OpenShift Container Platform 4.13

Hosted control planes

Using hosted control planes with OpenShift Container Platform

Last Updated: 2024-04-22

OpenShift Container Platform 4.13 Hosted control planes

Using hosted control planes with OpenShift Container Platform

Legal Notice

Abstract

This document provides instructions for managing hosted control planes for OpenShift Container Platform. With hosted control planes, you create control planes as pods on a hosting cluster without the need for dedicated physical or virtual machines for each control plane.

Table of Contents

CHAPTER 1. HOSTED CONTROL PLANES OVERVIEW

You can deploy OpenShift Container Platform clusters by using two different control plane configurations: standalone or hosted control planes. The standalone configuration uses dedicated virtual machines or physical machines to host the control plane. With hosted control planes for OpenShift Container Platform, you create control planes as pods on a hosting cluster without the need for dedicated virtual or physical machines for each control plane.

1.1. INTRODUCTION TO HOSTED CONTROL PLANES (TECHNOLOGY PREVIEW)

You can use hosted control planes for Red Hat OpenShift Container Platform to reduce management costs, optimize cluster deployment time, and separate management and workload concerns so that you can focus on your applications.

You can enable hosted control planes as a Technology Preview feature by using the multicluster engine for Kubernetes operator version 2.0 or later on Amazon Web Services (AWS), bare metal by using the Agent provider, or OpenShift Virtualization.

IMPORTANT

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

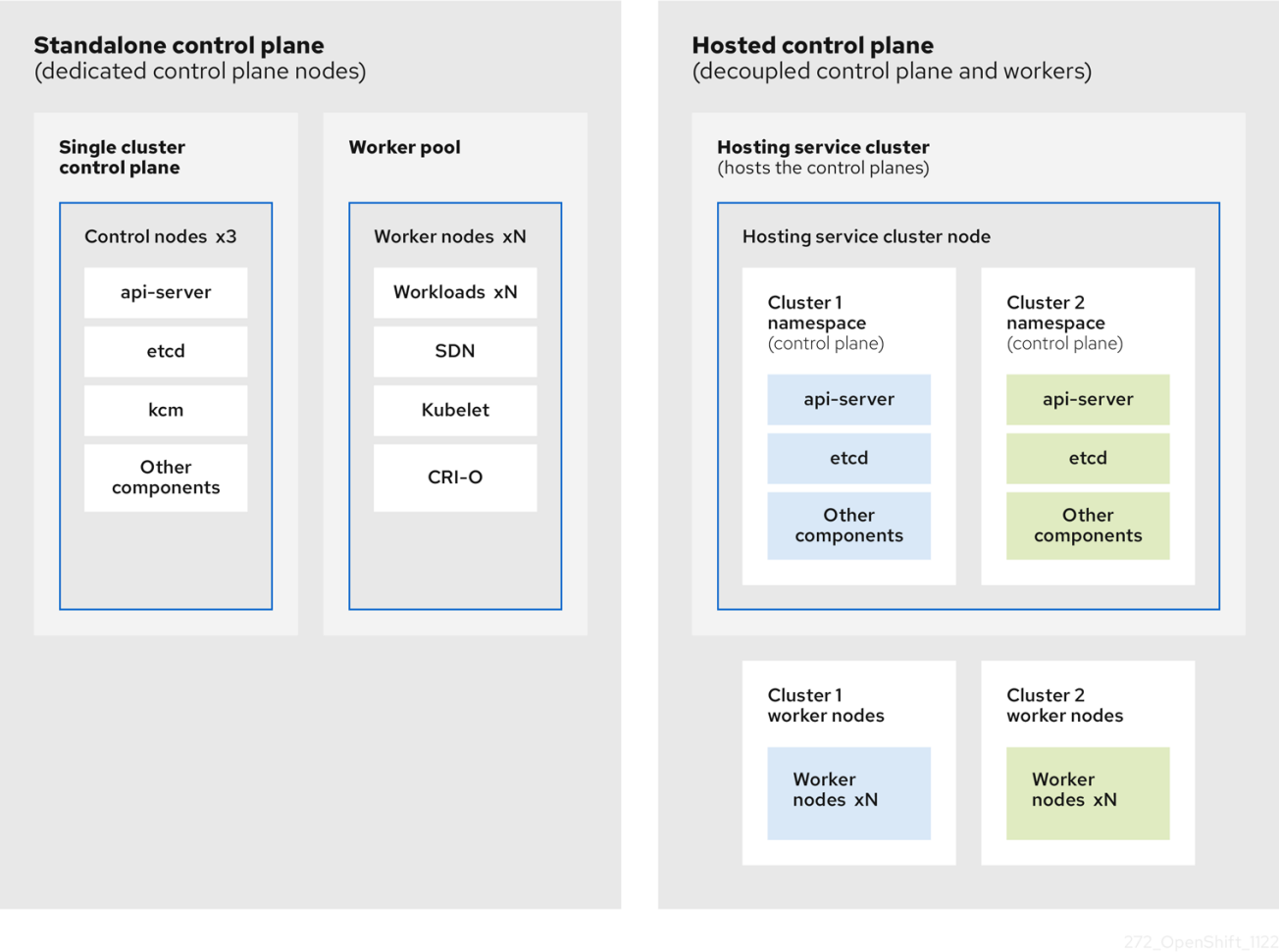
For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

1.1.1. Architecture of hosted control planes

OpenShift Container Platform is often deployed in a coupled, or standalone, model, where a cluster consists of a control plane and a data plane. The control plane includes an API endpoint, a storage endpoint, a workload scheduler, and an actuator that ensures state. The data plane includes compute, storage, and networking where workloads and applications run.

The standalone control plane is hosted by a dedicated group of nodes, which can be physical or virtual, with a minimum number to ensure quorum. The network stack is shared. Administrator access to a cluster offers visibility into the cluster's control plane, machine management APIs, and other components that contribute to the state of a cluster.

Although the standalone model works well, some situations require an architecture where the control plane and data plane are decoupled. In those cases, the data plane is on a separate network domain with a dedicated physical hosting environment. The control plane is hosted by using high-level primitives such as deployments and stateful sets that are native to Kubernetes. The control plane is treated as any other workload.



1.1.2. Benefits of hosted control planes

With hosted control planes for OpenShift Container Platform, you can pave the way for a true hybrid-cloud approach and enjoy several other benefits.

- The security boundaries between management and workloads are stronger because the control plane is decoupled and hosted on a dedicated hosting service cluster. As a result, you are less likely to leak credentials for clusters to other users. Because infrastructure secret account management is also decoupled, cluster infrastructure administrators cannot accidentally delete control plane infrastructure.
- With hosted control planes, you can run many control planes on fewer nodes. As a result, clusters are more affordable.
- Because the control planes consist of pods that are launched on OpenShift Container Platform, control planes start quickly. The same principles apply to control planes and workloads, such as monitoring, logging, and auto-scaling.
- From an infrastructure perspective, you can push registries, HAProxy, cluster monitoring, storage nodes, and other infrastructure components to the tenant's cloud provider account, isolating usage to the tenant.
- From an operational perspective, multicluster management is more centralized, which results in fewer external factors that affect the cluster status and consistency. Site reliability engineers have a central place to debug issues and navigate to the cluster data plane, which can lead to shorter Time to Resolution (TTR) and greater productivity.

Additional resources

- HyperShift add-on (Technology Preview)
- Hosted control planes (Technology Preview)

1.2. VERSIONING FOR HOSTED CONTROL PLANES

With each major, minor, or patch version release of OpenShift Container Platform, two components of hosted control planes are released:

- HyperShift Operator
- Command-line interface (CLI)

The HyperShift Operator manages the lifecycle of hosted clusters that are represented by **HostedCluster** API resources. The HyperShift Operator is released with each OpenShift Container Platform release. After the HyperShift Operator is installed, it creates a config map called **supported-versions** in the HyperShift namespace, as shown in the following example. The config map describes the HostedCluster versions that can be deployed.

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.13","4.12","4.11"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

The CLI is a helper utility for development purposes. The CLI is released as part of any HyperShift Operator release. No compatibility policies are guaranteed.

The API, **hypershift.openshift.io**, provides a way to create and manage lightweight, flexible, heterogeneous OpenShift Container Platform clusters at scale. The API exposes two user-facing resources: **HostedCluster** and **NodePool**. A **HostedCluster** resource encapsulates the control plane and common data plane configuration. When you create a **HostedCluster** resource, you have a fully functional control plane with no attached nodes. A **NodePool** resource is a scalable set of worker nodes that is attached to a **HostedCluster** resource.

The API version policy generally aligns with the policy for Kubernetes API versioning.

CHAPTER 2. CONFIGURING HOSTED CONTROL PLANES

To get started with hosted control planes for OpenShift Container Platform, you first configure your hosted cluster on the provider that you want to use. Then, you complete a few management tasks.

IMPORTANT

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

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

You can view the procedures by selecting from one of the following providers:

2.1. AMAZON WEB SERVICES (AWS)

- Configuring the hosting cluster on AWS (Technology Preview): The tasks to configure a hosted cluster on AWS include creating the AWS S3 OIDC secret, creating a routable public zone, enabling external DNS, enabling AWS PrivateLink, enabling the hosted control planes feature, and installing the hosted control planes CLI.
- Managing hosted control plane clusters on AWS (Technology Preview) : Management tasks include creating, importing, accessing, or deleting a hosted cluster on AWS.

2.2. BARE METAL

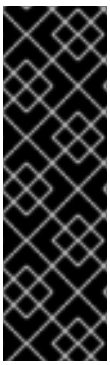
- Configuring the hosting cluster on bare metal (Technology Preview): Configure DNS before you create a hosted cluster.
- Managing hosted control plane clusters on bare metal (Technology Preview) : Create a hosted cluster, create an **InfraEnv** resource, add agents, access the hosted cluster, scale the **NodePool** object, handle Ingress, enable node auto-scaling, or delete a hosted cluster.

2.3. OPENSHIFT VIRTUALIZATION

- Managing hosted control plane clusters on OpenShift Virtualization (Technology Preview) : Create OpenShift Container Platform clusters with worker nodes that are hosted by KubeVirt virtual machines.

CHAPTER 3. MANAGING HOSTED CONTROL PLANES

After you configure your environment for hosted control planes and create a hosted cluster, you can further manage your clusters and nodes.



IMPORTANT

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

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

3.1. UPDATES FOR HOSTED CONTROL PLANES

Updates for hosted control planes involve updating the hosted cluster and the node pools. For a cluster to remain fully operational during an update process, you must meet the requirements of the Kubernetes version skew policy while completing the control plane and node updates.

3.1.1. Updates for the hosted cluster

The **spec.release** value dictates the version of the control plane. The **HostedCluster** object transmits the intended **spec.release** value to the **HostedControlPlane.spec.release** value and runs the appropriate Control Plane Operator version.

The hosted control plane manages the rollout of the new version of the control plane components along with any OpenShift Container Platform components through the new version of the Cluster Version Operator (CVO).

3.1.2. Updates for node pools

With node pools, you can configure the software that is running in the nodes by exposing the **spec.release** and **spec.config** values. You can start a rolling node pool update in the following ways:

- Changing the **spec.release** or **spec.config** values.
- Changing any platform-specific field, such as the AWS instance type. The result is a set of new instances with the new type.
- Changing the cluster configuration, if the change propagates to the node.

Node pools support replace updates and in-place updates. The **nodepool.spec.release** value dictates the version of any particular node pool. A **NodePool** object completes a replace or an in-place rolling update according to the **.spec.management.upgradeType** value.

After you create a node pool, you cannot change the update type. If you want to change the update type, you must create a node pool and delete the other one.

3.1.2.1. Replace updates for node pools

A *replace* update creates instances in the new version while it removes old instances from the previous version. This update type is effective in cloud environments where this level of immutability is cost effective.

Replace updates do not preserve any manual changes because the node is entirely re-provisioned.

3.1.2.2. In place updates for node pools

An *in-place* update directly updates the operating systems of the instances. This type is suitable for environments where the infrastructure constraints are higher, such as bare metal.

In-place updates can preserve manual changes, but will report errors if you make manual changes to any file system or operating system configuration that the cluster directly manages, such as kubelet certificates.

3.2. UPDATING NODE POOLS FOR HOSTED CONTROL PLANES

On hosted control planes, you update your version of OpenShift Container Platform by updating the node pools. The node pool version must not surpass the hosted control plane version.

Procedure

- To start the process to update to a new version of OpenShift Container Platform, change the **spec.release.image** value of the node pool by entering the following command:

```
$ oc -n NAMESPACE patch HC HCNAME --patch '{"spec":{"release":{"image": "example"}}}'
--type=merge
```

Verification

- To verify that the new version was rolled out, check the **.status.version** value and the status conditions.

3.3. CONFIGURING NODE POOLS FOR HOSTED CONTROL PLANES

On hosted control planes, you can configure node pools by creating a **MachineConfig** object inside of a config map in the management cluster.

Procedure

1. To create a **MachineConfig** object inside of a config map in the management cluster, enter the following information:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <configmap-name>
  namespace: clusters
data:
  config: |
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    metadata:
      labels:
```

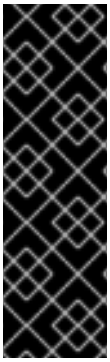
```
    machineconfiguration.openshift.io/role: worker
  name: <machineconfig-name>
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:...
        mode: 420
        overwrite: true
        path: ${PATH} 1
```

1 Sets the path on the node where the **MachineConfig** object is stored.

2. After you add the object to the config map, you can apply the config map to the node pool as follows:

```
spec:
  config:
    - name: ${CONFIGMAP_NAME}
```

3.4. CONFIGURING NODE TUNING IN A HOSTED CLUSTER



IMPORTANT

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

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

To set node-level tuning on the nodes in your hosted cluster, you can use the Node Tuning Operator. In hosted control planes, you can configure node tuning by creating config maps that contain **Tuned** objects and referencing those config maps in your node pools.

Procedure

1. Create a config map that contains a valid tuned manifest, and reference the manifest in a node pool. In the following example, a **Tuned** manifest defines a profile that sets **vm.dirty_ratio** to 55 on nodes that contain the **tuned-1-node-label** node label with any value. Save the following **ConfigMap** manifest in a file named **tuned-1.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-1
  namespace: clusters
data:
```

```
tuning: |
  apiVersion: tuned.openshift.io/v1
  kind: Tuned
  metadata:
    name: tuned-1
    namespace: openshift-cluster-node-tuning-operator
  spec:
    profile:
    - data: |
        [main]
        summary=Custom OpenShift profile
        include=openshift-node
        [sysctl]
        vm.dirty_ratio="55"
      name: tuned-1-profile
    recommend:
    - priority: 20
      profile: tuned-1-profile
```

NOTE

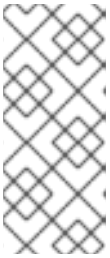
If you do not add any labels to an entry in the **spec.recommend** section of the Tuned spec, node-pool-based matching is assumed, so the highest priority profile in the **spec.recommend** section is applied to nodes in the pool. Although you can achieve more fine-grained node-label-based matching by setting a label value in the Tuned **.spec.recommend.match** section, node labels will not persist during an upgrade unless you set the **.spec.management.upgradeType** value of the node pool to **InPlace**.

2. Create the **ConfigMap** object in the management cluster:

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-1.yaml
```

3. Reference the **ConfigMap** object in the **spec.tuningConfig** field of the node pool, either by editing a node pool or creating one. In this example, assume that you have only one **NodePool**, named **nodepool-1**, which contains 2 nodes.

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  ...
  name: nodepool-1
  namespace: clusters
...
spec:
  ...
  tuningConfig:
  - name: tuned-1
status:
...
```



NOTE

You can reference the same config map in multiple node pools. In hosted control planes, the Node Tuning Operator appends a hash of the node pool name and namespace to the name of the Tuned CRs to distinguish them. Outside of this case, do not create multiple TuneD profiles of the same name in different Tuned CRs for the same hosted cluster.

Verification

Now that you have created the **ConfigMap** object that contains a **Tuned** manifest and referenced it in a **NodePool**, the Node Tuning Operator syncs the **Tuned** objects into the hosted cluster. You can verify which **Tuned** objects are defined and which TuneD profiles are applied to each node.

1. List the **Tuned** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

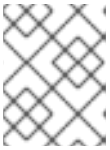
```
NAME       AGE
default    7m36s
rendered   7m36s
tuned-1    65s
```

2. List the **Profile** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME                    TUNED          APPLIED   DEGRADED   AGE
nodepool-1-worker-1     tuned-1-profile  True      False      7m43s
nodepool-1-worker-2     tuned-1-profile  True      False      7m14s
```



NOTE

If no custom profiles are created, the **openshift-node** profile is applied by default.

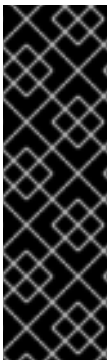
3. To confirm that the tuning was applied correctly, start a debug shell on a node and check the sysctl values:

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host sysctl vm.dirty_ratio
```

Example output

```
vm.dirty_ratio = 55
```

3.5. DEPLOYING THE SR-IOV OPERATOR FOR HOSTED CONTROL PLANES



IMPORTANT

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

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

After you configure and deploy your hosting service cluster, you can create a subscription to the SR-IOV Operator on a hosted cluster. The SR-IOV pod runs on worker machines rather than the control plane.

Prerequisites

You must configure and deploy the hosted cluster on AWS. For more information, see Configuring the hosting cluster on AWS (Technology Preview).

Procedure

1. Create a namespace and an Operator group:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
```

2. Create a subscription to the SR-IOV Operator:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: "4.13"
  name: sriov-network-operator
  config:
    nodeSelector:
```

```
    node-role.kubernetes.io/worker: ""
  source: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

Verification

1. To verify that the SR-IOV Operator is ready, run the following command and view the resulting output:

```
$ oc get csv -n openshift-sriov-network-operator
```

Example output

```
NAME                                         DISPLAY                   VERSION               REPLACES
PHASE
sriov-network-operator.4.13.0-202211021237   SR-IOV Network Operator   4.13.0-
202211021237   sriov-network-operator.4.13.0-202210290517   Succeeded
```

2. To verify that the SR-IOV pods are deployed, run the following command:

```
$ oc get pods -n openshift-sriov-network-operator
```

3.6. DELETING A HOSTED CLUSTER

The steps to delete a hosted cluster differ depending on which provider you use.

Procedure

- If the cluster is on AWS, follow the instructions in Destroying a hosted cluster on AWS .
- If the cluster is on bare metal, follow the instructions in Destroying a hosted cluster on bare metal.
- If the cluster is on OpenShift Virtualization, follow the instructions in Destroying a hosted cluster on OpenShift Virtualization.

Next steps

If you want to disable the hosted control plane feature, see Disabling the hosted control plane feature .

CHAPTER 4. BACKUP, RESTORE, AND DISASTER RECOVERY FOR HOSTED CONTROL PLANES

If you need to back up and restore etcd on a hosted cluster or provide disaster recovery for a hosted cluster, see the following procedures.

IMPORTANT

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

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

4.1. BACKING UP AND RESTORING ETCD ON A HOSTED CLUSTER

If you use hosted control planes on OpenShift Container Platform, the process to back up and restore etcd is different from the usual etcd backup process.

4.1.1. Taking a snapshot of etcd on a hosted cluster

As part of the process to back up etcd for a hosted cluster, you take a snapshot of etcd. After you take the snapshot, you can restore it, for example, as part of a disaster recovery operation.

IMPORTANT

This procedure requires API downtime.

Procedure

1. Pause reconciliation of the hosted cluster by entering this command:

```
$ oc patch -n clusters hostedclusters/${CLUSTER_NAME} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
```

2. Stop all etcd-writer deployments by entering this command:

```
$ oc scale deployment -n ${HOSTED_CLUSTER_NAMESPACE} --replicas=0 kube-
apiserver openshift-apiserver openshift-oauth-apiserver
```

3. Take an etcd snapshot by using the **exec** command in each etcd container:

```
$ oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- env ETCDCTL_API=3
/usr/bin/etcdctl --cacert /etc/etcd/tls/client/etcd-client-ca.crt --cert /etc/etcd/tls/client/etcd-
client.crt --key /etc/etcd/tls/client/etcd-client.key --endpoints=localhost:2379 snapshot save
/var/lib/data/snapshot.db
$ oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- env ETCDCTL_API=3
/usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db
```

4. Copy the snapshot data to a location where you can retrieve it later, such as an S3 bucket, as shown in the following example.

NOTE

The following example uses signature version 2. If you are in a region that supports signature version 4, such as the us-east-2 region, use signature version 4. Otherwise, if you use signature version 2 to copy the snapshot to an S3 bucket, the upload fails and signature version 2 is deprecated.

Example

```
BUCKET_NAME=somebucket
FILEPATH="/${BUCKET_NAME}/${CLUSTER_NAME}-snapshot.db"
CONTENT_TYPE="application/x-compressed-tar"
DATE_VALUE=`date -R`
SIGNATURE_STRING="PUT\n\n${CONTENT_TYPE}\n${DATE_VALUE}\n${FILEPATH}"
ACCESS_KEY=accesskey
SECRET_KEY=secret
SIGNATURE_HASH=`echo -en ${SIGNATURE_STRING} | openssl sha1 -hmac
${SECRET_KEY} -binary | base64`

oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- curl -X PUT -T
"/var/lib/data/snapshot.db" \
  -H "Host: ${BUCKET_NAME}.s3.amazonaws.com" \
  -H "Date: ${DATE_VALUE}" \
  -H "Content-Type: ${CONTENT_TYPE}" \
  -H "Authorization: AWS ${ACCESS_KEY}:${SIGNATURE_HASH}" \
  https://${BUCKET_NAME}.s3.amazonaws.com/${CLUSTER_NAME}-snapshot.db
```

5. If you want to be able to restore the snapshot on a new cluster later, save the encryption secret that the hosted cluster references, as shown in this example:

Example

```
oc get hostedcluster $CLUSTER_NAME -o=jsonpath='{.spec.secretEncryption.aescbc}'
{"activeKey":{"name":"CLUSTER_NAME-etcd-encryption-key"}}

# Save this secret, or the key it contains so the etcd data can later be decrypted
oc get secret ${CLUSTER_NAME}-etcd-encryption-key -o=jsonpath='{.data.key}'
```

Next steps

Restore the etcd snapshot.

4.1.2. Restoring an etcd snapshot on a hosted cluster

If you have a snapshot of etcd from your hosted cluster, you can restore it. Currently, you can restore an etcd snapshot only during cluster creation.

To restore an etcd snapshot, you modify the output from the **create cluster --render** command and define a **restoreSnapshotURL** value in the etcd section of the **HostedCluster** specification.

Prerequisites

You took an etcd snapshot on a hosted cluster.

Procedure

1. On the **aws** command-line interface (CLI), create a pre-signed URL so that you can download your etcd snapshot from S3 without passing credentials to the etcd deployment:

```
ETCD_SNAPSHOT=${ETCD_SNAPSHOT:-"s3://${BUCKET_NAME}/${CLUSTER_NAME}-snapshot.db"}
ETCD_SNAPSHOT_URL=$(aws s3 presign ${ETCD_SNAPSHOT})
```

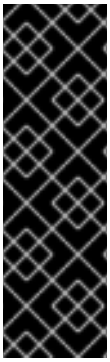
2. Modify the **HostedCluster** specification to refer to the URL:

```
spec:
  etcd:
    managed:
      storage:
        persistentVolume:
          size: 4Gi
        type: PersistentVolume
        restoreSnapshotURL:
        - "${ETCD_SNAPSHOT_URL}"
    managementType: Managed
```

3. Ensure that the secret that you referenced from the **spec.secretEncryption.aescbc** value contains the same AES key that you saved in the previous steps.

4.2. DISASTER RECOVERY FOR A HOSTED CLUSTER WITHIN AN AWS REGION

In a situation where you need disaster recovery (DR) for a hosted cluster, you can recover a hosted cluster to the same region within AWS. For example, you need DR when the upgrade of a management cluster fails and the hosted cluster is in a read-only state.



IMPORTANT

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

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

The DR process involves three main steps:

1. Backing up the hosted cluster on the source management cluster
2. Restoring the hosted cluster on a destination management cluster
3. Deleting the hosted cluster from the source management cluster

Your workloads remain running during the process. The Cluster API might be unavailable for a period, but that will not affect the services that are running on the worker nodes.

IMPORTANT

Both the source management cluster and the destination management cluster must have the **--external-dns** flags to maintain the API server URL, as shown in this example:

Example: External DNS flags

```
--external-dns-provider=aws \
--external-dns-credentials=<AWS Credentials location> \
--external-dns-domain-filter=<DNS Base Domain>
```

That way, the server URL ends with **https://api-sample-hosted.sample-hosted.aws.openshift.com**.

If you do not include the **--external-dns** flags to maintain the API server URL, the hosted cluster cannot be migrated.

4.2.1. Example environment and context

Consider an scenario where you have three clusters to restore. Two are management clusters, and one is a hosted cluster. You can restore either the control plane only or the control plane and the nodes. Before you begin, you need the following information:

- Source MGMT Namespace: The source management namespace
- Source MGMT ClusterName: The source management cluster name
- Source MGMT Kubeconfig: The source management **kubeconfig** file
- Destination MGMT Kubeconfig: The destination management **kubeconfig** file
- HC Kubeconfig: The hosted cluster **kubeconfig** file
- SSH key file: The SSH public key
- Pull secret: The pull secret file to access the release images
- AWS credentials
- AWS region
- Base domain: The DNS base domain to use as an external DNS
- S3 bucket name: The bucket in the AWS region where you plan to upload the etcd backup

This information is shown in the following example environment variables.

Example environment variables

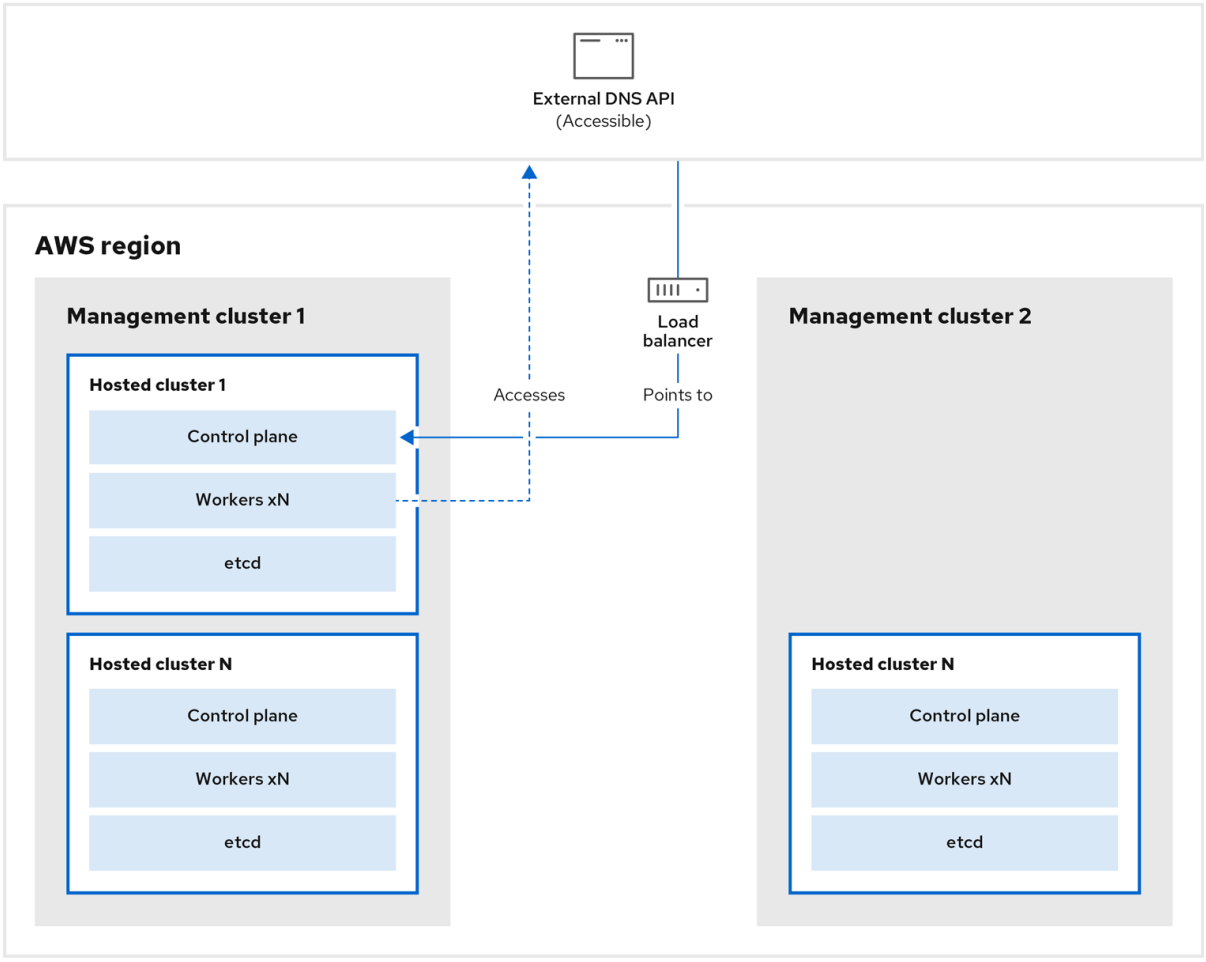
```
SSH_KEY_FILE=${HOME}/.ssh/id_rsa.pub
BASE_PATH=${HOME}/hypershift
BASE_DOMAIN="aws.sample.com"
PULL_SECRET_FILE="${HOME}/pull_secret.json"
```

```
AWS_CREDS="${HOME}/.aws/credentials"
AWS_ZONE_ID="Z02718293M33QHDEQBROL"

CONTROL_PLANE_AVAILABILITY_POLICY=SingleReplica
HYPERSHIFT_PATH=${BASE_PATH}/src/hypershift
HYPERSHIFT_CLI=${HYPERSHIFT_PATH}/bin/hypershift
HYPERSHIFT_IMAGE=${HYPERSHIFT_IMAGE:-"quay.io/${USER}/hypershift:latest"}
NODE_POOL_REPLICAS=${NODE_POOL_REPLICAS:-2}

# MGMT Context
MGMT_REGION=us-west-1
MGMT_CLUSTER_NAME="${USER}-dev"
MGMT_CLUSTER_NS=${USER}
MGMT_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT_CLUSTER_NS}-
${MGMT_CLUSTER_NAME}"
MGMT_KUBECONFIG="${MGMT_CLUSTER_DIR}/kubeconfig"

# MGMT2 Context
MGMT2_CLUSTER_NAME="${USER}-dest"
MGMT2_CLUSTER_NS=${USER}
MGMT2_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT2_CLUSTER_NS}-
${MGMT2_CLUSTER_NAME}"
MGMT2_KUBECONFIG="${MGMT2_CLUSTER_DIR}/kubeconfig"

# Hosted Cluster Context
HC_CLUSTER_NS=clusters
HC_REGION=us-west-1
HC_CLUSTER_NAME="${USER}-hosted"
HC_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}"
HC_KUBECONFIG="${HC_CLUSTER_DIR}/kubeconfig"
BACKUP_DIR=${HC_CLUSTER_DIR}/backup

BUCKET_NAME="${USER}-hosted-${MGMT_REGION}"

# DNS
AWS_ZONE_ID="Z07342811SH9AA102K1AC"
EXTERNAL_DNS_DOMAIN="hc.jpdv.aws.kerbeross.com"
```

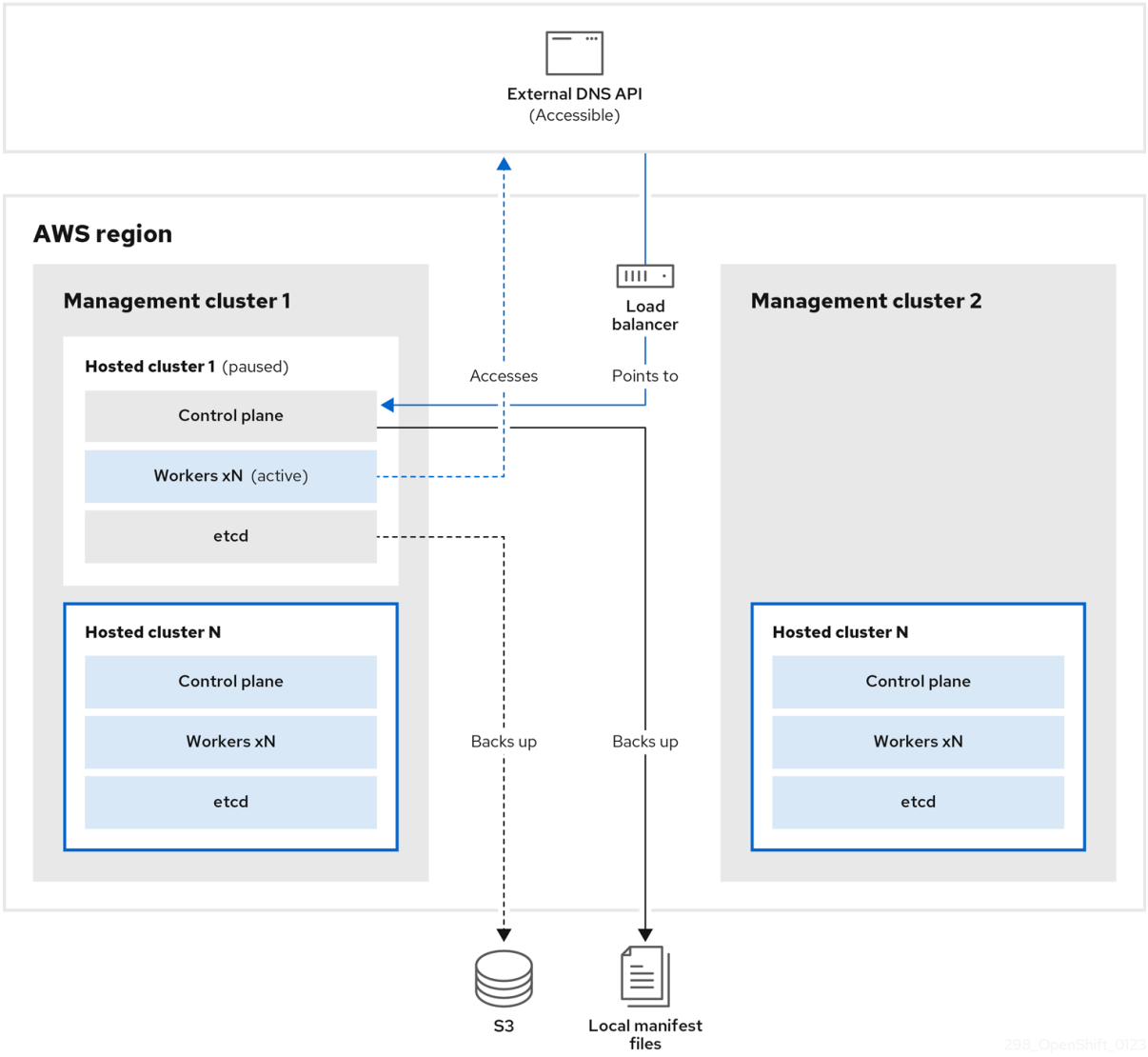
4.2.2. Overview of the backup and restore process

The backup and restore process works as follows:

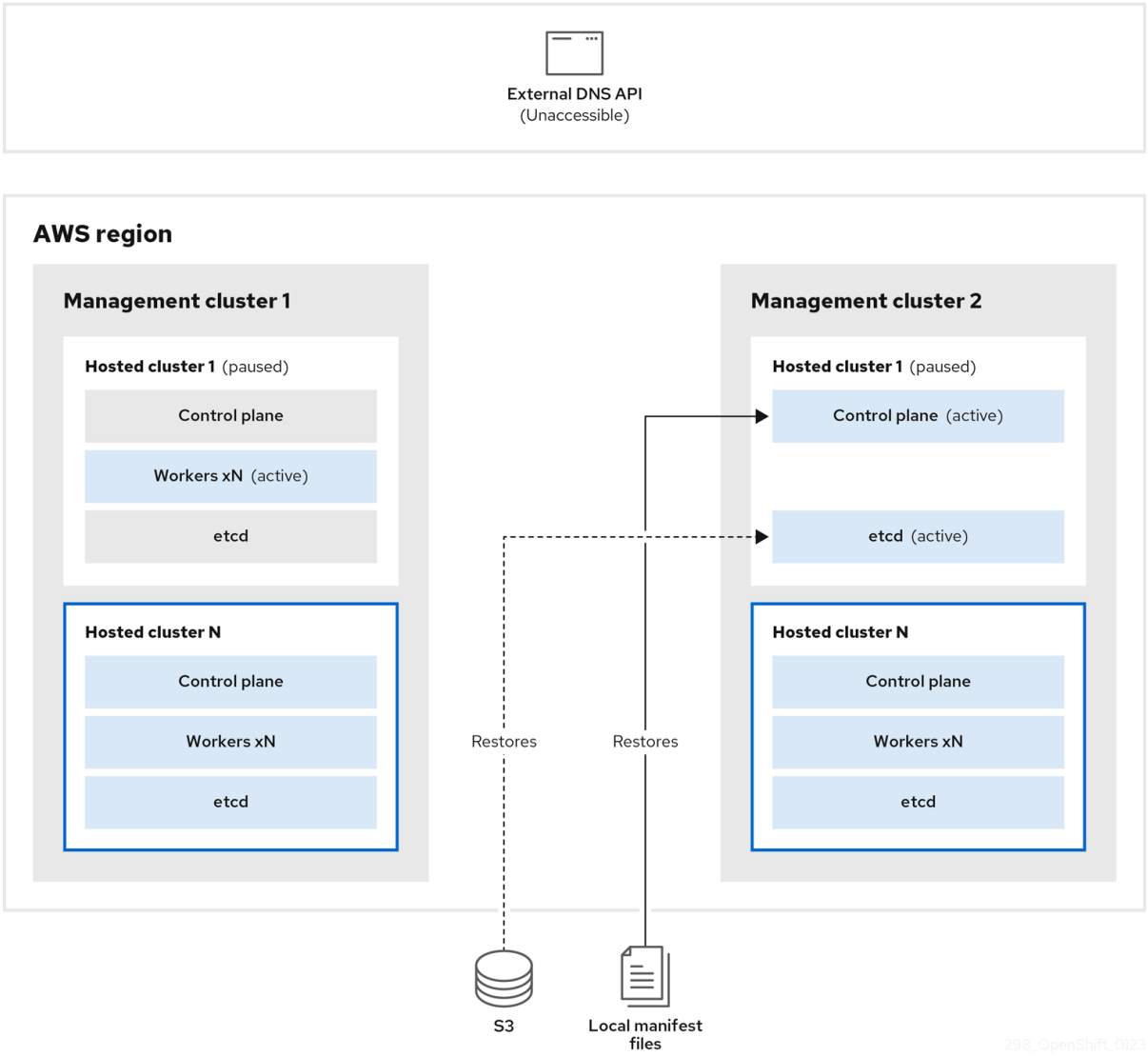
1. On management cluster 1, which you can think of as the source management cluster, the control plane and workers interact by using the external DNS API. The external DNS API is accessible, and a load balancer sits between the management clusters.



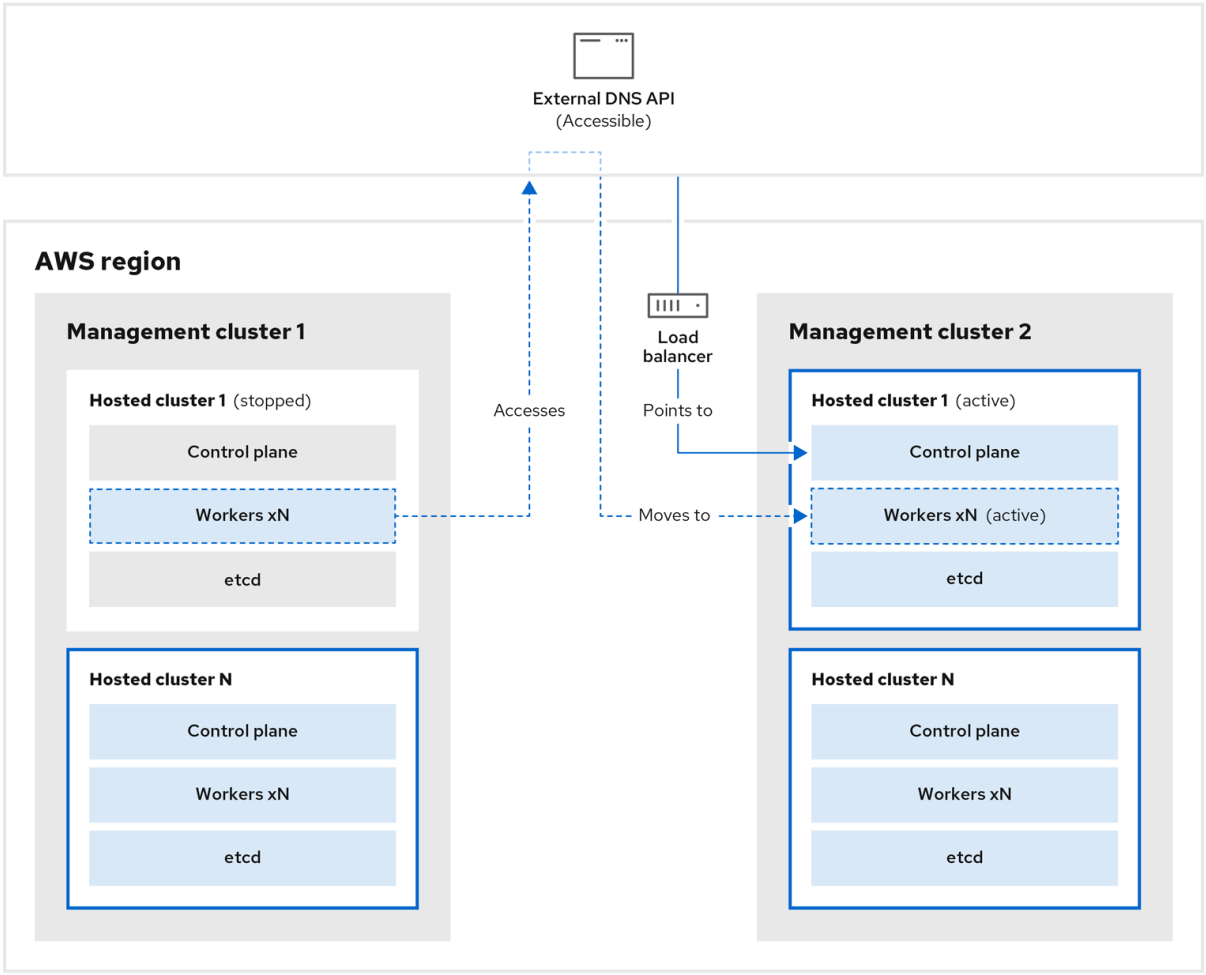
2. You take a snapshot of the hosted cluster, which includes etcd, the control plane, and the worker nodes. During this process, the worker nodes continue to try to access the external DNS API even if it is not accessible, the workloads are running, the control plane is saved in a local manifest file, and etcd is backed up to an S3 bucket. The data plane is active and the control plane is paused.



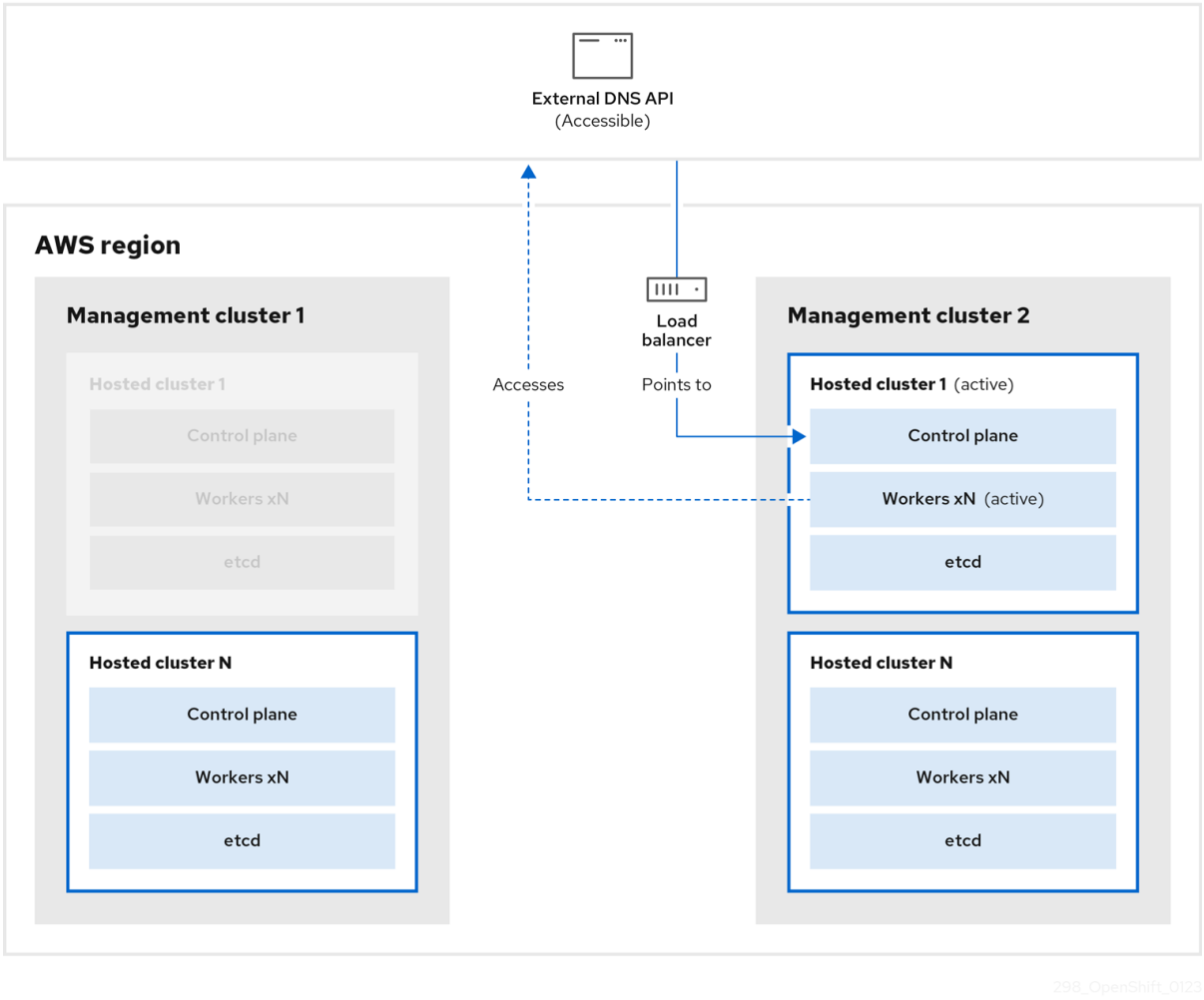
3. On management cluster 2, which you can think of as the destination management cluster, you restore etcd from the S3 bucket and restore the control plane from the local manifest file. During this process, the external DNS API is stopped, the hosted cluster API becomes inaccessible, and any workers that use the API are unable to update their manifest files, but the workloads are still running.



4. The external DNS API is accessible again, and the worker nodes use it to move to management cluster 2. The external DNS API can access the load balancer that points to the control plane.



5. On management cluster 2, the control plane and worker nodes interact by using the external DNS API. The resources are deleted from management cluster 1, except for the S3 backup of etcd. If you try to set up the hosted cluster again on mangagement cluster 1, it will not work.



You can manually back up and restore your hosted cluster, or you can run a script to complete the process. For more information about the script, see "Running a script to back up and restore a hosted cluster".

4.2.3. Backing up a hosted cluster

To recover your hosted cluster in your target management cluster, you first need to back up all of the relevant data.

Procedure

1. Create a configmap file to declare the source management cluster by entering this command:

```
$ oc create configmap mgmt-parent-cluster -n default --from-literal=from=${MGMT_CLUSTER_NAME}
```

2. Shut down the reconciliation in the hosted cluster and in the node pools by entering these commands:

```
PAUSED_UNTIL="true"
oc patch -n ${HC_CLUSTER_NS} hostedclusters/${HC_CLUSTER_NAME} -p '{"spec":{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 kube-apiserver openshift-apiserver openshift-oauth-apiserver control-plane-operator
```

```
PAUSED_UNTIL="true"
oc patch -n ${HC_CLUSTER_NS} hostedclusters/${HC_CLUSTER_NAME} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc patch -n ${HC_CLUSTER_NS} nodepools/${NODEPOOLS} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 kube-
apiserver openshift-apiserver openshift-oauth-apiserver control-plane-operator
```

3. Back up etcd and upload the data to an S3 bucket by running this bash script:

   TIP

   Wrap this script in a function and call it from the main function.

```
# ETCD Backup
ETCD_PODS="etcd-0"
if [ "${CONTROL_PLANE_AVAILABILITY_POLICY}" = "HighlyAvailable" ]; then
  ETCD_PODS="etcd-0 etcd-1 etcd-2"
fi

for POD in ${ETCD_PODS}; do
  # Create an etcd snapshot
  oc exec -it ${POD} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- env
ETCDCTL_API=3 /usr/bin/etcdctl --cacert /etc/etcd/tls/client/etcd-client-ca.crt --cert
/etc/etcd/tls/client/etcd-client.crt --key /etc/etcd/tls/client/etcd-client.key --
endpoints=localhost:2379 snapshot save /var/lib/data/snapshot.db
  oc exec -it ${POD} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- env
ETCDCTL_API=3 /usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db

  FILEPATH="/${BUCKET_NAME}/${HC_CLUSTER_NAME}-${POD}-snapshot.db"
  CONTENT_TYPE="application/x-compressed-tar"
  DATE_VALUE=`date -R`
  SIGNATURE_STRING="PUT\n\n${CONTENT_TYPE}\n${DATE_VALUE}\n${FILEPATH}"

  set +x
  ACCESS_KEY=$(grep aws_access_key_id ${AWS_CREDS} | head -n1 | cut -d= -f2 | sed
"s/ //g")
  SECRET_KEY=$(grep aws_secret_access_key ${AWS_CREDS} | head -n1 | cut -d= -f2 |
sed "s/ //g")
  SIGNATURE_HASH=$(echo -en ${SIGNATURE_STRING} | openssl sha1 -hmac
"${SECRET_KEY}" -binary | base64)
  set -x

  # FIXME: this is pushing to the OIDC bucket
  oc exec -it etcd-0 -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- curl -X PUT -T
"/var/lib/data/snapshot.db" \
    -H "Host: ${BUCKET_NAME}.s3.amazonaws.com" \
    -H "Date: ${DATE_VALUE}" \
    -H "Content-Type: ${CONTENT_TYPE}" \
    -H "Authorization: AWS ${ACCESS_KEY}:${SIGNATURE_HASH}" \
    https://${BUCKET_NAME}.s3.amazonaws.com/${HC_CLUSTER_NAME}-${POD}-
snapshot.db
done
```

For more information about backing up etcd, see "Backing up and restoring etcd on a hosted cluster".

4. Back up Kubernetes and OpenShift Container Platform objects by entering the following commands. You need to back up the following objects:

   - **HostedCluster** and **NodePool** objects from the HostedCluster namespace
   - **HostedCluster** secrets from the HostedCluster namespace
   - **HostedControlPlane** from the Hosted Control Plane namespace
   - **Cluster** from the Hosted Control Plane namespace
   - **AWSCluster**, **AWSMachineTemplate**, and **AWSMachine** from the Hosted Control Plane namespace
   - **MachineDeployments**, **MachineSets**, and **Machines** from the Hosted Control Plane namespace
   - **ControlPlane** secrets from the Hosted Control Plane namespace

```
mkdir -p ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
chmod 700 ${BACKUP_DIR}/namespaces/

# HostedCluster
echo "Backing Up HostedCluster Objects:"
oc get hc ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-${HC_CLUSTER_NAME}.yaml
echo "--> HostedCluster"
sed -i '' -e '/^status:$/,$d' ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}.yaml

# NodePool
oc get np ${NODEPOOLS} -n ${HC_CLUSTER_NS} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-${NODEPOOLS}.yaml
echo "--> NodePool"
sed -i '' -e '/^status:$/,$ d' ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-
${NODEPOOLS}.yaml

# Secrets in the HC Namespace
echo "--> HostedCluster Secrets:"
for s in $(oc get secret -n ${HC_CLUSTER_NS} | grep "^${HC_CLUSTER_NAME}" |
awk '{print $1}'); do
    oc get secret -n ${HC_CLUSTER_NS} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/secret-${s}.yaml
done

# Secrets in the HC Control Plane Namespace
echo "--> HostedCluster ControlPlane Secrets:"
for s in $(oc get secret -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} | egrep -v
"docker|service-account-token|oauth-openshift|NAME|token-${HC_CLUSTER_NAME}" |
awk '{print $1}'); do
    oc get secret -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/secret-
${s}.yaml
```

```
done

# Hosted Control Plane
echo "--> HostedControlPlane:"
oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
-o yaml > ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/hcp-${HC_CLUSTER_NAME}.yaml

# Cluster
echo "--> Cluster:"
CL_NAME=$(oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o jsonpath={.metadata.labels.\*} | grep
${HC_CLUSTER_NAME})
oc get cluster ${CL_NAME} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o yaml
> ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/cl-
${HC_CLUSTER_NAME}.yaml

# AWS Cluster
echo "--> AWS Cluster:"
oc get awscluster ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awscl-
${HC_CLUSTER_NAME}.yaml

# AWS MachineTemplate
echo "--> AWS Machine Template:"
oc get awsmachinetemplate ${NODEPOOLS} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awsmt-
${HC_CLUSTER_NAME}.yaml

# AWS Machines
echo "--> AWS Machine:"
CL_NAME=$(oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o jsonpath={.metadata.labels.\*} | grep
${HC_CLUSTER_NAME})
for s in $(oc get awsmachines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --no-
headers | grep ${CL_NAME} | cut -f1 -d\ ); do
  oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} awsmachines $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awsm-
${s}.yaml
done

# MachineDeployments
echo "--> HostedCluster MachineDeployments:"
for s in $(oc get machinedeployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
-o name); do
  mdp_name=$(echo ${s} | cut -f 2 -d /)
  oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machinedeployment-${mdp_name}.yaml
done

# MachineSets
echo "--> HostedCluster MachineSets:"
for s in $(oc get machineset -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o
```

```
name); do
    ms_name=$(echo ${s} | cut -f 2 -d /)
    oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machineset-${ms_name}.yaml
done

# Machines
echo "--> HostedCluster Machine:"
for s in $(oc get machine -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name);
do
    m_name=$(echo ${s} | cut -f 2 -d /)
    oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machine-${m_name}.yaml
done
```

5. Clean up the **ControlPlane** routes by entering this command:

```
$ oc delete routes -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all
```

By entering that command, you enable the ExternalDNS Operator to delete the Route53 entries.

6. Verify that the Route53 entries are clean by running this script:

```
function clean_routes() {

    if [[ -z "${1}" ]];then
        echo "Give me the NS where to clean the routes"
        exit 1
    fi

    # Constants
    if [[ -z "${2}" ]];then
        echo "Give me the Route53 zone ID"
        exit 1
    fi

    ZONE_ID=${2}
    ROUTES=10
    timeout=40
    count=0

    # This allows us to remove the ownership in the AWS for the API route
    oc delete route -n ${1} --all

    while [ ${ROUTES} -gt 2 ]
    do
        echo "Waiting for ExternalDNS Operator to clean the DNS Records in AWS Route53
where the zone id is: ${ZONE_ID}..."
        echo "Try: (${count}/${timeout})"
        sleep 10
        if [[ $count -eq timeout ]];then
            echo "Timeout waiting for cleaning the Route53 DNS records"
            exit 1
```

```
        fi
        count=$((count+1))
        ROUTES=$(aws route53 list-resource-record-sets --hosted-zone-id ${ZONE_ID} --max-
items 10000 --output json | grep -c ${EXTERNAL_DNS_DOMAIN})
    done
}

# SAMPLE: clean_routes "<HC ControlPlane Namespace>" "<AWS_ZONE_ID>"
clean_routes "${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}" "${AWS_ZONE_ID}"
```

Verification

Check all of the OpenShift Container Platform objects and the S3 bucket to verify that everything looks as expected.

Next steps

Restore your hosted cluster.

4.2.4. Restoring a hosted cluster

Gather all of the objects that you backed up and restore them in your destination management cluster.

Prerequisites

You backed up the data from your source management cluster.

TIP

Ensure that the **kubeconfig** file of the destination management cluster is placed as it is set in the **KUBECONFIG** variable or, if you use the script, in the **MGMT2_KUBECONFIG** variable. Use **export KUBECONFIG=<Kubeconfig FilePath>** or, if you use the script, use **export KUBECONFIG=${MGMT2_KUBECONFIG}**.

Procedure

1. Verify that the new management cluster does not contain any namespaces from the cluster that you are restoring by entering these commands:

```
# Just in case
export KUBECONFIG=${MGMT2_KUBECONFIG}
BACKUP_DIR=${HC_CLUSTER_DIR}/backup

# Namespace deletion in the destination Management cluster
$ oc delete ns ${HC_CLUSTER_NS} || true
$ oc delete ns ${HC_CLUSTER_NS}-{HC_CLUSTER_NAME} || true
```

2. Re-create the deleted namespaces by entering these commands:

```
# Namespace creation
$ oc new-project ${HC_CLUSTER_NS}
$ oc new-project ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
```

3. Restore the secrets in the HC namespace by entering this command:

```
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/secret-*
```

4. Restore the objects in the **HostedCluster** control plane namespace by entering these commands:

```
# Secrets
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/secret-*

# Cluster
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/hcp-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/cl-*
```

5. If you are recovering the nodes and the node pool to reuse AWS instances, restore the objects in the HC control plane namespace by entering these commands:

```
# AWS
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awscl-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awsmt-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awsm-*

# Machines
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machinedeployment-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machineset-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machine-*
```

6. Restore the etcd data and the hosted cluster by running this bash script:

```
ETCD_PODS="etcd-0"
if [ "${CONTROL_PLANE_AVAILABILITY_POLICY}" = "HighlyAvailable" ]; then
  ETCD_PODS="etcd-0 etcd-1 etcd-2"
fi

HC_RESTORE_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}-restore.yaml
HC_BACKUP_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}.yaml
HC_NEW_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}-new.yaml
cat ${HC_BACKUP_FILE} > ${HC_NEW_FILE}
cat > ${HC_RESTORE_FILE} <<EOF
    restoreSnapshotURL:
EOF

for POD in ${ETCD_PODS}; do
  # Create a pre-signed URL for the etcd snapshot
  ETCD_SNAPSHOT="s3://${BUCKET_NAME}/${HC_CLUSTER_NAME}-${POD}-
```

```
snapshot.db"
  ETCD_SNAPSHOT_URL=$(AWS_DEFAULT_REGION=${MGMT2_REGION} aws s3
presign ${ETCD_SNAPSHOT})

  # FIXME no CLI support for restoreSnapshotURL yet
  cat >> ${HC_RESTORE_FILE} <<EOF
    - "${ETCD_SNAPSHOT_URL}"
EOF
done

cat ${HC_RESTORE_FILE}

if ! grep ${HC_CLUSTER_NAME}-snapshot.db ${HC_NEW_FILE}; then
  sed -i '' -e "/type: PersistentVolume/r ${HC_RESTORE_FILE}" ${HC_NEW_FILE}
  sed -i '' -e '/pausedUntil:/d' ${HC_NEW_FILE}
fi

HC=$(oc get hc -n ${HC_CLUSTER_NS} ${HC_CLUSTER_NAME} -o name || true)
if [[ ${HC} == "" ]];then
    echo "Deploying HC Cluster: ${HC_CLUSTER_NAME} in ${HC_CLUSTER_NS}
namespace"
    oc apply -f ${HC_NEW_FILE}
else
    echo "HC Cluster ${HC_CLUSTER_NAME} already exists, avoiding step"
fi
```

7. If you are recovering the nodes and the node pool to reuse AWS instances, restore the node pool by entering this command:

```
oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-*
```

Verification

- To verify that the nodes are fully restored, use this function:

```
timeout=40
count=0
NODE_STATUS=$(oc get nodes --kubeconfig=${HC_KUBECONFIG} | grep -v NotReady |
grep -c "worker") || NODE_STATUS=0

while [ ${NODE_POOL_REPLICAS} != ${NODE_STATUS} ]
do
    echo "Waiting for Nodes to be Ready in the destination MGMT Cluster:
${MGMT2_CLUSTER_NAME}"
    echo "Try: (${count}/${timeout})"
    sleep 30
    if [[ $count -eq timeout ]];then
        echo "Timeout waiting for Nodes in the destination MGMT Cluster"
        exit 1
    fi
    count=$((count+1))
    NODE_STATUS=$(oc get nodes --kubeconfig=${HC_KUBECONFIG} | grep -v NotReady |
grep -c "worker") || NODE_STATUS=0
done
```

Next steps

Shut down and delete your cluster.

4.2.5. Deleting a hosted cluster from your source management cluster

After you back up your hosted cluster and restore it to your destination management cluster, you shut down and delete the hosted cluster on your source management cluster.

Prerequisites

You backed up your data and restored it to your source management cluster.

TIP

Ensure that the **kubeconfig** file of the destination management cluster is placed as it is set in the **KUBECONFIG** variable or, if you use the script, in the **MGMT_KUBECONFIG** variable. Use **export KUBECONFIG=<Kubeconfig FilePath>** or, if you use the script, use **export KUBECONFIG=${MGMT_KUBECONFIG}**.

Procedure

1. Scale the **deployment** and **statefulset** objects by entering these commands:

```
# Just in case
export KUBECONFIG=${MGMT_KUBECONFIG}

# Scale down deployments
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 --all
oc scale statefulset.apps -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 --
all
sleep 15
```

2. Delete the **NodePool** objects by entering these commands:

```
NODEPOOLS=$(oc get nodepools -n ${HC_CLUSTER_NS} -o=jsonpath='{.items[?
(@.spec.clusterName=="'${HC_CLUSTER_NAME}'")].metadata.name}')
if [[ ! -z "${NODEPOOLS}" ]];then
    oc patch -n "${HC_CLUSTER_NS}" nodepool ${NODEPOOLS} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]'
    oc delete np -n ${HC_CLUSTER_NS} ${NODEPOOLS}
fi
```

3. Delete the **machine** and **machineset** objects by entering these commands:

```
# Machines
for m in $(oc get machines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name); do
    oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]' || true
    oc delete -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} || true
done

oc delete machineset -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all || true
```

4. Delete the cluster object by entering these commands:

```
# Cluster
C_NAME=$(oc get cluster -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name)
oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${C_NAME} --type=json --
patch='[ { "op":"remove", "path": "/metadata/finalizers" }]'
oc delete cluster.cluster.x-k8s.io -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all
```

5. Delete the AWS machines (Kubernetes objects) by entering these commands. Do not worry about deleting the real AWS machines. The cloud instances will not be affected.

```
# AWS Machines
for m in $(oc get awsmachine.infrastructure.cluster.x-k8s.io -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o name)
do
   oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]' || true
   oc delete -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} || true
done
```

6. Delete the **HostedControlPlane** and **ControlPlane** HC namespace objects by entering these commands:

```
# Delete HCP and ControlPlane HC NS
oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
hostedcontrolplane.hypershift.openshift.io ${HC_CLUSTER_NAME} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]'
oc delete hostedcontrolplane.hypershift.openshift.io -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} --all
oc delete ns ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} || true
```

7. Delete the **HostedCluster** and HC namespace objects by entering these commands:

```
# Delete HC and HC Namespace
oc -n ${HC_CLUSTER_NS} patch hostedclusters ${HC_CLUSTER_NAME} -p '{"metadata":
{"finalizers":null}}' --type merge || true
oc delete hc -n ${HC_CLUSTER_NS} ${HC_CLUSTER_NAME}  || true
oc delete ns ${HC_CLUSTER_NS} || true
```

Verification

- To verify that everything works, enter these commands:

```
# Validations
export KUBECONFIG=${MGMT2_KUBECONFIG}

oc get hc -n ${HC_CLUSTER_NS}
oc get np -n ${HC_CLUSTER_NS}
oc get pod -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
oc get machines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}

# Inside the HostedCluster
export KUBECONFIG=${HC_KUBECONFIG}
oc get clusterversion
oc get nodes
```

Next steps

Delete the OVN pods in the hosted cluster so that you can connect to the new OVN control plane that runs in the new management cluster:

1. Load the **KUBECONFIG** environment variable with the hosted cluster's kubeconfig path.
2. Enter this command:

```
$ oc delete pod -n openshift-ovn-kubernetes --all
```

4.2.6. Running a script to back up and restore a hosted cluster

To expedite the process to back up a hosted cluster and restore it within the same region on AWS, you can modify and run a script.

Procedure

1. Replace the variables in the following script with your information:

```
# Fill the Common variables to fit your environment, this is just a sample
SSH_KEY_FILE=${HOME}/.ssh/id_rsa.pub
BASE_PATH=${HOME}/hypershift
BASE_DOMAIN="aws.sample.com"
PULL_SECRET_FILE="${HOME}/pull_secret.json"
AWS_CREDS="${HOME}/.aws/credentials"
CONTROL_PLANE_AVAILABILITY_POLICY=SingleReplica
HYPERSHIFT_PATH=${BASE_PATH}/src/hypershift
HYPERSHIFT_CLI=${HYPERSHIFT_PATH}/bin/hypershift
HYPERSHIFT_IMAGE=${HYPERSHIFT_IMAGE:-"quay.io/${USER}/hypershift:latest"}
NODE_POOL_REPLICAS=${NODE_POOL_REPLICAS:-2}

# MGMT Context
MGMT_REGION=us-west-1
MGMT_CLUSTER_NAME="${USER}-dev"
MGMT_CLUSTER_NS=${USER}
MGMT_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT_CLUSTER_NS}-
${MGMT_CLUSTER_NAME}"
MGMT_KUBECONFIG="${MGMT_CLUSTER_DIR}/kubeconfig"

# MGMT2 Context
MGMT2_CLUSTER_NAME="${USER}-dest"
MGMT2_CLUSTER_NS=${USER}
MGMT2_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT2_CLUSTER_NS}-
${MGMT2_CLUSTER_NAME}"
MGMT2_KUBECONFIG="${MGMT2_CLUSTER_DIR}/kubeconfig"

# Hosted Cluster Context
HC_CLUSTER_NS=clusters
HC_REGION=us-west-1
HC_CLUSTER_NAME="${USER}-hosted"
HC_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}"
HC_KUBECONFIG="${HC_CLUSTER_DIR}/kubeconfig"
BACKUP_DIR=${HC_CLUSTER_DIR}/backup
```

```
BUCKET_NAME="${USER}-hosted-${MGMT_REGION}"

# DNS
AWS_ZONE_ID="Z026552815SS3YPH9H6MG"
EXTERNAL_DNS_DOMAIN="guest.jpdv.aws.kerbeross.com"
```

2. Save the script to your local file system.
3. Run the script by entering the following command:

```
source <env_file>
```

where: **env_file** is the name of the file where you saved the script.

The migration script is maintained at the following repository: https://github.com/openshift/hypershift/blob/main/contrib/migration/migrate-hcp.sh.