OpenShift Container Platform 4.15

托管 control plane

在 OpenShift Container Platform 中使用托管 control plane

Last Updated: 2024-07-23

OpenShift Container Platform 4.15 托管 control plane

在 OpenShift Container Platform 中使用托管 control plane

法律通告

摘要

本文档提供在 OpenShift Container Platform 管理托管 control plane 的说明。使用托管 control plane，您可以在托管集群中创建 pod 作为 control plane，而无需为每个 control plane 都使用专用的物理或虚拟机。

目录

第 1 章 托管 CONTROL PLANE 概述

您可以使用两个不同的 control plane 配置部署 OpenShift Container Platform 集群：独立或托管的 control plane。独立配置使用专用虚拟机或物理机器来托管 control plane。通过为 OpenShift Container Platform 托管 control plane，您可以在托管集群中创建 pod 作为 control plane，而无需为每个 control plane 使用专用虚拟机或物理机器。

1.1. 托管控制平面（CONTROL PLANE）的常见概念和用户角色表

当使用托管的 control plane 用于 OpenShift Container Platform 时，了解其关键概念和涉及的用户角色非常重要。

1.1.1. 概念

托管的集群

一个 OpenShift Container Platform 集群，其控制平面和 API 端点托管在管理集群中。托管的集群包括控制平面和它的对应的数据平面。

托管的集群基础架构

存在于租户或最终用户云账户中的网络、计算和存储资源。

托管控制平面

在管理集群上运行的 OpenShift Container Platform 控制平面，它由托管集群的 API 端点公开。控制平面的组件包括 etcd、Kubernetes API 服务器、Kubernetes 控制器管理器和 VPN。

托管集群

请参阅 *管理集群*。

受管集群

hub 集群管理的集群。此术语特定于在 Red Hat Advanced Cluster Management 中管理 Kubernetes Operator 的多集群引擎的集群生命周期。受管集群（managed cluster）与 *管理集群（management cluster）* 不同。如需更多信息，请参阅管理的集群。

管理集群

部署 HyperShift Operator，以及用于托管集群的控制平面所在的 OpenShift Container Platform 集群。管理集群与 *托管集群（hosting cluster）* 是同义的。

管理集群基础架构

管理集群的网络、计算和存储资源。

节点池

包含计算节点的资源。control plane 包含节点池。计算节点运行应用程序和工作负载。

1.1.2. Personas

集群实例管理员

假设此角色的用户等同于独立 OpenShift Container Platform 中的管理员。此用户在置备的集群中具有 **cluster-admin** 角色，但可能无法在更新或配置集群时关闭。此用户可能具有只读访问权限，来查看投射到集群中的一些配置。

集群实例用户

假设此角色的用户等同于独立 OpenShift Container Platform 中的开发人员。此用户没有 OperatorHub 或机器的视图。

集群服务消费者

假设此角色的用户可以请求控制平面和 worker 节点，驱动更新或修改外部化配置。通常，此用户无法管理或访问云凭证或基础架构加密密钥。集群服务消费者人员可以请求托管集群并与节点池交互。假设此角色的用户具有在逻辑边界中创建、读取、更新或删除托管集群和节点池的用户。

集群服务提供商

假设此角色的用户通常具有管理集群上的 **cluster-admin** 角色，并具有 RBAC 来监控并拥有 HyperShift Operator 的可用性，以及租户托管的集群的 control plane。集群服务提供商用户角色负责多个活动，包括以下示例：

- 拥有服务级别的对象，用于实现控制平面可用性、正常运行时间和稳定性。
- 为管理集群配置云帐户以托管控制平面
- 配置用户置备的基础架构，其中包括主机对可用计算资源的了解

1.2. 托管 CONTROL PLANE 简介

您可以使用 Red Hat OpenShift Container Platform 托管 control plane 来降低管理成本，优化集群部署时间，并分离管理和工作负载问题，以便专注于应用程序。

通过使用以下平台上的 multicluster engine for Kubernetes operator 版本 2.0 或更高版本 提供托管的 control plane：

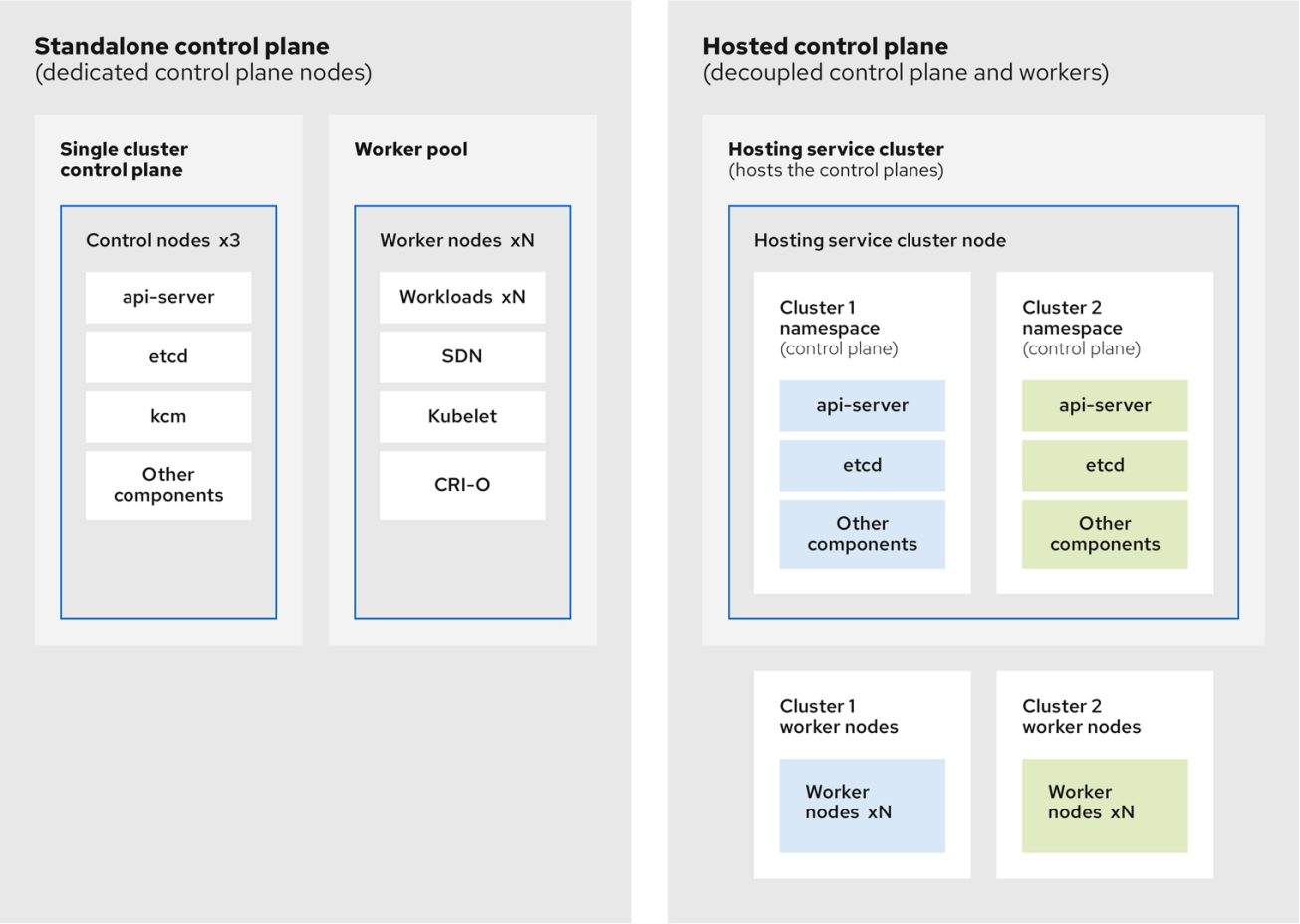
- 使用 Agent 供应商进行裸机
- OpenShift Virtualization 作为连接的环境中正式发布的功能，以及断开连接的环境中的技术预览功能
- Amazon Web Services (AWS) 为技术预览功能
- IBM Z 作为技术预览
- IBM Power，一个技术预览功能

1.2.1. 托管 control plane 的架构

OpenShift Container Platform 通常以组合或独立部署，集群由 control plane 和数据平面组成。control plane 包括 API 端点、存储端点、工作负载调度程序和确保状态的指示器。data plane 包括运行工作负载的计算、存储和网络。

独立的 control plane 由一组专用的节点（可以是物理或虚拟）托管，最小数字来确保仲裁数。网络堆栈被共享。对集群的管理员访问权限提供了对集群的 control plane、机器管理 API 和有助于对集群状态贡献的其他组件的可见性。

虽然独立模式运行良好，但在某些情况下需要与 control plane 和数据平面分离的架构。在这些情况下，data plane 位于带有专用物理托管环境的独立网络域中。control plane 使用 Kubernetes 原生的高级别原语（如部署和有状态集）托管。control plane 被视为其他工作负载。



1.2.2. 托管 control plane 的优点

使用托管 OpenShift Container Platform 的 control plane，您可以为真正的混合云方法打下基础，并享受一些其他优势。

- 管理和工作负载之间的安全界限很强大，因为 control plane 分离并在专用的托管服务集群中托管。因此，您无法将集群的凭证泄漏到其他用户。因为基础架构 secret 帐户管理也已被分离，所以集群基础架构管理员无法意外删除 control plane 基础架构。
- 使用托管 control plane，您可以在较少的节点上运行多个 control plane。因此，集群更为经济。
- 因为 control plane 由 OpenShift Container Platform 上启动的 pod 组成，所以 control planes 快速启动。同样的原则适用于 control plane 和工作负载，如监控、日志记录和自动扩展。
- 从基础架构的角度来看，您可以将 registry、HAProxy、集群监控、存储节点和其他基础架构组件推送到租户的云供应商帐户，将使用情况隔离到租户。
- 从操作的角度来看，多集群管理更为集中，从而减少了影响集群状态和一致性的外部因素。站点可靠性工程师具有调试问题并进入集群的数据平面的中心位置，这可能会导致更短的时间解析 (TTR) 并提高生产效率。

其他资源

- 托管 control plane

1.3. 托管 CONTROL PLANE 的版本控制

对于 OpenShift Container Platform 的每个主要、次版本或补丁版本，会发布两个托管的 control plane 组件：

- HyperShift Operator
- **hcp** 命令行界面 (CLI)

HyperShift Operator 管理由 **HostedCluster** API 资源表示的托管集群的生命周期。HyperShift Operator 会随每个 OpenShift Container Platform 发行版本一起发布。HyperShift Operator 在 **hypershift** 命名空间中创建 **supported-versions** 配置映射。配置映射包含受支持的托管集群版本。

您可以在同一管理集群中托管不同版本的 control plane。

supported-versions 配置映射对象示例

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.15"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

您可以使用 **hcp** CLI 创建托管集群。

您可以使用 **hypershift.openshift.io** API 资源，如 **HostedCluster** 和 **NodePool**，以大规模创建和管理 OpenShift Container Platform 集群。**HostedCluster** 资源包含 control plane 和通用数据平面配置。当您创建 **HostedCluster** 资源时，您有一个完全正常工作的 control plane，没有附加的节点。 **NodePool** 资源是一组可扩展的 worker 节点，附加到 **HostedCluster** 资源。

API 版本策略通常与 Kubernetes API 版本 的策略一致。

其他资源

- 在托管集群中配置节点性能优化
- 通过设置内核引导参数来对托管集群进行高级节点调整

第 2 章 托管的 CONTROL PLANE 入门

要开始使用 OpenShift Container Platform 的托管 control plane，首先需要在您要使用的供应商上配置托管集群。然后，完成几个管理任务。

您可以选择以下供应商来查看相关的步骤：

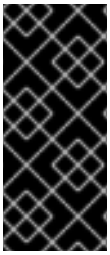
2.1. 裸机

- 托管 control plane 大小指导
- 安装托管的 control plane 命令行界面
- 分发托管集群工作负载
- 裸机防火墙和端口要求
- 裸机基础架构要求：检查基础架构要求以在裸机上创建托管集群。
- 在裸机上配置托管的 control plane 集群 ：
  - 配置 DNS
  - 创建托管集群并验证集群创建
  - 为托管集群扩展 **NodePool** 对象
  - 为托管集群处理入口流量
  - 为托管集群启用节点自动扩展
- 在断开连接的环境中配置托管的 control plane
- 要在裸机上销毁托管集群，请按照在裸机上销毁托管集群的说明进行操作。
- 如果要禁用托管的 control plane 功能，请参阅禁用托管的 control plane 功能。

2.2. OPENSHIFT VIRTUALIZATION

- 托管 control plane 大小指导
- 安装托管的 control plane 命令行界面
- 分发托管集群工作负载
- 在 OpenShift Virtualization 上管理托管的控制平面集群 ：创建带有由 KubeVirt 虚拟机托管的 worker 节点的 OpenShift Container Platform 集群。
- 在断开连接的环境中配置托管的 control plane
- 要销毁托管集群在 OpenShift Virtualization 上，请按照在 OpenShift Virtualization 上销毁托管集群的说明进行操作。
- 如果要禁用托管的 control plane 功能，请参阅禁用托管的 control plane 功能。

2.3. AMAZON WEB SERVICES (AWS)



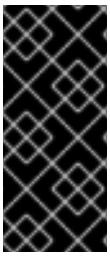
重要

在 AWS 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- AWS 基础架构要求 ：查看基础架构要求，以便在 AWS 上创建托管集群。
- 在 AWS 上配置托管的 control plane 集群（技术预览） ：在 AWS 上配置托管的 control plane 集群的任务包括创建 AWS S3 OIDC secret、创建可路由的公共区、启用外部 DNS、启用 AWS PrivateLink 和部署托管集群。
- 为托管控制平面部署 SR-IOV Operator：在配置和部署托管服务集群后，您可以在托管集群上创建到 Single Root I/O Virtualization (SR-IOV) Operator 的订阅。SR-IOV pod 在 worker 机器上运行而不是在 control plane 上运行。
- 要销毁 AWS 上的托管集群，请按照 AWS 上销毁托管集群的说明进行操作。
- 如果要禁用托管的 control plane 功能，请参阅禁用托管的 control plane 功能。

2.4. IBM Z



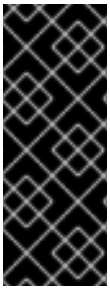
重要

在 IBM Z 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 安装托管的 control plane 命令行界面
- 为 IBM Z 计算节点在 x86 裸机上配置托管集群（技术预览）

2.5. IBM POWER



重要

在 IBM Power 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 安装托管的 control plane 命令行界面

- 在 64 位 x86 OpenShift Container Platform 集群上配置托管集群，为 IBM Power 计算节点创建托管的 control plane （技术预览）

2.6. 非裸机代理机器

重要

使用非裸机代理机器托管 control plane 集群只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 安装托管的 control plane 命令行界面
- 使用非裸机代理机器配置托管的 control plane 集群（技术预览）
- 要在非裸机代理机器上销毁托管集群，请按照在非裸机代理机器上销毁托管集群的说明
- 如果要禁用托管的 control plane 功能，请参阅禁用托管的 control plane 功能。

第 3 章 托管 CONTROL PLANE 的身份验证和授权

OpenShift Container Platform control plane 包含内置的 OAuth 服务器。您可以获取 OAuth 访问令牌来向 OpenShift Container Platform API 进行身份验证。创建托管集群后，您可以通过指定身份提供程序来配置 OAuth。

3.1. 使用 CLI 为托管集群配置 OAUTH 服务器

您可以使用 OpenID Connect 身份提供程序 (**oidc**) 为您的托管集群配置内部 OAuth 服务器。

您可以为以下支持的身份提供程序配置 OAuth：

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

在 OAuth 配置中添加任何身份提供程序会删除默认的 **kubeadmin** 用户提供程序。

先决条件

- 已创建托管集群。

流程

1. 运行以下命令，在托管集群上编辑 **HostedCluster** 自定义资源(CR)：

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

2. 使用以下示例在 **HostedCluster** CR 中添加 OAuth 配置：

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name> 1
  namespace: <hosted_cluster_namespace> 2
spec:
  configuration:
    oauth:
      identityProviders:
      - openID: 3
          claims:
```

```
      email: 4
        - <email_address>
      name: 5
        - <display_name>
      preferredUsername: 6
        - <preferred_username>
    clientID: <client_id> 7
    clientSecret:
      name: <client_id_secret_name> 8
    issuer: https://example.com/identity 9
  mappingMethod: lookup 10
  name: IAM
  type: OpenID
```

1. 指定托管的集群名称。
2. 指定托管集群的命名空间。
3. 此提供程序名称作为前缀放在身份声明值前，以此组成身份名称。提供程序名称也用于构建重定向 URL。
4. 定义用作电子邮件地址的属性列表。
5. 定义用作显示名称的属性列表。
6. 定义用作首选用户名的属性列表。
7. 定义在 OpenID 提供程序中注册的客户端的 ID。您必须允许客户端重定向到 **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** URL。
8. 定义在 OpenID 供应商中注册的客户端的机密。
9. OpenID 规范中描述的颁发者标识符。您必须使用没有查询或片段组件的 **https**。
10. 定义一个映射方法，用于控制如何在此供应商和 **User** 对象的身份之间建立映射。

3. 保存文件以使改变生效。

3.2. 使用 WEB 控制台为托管集群配置 OAUTH 服务器

您可以使用 OpenShift Container Platform Web 控制台为托管集群配置内部 OAuth 服务器。

您可以为以下支持的身份提供程序配置 OAuth：

- **oidc**
- **htpasswd**
- **keystone**
- **ldap**
- **basic-authentication**

- **request-header**
- **github**
- **gitlab**
- **google**

在 OAuth 配置中添加任何身份提供程序会删除默认的 **kubeadmin** 用户提供程序。

先决条件

- 以具有 **cluster-admin** 权限的用户身份登录。
- 已创建托管集群。

流程

1. 进入到 **Home → API Explorer**。
2. 使用 **Filter by kind** 复选框搜索您的 **HostedCluster** 资源。
3. 点您要编辑的 **HostedCluster** 资源。
4. 点 **实例** 选项卡。
5. 点托管集群名称条目旁边的 Options 菜单 ，然后点 **Edit HostedCluster**。
6. 在 YAML 文件中添加 OAuth 配置：

```
spec:
  configuration:
    oauth:
      identityProviders:
      - openID: 1
          claims:
            email: 2
              - <email_address>
            name: 3
              - <display_name>
            preferredUsername: 4
              - <preferred_username>
          clientID: <client_id> 5
          clientSecret:
            name: <client_id_secret_name> 6
          issuer: https://example.com/identity 7
        mappingMethod: lookup 8
        name: IAM
        type: OpenID
```

1 此提供程序名称作为前缀放在身份声明值前，以此组成身份名称。提供程序名称也用于构建重定向 URL。

2. 定义用作电子邮件地址的属性列表。
3. 定义用作显示名称的属性列表。
4. 定义用作首选用户名的属性列表。
5. 定义在 OpenID 提供程序中注册的客户端的 ID。您必须允许客户端重定向到 **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** URL。
6. 定义在 OpenID 供应商中注册的客户端的机密。
7. OpenID 规范中描述的颁发者标识符。您必须使用没有查询或片段组件的 **https**。
8. 定义一个映射方法，用于控制如何在此供应商和 **User** 对象的身份之间建立映射。

7. 点击 **Save**。

其他资源

- 要了解有关支持的身份提供程序的更多信息，请参阅*身份验证和授权*中的 "了解身份提供程序配置"。

第 4 章 为托管的 CONTROL PLANE 处理机器配置

在一个独立的 OpenShift Container Platform 集群中，机器配置池管理一组节点。您可以使用 **MachineConfigPool** 自定义资源 (CR) 处理机器配置。

在托管的 control plane 中，**MachineConfigPool** CR 不存在。节点池包含一组计算节点。您可以使用节点池处理机器配置。

4.1. 为托管的 CONTROL PLANE 配置节点池

在托管的 control plane 上，您可以通过在管理集群中的配置映射中创建 **MachineConfig** 对象来配置节点池。

流程

1. 要在管理集群中的配置映射中创建 **MachineConfig** 对象，请输入以下信息：

```
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 在存储 **MachineConfig** 对象的节点上设置路径。

2. 将对象添加到配置映射后，您可以将配置映射应用到节点池，如下所示：

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

4.2. 在托管集群中配置节点性能优化

要在托管集群中的节点上设置节点级别性能优化，您可以使用 Node Tuning Operator。在托管的 control plane 中，您可以通过创建包含 **Tuned** 对象并在节点池中引用这些配置映射的配置映射来配置节点调整。

流程

1. 创建包含有效 tuned 清单的配置映射，并引用节点池中的清单。在以下示例中，**Tuned** 清单定义了一个配置文件，在包含 **tuned-1-node-label** 节点标签的节点上将 **vm.dirty_ratio** 设为 55。将以下 **ConfigMap** 清单保存到名为 **tuned-1.yaml** 的文件中：

```
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
```

注意

如果您没有将任何标签添加到 Tuned spec 的 **spec.recommend** 部分中的条目中，则假定基于 node-pool 的匹配，因此 **spec.recommend** 部分中的最高优先级配置集应用于池中的节点。虽然您可以通过在 Tuned **.spec.recommend.match** 部分中设置标签值来实现更精细的节点标记匹配，除非您将节点池的 **.spec.management.upgradeType** 值设置为 **InPlace**。

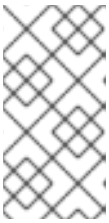
2. 在管理集群中创建 **ConfigMap** 对象：

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

3. 通过编辑节点池或创建节点池的 **spec.tuningConfig** 字段中引用 **ConfigMap** 对象。在本例中，假设您只有一个 **NodePool**，名为 **nodepool-1**，它含有 2 个节点。

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  ...
  name: nodepool-1
```

```
  namespace: clusters
...
spec:
  ...
  tuningConfig:
  - name: tuned-1
status:
...
```



注意

您可以在多个节点池中引用同一配置映射。在托管的 control plane 中，Node Tuning Operator 会将节点池名称和命名空间的哈希值附加到 Tuned CR 的名称中，以区分它们。在这种情况下，请不要为同一托管集群在不同的 Tuned CR 中创建多个名称相同的 TuneD 配置集。

验证

现在，您已创建包含 **Tuned** 清单的 **ConfigMap** 对象并在 **NodePool** 中引用它，Node Tuning Operator 会将 **Tuned** 对象同步到托管集群中。您可以验证定义了 **Tuned** 对象，以及将 TuneD 配置集应用到每个节点。

1. 列出托管的集群中的 **Tuned** 对象：

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

输出示例

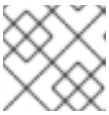
```
NAME       AGE
default    7m36s
rendered   7m36s
tuned-1    65s
```

2. 列出托管的集群中的 **Profile** 对象：

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

输出示例

```
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
```



注意

如果没有创建自定义配置集，则默认应用 **openshift-node** 配置集。

3. 要确认正确应用了调整，请在节点上启动一个 debug shell，并检查 sysctl 值：

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

输出示例

```
vm.dirty_ratio = 55
```

4.3. 为托管 CONTROL PLANE 部署 SR-IOV OPERATOR

重要

在 AWS 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

配置和部署托管服务集群后，您可以在托管集群中创建 SR-IOV Operator 订阅。SR-IOV pod 在 worker 机器上运行而不是在 control plane 上运行。

先决条件

您必须在 AWS 上配置和部署托管集群。如需更多信息，请参阅在 AWS 上配置托管集群（技术预览）。

流程

1. 创建命名空间和 Operator 组：

```
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. 创建 SR-IOV Operator 的订阅：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  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
```

验证

1. 要验证 SR-IOV Operator 是否已就绪，请运行以下命令并查看生成的输出：

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

输出示例

```
NAME                                         DISPLAY                   VERSION               REPLACES
PHASE
sriov-network-operator.4.15.0-202211021237   SR-IOV Network Operator   4.15.0-
202211021237   sriov-network-operator.4.15.0-202210290517   Succeeded
```

2. 要验证 SR-IOV pod 是否已部署，请运行以下命令：

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

第 5 章 在托管集群中使用功能门

您可以使用托管集群中的功能门启用不是默认功能集合的功能。您可以使用托管集群中的功能门启用 **TechPreviewNoUpgrade** 功能集。

5.1. 使用功能门启用功能集

您可以使用 OpenShift CLI 编辑 **HostedCluster** 自定义资源 (CR)，在托管集群中启用 **TechPreviewNoUpgrade** 功能集。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

1. 运行以下命令，打开 **HostedCluster** CR 以在托管集群上进行编辑：

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

2. 通过在 **featureSet** 字段中输入值来定义功能集。例如：

```
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. 指定托管的集群名称。
2. 指定托管集群的命名空间。
3. 这个功能集是当前技术预览功能的子集。

警告

在集群中启用 **TechPreviewNoUpgrade** 功能集无法撤消，并会阻止次版本更新。此功能集允许您在测试集群中启用这些技术预览功能，您可以在测试集群中完全测试它们。不要在生产环境集群中启用此功能。

3. 保存文件以使改变生效。

验证

- 运行以下命令，验证您的受管集群中是否启用了 **TechPreviewNoUpgrade** 功能门：

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

其他资源

- FeatureGate [config.openshift.io/v1]

第 6 章 更新托管的 CONTROL PLANE

托管 control plane 的更新涉及更新托管集群和节点池。要使集群在更新过程中完全正常工作，您必须在完成 control plane 和节点更新时满足 Kubernetes 版本偏移策略的要求。

6.1. 升级托管的 CONTROL PLANE 的要求

Kubernetes operator 的多集群引擎可以管理一个或多个 OpenShift Container Platform 集群。在 OpenShift Container Platform 上创建托管集群后，您必须在 multicluster engine operator 中导入托管集群作为受管集群。然后，您可以使用 OpenShift Container Platform 集群作为管理集群。

在开始更新托管的 control plane 前，请考虑以下要求：

- 当使用 OpenShift Virtualization 作为供应商时，您必须为 OpenShift Container Platform 集群使用裸机平台。
- 您必须使用裸机或 OpenShift Virtualization 作为托管集群的云平台。您可以在 **HostedCluster** 自定义资源 (CR) 的 **spec.Platform.type** 规格中找到托管集群的平台类型。

您必须通过完成以下任务来升级 OpenShift Container Platform 集群、多集群引擎 Operator、托管集群和节点池：

1. 将 OpenShift Container Platform 集群升级到最新版本。如需更多信息，请参阅"使用 Web 控制台更新集群"或"使用 CLI 更新集群"。
2. 将多集群引擎 Operator 升级到最新版本。如需更多信息，请参阅"更新已安装的 Operator"。
3. 将托管的集群和节点池从以前的 OpenShift Container Platform 版本升级到最新版本。如需更多信息，请参阅"为托管 control plane 更新托管集群"和"为托管 control plane 更新节点池"。

其他资源

- 使用 Web 控制台更新集群
- 使用 CLI 更新集群
- 更新安装的 Operator

6.2. 托管集群的更新

spec.release 值决定了 control plane 的版本。 **HostedCluster** 对象将预期的 **spec.release** 值传送到 **HostedControlPlane.spec.release** 值，并运行适当的 Control Plane Operator 版本。

托管的 control plane 会管理 control plane 组件的新版本的推出，以及任何 OpenShift Container Platform 组件通过 Cluster Version Operator (CVO) 的新版本。

6.3. 节点池的更新

使用节点池，您可以通过公开 **spec.release** 和 **spec.config** 值来配置在节点上运行的软件。您可以使用以下方法启动滚动节点池更新：

- 更改 **spec.release** 或 **spec.config** 值。
- 更改任何特定于平台的字段，如 AWS 实例类型。结果是一组带有新类型的新实例。

- 如果要将更改传播到节点，修改集群配置。

节点池支持替换更新和原位升级。**nodepool.spec.release** 值指定任何特定节点池的版本。**NodePool** 对象根据 **.spec.management.upgradeType** 值完成替换或原位滚动更新。

创建节点池后，您无法更改更新类型。如果要更改更新类型，您必须创建一个节点池并删除另一个节点池。

6.3.1. 替换节点池的更新

一个*替换（replace）*更新会在新版本中创建实例，并从以前的版本中删除旧的实例。对于这个级别的不可变性具有成本效率的云环境中，这个更新类型会非常有效。

替换更新不会保留任何手动更改，因为节点会被完全重新置备。

6.3.2. 节点池的原位更新

*原位（in-place）*会直接更新实例的操作系统。这个类型适用于对于基础架构的限制比较高的环境（如裸机）。

原位升级可保留手动更改，但在对集群直接关键的任何文件系统获操作系统配置（如 kubelet 证书）进行手工修改时会报告错误。

6.4. 为托管 CONTROL PLANE 更新节点池

在托管的 control plane 上，您可以通过更新节点池来更新 OpenShift Container Platform 的版本。节点池版本不能超过托管的 control plane 版本。

流程

- 输入以下命令更改节点池中的 **spec.release.image** 值：

```
$ oc patch nodepool <node_pool_name> -n <hosted_cluster_namespace> --type=merge -p
'{"spec":{"nodeDrainTimeout":"60s","release":{"image":"<openshift_release_image>"}}}' 1
2
```

1. 将 **<node_pool_name>** 和 **<hosted_cluster_namespace>** 替换为节点池名称和托管集群命名空间。
2. **<openshift_release_image>** 变量指定要升级到的新 OpenShift Container Platform 发行镜像，例如 **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**。将 **<4.y.z>** 替换为支持的 OpenShift Container Platform 版本。

验证

- 要验证新版本是否已推出，请运行以下命令检查节点池中的 **.status.conditions** 值：

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

输出示例

```
status:
```

```
conditions:
- lastTransitionTime: "2024-05-20T15:00:40Z"
    message: 'Using release image: quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64'
(1)
    reason: AsExpected
    status: "True"
    type: ValidReleaseImage
```

(1) 将 **<4.y.z>** 替换为支持的 OpenShift Container Platform 版本。

6.5. 为托管的 CONTROL PLANE 更新托管集群

在托管的 control plane 上，您可以通过更新托管集群来升级 OpenShift Container Platform 的版本。

流程

1. 输入以下命令将 **hypershift.openshift.io/force-upgrade-to=<openshift_release_image>** 注解添加到托管集群：

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

(1) 将 **<hosted_cluster_name>** 和 **<hosted_cluster_namespace>** 替换为托管的集群名称和托管集群命名空间。

(2) **<openshift_release_image>** 变量指定要升级到的新 OpenShift Container Platform 发行镜像，例如 **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**。将 **<4.y.z>** 替换为支持的 OpenShift Container Platform 版本。

2. 输入以下命令更改托管的集群中的 **spec.release.image** 值：

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

验证

- 要验证新版本是否已推出，请运行以下命令检查托管的集群中的 **.status.conditions** 和 **.status.version** 值：

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

输出示例

```
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 将 **<4.y.z>** 替换为支持的 OpenShift Container Platform 版本。

第 7 章 托管 CONTROL PLANE OBSERVABILITY

您可以通过配置指标集来收集托管 control plane 的指标。HyperShift Operator 可以为它管理的每个托管集群在管理集群中创建或删除监控仪表板。

7.1. 为托管 CONTROL PLANE 配置指标集

为 Red Hat OpenShift Container Platform 托管 control plane 会在每个 control plane 命名空间中创建 **ServiceMonitor** 资源，允许 Prometheus 堆栈从 control plane 收集指标。 **ServiceMonitor** 资源使用指标重新标记来定义从特定组件（如 etcd 或 Kubernetes API 服务器）包含或排除哪些指标。control plane 生成的指标数量会直接影响收集它们的监控堆栈的资源要求。

您可以配置一个指标集来标识为每个 control plane 生成的一组指标，而不是生成固定的指标数量。支持以下指标集：

- **Telemetry** ：遥测需要这些指标。这个集合是默认设置，是最小指标集合。
- **SRE** ：此集合包含生成警报并允许对 control plane 组件的故障排除所需的指标。
- **All** ：此集合包括由独立 OpenShift Container Platform control plane 组件生成的所有指标。

要配置指标集，请输入以下命令在 HyperShift Operator 部署中设置 **METRICS_SET** 环境变量：

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

7.1.1. 配置 SRE 指标集

当您指定 **SRE** 指标集时，HyperShift Operator 会查找带有一个键 **config** 的名为 **sre-metric-set** 的配置映射。**config** 键的值必须包含一组由 control plane 组件组织的 **RelabelConfigs**。

您可以指定以下组件：

- **etcd**
- **kubeAPIServer**
- **kubeControllerManager**
- **openshiftAPIServer**
- **openshiftControllerManager**
- **openshiftRouteControllerManager**
- **cvo**
- **olm**
- **catalogOperator**
- **registryOperator**
- **nodeTuningOperator**
- **controlPlaneOperator**

- **hostedClusterConfigOperator**

以下示例中演示了 **SRE** 指标集的配置：

```
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|schedu
ng_algorithm_priority_evaluation|scheduling_algorithm_preemption_evaluation|scheduling_algorithm_la
ency_microseconds|binding_latency_microseconds|scheduling_latency_seconds)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"apiserver_(request_count|request_latencies|request_latencies_summary|dropped_requests|storage_d
ata_key_generation_latencies_microseconds|storage_transformation_failures_total|storage_transformat
ion_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_ge
t_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. 在托管集群中启用监控仪表板

要在托管集群中启用监控仪表板，请完成以下步骤：

流程

1. 在 **local-cluster** 命名空间中创建 **hypershift-operator-install-flags** 配置映射，确保在 **data.installFlagsToAdd** 部分中指定 **--monitoring-dashboards** 标志。例如：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: hypershift-operator-install-flags
  namespace: local-cluster
```

```
data:
  installFlagsToAdd: "--monitoring-dashboards"
  installFlagsToRemove: ""
```

2. 等待几分钟，使 **hypershift** 命名空间中的 HyperShift Operator 部署被更新，使其包含以下环境变量：

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

当启用监控仪表板时，对于 HyperShift Operator 管理的每个托管集群，Operator 会在 **openshift-config-managed** 命名空间中创建一个名为 **cp-<hosted_cluster_namespace>-<hosted_cluster_name>** 的配置映射，其中 **<hosted_cluster_namespace>** 是托管集群的命名空间，**<hosted_cluster_name>** 是托管集群的名称。因此，在管理集群的管理控制台中会添加新仪表板。

3. 要查看仪表板，请登录到管理控制台，并通过点 **Observe → Dashboards**进入托管集群的仪表板。

4. 可选：要在托管集群中禁用监控仪表板，请从 **hypershift-operator-install-flags** 配置映射中删除 **--monitoring-dashboards** 标志。当您删除托管集群时，其对应的仪表板也会被删除。

7.2.1. 仪表板自定义

要为每个托管集群生成仪表板，HyperShift Operator 使用存储在 Operator 命名空间中的 **monitoring-dashboard-template** 配置映射中的模板 (**hypershift**)。此模板包含一组 Grafana 面板，其中包含仪表板的指标。您可以编辑配置映射的内容来自定义仪表板。

当生成仪表板时，以下字符串将被替换为与特定托管集群对应的值：

Name	描述
__NAME__	托管集群的名称
__NAMESPACE__	托管集群的命名空间
__CONTROL_PLANE_NAMESPACE__	放置托管集群的 control plane pod 的命名空间
__CLUSTER_ID__	托管集群的 UUID，它与托管集群指标的 **_id** 标签匹配

第 8 章 托管 CONTROL PLANE 的高可用性

8.1. 恢复不健康的 ETCD 集群

在高可用性 control plane 中，三个 etcd pod 作为 etcd 集群中有状态集的一部分运行。要恢复 etcd 集群，请通过检查 etcd 集群健康状况来识别不健康的 etcd pod。

8.1.1. 检查 etcd 集群的状态

您可以通过登录到任何 etcd pod 来检查 etcd 集群健康状态。

流程

1. 输入以下命令登录到 etcd pod：

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

2. 输入以下命令输出 etcd 集群的健康状况：

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

输出示例

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

8.1.2. 恢复失败的 etcd pod

3 节点集群的每个 etcd pod 都有自己的持久性卷声明 (PVC) 来存储其数据。由于数据损坏或缺少数据，etcd pod 可能会失败。您可以恢复 etcd pod 及其 PVC 失败。

流程

1. 要确认 etcd pod 失败，请输入以下命令：

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

输出示例

```
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
```

失败的 etcd pod 可能具有 **CrashLoopBackOff** 或 **Error** 状态。

2. 输入以下命令删除失败的 pod 及其 PVC：

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

验证

- 输入以下命令验证新 etcd pod 是否正在运行：

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

输出示例

```
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
```

8.2. 在内部环境中备份和恢复 ETCD

您可以在内部环境中的托管集群中备份和恢复 etcd，以修复失败。

8.2.1. 在内部环境中的托管集群中备份和恢复 etcd

通过在托管集群中备份和恢复 etcd，您可以修复故障，如在三个节点集群的 etcd 成员中损坏或缺少数据。如果 etcd 集群的多个成员遇到数据丢失或具有 **CrashLoopBackOff** 状态，则这种方法有助于防止 etcd 仲裁丢失。

重要

此流程需要 API 停机时间。

先决条件

- 已安装 **oc** 和 **jq** 二进制文件。

流程

1. 首先，设置环境变量并缩减 API 服务器：
   a. 输入以下命令为您的托管集群设置环境变量，根据需要替换值：

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

```
$ HOSTED_CLUSTER_NAMESPACE=clusters
```

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

   b. 输入以下命令暂停托管集群的协调，根据需要替换值：

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

   c. 输入以下命令缩减 API 服务器：
      i. 缩减 **kube-apiserver** ：

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

ii. 缩减 **openshift-apiserver** ：

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

iii. 缩减 **openshift-oauth-apiserver** ：

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

2. 接下来，使用以下方法之一生成 etcd 快照：

a. 使用之前备份的 etcd 快照。

b. 如果您有可用的 etcd pod，通过完成以下步骤从活跃 etcd pod 创建快照：

i. 输入以下命令列出 etcd pod：

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

ii. 输入以下命令为 pod 数据库生成快照并将其保存在您的机器中：

```
$ 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. 输入以下命令验证快照是否成功：

```
$ 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. 输入以下命令制作快照的本地副本：

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

i. 从 etcd 持久性存储生成快照数据库副本：

A. 输入以下命令列出 etcd pod：

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

B. 输入以下命令查找正在运行的 pod 并将其名称设置为 **ETCD_POD**：

B. 输入以下命令查找正在运行的 pod，并将其名称设置为 **ETCD_POD: ETCD_POD=etcd-0**，然后复制其快照数据库：

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

3. 接下来，输入以下命令缩减 etcd statefulset：

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

a. 输入以下命令删除第二个和第三个成员的卷：

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

b. 创建 pod 以访问第一个 etcd 成员的数据：

i. 输入以下命令来获取 etcd 镜像：

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

ii. 创建允许访问 etcd 数据的 pod：

```
$ 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. 输入以下命令检查 **etcd-data** pod 的状态并等待它正在运行：

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

iv. 输入以下命令来获取 **etcd-data** pod 的名称：

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

c. 输入以下命令将 etcd 快照复制到 pod 中：

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

d. 输入以下命令从 **etcd-data** pod 中删除旧数据：

```
$ 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. 输入以下命令恢复 etcd 快照：

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- etcdutl snapshot
restore /var/lib/restored.snap.db \
    --data-dir=/var/lib/data --skip-hash-check \
    --name etcd-0 \
    --initial-cluster-token=etcd-cluster \
    --initial-cluster etcd-0=https://etcd-0.etcd-
discovery.${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. 输入以下命令从 pod 中删除临时 etcd 快照：

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

g. 输入以下命令删除数据访问部署：

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

h. 输入以下命令扩展 etcd 集群：

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

i. 输入以下命令等待 etcd 成员 pod 返回并报告 available：

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

j. 输入以下命令扩展所有 etcd-writer 部署：

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

4. 输入以下命令恢复托管集群的协调：

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

8.3. 在 AWS 上备份和恢复 ETCD

您可以在 Amazon Web Services (AWS) 上的托管集群中备份和恢复 etcd，以修复失败。

重要

在 AWS 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

8.3.1. 为托管集群生成 etcd 快照

要为托管集群备份 etcd，您必须执行 etcd 快照。之后，您可以使用快照恢复 etcd。

重要

此流程需要 API 停机时间。

流程

1. 输入以下命令暂停托管集群的协调：

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

2. 输入以下命令停止所有 etcd-writer 部署：

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

3. 要进行 etcd 快照，请输入以下命令在每个 etcd 容器中使用 **exec** 命令：

```
$ oc exec -it <etcd_pod_name> -n <hosted_cluster_namespace> -- env ETCDCTL_API=3 /usr/bin/etcdctl --cacert /etc/etcd/tls/client/etcd-client-ca.crt --cert /etc/etcd/tls/client/etcd-client.crt --key /etc/etcd/tls/client/etcd-client.key --endpoints=localhost:2379 snapshot save /var/lib/data/snapshot.db
```

4. 要检查快照状态，请运行以下命令在每个 etcd 容器中使用 **exec** 命令：

```
$ 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. 将快照数据复制到稍后检索它的位置，如 S3 存储桶。请参见以下示例。

注意

以下示例使用签名版本 2。如果您位于支持签名版本 4 的区域，如 **us-east-2** 区域，请使用签名版本 4。否则，当将快照复制到 S3 存储桶时，上传会失败。

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. 要稍后在新集群中恢复快照，请保存托管集群引用的加密 secret。

   a. 输入以下命令来获取 secret 加密密钥：

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

   b. 输入以下命令保存 secret 加密密钥：

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

   您可以在新集群中恢复快照时解密此密钥。

后续步骤

恢复 etcd 快照。

8.3.2. 在托管集群中恢复 etcd 快照

如果从托管集群中有 etcd 快照，可以恢复它。目前，您只能在集群创建过程中恢复 etcd 快照。

要恢复 etcd 快照，您需要修改 **create cluster --render** 命令的输出，并在 **HostedCluster** 规格的 etcd 部分中定义 **restoreSnapshotURL** 值。

先决条件

在托管集群中执行 etcd 快照。

流程

1. 在 **aws** 命令行界面 (CLI) 中，创建一个预签名的 URL，以便您可以从 S3 下载 etcd 快照，而无需将凭证传递给 etcd 部署：

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

2. 修改 **HostedCluster** 规格以引用 URL：

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

3. 确保从 **spec.secretEncryption.aescbc** 值引用的 secret 包含您在前面的步骤中保存的相同 AES 密钥。

8.4. AWS 中托管集群的灾难恢复

您可以将托管集群恢复到 Amazon Web Services (AWS) 中的同一区域。例如，当升级管理集群时，需要灾难恢复，且托管集群处于只读状态。

重要

托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

灾难恢复过程涉及以下步骤：

1. 在源集群中备份托管集群
2. 在目标管理集群中恢复托管集群
3. 从源管理集群中删除托管的集群

您的工作负载在此过程中保持运行。集群 API 可能会在一段时间内不可用，但不会影响 worker 节点上运行的服务。

重要

源管理集群和目标管理集群必须具有 **--external-dns** 标志才能维护 API 服务器 URL。这样，服务器 URL 以 **https://api-sample-hosted.sample-hosted.aws.openshift.com** 结尾。请参见以下示例：

示例：外部 DNS 标记

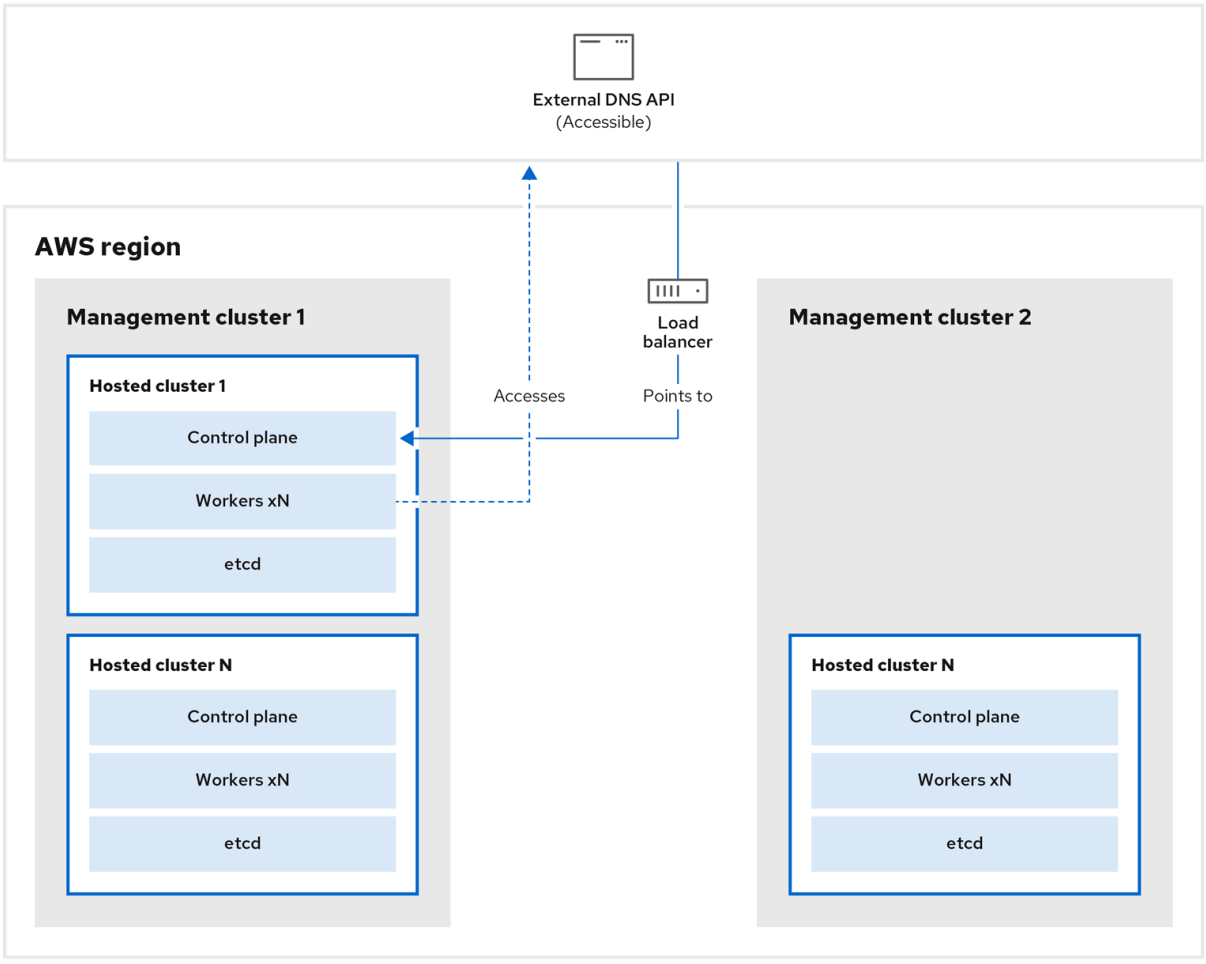
```
--external-dns-provider=aws \
--external-dns-credentials=<path_to_aws_credentials_file> \
--external-dns-domain-filter=<basedomain>
```

如果您没有包含 **--external-dns** 标志来维护 API 服务器 URL，则无法迁移托管集群。

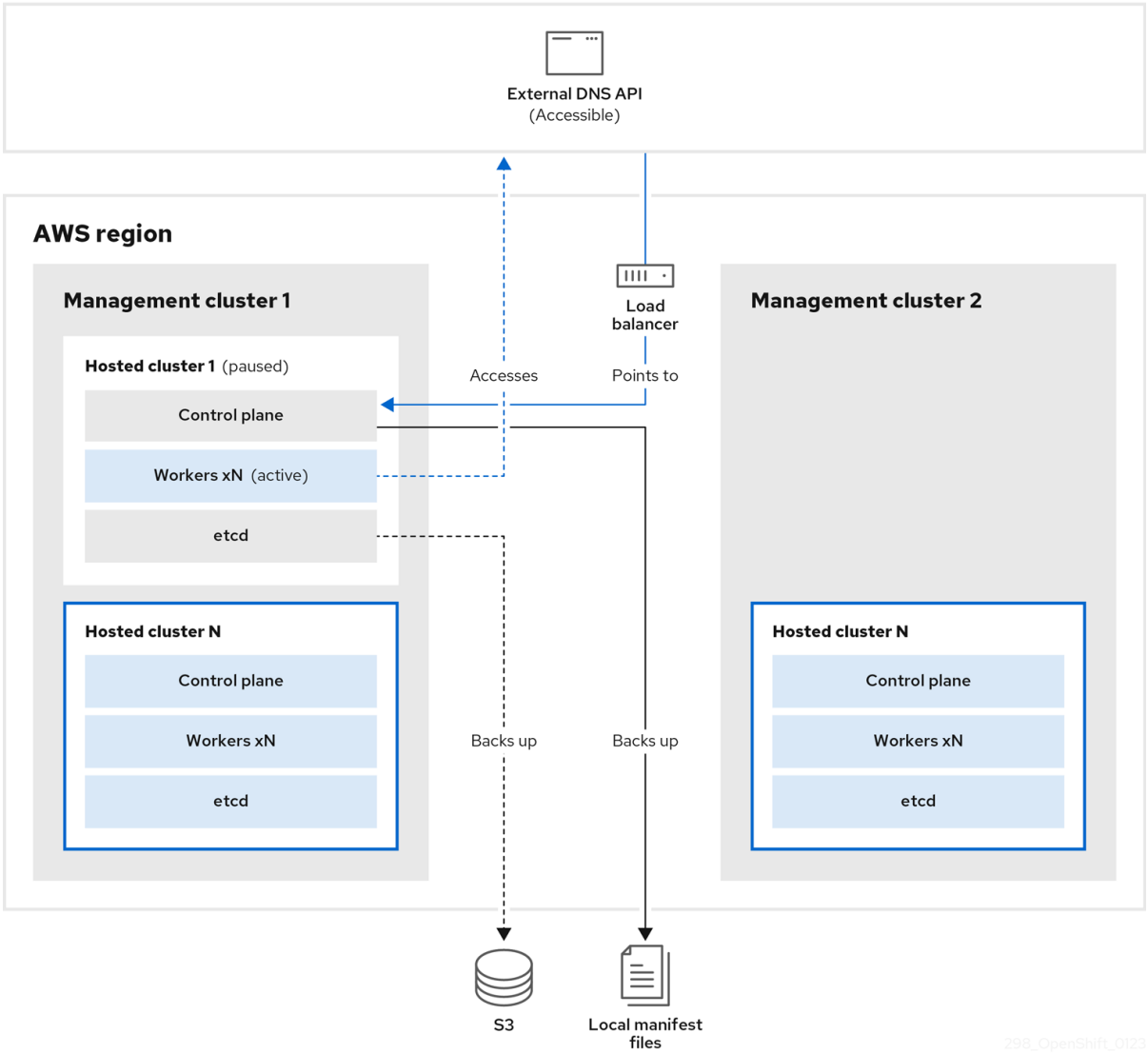
8.4.1. 备份和恢复过程概述

备份和恢复过程按如下方式工作：

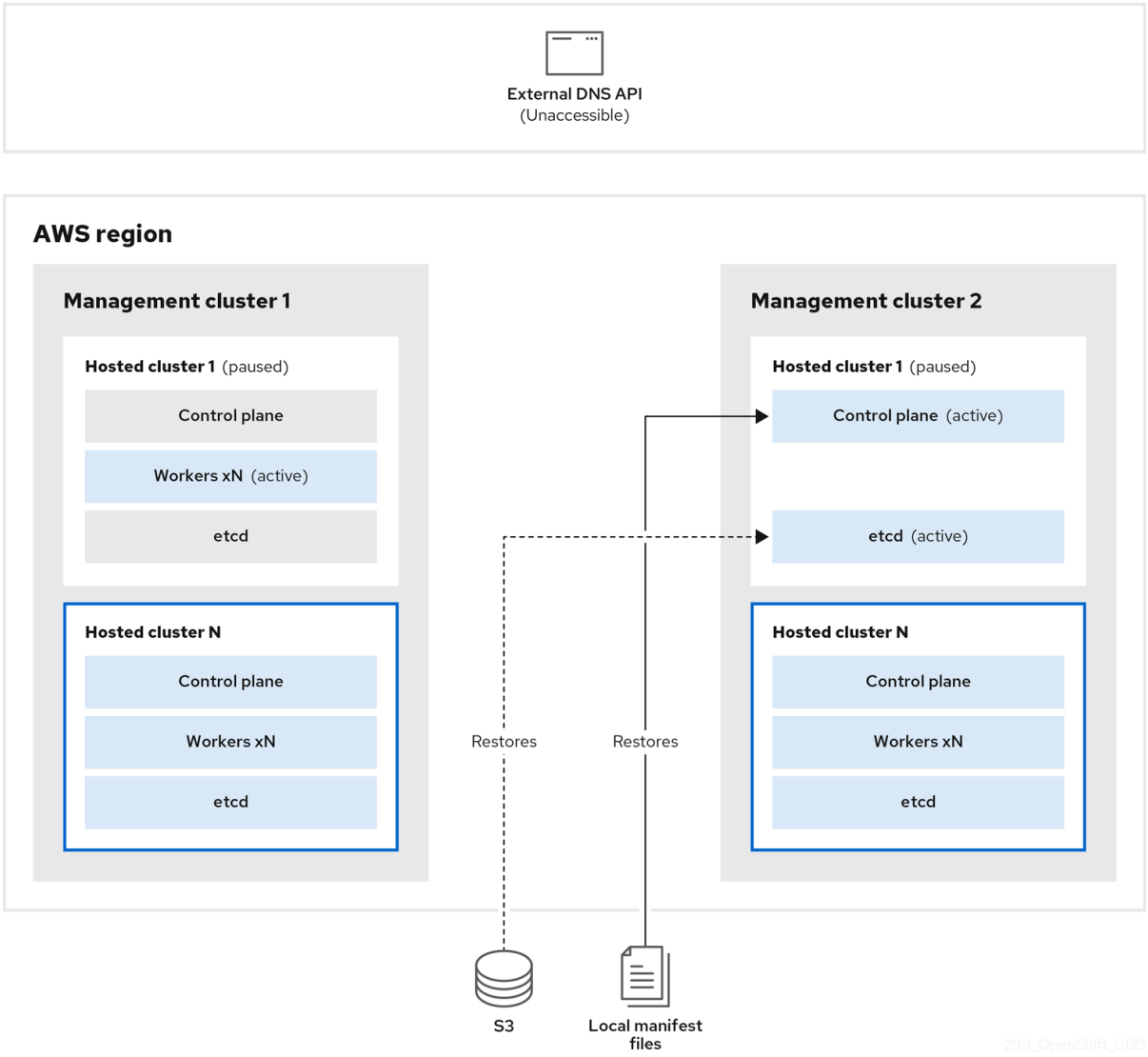
1. 在管理集群 1 中，您可以将其视为源管理集群，control plane 和 worker 使用 ExternalDNS API 进行交互。可以访问外部 DNS API，并且一个负载均衡器位于管理集群之间。



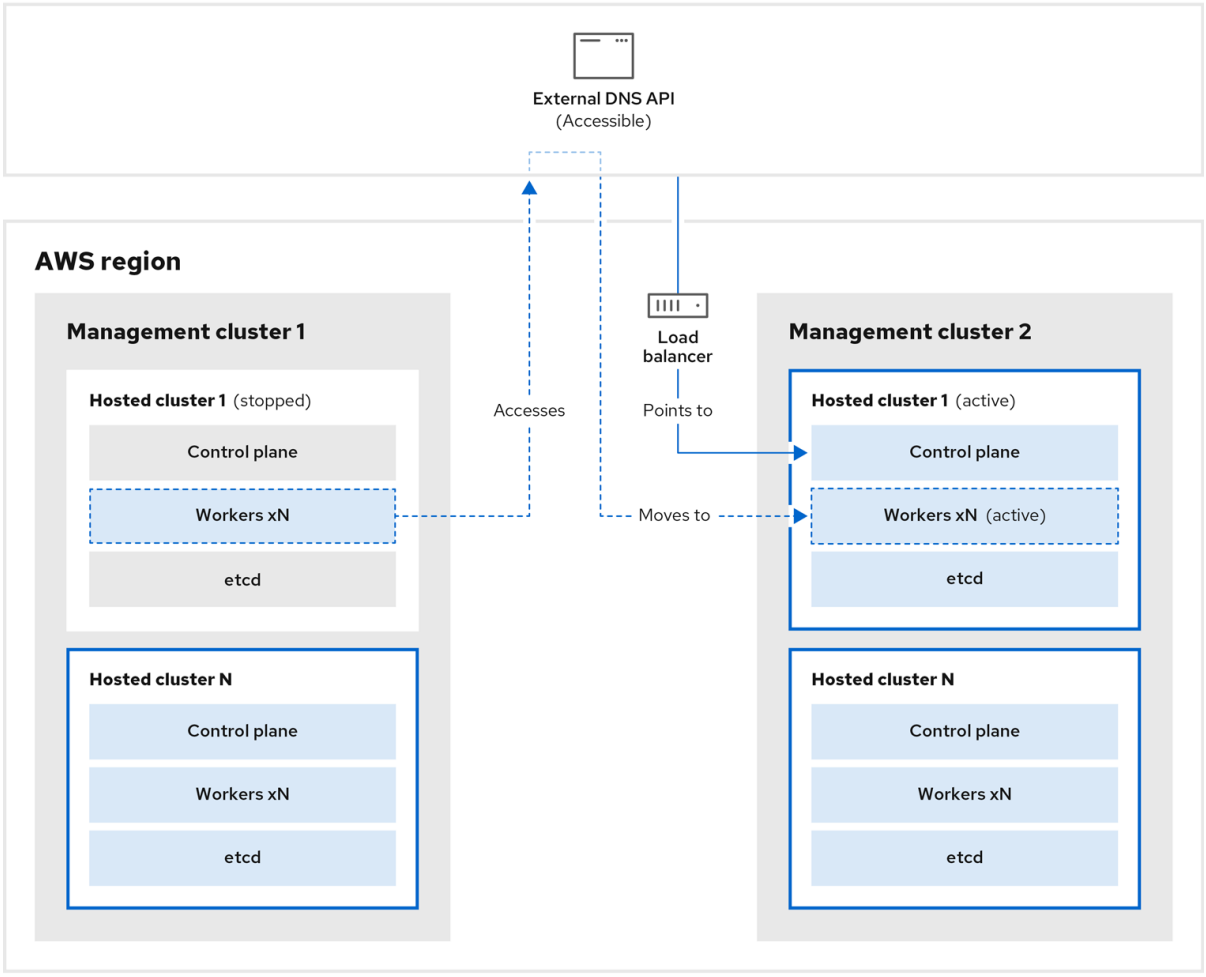
2. 您为托管集群执行快照，其中包括 etcd、control plane 和 worker 节点。在此过程中，worker 节点仍然会尝试访问外部 DNS API，即使无法访问它，工作负载正在运行，control plane 存储在本地清单文件中，etcd 会备份到 S3 存储桶。data plane 处于活跃状态，control plane 已暂停。



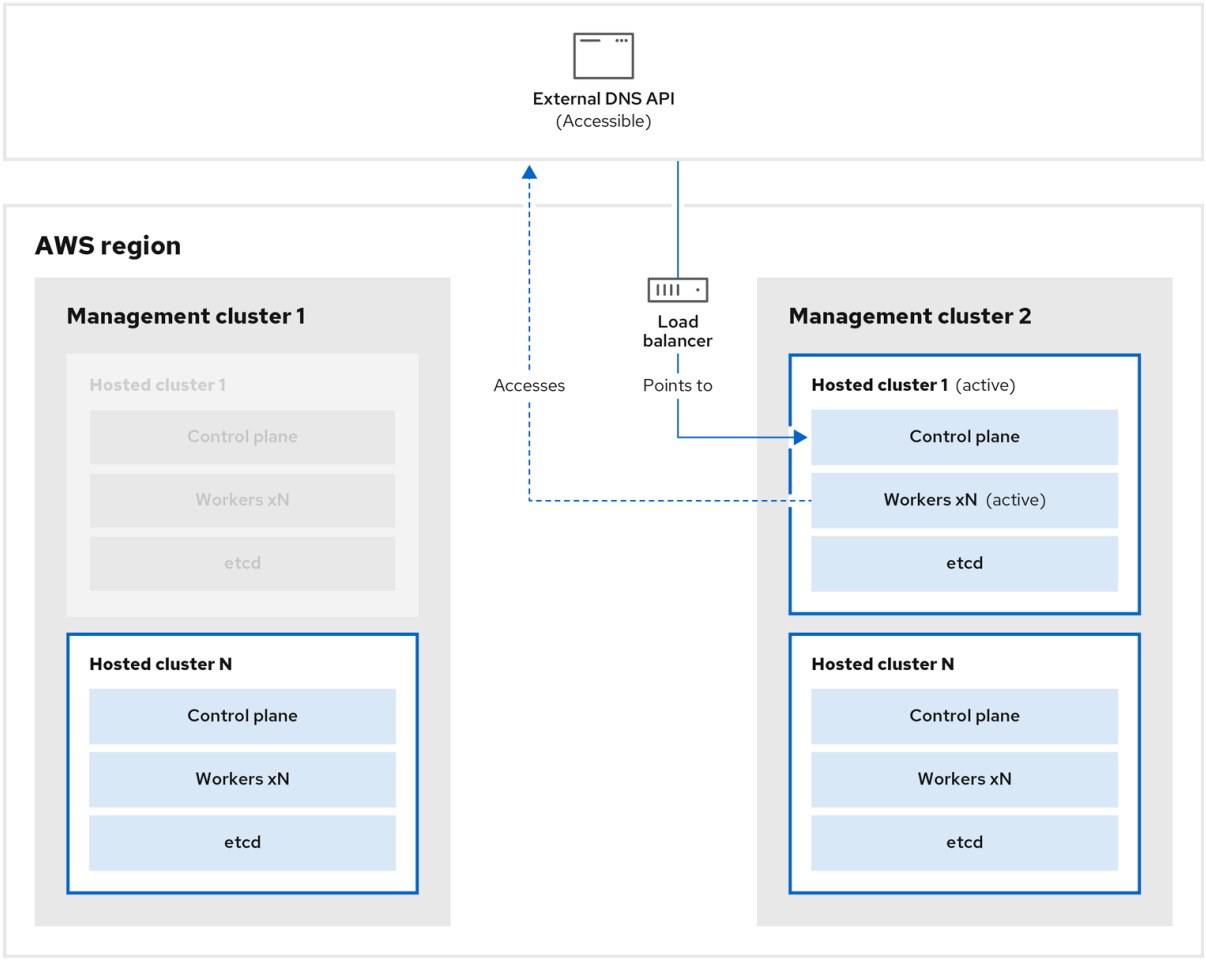
3. 在管理集群 2 中，您可以将其视为目标管理集群，您可以从 S3 存储桶中恢复 etcd，并从本地清单文件恢复 control plane。在此过程中，外部 DNS API 会停止，托管集群 API 变得不可访问，任何使用 API 的 worker 都无法更新其清单文件，但工作负载仍在运行。



4. 外部 DNS API 可以再次访问，worker 节点使用它来移至管理集群 2。外部 DNS API 可以访问指向 control plane 的负载均衡器。



5. 在管理集群 2 中，control plane 和 worker 节点使用外部 DNS API 进行交互。资源从管理集群 1 中删除，但 etcd 的 S3 备份除外。如果您尝试在 mangagement 集群 1 上再次设置托管集群，它将无法正常工作。



8.4.2. 备份托管集群

要在目标管理集群中恢复托管集群，首先需要备份所有相关数据。

流程

1. 输入以下命令创建 configmap 文件来声明源管理集群：

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

2. 输入这些命令，在托管集群和节点池中关闭协调：

```
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. 运行此 bash 脚本备份 etcd 并将数据上传到 S3 存储桶：

提示

将此脚本嵌套在函数中，并从主功能调用它。

```
# 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
```

有关备份 etcd 的更多信息，请参阅 "Backing and restore etcd on a hosted cluster"。

4. 输入以下命令备份 Kubernetes 和 OpenShift Container Platform 对象。您需要备份以下对象：

- 来自 HostedCluster 命名空间的 **HostedCluster** 和 **NodePool** 对象

- 来自 HostedCluster 命名空间中的 **HostedCluster** secret
- 来自 Hosted Control Plane 命名空间中的 **HostedControlPlane**
- 来自 Hosted Control Plane 命名空间的 **Cluster**
- 来自 Hosted Control Plane 命名空间的**AWSCluster**, **AWSMachineTemplate**, 和 **AWSMachine**
- 来自 Hosted Control Plane 命名空间的 **MachineDeployments**, **MachineSets**, 和 **Machines**。
- 来自 Hosted Control Plane 命名空间的 **ControlPlane**

```
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. 输入以下命令清理 **ControlPlane** 路由：

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

输入该命令，您可以启用 ExternalDNS Operator 来删除 Route53 条目。

6. 运行该脚本验证 Route53 条目是否清理：

```
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}"
```

验证

检查所有 OpenShift Container Platform 对象和 S3 存储桶，以验证所有内容是否如预期。

后续步骤

恢复托管集群。

8.4.3. 恢复托管集群

收集您备份和恢复目标管理集群中的所有对象。

先决条件

您已从源集群备份数据。

提示

确保目标管理集群的 **kubeconfig** 文件放置在 **KUBECONFIG** 变量中，或者在 **MGMT2_KUBECONFIG** 变量中设置。使用 **export KUBECONFIG=<Kubeconfig FilePath>**，或者使用 **export KUBECONFIG=${MGMT2_KUBECONFIG}**。

流程

1. 输入以下命令验证新管理集群是否不包含您要恢复的集群中的任何命名空间：

```
# 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. 输入以下命令重新创建已删除的命名空间：

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

3. 输入以下命令恢复 HC 命名空间中的 secret：

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

4. 输入以下命令恢复 **HostedCluster** control plane 命名空间中的对象：

```
# 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. 如果您要恢复节点和节点池来重复利用 AWS 实例，请输入以下命令恢复 HC control plane 命名空间中的对象：

```
# 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. 运行此 bash 脚本恢复 etcd 数据和托管集群：

```
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. 如果您要恢复节点和节点池来重复利用 AWS 实例，请输入以下命令恢复节点池：

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

验证

- 要验证节点是否已完全恢复，请使用此功能：

```
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
```

后续步骤

关闭并删除集群。

8.4.4. 从源集群中删除托管集群

备份托管集群并将其恢复到目标管理集群后，您将关闭并删除源管理集群中的托管集群。

先决条件

您备份了数据并将其恢复到源管理集群。

提示

确保目标管理集群的 **kubeconfig** 文件放置在 **KUBECONFIG** 变量中，或者在 **MGMT_KUBECONFIG** 变量中设置。使用 **export KUBECONFIG=<Kubeconfig FilePath>** 或使用脚本，请使用 **export KUBECONFIG=${MGMT_KUBECONFIG}**。

流程

1. 输入以下命令来扩展 **deployment** 和 **statefulset** 对象：

重要

如果其 **spec.persistentVolumeClaimRetentionPolicy.whenScaled** 字段被设置为 **Delete**，则不要扩展有状态的集合，因为这可能会导致数据丢失。

作为临时解决方案，将 **spec.persistentVolumeClaimRetentionPolicy.whenScaled** 字段的值更新为 **Retain**。确保不存在协调有状态集的控制器，并将值返回为 **Delete**，这可能会导致丢失数据。

```
# 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. 输入以下命令来删除 **NodePool** 对象：

```
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. 输入以下命令删除 **machine** 和 **machineset** 对象：

```
# 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. 输入以下命令删除集群对象：

```
# 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. 输入这些命令来删除 AWS 机器 (Kubernetes 对象)。不用担心删除实际的 AWS 机器。云实例不会受到影响。

```
# 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. 输入以下命令删除 **HostedControlPlane** 和 **ControlPlane** HC 命名空间对象：

```
# 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. 输入以下命令删除 **HostedCluster** 和 HC 命名空间对象：

```
# 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
```

验证

- 要验证所有内容是否正常工作，请输入以下命令：

```
# 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
```

后续步骤

删除托管的集群中的 OVN pod，以便您可以连接到新管理集群中运行的新 OVN control plane：

1. 使用托管的集群的 kubeconfig 路径加载 **KUBECONFIG** 环境变量。
2. 输入这个命令：

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

第 9 章 托管 CONTROL PLANE 故障排除

如果您在托管 control plane 时遇到问题，请参阅以下信息来引导您完成故障排除。

9.1. 收集信息以对托管 CONTROL PLANE 进行故障排除

当您需要对托管的 control plane 集群问题进行故障排除时，您可以通过运行 **hypershift dump cluster** 命令来收集信息。该命令为管理集群和托管集群生成输出。

管理集群的输出包含以下内容：

- **集群范围的资源：**这些资源是管理集群的节点定义。
- **hypershift-dump 压缩文件：** 如果您需要与其他人员共享内容，该文件非常有用。
- **命名空间资源：**这些资源包括来自相关命名空间中的所有对象，如配置映射、服务、事件和日志。
- **网络日志：**这些日志包括 OVN 北向和南向数据库和每个数据库的状态。
- **托管的集群：**此级别的输出涉及托管集群内的所有资源。

托管集群的输出包含以下内容：

- **集群范围的资源：**这些资源包含所有集群范围的对象，如节点和 CRD。
- **命名空间资源：**这些资源包括来自相关命名空间中的所有对象，如配置映射、服务、事件和日志。

虽然输出不包含集群中的任何 secret 对象，但它可以包含对 secret 名称的引用。

先决条件

- 您必须具有对管理集群的 **cluster-admin** 访问权限。
- 您需要 **HostedCluster** 资源的 **name** 值以及部署 CR 的命名空间。
- 已安装 **hcp** 命令行界面。如需更多信息，请参阅安装托管的 control plane 命令行界面。
- 已安装 OpenShift CLI (**oc**)。
- 您必须确保 **kubeconfig** 文件已被加载，并指向管理集群。

流程

- 要收集故障排除的输出，请输入以下命令：

```
$ hypershift dump cluster \
    --name <hosted_cluster_name> \ 1
    --namespace <hosted_cluster_namespace> \ 2
    --dump-guest-cluster \
    --artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
```

1 指定托管的集群名称。

2 指定托管集群的命名空间，例如 **clusters**。

输出示例

```
2023-06-06T12:18:20+02:00  INFO  Archiving dump  {"command": "tar", "args": ["-cvzf",
"hypershift-dump.tar.gz", "cluster-scoped-resources", "event-filter.html", "namespaces",
"network_logs", "timestamp"]}
2023-06-06T12:18:21+02:00  INFO  Successfully archived dump  {"duration":
"1.519376292s"}
```

- 要配置命令行界面，使其通过使用用户名或服务帐户模拟对管理集群的所有查询，请输入带有 **--as** 标志的 **hypershift dump cluster** 命令。
服务帐户必须具有足够的权限来从命名空间中查询所有对象，因此建议 **cluster-admin** 角色确保有足够的权限。服务帐户必须位于其中，或者具有查询 **HostedControlPlane** 资源命名空间的权限。

  如果您的用户名或服务帐户没有足够的权限，输出只会包含您有权访问的对象。在此过程中，您可能会看到 **forbidden** 错误。

  - 要使用服务帐户使用模拟，请输入以下命令：

```
$ hypershift dump cluster \
  --name <hosted_cluster_name> \1
  --namespace <hosted_cluster_namespace> \2
  --dump-guest-cluster \
  --as "system:serviceaccount:<service_account_namespace>:
<service_account_name>" \3
  --artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
```

1 指定托管的集群名称。

2 指定托管集群的命名空间，例如 **clusters**。

3 指定 **默认** 命名空间和名称，例如 **"system:serviceaccount:default:samplesa"**。

  - 要使用用户名来模拟，请输入以下命令：

```
$ hypershift dump cluster \
  --name <hosted_cluster_name> \1
  --namespace <hosted_cluster_namespace> \2
  --dump-guest-cluster \
  --as "<cluster_user_name>" \3
  --artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
```

1 指定托管的集群名称。

2 指定托管集群的命名空间，例如 **clusters**。

3 指定集群用户名，如 **cloud-admin**。

9.2. 暂停托管集群和托管的 CONTROL PLANE 的协调

如果您是集群管理员，您可以暂停托管集群和托管的控制平面的协调。当您备份和恢复 etcd 数据库或需要调试托管集群或托管的 control plane 的问题时，您可能希望暂停协调。

流程

1. 要暂停托管集群和托管的 control plane 的协调，请填充 **HostedCluster** 资源的 **pausedUntil** 字段。

   - 要暂停协调直到特定时间，请输入以下命令：

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

     1 以 RFC339 格式指定时间戳，例如 **2024-03-03T03:28:48Z**。协调会暂停，直到经过了指定的时间。

   - 要无限期暂停协调，请输入以下命令：

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

     协调会暂停，直到您从 **HostedCluster** 资源中删除了字段。

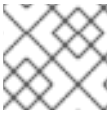
     当为 **HostedCluster** 资源填充暂停协调字段时，该字段会自动添加到关联的 **HostedControlPlane** 资源中。

2. 要删除 **pausedUntil** 字段，请输入以下 patch 命令：

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

9.3. 将数据平面缩减为零

如果您没有使用托管的控制平面，为了保存资源和成本，您可以将数据平面缩减为零。



注意

确保您准备将数据平面缩减为零。因为 worker 节点的工作负载在缩减后会消失。

流程

1. 运行以下命令，设置 **kubeconfig** 文件以访问托管集群：

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

2. 运行以下命令，获取与托管集群关联的 **NodePool** 资源的名称：

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

3. 可选： 要防止 pod 排空，请运行以下命令在 **NodePool** 资源中添加 **nodeDrainTimeout** 字段：

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

输出示例

```
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)
# ...
```

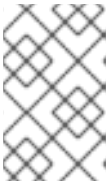
1. 定义托管集群的名称。
2. 指定控制器排空节点的时间总量。默认情况下，**nodeDrainTimeout: 0s** 设置会阻止节点排空过程。

注意

要允许节点排空过程在特定时间段内继续，您可以相应地设置 **nodeDrainTimeout** 字段的值，例如 **nodeDrainTimeout: 1m**。

4. 运行以下命令来缩减与托管集群关联的 **NodePool** 资源：

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



注意

将数据计划缩减为零后，control plane 中的一些 pod 会一直处于 **Pending** 状态，托管的 control plane 会保持启动并运行。如果需要，您可以扩展 **NodePool** 资源。

5. 可选：运行以下命令来扩展与托管集群关联的 **NodePool** 资源：

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

在重新扩展 **NodePool** 资源后，请等待几分钟，让 **NodePool** 资源变为 **Ready** 状态。

验证

- 运行以下命令，验证 **nodeDrainTimeout** 字段的值是否大于 **0s** ：

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

其他资源

- 托管集群的 must-gather