with OpenShift Container Platform prior to shutdown.

Cluster administrators are responsible for ensuring a clean restart of their own workloads after the cluster is restarted. If you drained control plane nodes prior to shutdown because of custom workloads, you must mark the control plane nodes as schedulable before the cluster will be functional again after restart.

3. Shut off any cluster dependencies that are no longer needed, such as external storage or an LDAP server. Be sure to consult your vendor's documentation before doing so.

Additional resources

- Restarting the cluster gracefully

CHAPTER 3. RESTARTING THE CLUSTER GRACEFULLY

This document describes the process to restart your cluster after a graceful shutdown.

Even though the cluster is expected to be functional after the restart, the cluster might not recover due to unexpected conditions, for example:

- etcd data corruption during shutdown
- Node failure due to hardware
- Network connectivity issues

If your cluster fails to recover, follow the steps to restore to a previous cluster state .

3.1. PREREQUISITES

- You have gracefully shut down your cluster .

3.2. RESTARTING THE CLUSTER

You can restart your cluster after it has been shut down gracefully.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- This procedure assumes that you gracefully shut down the cluster.

Procedure

1. Power on any cluster dependencies, such as external storage or an LDAP server.
2. Start all cluster machines.
   Use the appropriate method for your cloud environment to start the machines, for example, from your cloud provider's web console.

   Wait approximately 10 minutes before continuing to check the status of control plane nodes.
3. Verify that all control plane nodes are ready.

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

   The control plane nodes are ready if the status is **Ready**, as shown in the following output:

   ```
   NAME                         STATUS   ROLES    AGE   VERSION
   ip-10-0-168-251.ec2.internal   Ready    master   75m   v1.22.1
   ip-10-0-170-223.ec2.internal   Ready    master   75m   v1.22.1
   ip-10-0-211-16.ec2.internal    Ready    master   75m   v1.22.1
   ```
4. If the control plane nodes are *not* ready, then check whether there are any pending certificate signing requests (CSRs) that must be approved.

   a. Get the list of current CSRs:

```
$ oc get csr
```

b. 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.

c. Approve each valid CSR:

```
$ oc adm certificate approve <csr_name>
```

5. After the control plane nodes are ready, verify that all worker nodes are ready.

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

The worker nodes are ready if the status is **Ready**, as shown in the following output:

```
NAME                         STATUS   ROLES    AGE   VERSION
ip-10-0-179-95.ec2.internal  Ready    worker   64m   v1.22.1
ip-10-0-182-134.ec2.internal Ready    worker   64m   v1.22.1
ip-10-0-250-100.ec2.internal Ready    worker   64m   v1.22.1
```

6. If the worker nodes are *not* ready, then check whether there are any pending certificate signing requests (CSRs) that must be approved.

a. Get the list of current CSRs:

```
$ oc get csr
```

b. 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.

c. Approve each valid CSR:

```
$ oc adm certificate approve <csr_name>
```

7. Verify that the cluster started properly.

a. Check that there are no degraded cluster Operators.

```
$ oc get clusteroperators
```

Check that there are no cluster Operators with the **DEGRADED** condition set to **True**.

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
SINCE
authentication                              4.9.0     True        False         False     59m
```

```
cloud-credential                           4.9.0     True        False         False      85m
cluster-autoscaler                         4.9.0     True        False         False      73m
config-operator                            4.9.0     True        False         False      73m
console                                    4.9.0     True        False         False      62m
csi-snapshot-controller                    4.9.0     True        False         False      66m
dns                                        4.9.0     True        False         False      76m
etcd                                       4.9.0     True        False         False      76m
...
```

b. Check that all nodes are in the **Ready** state:

```
$ oc get nodes
```

Check that the status for all nodes is **Ready**.

```
NAME                           STATUS   ROLES    AGE   VERSION
ip-10-0-168-251.ec2.internal   Ready    master   82m   v1.22.1
ip-10-0-170-223.ec2.internal   Ready    master   82m   v1.22.1
ip-10-0-179-95.ec2.internal    Ready    worker   70m   v1.22.1
ip-10-0-182-134.ec2.internal   Ready    worker   70m   v1.22.1
ip-10-0-211-16.ec2.internal    Ready    master   82m   v1.22.1
ip-10-0-250-100.ec2.internal   Ready    worker   69m   v1.22.1
```

If the cluster did not start properly, you might need to restore your cluster using an etcd backup.

Additional resources

- See Restoring to a previous cluster state for how to use an etcd backup to restore if your cluster failed to recover after restarting.

CHAPTER 4. CONTROL PLANE BACKUP AND RESTORE

4.1. BACKING UP ETCD

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

Back up your cluster's etcd data regularly and store 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, as it is a blocking action.

Be sure to take an etcd backup after you upgrade your cluster. This 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.y.z cluster must use an etcd backup that was taken from 4.y.z.

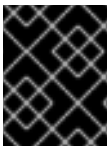
IMPORTANT

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 for a control plane node:

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

2. Change your root directory to the host:

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

3. If the cluster-wide proxy is enabled, be sure that you have exported the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables.

4. Run the **cluster-backup.sh** script 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

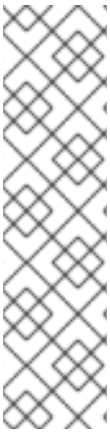
If etcd encryption is enabled, it is recommended to store this second file separately from the etcd snapshot for security reasons. However, this file is required to restore from the etcd snapshot.

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

4.2. REPLACING AN UNHEALTHY ETCD MEMBER

This document describes the process to replace a single unhealthy etcd member.

This process depends on whether the etcd member is unhealthy because the machine is not running or the node is not ready, or whether it is unhealthy because the etcd pod is crashlooping.



NOTE

If you have lost the majority of your control plane hosts, leading to etcd quorum loss, then you must 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. Prerequisites

- Take an etcd backup prior to replacing an unhealthy etcd member.

4.2.2. Identifying an unhealthy etcd member

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

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

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"}'
```

2. Review the output:

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

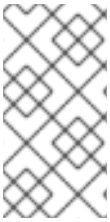
This example output shows that the **ip-10-0-131-183.ec2.internal** etcd member is unhealthy.

4.2.3. 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   v1.22.1 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   v1.22.1
ip-10-0-164-97.ec2.internal    Ready    master   6h13m   v1.22.1
ip-10-0-154-204.ec2.internal   Ready    master   6h13m   v1.22.1
```

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

```
$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep 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.

4.2.4. Replacing the unhealthy etcd member

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

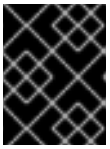
- Replacing an unhealthy etcd member whose machine is not running or whose node is not ready
- Replacing an unhealthy etcd member whose etcd pod is crashlooping

4.2.4.1. 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.

Prerequisites

- You have identified the unhealthy 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

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. 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 get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep 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.

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 baa565c8919b060e
```

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. 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
```

3. Delete and recreate the control plane machine. After this machine is recreated, 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 master 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.

It is recommended to 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. Update the **metadata.selfLink** field to use the new machine name from the previous step.

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  selfLink: /apis/machine.openshift.io/v1beta1/namespaces/openshift-machine-api/machines/clustername-8qw5l-master-3
  ...
```

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

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

v. Remove the **metadata.annotations** and **metadata.generation** fields:

```
  annotations:
    machine.openshift.io/instance-state: running
  ...
  generation: 2
```

vi. Remove the **metadata.resourceVersion** and **metadata.uid** fields:

```
  resourceVersion: "13291"
  uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

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 using the **new-master-machine.yaml** file:

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

g. Verify that the 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
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 will automatically sync when the machine or node returns to a healthy state.

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 get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep 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

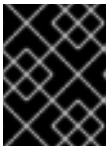
Be sure to remove the correct etcd member; removing a good etcd member might lead to quorum loss.

4.2.4.2. 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 the 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 get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep 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. 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
```

4. 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.

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 --cluster
```

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
```

4.3. DISASTER RECOVERY

4.3.1. About 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.

Restoring to a previous cluster state

This solution handles situations where you want to restore your cluster to a previous state, for example, if an administrator deletes something critical. This also includes situations where you have lost the majority of your control plane hosts, leading to etcd quorum loss and the cluster going offline. As long as you have taken an etcd backup, you can follow this procedure to restore your cluster to a previous state.
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.

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

NOTE

If you have a majority of your masters still available and have an etcd quorum, then follow the procedure to replace a single unhealthy etcd member.

Recovering from expired control plane certificates

This solution handles situations where your control plane certificates have expired. For example, if you shut down your cluster before the first certificate rotation, which occurs 24 hours after installation, your certificates will not be rotated and will expire. You can follow this procedure to recover from expired control plane certificates.

4.3.2. Restoring to a previous cluster state

To restore the cluster to a previous state, you must have previously backed up etcd data by creating a snapshot. You will use this snapshot to restore the cluster state.

4.3.2.1. About restoring cluster state

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, SDN controllers, and persistent volume controllers.

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

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.

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 4.7.2 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.
- A healthy control plane host to use as the recovery host.
- 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. Select a control plane host to use as the recovery host. This is the host that you will run the restore operation on.
2. Establish SSH connectivity to each of the control plane nodes, including the recovery host.

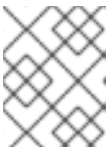
The Kubernetes API server becomes inaccessible after the restore process starts, so you cannot access the control plane nodes. For this reason, it is recommended to establish SSH connectivity to each control plane host in a separate terminal.

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. 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.

4. Stop the static pods on any other control plane nodes.



NOTE

It is not required to manually stop the pods on the recovery host. The recovery script will stop the pods on the recovery host.

a. Access a control plane host that is not the recovery host.

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

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

c. Verify that the etcd pods are stopped.

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

d. Move the existing Kubernetes API server pod file out of the kubelet manifest directory:

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml
/tmp
```

e. Verify that the Kubernetes API server pods are stopped.

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep kube-apiserver | grep -v operator
```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

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

```
[core@ip-10-0-154-194 ~]$ sudo mv /var/lib/etcd/ /tmp
```

g. Repeat this step on each of the other control plane hosts that is not the recovery host.

5. Access the recovery control plane host.

6. If the cluster-wide proxy is enabled, be sure that you have exported the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables.

   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.

7. Run the restore script on the recovery control plane host and pass in the path to the etcd backup directory:

   ```
   [core@ip-10-0-143-125 ~]$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/backup
   ```

   Example script output

   ```
   ...stopping kube-scheduler-pod.yaml
   ...stopping kube-controller-manager-pod.yaml
   ...stopping etcd-pod.yaml
   ...stopping kube-apiserver-pod.yaml
   Waiting for container etcd to stop
   .complete
   Waiting for container etcdctl to stop
   .....complete
   Waiting for container etcd-metrics to stop
   complete
   Waiting for container kube-controller-manager to stop
   complete
   Waiting for container kube-apiserver to stop
   .....complete
   Waiting for container kube-scheduler to stop
   complete
   Moving etcd data-dir /var/lib/etcd/member to /var/lib/etcd-backup
   starting restore-etcd static pod
   starting kube-apiserver-pod.yaml
   static-pod-resources/kube-apiserver-pod-7/kube-apiserver-pod.yaml
   starting kube-controller-manager-pod.yaml
   static-pod-resources/kube-controller-manager-pod-7/kube-controller-manager-pod.yaml
   starting kube-scheduler-pod.yaml
   static-pod-resources/kube-scheduler-pod-8/kube-scheduler-pod.yaml
   ```

8. Restart the kubelet service on all control plane hosts.

   a. From the recovery host, run the following command:

      ```
      [core@ip-10-0-143-125 ~]$ sudo systemctl restart kubelet.service
      ```

   b. Repeat this step on all other control plane hosts.

9. Approve the pending CSRs:

   a. 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 (2)
csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
(3)
csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
(4)
...
```

(1) (2) A pending kubelet service CSR (for user-provisioned installations).

(3) (4) A pending **node-bootstrapper** CSR.

b. 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.

c. Approve each valid **node-bootstrapper** CSR:

```
$ oc adm certificate approve <csr_name>
```

d. For user-provisioned installations, approve each valid kubelet service CSR:

```
$ oc adm certificate approve <csr_name>
```

10. Verify that the single member control plane has started successfully.

a. From the recovery host, verify that the etcd container is running.

```
[core@ip-10-0-143-125 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

Example output

```
3ad41b7908e32
36f86e2eeaaffe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running             etcd                                0
7c05f8af362f0
```

b. From the recovery host, verify that the etcd pod is running.

```
[core@ip-10-0-143-125 ~]$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard |
grep etcd
```

NOTE

If you attempt to run **oc login** prior to running this command and receive the following error, wait a few moments for the authentication controllers to start and try again.

```
Unable to connect to the server: EOF
```

Example output

```
NAME                                READY   STATUS      RESTARTS   AGE
etcd-ip-10-0-143-125.ec2.internal   1/1     Running     1          2m47s
```

If the status is **Pending**, or the output lists more than one running etcd pod, wait a few minutes and check again.

c. Repeat this step for each lost control plane host that is not the recovery host.

11. Delete and recreate other non-recovery, control plane machines, one by one. After these machines are recreated, 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 master node using the same method that was used to originally create it.

WARNING

Do not delete and recreate the machine for the recovery host.

a. Obtain the machine for one of the lost control plane hosts.
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 lost control plane host, **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 lost control plane host.

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.

It is recommended to 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**:

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

iii. Update the **metadata.selfLink** field to use the new machine name from the previous step:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  selfLink: /apis/machine.openshift.io/v1beta1/namespaces/openshift-machine-api/machines/clustername-8qw5l-master-3
  ...
```

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

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

v. Remove the **metadata.annotations** and **metadata.generation** fields:

```
annotations:
  machine.openshift.io/instance-state: running
...
generation: 2
```

vi. Remove the **metadata.resourceVersion** and **metadata.uid** fields:

```
resourceVersion: "13291"
uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

d. Delete the machine of the lost control plane host:

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

1 Specify the name of the control plane machine for the lost control plane host.

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-10-0-143-125.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
```

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

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

g. Verify that the 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
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 sync when the machine or node returns to a healthy state.

h. Repeat these steps for each lost control plane host that is not the recovery host.

12. In a separate terminal window, log in to the cluster as a user with the **cluster-admin** role by using the following command:

```
$ oc login -u <cluster_admin> (1)
```

-

1 For **<cluster_admin>**, specify a user name with the **cluster-admin** role.

13. 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": "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, the existing nodes are started with new pods similar to the initial bootstrap scale up.

14. Verify all nodes are updated to the latest revision.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

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

Review the **NodeInstallerProgressing** status condition for etcd to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1 In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

15. After etcd is redeployed, force new rollouts for the control plane. The Kubernetes API server will reinstall itself on the other nodes because the kubelet is connected to API servers using an internal load balancer.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following commands.

a. Force a new rollout for the Kubernetes API server:

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

Verify all nodes are updated to the latest revision.

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

Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

-

```
AllNodesAtLatestRevision
3 nodes are at revision 7 (1)
```

(1) In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

b. Force a new rollout for the Kubernetes controller manager:

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

Verify all nodes are updated to the latest revision.

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

Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 (1)
```

(1) In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

c. Force a new rollout for the Kubernetes scheduler:

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

Verify all nodes are updated to the latest revision.

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

Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 (1)
```

(1) In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

16. Verify that all control plane hosts have started and joined the cluster.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep etcd
```

Example output

```
etcd-ip-10-0-143-125.ec2.internal        2/2     Running     0          9h
etcd-ip-10-0-154-194.ec2.internal        2/2     Running     0          9h
etcd-ip-10-0-173-171.ec2.internal        2/2     Running     0          9h
```

Note that it might take several minutes after completing this procedure for all services to be restored. For example, authentication by using **oc login** might not immediately work until the OAuth server pods are restarted.

4.3.2.3. 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. It might be an Elasticsearch cluster running in a pod or a database running in a **StatefulSet** object. 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

4.3.3.1. 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 2
csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending 3
csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending 4
...
```

1 2 A pending kubelet service CSR (for user-provisioned installations).

3 4 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>
```