OpenShift Container Platform 4.4

Backup and restore

Backing up and restoring your OpenShift Container Platform cluster

Last Updated: 2021-02-14

OpenShift Container Platform 4.4 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. 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 4.4.2 cluster must use an etcd backup that was taken from 4.4.2.

IMPORTANT

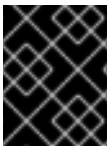
Back up your cluster's etcd data by performing a single invocation of the backup script on a master host. Do not take a backup for each master host.

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

You can perform the etcd data backup process on any master host that has a running etcd instance.

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 master host. Do not take a backup from each master 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 master node:

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

2. Change your root directory to the host:

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

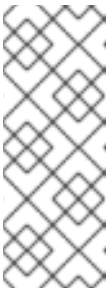
3. If the cluster-wide proxy is enabled, be sure that you have exported the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables.

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

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
1bf371f1b5a483927cd01bb593b0e12cff406eb8d7d0acf4ab079c36a0abd3f7
etcdctl version: 3.3.18
API version: 3.3
found latest kube-apiserver-pod: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-7
found latest kube-controller-manager-pod: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-8
found latest kube-scheduler-pod: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd-pod: /etc/kubernetes/static-pod-resources/etcd-pod-2
Snapshot saved at /home/core/assets/backup/snapshot_2020-03-18_220218.db
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 master host:

- **snapshot_<datetimestamp>.db**: This file is the etcd snapshot.
- **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 in order 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.

CHAPTER 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 master 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 master 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.

2.1. PREREQUISITES

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

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.

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
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
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"
ip-10-0-131-183.ec2.internal   NotReady   master   122m   v1.17.1 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 master nodes are listed as **Ready**:

```
$ oc get nodes -l node-role.kubernetes.io/master
NAME                          STATUS   ROLES    AGE     VERSION
ip-10-0-131-183.ec2.internal   Ready    master   6h13m   v1.17.1
ip-10-0-164-97.ec2.internal    Ready    master   6h13m   v1.17.1
ip-10-0-154-204.ec2.internal   Ready    master   6h13m   v1.17.1
```

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

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

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

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 etcd
   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

   +------------------+---------+------------------------------+---------------------------+---------------
   -----------+
   |        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
   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

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

```
|  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:

   ```
   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 master 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

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 master 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 master machine for the unhealthy node.

c. Edit the **new-master-machine.yaml** file that was created in the previous step.

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. Remove the **providerID** field:

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

iii. Change the **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
  ...
```

iv. Update the **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
  ...
```

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 master machine for the unhealthy node.

e. Verify that the machine was deleted:

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

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

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.

4. 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 etcd
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.

2.4.2. 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 etcd
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

+------------------+---------+-----------------------------+---------------------------+---------------
-----------+
|        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
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

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

```
|        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:

```
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 master nodes have a functioning etcd pod.

5. 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
   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
   ```

CHAPTER 3. DISASTER RECOVERY

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 in order to return your cluster to a working state.

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 master 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 .

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.

3.2. RECOVERING FROM LOST MASTER HOSTS

As of OpenShift Container Platform 4.4, follow the procedure to restore to a previous cluster state in order to recover from lost master hosts.

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.

3.3. 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.

3.3.1. Restoring to a previous cluster state

You can use a saved etcd backup to restore back to a previous cluster state. You use the etcd backup to restore a single master host. Then the etcd cluster Operator handles scaling to the remaining master 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 4.4.2 cluster must use an etcd backup that was taken from 4.4.2.

Prerequisites

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

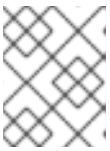
1. Select a master host to use as the recovery host. This is the host that you will run the restore operation on.
2. Establish SSH connectivity to each of the master nodes, including the recovery host.
   The Kubernetes API server will become inaccessible once the restore process has started, so you cannot access the master nodes. For this reason, it is recommended to establish SSH connectivity to each master host in a separate terminal.

   IMPORTANT

   If you do not complete this step, you will not be able to access the master 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 master 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 master host.
4. Stop the static pods on all other master 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 master 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 master hosts that is not the recovery host.

5. Access the recovery master 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 master 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
...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 master 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 master hosts.

9. 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
   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 etcd

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

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
```

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

10. 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.

11. 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 3
```

If the output shows a message such as **2 nodes are at revision 3; 1 nodes are at revision 4**, this means that the update is still in progress. Wait a few minutes and try again.

12. 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. Update the **kubeapiserver**:

```
$ 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 3
```

    b. Update the **kubecontrollermanager**:

```
$ 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 3
```

c. Update the **kubescheduler**:

```
$ 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 3
```

13. Verify that all master 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 etcd
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.

3.4. RECOVERING FROM EXPIRED CONTROL PLANE CERTIFICATES

3.4.1. Recovering from expired control plane certificates

As of OpenShift Container Platform 4.4.8, the cluster can automatically recover from expired control plane certificates. You no longer need to perform the manual steps that were required in previous versions.

The exception is that you must manually approve the pending **node-bootstrapper** certificate signing requests (CSRs) to recover kubelet certificates.

Use the following steps to approve the pending **node-bootstrapper** CSRs.

Procedure

1. Get the list of current CSRs:

```
$ oc get 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>
```