



OpenShift Container Platform 4.16

Hosted control planes

Using hosted control planes with OpenShift Container Platform

OpenShift Container Platform 4.16 Hosted control planes

Using hosted control planes with OpenShift Container Platform

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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	4
1.1. GLOSSARY OF COMMON CONCEPTS AND PERSONAS FOR HOSTED CONTROL PLANES	4
1.1.1. Concepts	4
1.1.2. Personas	4
1.2. INTRODUCTION TO HOSTED CONTROL PLANES	5
1.2.1. Architecture of hosted control planes	5
1.2.2. Benefits of hosted control planes	6
1.3. VERSIONING FOR HOSTED CONTROL PLANES	7
CHAPTER 2. GETTING STARTED WITH HOSTED CONTROL PLANES	9
2.1. BARE METAL	9
2.2. OPENSIFT VIRTUALIZATION	9
2.3. AMAZON WEB SERVICES (AWS)	10
2.4. IBM Z	10
2.5. IBM POWER	10
2.6. NON BARE METAL AGENT MACHINES	11
CHAPTER 3. AUTHENTICATION AND AUTHORIZATION FOR HOSTED CONTROL PLANES	12
3.1. CONFIGURING THE OAUTH SERVER FOR A HOSTED CLUSTER BY USING THE CLI	12
3.2. CONFIGURING THE OAUTH SERVER FOR A HOSTED CLUSTER BY USING THE WEB CONSOLE	13
3.3. ASSIGNING COMPONENTS IAM ROLES BY USING THE CCO IN A HOSTED CLUSTER ON AWS	15
3.4. VERIFYING THE CCO INSTALLATION IN A HOSTED CLUSTER ON AWS	15
3.5. ENABLING OPERATORS TO SUPPORT CCO-BASED WORKFLOWS WITH AWS STS	16
CHAPTER 4. HANDLING A MACHINE CONFIGURATION FOR HOSTED CONTROL PLANES	23
4.1. CONFIGURING NODE POOLS FOR HOSTED CONTROL PLANES	23
4.2. CONFIGURING NODE TUNING IN A HOSTED CLUSTER	24
4.3. DEPLOYING THE SR-IOV OPERATOR FOR HOSTED CONTROL PLANES	26
CHAPTER 5. USING FEATURE GATES IN A HOSTED CLUSTER	28
5.1. ENABLING FEATURE SETS BY USING FEATURE GATES	28
CHAPTER 6. UPDATING HOSTED CONTROL PLANES	30
6.1. REQUIREMENTS TO UPGRADE HOSTED CONTROL PLANES	30
6.2. UPDATES FOR THE HOSTED CLUSTER	30
6.3. UPDATES FOR NODE POOLS	30
6.3.1. Replace updates for node pools	31
6.3.2. In place updates for node pools	31
6.4. UPDATING NODE POOLS FOR HOSTED CONTROL PLANES	31
6.5. UPDATING THE HOSTED CLUSTER FOR HOSTED CONTROL PLANES	32
CHAPTER 7. HOSTED CONTROL PLANES OBSERVABILITY	34
7.1. CONFIGURING METRICS SETS FOR HOSTED CONTROL PLANES	34
7.1.1. Configuring the SRE metrics set	34
7.2. ENABLING MONITORING DASHBOARDS IN A HOSTED CLUSTER	36
7.2.1. Dashboard customization	37
CHAPTER 8. HIGH AVAILABILITY FOR HOSTED CONTROL PLANES	39
8.1. ABOUT HIGH AVAILABILITY FOR HOSTED CONTROL PLANES	39
8.1.1. Impact of the failed management cluster component	39
8.2. RECOVERING AN UNHEALTHY ETCD CLUSTER	39
8.2.1. Checking the status of an etcd cluster	39
8.2.2. Recovering a failing etcd pod	40

8.3. BACKING UP AND RESTORING ETCD IN AN ON-PREMISE ENVIRONMENT	40
8.3.1. Backing up and restoring etcd on a hosted cluster in an on-premise environment	41
8.4. BACKING UP AND RESTORING ETCD ON AWS	45
8.4.1. Taking a snapshot of etcd for a hosted cluster	45
8.4.2. Restoring an etcd snapshot on a hosted cluster	46
8.5. DISASTER RECOVERY FOR A HOSTED CLUSTER IN AWS	47
8.5.1. Overview of the backup and restore process	48
8.5.2. Backing up a hosted cluster	53
8.5.3. Restoring a hosted cluster	58
8.5.4. Deleting a hosted cluster from your source management cluster	61
8.6. DISASTER RECOVERY FOR A HOSTED CLUSTER BY USING OADP	63
8.6.1. Prerequisites	63
8.6.2. Preparing AWS to use OADP	64
8.6.3. Preparing bare metal to use OADP	64
8.6.4. Backing up the data plane workload	64
8.6.5. Backing up the control plane workload	65
8.6.6. Restoring a hosted cluster by using OADP	67
8.6.7. Observing the backup and restore process	69
8.6.8. Using the velero CLI to describe the Backup and Restore resources	69
CHAPTER 9. TROUBLESHOOTING HOSTED CONTROL PLANES	71
9.1. GATHERING INFORMATION TO TROUBLESHOOT HOSTED CONTROL PLANES	71
9.2. RESTARTING HOSTED CONTROL PLANE COMPONENTS	72
9.3. PAUSING THE RECONCILIATION OF A HOSTED CLUSTER AND HOSTED CONTROL PLANE	73
9.4. SCALING DOWN THE DATA PLANE TO ZERO	74

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. GLOSSARY OF COMMON CONCEPTS AND PERSONAS FOR HOSTED CONTROL PLANES

When you use hosted control planes for OpenShift Container Platform, it is important to understand its key concepts and the personas that are involved.

1.1.1. Concepts

hosted cluster

An OpenShift Container Platform cluster with its control plane and API endpoint hosted on a management cluster. The hosted cluster includes the control plane and its corresponding data plane.

hosted cluster infrastructure

Network, compute, and storage resources that exist in the tenant or end-user cloud account.

hosted control plane

An OpenShift Container Platform control plane that runs on the management cluster, which is exposed by the API endpoint of a hosted cluster. The components of a control plane include etcd, the Kubernetes API server, the Kubernetes controller manager, and a VPN.

hosting cluster

See [management cluster](#).

managed cluster

A cluster that the hub cluster manages. This term is specific to the cluster lifecycle that the multicluster engine for Kubernetes Operator manages in Red Hat Advanced Cluster Management. A managed cluster is not the same thing as a *management cluster*. For more information, see [Managed cluster](#).

management cluster

An OpenShift Container Platform cluster where the HyperShift Operator is deployed and where the control planes for hosted clusters are hosted. The management cluster is synonymous with the *hosting cluster*.

management cluster infrastructure

Network, compute, and storage resources of the management cluster.

node pool

A resource that contains the compute nodes. The control plane contains node pools. The compute nodes run applications and workloads.

1.1.2. Personas

cluster instance administrator

Users who assume this role are the equivalent of administrators in standalone OpenShift Container Platform. This user has the **cluster-admin** role in the provisioned cluster, but might not have power over when or how the cluster is updated or configured. This user might have read-only access to see

some configuration projected into the cluster.

cluster instance user

Users who assume this role are the equivalent of developers in standalone OpenShift Container Platform. This user does not have a view into OperatorHub or machines.

cluster service consumer

Users who assume this role can request control planes and worker nodes, drive updates, or modify externalized configurations. Typically, this user does not manage or access cloud credentials or infrastructure encryption keys. The cluster service consumer persona can request hosted clusters and interact with node pools. Users who assume this role have RBAC to create, read, update, or delete hosted clusters and node pools within a logical boundary.

cluster service provider

Users who assume this role typically have the **cluster-admin** role on the management cluster and have RBAC to monitor and own the availability of the HyperShift Operator as well as the control planes for the tenant's hosted clusters. The cluster service provider persona is responsible for several activities, including the following examples:

- Owning service-level objects for control plane availability, uptime, and stability
- Configuring the cloud account for the management cluster to host control planes
- Configuring the user-provisioned infrastructure, which includes the host awareness of available compute resources

1.2. INTRODUCTION TO HOSTED CONTROL PLANES

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.

Hosted control planes is available by using the [multicluster engine for Kubernetes Operator version 2.0 or later](#) on the following platforms:

- Bare metal by using the Agent provider
- OpenShift Virtualization, as a Generally Available feature in connected environments and a Technology Preview feature in disconnected environments
- Amazon Web Services (AWS), as a Technology Preview feature
- IBM Z, as a Technology Preview feature
- IBM Power, as a Technology Preview feature

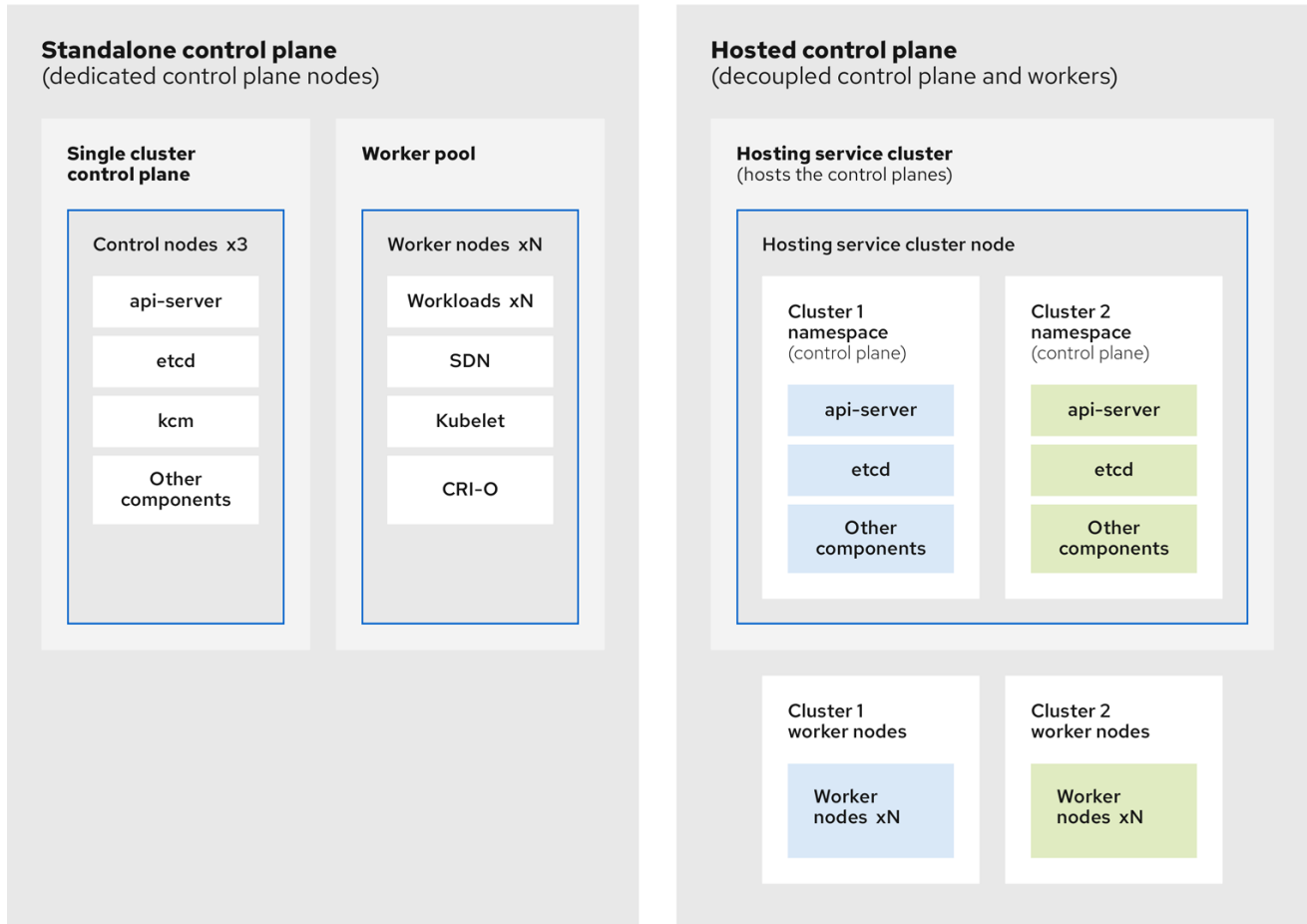
1.2.1. Architecture of hosted control planes

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.



272_OpenShift_1122

1.2.2. 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

- [Hosted control planes](#)

1.3. 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:

- The HyperShift Operator
- The **hcp** command-line interface (CLI)

The HyperShift Operator manages the lifecycle of hosted clusters that are represented by the **HostedCluster** API resources. The HyperShift Operator is released with each OpenShift Container Platform release. The HyperShift Operator creates the **supported-versions** config map in the **hypershift** namespace. The config map contains the supported hosted cluster versions.

You can host different versions of control planes on the same management cluster.

Example **supported-versions** config map object

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.16"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

You can use the **hcp** CLI to create hosted clusters.

You can use the **hypershift.openshift.io** API resources, such as, **HostedCluster** and **NodePool**, to create and manage OpenShift Container Platform clusters at scale. A **HostedCluster** resource contains 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](#).

Additional resources

- [Configuring node tuning in a hosted cluster](#)

- [Advanced node tuning for hosted clusters by setting kernel boot parameters](#)

CHAPTER 2. GETTING STARTED WITH 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.

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

2.1. BARE METAL

- [Hosted control plane sizing guidance](#)
- [Installing the hosted control plane command line interface](#)
- [Distributing hosted cluster workloads](#)
- [Bare metal firewall and port requirements](#)
- [Bare metal infrastructure requirements](#): Review the infrastructure requirements to create a hosted cluster on bare metal.
- [Configuring hosted control plane clusters on bare metal](#) :
 - Configure DNS
 - Create a hosted cluster and verify cluster creation
 - Scale the **NodePool** object for the hosted cluster
 - Handle ingress traffic for the hosted cluster
 - Enable node auto-scaling for the hosted cluster
- [Configuring hosted control planes in a disconnected environment](#)
- To destroy a hosted cluster on bare metal, follow the instructions in [Destroying a hosted cluster on bare metal](#).
- If you want to disable the hosted control plane feature, see [Disabling the hosted control plane feature](#).

2.2. OPENSIFT VIRTUALIZATION

- [Hosted control plane sizing guidance](#)
- [Installing the hosted control plane command line interface](#)
- [Distributing hosted cluster workloads](#)
- [Managing hosted control plane clusters on OpenShift Virtualization](#) : Create OpenShift Container Platform clusters with worker nodes that are hosted by KubeVirt virtual machines.
- [Configuring hosted control planes in a disconnected environment](#)
- To destroy a hosted cluster is on OpenShift Virtualization, follow the instructions in [Destroying a hosted cluster on OpenShift Virtualization](#).

- If you want to disable the hosted control plane feature, see [Disabling the hosted control plane feature](#).

2.3. AMAZON WEB SERVICES (AWS)

- [AWS infrastructure requirements](#): Review the infrastructure requirements to create a hosted cluster on AWS.
- [Configuring hosted control plane clusters on AWS](#): The tasks to configure hosted control plane clusters on AWS include creating the AWS S3 OIDC secret, creating a routable public zone, enabling external DNS, enabling AWS PrivateLink, and deploying a hosted cluster.
- [Deploying the SR-IOV Operator for hosted control planes](#): After you configure and deploy your hosting service cluster, you can create a subscription to the Single Root I/O Virtualization (SR-IOV) Operator on a hosted cluster. The SR-IOV pod runs on worker machines rather than the control plane.
- To destroy a hosted cluster on AWS, follow the instructions in [Destroying a hosted cluster on AWS](#).
- If you want to disable the hosted control plane feature, see [Disabling the hosted control plane feature](#).

2.4. IBM Z



IMPORTANT

Hosted control planes on the IBM Z platform 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](#).

- [Installing the hosted control plane command line interface](#)
- [Configuring the hosting cluster on x86 bare metal for IBM Z compute nodes \(Technology Preview\)](#)

2.5. IBM POWER



IMPORTANT

Hosted control planes on the IBM Power platform 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](#).

- [Installing the hosted control plane command line interface](#)
- [Configuring the hosting cluster on a 64-bit x86 OpenShift Container Platform cluster to create hosted control planes for IBM Power compute nodes \(Technology Preview\)](#)

2.6. NON BARE METAL AGENT MACHINES



IMPORTANT

Hosted control planes clusters using non bare metal agent machines 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](#).

- [Installing the hosted control plane command line interface](#)
- [Configuring hosted control plane clusters using non bare metal agent machines \(Technology Preview\)](#)
- To destroy a hosted cluster on non bare metal agent machines, follow the instructions in [Destroying a hosted cluster on non bare metal agent machines](#)
- If you want to disable the hosted control plane feature, see [Disabling the hosted control plane feature](#).

CHAPTER 3. AUTHENTICATION AND AUTHORIZATION FOR HOSTED CONTROL PLANES

The OpenShift Container Platform control plane includes a built-in OAuth server. You can obtain OAuth access tokens to authenticate to the OpenShift Container Platform API. After you create your hosted cluster, you can configure OAuth by specifying an identity provider.

3.1. CONFIGURING THE OAUTH SERVER FOR A HOSTED CLUSTER BY USING THE CLI

You can configure the internal OAuth server for your hosted cluster by using an OpenID Connect identity provider (**oidc**).

You can configure OAuth for the following supported identity providers:

- **oidc**
- **htpasswd**
- **keystone**
- **ldap**
- **basic-authentication**
- **request-header**
- **github**
- **gitlab**
- **google**

Adding any identity provider in the OAuth configuration removes the default **kubeadmin** user provider.

Prerequisites

- You created your hosted cluster.

Procedure

1. Edit the **HostedCluster** custom resource (CR) on the hosting cluster by running the following command:

```
$ oc edit <hosted_cluster_name> -n <hosted_cluster_namespace>
```

2. Add the OAuth configuration in the **HostedCluster** CR by using the following example:

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name> 1
  namespace: <hosted_cluster_namespace> 2
spec:
```



```

configuration:
  oauth:
    identityProviders:
      - openID: ❸
        claims:
          email: ❹
            - <email_address>
          name: ❺
            - <display_name>
          preferredUsername: ❻
            - <preferred_username>
        clientID: <client_id> ❷
        clientSecret:
          name: <client_id_secret_name> ❸
        issuer: https://example.com/identity ❹
        mappingMethod: lookup ❺
        name: IAM
        type: OpenID

```

- ❶ Specifies your hosted cluster name.
- ❷ Specifies your hosted cluster namespace.
- ❸ This provider name is prefixed to the value of the identity claim to form an identity name. The provider name is also used to build the redirect URL.
- ❹ Defines a list of attributes to use as the email address.
- ❺ Defines a list of attributes to use as a display name.
- ❻ Defines a list of attributes to use as a preferred user name.
- ❼ Defines the ID of a client registered with the OpenID provider. You must allow the client to redirect to the **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** URL.
- ❽ Defines a secret of a client registered with the OpenID provider.
- ❾ The [Issuer Identifier](#) described in the OpenID spec. You must use **https** without query or fragment component.
- ❿ Defines a mapping method that controls how mappings are established between identities of this provider and **User** objects.

3. Save the file to apply the changes.

3.2. CONFIGURING THE OAUTH SERVER FOR A HOSTED CLUSTER BY USING THE WEB CONSOLE

You can configure the internal OAuth server for your hosted cluster by using the OpenShift Container Platform web console.

You can configure OAuth for the following supported identity providers:


- **oidc**
- **htpasswd**
- **keystone**
- **ldap**
- **basic-authentication**
- **request-header**
- **github**
- **gitlab**
- **google**

Adding any identity provider in the OAuth configuration removes the default **kubeadmin** user provider.

Prerequisites

- You logged in as a user with **cluster-admin** privileges.
- You created your hosted cluster.

Procedure

1. Navigate to **Home → API Explorer**.
2. Use the **Filter by kind** box to search for your **HostedCluster** resource.
3. Click the **HostedCluster** resource that you want to edit.
4. Click the **Instances** tab.
5. Click the Options menu  next to your hosted cluster name entry and click **Edit HostedCluster**.
6. Add the OAuth configuration in the YAML file:

```
spec:
  configuration:
    oauth:
      identityProviders:
        - openID: ❶
          claims:
            email: ❷
              - <email_address>
            name: ❸
              - <display_name>
            preferredUsername: ❹
              - <preferred_username>
          clientID: <client_id> ❺
```

```

clientSecret:
  name: <client_id_secret_name> 6
  issuer: https://example.com/identity 7
mappingMethod: lookup 8
name: IAM
type: OpenID

```

- 1** This provider name is prefixed to the value of the identity claim to form an identity name. The provider name is also used to build the redirect URL.
- 2** Defines a list of attributes to use as the email address.
- 3** Defines a list of attributes to use as a display name.
- 4** Defines a list of attributes to use as a preferred user name.
- 5** Defines the ID of a client registered with the OpenID provider. You must allow the client to redirect to the **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** URL.
- 6** Defines a secret of a client registered with the OpenID provider.
- 7** The *Issuer Identifier* described in the OpenID spec. You must use **https** without query or fragment component.
- 8** Defines a mapping method that controls how mappings are established between identities of this provider and **User** objects.

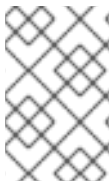
7. Click **Save**.

Additional resources

- To know more about supported identity providers, see "[Understanding identity provider configuration](#)" in *Authentication and authorization*.

3.3. ASSIGNING COMPONENTS IAM ROLES BY USING THE CCO IN A HOSTED CLUSTER ON AWS

You can assign components IAM roles that provide short-term, limited-privilege security credentials by using the Cloud Credential Operator (CCO) in hosted clusters on Amazon Web Services (AWS). By default, the CCO runs in a hosted control plane.



NOTE

The CCO supports a manual mode only for hosted clusters on AWS. By default, hosted clusters are configured in a manual mode. The management cluster might use modes other than manual.

3.4. VERIFYING THE CCO INSTALLATION IN A HOSTED CLUSTER ON AWS

You can verify that the Cloud Credential Operator (CCO) is running correctly in your hosted control plane.

Prerequisites

- You configured the hosted cluster on Amazon Web Services (AWS).

Procedure

1. Verify that the CCO is configured in a manual mode in your hosted cluster by running the following command:

```
$ oc get cloudcredentials <hosted_cluster_name> -n <hosted_cluster_namespace> -o=jsonpath={.spec.credentialsMode}
```

Expected output

```
Manual
```

2. Verify that the value for the **serviceAccountIssuer** resource is not empty by running the following command:

```
$ oc get authentication cluster --kubeconfig <hosted_cluster_name>.kubeconfig -o jsonpath -template '{.spec.serviceAccountIssuer}'
```

Example output

```
https://aos-hypershift-ci-oidc-299999.s3.us-east-2.amazonaws.com/hypershift-ci-299999
```

3.5. ENABLING OPERATORS TO SUPPORT CCO-BASED WORKFLOWS WITH AWS STS

As an Operator author designing your project to run on Operator Lifecycle Manager (OLM), you can enable your Operator to authenticate against AWS on STS-enabled OpenShift Container Platform clusters by customizing your project to support the Cloud Credential Operator (CCO).

With this method, the Operator is responsible for creating the **CredentialsRequest** object, which means the Operator requires RBAC permission to create these objects. Then, the Operator must be able to read the resulting **Secret** object.



NOTE

By default, pods related to the Operator deployment mount a **serviceAccountToken** volume so that the service account token can be referenced in the resulting **Secret** object.

Prerequisites

- OpenShift Container Platform 4.14 or later
- Cluster in STS mode
- OLM-based Operator project

Procedure

1. Update your Operator project's **ClusterServiceVersion** (CSV) object:
 - a. Ensure your Operator has RBAC permission to create **CredentialsRequests** objects:

Example 3.1. Example clusterPermissions list

```
# ...
install:
spec:
  clusterPermissions:
  - rules:
    - apiGroups:
      - "cloudcredential.openshift.io"
    resources:
      - credentialsrequests
    verbs:
      - create
      - delete
      - get
      - list
      - patch
      - update
      - watch
```

- b. Add the following annotation to claim support for this method of CCO-based workflow with AWS STS:

```
# ...
metadata:
  annotations:
    features.operators.openshift.io/token-auth-aws: "true"
```

2. Update your Operator project code:
 - a. Get the role ARN from the environment variable set on the pod by the **Subscription** object. For example:

```
// Get ENV var
roleARN := os.Getenv("ROLEARN")
setupLog.Infof("getting role ARN", "role ARN = ", roleARN)
webIdentityTokenPath := "/var/run/secrets/openshift/serviceaccount/token"
```

- b. Ensure you have a **CredentialsRequest** object ready to be patched and applied. For example:

Example 3.2. Example CredentialsRequest object creation

```
import (
  minterv1 "github.com/openshift/cloud-credential-
operator/pkg/apis/cloudcredential/v1"
  corev1 "k8s.io/api/core/v1"
  metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

var in = minterv1.AWSPProviderSpec{
```

```

StatementEntries: []minterv1.StatementEntry{
    {
        Action: []string{
            "s3:*",
        },
        Effect: "Allow",
        Resource: "arn:aws:s3:*:*:*:*",
    },
},
STSIAMRoleARN: "<role_arn>",
}

var codec = minterv1.Codec
var ProviderSpec, _ = codec.EncodeProviderSpec(in.DeepCopyObject())

const (
    name      = "<credential_request_name>"
    namespace = "<namespace_name>"
)

var CredentialsRequestTemplate = &minterv1.CredentialsRequest{
    ObjectMeta: metav1.ObjectMeta{
        Name:      name,
        Namespace: "openshift-cloud-credential-operator",
    },
    Spec: minterv1.CredentialsRequestSpec{
        ProviderSpec: ProviderSpec,
        SecretRef: corev1.ObjectReference{
            Name:      "<secret_name>",
            Namespace: namespace,
        },
        ServiceAccountNames: []string{
            "<service_account_name>",
        },
        CloudTokenPath: "",
    },
}

```

Alternatively, if you are starting from a **CredentialsRequest** object in YAML form (for example, as part of your Operator project code), you can handle it differently:

Example 3.3. Example **CredentialsRequest** object creation in YAML form

```

// CredentialsRequest is a struct that represents a request for credentials
type CredentialsRequest struct {
    APIVersion string `yaml:"apiVersion"`
    Kind       string `yaml:"kind"`
    Metadata   struct {
        Name      string `yaml:"name"`
        Namespace string `yaml:"namespace"`
    } `yaml:"metadata"`
    Spec struct {
        SecretRef struct {
            Name string `yaml:"name"`

```

```

    Namespace string `yaml:"namespace"`
} `yaml:"secretRef"`
ProviderSpec struct {
    APIVersion string `yaml:"apiVersion"`
    Kind       string `yaml:"kind"`
    StatementEntries []struct {
        Effect string `yaml:"effect"`
        Action []string `yaml:"action"`
        Resource string `yaml:"resource"`
    } `yaml:"statementEntries"`
    STSIAMRoleARN string `yaml:"stsiAMRoleARN"`
} `yaml:"providerSpec"`

// added new field
CloudTokenPath string `yaml:"cloudTokenPath"`
} `yaml:"spec"`
}

// ConsumeCredsRequestAddingTokenInfo is a function that takes a YAML filename
// and two strings as arguments
// It unmarshals the YAML file to a CredentialsRequest object and adds the token
// information.
func ConsumeCredsRequestAddingTokenInfo(fileName, tokenString, tokenPath
string) (*CredentialsRequest, error) {
    // open a file containing YAML form of a CredentialsRequest
    file, err := os.Open(fileName)
    if err != nil {
        return nil, err
    }
    defer file.Close()

    // create a new CredentialsRequest object
    cr := &CredentialsRequest{}

    // decode the yaml file to the object
    decoder := yaml.NewDecoder(file)
    err = decoder.Decode(cr)
    if err != nil {
        return nil, err
    }

    // assign the string to the existing field in the object
    cr.Spec.CloudTokenPath = tokenPath

    // return the modified object
    return cr, nil
}

```



NOTE

Adding a **CredentialsRequest** object to the Operator bundle is not currently supported.

- c. Add the role ARN and web identity token path to the credentials request and apply it during Operator initialization:

Example 3.4. Example applying `CredentialsRequest` object during Operator initialization

```
// apply credentialsRequest on install
credReq := credreq.CredentialsRequestTemplate
credReq.Spec.CloudTokenPath = webIdentityTokenPath

c := mgr.GetClient()
if err := c.Create(context.TODO(), credReq); err != nil {
    if !errors.IsAlreadyExists(err) {
        setupLog.Error(err, "unable to create CredRequest")
        os.Exit(1)
    }
}
```

- d. Ensure your Operator can wait for a **Secret** object to show up from the CCO, as shown in the following example, which is called along with the other items you are reconciling in your Operator:

Example 3.5. Example wait for `Secret` object

```
// WaitForSecret is a function that takes a Kubernetes client, a namespace, and a v1
// "k8s.io/api/core/v1" name as arguments
// It waits until the secret object with the given name exists in the given namespace
// It returns the secret object or an error if the timeout is exceeded
func WaitForSecret(client kubernetes.Interface, namespace, name string)
(*v1.Secret, error) {
    // set a timeout of 10 minutes
    timeout := time.After(10 * time.Minute) ❶

    // set a polling interval of 10 seconds
    ticker := time.NewTicker(10 * time.Second)

    // loop until the timeout or the secret is found
    for {
        select {
        case <-timeout:
            // timeout is exceeded, return an error
            return nil, fmt.Errorf("timed out waiting for secret %s in namespace %s", name,
namespace)
            // add to this error with a pointer to instructions for following a manual path to a
            Secret that will work on STS
        case <-ticker.C:
            // polling interval is reached, try to get the secret
            secret, err := client.CoreV1().Secrets(namespace).Get(context.Background(),
name, metav1.GetOptions{})
            if err != nil {
                if errors.IsNotFound(err) {
                    // secret does not exist yet, continue waiting
                    continue
                } else {
                    // some other error occurred, return it
                    return nil, err
                }
            }
        }
    }
}
```



```

    }
  } else {
    // secret is found, return it
    return secret, nil
  }
}
}
}

```

1

The **timeout** value is based on an estimate of how fast the CCO might detect an added **CredentialsRequest** object and generate a **Secret** object. You might consider lowering the time or creating custom feedback for cluster administrators that could be wondering why the Operator is not yet accessing the cloud resources.

- e. Set up the AWS configuration by reading the secret created by the CCO from the credentials request and creating the AWS config file containing the data from that secret:

Example 3.6. Example AWS configuration creation

```

func SharedCredentialsFileFromSecret(secret *corev1.Secret) (string, error) {
    var data []byte
    switch {
    case len(secret.Data["credentials"]) > 0:
        data = secret.Data["credentials"]
    default:
        return "", errors.New("invalid secret for aws credentials")
    }

    f, err := ioutil.TempFile("", "aws-shared-credentials")
    if err != nil {
        return "", errors.Wrap(err, "failed to create file for shared credentials")
    }
    defer f.Close()
    if _, err := f.Write(data); err != nil {
        return "", errors.Wrapf(err, "failed to write credentials to %s", f.Name())
    }
    return f.Name(), nil
}

```

IMPORTANT

The secret is assumed to exist, but your Operator code should wait and retry when using this secret to give time to the CCO to create the secret.

Additionally, the wait period should eventually time out and warn users that the OpenShift Container Platform cluster version, and therefore the CCO, might be an earlier version that does not support the **CredentialsRequest** object workflow with STS detection. In such cases, instruct users that they must add a secret by using another method.

- f. Configure the AWS SDK session, for example:

Example 3.7. Example AWS SDK session configuration

```
sharedCredentialsFile, err := SharedCredentialsFileFromSecret(secret)
if err != nil {
    // handle error
}
options := session.Options{
    SharedConfigState: session.SharedConfigEnable,
    SharedConfigFiles: []string{sharedCredentialsFile},
}
```

Additional resources

- [Cluster Operators reference page for the Cloud Credential Operator](#)

CHAPTER 4. HANDLING A MACHINE CONFIGURATION FOR HOSTED CONTROL PLANES

In a standalone OpenShift Container Platform cluster, a machine config pool manages a set of nodes. You can handle a machine configuration by using the **MachineConfigPool** custom resource (CR).

In hosted control planes, the **MachineConfigPool** CR does not exist. A node pool contains a set of compute nodes. You can handle a machine configuration by using node pools.

4.1. 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}
```

4.2. CONFIGURING NODE TUNING IN A HOSTED CLUSTER

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
```

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

**IMPORTANT**

Hosted control planes on the AWS platform 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-subscription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  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
sriov-network-operator.4.16.0-202211021237	SR-IOV Network Operator	4.16.0-202211021237	sriov-network-operator.4.16.0-202210290517
		Succeeded	

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

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

CHAPTER 5. USING FEATURE GATES IN A HOSTED CLUSTER

You can use feature gates in a hosted cluster to enable features that are not part of the default set of features. You can enable the **TechPreviewNoUpgrade** feature set by using feature gates in your hosted cluster.

5.1. ENABLING FEATURE SETS BY USING FEATURE GATES

You can enable the **TechPreviewNoUpgrade** feature set in a hosted cluster by editing the **HostedCluster** custom resource (CR) with the OpenShift CLI.

Prerequisites

- You installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HostedCluster** CR for editing on the hosting cluster by running the following command:

```
$ oc edit <hosted_cluster_name> -n <hosted_cluster_namespace>
```

2. Define the feature set by entering a value in the **featureSet** field. For example:

```
apiVersion: hypershift.openshift.io/v1beta1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name> 1
  namespace: <hosted_cluster_namespace> 2
spec:
  configuration:
    featureGate:
      featureSet: TechPreviewNoUpgrade 3
```

- 1** Specifies your hosted cluster name.
- 2** Specifies your hosted cluster namespace.
- 3** This feature set is a subset of the current Technology Preview features.



WARNING

Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. This feature set allows you to enable these Technology Preview features on test clusters, where you can fully test them. Do not enable this feature set on production clusters.

3. Save the file to apply the changes.

Verification

- Verify that the **TechPreviewNoUpgrade** feature gate is enabled in your hosted cluster by running the following command:

```
$ oc get featuregate cluster -o yaml
```

Additional resources

- [FeatureGate \[config.openshift.io/v1\]](https://config.openshift.io/v1)

CHAPTER 6. UPDATING 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.

6.1. REQUIREMENTS TO UPGRADE HOSTED CONTROL PLANES

The multicluster engine for Kubernetes Operator can manage one or more OpenShift Container Platform clusters. After you create a hosted cluster on OpenShift Container Platform, you must import your hosted cluster in the multicluster engine Operator as a managed cluster. Then, you can use the OpenShift Container Platform cluster as a management cluster.

Consider the following requirements before you start updating hosted control planes:

- You must use the bare metal platform for an OpenShift Container Platform cluster when using OpenShift Virtualization as a provider.
- You must use bare metal or OpenShift Virtualization as the cloud platform for the hosted cluster. You can find the platform type of your hosted cluster in the **spec.Platform.type** specification of the **HostedCluster** custom resource (CR).

You must upgrade the OpenShift Container Platform cluster, multicluster engine Operator, hosted cluster, and node pools by completing the following tasks:

1. Upgrade an OpenShift Container Platform cluster to the latest version. For more information, see "Updating a cluster using the web console" or "Updating a cluster using the CLI".
2. Upgrade the multicluster engine Operator to the latest version. For more information, see "Updating installed Operators".
3. Upgrade the hosted cluster and node pools from the previous OpenShift Container Platform version to the latest version. For more information, see "Updating the hosted cluster for hosted control planes" and "Updating node pools for hosted control planes".

Additional resources

- [Updating a cluster using the web console](#)
- [Updating a cluster using the CLI](#)
- [Updating installed Operators](#)

6.2. 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).

6.3. 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.

6.3.1. 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.

6.3.2. 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.

6.4. UPDATING NODE POOLS FOR HOSTED CONTROL PLANES

On hosted control planes, you can 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

- Change the **spec.release.image** value in the node pool by entering the following command:

```
$ oc patch nodepool <node_pool_name> -n <hosted_cluster_namespace> --type=merge -p '{"spec":{"nodeDrainTimeout":"60s","release":{"image":"<openshift_release_image>"}}}' 1
```

2

- 1 Replace **<node_pool_name>** and **<hosted_cluster_namespace>** with your node pool name and hosted cluster namespace, respectively.

- 2 The **<openshift_release_image>** variable specifies the new OpenShift Container Platform release image that you want to upgrade to, for example, **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**. Replace **<4.y.z>** with the supported OpenShift Container Platform version.

Verification

- To verify that the new version was rolled out, check the **.status.conditions** value in the node pool by running the following command:

```
$ oc get -n <hosted_cluster_namespace> nodepool <node_pool_name> -o yaml
```

Example output

```
status:
  conditions:
  - lastTransitionTime: "2024-05-20T15:00:40Z"
    message: 'Using release image: quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64'
    reason: AsExpected
    status: "True"
    type: ValidReleaseImage
```

- 1 Replace **<4.y.z>** with the supported OpenShift Container Platform version.

6.5. UPDATING THE HOSTED CLUSTER FOR HOSTED CONTROL PLANES

On hosted control planes, you can upgrade your version of OpenShift Container Platform by updating the hosted cluster.

Procedure

1. Add the **hypershift.openshift.io/force-upgrade-to=<openshift_release_image>** annotation to the hosted cluster by entering the following command:

```
$ oc annotate hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name>
"hypershift.openshift.io/force-upgrade-to=<openshift_release_image>" --overwrite 1 2
```

- 1 Replace **<hosted_cluster_name>** and **<hosted_cluster_namespace>** with your hosted cluster name and hosted cluster namespace, respectively.
- 2 The **<openshift_release_image>** variable specifies the new OpenShift Container Platform release image that you want to upgrade to, for example, **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**. Replace **<4.y.z>** with the supported OpenShift Container Platform version.

2. Change the **spec.release.image** value in the hosted cluster by entering the following command:

```
$ oc patch hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace> --
type=merge -p '{"spec":{"release":{"image":"<openshift_release_image>"}}}'
```

Verification

- To verify that the new version was rolled out, check the **.status.conditions** and **.status.version** values in the hosted cluster by running the following command:

```
$ oc get -n <hosted_cluster_namespace> hostedcluster <hosted_cluster_name> -o yaml
```

Example output

```
status:
  conditions:
  - lastTransitionTime: "2024-05-20T15:01:01Z"
    message: Payload loaded version="4.y.z" image="quay.io/openshift-release-dev/ocp-
release:4.y.z-x86_64" 1
    status: "True"
    type: ClusterVersionReleaseAccepted
  #...
version:
  availableUpdates: null
  desired:
    image: quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64 2
    version: 4.y.z
```

- 1** **2** Replace **<4.y.z>** with the supported OpenShift Container Platform version.

CHAPTER 7. HOSTED CONTROL PLANES OBSERVABILITY

You can gather metrics for hosted control planes by configuring metrics sets. The HyperShift Operator can create or delete monitoring dashboards in the management cluster for each hosted cluster that it manages.

7.1. CONFIGURING METRICS SETS FOR HOSTED CONTROL PLANES

Hosted control planes for Red Hat OpenShift Container Platform creates **ServiceMonitor** resources in each control plane namespace that allow a Prometheus stack to gather metrics from the control planes. The **ServiceMonitor** resources use metrics relabelings to define which metrics are included or excluded from a particular component, such as etcd or the Kubernetes API server. The number of metrics that are produced by control planes directly impacts the resource requirements of the monitoring stack that gathers them.

Instead of producing a fixed number of metrics that apply to all situations, you can configure a metrics set that identifies a set of metrics to produce for each control plane. The following metrics sets are supported:

- **Telemetry**: These metrics are needed for telemetry. This set is the default set and is the smallest set of metrics.
- **SRE**: This set includes the necessary metrics to produce alerts and allow the troubleshooting of control plane components.
- **All**: This set includes all of the metrics that are produced by standalone OpenShift Container Platform control plane components.

To configure a metrics set, set the **METRICS_SET** environment variable in the HyperShift Operator deployment by entering the following command:

```
$ oc set env -n hypershift deployment/operator METRICS_SET=All
```

7.1.1. Configuring the SRE metrics set

When you specify the **SRE** metrics set, the HyperShift Operator looks for a config map named **sre-metric-set** with a single key: **config**. The value of the **config** key must contain a set of **RelabelConfigs** that are organized by control plane component.

You can specify the following components:

- **etcd**
- **kubeAPIServer**
- **kubeControllerManager**
- **openshiftAPIServer**
- **openshiftControllerManager**
- **openshiftRouteControllerManager**
- **cvo**

- **olm**
- **catalogOperator**
- **registryOperator**
- **nodeTuningOperator**
- **controlPlaneOperator**
- **hostedClusterConfigOperator**

A configuration of the **SRE** metrics set is illustrated in the following example:

```
kubeAPIServer:
- action: "drop"
  regex: "etcd_(debugging|disk|server).*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_admission_controller_admission_latencies_seconds.*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_admission_step_admission_latencies_seconds.*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "scheduler_(e2e_scheduling_latency_microseconds|scheduling_algorithm_predicate_evaluation|scheduling_algorithm_priority_evaluation|scheduling_algorithm_preemption_evaluation|scheduling_algorithm_latency_microseconds|binding_latency_microseconds|scheduling_latency_seconds)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_(request_count|request_latencies|request_latencies_summary|dropped_requests|storage_data_key_generation_latencies_microseconds|storage_transformation_failures_total|storage_transformation_latencies_microseconds|proxy_tunnel_sync_latency_secs)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "docker_(operations|operations_latency_microseconds|operations_errors|operations_timeout)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "reflector_(items_per_list|items_per_watch|list_duration_seconds|lists_total|short_watches_total|watch_duration_seconds|watches_total)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "etcd_(helper_cache_hit_count|helper_cache_miss_count|helper_cache_entry_count|request_cache_get_latencies_summary|request_cache_add_latencies_summary|request_latencies_summary)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "transformation_(transformation_latencies_microseconds|failures_total)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "network_plugin_operations_latency_microseconds|sync_proxy_rules_latency_microseconds|rest_client"
```

```

_request_latency_seconds"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_request_duration_seconds_bucket;
(0.15|0.25|0.3|0.35|0.4|0.45|0.6|0.7|0.8|0.9|1.25|1.5|1.75|2.5|3|3.5|4.5|6|7|8|9|15|25|30|50)"
  sourceLabels: ["__name__", "le"]
kubeControllerManager:
- action: "drop"
  regex: "etcd_(debugging|disk|request|server).*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "rest_client_request_latency_seconds_(bucket|count|sum)"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "root_ca_cert_publisher_sync_duration_seconds_(bucket|count|sum)"
  sourceLabels: ["__name__"]
openshiftAPIServer:
- action: "drop"
  regex: "etcd_(debugging|disk|server).*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_admission_controller_admission_latencies_seconds_.*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_admission_step_admission_latencies_seconds_.*"
  sourceLabels: ["__name__"]
- action: "drop"
  regex: "apiserver_request_duration_seconds_bucket;
(0.15|0.25|0.3|0.35|0.4|0.45|0.6|0.7|0.8|0.9|1.25|1.5|1.75|2.5|3|3.5|4.5|6|7|8|9|15|25|30|50)"
  sourceLabels: ["__name__", "le"]
openshiftControllerManager:
- action: "drop"
  regex: "etcd_(debugging|disk|request|server).*"
  sourceLabels: ["__name__"]
openshiftRouteControllerManager:
- action: "drop"
  regex: "etcd_(debugging|disk|request|server).*"
  sourceLabels: ["__name__"]
olm:
- action: "drop"
  regex: "etcd_(debugging|disk|server).*"
  sourceLabels: ["__name__"]
catalogOperator:
- action: "drop"
  regex: "etcd_(debugging|disk|server).*"
  sourceLabels: ["__name__"]
cvo:
- action: drop
  regex: "etcd_(debugging|disk|server).*"
  sourceLabels: ["__name__"]

```

7.2. ENABLING MONITORING DASHBOARDS IN A HOSTED CLUSTER

To enable monitoring dashboards in a hosted cluster, complete the following steps:

Procedure

Procedure

1. Create the **hypershift-operator-install-flags** config map in the **local-cluster** namespace, being sure to specify the **--monitoring-dashboards** flag in the **data.installFlagsToAdd** section. For example:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: hypershift-operator-install-flags
  namespace: local-cluster
data:
  installFlagsToAdd: "--monitoring-dashboards"
  installFlagsToRemove: ""
```

2. Wait a couple of minutes for the HyperShift Operator deployment in the **hypershift** namespace to be updated to include the following environment variable:

```
- name: MONITORING_DASHBOARDS
  value: "1"
```

When monitoring dashboards are enabled, for each hosted cluster that the HyperShift Operator manages, the Operator creates a config map named **cp-`<hosted_cluster_namespace>-<hosted_cluster_name>`** in the **openshift-config-managed** namespace, where **<hosted_cluster_namespace>** is the namespace of the hosted cluster and **<hosted_cluster_name>** is the name of the hosted cluster. As a result, a new dashboard is added in the administrative console of the management cluster.

3. To view the dashboard, log in to the management cluster's console and go to the dashboard for the hosted cluster by clicking **Observe → Dashboards**.
4. Optional: To disable a monitoring dashboards in a hosted cluster, remove the **--monitoring-dashboards** flag from the **hypershift-operator-install-flags** config map. When you delete a hosted cluster, its corresponding dashboard is also deleted.

7.2.1. Dashboard customization

To generate dashboards for each hosted cluster, the HyperShift Operator uses a template that is stored in the **monitoring-dashboard-template** config map in the Operator namespace (**hypershift**). This template contains a set of Grafana panels that contain the metrics for the dashboard. You can edit the content of the config map to customize the dashboards.

When a dashboard is generated, the following strings are replaced with values that correspond to a specific hosted cluster:

Name	Description
__NAME__	The name of the hosted cluster
__NAMESPACE__	The namespace of the hosted cluster
__CONTROL_PLANE_NAMESPACE__	The namespace where the control plane pods of the hosted cluster are placed

<code>__CLUSTER_ID__</code>	The UUID of the hosted cluster, which matches the <code>_id</code> label of the hosted cluster metrics
-----------------------------	--

CHAPTER 8. HIGH AVAILABILITY FOR HOSTED CONTROL PLANES

8.1. ABOUT HIGH AVAILABILITY FOR HOSTED CONTROL PLANES

You can maintain high availability (HA) of hosted control planes by implementing the following actions:

- Recover etcd members for a hosted cluster.
- Back up and restore etcd for a hosted cluster.
- Perform a disaster recovery process for a hosted cluster.

8.1.1. Impact of the failed management cluster component

If the management cluster component fails, your workload remains unaffected. In the OpenShift Container Platform management cluster, the control plane is decoupled from the data plane to provide resiliency.

The following table covers the impact of a failed management cluster component on the control plane and the data plane. However, the table does not cover all scenarios for the management cluster component failures.

Table 8.1. Impact of the failed component on hosted control planes

Name of the failed component	Hosted control plane API status	Hosted cluster data plane status
Worker node	Available	Available
Availability zone	Available	Available
Management cluster control plane	Available	Available
Management cluster control plane and worker nodes	Not available	Available

8.2. RECOVERING AN UNHEALTHY ETCD CLUSTER

In a highly available control plane, three etcd pods run as a part of a stateful set in an etcd cluster. To recover an etcd cluster, identify unhealthy etcd pods by checking the etcd cluster health.

8.2.1. Checking the status of an etcd cluster

You can check the status of the etcd cluster health by logging into any etcd pod.

Procedure

1. Log in to an etcd pod by entering the following command:

```
$ oc rsh -n <hosted_control_plane_namespace> -c etcd <etcd_pod_name>
```

- Print the health status of an etcd cluster by entering the following command:

```
sh-4.4$ etcdctl endpoint health --cluster -w table
```

Example output

```
ENDPOINT                                HEALTH TOOK    ERROR
https://etcd-0.etcd-discovery.clusters-hosted.svc:2379 true    9.117698ms
```

8.2.2. Recovering a failing etcd pod

Each etcd pod of a 3-node cluster has its own persistent volume claim (PVC) to store its data. An etcd pod might fail because of corrupted or missing data. You can recover a failing etcd pod and its PVC.

Procedure

- To confirm that the etcd pod is failing, enter the following command:

```
$ oc get pods -l app=etcd -n <hosted_control_plane_namespace>
```

Example output

```
NAME    READY  STATUS      RESTARTS  AGE
etcd-0  2/2    Running     0          64m
etcd-1  2/2    Running     0          45m
etcd-2  1/2    CrashLoopBackOff 1 (5s ago) 64m
```

The failing etcd pod might have the **CrashLoopBackOff** or **Error** status.

- Delete the failing pod and its PVC by entering the following command:

```
$ oc delete pvc/<etcd_pvc_name> pod/<etcd_pod_name> --wait=false
```

Verification

- Verify that a new etcd pod is up and running by entering the following command:

```
$ oc get pods -l app=etcd -n <hosted_control_plane_namespace>
```

Example output

```
NAME    READY  STATUS  RESTARTS  AGE
etcd-0  2/2    Running  0          67m
etcd-1  2/2    Running  0          48m
etcd-2  2/2    Running  0          2m2s
```

8.3. BACKING UP AND RESTORING ETCD IN AN ON-PREMISE ENVIRONMENT

You can back up and restore etcd on a hosted cluster in an on-premise environment to fix failures.

8.3.1. Backing up and restoring etcd on a hosted cluster in an on-premise environment

By backing up and restoring etcd on a hosted cluster, you can fix failures, such as corrupted or missing data in an etcd member of a three node cluster. If multiple members of the etcd cluster encounter data loss or have a **CrashLoopBackOff** status, this approach helps prevent an etcd quorum loss.



IMPORTANT

This procedure requires API downtime.

Prerequisites

- The **oc** and **jq** binaries have been installed.

Procedure

1. First, set up your environment variables and scale down the API servers:
 - a. Set up environment variables for your hosted cluster by entering the following commands, replacing values as necessary:

```
$ CLUSTER_NAME=my-cluster
```

```
$ HOSTED_CLUSTER_NAMESPACE=clusters
```

```
$ CONTROL_PLANE_NAMESPACE="${HOSTED_CLUSTER_NAMESPACE}-  
${CLUSTER_NAME}"
```

- b. Pause reconciliation of the hosted cluster by entering the following command, replacing values as necessary:

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

- c. Scale down the API servers by entering the following commands:

- i. Scale down the **kube-apiserver**:

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/kube-apiserver --  
replicas=0
```

- ii. Scale down the **openshift-apiserver**:

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/openshift-apiserver -  
-replicas=0
```

- iii. Scale down the **openshift-oauth-apiserver**:

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/openshift-oauth-  
apiserver --replicas=0
```

2. Next, take a snapshot of etcd by using one of the following methods:

- a. Use a previously backed-up snapshot of etcd.
- b. If you have an available etcd pod, take a snapshot from the active etcd pod by completing the following steps:
 - i. List etcd pods by entering the following command:

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd
```

- ii. Take a snapshot of the pod database and save it locally to your machine by entering the following commands:

```
$ ETCD_POD=etcd-0
```

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} -c etcd -t ${ETCD_POD} -- env
ETCDCTL_API=3 /usr/bin/etcdctl \
--cacert /etc/etcd/tls/etcd-ca/ca.crt \
--cert /etc/etcd/tls/client/etcd-client.crt \
--key /etc/etcd/tls/client/etcd-client.key \
--endpoints=https://localhost:2379 \
snapshot save /var/lib/snapshot.db
```

- iii. Verify that the snapshot is successful by entering the following command:

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} -c etcd -t ${ETCD_POD} -- env
ETCDCTL_API=3 /usr/bin/etcdctl -w table snapshot status /var/lib/snapshot.db
```

- c. Make a local copy of the snapshot by entering the following command:

```
$ oc cp -c etcd
${CONTROL_PLANE_NAMESPACE}/${ETCD_POD}:/var/lib/snapshot.db
/tmp/etcd.snapshot.db
```

- i. Make a copy of the snapshot database from etcd persistent storage:

- A. List etcd pods by entering the following command:

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd
```

- B. Find a pod that is running and set its name as the value of **ETCD_POD**:
ETCD_POD=etcd-0, and then copy its snapshot database by entering the following command:

```
$ oc cp -c etcd
${CONTROL_PLANE_NAMESPACE}/${ETCD_POD}:/var/lib/data/member/snap/
db /tmp/etcd.snapshot.db
```

3. Next, scale down the etcd statefulset by entering the following command:

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd --replicas=0
```

- a. Delete volumes for second and third members by entering the following command:

```
$ oc delete -n ${CONTROL_PLANE_NAMESPACE} pvc/data-etcd-1 pvc/data-etcd-2
```

b. Create a pod to access the first etcd member's data:

i. Get the etcd image by entering the following command:

```
$ ETCD_IMAGE=$(oc get -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd -
o jsonpath='{ .spec.template.spec.containers[0].image }')
```

ii. Create a pod that allows access to etcd data:

```
$ cat << EOF | oc apply -n ${CONTROL_PLANE_NAMESPACE} -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: etcd-data
spec:
  replicas: 1
  selector:
    matchLabels:
      app: etcd-data
  template:
    metadata:
      labels:
        app: etcd-data
    spec:
      containers:
        - name: access
          image: $ETCD_IMAGE
          volumeMounts:
            - name: data
              mountPath: /var/lib
          command:
            - /usr/bin/bash
          args:
            - -C
            - |-
              while true; do
                sleep 1000
              done
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: data-etcd-0
EOF
```

iii. Check the status of the **etcd-data** pod and wait for it to be running by entering the following command:

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd-data
```

iv. Get the name of the **etcd-data** pod by entering the following command:

```
$ DATA_POD=$(oc get -n ${CONTROL_PLANE_NAMESPACE} pods --no-headers  
-l app=etcd-data -o name | cut -d/ -f2)
```

- c. Copy an etcd snapshot into the pod by entering the following command:

```
$ oc cp /tmp/etcd.snapshot.db  
${CONTROL_PLANE_NAMESPACE}/${DATA_POD}:/var/lib/restored.snap.db
```

- d. Remove old data from the **etcd-data** pod by entering the following commands:

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- rm -rf /var/lib/data  
  
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- mkdir -p  
/var/lib/data
```

- e. Restore the etcd snapshot by entering the following command:

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- etcdctl snapshot  
restore /var/lib/restored.snap.db \  
--data-dir=/var/lib/data --skip-hash-check \  
--name etcd-0 \  
--initial-cluster-token=etcd-cluster \  
--initial-cluster etcd-0=https://etcd-0.etcd-  
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380,etcd-1=https://etcd-1.etcd-  
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380,etcd-2=https://etcd-2.etcd-  
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380 \  
--initial-advertise-peer-urls https://etcd-0.etcd-  
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380
```

- f. Remove the temporary etcd snapshot from the pod by entering the following command:

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- rm  
/var/lib/restored.snap.db
```

- g. Delete data access deployment by entering the following command:

```
$ oc delete -n ${CONTROL_PLANE_NAMESPACE} deployment/etcd-data
```

- h. Scale up the etcd cluster by entering the following command:

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd --replicas=3
```

- i. Wait for the etcd member pods to return and report as available by entering the following command:

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd -w
```

- j. Scale up all etcd-writer deployments by entering the following command:

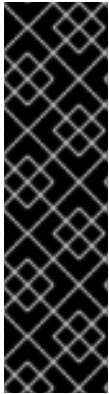
```
$ oc scale deployment -n ${CONTROL_PLANE_NAMESPACE} --replicas=3 kube-  
apiserver openshift-apiserver openshift-oauth-apiserver
```


4. Restore reconciliation of the hosted cluster by entering the following command:

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

8.4. BACKING UP AND RESTORING ETCD ON AWS

You can back up and restore etcd on a hosted cluster on Amazon Web Services (AWS) to fix failures.



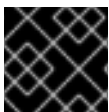
IMPORTANT

Hosted control planes on the AWS platform 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](#).

8.4.1. Taking a snapshot of etcd for a hosted cluster

To back up etcd for a hosted cluster, you must take a snapshot of etcd. Later, you can restore etcd by using the snapshot.



IMPORTANT

This procedure requires API downtime.

Procedure

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

```
$ oc patch -n clusters hostedclusters/<hosted_cluster_name> -p '{"spec": {"pausedUntil":"true"}}' --type=merge
```

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

```
$ oc scale deployment -n <hosted_cluster_namespace> --replicas=0 kube-apiserver  
openshift-apiserver openshift-oauth-apiserver
```

3. To take an etcd snapshot, use the **exec** command in each etcd container by entering the following command:

```
$ oc exec -it <etcd_pod_name> -n <hosted_cluster_namespace> -- env ETCDCTL_API=3  
/usr/bin/etcdctl --cacert /etc/etcd/tls/etcd-ca/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
```

4. To check the snapshot status, use the **exec** command in each etcd container by running the following command:

■

```
$ oc exec -it <etcd_pod_name> -n <hosted_cluster_namespace> -- env ETCDCTL_API=3
/usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db
```

5. Copy the snapshot data to a location where you can retrieve it later, such as an S3 bucket. See 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, when copying the snapshot to an S3 bucket, the upload fails.

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
```

6. To restore the snapshot on a new cluster later, save the encryption secret that the hosted cluster references.
 - a. Get the secret encryption key by entering the following command:

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

- b. Save the secret encryption key by entering the following command:

```
$ oc get secret <hosted_cluster_name>-etcd-encryption-key -o=jsonpath='{.data.key}'
```

You can decrypt this key when restoring a snapshot on a new cluster.

Next steps

Restore the etcd snapshot.

8.4.2. 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.

8.5. DISASTER RECOVERY FOR A HOSTED CLUSTER IN AWS

You can recover a hosted cluster to the same region within Amazon Web Services (AWS). For example, you need disaster recovery 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 disaster recovery process involves the following 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 does 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. That way, the server URL ends with <https://api-sample-hosted.sample-hosted.aws.openshift.com>. See the following example:

Example: External DNS flags

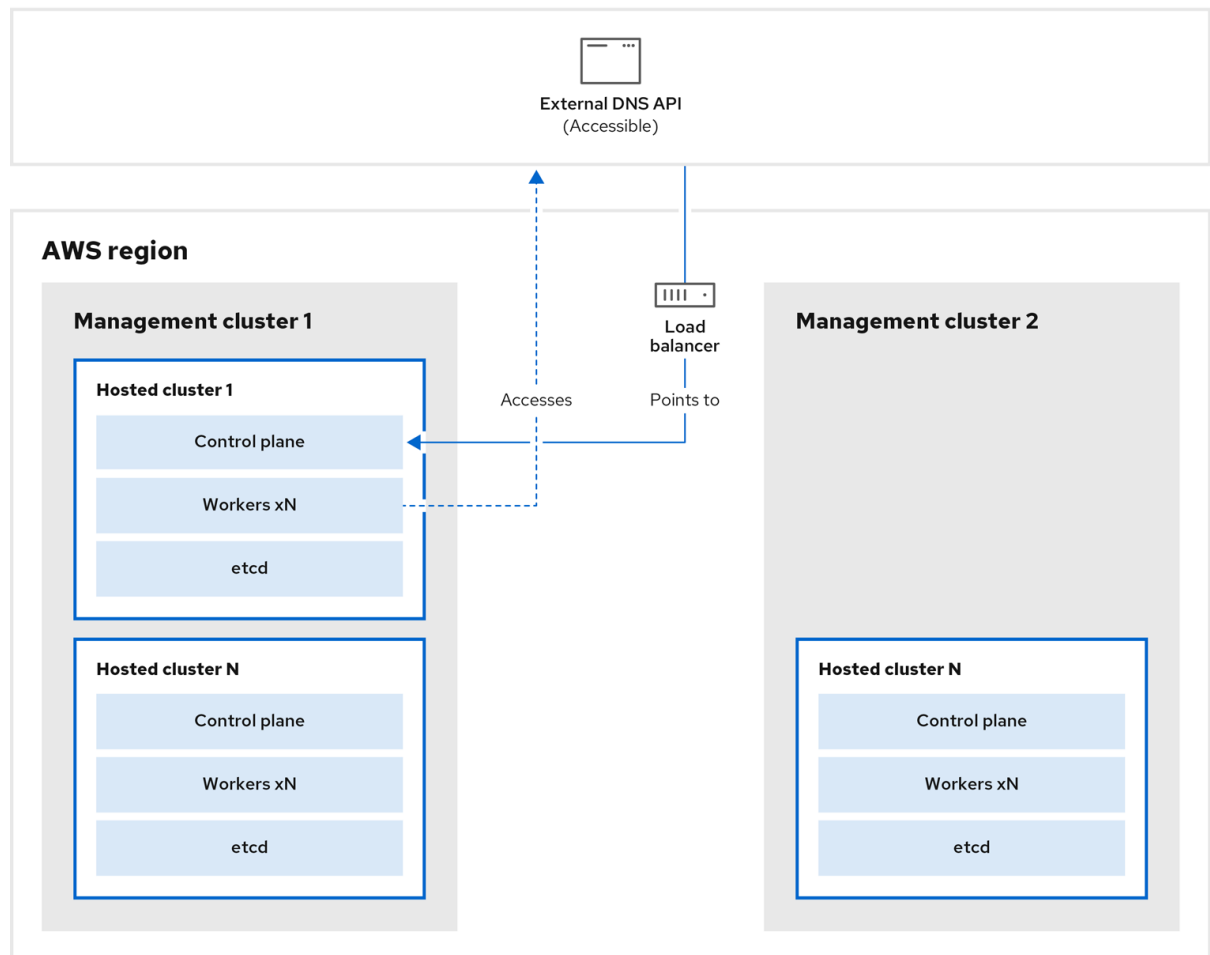
```
--external-dns-provider=aws \  
--external-dns-credentials=<path_to_aws_credentials_file> \  
--external-dns-domain-filter=<basedomain>
```

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

8.5.1. Overview of the backup and restore process

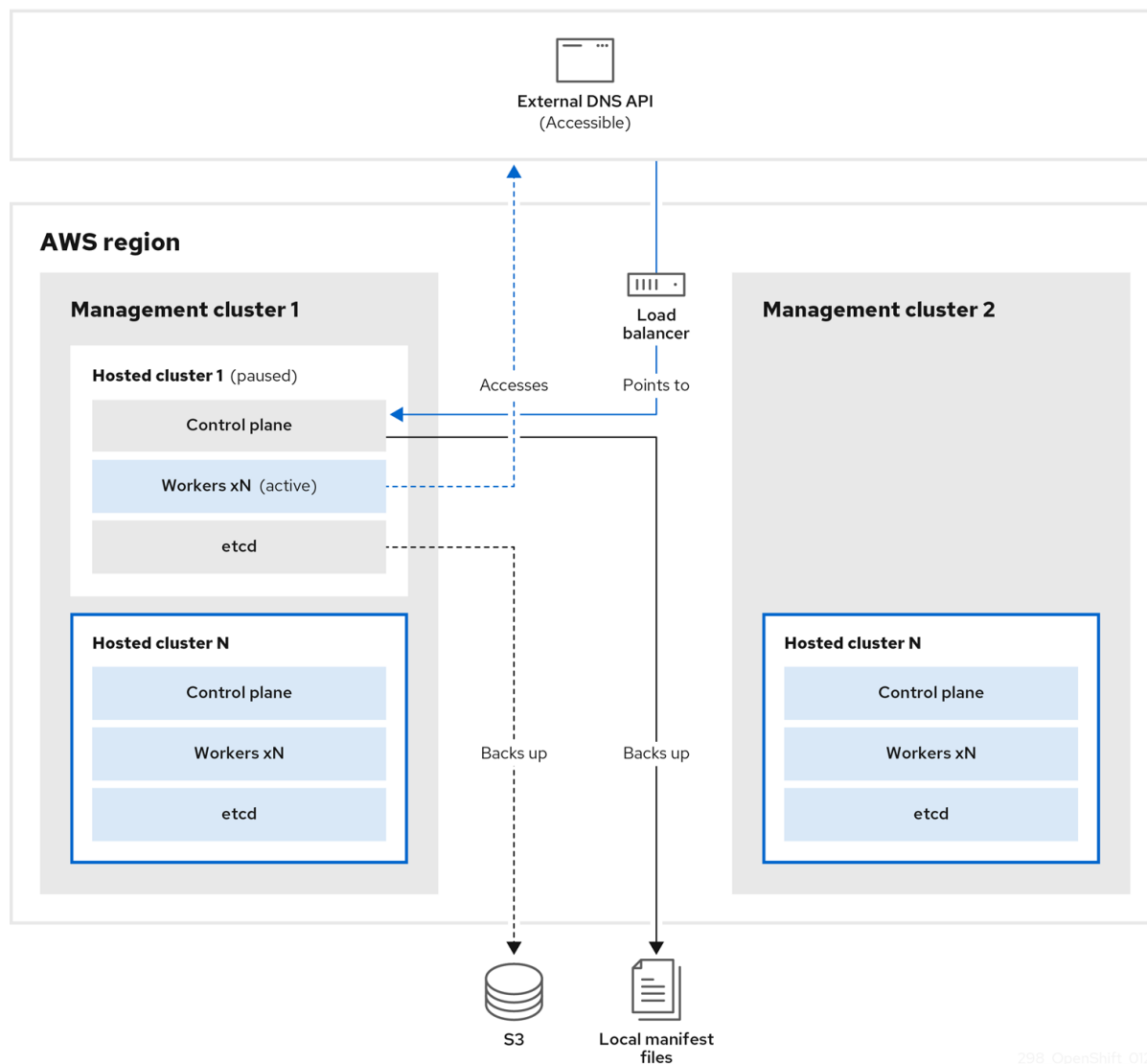
The backup and restore process works as follows:

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.

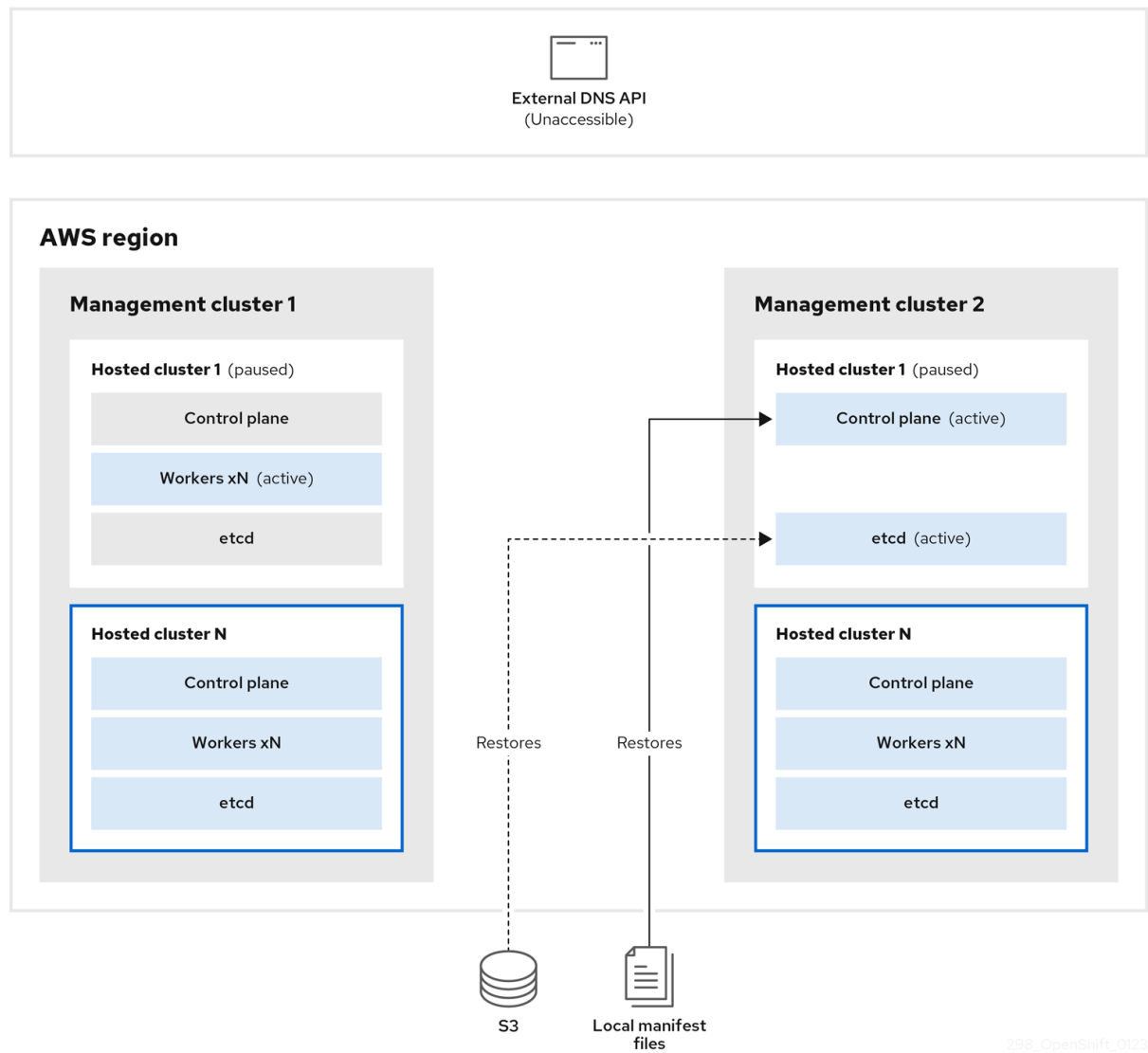


298_OpenShift_0123

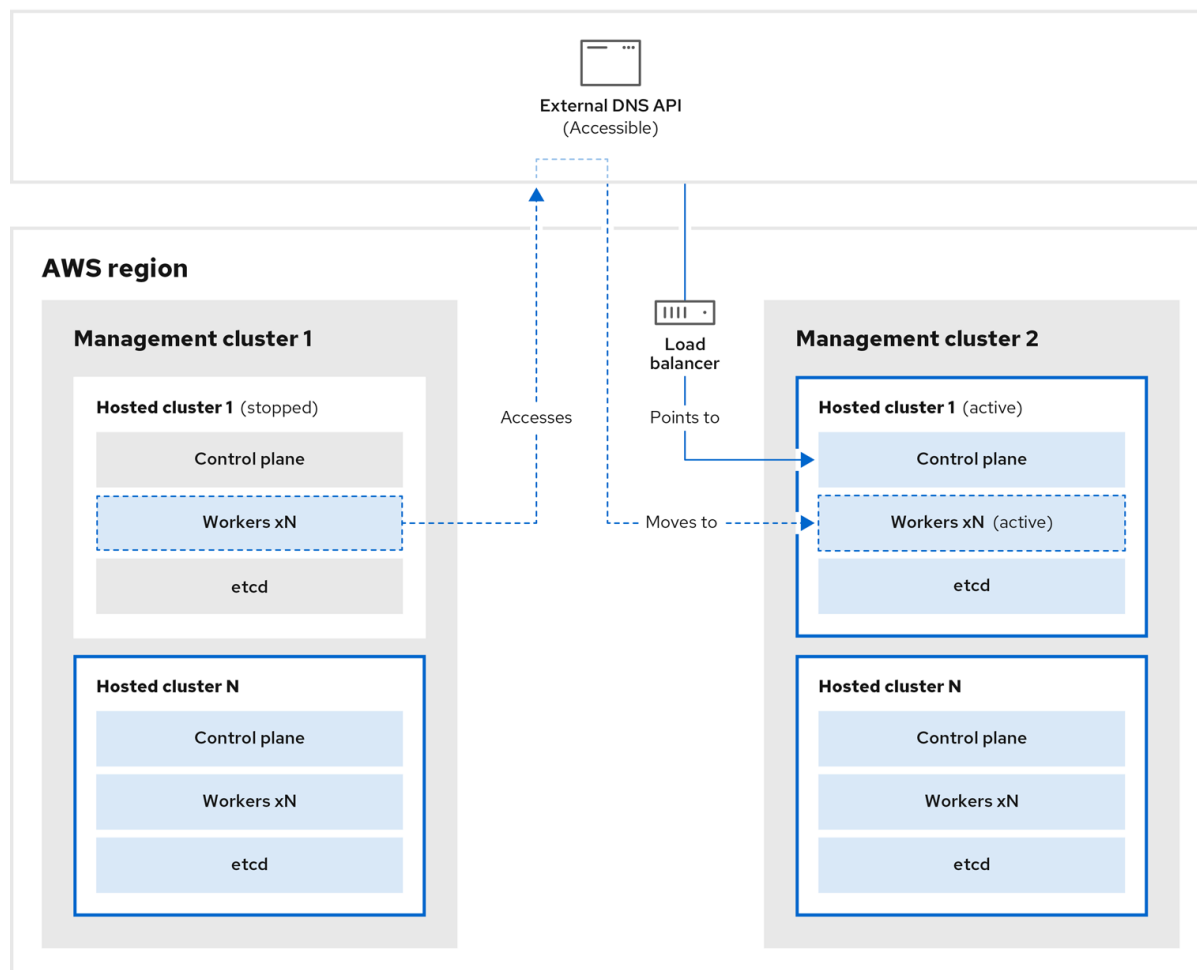
2. You take a snapshot of the hosted cluster, which includes etcd, the control plane, and the worker nodes. During this process, the worker nodes continue to try to access the external DNS API even if it is not accessible, the workloads are running, the control plane is saved in a local manifest file,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



- 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.

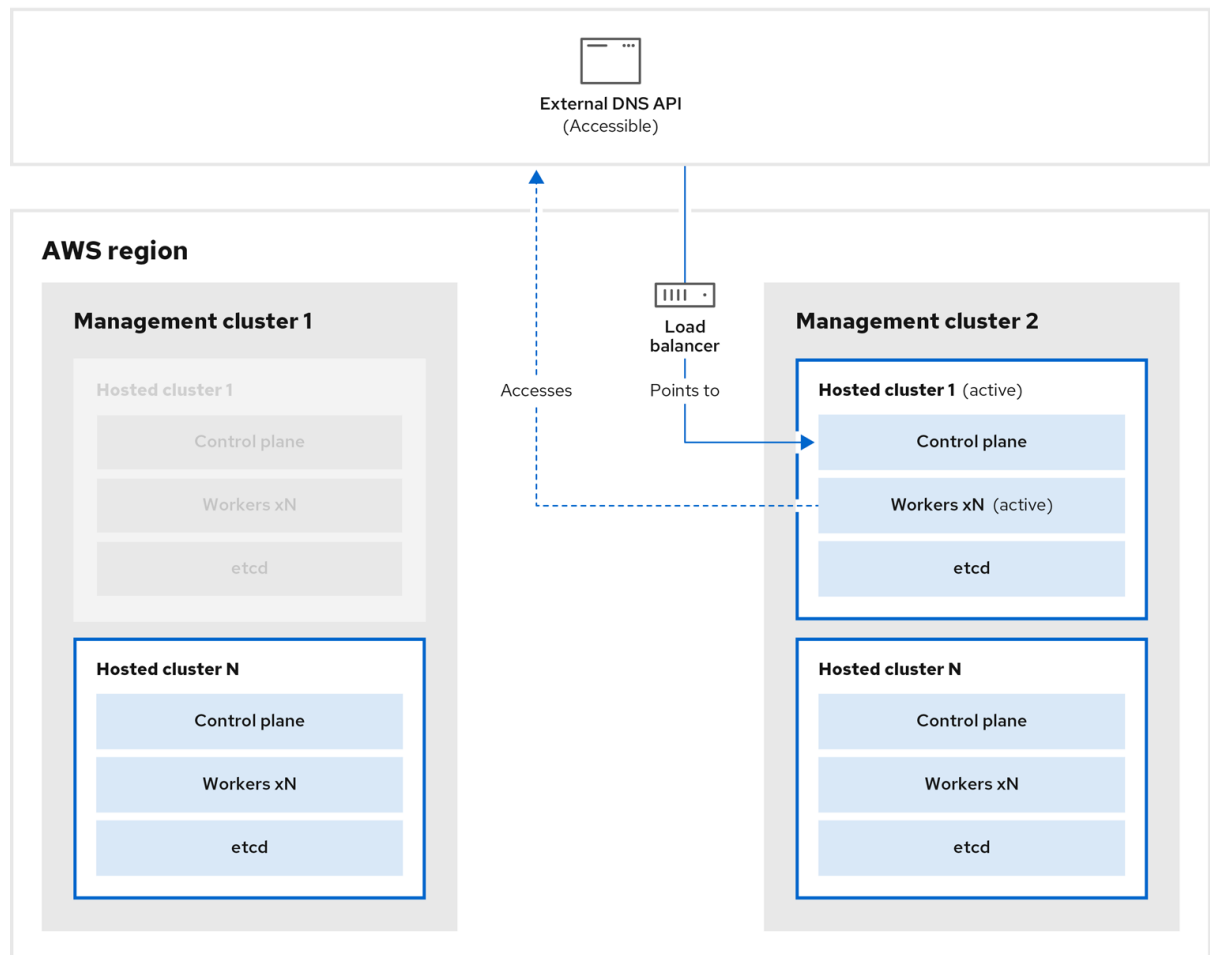


4. The external DNS API is accessible again, and the worker nodes use it to move to management cluster 2. The external DNS API can access the load balancer that points to the control plane.



298_OpenShift_0123

- 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 management cluster 1, it will not work.



298_OpenShift_0123

8.5.2. 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.

8.5.3. 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.

8.5.4. 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:



IMPORTANT

Do not scale the stateful set if the value of its **spec.persistentVolumeClaimRetentionPolicy.whenScaled** field is set to **Delete**, because this could lead to a loss of data.

As a workaround, update the value of the **spec.persistentVolumeClaimRetentionPolicy.whenScaled** field to **Retain**. Ensure that no controllers exist that reconcile the stateful set and would return the value back to **Delete**, which could lead to a loss of data.

```
# 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
```

8.6. DISASTER RECOVERY FOR A HOSTED CLUSTER BY USING OADP

You can use the OpenShift API for Data Protection (OADP) Operator to perform disaster recovery on Amazon Web Services (AWS) and bare metal.

The disaster recovery process with OpenShift API for Data Protection (OADP) involves the following steps:

1. Preparing your platform, such as Amazon Web Services or bare metal, to use OADP
2. Backing up the data plane workload
3. Backing up the control plane workload
4. Restoring a hosted cluster by using OADP

8.6.1. Prerequisites

You must meet the following prerequisites on the management cluster:

- You [installed the OADP Operator](#).
- You created a storage class.
- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OADP subscription through a catalog source.
- You have access to a cloud storage provider that is compatible with OADP, such as S3, Microsoft Azure, Google Cloud Platform, or MinIO.
- In a disconnected environment, you have access to a self-hosted storage provider, for example [Red Hat OpenShift Data Foundation](#) or [MinIO](#), that is compatible with OADP.

- Your hosted control planes pods are up and running.

8.6.2. Preparing AWS to use OADP

To perform disaster recovery for a hosted cluster, you can use OpenShift API for Data Protection (OADP) on Amazon Web Services (AWS) S3 compatible storage. After creating the **DataProtectionApplication** object, new **velero** deployment and **node-agent** pods are created in the **openshift-adp** namespace.

To prepare AWS to use OADP, see "Configuring the OpenShift API for Data Protection with Multicloud Object Gateway".

Additional resources

- [Configuring the OpenShift API for Data Protection with Multicloud Object Gateway](#)

Next steps

- Backing up the data plane workload
- Backing up the control plane workload

8.6.3. Preparing bare metal to use OADP

To perform disaster recovery for a hosted cluster, you can use OpenShift API for Data Protection (OADP) on bare metal. After creating the **DataProtectionApplication** object, new **velero** deployment and **node-agent** pods are created in the **openshift-adp** namespace.

To prepare bare metal to use OADP, see "Configuring the OpenShift API for Data Protection with AWS S3 compatible storage".

Additional resources

- [Configuring the OpenShift API for Data Protection with AWS S3 compatible storage](#)

Next steps

- Backing up the data plane workload
- Backing up the control plane workload

8.6.4. Backing up the data plane workload

If the data plane workload is not important, you can skip this procedure. To back up the data plane workload by using the OADP Operator, see "Backing up applications".

Additional resources

- [Backing up applications](#)

Next steps

- Restoring a hosted cluster by using OADP

8.6.5. Backing up the control plane workload

You can back up the control plane workload by creating the **Backup** custom resource (CR).

To monitor and observe the backup process, see "Observing the backup and restore process".

Procedure

1. Scale down the **NodePool** replicas to **0** by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  scale nodepool -n <hosted_cluster_namespace> \
  <node_pool_name> --replicas 0
```

2. Pause the reconciliation of the **HostedCluster** resource by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "true"}]'
```

3. Pause the reconciliation of the **NodePool** resource by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch nodepool -n <hosted_cluster_namespace> <node_pool_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "true"}]'
```

4. Create a YAML file that defines the **Backup** CR:

Example 8.1. Example backup-control-plane.yaml file

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup_resource_name> ❶
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
spec:
  hooks: {}
  includedNamespaces: ❷
  - <hosted_cluster_namespace> ❸
  - <hosted_control_plane_namespace> ❹
  includedResources:
    - sa
    - role
    - rolebinding
    - pod
    - pvc
    - pv
    - bmh
    - configmap
    - infraenv ❺
    - priorityclasses
    - pdb
```

```

- agents
- hostedcluster
- nodepool
- secrets
- services
- deployments
- hostedcontrolplane
- cluster
- agentcluster
- agentmachinetemplate
- agentmachine
- machinedeployment
- machineset
- machine
excludedResources: []
storageLocation: default
ttl: 2h0m0s
snapshotMoveData: true 6
datamover: "velero" 7
defaultVolumesToFsBackup: true 8

```

- 1 Replace **backup_resource_name** with the name of your **Backup** resource.
- 2 Selects specific namespaces to back up objects from them. You must include your hosted cluster namespace and the hosted control plane namespace.
- 3 Replace **<hosted_cluster_namespace>** with the name of the hosted cluster namespace, for example, **clusters**.
- 4 Replace **<hosted_control_plane_namespace>** with the name of the hosted control plane namespace, for example, **clusters-hosted**.
- 5 You must create the **infraenv** resource in a separate namespace. Do not delete the **infraenv** resource during the backup process.
- 6 7 Enables the CSI volume snapshots and uploads the control plane workload automatically to the cloud storage.
- 8 Sets the **fs-backup** backing up method for persistent volumes (PVs) as default. This setting is useful when you use a combination of Container Storage Interface (CSI) volume snapshots and the **fs-backup** method.



NOTE

If you want to use CSI volume snapshots, you must add the **backup.velero.io/backup-volumes-excludes=<pv_name>** annotation to your PVs.

5. Apply the **Backup** CR by running the following command:

```
$ oc apply -f backup-control-plane.yaml
```

Verification

- Verify if the value of the **status.phase** is **Completed** by running the following command:

```
$ oc get backup <backup_resource_name> -n openshift-adp -o jsonpath='{.status.phase}'
```

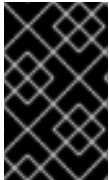
Next steps

- Restoring a hosted cluster by using OADP

8.6.6. Restoring a hosted cluster by using OADP

You can restore the hosted cluster by creating the **Restore** custom resource (CR).

- If you are using an *in-place* update, InfraEnv does not need spare nodes. You need to re-provision the worker nodes from the new management cluster.
- If you are using a *replace* update, you need some spare nodes for InfraEnv to deploy the worker nodes.



IMPORTANT

After you back up your hosted cluster, you must destroy it to initiate the restoring process. To initiate node provisioning, you must back up workloads in the data plane before deleting the hosted cluster.

Prerequisites

- You completed the steps in [Removing a cluster by using the console](#) to delete your hosted cluster.
- You completed the steps in [Removing remaining resources after removing a cluster](#).

To monitor and observe the backup process, see "Observing the backup and restore process".

Procedure

1. Verify that no pods and persistent volume claims (PVCs) are present in the hosted control plane namespace by running the following command:

```
$ oc get pod pvc -n <hosted_control_plane_namespace>
```

Expected output

```
No resources found
```

2. Create a YAML file that defines the **Restore** CR:

Example restore-hosted-cluster.yaml file

```
apiVersion: velero.io/v1
kind: Restore
metadata:
```

```

name: <restore_resource_name> ❶
namespace: openshift-adp
spec:
  backupName: <backup_resource_name> ❷
  restorePVs: true ❸
  existingResourcePolicy: update ❹
  excludedResources:
    - nodes
    - events
    - events.events.k8s.io
    - backups.velero.io
    - restores.velero.io
    - resticrepositories.velero.io

```

- ❶ Replace **<restore_resource_name>** with the name of your **Restore** resource.
- ❷ Replace **<backup_resource_name>** with the name of your **Backup** resource.
- ❸ Initiates the recovery of persistent volumes (PVs) and its pods.
- ❹ Ensures that the existing objects are overwritten with the backed up content.



IMPORTANT

You must create the **infraenv** resource in a separate namespace. Do not delete the **infraenv** resource during the restore process. The **infraenv** resource is mandatory for the new nodes to be reprovisioned.

3. Apply the **Restore** CR by running the following command:

```
$ oc apply -f restore-hosted-cluster.yaml
```

4. Verify if the value of the **status.phase** is **Completed** by running the following command:

```
$ oc get hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace> -o
jsonpath='{.status.phase}'
```

5. After the restore process is complete, start the reconciliation of the **HostedCluster** and **NodePool** resources that you paused during backing up of the control plane workload:

- a. Start the reconciliation of the **HostedCluster** resource by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "false"}]'
```

- b. Start the reconciliation of the **NodePool** resource by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch nodepool -n <hosted_cluster_namespace> <node_pool_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "false"}]'
```


6. Scale the **NodePool** resource to the desired number of replicas by running the following command:

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  scale nodepool -n <hosted_cluster_namespace> <node_pool_name> \
  --replicas <replica_count> 1
```

- 1** Replace **<replica_count>** by an integer value, for example, **3**.

8.6.7. Observing the backup and restore process

When using OpenShift API for Data Protection (OADP) to backup and restore a hosted cluster, you can monitor and observe the process.

Procedure

1. Observe the backup process by running the following command:

```
$ watch "oc get backup -n openshift-adp <backup_resource_name> -o jsonpath='{.status}'"
```

2. Observe the restore process by running the following command:

```
$ watch "oc get restore -n openshift-adp <backup_resource_name> -o jsonpath='{.status}'"
```

3. Observe the Velero logs by running the following command:

```
$ oc logs -n openshift-adp -ldeploy=velero -f
```

4. Observe the progress of all of the OADP objects by running the following command:

```
$ watch "echo BackupRepositories::echo;oc get backuprepositories.velero.io -A;echo; echo
BackupStorageLocations: ;echo; oc get backupstoragelocations.velero.io -A;echo;echo
DataUploads: ;echo;oc get datauploads.velero.io -A;echo;echo DataDownloads: ;echo;oc get
datadownloads.velero.io -n openshift-adp; echo;echo VolumeSnapshotLocations: ;echo;oc
get volumesnapshotlocations.velero.io -A;echo;echo Backups::echo;oc get backup -A;
echo;echo Restores::echo;oc get restore -A"
```

8.6.8. Using the velero CLI to describe the Backup and Restore resources

When using OpenShift API for Data Protection, you can get more details of the **Backup** and **Restore** resources by using the **velero** command-line interface (CLI).

Procedure

1. Create an alias to use the **velero** CLI from a container by running the following command:

```
$ alias velero='oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'
```

2. Get details of your **Restore** custom resource (CR) by running the following command:

```
$ velero restore describe <restore_resource_name> --details 1
```

1 Replace `<restore_resource_name>` with the name of your **Restore** resource.

3. Get details of your **Backup** CR by running the following command:

```
$ velero restore describe <backup_resource_name> --details 1
```

1 Replace `<backup_resource_name>` with the name of your **Backup** resource.

CHAPTER 9. TROUBLESHOOTING HOSTED CONTROL PLANES

If you encounter issues with hosted control planes, see the following information to guide you through troubleshooting.

9.1. GATHERING INFORMATION TO TROUBLESHOOT HOSTED CONTROL PLANES

When you need to troubleshoot an issue with hosted control plane clusters, you can gather information by running the **must-gather** command. The command generates output for the management cluster and the hosted cluster.

The output for the management cluster contains the following content:

- **Cluster-scoped resources:** These resources are node definitions of the management cluster.
- **The hypershift-dump compressed file:** This file is useful if you need to share the content with other people.
- **Namespaced resources:** These resources include all of the objects from the relevant namespaces, such as config maps, services, events, and logs.
- **Network logs:** These logs include the OVN northbound and southbound databases and the status for each one.
- **Hosted clusters:** This level of output involves all of the resources inside of the hosted cluster.

The output for the hosted cluster contains the following content:

- **Cluster-scoped resources:** These resources include all of the cluster-wide objects, such as nodes and CRDs.
- **Namespaced resources:** These resources include all of the objects from the relevant namespaces, such as config maps, services, events, and logs.

Although the output does not contain any secret objects from the cluster, it can contain references to the names of secrets.

Prerequisites

- You must have **cluster-admin** access to the management cluster.
- You need the **name** value for the **HostedCluster** resource and the namespace where the CR is deployed.
- You must have the **hcp** command line interface installed. For more information, see [Installing the hosted control planes command line interface](#).
- You must have the OpenShift CLI (**oc**) installed.
- You must ensure that the **kubeconfig** file is loaded and is pointing to the management cluster.

Procedure

- To gather the output for troubleshooting, enter the following command:

```
$ oc adm must-gather --image=registry.redhat.io/multicloud-engine/must-gather-
rhel9:v<mce_version> \
  /usr/bin/gather hosted-cluster-namespace=HOSTEDCLUSTERNAMESPACE hosted-
cluster-name=HOSTEDCLUSTERNAME \
  --dest-dir=NAME ; tar -cvzf NAME.tgz NAME
```

where:

- You replace **<mce_version>** with the version of multicloud engine Operator that you are using; for example, **2.6**.
- The **hosted-cluster-namespace=HOSTEDCLUSTERNAMESPACE** parameter is optional. If you do not include it, the command runs as though the hosted cluster is in the default namespace, which is **clusters**.
- The **--dest-dir=NAME** parameter is optional. Specify that parameter if you want to save the results of the command to a compressed file, replacing **NAME** with the name of the directory where you want to save the results.

9.2. RESTARTING HOSTED CONTROL PLANE COMPONENTS

If you are an administrator for hosted control planes, you can use the **hypershift.openshift.io/restart-date** annotation to restart all control plane components for a particular **HostedCluster** resource. For example, you might need to restart control plane components for certificate rotation.

Procedure

To restart a control plane, annotate the **HostedCluster** resource by entering the following command:

```
$ oc annotate hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name>
hypershift.openshift.io/restart-date=$(date --iso-8601=seconds)
```

Verification

The control plane is restarted whenever the value of the annotation changes. The **date** command in the example serves as the source of a unique string. The annotation is treated as a string, not a timestamp.

The following components are restarted:

- catalog-operator
- certified-operators-catalog
- cluster-api
- cluster-autoscaler
- cluster-policy-controller
- cluster-version-operator
- community-operators-catalog
- control-plane-operator

- hosted-cluster-config-operator
- ignition-server
- ingress-operator
- konnectivity-agent
- konnectivity-server
- kube-apiserver
- kube-controller-manager
- kube-scheduler
- machine-approver
- oauth-openshift
- olm-operator
- openshift-apiserver
- openshift-controller-manager
- openshift-oauth-apiserver
- packageserver
- redhat-marketplace-catalog
- redhat-operators-catalog

9.3. PAUSING THE RECONCILIATION OF A HOSTED CLUSTER AND HOSTED CONTROL PLANE

If you are a cluster instance administrator, you can pause the reconciliation of a hosted cluster and hosted control plane. You might want to pause reconciliation when you back up and restore an etcd database or when you need to debug problems with a hosted cluster or hosted control plane.

Procedure

1. To pause reconciliation for a hosted cluster and hosted control plane, populate the **pausedUntil** field of the **HostedCluster** resource.

- To pause the reconciliation until a specific time, enter the following command:

```
$ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p
'{"spec":{"pausedUntil":"<timestamp>"}}' --type=merge 1
```

- 1 Specify a timestamp in the RFC339 format, for example, **2024-03-03T03:28:48Z**. The reconciliation is paused until the specified time is passed.

- To pause the reconciliation indefinitely, enter the following command:

```
$ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p
'{"spec":{"pausedUntil":"true"}}' --type=merge
```

The reconciliation is paused until you remove the field from the **HostedCluster** resource.

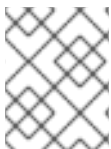
When the pause reconciliation field is populated for the **HostedCluster** resource, the field is automatically added to the associated **HostedControlPlane** resource.

2. To remove the **pausedUntil** field, enter the following patch command:

```
$ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p
'{"spec":{"pausedUntil":null}}' --type=merge
```

9.4. SCALING DOWN THE DATA PLANE TO ZERO

If you are not using the hosted control plane, to save the resources and cost you can scale down a data plane to zero.



NOTE

Ensure you are prepared to scale down the data plane to zero. Because the workload from the worker nodes disappears after scaling down.

Procedure

1. Set the **kubeconfig** file to access the hosted cluster by running the following command:

```
$ export KUBECONFIG=<install_directory>/auth/kubeconfig
```

2. Get the name of the **NodePool** resource associated to your hosted cluster by running the following command:

```
$ oc get nodepool --namespace <HOSTED_CLUSTER_NAMESPACE>
```

3. Optional: To prevent the pods from draining, add the **nodeDrainTimeout** field in the **NodePool** resource by running the following command:

```
$ oc edit NodePool <nodepool> -o yaml --namespace
<HOSTED_CLUSTER_NAMESPACE>
```

Example output

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  # ...
  name: nodepool-1
  namespace: clusters
  # ...
spec:
  arch: amd64
  clusterName: clustername 1
```

```
management:
  autoRepair: false
  replace:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    strategy: RollingUpdate
  upgradeType: Replace
nodeDrainTimeout: 0s 2
# ...
```

- 1 Defines the name of your hosted cluster.
- 2 Specifies the total amount of time that the controller spends to drain a node. By default, the **nodeDrainTimeout: 0s** setting blocks the node draining process.



NOTE

To allow the node draining process to continue for a certain period of time, you can set the value of the **nodeDrainTimeout** field accordingly, for example, **nodeDrainTimeout: 1m**.

4. Scale down the **NodePool** resource associated to your hosted cluster by running the following command:

```
$ oc scale nodepool/<NODEPOOL_NAME> --namespace
<HOSTED_CLUSTER_NAMESPACE> --replicas=0
```



NOTE

After scaling down the data plan to zero, some pods in the control plane stay in the **Pending** status and the hosted control plane stays up and running. If necessary, you can scale up the **NodePool** resource.

5. Optional: Scale up the **NodePool** resource associated to your hosted cluster by running the following command:

```
$ oc scale nodepool/<NODEPOOL_NAME> --namespace
<HOSTED_CLUSTER_NAMESPACE> --replicas=1
```

After rescaling the **NodePool** resource, wait for couple of minutes for the **NodePool** resource to become available in a **Ready** state.

Verification

- Verify that the value for the **nodeDrainTimeout** field is greater than **0s** by running the following command:

```
$ oc get nodepool -n <hosted_cluster_namespace> <nodepool_name> -
  ojsonpath='{.spec.nodeDrainTimeout}'
```

Additional resources

- [Must-gather for a hosted cluster](#)