OpenShift Container Platform 4.15

安装后配置

OpenShift Container Platform 的第二天操作

Last Updated: 2024-07-26

OpenShift Container Platform 4.15 安装后配置

OpenShift Container Platform 的第二天操作

法律通告

摘要

本文档提供有关 OpenShift Container Platform 安装后进行的操作说明。

目录

第 1 章 安装后配置概述

安装 OpenShift Container Platform 后，集群管理员可以配置和自定义以下组件：

- 机器
- 裸机
- 集群
- 节点
- Network
- 存储
- 用户
- 警报和通知

1.1. 安装后配置任务

您可以执行安装后配置任务来配置环境，以满足您的需要。

以下列表详细介绍了这些配置：

- 配置操作系统功能 ：Machine Config Operator(MCO) 管理 **MachineConfig** 对象。通过使用 MCO，您可以配置节点和自定义资源。
- 配置裸机节点 ：您可以使用 Bare Metal Operator (BMO) 来管理裸机主机。BMO 可以完成以下操作：
  - 检查主机的硬件详情，并将其报告到裸机主机。
  - 检查固件并配置 BIOS 设置。
  - 使用所需镜像调配主机。
  - 在置备主机之前或之后清理主机的磁盘内容。
- 配置集群功能。您可以修改 OpenShift Container Platform 集群的以下功能：
  - 镜像 registry
  - 网络配置
  - 镜像构建行为
  - 用户身份提供程序
  - etcd 配置
  - 用于处理工作负载的机器集
  - 云供应商凭证管理

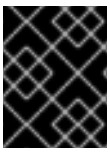
- 配置私有集群：默认情况下，安装程序使用公开的 DNS 和端点置备 OpenShift Container Platform。要使集群只能从内部网络进行访问，请配置以下组件使其私有：
  - DNS
  - Ingress Controller
  - API Server
- 执行节点操作 ：默认情况下，OpenShift Container Platform 使用 Red Hat Enterprise Linux CoreOS(RHCOS)计算机器。您可以执行以下操作：
  - 添加和删除计算机器。
  - 添加和删除污点和容限。
  - 配置每个节点的最大 pod 数量。
  - 启用设备管理器。
- 配置网络：安装 OpenShift Container Platform 后，您可以配置以下组件：
  - 入口集群流量
  - 节点端口服务范围
  - 网络策略
  - 启用集群范围代理
- 配置存储 ：默认情况下，容器使用临时存储或临时存储进行操作。临时存储具有生命周期限制。要长期存储数据，您必须配置持久性存储。您可以使用以下方法之一配置存储：
  - **动态置备** ：您可以通过定义并创建控制不同级别的存储类（包括存储访问）来按需动态置备存储。
  - **静态置备** ：您可以使用 Kubernetes 持久性卷使现有存储可供集群使用。静态置备支持各种设备配置和挂载选项。
- 配置用户 ：OAuth 访问令牌允许用户自行验证 API。您可以配置 OAuth 以执行以下任务：
- 指定身份提供程序
- 使用基于角色的访问控制为用户定义和授予权限
- 从 OperatorHub 安装 Operator
- 配置警报通知 ：默认情况下，触发的警报显示在 web 控制台的 Alerting UI 中。您还可以配置 OpenShift Container Platform，将警报通知发送到外部系统。

第 2 章 配置私有集群

安装 OpenShift Container Platform 版本 4.15 集群后，您可以将其某些核心组件设置为私有。

2.1. 关于私有集群

默认情况下，OpenShift Container Platform 被置备为使用可公开访问的 DNS 和端点。在部署私有集群后，您可以将 DNS、Ingress Controller 和 API 服务器设置为私有。



重要

如果集群有任何公共子网，管理员创建的负载均衡器服务可能会公开访问。为确保集群安全性，请验证这些服务是否已明确标注为私有。

DNS

如果在安装程序置备的基础架构上安装 OpenShift Container Platform，安装程序会在预先存在的公共区中创建记录，并在可能的情况下为集群自己的 DNS 解析创建一个私有区。在公共区和私有区中，安装程序或集群为 ***.apps** 和 **Ingress** 对象创建 DNS 条目，并为 API 服务器创建 **api**。

公共和私有区中的 ***.apps** 记录是相同的，因此当您删除公有区时，私有区为集群无缝地提供所有 DNS 解析。

Ingress Controller

由于默认 **Ingress** 对象是作为公共对象创建的，所以负载均衡器是面向互联网的，因此在公共子网中。

Ingress Operator 为 Ingress Controller 生成默认证书，以充当占位符，直到您配置了自定义默认证书为止。不要在生产环境集群中使用 Operator 生成的默认证书。Ingress Operator 不轮转其自身的签名证书或它生成的默认证书。Operator 生成的默认证书的目的是作为您配置的自定义默认证书的占位者。

API Server

默认情况下，安装程序为 API 服务器创建适当的网络负载均衡器，供内部和外部流量使用。

在 Amazon Web Services（AWS）上，会分别创建独立的公共和私有负载均衡器。负载均衡器是基本相同的，唯一不同是带有一个额外的、用于在集群内部使用的端口。虽然安装程序根据 API 服务器要求自动创建或销毁负载均衡器，但集群并不管理或维护它们。只要保留集群对 API 服务器的访问，您可以手动修改或移动负载均衡器。对于公共负载均衡器，需要打开端口 6443，并根据 **/readyz** 路径配置 HTTPS 用于健康检查。

在 Google Cloud Platform 上，会创建一个负载均衡器来管理内部和外部 API 流量，因此您无需修改负载均衡器。

在 Microsoft Azure 上，会创建公共和私有负载均衡器。但是，由于当前实施的限制，您刚刚在私有集群中保留两个负载均衡器。

2.2. 将 DNS 设置为私有

部署集群后，您可以修改其 DNS 使其只使用私有区。

流程

1. 查看集群的 **DNS** 自定义资源：

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

输出示例

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

请注意，**spec** 部分包含一个私有区和一个公共区。

2. 修补 **DNS** 自定义资源以删除公共区：

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec": {"publicZone": null}}'
dns.config.openshift.io/cluster patched
```

因为 Ingress Controller 在创建 **Ingress** 对象时会参考 **DNS** 定义，因此当您创建或修改 **Ingress** 对象时，只会创建私有记录。

重要

在删除公共区时，现有 Ingress 对象的 DNS 记录不会修改。

3. 可选：查看集群的 **DNS** 自定义资源，并确认已删除公共区：

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

输出示例

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
```

```
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>-wfpg4: owned
status: {}
```

2.3. 将 INGRESS CONTROLLER 设置为私有

部署集群后，您可以修改其 Ingress Controller 使其只使用私有区。

流程

1. 修改默认 Ingress Controller，使其仅使用内部端点：

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

输**出示例**

```
ingresscontroller.operator.openshift.io "default" deleted
ingresscontroller.operator.openshift.io/default replaced
```

删除公共 DNS 条目，并更新私有区条目。

2.4. 将 API 服务器限制为私有

将集群部署到 Amazon Web Services（AWS）或 Microsoft Azure 后，可以重新配置 API 服务器，使其只使用私有区。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 使用具有 **admin** 权限的用户登陆到 web 控制台。

流程

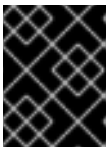
1. 在云供应商的 web 门户或控制台中，执行以下操作：
   a. 找到并删除相关的负载均衡器组件：
      - 对于 AWS，删除外部负载均衡器。私有区的 API DNS 条目已指向内部负载均衡器，它使用相同的配置，因此您无需修改内部负载均衡器。
      - 对于 Azure，删除公共负载均衡器的 **api-internal-v4** 规则。

b. 对于 Azure，将 Ingress Controller 端点发布范围配置为 **Internal**。如需更多信息，请参阅"将 Ingress Controller 端点发布范围配置为 Internal"。

c. 对于 Azure 公共负载均衡器，如果您将 Ingress Controller 端点发布范围配置为 **Internal**，且公共负载均衡器中没有现有的入站规则，则必须明确创建一个出站规则来为后端地址池提供出站流量。如需更多信息，请参阅 Microsoft Azure 文档中有关添加出站规则的内容。

d. 删除 public 区域中的 **api.$clustername.$yourdomain** 或 **api.$clustername** DNS 条目。

2. AWS 集群：删除外部负载均衡器：



重要

您只能对安装程序置备的基础架构 (IPI) 集群执行以下步骤。对于用户置备的基础架构 (UPI) 集群，您必须手动删除或禁用外部负载均衡器。

- 如果您的集群使用 control plane 机器集，删除配置公共或外部负载均衡器的 control plane 机器集自定义资源中的行：

```
# ...
providerSpec:
  value:
# ...
    loadBalancers:
    - name: lk4pj-ext 1
      type: network 2
    - name: lk4pj-int
      type: network
# ...
```

1 删除外部负载均衡器的 **name** 值，它以 **-ext** 结尾。

2 删除外部负载均衡器的 **type** 值。

- 如果您的集群没有使用 control plane 机器集，您必须从每个 control plane 机器中删除外部负载均衡器。

i. 在终端中，运行以下命令来列出集群机器：

```
$ oc get machine -n openshift-machine-api
```

输**出示例**

```
NAME                          STATE    TYPE       REGION     ZONE         AGE
lk4pj-master-0                running  m4.xlarge  us-east-1  us-east-1a   17m
lk4pj-master-1                running  m4.xlarge  us-east-1  us-east-1b   17m
lk4pj-master-2                running  m4.xlarge  us-east-1  us-east-1a   17m
lk4pj-worker-us-east-1a-5fzfj running  m4.xlarge  us-east-1  us-east-1a   15m
lk4pj-worker-us-east-1a-vbghs running  m4.xlarge  us-east-1  us-east-1a   15m
lk4pj-worker-us-east-1b-zgpzg running  m4.xlarge  us-east-1  us-east-1b   15m
```

control plane 机器在名称中包含 **master**。

ii. 从每个 control plane 机器中删除外部负载均衡器：

A. 运行以下命令，将 control plane 机器对象编辑为：

```
$ oc edit machines -n openshift-machine-api <control_plane_name> 1
```

1 指定要修改的 control plane 机器对象名称。

B. 删除描述外部负载均衡器的行，它们在以下示例中被标记：

```
# ...
providerSpec:
  value:
# ...
    loadBalancers:
    - name: lk4pj-ext 1
      type: network 2
    - name: lk4pj-int
      type: network
# ...
```

1 删除外部负载均衡器的 **name** 值，它以 **-ext** 结尾。

2 删除外部负载均衡器的 **type** 值。

C. 保存更改并退出对象规格。

D. 为每个 control plane 机器重复此步骤。

其他资源

- 将 Ingress Controller 端点发布范围配置为 Internal

2.4.1. 将 Ingress Controller 端点发布范围配置为 Internal

当集群管理员在没有指定集群为私有的情况下安装新集群时，将默认 Ingress Controller 创建，并将 **scope** 设置为 **External**。集群管理员可以将 **External** 范围的 Ingress Controller 更改为 **Internal**。

先决条件

- 已安装 **oc** CLI。

流程

- 要将 **External** 范围的 Ingress Controller 更改为 **Internal**，请输入以下命令：

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"scope":"Internal"}}}}'
```

- 要检查 Ingress Controller 的状态，请输入以下命令：

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- **Progressing** 状态条件指示您必须执行进一步的操作。例如，状态条件可以通过输入以下命令来指示需要删除该服务：

```
$ oc -n openshift-ingress delete services/router-default
```

如果删除了该服务，Ingress Operator 会重新创建为 **Internal**。

2.5. 在 AZURE 上配置私有存储端点

您可以使用 Image Registry Operator 在 Azure 上使用私有端点，它会在 OpenShift Container Platform 部署到私有 Azure 集群上时启用私有存储帐户的无缝配置。这可让您在不公开面向公共的存储端点的情况下部署镜像 registry。

您可以通过两种方式之一将 Image Registry Operator 配置为使用 Azure 上的私有存储端点：

- 通过配置 Image Registry Operator 来发现 VNet 和子网名称
- 使用用户提供的 Azure Virtual Network (VNet) 和子网名称

2.5.1. 在 Azure 上配置私有存储端点的限制

在 Azure 上配置私有存储端点时有以下限制：

- 在将 Image Registry Operator 配置为使用私有存储端点时，禁用对存储帐户的公共网络访问。因此，在 registry Operator 配置中设置 **disableRedirect: true** 才能从 OpenShift Container Platform 之外拉取镜像。启用重定向后，registry 会重定向客户端直接从存储帐户拉取镜像，这不再会因为公共网络访问被禁用而工作。如需更多信息，请参阅"在 Azure 上使用私有存储端点时禁用重定向"。
- Image Registry Operator 无法撤销此操作。

2.5.2. 通过启用 Image Registry Operator 发现 VNet 和子网名称，在 Azure 上配置私有存储端点

以下流程演示了如何通过配置 Image Registry Operator 来发现 VNet 和子网名称在 Azure 上设置私有存储端点。

先决条件

- 您已将镜像 registry 配置为在 Azure 上运行。
- 使用 Installer Provisioned Infrastructure 安装方法设置了您的网络。
  对于具有自定义网络设置的用户，请参阅"使用用户提供的 VNet 和子网名称在 Azure 上配置私有存储端点"。

流程

1. 编辑 Image Registry Operator **config** 对象，并将 **networkAccess.type** 设置为 **Internal** ：

```
$ oc edit configs.imageregistry/cluster
```

```
# ...
spec:
  # ...
   storage:
     azure:
       # ...
       networkAccess:
         type: Internal
# ...
```

2. 可选：输入以下命令确认 Operator 已完成置备。这可能需要几分钟时间。

```
$ oc get configs.imageregistry/cluster -o=jsonpath="
{.spec.storage.azure.privateEndpointName}" -w
```

3. 可选：如果 registry 由路由公开，并且要将存储帐户配置为私有，则必须禁用重定向（如果集群外部拉取到集群），则必须禁用重定向才能继续工作。输入以下命令禁用 Image Operator 配置中的重定向：

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

注意

启用重定向后，从集群外部拉取镜像将无法正常工作。

验证

1. 运行以下命令来获取 registry 服务名称：

```
$ oc registry info --internal=true
```

输**出示例**

```
image-registry.openshift-image-registry.svc:5000
```

2. 运行以下命令来进入 debug 模式：

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

3. 运行推荐的 **chroot** 命令。例如：

```
$ chroot /host
```

4. 输入以下命令登录到容器 registry：

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-
image-registry.svc:5000
```

输**出示例**

```
Login Succeeded!
```

5. 输入以下命令验证您可以从 registry 中拉取镜像：

```
$ podman pull --tls-verify=false image-registry.openshift-image-
registry.svc:5000/openshift/tools
```

输**出示例**

```
Trying to pull image-registry.openshift-image-
registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.3. 使用用户提供的 VNet 和子网名称在 Azure 上配置私有存储端点

使用以下步骤配置禁用公共网络访问权限的存储帐户，并在 Azure 上的私有存储端点后公开。

先决条件

- 您已将镜像 registry 配置为在 Azure 上运行。
- 您必须知道用于 Azure 环境的 VNet 和子网名称。
- 如果您的网络是在 Azure 中的单独资源组中配置的，还必须知道其名称。

流程

1. 编辑 Image Registry Operator **config** 对象并使用 VNet 和子网名称配置私有端点：

```
$ oc edit configs.imageregistry/cluster
```

```
# ...
spec:
  # ...
    storage:
      azure:
        # ...
        networkAccess:
          type: Internal
          internal:
            subnetName: <subnet_name>
            vnetName: <vnet_name>
            networkResourceGroupName: <network_resource_group_name>
# ...
```

2. 可选：输入以下命令确认 Operator 已完成置备。这可能需要几分钟时间。

```
$ oc get configs.imageregistry/cluster -o=jsonpath="
{.spec.storage.azure.privateEndpointName}" -w
```

注意

启用重定向后，从集群外部拉取镜像将无法正常工作。

验证

1. 运行以下命令来获取 registry 服务名称：

```
$ oc registry info --internal=true
```

输**出示例**

```
image-registry.openshift-image-registry.svc:5000
```

2. 运行以下命令来进入 debug 模式：

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

3. 运行推荐的 **chroot** 命令。例如：

```
$ chroot /host
```

4. 输入以下命令登录到容器 registry：

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```

输**出示例**

```
Login Succeeded!
```

5. 输入以下命令验证您可以从 registry 中拉取镜像：

```
$ podman pull --tls-verify=false image-registry.openshift-image-registry.svc:5000/openshift/tools
```

输**出示例**

```
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.4. 可选：在 Azure 上使用私有存储端点时禁用重定向

默认情况下，在使用镜像 registry 时会启用重定向。重定向允许将 registry pod 的流量卸载到对象存储中，从而加快拉取速度。当启用重定向且存储帐户为私有时，来自集群外部的用户无法从 registry 中拉取镜像。

在某些情况下，用户可能希望禁用重定向，以便集群外部的用户可以从 registry 中拉取镜像。

使用以下步骤禁用重定向。

先决条件

- 您已将镜像 registry 配置为在 Azure 上运行。
- 您已配置了路由。

流程

- 输入以下命令禁用镜像 registry 配置中的重定向：

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

验证

1. 运行以下命令来获取 registry 服务名称：

```
$ oc registry info
```

输**出示例**

```
default-route-openshift-image-registry.<cluster_dns>
```

2. 输入以下命令登录到容器 registry：

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) default-route-openshift-image-registry.<cluster_dns>
```

输**出示例**

```
Login Succeeded!
```

3. 输入以下命令验证您可以从 registry 中拉取镜像：

```
$ podman pull --tls-verify=false default-route-openshift-image-registry.<cluster_dns>/openshift/tools
```

输**出示例**

```
Trying to pull default-route-openshift-image-registry.<cluster_dns>/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

第 3 章 裸机配置

在裸机主机上部署 OpenShift Container Platform 时，在置备前或置备后，有时您需要对主机进行更改。这包括检查主机的硬件、固件和固件详情。它还可以包含格式化磁盘或更改可修改的固件设置。

3.1. 关于 BARE METAL OPERATOR

使用 Bare Metal Operator (BMO) 来置备、管理和检查集群中的裸机主机。

BMO 使用三个资源来完成这些任务：

- **BareMetalHost**
- **HostFirmwareSettings**
- **FirmwareSchema**

BMO 通过将每个裸机主机映射到 **BareMetalHost** 自定义资源定义的实例来维护集群中的物理主机清单。每个 **BareMetalHost** 资源都有硬件、软件和固件详情。BMO 持续检查集群中的裸机主机，以确保每个 **BareMetalHost** 资源准确详细说明相应主机的组件。

BMO 还使用 **HostFirmwareSettings** 资源和 **FirmwareSchema** 资源来详细说明裸机主机的固件规格。

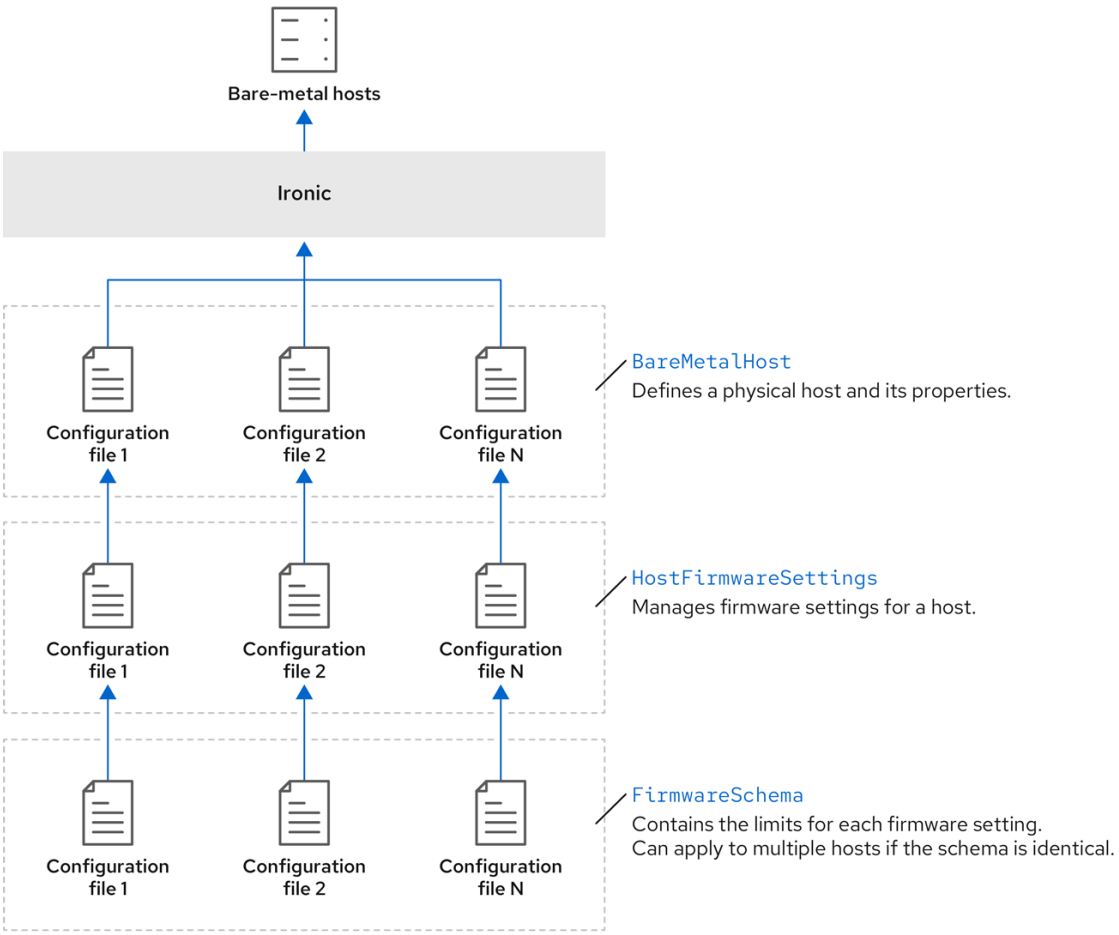
使用 Ironic API 服务在集群中的裸机主机的 BMO 接口。Ironic 服务使用主机上的 Baseboard Management Controller (BMC) 来与机器进行接口。

使用 BMO 可以完成的一些常见任务包括：

- 使用特定镜像置备裸机主机到集群
- 在置备前或取消置备后格式化主机的磁盘内容
- 打开或关闭主机
- 更改固件设置
- 查看主机的硬件详情

3.1.1. 裸机 Operator 架构

Bare Metal Operator (BMO) 使用三个资源来置备、管理和检查集群中的裸机主机。下图演示了这些资源的架构：



BareMetalHost

BareMetalHost 资源定义物理主机及其属性。将裸机主机置备到集群时，您必须为该主机定义 **BareMetalHost** 资源。对于主机的持续管理，您可以检查 **BareMetalHost** 中的信息或更新此信息。

BareMetalHost 资源具有置备信息，如下所示：

- 部署规格，如操作系统引导镜像或自定义 RAM 磁盘
- 置备状态
- 基板管理控制器 (BMC) 地址
- 所需的电源状态

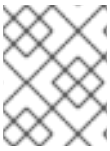
BareMetalHost 资源具有硬件信息，如下所示：

- CPU 数量
- NIC 的 MAC 地址
- 主机的存储设备的大小
- 当前电源状态

HostFirmwareSettings

您可以使用 **HostFirmwareSettings** 资源来检索和管理主机的固件设置。当主机进入 **Available** 状态时，Ironic 服务读取主机的固件设置并创建 **HostFirmwareSettings** 资源。**BareMetalHost** 资源和 **HostFirmwareSettings** 资源之间存在一对一映射。

您可以使用 **HostFirmwareSettings** 资源来检查主机的固件规格，或更新主机的固件规格。



注意

在编辑 **HostFirmwareSettings** 资源的 **spec** 字段时，您必须遵循特定于供应商固件的 schema。这个模式在只读 **FirmwareSchema** 资源中定义。

FirmwareSchema

固件设置因硬件供应商和主机模型而异。**FirmwareSchema** 资源是一个只读资源，其中包含每个主机模型中每个固件设置的类型和限制。数据通过使用 Ironic 服务直接从 BMC 传递。**FirmwareSchema** 资源允许您识别 **HostFirmwareSettings** 资源的 **spec** 字段中可以指定的有效值。

如果 schema 相同，则 **FirmwareSchema** 资源可应用到许多 **BareMetalHost** 资源。

其他资源

- 用于置备裸机主机的 metal的 API 服务
- 用于管理裸机基础架构的 ironic API 服务

3.2. 关于 BAREMETALHOST 资源

裸机[3] 引入了 **BareMetalHost** 资源的概念，它定义了物理主机及其属性。**BareMetalHost** 资源包含两个部分：

1. **BareMetalHost** 规格
2. **BareMetalHost** 状态

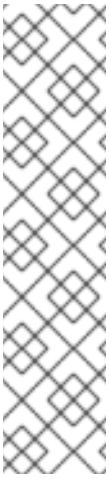
3.2.1. BareMetalHost 规格

BareMetalHost 资源的 **spec** 部分定义了主机所需状态。

表 3.1. BareMetalHost spec

参数	描述
automatedCleaningMode	在置备和取消置备过程中启用或禁用自动清理的接口。当设置为 **disabled** 时，它将跳过自动清理。当设置为 **metadata** 时，会自动清理会被启用。默认设置为 **metadata**。

参数	描述
bmc: address: credentialsName: disableCertificateVerification:	**bmc** 配置设置包含主机上基板管理控制器(BMC)的连接信息。这些字段包括： • **address** ：与主机的 BMC 控制器通信的 URL。 • **credentialsName** ：引用包含 BMC 的用户名和密码的 secret。 • **disableCertificateVerification** ：当设置为 **true** 时跳过证书验证的布尔值。
bootMACAddress	用于置备主机的 NIC 的 MAC 地址。
bootMode	主机的引导模式。它默认为 **UEFI**，但也可以设置为 **legacy** 用于 BIOS 引导，或 **UEFISecureBoot**。
consumerRef	对使用主机的另一个资源的引用。如果另一个资源目前没有使用主机，则它可能为空。例如，当 **machine-api** 使用主机时，**Machine** 资源可能会使用主机。
description	提供的字符串，用于帮助识别主机。
externallyProvisioned	指明主机置备和取消置备是在外部管理的布尔值。当设置时： • 仍可使用在线字段管理电源状态。 • 将监控硬件清单，但不会在主机上执行置备或取消置备操作。
firmware	包含有关裸机主机的 BIOS 配置的信息。目前，只有 iRMC、S iDRAC、i iLO4 和 iLO5 BMC 支持 **firmware**。子字段有： • **simultaneousMultithreadingEnabled** ：允许单个物理处理器内核显示为多个逻辑处理器。有效设置为 **true** 或 **false**。 • **sriovEnabled**: SR-IOV 支持可让虚拟机监控程序创建 PCI-express 设备的虚拟实例，这可能会提高性能。有效设置为 **true** 或 **false**。 • **virtualizationEnabled**：支持平台硬件的虚拟化。有效设置为 **true** 或 **false**。

参数	描述
image: url: checksum: checksumType: format:	**image** 配置设置包含要部署到主机上的镜像的详细信息。Ironic 需要镜像字段。但是，当 **externallyProvisioned** 配置设置被设置为 **true**，且外部管理不需要电源控制时，字段可以为空。这些字段包括： ● **url**：部署到主机的镜像的 URL。 ● **checksum** ：位于 **image.url** 的镜像的实际校验和值，或包括镜像的校验和的文件 URL。 ● **checksumType** ：指定 checksum 算法。目前，**Image.checksumType** 只支持 **md5**、**sha256** 和 **sha512**。默认 checksum 类型为 **md5**。 ● **format** ：镜像的磁盘格式。它可以是 **raw**, **qcow2**, **vdi**, **vmdk**, **live-iso** 或不设置。把它设置为 **raw** 可为该镜像在 Ironic 代理中进行原始镜像流处理。将它设置为 **live-iso** 会启用 iso 镜像在没有部署到磁盘的情况下进行实时引导，它会忽略 **checksum** 字段。
networkData	对包含网络配置数据及其命名空间的 secret 的引用，以便在主机引导以设置网络前将其附加到主机。
online	指示主机是否应开启的布尔值，**true** 代表开启，**false** 代表关闭。更改此值将触发对物理主机的电源状态变化。
raid: hardwareRAIDVolumes: softwareRAIDVolumes:	（可选）包含有关裸机主机的 RAID 配置的信息。如果没有指定，它会保留当前的配置。  **注意** OpenShift Container Platform 4.15 支持 BMC 的硬件 RAID，包括： ● Fujitsu iRMC 支持 RAID 0、1、5、6 和 10 ● Dell iDRAC 使用带有固件版本 6.10.30.20 或更高版本和 RAID 级别 0、1 和 5 的 Redfish API OpenShift Container Platform 4.15 不支持软件 RAID。 请参见以下配置设置： ● **hardwareRAIDVolumes** ：包含硬件 RAID 的逻辑驱动器列表，并在硬件 RAID 中定义所需的卷配置。如果您没有指定

参数	描述
	rootDeviceHints，则第一个卷是 root 卷。子字段是： ○ **level** ：逻辑驱动器的 RAID 级别。支持以下级别：**0**,**1**,**2**,**5**,**6**,**1+0**,**5+0**,**6+0**. ○ **name** ：卷名称（字符串）。它在服务器中应该是唯一的。如果未指定，则自动生成卷名称。 ○ **numberOfPhysicalDisks** ：物理驱动器的数量，作为用于逻辑 drove 的整数。默认为特定 RAID 级别所需的最小磁盘驱动器数。 ○ **physicalDisks** ：物理磁盘驱动器的名称列表作为字符串。这是可选字段。如果指定，还必须指定 controller 字段。 ○ **controller** ：（可选）RAID 控制器的名称作为要在硬件 RAID 卷中使用的字符串。 ○ **rotational** ：如果设为 **true**，则它只会选择轮转磁盘驱动器。如果设置为 **false**，它将只选择固态和 NVMe 驱动器。如果没有设置，则会选择任何驱动器类型，这是默认行为。 ○ **sizeGibibytes** ：逻辑驱动器的大小作为在 GiB 中创建的整数。如果未指定或设置为 **0**，它将为逻辑驱动器使用物理驱动器的最大容量。 ● **softwareRAIDVolumes**: OpenShift Container Platform 4.15 不支持软件 RAID。以下信息仅供参考。这个配置包含软件 RAID 的逻辑磁盘列表。如果您没有指定 **rootDeviceHints**，则第一个卷是 root 卷。如果您设置了 **HardwareRAIDVolumes**，则此项将无效。软件 RAID 总是被删除。创建的软件 RAID 设备的数量必须是 **1** 或 **2**。如果只有一个软件 RAID 设备，它必须是 **RAID-1**。如果有两个 RAID 设备，则第一个设备必须是 **RAID-1**，而第二个设备的 RAID 级别可以为 **0**, **1**, 或 **1+0**。第一个 RAID 设备将是部署设备。因此，当设备出现故障时，强制 **RAID-1** 降低了非引导节点的风险。**softwareRAIDVolume** 字段定义软件 RAID 中卷所需的配置。子字段是： ○ **level** ：逻辑驱动器的 RAID 级别。支持以下级别：**0**,**1**,**1+0**。 ○ **physicalDisks** ：设备提示列表.项目数量应大于或等于 **2**。 ○ **sizeGibibytes** ：逻辑磁盘驱动器的大小作为整数，以 GiB 为单位创建。如果未指定或设置为 **0**，它将为逻辑驱动器使用物理驱动器的最大容量。 您可以将 **hardwareRAIDVolume** 设置为空片段，以清除硬件 RAID 配置。例如：

参数	描述
	spec: raid: hardwareRAIDVolume: [] 如果您收到出错信息表示驱动程序不支持 RAID，则将 **raid**, **hardwareRAIDVolumes** 或 **softwareRAIDVolumes** 设置为 nil。您可能需要确保主机具有 RAID 控制器。
rootDeviceHints: deviceName: hctl: model: vendor: serialNumber: minSizeGigabytes: wwn: wwnWithExtension: wwnVendorExtension: rotational:	**rootDeviceHints** 参数启用将 RHCOS 镜像置备到特定设备。它会按照发现设备的顺序检查设备，并将发现的值与 hint 值进行比较。它使用第一个与 hint 值匹配的发现设备。该配置可组合多个 hints，但设备必须与所有提示都匹配才能被选择。这些字段包括： • **deviceName**：包含类似 **/dev/vda** 的 Linux 设备名称的字符串。hint 必须与实际值完全匹配。 • **hctl** ：包含类似 **0:0:0:0** 的 SCSI 总线地址的字符串。hint 必须与实际值完全匹配。 • **model** ：包含特定厂商的设备标识符的字符串。hint 可以是实际值的子字符串。 • **vendor** ：包含该设备厂商或制造商名称的字符串。hint 可以是实际值的子字符串。 • **serialNumber** ：包含设备序列号的字符串。hint 必须与实际值完全匹配。 • **minSizeGigabytes** ：一个整数，代表设备的最小大小（以 GB 为单位）。 • **wwn** ：包含唯一存储标识符的字符串。hint 必须与实际值完全匹配。 • **wwnWithExtension** ：包含附加厂商扩展的唯一存储标识符的字符串。hint 必须与实际值完全匹配。 • **wwnVendorExtension** ：包含唯一厂商存储标识符的字符串。hint 必须与实际值完全匹配。 • **rotational** ：指示该设备应该是旋转磁盘(true)还是非旋转磁盘(false)的布尔值。

3.2.2. BareMetalHost 状态

BareMetalHost 状态代表主机的当前状态，包括经过测试的凭证、当前的硬件详情和其他信息。

表 3.2. BareMetalHost 状态

参数	描述
goodCredentials	对 secret 及其命名空间的引用，其中包含最近一组基板管理控制器(BMC)凭证，以便系统能够验证。

参数	描述
errorMessage	置备后端的最后一个错误的详情（若有）。
errorType	表示导致主机进入错误状态的问题类别。错误类型包括： • **provisioned registration error**：当控制器无法重新注册已置备的主机时发生。 • **registration error** ：当控制器无法连接到主机的基板管理控制器时，请注意。 • **inspection error** ：尝试从主机获取硬件详细信息时发生错误。 • **preparation error** ：在清理失败时生成。 • **provisioning error** ：当控制器无法置备或取消置备主机时会发生。 • **power management error** ：当控制器无法修改主机的电源状态时会发生。 • **detach error**: 当控制器无法从置备程序卸载主机时会发生。
hardware: cpu arch: model: clockMegahertz: flags: count:	系统中的 CPU 的 **hardware.cpu** 字段详情。这些字段包括： • **arch** ：CPU 的架构。 • **model**: CPU 系列（字符串） • **clockMegahertz** ：CPU 的速度（MHz）。 • **flags**: CPU 标记列表。例如，**'mmx','sse','sse2','vmx'** 等。 • **count**: 系统中可用的 CPU 数量。
hardware: firmware:	包含 BIOS 固件信息。例如，硬件供应商和版本。

参数	描述
hardware: nics: - ip: name: mac: speedGbps: vlans: vlanId: pxe:	**hardware.nics** 字段包含主机的网络接口列表。这些字段包括： • **ip**：NIC 的 IP 地址，如果在发现代理运行时是否被分配。 • **name**: 标识网络设备的字符串。例如，**nic-1**。 • **mac**: NIC 的 MAC 地址。 • **speedGbps**: 设备的速度（Gbps）。 • **vlans**: 保存此 NIC 可用的所有 VLAN 的列表。 • **vlanId** ：未标记的 VLAN ID。 • **pxe**: NIC 是否能够使用 PXE 引导。
hardware: ramMebibytes:	主机的内存量（兆字节(MiB)）。
hardware: storage: - name: rotational: sizeBytes: serialNumber:	**hardware.storage** 字段包含可用于主机的存储设备列表。这些字段包括： • **name**: 标识存储设备的字符串。例如，**disk 1 (boot)**. • **rotational** ：指示磁盘是否为轮转的，返回 **true** 或 **false**。 • **sizeBytes** ：存储设备的大小。 • **serialNumber** ：设备的序列号。
hardware: systemVendor: manufacturer: productName: serialNumber:	包含主机的 **manufacturer**, **productName**, 和 **serialNumber** 的信息。
lastUpdated	主机状态最后一次更新的时间戳。

参数	描述
operationalStatus	服务器的状态。状态为以下之一： • **OK** ：指示主机的所有详细信息均为已知、正确配置、正常工作和可以管理。 • **discovered**: 指定某些主机的详细信息无法正常工作或缺失。例如，BMC 地址已知，但登录凭证未知。 • **error**: 指示系统发现某种不可恢复的错误。如需更多详细信息，请参阅 status 部分中的 **errorMessage** 字段。 • **delayed**：指示置备延迟来限制同时置备多个主机。 • **detached**: 指示主机被标记为 **unmanaged**。
poweredOn	指明主机是否开机的布尔值。

参数	描述
provisioning: state: id: image: raid: firmware: rootDeviceHints:	**provisioning** 字段包含与主机部署镜像相关的值。子字段包括： • **state**: 任何持续置备操作的当前状态。状态包括： ◦ **<empty string>**: 目前没有进行置备。 ◦ **unmanaged**: 没有足够的信息来注册主机。 ◦ **registering**: 用来检查主机的 BMC 详情的代理 ◦ **match profile** ：代理将主机上发现的硬件详情与已知的配置集进行比较。 ◦ **available**: 主机可用于置备。这个状态之前被称为 **ready**。 ◦ **preparing** ：现有配置将被删除，新的配置将在主机上设置。 ◦ **provisioning** ：置备程序正在将镜像写入主机的存储。 ◦ **provisioned**: 置备程序已将镜像写入主机的存储。 ◦ **externally provisioned**: Metal^3 不管理主机上的镜像。 ◦ **deprovisioning**: 置备程序正在从主机的存储中清除镜像。 ◦ **inspecting**: 代理正在收集主机的硬件详情。 ◦ **deleting**: 代理正在从集群中删除。 • **id**: 底层调配工具中服务的唯一标识符。 • **Image** ：最近部署到主机的镜像。 • **raid**: 最近设定的硬件或软件 RAID 卷列表。 • **firmware**: 裸机服务器的 BIOS 配置。 • **rootDeviceHints** ：用于最新置备操作的根设备选择说明。
triedCredentials	对 secret 及其命名空间的引用，其中包含发送到置备后端的最后一个 BMC 凭证集合。

3.3. 获取 BAREMETALHOST 资源

BareMetalHost 资源包含物理主机的属性。您必须获取物理主机的 **BareMetalHost** 资源才能查看其属性。

流程

1. 获取 **BareMetalHost** 资源列表：

```
$ oc get bmh -n openshift-machine-api -o yaml
```

注意

您可以使用 **baremetalhost** 作为 **oc get** 命令的 **bmh** 长形式。

2. 获取主机列表：

```
$ oc get bmh -n openshift-machine-api
```

3. 获取特定主机的 **BareMetalHost** 资源：

```
$ oc get bmh <host_name> -n openshift-machine-api -o yaml
```

其中 **<host_name>** 是主机的名称。

输**出示例**

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  creationTimestamp: "2022-06-16T10:48:33Z"
  finalizers:
  - baremetalhost.metal3.io
  generation: 2
  name: openshift-worker-0
  namespace: openshift-machine-api
  resourceVersion: "30099"
  uid: 1513ae9b-e092-409d-be1b-ad08edeb1271
spec:
  automatedCleaningMode: metadata
  bmc:
    address: redfish://10.46.61.19:443/redfish/v1/Systems/1
    credentialsName: openshift-worker-0-bmc-secret
    disableCertificateVerification: true
  bootMACAddress: 48:df:37:c7:f7:b0
  bootMode: UEFI
  consumerRef:
    apiVersion: machine.openshift.io/v1beta1
    kind: Machine
    name: ocp-edge-958fk-worker-0-nrfcg
    namespace: openshift-machine-api
  customDeploy:
    method: install_coreos
  online: true
  rootDeviceHints:
    deviceName: /dev/disk/by-id/scsi-<serial_number>
```

```
    userData:
      name: worker-user-data-managed
      namespace: openshift-machine-api
  status:
    errorCount: 0
    errorMessage: ""
    goodCredentials:
      credentials:
        name: openshift-worker-0-bmc-secret
        namespace: openshift-machine-api
      credentialsVersion: "16120"
    hardware:
      cpu:
        arch: x86_64
        clockMegahertz: 2300
        count: 64
        flags:
        - 3dnowprefetch
        - abm
        - acpi
        - adx
        - aes
        model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
      firmware:
        bios:
          date: 10/26/2020
          vendor: HPE
          version: U30
      hostname: openshift-worker-0
      nics:
      - mac: 48:df:37:c7:f7:b3
        model: 0x8086 0x1572
        name: ens1f3
      ramMebibytes: 262144
      storage:
      - hctl: "0:0:0:0"
        model: VK000960GWTTB
        name: /dev/disk/by-id/scsi-<serial_number>
        sizeBytes: 960197124096
        type: SSD
        vendor: ATA
      systemVendor:
        manufacturer: HPE
        productName: ProLiant DL380 Gen10 (868703-B21)
        serialNumber: CZ200606M3
    lastUpdated: "2022-06-16T11:41:42Z"
    operationalStatus: OK
    poweredOn: true
    provisioning:
      ID: 217baa14-cfcf-4196-b764-744e184a3413
      bootMode: UEFI
      customDeploy:
        method: install_coreos
      image:
        url: ""
      raid:
```

```
    hardwareRAIDVolumes: null
    softwareRAIDVolumes: []
  rootDeviceHints:
    deviceName: /dev/disk/by-id/scsi-<serial_number>
  state: provisioned
triedCredentials:
  credentials:
    name: openshift-worker-0-bmc-secret
    namespace: openshift-machine-api
  credentialsVersion: "16120"
```

3.4. 关于 HOSTFIRMWARESETTINGS 资源

您可以使用 **HostFirmwareSettings** 资源来检索和管理主机的 BIOS 设置。当主机进入 **Available** 状态时，Ironic 会读取主机的 BIOS 设置并创建 **HostFirmwareSettings** 资源。资源包含从基板管理控制器(BMC)返回的完整 BIOS 配置。**BareMetalHost** 资源中的 **firmware** 字段会返回三个供应商独立的字段，**HostFirmwareSettings** 资源通常包含每个主机中特定供应商的字段的许多 BIOS 设置。

HostFirmwareSettings 资源包含两个部分：

1. **HostFirmwareSettings** spec。
2. **HostFirmwareSettings** 状态。

3.4.1. HostFirmwareSettings spec

HostFirmwareSettings 资源的 **spec** 部分定义了主机的 BIOS 所需的状态，默认为空。Ironic 使用 **spec.settings** 部分中的设置，在主机处于 **Preparing** 状态时更新基板管理控制器(BMC)。使用 **FirmwareSchema** 资源，确保不向主机发送无效的名称/值对。如需了解更多详细信息，请参阅 "About the FirmwareSchema resource"。

Example

```
spec:
  settings:
    ProcTurboMode: Disabled (1)
```

(1) 在foregoing示例中，**spec.settings** 部分包含一个 name/value 对，它将把 **ProcTurboMode** BIOS 设置为 **Disabled**。

注意

status 部分中列出的整数参数显示为字符串。例如，**"1"**。当在 **spec.settings** 部分中设置整数时，这些值应设置为不带引号的整数。例如，**1**.

3.4.2. HostFirmwareSettings 状态

status 代表主机的 BIOS 的当前状态。

表 3.3. HostFirmwareSettings

参数	描述
status: conditions: - lastTransitionTime: message: observedGeneration: reason: status: type:	**conditions** 字段包含状态更改列表。子字段包括： • **lastTransitionTime** ：状态更改最后一次的时间。 • **message**: 状态更改的描述。 • **observedGeneration** ：当前 **status** 的世代。如果 **metadata.generation** 和此字段不同，则 **status.conditions** 可能已过时。 • **reason**: 状态更改的原因。 • **status**: 状态更改的状态。状态可以是 **True**, **False** 或 **Unknown**。 • **type**: 状态更改的类型。类型为 **Valid** 和 **ChangeDetected**。
status: schema: name: namespace: lastUpdated:	固件设置的 **FirmwareSchema**。这些字段包括： • **name**: 引用该模式的名称或唯一标识符。 • **namespace** ：存储 schema 的命名空间。 • **lastUpdated**: 资源最后一次更新的时间。
status: settings:	**settings** 字段包含主机当前 BIOS 设置的名称/值对列表。

3.5. 获取 HOSTFIRMWARESETTINGS 资源

HostFirmwareSettings 资源包含物理主机的特定于供应商的 BIOS 属性。您必须获取物理主机的 **HostFirmwareSettings** 资源才能查看其 BIOS 属性。

流程

1. 获取 **HostFirmwareSettings** 资源的详细列表：

```
$ oc get hfs -n openshift-machine-api -o yaml
```

注意

您可以使用 **hostfirmwaresettings** 作为 **oc get** 命令的 **hfs** 长形式。

2. 获取 **HostFirmwareSettings** 资源列表：

```
$ oc get hfs -n openshift-machine-api
```

3. 获取特定主机的 **HostFirmwareSettings** 资源

```
$ oc get hfs <host_name> -n openshift-machine-api -o yaml
```

其中 **<host_name>** 是主机的名称。

3.6. 编辑 HOSTFIRMWARESETTINGS 资源

您可以编辑已置备的主机的 **HostFirmwareSettings**。

重要

您只能在主机处于 **provisioned** 状态时编辑主机，不包括只读值。您不能编辑状态为 **externally provisioned** 的主机。

流程

1. 获取 **HostFirmwareSettings** 资源列表：

```
$ oc get hfs -n openshift-machine-api
```

2. 编辑主机的 **HostFirmwareSettings** 资源：

```
$ oc edit hfs <host_name> -n openshift-machine-api
```

其中 **<host_name>** 是置备的主机的名称。**HostFirmwareSettings** 资源将在终端的默认编辑器中打开。

3. 将 name/value 对添加到 **spec.settings** 部分：

Example

```
spec:
  settings:
    name: value (1)
```

(1) 使用 **FirmwareSchema** 资源来识别主机的可用设置。您不能设置只读值。

4. 保存更改并退出编辑器。

5. 获取主机的机器名称：

```
$ oc get bmh <host_name> -n openshift-machine name
```

其中 **<host_name>** 是主机的名称。机器名称会出现在 **CONSUMER** 字段下。

6. 注解机器将其从 machineset 中删除：

```
$ oc annotate machine <machine_name> machine.openshift.io/delete-machine=true -n
openshift-machine-api
```

其中 **<machine_name>** 是要删除的机器的名称。

7. 获取节点列表并计算 worker 节点数量：

```
$ oc get nodes
```

8. 获取 machineset：

```
$ oc get machinesets -n openshift-machine-api
```

9. 缩放 machineset：

```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n-1>
```

其中 **<machineset_name>** 是 machineset 的名称， **<n-1>** 是 worker 节点数量的减少。

10. 当主机进入 **Available** 状态时，扩展 machineset，使 **HostFirmwareSettings** 资源更改生效：

```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n>
```

其中 **<machineset_name>** 是 machineset 的名称， **<n>** 是 worker 节点的数量。

3.7. 验证 HOSTFIRMWARE SETTINGS 资源是否有效

当用户编辑 **spec.settings** 部分以更改为 **HostFirmwareSetting**(HFS)资源时，BMO 会针对 **FimwareSchema** 资源验证更改，这是只读资源。如果设置无效，BMO 会将 **status.Condition** 的 **Type** 值设置为 **False**，并生成事件并将其存储在 HFS 资源中。使用以下步骤验证资源是否有效。

流程

1. 获取 **HostFirmwareSetting** 资源列表：

```
$ oc get hfs -n openshift-machine-api
```

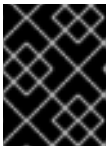
2. 验证特定主机的 **HostFirmwareSettings** 资源是否有效：

```
$ oc describe hfs <host_name> -n openshift-machine-api
```

其中 **<host_name>** 是主机的名称。

输**出示例**

```
Events:
  Type    Reason          Age    From                                Message
  ----    ------          ----   ----                                -------
  Normal  ValidationFailed  2m49s  metal3-hostfirmwaresettings-controller  Invalid BIOS setting: Setting ProcTurboMode is invalid, unknown enumeration value - Foo
```



重要

如果响应返回 **ValidationFailed**，资源配置中会出现一个错误，您必须更新这些值以符合 **FirmwareSchema** 资源。

3.8. 关于 FIRMWARESCHEMA 资源

BIOS 设置因硬件供应商和主机模型而异。**FirmwareSchema** 资源是一个只读资源，其中包含每个主机模型中的每个 BIOS 设置的类型和限值。数据直接通过 Ironic 来自 BMC。**FirmwareSchema** 允许您识别 **HostFirmwareSettings** 资源的 **spec** 字段中可以指定的有效值。**FirmwareSchema** 资源具有从其设置和限值派生的唯一标识符。相同的主机模型使用相同的 **FirmwareSchema** 标识符。**HostFirmwareSettings** 的多个实例可能使用相同的 **FirmwareSchema**。

表 3.4. FirmwareSchema 规格

参数	描述
<BIOS_setting_name> attribute_type: allowable_values: lower_bound: upper_bound: min_length: max_length: read_only: unique:	**spec** 是由 BIOS 设置名称和设置的限制组成的简单映射。这些字段包括： • **attribute_type** ：设置类型。支持的类型有： ◦ **Enumeration** ◦ **整数** ◦ **字符串** ◦ **布尔值** • **allowable_values** ：当 **attribute_type** 是 **Enumeration** 时，允许值的列表。 • **lower_bound** ： **attribute_type** 为 **Integer** 时允许的最小值。 • **upper_bound** ： **attribute_type** 为 **Integer** 时允许的最大值。 • **min_length** ：当 **attribute_type** 为 **String** 时，值的字符串最短长度。 • **max_length** ：当 **attribute_type** 为 **String** 时，值第字符串最长长度。 • **read_only** ： 设置为只读，不可修改。 • **unique** ：该设置特定于此主机。

3.9. 获取 FIRMWARESCHEMA 资源

每个供应商的每个主机模型都有不同的 BIOS 设置。在编辑 **HostFirmwareSettings** 资源的 **spec** 部分时，您设置的名称/值对必须符合该主机的固件模式。要确保设置有效的名称/值对，请获取主机的 **FirmwareSchema** 并查看它。

流程

1. 要获得 **FirmwareSchema** 资源实例列表，请执行以下操作：

```
$ oc get firmwareschema -n openshift-machine-api
```

2. 要获得特定 **FirmwareSchema** 实例，请执行：

```
$ oc get firmwareschema <instance_name> -n openshift-machine-api -o yaml
```

其中 **<instance_name>** 是 **HostFirmwareSettings** 资源中所述的 schema 实例的名称（请参阅表 3）。

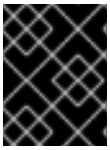
第 4 章 在 OPENSHIFT 集群中配置多架构计算机器

4.1. 关于带有多架构计算机器的集群

带有多架构计算机器的 OpenShift Container Platform 集群是一个支持具有不同架构的计算机器的集群。具有多架构计算机器的集群仅在带有 64 位 x86 control plane 机器的 Amazon Web Services (AWS) 或 Microsoft Azure 安装程序置备的基础架构和裸机、IBM Power 和 IBM Z 用户置备的基础架构上可用。

注意

当集群中有多个架构的节点时，镜像的构架必须与节点的构架一致。您需要确保将 pod 分配给具有适当架构的节点，并将它与镜像架构匹配。如需有关将 pod 分配给节点的更多信息，请参阅将 pod 分配给节点。



重要

具有多架构计算机器的集群中不支持 Cluster Samples Operator。集群可以在没有此功能的情况下创建。如需更多信息，请参阅启用集群功能

有关将单架构集群迁移到支持多架构计算机器的集群的详情，请参考使用多架构计算机器迁移到集群。

4.1.1. 使用多架构计算机器配置集群

要使用不同安装选项和平台的多架构计算机器创建集群，您可以使用下表中的文档：

表 4.1. 带有多架构计算机器安装选项的集群

文档部分	平台	用户置备的安装	安装程序置备的安装	Control Plane	Compute 节点
在 Azure 上使用多架构计算机器创建集群	Microsoft Azure		✓	**x86_64**	**aarch64**
在 AWS 上使用多架构计算机器创建集群	Amazon Web Services (AWS)		✓	**x86_64**	**aarch64**
在 GCP 上使用多架构计算机器创建集群	Google Cloud Platform (GCP)		✓	**x86_64**	**aarch64**
在裸机、IBM Power 或 IBM Z 上使用多架构计算机器创建集群	裸机	✓		**x86_64**	**aarch64**
	IBM Power	✓		**x86_64** 或 **ppc64le**	**x86_64**,**ppc64le**

文档部分	平台	用户置备的安装	安装程序置备的安装	Control Plane	Compute 节点
	IBM Z	✓		**x86_64 或 s390x**	**x86_64,s390x**
在 IBM Z 和使用 z/VM 的 IBM® LinuxONE 上使用多架构计算机器创建集群	IBM Z® 和 IBM® LinuxONE	✓		**x86_64**	**x86_64,s390x**
在带有 RHEL KVM 的 IBM Z 和 IBM® LinuxONE 上使用多架构计算机器创建集群	IBM Z® 和 IBM® LinuxONE	✓		**x86_64**	**x86_64,s390x**
在 IBM Power® 上使用多架构计算机器创建集群	IBM Power®	✓		**x86_64**	**x86_64,ppc64le**

重要

Google Cloud Platform (GCP) 目前不支持从零自动扩展。

4.2. 在 AZURE 上使用多架构计算机器创建集群

要使用多架构计算机器部署 Azure 集群，您必须首先创建一个使用多架构安装程序二进制文件的单架构 Azure 安装程序置备集群。如需有关 Azure 安装的更多信息，请参阅使用自定义在 Azure 上安装集群。然后，您可以在集群中添加 ARM64 计算机器集，以使用多架构计算机器创建集群。

以下流程描述了如何生成 ARM64 引导镜像并创建使用 ARM64 引导镜像的 Azure 计算机器集。这会在集群中添加 ARM64 计算节点，并部署您需要的 ARM64 虚拟机 (VM) 的数量。

4.2.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
```

```
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.2.2. 使用 Azure 镜像 gallery 创建 ARM64 引导镜像

以下流程描述了如何手动生成 ARM64 引导镜像。

先决条件

- 已安装 Azure CLI (**az**)。
- 已使用多架构安装程序二进制文件创建了单架构 Azure 安装程序置备集群。

流程

1. 登录到您的 Azure 帐户：

```
$ az login
```

2. 创建存储帐户并将 **arm64** 虚拟硬盘 (VHD) 上传到您的存储帐户。OpenShift Container Platform 安装程序会创建一个资源组，但引导镜像也可以上传到名为资源组的自定义中：

```
$ az storage account create -n ${STORAGE_ACCOUNT_NAME} -g
${RESOURCE_GROUP} -l westus --sku Standard_LRS (1)
```

(1) **westus** 对象是一个示例区域。

3. 使用您生成的存储帐户创建存储容器：

```
$ az storage container create -n ${CONTAINER_NAME} --account-name
${STORAGE_ACCOUNT_NAME}
```

4. 您必须使用 OpenShift Container Platform 安装程序 JSON 文件提取 URL 和 **aarch64** VHD 名称：

   a. 运行以下命令，提取 **URL** 字段并将其设置为 **RHCOS_VHD_ORIGIN_URL** ：

```
$ RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r
'.architectures.aarch64."rhel-coreos-extensions"."azure-disk".url')
```

b. 运行以下命令，提取 **aarch64** VHD 名称并将其设置为 **BLOB_NAME** 作为文件名：

```
$ BLOB_NAME=rhcos-$(oc -n openshift-machine-config-operator get configmap/coreos-
bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.aarch64."rhel-coreos-
extensions"."azure-disk".release')-azure.aarch64.vhd
```

5. 生成共享访问令牌 (SAS) 令牌。通过以下命令，使用此令牌将 RHCOS VHD 上传到存储容器：

```
$ end=`date -u -d "30 minutes" '+%Y-%m-%dT%H:%MZ'`
```

```
$ sas=`az storage container generate-sas -n ${CONTAINER_NAME} --account-name
${STORAGE_ACCOUNT_NAME} --https-only --permissions dlrw --expiry $end -o tsv`
```

6. 将 RHCOS VHD 复制到存储容器中：

```
$ az storage blob copy start --account-name ${STORAGE_ACCOUNT_NAME} --sas-token
"$sas" \
 --source-uri "${RHCOS_VHD_ORIGIN_URL}" \
 --destination-blob "${BLOB_NAME}" --destination-container ${CONTAINER_NAME}
```

您可以使用以下命令检查复制过程的状态：

```
$ az storage blob show -c ${CONTAINER_NAME} -n ${BLOB_NAME} --account-name
${STORAGE_ACCOUNT_NAME} | jq .properties.copy
```

输**出示例**

```
{
 "completionTime": null,
 "destinationSnapshot": null,
 "id": "1fd97630-03ca-489a-8c4e-cfe839c9627d",
 "incrementalCopy": null,
 "progress": "17179869696/17179869696",
 "source": "https://rhcos.blob.core.windows.net/imagebucket/rhcos-411.86.202207130959-0-
azure.aarch64.vhd",
 "status": "success", (1)
 "statusDescription": null
}
```

(1) 如果 status 参数显示 **success** 对象，则复制过程已完成。

7. 使用以下命令创建镜像 gallery：

```
$ az sig create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME}
```

使用镜像 gallery 创建镜像定义。在以下示例中，**rhcos-arm64** 是镜像定义的名称。

```
$ az sig image-definition create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --publisher RedHat --offer arm -
-sku arm64 --os-type linux --architecture Arm64 --hyper-v-generation V2
```

8. 要获取 VHD 的 URL，并将其设置为 **RHCOS_VHD_URL** 作为文件名称，请运行以下命令：

```
$ RHCOS_VHD_URL=$(az storage blob url --account-name
${STORAGE_ACCOUNT_NAME} -c ${CONTAINER_NAME} -n "${BLOB_NAME}" -o tsv)
```

9. 使用 **RHCOS_VHD_URL** 文件、您的存储帐户、资源组和镜像 gallery 创建镜像版本。在以下示例中，**1.0.0** 是镜像版本。

```
$ az sig image-version create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --gallery-image-version 1.0.0 --
os-vhd-storage-account ${STORAGE_ACCOUNT_NAME} --os-vhd-uri
${RHCOS_VHD_URL}
```

10. 现在生成您的 **arm64** 引导镜像。您可以使用以下命令访问镜像的 ID：

```
$ az sig image-version show -r $GALLERY_NAME -g $RESOURCE_GROUP -i rhcos-arm64
-e 1.0.0
```

以下示例镜像 ID 在计算机器设置的 **recourseID** 参数中使用：

resourceID 示例

```
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-arm64/versions/1.0.0
```

4.2.3. 在集群中添加多架构计算机器集

要在集群中添加 ARM64 计算节点，您必须创建一个使用 ARM64 引导镜像的 Azure 计算机器集。要在 Azure 上创建自己的自定义计算机器集，请参阅"创建 Azure 上的计算机器集"。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

- 使用以下命令，创建计算机器集并修改 **resourceID** 和 **vmSize** 参数。此计算机器集将控制集群中的 **arm64** worker 节点：

```
$ oc create -f arm64-machine-set-0.yaml
```

使用 arm64 引导镜像的 YAML 计算机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
```

```
    machine.openshift.io/cluster-api-machine-role: worker
    machine.openshift.io/cluster-api-machine-type: worker
  name: <infrastructure_id>-arm64-machine-set-0
  namespace: openshift-machine-api
spec:
  replicas: 2
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-arm64-machine-set-0
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-arm64-machine-set-0
    spec:
      lifecycleHooks: {}
      metadata: {}
      providerSpec:
        value:
          acceleratedNetworking: true
          apiVersion: machine.openshift.io/v1beta1
          credentialsSecret:
            name: azure-cloud-credentials
            namespace: openshift-machine-api
          image:
            offer: ""
            publisher: ""
            resourceID:
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-arm64/versions/1.0.0 (1)
            sku: ""
            version: ""
          kind: AzureMachineProviderSpec
          location: <region>
          managedIdentity: <infrastructure_id>-identity
          networkResourceGroup: <infrastructure_id>-rg
          osDisk:
            diskSettings: {}
            diskSizeGB: 128
            managedDisk:
              storageAccountType: Premium_LRS
            osType: Linux
          publicIP: false
          publicLoadBalancer: <infrastructure_id>
          resourceGroup: <infrastructure_id>-rg
          subnet: <infrastructure_id>-worker-subnet
          userDataSecret:
            name: worker-user-data
          vmSize: Standard_D4ps_v5 (2)
          vnet: <infrastructure_id>-vnet
          zone: "<zone>"
```

1. 将 **resourceID** 参数设置为 **arm64** 引导镜像。
2. 将 **vmSize** 参数设置为安装中使用的实例类型。一些实例类型是 **Standard_D4ps_v5** 或 **D8ps**。

验证

1. 输入以下命令验证新的 ARM64 机器是否正在运行：

```
$ oc get machineset -n openshift-machine-api
```

输出示例

```
NAME                                DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-arm64-machine-set-0   2        2      2          2  10m
```

2. 您可以使用以下命令检查节点是否就绪并可访问：

```
$ oc get nodes
```

其他资源

- 在 Azure 上创建计算机器设置

4.3. 在 AWS 上使用多架构计算机器创建集群

要使用多架构计算机器创建 AWS 集群，您必须首先使用多架构安装程序二进制文件创建一个单架构 AWS 安装程序置备集群。如需有关 AWS 安装的更多信息，请参阅使用自定义在 AWS 上安装集群。然后，您可以在 AWS 集群中添加 ARM64 计算机器集。

4.3.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

1. 如果您看到以下输出，集群将使用多架构有效负载：

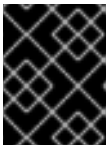
```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
```

```
}
```

然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.3.2. 在集群中添加 ARM64 计算机器集

要使用多架构计算机器配置集群，您必须创建一个 AWS ARM64 计算机器集。这会在集群中添加 ARM64 计算节点，以便集群有多架构计算机器。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用安装程序创建带有多架构安装程序二进制文件的 AMD64 单架构 AWS 集群。

流程

- 创建和修改计算机器集，这将控制集群中的 ARM64 计算节点。

```
$ oc create -f aws-arm64-machine-set-0.yaml
```

部署 ARM64 计算节点的 YAML 计算机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-aws-arm64-machine-set-0 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role> 5
        machine.openshift.io/cluster-api-machine-type: <role> 6
```

```
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 7
spec:
  metadata:
    labels:
      node-role.kubernetes.io/<role>: ""
  providerSpec:
    value:
      ami:
        id: ami-02a574449d4f4d280 8
      apiVersion: awsproviderconfig.openshift.io/v1beta1
      blockDevices:
        - ebs:
            iops: 0
            volumeSize: 120
            volumeType: gp2
      credentialsSecret:
        name: aws-cloud-credentials
      deviceIndex: 0
      iamInstanceProfile:
        id: <infrastructure_id>-worker-profile 9
      instanceType: m6g.xlarge 10
      kind: AWSMachineProviderConfig
      placement:
        availabilityZone: us-east-1a 11
        region: <region> 12
      securityGroups:
        - filters:
            - name: tag:Name
              values:
                - <infrastructure_id>-worker-sg 13
      subnet:
        filters:
          - name: tag:Name
            values:
              - <infrastructure_id>-private-<zone>
      tags:
        - name: kubernetes.io/cluster/<infrastructure_id> 14
          value: owned
        - name: <custom_tag_name>
          value: <custom_tag_value>
      userDataSecret:
        name: worker-user-data
```

1 2 3 9 13 14 指定基于置备集群时所设置的集群 ID 的基础架构 ID。如果已安装 OpenShift CLI，您可以通过运行以下命令来获取基础架构 ID：

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

4 7 指定基础架构 ID、角色节点标签和区域。

5 6 指定要添加的角色节点标签。

8 为 OpenShift Container Platform 节点的 AWS 区域指定支持 Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Image (AMI) 的 ARM64。

```
$ oc get configmap/coreos-bootimages \
  -n openshift-machine-config-operator \
  -o jsonpath='{.data.stream}' | jq \
  -r '.architectures.<arch>.images.aws.regions."<region>".image'
```

10 指定支持 ARM64 的机器类型。如需更多信息，请参阅"为 AWS 64 位 ARM 测试实例类型"

11 指定区域，如 **us-east-1a**。确保您选择的区域提供 64 位 ARM 机器。

12 指定区域，如 **us-east-1**。确保您选择的区域提供 64 位 ARM 机器。

验证

1. 输入以下命令来查看计算机器集列表：

```
$ oc get machineset -n openshift-machine-api
```

然后，您可以看到创建的 ARM64 机器集。

输**出示例**

```
NAME                                        DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-aws-arm64-machine-set-0              2        2      2          2  10m
```

2. 您可以使用以下命令检查节点是否就绪并可访问：

```
$ oc get nodes
```

其他资源

- 为 AWS 64 位 ARM 测试的实例类型

4.4. 在 GCP 上使用多架构计算机器创建集群

要使用多架构计算机器创建 Google Cloud Platform (GCP)集群，您必须首先使用多架构安装程序二进制文件创建一个单架构 GCP 安装程序置备集群。如需有关 AWS 安装的更多信息，请参阅使用自定义在 GCP 上安装集群。然后，您可以在 GCP 集群中添加 ARM64 计算机器集。

注意

GCP 的 ARM64 机器目前不支持安全引导

4.4.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

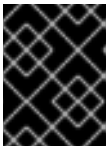
1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.4.2. 在 GCP 集群中添加 ARM64 计算机器集

要使用多架构计算机器配置集群，您必须创建一个 GCP ARM64 计算机器集。这会在集群中添加 ARM64 计算节点。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用安装程序创建带有多架构安装程序二进制文件的 AMD64 单架构 AWS 集群。

流程

- 创建和修改计算机器集，这会控制集群中的 ARM64 计算节点：

```
$ oc create -f gcp-arm64-machine-set-0.yaml
```

部署 ARM64 计算节点的 GCP YAML 计算机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-w-a
  namespace: openshift-machine-api
spec:
```

```
replicas: 1
selector:
  matchLabels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-w-a
template:
  metadata:
    creationTimestamp: null
    labels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-w-a
  spec:
    metadata:
      labels:
        node-role.kubernetes.io/<role>: ""
    providerSpec:
      value:
        apiVersion: gcpprovider.openshift.io/v1beta1
        canIPForward: false
        credentialsSecret:
          name: gcp-cloud-credentials
        deletionProtection: false
        disks:
        - autoDelete: true
          boot: true
          image: <path_to_image> 3
          labels: null
          sizeGb: 128
          type: pd-ssd
        gcpMetadata: 4
        - key: <custom_metadata_key>
          value: <custom_metadata_value>
        kind: GCPMachineProviderSpec
        machineType: n1-standard-4 5
        metadata:
          creationTimestamp: null
        networkInterfaces:
        - network: <infrastructure_id>-network
          subnetwork: <infrastructure_id>-worker-subnet
        projectID: <project_name> 6
        region: us-central1 7
        serviceAccounts:
        - email: <infrastructure_id>-w@<project_name>.iam.gserviceaccount.com
          scopes:
          - https://www.googleapis.com/auth/cloud-platform
        tags:
          - <infrastructure_id>-worker
        userDataSecret:
          name: worker-user-data
        zone: us-central1-a
```

1 指定基于置备集群时所设置的集群 ID 的基础架构 ID。您可以运行以下命令来获取基础架构 ID:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 指定要添加的角色节点标签。

3 指定当前计算机器集中使用的镜像的路径。您需要到镜像的路径的项目和镜像名称。

要访问项目和镜像名称，请运行以下命令：

```
$ oc get configmap/coreos-bootimages \
  -n openshift-machine-config-operator \
  -o jsonpath='{.data.stream}' | jq \
  -r '.architectures.aarch64.images.gcp'
```

输**出示例**

```
"gcp": {
  "release": "415.92.202309142014-0",
  "project": "rhcos-cloud",
  "name": "rhcos-415-92-202309142014-0-gcp-aarch64"
}
```

使用输出中的 **project** 和 **name** 参数，在机器集中创建 image 字段的路径。镜像的路径应该采用以下格式：

```
$ projects/<project>/global/images/<image_name>
```

4 可选：以 **key:value** 对的形式指定自定义元数据。有关用例，请参阅 GCP 文档，以查看设置自定义元数据。

5 指定支持 ARM64 的机器类型。如需更多信息，请参阅"添加资源"的 *64 位 ARM 基础架构上为 GCP 测试的实例类型*。

6 指定用于集群的 GCP 项目的名称。

7 指定区域，如 **us-central1**。确保您选择的区域提供 64 位 ARM 机器。

验证

1. 输入以下命令来查看计算机器集列表：

```
$ oc get machineset -n openshift-machine-api
```

然后，您可以看到创建的 ARM64 机器集。

输**出示例**

```
NAME                                DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-gcp-arm64-machine-set-0   2        2      2          2  10m
```

2. 您可以使用以下命令检查节点是否就绪并可访问：

```
$ oc get nodes
```

其他资源

- 在 64 位 ARM 基础架构上为 GCP 测试的实例类型

4.5. 在裸机、IBM POWER 或 IBM Z 上使用多架构计算机器创建集群

要在裸机 (**x86_64**)、IBM Power® (**ppc64le**) 或 IBM Z® (**s390x**) 上使用多架构计算机器创建集群，您必须在其中一个平台上有一个现有的单架构集群。按照您的平台安装过程操作：

- 在裸机上安装用户置备的集群。然后，您可以将 64 位 ARM 计算机器添加到裸机上的 OpenShift Container Platform 集群中。
- 在 IBM Power® 上安装集群 。然后，您可以将 **x86_64** 计算机器添加到 IBM Power® 上的 OpenShift Container Platform 集群中。
- 在 IBM Z® 和 IBM® LinuxONE 上安装集群 。然后，您可以将 **x86_64** 计算机器添加到 IBM Z® 和 IBM® LinuxONE 上的 OpenShift Container Platform 集群中。

在为集群添加额外的计算节点前，您必须将集群升级到使用多架构有效负载的节点。有关迁移到多架构有效负载的更多信息，请参阅迁移到具有多架构计算机器的集群。

以下流程解释了如何使用 ISO 镜像或网络 PXE 引导创建 RHCOS 计算机器。这将允许您向集群添加额外的节点，并使用多架构计算机器部署集群。

4.5.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

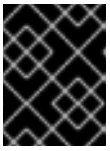
1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

   然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.5.2. 使用 ISO 镜像创建 RHCOS 机器

您可以使用 ISO 镜像为裸机集群创建更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器，以创建机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 已安装 OpenShift CLI (**oc**)。

流程

1. 运行以下命令，从集群中删除 Ignition 配置文件：

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

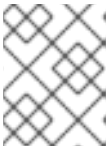
2. 将您从集群导出的 **worker.ign** Ignition 配置文件上传到 HTTP 服务器。注意这些文件的 URL。
3. 您可以验证 ignition 文件是否在 URL 上可用。以下示例获取计算节点的 Ignition 配置文件：

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. 您可以运行以下命令来访问 ISO 镜像以引导新机器：

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 使用 ISO 文件在更多计算机器上安装 RHCOS。在安装集群前，使用创建机器时使用的相同方法：
   - 将 ISO 镜像刻录到磁盘并直接启动。
   - 在 LOM 接口中使用 ISO 重定向。
6. 在不指定任何选项或中断实时引导序列的情况下引导 RHCOS ISO 镜像。等待安装程序在 RHCOS live 环境中引导进入 shell 提示符。



注意

您可以中断 RHCOS 安装引导过程来添加内核参数。但是，在这个 ISO 过程中，您应该使用以下步骤中所述的 **coreos-installer** 命令，而不是添加内核参数。

7. 运行 **coreos-installer** 命令并指定满足您的安装要求的选项。您至少必须指定指向节点类型的 Ignition 配置文件的 URL，以及您要安装到的设备：

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

1 您必须使用 **sudo** 运行 **coreos-installer** 命令，因为 **core** 用户没有执行安装所需的 root 权限。

2 当 Ignition 配置文件通过 HTTP URL 获取时，需要 **--ignition-hash** 选项来验证集群节点上 Ignition 配置文件的真实性。**<digest>** 是上一步中获取的 Ignition 配置文件 SHA512 摘要。

注意

如果要通过使用 TLS 的 HTTPS 服务器提供 Ignition 配置文件，您可以在运行 **coreos-installer** 前将内部证书颁发机构(CA)添加到系统信任存储中。

以下示例将引导节点安装初始化到 **/dev/sda** 设备。bootstrap 节点的 Ignition 配置文件从 IP 地址 192.168.1.2 的 HTTP Web 服务器获取：

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. 在机器的控制台上监控 RHCOS 安装的进度。

重要

在开始安装 OpenShift Container Platform 之前，确保每个节点中安装成功。观察安装过程可以帮助确定可能会出现 RHCOS 安装问题的原因。

9. 继续为集群创建更多计算机器。

4.5.3. 通过 PXE 或 iPXE 启动来创建 RHCOS 机器

您可以使用 PXE 或 iPXE 引导为裸机集群创建更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 获取您在集群安装过程中上传到 HTTP 服务器的 RHCOS ISO 镜像、压缩的裸机 BIOS、**kernel** 和 **initramfs** 文件的 URL。
- 您可以访问在安装过程中为 OpenShift Container Platform 集群创建机器时使用的 PXE 引导基础架构。机器必须在安装 RHCOS 后从本地磁盘启动。
- 如果使用 UEFI，您可以访问在 OpenShift Container Platform 安装过程中修改的 **grub.conf** 文件。

流程

1. 确认 RHCOS 镜像的 PXE 或 iPXE 安装正确。

- 对于 PXE：

```
DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> (1)
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img (2)
```

(1) 指定上传到 HTTP 服务器的 live **kernel** 文件位置。

(2) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**initrd** 参数值是 live **initramfs** 文件的位置，**coreos.inst.ignition_url** 参数值是 worker Ignition 配置文件的位置，**coreos.live.rootfs_url** 参数值是 live **rootfs** 文件的位置。**coreos.inst.ignition_url** 和 **coreos.live.rootfs_url** 参数仅支持 HTTP 和 HTTPS。

注意

此配置不会在图形控制台的机器上启用串行控制台访问。要配置不同的控制台，请在 **APPEND** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅 如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？

- 对于 iPXE (**x86_64** + **aarch64**)：

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img (3)
boot
```

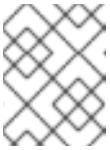
(1) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 **kernel** 文件的位置，在 UEFI 系统中引导时需要 **initrd=main** 参数，**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位置，**coreos.inst.ignition_url** 参数值则是 worker Ignition 配置文件的位置。

(2) 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。

(3) 指定上传到 HTTP 服务器的 **initramfs** 文件的位置。

注意

此配置不会在使用图形控制台配置不同的控制台的机器上启用串行控制台访问，请在 **kernel** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？ 和"启用 PXE 和 ISO 安装的串行控制台"部分。



注意

要在 **aarch64** 架构中网络引导 CoreOS **内核**，您需要使用启用了 **IMAGE_GZIP** 选项的 iPXE 构建版本。请参阅 iPXE 中的**IMAGE_GZIP** 选项。

- 对于 **aarch64** 中的 PXE（使用 UEFI 和 GRUB 作为第二阶段）：

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
    initrd rhcos-<version>-live-initramfs.<architecture>.img 3
}
```

1. 指定上传到 HTTP/TFTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 TFTP 服务器中的 **kernel** 文件的位置。**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位置，**coreos.inst.ignition_url** 参数值则是 HTTP 服务器上的 worker Ignition 配置文件的位置。
2. 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。
3. 指定上传到 TFTP 服务器的 **initramfs** 文件的位置。

2. 使用 PXE 或 iPXE 基础架构为集群创建所需的计算机器。

4.5.4. 批准机器的证书签名请求

当您将机器添加到集群时，会为您添加的每台机器生成两个待处理证书签名请求(CSR)。您必须确认这些 CSR 已获得批准，或根据需要自行批准。必须首先批准客户端请求，然后批准服务器请求。

先决条件

- 您已将机器添加到集群中。

流程

1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输**出示例**

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.28.5
master-1 Ready    master  63m  v1.28.5
master-2 Ready    master  64m  v1.28.5
```

输出中列出了您创建的所有机器。

注意

在有些 CSR 被批准前，前面的输出可能不包括计算节点（也称为 worker 节点）。

2. 检查待处理的 CSR，并确保添加到集群中的每台机器都有 **Pending** 或 **Approved** 状态的客户端请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE    REQUESTOR                                                                   CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
...
```

在本例中，两台机器加入集群。您可能会在列表中看到更多已批准的 CSR。

3. 如果 CSR 没有获得批准，在您添加的机器的所有待处理 CSR 都处于 **Pending 状态** 后，请批准集群机器的 CSR：

注意

由于 CSR 会自动轮转，因此请在将机器添加到集群后一小时内批准您的 CSR。如果没有在一小时内批准它们，证书将会轮转，每个节点会存在多个证书。您必须批准所有这些证书。批准客户端 CSR 后，Kubelet 为服务证书创建一个二级 CSR，这需要手动批准。然后，如果 Kubelet 请求具有相同参数的新证书，则后续提供证书续订请求由 **machine-approver** 自动批准。

注意

对于在未启用机器 API 的平台上运行的集群，如裸机和其他用户置备的基础架构，您必须实施一种方法来自动批准 kubelet 提供证书请求(CSR)。如果没有批准请求，则 **oc exec**、**oc rsh** 和 **oc logs** 命令将无法成功，因为 API 服务器连接到 kubelet 时需要服务证书。与 Kubelet 端点联系的任何操作都需要此证书批准。该方法必须监视新的 CSR，确认 CSR 由 system: **node** 或 **system:admin** 组中的 **node-bootstrapper** 服务帐户提交，并确认节点的身份。

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

注意

在有些 CSR 被批准前，一些 Operator 可能无法使用。

4. 现在，您的客户端请求已被批准，您必须查看添加到集群中的每台机器的服务器请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE     REQUESTOR                                                   CONDITION
csr-bfd72   5m26s   system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv   5m26s   system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. 如果剩余的 CSR 没有被批准，且处于 **Pending** 状态，请批准集群机器的 CSR：

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 批准所有客户端和服务器 CSR 后，机器将 处于 **Ready 状态**。运行以下命令验证：

```
$ oc get nodes
```

输**出示例**

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  73m  v1.28.5
master-1  Ready     master  73m  v1.28.5
master-2  Ready     master  74m  v1.28.5
worker-0  Ready     worker  11m  v1.28.5
worker-1  Ready     worker  11m  v1.28.5
```

注意

批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready 状态**。

其他信息

- 如需有关 CSR 的更多信息，请参阅 证书签名请求。

4.6. 在带有 Z/VM 的 IBM Z 和 IBM LINUXONE 上使用多架构计算机器创建集群

要使用 IBM® Z 和 IBM® LinuxONE (**s390x**)上的多架构计算机器创建集群，您必须使用现有的单架构 **x86_64** 集群。然后，您可以在 OpenShift Container Platform 集群中添加 **s390x** 计算机器。

在为集群添加 **s390x** 节点前，您必须将集群升级到使用多架构有效负载的节点。有关迁移到多架构有效负载的更多信息，请参阅迁移到具有多架构计算机器的集群。

以下流程描述了如何使用 z/VM 实例创建 RHCOS 计算机器。这将允许您在集群中添加 **s390x** 节点，并使用多架构计算机器部署集群。

注意

要使用 **x86_64** 上的多架构计算机器创建 IBM Z® 或 IBM® LinuxONE (**s390x**) 集群，请按照在 IBM Z® 和 IBM® LinuxONE 上安装集群 的说明进行操作。然后，您可以添加 **x86_64** 计算机器，如在裸机、IBM Power 或 IBM Z 上创建带有多架构计算机器的集群 中所述。

4.6.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

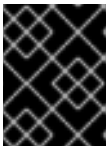
1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

   然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.6.2. 使用 z/VM 在 IBM Z 上创建 RHCOS 机器

您可以使用 z/VM 创建在 IBM Z® 上运行的更多 Red Hat Enterprise Linux CoreOS (RHCOS)计算机器，并将它们附加到现有集群。

先决条件

- 您有一个域名服务器(DNS)，它可以对节点执行主机名和反向查找。
- 您有在置备机器上运行的 HTTP 或 HTTPS 服务器，可供您创建的机器访问。

流程

1. 禁用 UDP 聚合。
   目前，IBM Z® 不支持 UDP 聚合，且不会在带有 **x86_64** control plane 和其他 **s390x** 计算机器的多架构计算机器上自动取消激活 UDP 聚合。为确保正确在集群中添加额外的计算节点，您必须手动禁用 UDP 聚合。

   a. 使用以下内容创建 YAML 文件 **udp-aggregation-config.yaml** ：

   ```
   apiVersion: v1
   kind: ConfigMap
   data:
     disable-udp-aggregation: "true"
   metadata:
     name: udp-aggregation-config
     namespace: openshift-network-operator
   ```

   b. 运行以下命令来创建 ConfigMap 资源：

   ```
   $ oc create -f udp-aggregation-config.yaml
   ```

2. 运行以下命令，从集群中删除 Ignition 配置文件：

   ```
   $ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
   to=- > worker.ign
   ```

3. 将您从集群导出的 **worker.ign** Ignition 配置文件上传到 HTTP 服务器。注意此文件的 URL。

4. 您可以验证 Ignition 文件是否在 URL 上可用。以下示例获取计算节点的 Ignition 配置文件：

   ```
   $ curl -k http://<HTTP_server>/worker.ign
   ```

5. 运行以下命令下载 RHEL live **kernel**、**initramfs** 和 **rootfs** 文件：

   ```
   $ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
   jsonpath='{.data.stream}' \
   | jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
   ```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

6. 将下载的 RHEL live **kernel**、**initramfs** 和 **rootfs** 文件移到可从您要添加的 z/VM 客户机访问的 HTTP 或 HTTPS 服务器中。

7. 为 z/VM 客户机创建一个参数文件。以下参数特定于虚拟机：

   - 可选：要指定一个静态 IP 地址，请添加带有以下条目的 **ip=** 参数，每个条目用冒号隔开：

     i. 计算机的 IP 地址。

     ii. 个空字符串。

     iii. 网关

     iv. 子网掩码.

     v. **hostname.domainname** 格式的机器主机和域名。省略这个值可让 RHCOS 决定。

     vi. 网络接口名称。省略这个值可让 RHCOS 决定。

     vii. 值 **none**。

   - 对于 **coreos.inst.ignition_url=**，请指定 **worker.ign** 文件的 URL。仅支持 HTTP 和 HTTPS 协议。

   - 对于 **coreos.live.rootfs_url=**，请为您引导的 **kernel** 和 **initramfs** 指定匹配的 rootfs 构件。仅支持 HTTP 和 HTTPS 协议。

   - 对于在 DASD 类型磁盘中安装，请完成以下任务：

     i. 对于 **coreos.inst.install_dev=**，请指定 **/dev/dasda**。

     ii. Userd **.dasd=** 指定安装 RHCOS 的 DASD。

     iii. 所有其他参数保持不变。
     以下是一个示例参数文件 **additional-worker-dasd.parm** ：

     ```
     rd.neednet=1 \
     console=ttysclp0 \
     coreos.inst.install_dev=/dev/dasda \
     coreos.live.rootfs_url=http://cl1.provide.example.com:8080/assets/rhcos-live-
     rootfs.s390x.img \
     coreos.inst.ignition_url=http://cl1.provide.example.com:8080/ignition/worker.ign \
     ip=172.18.78.2::172.18.78.1:255.255.255.0:::none nameserver=172.18.78.1 \
     rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
     zfcp.allow_lun_scan=0 \
     rd.dasd=0.0.3490
     ```

     将参数文件中的所有选项写为一行，并确保您没有换行字符。

- 对于在 FCP 类型磁盘中安装，请完成以下任务：

  i. User **d.zfcp=<adapter>,<wwpn>,<lun>** 以指定要安装 RHCOS 的 FCP 磁盘。对于多路径，为每个额外路径重复此步骤。

  注意

  当使用多个路径安装时，您必须在安装后直接启用多路径，而不是在以后启用多路径，因为这可能导致问题。

  ii. 将安装设备设置为：**coreos.inst.install_dev=/dev/sda**。

  注意

  如果使用 NPIV 配置额外的 LUN，FCP 需要 **zfcp.allow_lun_scan=0**。如果必须启用 **zfcp.allow_lun_scan=1**，因为您使用 CSI 驱动程序，则必须配置 NPIV，以便每个节点无法访问另一个节点的引导分区。

  iii. 所有其他参数保持不变。

  重要

  需要额外的安装后步骤才能完全启用多路径。如需更多信息，请参阅 *安装后机器配置任务* 中的"使用 RHCOS 上内核参数启用多路径"。

  以下是使用多路径的 worker 节点的 **additional-worker-fcp.parm** 示例参数文件：

```
rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/sda \
coreos.live.rootfs_url=http://cl1.provide.example.com:8080/assets/rhcos-live-rootfs.s390x.img \
coreos.inst.ignition_url=http://cl1.provide.example.com:8080/ignition/worker.ign \
ip=172.18.78.2::172.18.78.1:255.255.255.0:::none nameserver=172.18.78.1 \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcp.allow_lun_scan=0 \
rd.zfcp=0.0.1987,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.1987,0x50050763071bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763071bc5e3,0x4008400B00000000
```

  将参数文件中的所有选项写为一行，并确保您没有换行字符。

8. 使用 FTP 将 **initramfs**、**kernel**、参数文件和 RHCOS 镜像传送到 z/VM 中。有关如何使用 FTP 传输文件并从虚拟读取器引导的详情，请参阅 在 Z/VM 中安装。

9. 将文件 punch 到 z/VM 虚拟机的虚拟读取器。
   请参阅 IBM® 文档中的 PUNCH。

   提示

   您可以使用 CP PUNCH 命令，或者，如果使用 Linux，则使用 **vmur** 命令在两个 z/VM 虚拟机之间传输文件。

10. 在 bootstrap 机器上登录到 CMS。

11. 运行以下命令，从 reader IPL bootstrap 机器：

```
$ ipl c
```

请参阅 IBM® 文档中的 IPL。

4.6.3. 批准机器的证书签名请求

当您将机器添加到集群时，会为您添加的每台机器生成两个待处理证书签名请求(CSR)。您必须确认这些 CSR 已获得批准，或根据需要自行批准。必须首先批准客户端请求，然后批准服务器请求。

先决条件

- 您已将机器添加到集群中。

流程

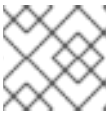
1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输**出示例**

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.28.5
master-1 Ready    master  63m  v1.28.5
master-2 Ready    master  64m  v1.28.5
```

输出中列出了您创建的所有机器。



注意

在有些 CSR 被批准前，前面的输出可能不包括计算节点（也称为 worker 节点）。

2. 检查待处理的 CSR，并确保添加到集群中的每台机器都有 **Pending** 或 **Approved** 状态的客户端请求：

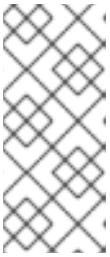
```
$ oc get csr
```

输**出示例**

```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
...
```

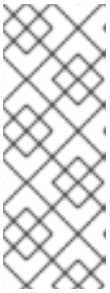
在本例中，两台机器加入集群。您可能会在列表中看到更多已批准的 CSR。

3. 如果 CSR 没有获得批准，在您添加的机器的所有待处理 CSR 都处于 **Pending 状态** 后，请批准集群机器的 CSR：



注意

由于 CSR 会自动轮转，因此请在将机器添加到集群后一小时内批准您的 CSR。如果没有在一小时内批准它们，证书将会轮转，每个节点会存在多个证书。您必须批准所有这些证书。批准客户端 CSR 后，Kubelet 为服务证书创建一个二级 CSR，这需要手动批准。然后，如果 Kubelet 请求具有相同参数的新证书，则后续提供证书续订请求由 **machine-approver** 自动批准。



注意

对于在未启用机器 API 的平台上运行的集群，如裸机和其他用户置备的基础架构，您必须实施一种方法来自动批准 kubelet 提供证书请求(CSR)。如果没有批准请求，则 **oc exec**、oc**rsh** 和 **oc logs** 命令将无法成功，因为 API 服务器连接到 kubelet 时需要服务证书。与 Kubelet 端点联系的任何操作都需要此证书批准。该方法必须监视新的 CSR，确认 CSR 由 system: **node** 或 **system:admin** 组中的 **node-bootstrapper** 服务帐户提交，并确认节点的身份。

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

注意

在有些 CSR 被批准前，一些 Operator 可能无法使用。

4. 现在，您的客户端请求已被批准，您必须查看添加到集群中的每台机器的服务器请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-bfd72   5m26s  system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv   5m26s  system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. 如果剩余的 CSR 没有被批准，且处于 **Pending** 状态，请批准集群机器的 CSR：

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 批准所有客户端和服务器 CSR 后，机器将 处于 **Ready 状态**。运行以下命令验证：

```
$ oc get nodes
```

输**出示例**

```
NAME     STATUS   ROLES   AGE  VERSION
master-0  Ready    master  73m  v1.28.5
master-1  Ready    master  73m  v1.28.5
master-2  Ready    master  74m  v1.28.5
worker-0  Ready    worker  11m  v1.28.5
worker-1  Ready    worker  11m  v1.28.5
```

注意

批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready 状态**。

其他信息

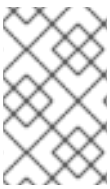
- 如需有关 CSR 的更多信息，请参阅 证书签名请求。

4.7. 使用 RHEL KVM 在 IBM Z 和 IBM LINUXONE 上使用多架构计算机器创建集群

要使用 RHEL KVM 在 IBM Z® 和 IBM® LinuxONE (**s390x**) 上使用多架构计算机器创建集群，您必须有一个现有的单架构 **x86_64** 集群。然后，您可以在 OpenShift Container Platform 集群中添加 **s390x** 计算机器。

在为集群添加 **s390x** 节点前，您必须将集群升级到使用多架构有效负载的节点。有关迁移到多架构有效负载的更多信息，请参阅迁移到具有多架构计算机器的集群。

以下流程描述了如何使用 RHEL KVM 实例创建 RHCOS 计算机器。这将允许您在集群中添加 **s390x** 节点，并使用多架构计算机器部署集群。



注意

要使用 **x86_64** 上的多架构计算机器创建 IBM Z® 或 IBM® LinuxONE (**s390x**) 集群，请按照在 IBM Z® 和 IBM® LinuxONE 上安装集群 的说明进行操作。然后，您可以添加 **x86_64** 计算机器，如在裸机、IBM Power 或 IBM Z 上创建带有多架构计算机器的集群 中所述。

4.7.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

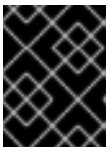
1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

   然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.7.2. 使用 virt-install 创建 RHCOS 机器

您可以使用 **virt-install** 为集群创建更多 Red Hat Enterprise Linux CoreOS (RHCOS)计算机器。

先决条件

- 您至少有一个 LPAR 在 KVM 的 RHEL 8.7 或更高版本中运行，在此过程中称为 RHEL KVM 主机。
- KVM/QEMU 管理程序安装在 RHEL KVM 主机上。
- 您有一个域名服务器(DNS)，它可以对节点执行主机名和反向查找。
- 设置了 HTTP 或 HTTPS 服务器。

流程

1. 禁用 UDP 聚合。

目前，IBM Z® 不支持 UDP 聚合，且不会在带有 **x86_64** control plane 和其他 **s390x** 计算机器的多架构计算机器上自动取消激活 UDP 聚合。为确保正确在集群中添加额外的计算节点，您必须手动禁用 UDP 聚合。

a. 使用以下内容创建 YAML 文件 **udp-aggregation-config.yaml** ：

```
apiVersion: v1
kind: ConfigMap
data:
  disable-udp-aggregation: "true"
metadata:
  name: udp-aggregation-config
  namespace: openshift-network-operator
```

b. 运行以下命令来创建 ConfigMap 资源：

```
$ oc create -f udp-aggregation-config.yaml
```

2. 运行以下命令，从集群中删除 Ignition 配置文件：

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

3. 将您从集群导出的 **worker.ign** Ignition 配置文件上传到 HTTP 服务器。注意此文件的 URL。

4. 您可以验证 Ignition 文件是否在 URL 上可用。以下示例获取计算节点的 Ignition 配置文件：

```
$ curl -k http://<HTTP_server>/worker.ign
```

5. 运行以下命令下载 RHEL live **kernel**、**initramfs** 和 **rootfs** 文件：

```
 $ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

6. 在启动 **virt-install** 前，将下载的 RHEL live **kernel**、**initramfs** 和 **rootfs** 文件移到 HTTP 或 HTTPS 服务器中。

7. 使用 RHEL **kernel**、**initramfs** 和 Ignition 文件创建新的 KVM 客户机节点；新磁盘镜像，并调整了 parm 行参数。

```
$ virt-install \
  --connect qemu:///system \
  --name <vm_name> \
  --autostart \
```

```
  --os-variant rhel9.2 \ 1
  --cpu host \
  --vcpus <vcpus> \
  --memory <memory_mb> \
  --disk <vm_name>.qcow2,size=<image_size> \
  --network network=<virt_network_parm> \
  --location <media_location>,kernel=<rhcos_kernel>,initrd=<rhcos_initrd> \ 2
  --extra-args "rd.neednet=1" \
  --extra-args "coreos.inst.install_dev=/dev/vda" \
  --extra-args "coreos.inst.ignition_url=<worker_ign>" \ 3
  --extra-args "coreos.live.rootfs_url=<rhcos_rootfs>" \ 4
  --extra-args "ip=<ip>::<default_gateway>:<subnet_mask_length>:<hostname>::none:
<MTU>" \ 5
  --extra-args "nameserver=<dns>" \
  --extra-args "console=ttysclp0" \
  --noautoconsole \
  --wait
```

1 对于 **os-variant**，为 RHCOS 计算机器指定 RHEL 版本。 **rhel9.2** 是推荐的版本。要查询支持的操作系统的 RHEL 版本，请运行以下命令：

```
$ osinfo-query os -f short-id
```

注意

os-variant 是区分大小写的。

2 对于 **--location**，指定 kernel/initrd 在 HTTP 或 HTTPS 服务器上的位置。

3 对于 **coreos.inst.ignition_url=**，为机器角色指定 **worker.ign** Ignition 文件。仅支持 HTTP 和 HTTPS 协议。

4 对于 **coreos.live.rootfs_url=**，请为您引导的 **kernel** 和 **initramfs** 指定匹配的 rootfs 构件。仅支持 HTTP 和 HTTPS 协议。

5 可选：对于 **hostname**，指定客户端机器的完全限定主机名。

注意

如果您使用 HAProxy 作为负载均衡器，在 **/etc/haproxy/haproxy.cfg** 配置文件中为 **ingress-router-443** 和 **ingress-router-80** 更新 HAProxy 规则。

8. 继续为集群创建更多计算机器。

4.7.3. 批准机器的证书签名请求

当您将机器添加到集群时，会为您添加的每台机器生成两个待处理证书签名请求(CSR)。您必须确认这些 CSR 已获得批准，或根据需要自行批准。必须首先批准客户端请求，然后批准服务器请求。

先决条件

- 您已将机器添加到集群中。

流程

1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输出示例

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  63m  v1.28.5
master-1  Ready     master  63m  v1.28.5
master-2  Ready     master  64m  v1.28.5
```

输出中列出了您创建的所有机器。

注意

在有些 CSR 被批准前，前面的输出可能不包括计算节点（也称为 worker 节点）。

2. 检查待处理的 CSR，并确保添加到集群中的每台机器都有 **Pending** 或 **Approved** 状态的客户端请求：

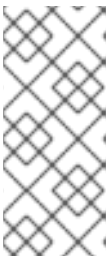
```
$ oc get csr
```

输出示例

```
NAME        AGE    REQUESTOR                                                    CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
...
```

在本例中，两台机器加入集群。您可能会在列表中看到更多已批准的 CSR。

3. 如果 CSR 没有获得批准，在您添加的机器的所有待处理 CSR 都处于 **Pending 状态** 后，请批准集群机器的 CSR：



注意

由于 CSR 会自动轮转，因此请在将机器添加到集群后一小时内批准您的 CSR。如果没有在一小时内批准它们，证书将会轮转，每个节点会存在多个证书。您必须批准所有这些证书。批准客户端 CSR 后，Kubelet 为服务证书创建一个二级 CSR，这需要手动批准。然后，如果 Kubelet 请求具有相同参数的新证书，则后续提供证书续订请求由 **machine-approver** 自动批准。

注意

对于在未启用机器 API 的平台上运行的集群，如裸机和其他用户置备的基础架构，您必须实施一种方法来自动批准 kubelet 提供证书请求(CSR)。如果没有批准请求，则 **oc exec**、oc**rsh** 和 **oc logs** 命令将无法成功，因为 API 服务器连接到 kubelet 时需要服务证书。与 Kubelet 端点联系的任何操作都需要此证书批准。该方法必须监视新的 CSR，确认 CSR 由 system: **node** 或 **system:admin** 组中的 **node-bootstrapper** 服务帐户提交，并确认节点的身份。

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

注意

在有些 CSR 被批准前，一些 Operator 可能无法使用。

4. 现在，您的客户端请求已被批准，您必须查看添加到集群中的每台机器的服务器请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE     REQUESTOR                                                   CONDITION
csr-bfd72   5m26s   system:node:ip-10-0-50-126.us-east-2.compute.internal       Pending
csr-c57lv   5m26s   system:node:ip-10-0-95-157.us-east-2.compute.internal       Pending
...
```

5. 如果剩余的 CSR 没有被批准，且处于 **Pending** 状态，请批准集群机器的 CSR：

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

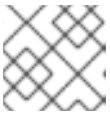
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 批准所有客户端和服务器 CSR 后，机器将 处于 **Ready** 状态。运行以下命令验证：

```
$ oc get nodes
```

输**出示例**

```
NAME      STATUS   ROLES   AGE  VERSION
master-0  Ready    master  73m  v1.28.5
master-1  Ready    master  73m  v1.28.5
master-2  Ready    master  74m  v1.28.5
worker-0  Ready    worker  11m  v1.28.5
worker-1  Ready    worker  11m  v1.28.5
```



注意

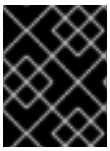
批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready** 状态。

其他信息

- 如需有关 CSR 的更多信息，请参阅 证书签名请求。

4.8. 在 IBM POWER 上使用多架构计算机器创建集群

要在 IBM Power® (**ppc64le**) 上使用多架构计算机器创建集群，您必须有一个现有的单架构 (**x86_64**) 集群。然后，您可以将 **ppc64le** 计算机器添加到 OpenShift Container Platform 集群中。



重要

在为集群添加 **ppc64le** 节点前，您必须将集群升级到使用多架构有效负载的节点。有关迁移到多架构有效负载的更多信息，请参阅迁移到具有多架构计算机器的集群。

以下流程解释了如何使用 ISO 镜像或网络 PXE 引导创建 RHCOS 计算机器。这将允许您在集群中添加 **ppc64le** 节点，并使用多架构计算机器部署集群。

注意

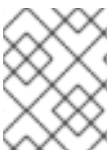
要使用 **x86_64** 上的多架构计算机器创建 IBM Power® (**ppc64le**) 集群，请按照在 IBM Power® 上安装集群的说明进行操作。然后，您可以添加 **x86_64** 计算机器 ， 如在裸机、IBM Power 或 IBM Z 上创建带有多架构计算机器的集群中所述。

4.8.1. 验证集群兼容性

在开始在集群中添加不同架构的计算节点前，您必须验证集群是否兼容多架构。

先决条件

- 已安装 OpenShift CLI (**oc**)



注意

在使用多个架构时，OpenShift Container Platform 节点的主机必须共享相同的存储层。如果它们没有相同的存储层，请使用 **nfs-provisioner** 等存储提供程序。

注意

您应该尽可能限制 compute 和 control plane 之间的网络跃点数量。

流程

- 您可以运行以下命令来检查集群是否使用构架有效负载：

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

验证

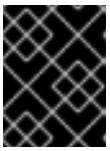
1. 如果您看到以下输出，集群将使用多架构有效负载：

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

然后，您可以开始在集群中添加多架构计算节点。

2. 如果您看到以下输出，集群不使用多架构有效负载：

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

要迁移集群以便集群支持多架构计算机器，请按照使用多架构计算机器迁移到集群的步骤进行操作。

4.8.2. 使用 ISO 镜像创建 RHCOS 机器

您可以使用 ISO 镜像为集群创建更多 Red Hat Enterprise Linux CoreOS (RHCOS) 计算机器，以创建机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 已安装 OpenShift CLI (**oc**)。

流程

1. 运行以下命令，从集群中删除 Ignition 配置文件：

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

2. 将您从集群导出的 **worker.ign** Ignition 配置文件上传到 HTTP 服务器。注意这些文件的 URL。
3. 您可以验证 ignition 文件是否在 URL 上可用。以下示例获取计算节点的 Ignition 配置文件：

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. 您可以运行以下命令来访问 ISO 镜像以引导新机器：

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 使用 ISO 文件在更多计算机器上安装 RHCOS。在安装集群前，使用创建机器时使用的相同方法：

   - 将 ISO 镜像刻录到磁盘并直接启动。
   - 在 LOM 接口中使用 ISO 重定向。

6. 在不指定任何选项或中断实时引导序列的情况下引导 RHCOS ISO 镜像。等待安装程序在 RHCOS live 环境中引导进入 shell 提示符。

注意

您可以中断 RHCOS 安装引导过程来添加内核参数。但是，在这个 ISO 过程中，您应该使用以下步骤中所述的 **coreos-installer** 命令，而不是添加内核参数。

7. 运行 **coreos-installer** 命令并指定满足您的安装要求的选项。您至少必须指定指向节点类型的 Ignition 配置文件的 URL，以及您要安装到的设备：

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

1. 您必须使用 **sudo** 运行 **coreos-installer** 命令，因为 **core** 用户没有执行安装所需的 root 权限。
2. 当 Ignition 配置文件通过 HTTP URL 获取时，需要 **--ignition-hash** 选项来验证集群节点上 Ignition 配置文件的真实性。**<digest>** 是上一步中获取的 Ignition 配置文件 SHA512 摘要。

注意

如果要通过使用 TLS 的 HTTPS 服务器提供 Ignition 配置文件，您可以在运行 **coreos-installer** 前将内部证书颁发机构(CA)添加到系统信任存储中。

以下示例将引导节点安装初始化到 **/dev/sda** 设备。bootstrap 节点的 Ignition 配置文件从 IP 地址 192.168.1.2 的 HTTP Web 服务器获取：

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. 在机器的控制台上监控 RHCOS 安装的进度。

重要

在开始安装 OpenShift Container Platform 之前，确保每个节点中安装成功。观察安装过程可以帮助确定可能会出现 RHCOS 安装问题的原因。

9. 继续为集群创建更多计算机器。

4.8.3. 通过 PXE 或 iPXE 启动来创建 RHCOS 机器

您可以使用 PXE 或 iPXE 引导为裸机集群创建更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 获取您在集群安装过程中上传到 HTTP 服务器的 RHCOS ISO 镜像、压缩的裸机 BIOS、**kernel** 和 **initramfs** 文件的 URL。
- 您可以访问在安装过程中为 OpenShift Container Platform 集群创建机器时使用的 PXE 引导基础架构。机器必须在安装 RHCOS 后从本地磁盘启动。
- 如果使用 UEFI，您可以访问在 OpenShift Container Platform 安装过程中修改的 **grub.conf** 文件。

流程

1. 确认 RHCOS 镜像的 PXE 或 iPXE 安装正确。
   - 对于 PXE：

```
DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> (1)
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img (2)
```

(1) 指定上传到 HTTP 服务器的 live **kernel** 文件位置。

(2) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**initrd** 参数值是 live **initramfs** 文件的位置，**coreos.inst.ignition_url** 参数值是 worker Ignition 配置文件的位置，**coreos.live.rootfs_url** 参数值是 live **rootfs** 文件的位置。**coreos.inst.ignition_url** 和 **coreos.live.rootfs_url** 参数仅支持 HTTP 和 HTTPS。

注意

此配置不会在图形控制台的机器上启用串行控制台访问。要配置不同的控制台，请在 **APPEND** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅 如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？

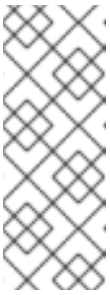
- 对于 iPXE (**x86_64** + **ppc64le**)：

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img (3)
boot
```

(1) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 **kernel** 文件的位置，在 UEFI 系统中引导时需要 **initrd=main** 参数，**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位置，**coreos.inst.ignition_url** 参数值则是 worker Ignition 配置文件的位置。

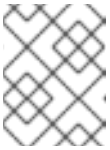
(2) 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。

(3) 指定上传到 HTTP 服务器的 **initramfs** 文件的位置。



注意

此配置不会在使用图形控制台配置不同的控制台的机器上启用串行控制台访问，请在 **kernel** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？ 和"启用 PXE 和 ISO 安装的串行控制台"部分。



注意

要在 **ppc64le** 架构中网络引导 CoreOS **kernel**，您需要使用启用了 **IMAGE_GZIP** 选项的 iPXE 构建版本。请参阅 iPXE 中的**IMAGE_GZIP** 选项。

- 对于 **ppc64le** 上的 PXE （使用 UEFI 和 GRUB 作为第二阶段）：

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
    initrd rhcos-<version>-live-initramfs.<architecture>.img (3)
}
```

1. 指定上传到 HTTP/TFTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 TFTP 服务器中的 **kernel** 文件的位置。**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位
2. 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。
3. 指定上传到 TFTP 服务器的 **initramfs** 文件的位置。

2. 使用 PXE 或 iPXE 基础架构为集群创建所需的计算机器。

4.8.4. 批准机器的证书签名请求

当您将机器添加到集群时，会为您添加的每台机器生成两个待处理证书签名请求(CSR)。您必须确认这些 CSR 已获得批准，或根据需要自行批准。必须首先批准客户端请求，然后批准服务器请求。

先决条件

- 您已将机器添加到集群中。

流程

1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输**出示例**

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.28.5
master-1 Ready    master  63m  v1.28.5
master-2 Ready    master  64m  v1.28.5
```

输出中列出了您创建的所有机器。

注意

在有些 CSR 被批准前，前面的输出可能不包括计算节点（也称为 worker 节点）。

2. 检查待处理的 CSR，并确保添加到集群中的每台机器都有 **Pending** 或 **Approved** 状态的客户端请求：

```
$ oc get csr
```

输**出示例**

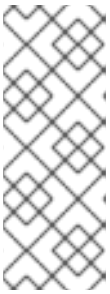
```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
...
```

在本例中，两台机器加入集群。您可能会在列表中看到更多已批准的 CSR。

3. 如果 CSR 没有获得批准，在您添加的机器的所有待处理 CSR 都处于 **Pending 状态** 后，请批准集群机器的 CSR：

注意

由于 CSR 会自动轮转，因此请在将机器添加到集群后一小时内批准您的 CSR。如果没有在一小时内批准它们，证书将会轮转，每个节点会存在多个证书。您必须批准所有这些证书。批准客户端 CSR 后，Kubelet 为服务证书创建一个二级 CSR，这需要手动批准。然后，如果 Kubelet 请求具有相同参数的新证书，则后续提供证书续订请求由 **machine-approver** 自动批准。



注意

对于在未启用机器 API 的平台上运行的集群，如裸机和其他用户置备的基础架构，您必须实施一种方法来自动批准 kubelet 提供证书请求(CSR)。如果没有批准请求，则 **oc exec**、ocrsh 和 **oc logs** 命令将无法成功，因为 API 服务器连接到 kubelet 时需要服务证书。与 Kubelet 端点联系的任何操作都需要此证书批准。该方法必须监视新的 CSR，确认 CSR 由 system: **node** 或 **system:admin** 组中的 **node-bootstrapper** 服务帐户提交，并确认节点的身份。

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

注意

在有些 CSR 被批准前，一些 Operator 可能无法使用。

4. 现在，您的客户端请求已被批准，您必须查看添加到集群中的每台机器的服务器请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-bfd72   5m26s  system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv   5m26s  system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. 如果剩余的 CSR 没有被批准，且处于 **Pending** 状态，请批准集群机器的 CSR：

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 批准所有客户端和服务器 CSR 后，机器将 处于 **Ready 状态**。运行以下命令验证：

```
$ oc get nodes -o wide
```

输**出示例**

```
NAME              STATUS   ROLES                  AGE   VERSION         INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                                        KERNEL-VERSION
CONTAINER-RUNTIME
worker-0-ppc64le  Ready    worker                 42d   v1.28.2+e3ba6d9   192.168.200.21
<none>        Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.ppc64le   cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
worker-1-ppc64le  Ready    worker                 42d   v1.28.2+e3ba6d9   192.168.200.20
<none>        Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.ppc64le   cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
master-0-x86      Ready    control-plane,master   75d   v1.28.2+e3ba6d9   10.248.0.38
10.248.0.38   Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
master-1-x86      Ready    control-plane,master   75d   v1.28.2+e3ba6d9   10.248.0.39
10.248.0.39   Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
master-2-x86      Ready    control-plane,master   75d   v1.28.2+e3ba6d9   10.248.0.40
10.248.0.40   Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
worker-0-x86      Ready    worker                 75d   v1.28.2+e3ba6d9   10.248.0.43
10.248.0.43   Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
worker-1-x86      Ready    worker                 75d   v1.28.2+e3ba6d9   10.248.0.44
10.248.0.44   Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.28.1-3.rhaos4.15.gitb36169e.el9
```

注意

批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready 状态**。

其他信息

- 如需有关 CSR 的更多信息，请参阅 证书签名请求。

4.9. 使用多架构计算机器管理集群

4.9.1. 使用多架构计算机器在集群中调度工作负载

使用不同架构的计算节点在集群中部署工作负载需要注意和监控集群。您可能需要进行进一步的操作，才能成功将 pod 放置到集群的节点中。

如需有关节点关联性、调度、污点和容限和容忍的更多信息，请参阅以下文档：

- 使用节点污点控制 pod 放置。
- 使用节点关联性控制节点上的 pod 放置
- 使用调度程序控制 pod 放置

4.9.1.1. 多架构节点工作负载部署示例

在使用不同架构的计算节点将工作负载调度到具有不同架构的计算节点上之前，请考虑以下用例：

使用节点关联性在节点上调度工作负载

您可以只允许将工作负载调度到其镜像支持的一组节点上，您可以在 pod 模板规格中设置 **spec.affinity.nodeAffinity** 字段。

将 nodeAffinity 设置为特定架构的部署示例

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values: 1
                - amd64
                - arm64
```

1 指定支持的构架。有效值包括 **amd64**,**arm64**, 或这两个值。

为特定架构污点每个节点

您可以污点节点，以避免与其架构不兼容的工作负载调度到该节点上。如果集群正在使用 **MachineSet** 对象，您可以在 **.spec.template.spec.taints** 字段中添加参数，以避免在带有不支持的架构的节点上调度工作负载。

- 在污点节点前，您必须缩减 **MachineSet** 对象或删除可用的机器。您可以使用以下命令之一缩减机器集：

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

或者：

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

有关扩展机器集的更多信息，请参阅"修改计算机器集"。

带有污点集的 MachineSet 示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      taints:
      - effect: NoSchedule
        key: multi-arch.openshift.io/arch
        value: arm64
```

您还可以运行以下命令来在特定节点上设置污点：

```
$ oc adm taint nodes <node-name> multi-arch.openshift.io/arch=arm64:NoSchedule
```

创建默认容限

您可以运行以下命令来注解命名空间，以便所有工作负载都获得相同的默认容限：

```
$ oc annotate namespace my-namespace \
  'scheduler.alpha.kubernetes.io/defaultTolerations'='[{"operator": "Exists", "effect": "NoSchedule",
"key": "multi-arch.openshift.io/arch"}]'
```

在工作负载中容忍架构污点

在具有定义污点的节点上，工作负载不会调度到该节点上。但是，您可以通过在 pod 的规格中设置容限来允许调度它们。

具有容限的部署示例

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      tolerations:
      - key: "multi-arch.openshift.io/arch"
        value: "arm64"
        operator: "Equal"
        effect: "NoSchedule"
```

本例部署也可以在指定了 **multi-arch.openshift.io/arch=arm64** 污点的节点上允许。

使用带有污点和容限的节点关联性

当调度程序计算一组要调度 pod 的节点时，容限可能会扩大集合，而节点关联性会限制集合。如果将污点设置为特定架构的节点，则需要以下示例容限才能调度 pod。

使用节点关联性和容限集的部署示例。

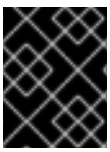
```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      tolerations:
      - key: "multi-arch.openshift.io/arch"
        value: "arm64"
        operator: "Equal"
        effect: "NoSchedule"
```

其他资源

- 修改计算机器集

4.9.2. 在 Red Hat Enterprise Linux CoreOS (RHCOS) 内核中启用 64k 页

您可以在集群中的 64 位 ARM 计算机器上启用 Red Hat Enterprise Linux CoreOS (RHCOS) 内核中的 64k 内存页。64k 页大小内核规格可用于大型 GPU 或高内存工作负载。这使用 Machine Config Operator (MCO) 完成，它使用机器配置池来更新内核。要启用 64k 页面大小，您必须为 ARM64 指定一个机器配置池，以便在内核中启用。



重要

使用 64k 页专用于 64 位 ARM 架构计算节点或在 64 位 ARM 机器上安装的集群。如果您使用 64 位 x86 机器在机器配置池中配置 64k 页内核，机器配置池和 MCO 将降级。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您在其中一个支持的平台中使用不同架构的计算节点创建集群。

流程

1. 标记您要运行 64k 页大小内核的节点：

```
$ oc label node <node_name> <label>
```

示例命令

```
$ oc label node worker-arm64-01 node-role.kubernetes.io/worker-64k-pages=
```

2. 创建包含使用 ARM64 架构和 **worker-64k-pages** 角色的 worker 角色的机器配置池：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-64k-pages
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
        - worker
        - worker-64k-pages
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-64k-pages: ""
      kubernetes.io/arch: arm64
```

3. 在计算节点上创建机器配置，以使用 **64k-pages** 参数启用 **64k-pages**。

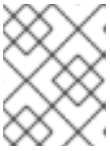
```
$ oc create -f <filename>.yaml
```

MachineConfig 示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker-64k-pages" (1)
  name: 99-worker-64kpages
spec:
  kernelType: 64k-pages (2)
```

(1) 在自定义机器配置池中指定 **machineconfiguration.openshift.io/role** 标签的值。MachineConfig 示例使用 **worker-64k-pages** 标签在 **worker-64k-pages** 池中启用 64k 页面。

(2) 指定所需的内核类型。有效值为 **64k-pages** 和 **default**



注意

64k-pages 类型仅支持基于 64 位 ARM 架构。 **realtime** 类型仅支持基于 64 位 x86 架构。

验证

- 要查看您的新 **worker-64k-pages** 机器配置池，请运行以下命令：

```
$ oc get mcp
```

输出示例

```
NAME     CONFIG                                              UPDATED   UPDATING
DEGRADED   MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-9d55ac9a91127c36314e1efe7d77fbf8                True      False
False      3            3                   3                   0                   361d
worker   rendered-worker-e7b61751c4a5b7ff995d64b967c421ff                True      False
False      7            7                   7                   0                   361d
worker-64k-pages  rendered-worker-64k-pages-e7b61751c4a5b7ff995d64b967c421ff   True
False      False      2            2                   2                   0                   35m
```

4.9.3. 在多架构计算机器上的镜像流中导入清单列表

在带有多架构计算机器的 OpenShift Container Platform 4.15 集群中，集群中的镜像流不会自动导入清单列表。您必须手动将默认的 **importMode** 选项改为 **PreserveOriginal** 选项，才能导入清单列表。

先决条件

- 已安装 OpenShift Container Platform CLI (**oc**)。

流程

- 以下示例命令演示了如何对 **ImageStream** cli-artifacts 进行补丁，以便 **cli-artifacts:latest** 镜像流标签作为清单列表导入。

```
$ oc patch is/cli-artifacts -n openshift -p '{"spec":{"tags":[{"name":"latest","importPolicy":
{"importMode":"PreserveOriginal"}}]}}'
```

验证

- 您可以通过检查镜像流标签来检查是否正确导入的清单列表。以下命令将列出特定标签的各个架构清单。

```
$ oc get istag cli-artifacts:latest -n openshift -oyaml
```

如果存在 **dockerImageManifests** 对象，则清单列表导入成功。

dockerImageManifests 对象的输出示例

```
dockerImageManifests:
  - architecture: amd64
```

```
    digest:
sha256:16d4c96c52923a9968fbfa69425ec703aff711f1db822e4e9788bf5d2bee5d77
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: arm64
    digest:
sha256:6ec8ad0d897bcdf727531f7d0b716931728999492709d19d8b09f0d90d57f626
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: ppc64le
    digest:
sha256:65949e3a80349cdc42acd8c5b34cde6ebc3241eae8daaeea458498fedb359a6a
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: s390x
    digest:
sha256:75f4fa21224b5d5d511bea8f92dfa8e1c00231e5c81ab95e83c3013d245d1719
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
```

第 5 章 在 VSPHERE 集群上启用加密

您可以通过一次排空和关闭节点，在 vSphere 上安装 OpenShift Container Platform 4.15 后对虚拟机进行加密。在每个虚拟机关闭时，您可以在 vCenter web 界面中启用加密。

5.1. 加密虚拟机

您可以使用以下过程对虚拟机进行加密。您可以排空虚拟机，关闭并使用 vCenter 接口加密它们。最后，您可以创建一个存储类以使用加密存储。

先决条件

- 您已在 vSphere 中配置了标准密钥供应商。如需更多信息，请参阅在 vCenter 服务器中添加 KMS。

重要

不支持 vCenter 中的原生密钥供应商。如需更多信息，请参阅 vSphere 原生密钥提供程序概述。

- 您已在托管集群的所有 ESXi 主机上启用了主机加密模式。如需更多信息，请参阅启用主机加密模式。
- 您有一个启用了所有加密权限的 vSphere 帐户。如需更多信息，请参阅 Cryptographic Operations Privileges。

流程

1. 排空和封锁其中一个节点。有关节点管理的详细信息，请参阅"使用节点"。
2. 在 vCenter 界面中关闭与该节点关联的虚拟机。
3. 在 vCenter 界面中右键单击虚拟机，然后选择 **VM Policies → Edit VM Storage Policies**。
4. 选择加密的存储策略并选择 **OK**。
5. 在 vCenter 界面中启动加密的虚拟机。
6. 对您要加密的所有节点重复步骤 1-5。
7. 配置使用加密存储策略的存储类。有关配置加密存储类的更多信息，请参阅"VMware vSphere CSI Driver Operator"。

5.2. 其他资源

- 操作节点
- vSphere 加密
- 加密虚拟机的要求

第 6 章 安装后配置 VSPHERE 连接设置

在启用了平台集成功能的 vSphere 上安装 OpenShift Container Platform 集群后，您可能需要根据安装方法手动更新 vSphere 连接设置。

对于使用 Assisted Installer 的安装，您必须更新连接设置。这是因为 Assisted Installer 在安装过程中将默认连接设置作为占位符添加到 **vSphere 连接配置**向导中。

对于安装程序置备的基础架构安装，您应该在安装过程中输入有效的连接设置。您可以随时使用 **vSphere 连接配置**向导来验证或修改连接设置，但这不是完成安装的强制设置。

6.1. 配置 VSPHERE 连接设置

根据需要修改以下 vSphere 配置设置：

- vCenter 地址
- vCenter 集群
- vCenter 用户名
- vCenter 密码
- vCenter 地址
- vSphere 数据中心
- vSphere 数据存储
- 虚拟机文件夹

先决条件

- Assisted Installer 成功完成安装集群。
- 集群连接到 **https://console.redhat.com**。

流程

1. 在 Administrator 视角中，进入到 **Home → Overview**。
2. 在 **Status** 下，点 **vSphere connection** 打开 **vSphere 连接配置**向导。
3. 在 **vCenter** 字段中，输入 vSphere vCenter 服务器的网络地址。这可以是域名，也可以是 IP 地址。它会出现在 vSphere Web 客户端 URL 中，例如 **https://[your_vCenter_address]/ui**。
4. 在 **vCenter cluster** 字段中，输入安装 OpenShift Container Platform 的 vSphere vCenter 集群名称。

   重要

   如果安装了 OpenShift Container Platform 4.13 或更高版本，则此步骤是必需的。

5. 在 **Username** 字段中，输入 vSphere vCenter 用户名。

6. 在 **Password** 字段中输入您的 vSphere vCenter 密码。

警告

系统将用户名和密码存储在集群的 **kube-system** 命名空间中的 **vsphere-creds** secret 中。不正确的 vCenter 用户名或密码使集群节点不可调度。

7. 在 **Datacenter** 字段中，输入 vSphere 数据中心的名称，其中包含用于托管集群的虚拟机；例如，**SDDC-Datacenter**。

8. 在 **Default data store** 字段中，输入存储持久数据卷的 vSphere 数据存储的路径和名称；例如：**/SDDC-Datacenter/datastore/datastorename**。

警告

在保存配置后，更新 vSphere 数据中心或默认数据存储会分离任何活跃的 vSphere **PersistentVolume**。

9. 在 **Virtual Machine Folder** 字段中，输入包含集群虚拟机的数据中心文件夹；例如，**/SDDC-Datacenter/vm/ci-ln-hjg4vg2-c61657-t2gzr**。要使 OpenShift Container Platform 安装成功，组成集群的所有虚拟机都必须位于单个数据中心文件夹中。

10. 点 **Save Configuration**。这会更新 **openshift-config** 命名空间中的 **cloud-provider-config** ConfigMap 资源，并启动配置过程。

11. 重新打开 **vSphere 连接配置**向导，再展开 **Monitored operators** 面板。检查 Operator 的状态是否为 **Progressing** 或 **Healthy**。

6.2. 验证配置

连接配置过程更新 Operator 状态和 control plane 节点。完成大约需要一小时才能完成。在配置过程中，节点将重新引导。以前绑定的 **PersistentVolumeClaims** 对象可能会断开连接。

先决条件

- 您已在 **vSphere 连接配置**向导中保存了配置设置。

流程

1. 检查配置过程是否已成功完成：

   a. 在 OpenShift Container Platform Administrator 视角中，进入到 **Home → Overview**。

   b. 在 **Status** 下点 **Operators**。等待所有操作器状态从 **Progressing** 变为 **All succeeded**。**Failed** 状态表示配置失败。

c. 在 **Status** 下，点 **Control Plane**。等待所有 Control Pane 组件的响应率返回到 100%。**失败**的 control plane 组件表示配置失败。

失败表示至少一个连接设置不正确。更改 **vSphere 连接配置**向导中的设置，然后再次保存配置。

2. 通过执行以下步骤来检查您是否可以绑定 **PersistentVolumeClaims** 对象：

a. 使用以下 YAML 创建 **StorageClass** 对象：

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: vsphere-sc
provisioner: kubernetes.io/vsphere-volume
parameters:
 datastore: YOURVCENTERDATASTORE
 diskformat: thin
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

b. 使用以下 YAML 创建 **PersistentVolumeClaims** 对象：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: test-pvc
 namespace: openshift-config
 annotations:
   volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/vsphere-volume
 finalizers:
   - kubernetes.io/pvc-protection
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
    storage: 10Gi
 storageClassName: vsphere-sc
 volumeMode: Filesystem
```

如果您无法创建 **PersistentVolumeClaims** 对象，您可以在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中进入 **Storage → PersistentVolumeClaims** 来进行故障排除。

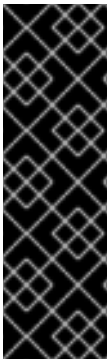
有关创建存储对象的说明，请参阅动态置备。

第 7 章 安装后机器配置任务

有时您需要更改 OpenShift Container Platform 节点上运行的操作系统。这包括更改网络时间服务的设置、添加内核参数或者以特定的方式配置日志。

除了一些特殊功能外，通过创建称为 Machine Config Operator 管理的 **MachineConfig** 对象，可以对 OpenShift Container Platform 节点上的操作系统进行大多数更改。

本节中的任务介绍了如何使用 Machine Config Operator 的功能在 OpenShift Container Platform 节点上配置操作系统功能。



重要

NetworkManager 以 key file 的格式将新网络配置保存到 **/etc/NetworkManager/system-connections/**

在以前的版本中，NetworkManager 将新的网络配置以 ifcfg 格式保存到 **/etc/sysconfig/network-scripts/**。从 RHEL 9.0 开始，RHEL 将新网络配置存储在 **/etc/NetworkManager/system-connections/** 中，采用 key 文件格式。以旧格式存储在 **/etc/sysconfig/network-scripts/** 中的连接配置仍可以正常工作。对现有配置集的修改会继续更新旧的文件。

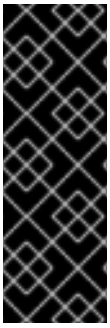
7.1. 了解 MACHINE CONFIG OPERATOR

7.1.1. Machine Config Operator

用途
Machine Congig Operator 管理并应用基本操作系统和容器运行时的配置和更新，包括内核和 kubelet 之间的所有配置和更新。

有四个组件：

- **machine-config-server**：为加入集群的新机器提供 Ignition 配置。
- **machine-config-controller**：协调机器升级到 **MachineConfig** 对象定义的配置。提供用来控制单独一组机器升级的选项。
- **machine-config-daemon**：在更新过程中应用新机器配置。验证并验证机器的状态到请求的机器配置。
- **machine-config**：提供安装、首次启动和更新一个机器的完整机器配置源。



重要

目前，不支持阻止或限制机器配置服务器端点。机器配置服务器必须公开给网络，以便新置备的机器没有现有配置或状态，才能获取其配置。在这个模型中，信任的根是证书签名请求 (CSR) 端点，即 kubelet 发送其证书签名请求以批准加入集群。因此，机器配置不应用于分发敏感信息，如 secret 和证书。

为确保机器配置服务器端点，端口 22623 和 22624 在裸机场景中是安全的，客户必须配置正确的网络策略。

其他资源

- 关于 OpenShift SDN 网络插件。

项目
openshift-machine-config-operator

7.1.2. 机器配置概述

Machine Config Operator（MCO）管理对 systemd、CRI-O 和 Kubelet、内核、Network Manager 和其他系统功能的更新。它还提供了一个 **MachineConfig** CRD，它可以在主机上写入配置文件（请参阅 machine-config-operator）。了解 MCO 的作用以及如何与其他组件交互对于对 OpenShift Container Platform 集群进行高级系统级更改至关重要。以下是您应该了解的 MCO、机器配置以及它们的使用方式：

- 机器配置按字母顺序处理，按字母顺序增加名称。呈现控制器使用列表中的第一个机器配置作为基础，并将剩余的附加到基本机器配置中。
- 机器配置可以对每个系统的操作系统上的文件或服务进行特定的更改，代表一个 OpenShift Container Platform 节点池。
- MCO 应用对机器池中的操作系统的更改。所有 OpenShift Container Platform 集群都以 worker 和 control plane 节点池开头。通过添加更多角色标签，您可以配置自定义节点池。例如，您可以设置一个自定义的 worker 节点池，其中包含应用程序所需的特定硬件功能。但是，本节中的示例着重介绍了对默认池类型的更改。

重要

一个节点可以应用多个标签来指示其类型，如 **master** 或 **worker**，但只能是一个单一机器配置池的成员。

- 机器配置更改后，MCO 根据 **topology.kubernetes.io/zone** 标签，按区字母更新受影响的节点。如果一个区域有多个节点，则首先更新最旧的节点。对于不使用区的节点，如裸机部署中的节点，节点会按使用的时间升级，首先更新最旧的节点。MCO 一次更新机器配置池中由 **maxUnavailable** 字段指定的节点数量。
- 在将 OpenShift Container Platform 安装到磁盘前，必须先进行一些机器配置。在大多数情况下，这可以通过创建直接注入 OpenShift Container Platform 安装程序进程中的机器配置来实现，而不必作为安装后机器配置运行。在其他情况下，您可能需要在 OpenShift Container Platform 安装程序启动时传递内核参数时进行裸机安装，以完成诸如设置每个节点的 IP 地址或高级磁盘分区等操作。
- MCO 管理机器配置中设置的项目。MCO 不会覆盖您对系统进行的手动更改，除非明确告知 MCO 管理冲突文件。换句话说，MCO 只提供您请求的特定更新，它不会声明对整个节点的控制。
- 强烈建议手动更改节点。如果您需要退出某个节点并启动一个新节点，则那些直接更改将会丢失。
- MCO 只支持写入 **/etc** 和 **/var** 目录里的文件，虽然有些目录的符号链接可以通过符号链接到那些区域之一来写入。例如 **/opt** 和 **/usr/local** 目录。
- Ignition 是 MachineConfig 中使用的配置格式。详情请参阅 Ignition 配置规格 v3.2.0。
- 虽然 Ignition 配置设置可以在 OpenShift Container Platform 安装时直接交付，且以 MCO 提供 Ignition 配置的方式格式化，但 MCO 无法查看这些原始 Ignition 配置是什么。因此，您应该在部署 Ignition 配置前将 Ignition 配置设置嵌套到机器配置中。

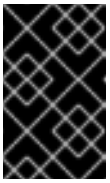
- 当由 MCO 管理的文件在 MCO 之外更改时，Machine Config Daemon（MCD）会将该节点设置为 **degraded**。然而，它不会覆盖这个错误的文件，它应该继续处于 **degraded（降级）**状态。
- 使用机器配置的一个关键原因是，当您为 OpenShift Container Platform 集群中的池添加新节点时，会应用它。**machine-api-operator** 置备一个新机器， MCO 配置它。

MCO 使用 Ignition 作为配置格式。OpenShift Container Platform 4.6 从 Ignition 配置规格版本 2 移到版本 3。

7.1.2.1. 机器配置可以更改什么？

MCO 可更改的组件类型包括：

- **config**：创建 Ignition 配置对象（请参阅 Ignition 配置规格），以完成修改 OpenShift Container Platform 机器上的文件、systemd 服务和其他功能，包括：
  - **Configuration files**：创建或覆盖 **/var** 或 **/etc** 目录中的文件。
  - **systemd units**：在附加设置中丢弃并设置 systemd 服务的状态，或者添加到现有 systemd 服务中。
  - **用户和组**：在安装后更改 passwd 部分中的 SSH 密钥。



重要

- 只有 **core** 用户才支持使用机器配置更改 SSH 密钥。
- 不支持使用机器配置添加新用户。

- **kernelArguments**：在 OpenShift Container Platform 节点引导时在内核命令行中添加参数。
- **kernelType**：（可选）使用非标准内核而不是标准内核。使用 **realtime** 来使用 RT 内核（用于 RAN）。这只在选择的平台上被支持。使用 **64k-pages** 参数启用 64k 页大小内核。此设置专用于具有 64 位 ARM 架构的机器。
- **fips**：启用 FIPS 模式。不应在安装时设置 FIPS，而不是安装后的步骤。

重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

- **extensions**：通过添加所选预打包软件来扩展 RHCOS 功能。对于这个功能，可用的扩展程序包括 usbguard 和内核模块。
- **Custom resources（用于 ContainerRuntime 和 Kubelet）**：在机器配置外，MCO 管理两个特殊自定义资源，用于修改 CRI-O 容器运行时设置（**ContainerRuntime** CR）和 Kubelet 服务（**Kubelet** CR）。

MCO 不是更改 OpenShift Container Platform 节点上的操作系统组件的唯一 Operator。其他 Operator 也可以修改操作系统级别的功能。一个例子是 Node Tuning Operator，它允许您通过 Tuned 守护进程配置集进行节点级别的性能优化。

安装后可以进行的 MCO 配置任务包括在以下步骤中。如需在 OpenShift Container Platform 安装过程中或之前完成的系统配置任务，请参阅 RHCOS 裸机安装的描述。

在某些情况下，节点上的配置与当前应用的机器配置指定不完全匹配。这个状态被称为 *配置偏移*。Machine Config Daemon(MCD)定期检查节点是否有配置偏移。如果 MCD 检测到配置偏移，MCO 会将节点标记为 **降级(degraded)**，直到管理员更正节点配置。降级的节点在线且可操作，但无法更新。有关配置偏移的更多信息，请参阅 *了解配置偏移检测*。

7.1.2.2. 项目

详情请参阅 openshift-machine-config-operator GitHub 站点。

7.1.3. 了解 Machine Config Operator 节点排空行为

当您使用机器配置更改系统功能时（如添加新配置文件、修改 systemd 单元或内核参数或更新 SSH 密钥），Machine Config Operator (MCO) 会应用这些更改，并确保每个节点处于所需的配置状态。

在进行更改后，MCO 会确保生成新的机器配置。在大多数情况下，当应用新的机器配置时，Operator 会在每个受影响的节点上执行以下步骤，直到所有受影响的节点都有更新的配置：

1. Cordon.对于额外的工作负载，MCO 会将节点标记为不可调度。
2. Drain.MCO 终止节点上运行的所有工作负载，导致工作负载重新调度到其他节点上。
3. Apply.MCO 根据需要将新配置写入节点。
4. 重新启动.MCO 重启节点。
5. Uncordon.对于工作负载，MCO 将节点标记为可调度。

在此过程中，MCO 根据机器配置池中设置的 **MaxUnavailable** 值维护所需的 pod 数量。

如果 MCO 在 master 节点上排空 pod，请注意以下条件：

- 在单节点 OpenShift 集群中，MCO 会跳过排空操作。
- MCO 不会排空静态 pod，以防止干扰服务（如 etcd）。

注意

在某些情况下，节点不会被排空。如需更多信息，请参阅 "About the Machine Config Operator"。

您可以通过禁用 control plane 重启来缓解排空和重启周期造成的中断。如需更多信息，请参阅"禁用 Machine Config Operator 自动重新引导"。

其他资源

- 关于 Machine Config Operator
- 禁用 Machine Config Operator 自动重新引导

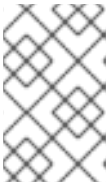
7.1.4. 了解配置偏移检测

当节点的磁盘上状态与机器配置中配置的内容不同时，可能会出现情况。这称为 *配置偏移(drift)*。例如，集群管理员可能会手动修改一个文件、systemd 单元文件，或者通过机器配置配置的文件权限。这会导致配置偏移。配置偏移可能会导致 Machine Config Pool 中的节点或机器配置更新时出现问题。

Machine Config Operator(MCO)使用 Machine Config Daemon(MCD)定期检查节点是否有配置偏移。如果检测到，MCO 会将节点和机器配置池(MCP)设置为 **Degraded**，并报告错误。降级的节点在线且可操作，但无法更新。

MCD 在出现任何以下条件时执行配置偏移检测：

- 当节点引导时。
- 在机器配置中指定的任何文件（Ignition 文件和 systemd 置入单元）后，会在机器配置外修改。
- 应用新机器配置前。



注意

如果您将新机器配置应用到节点，MCD 会临时关闭配置偏移检测。这个关闭是必需的，因为新机器配置必须与节点上的机器配置不同。应用新机器配置后，MCD 将使用新机器配置重启检测配置偏移。

在执行配置偏移检测时，MCD 会验证文件内容和权限是否与当前应用的机器配置指定完全匹配。通常，MCD 在触发检测后检测到小于第二个配置偏移。

如果 MCD 检测到配置偏移，MCD 执行以下任务：

- 向控制台日志发送错误
- 发送 Kubernetes 事件
- 在节点上停止进一步检测
- 将节点和 MCP 设置为 **degraded**

您可以通过列出 MCP 检查是否有降级的节点：

```
$ oc get mcp worker
```

如果您有一个降级的 MCP，**DEGRADEDMACHINECOUNT** 字段将不为零，类似于以下输出：

输出示例

```
NAME    CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
worker   rendered-worker-404caf3180818d8ac1f50c32f14b57c3   False     True       True       2
1              1                   1                   5h51m
```

您可以通过检查机器配置池来确定问题是否由配置偏移导致：

```
$ oc describe mcp worker
```

输**出示例**

```
...
  Last Transition Time:  2021-12-20T18:54:00Z
  Message:               Node ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4 is reporting: "content mismatch
for file \"/etc/mco-test-file\"" 1
  Reason:                1 nodes are reporting degraded status on sync
  Status:                True
  Type:                  NodeDegraded 2
...
```

1 此消息显示节点的 **/etc/mco-test-file** 文件（由机器配置添加）已在机器配置外有所变化。

2 节点的状态为 **NodeDegraded**。

或者，如果您知道哪个节点已降级，请检查该节点：

```
$ oc describe node/ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4
```

输**出示例**

```
...

Annotations:        cloud.network.openshift.io/egress-ipconfig: [{"interface":"nic0","ifaddr":
{"ipv4":"10.0.128.0/17"},"capacity":{"ip":10}}]
                    csi.volume.kubernetes.io/nodeid:
                      {"pd.csi.storage.gke.io":"projects/openshift-gce-devel-ci/zones/us-central1-
a/instances/ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4"}
                    machine.openshift.io/machine: openshift-machine-api/ci-ln-j4h8nkb-72292-pxqxz-worker-
a-fjks4
                    machineconfiguration.openshift.io/controlPlaneTopology: HighlyAvailable
                    machineconfiguration.openshift.io/currentConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                    machineconfiguration.openshift.io/desiredConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                    machineconfiguration.openshift.io/reason: content mismatch for file "/etc/mco-test-file"
1
                    machineconfiguration.openshift.io/state: Degraded 2
...
```

1 错误消息表示在节点和列出的机器配置间检测到配置偏移。此处的错误消息表示 **/etc/mco-test-file** 的内容由机器配置添加，在机器配置外有所变化。

2 节点的状态为 **Degraded**。

您可以通过执行以下补救之一来更正配置偏移并将节点返回到 **Ready** 状态：

- 确保节点上文件的内容和文件权限与机器配置中配置的内容匹配。您可以手动重写文件内容或更改文件权限。
- 在降级节点上生成一个强制文件。强制文件使 MCD 绕过常见的配置偏移检测并消除了当前的机器配置。

注意

在节点上生成强制文件会导致该节点重新引导。

7.1.5. 检查机器配置池状态

要查看 Machine Config Operator（MCO）、其子组件及其管理的资源的状态，请使用以下 **oc** 命令：

流程

1. 要查看集群中为每个机器配置池 (MCP) 中可用 MCO 管理的节点数量，请运行以下命令：

```
$ oc get machineconfigpool
```

输出示例

```
NAME     CONFIG                 UPDATED  UPDATING   DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master   rendered-master-06c9c4…  True     False      False     3          3              3
0                  4h42m
worker   rendered-worker-f4b64…   False    True       False     3          2              2
0                  4h42m
```

其中：

UPDATED

True 状态表示 MCO 已将当前机器配置应用到该 MCP 中的节点。当前机器配置在 **oc get mcp** 输出中的 **STATUS** 字段中指定。**False** 状态表示 MCP 中的节点正在更新。

UPDATING

True 状态表示 MCO 正在按照 **MachineConfigPool** 自定义资源中的规定应用到该 MCP 中的至少一个节点。所需的机器配置是新编辑的机器配置。要进行更新的节点可能不适用于调度。**False** 状态表示 MCP 中的所有节点都已更新。

DEGRADED

True 状态表示 MCO 被禁止将当前或所需的机器配置应用到该 MCP 中的至少一个节点，或者配置失败。降级的节点可能不适用于调度。**False** 状态表示 MCP 中的所有节点都就绪。

MACHINECOUNT

表示该 MCP 中的机器总数。

READYMACHINECOUNT

指明 MCP 中准备进行调度的机器总数。

UPDATEDMACHINECOUNT

指明 MCP 中有当前机器配置的机器总数。

DEGRADEDMACHINECOUNT

指明 MCP 中标记为 degraded 或 unreconcilable 的机器总数。

在前面的输出中，有三个 control plane (master) 节点和三个 worker 节点。control plane MCP 和关联的节点更新至当前机器配置。worker MCP 中的节点会更新为所需的机器配置。worker MCP 中的两个节点被更新，一个仍在更新，如 **UPDATEDMACHINECOUNT** 为 **2**。没有问题，如 **DEGRADEDMACHINECOUNT** 为 **0**，**DEGRADED** 为 **False**。

虽然 MCP 中的节点正在更新，但 **CONFIG** 下列出的机器配置是当前的机器配置，该配置会从这个配置进行更新。更新完成后，列出的机器配置是所需的机器配置，它被更新为 MCP。

注意

如果节点被封锁，则该节点不包含在 **READYMACHINECOUNT** 中，但包含在 **MACHINECOUNT** 中。另外，MCP 状态被设置为 **UPDATING**。因为节点具有当前的机器配置，所以它被计算在 **UPDATEDMACHINECOUNT** 总计：

输**出示例**

```
NAME     CONFIG                 UPDATED  UPDATING   DEGRADED
MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
master   rendered-master-06c9c4…  True     False      False    3          3
3                  0                  4h42m
worker   rendered-worker-c1b41a…  False    True       False    3          2
3                  0                  4h42m
```

2. 要通过检查 **MachineConfigPool** 自定义资源来检查 MCP 中的节点状态，请运行以下命令：

```
$ oc describe mcp worker
```

输**出示例**

```
...
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        2
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
Events:                       <none>
```

注意

如果节点被封锁，则节点不包含在 **Ready Machine Count** 中。它包含在 **Unavailable Machine Count** 中：

输**出示例**

```
...
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        2
  Ready Machine Count:        2
  Unavailable Machine Count:  1
  Updated Machine Count:      3
```

3. 要查看每个现有的 **MachineConfig** 对象，请运行以下命令：

```
$ oc get machineconfigs
```

输**出示例**

```
NAME                        GENERATEDBYCONTROLLER          IGNITIONVERSION  AGE
00-master                   2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
00-worker                   2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
01-master-container-runtime      2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
01-master-kubelet           2c9371fbb673b97a6fe8b1c52…     3.2.0            5h18m
...
rendered-master-dde...          2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
rendered-worker-fde...          2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
```

请注意，列为 **rendered** 的 **MachineConfig** 对象并不意味着要更改或删除。

4. 要查看特定机器配置的内容（本例中为 **01-master-kubelet**），请运行以下命令：

```
$ oc describe machineconfigs 01-master-kubelet
```

命令的输出显示此 **MachineConfig** 对象同时包含配置文件(**cloud.conf** 和 **kubelet.conf**) 和 systemd 服务(Kubernetes Kubelet)：

输**出示例**

```
Name:         01-master-kubelet
...
Spec:
  Config:
    Ignition:
      Version:  3.2.0
    Storage:
      Files:
        Contents:
          Source:   data:,
        Mode:       420
        Overwrite:  true
        Path:       /etc/kubernetes/cloud.conf
        Contents:
          Source:
data:,kind%3A%20KubeletConfiguration%0AapiVersion%3A%20kubelet.config.k8s.io%2Fv1bet
a1%0Aauthentication%3A%0A%20%20x509%3A%0A%20%20%20%20clientCAFile%3A%20
%2Fetc%2Fkubernetes%2Fkubelet-ca.crt%0A%20%20anonymous...
        Mode:       420
        Overwrite:  true
        Path:       /etc/kubernetes/kubelet.conf
    Systemd:
      Units:
        Contents:  [Unit]
Description=Kubernetes Kubelet
Wants=rpc-statd.service network-online.target crio.service
After=network-online.target crio.service

ExecStart=/usr/bin/hyperkube \
    kubelet \
      --config=/etc/kubernetes/kubelet.conf \ ...
```

如果应用的机器配置出现问题，您可以随时退出这一更改。例如，如果您运行 **oc create -f ./myconfig.yaml** 以应用机器配置，您可以运行以下命令来删除该机器配置：

```
$ oc delete -f ./myconfig.yaml
```

如果这是唯一的问题，则受影响池中的节点应返回非降级状态。这会导致呈现的配置回滚到其之前更改的状态。

如果在集群中添加自己的机器配置，您可以使用上例中显示的命令检查其状态以及应用到它们的池的相关状态。

7.1.6. 检查机器配置节点状态

在更新过程中，您可能希望监控单个节点的进度，以防出现问题，您需要对节点进行故障排除。

要查看集群的 Machine Config Operator (MCO) 更新的状态，请使用以下 **oc** 命令：

重要

改进了 MCO 状态报告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。

流程

1. 运行以下命令，获取所有机器配置池中所有节点的更新状态概述：

```
$ oc get machineconfignodes
```

输出示例

```
NAME                        UPDATED   UPDATEPREPARED   UPDATEEXECUTED   UPDATEPOSTACTIONCOMPLETED   UPDATECOMPLETED   RESUMED
ip-10-0-12-194.ec2.internal   True      False            False            False                       False             False
ip-10-0-17-102.ec2.internal   False     True             False            False                       False             False
ip-10-0-2-232.ec2.internal    False     False            True             False                       False             False
ip-10-0-59-251.ec2.internal   False     False            False            True                        False             False
ip-10-0-59-56.ec2.internal    False     False            False            False                       True              True
ip-10-0-6-214.ec2.internal    False     False            Unknown          False                       False             False
```

其中：

UPDATED

True 状态表示 MCO 已将当前机器配置应用到该特定节点。**False** 状态表示节点当前正在更新。**Unknown** 状态表示操作正在处理。

UPDATEPREPARED

False 状态表示 MCO 尚未开始协调要分发的新机器配置。**True** 状态表示 MCO 已完成更新这个阶段。**Unknown** 状态表示操作正在处理。

UPDATEEXECUTED

False 状态表示 MCO 尚未启动封锁和排空节点。它还表示磁盘状态和操作系统尚未启动更新。**True** 状态表示 MCO 已完成更新这个阶段。**Unknown** 状态表示操作正在处理。

UPDATEPOSTACTIONCOMPLETED

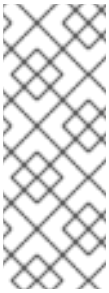
False 状态表示 MCO 尚未启动重新引导节点或关闭守护进程。**True** 状态表示 MCO 已完成重新引导并更新节点状态。**Unknown** 状态表示在这个阶段更新过程中发生了错误，或者 MCO 当前正在应用更新。

UPDATECOMPLETED

False 状态表示 MCO 尚未启动节点并更新节点状态和指标。**True** 状态表示 MCO 完成更新节点状态和可用指标。

RESUMED

False 状态表示 MCO 尚未启动配置偏移监控器。**True** 状态表示节点有恢复的操作。**Unknown** 状态表示操作正在处理。



注意

在前面描述的主要阶段中，您可以有二级阶段，可用于查看更新进度。您可以使用上一命令的 **-o wide** 选项获取包含更新的辅助阶段的更多信息。这提供了额外的 **UPDATECOMPATIBLE**,**UPDATEFILESANDOS**,**DRAINEDNODE**,**CORDONEDNODE**,**REBOOTNODE**,**RELOADEDCRIO** 和 **UNCORDONED** 列。这些辅助阶段并不总是发生，并依赖于您要应用的更新类型。

2. 运行以下命令，检查特定机器配置池中节点的更新状态：

```
$ oc get machineconfignodes $(oc get machineconfignodes -o json | jq -r
'.items[]|select(.spec.pool.name=="<pool_name>")|.metadata.name') 1
```

1 池的名称是 **MachineConfigPool** 对象名称。

输**出示例**

```
NAME                        UPDATED   UPDATEPREPARED   UPDATEEXECUTED
UPDATEPOSTACTIONCOMPLETE   UPDATECOMPLETE   RESUMED
ip-10-0-48-226.ec2.internal   True      False          False          False                False
False
ip-10-0-5-241.ec2.internal    True      False          False          False                False
False
ip-10-0-74-108.ec2.internal   True      False          False          False                False
False
```

3. 运行以下命令，检查单个节点的更新状态：

```
$ oc describe machineconfignode/<node_name> 1
```

1 节点的名称是 **MachineConfigNode** 对象名称。

输**出示例**

```
Name:         <node_name>
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  machineconfiguration.openshift.io/v1alpha1
Kind:         MachineConfigNode
Metadata:
  Creation Timestamp:  2023-10-17T13:08:58Z
  Generation:          1
  Resource Version:    49443
  UID:                 4bd758ab-2187-413c-ac42-882e61761b1d
Spec:
  Node Ref:
    Name:        <node_name>
  Pool:
    Name:        master
  ConfigVersion:
    Desired: rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b (1)
Status:
  Conditions:
    Last Transition Time:  2023-10-17T13:09:02Z
    Message:               Node has completed update to config rendered-master-
cf99e619747ab19165f11e3546c71f1e
    Reason:                NodeUpgradeComplete
    Status:                True
    Type:                  Updated
    Last Transition Time:  2023-10-17T13:09:02Z
    Message:               This node has not yet entered the UpdatePreparing phase
    Reason:                NotYetOccured
    Status:                False
  Config Version:
    Current:           rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b
    Desired:           rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b (2)
  Health:              Healthy
  Most Recent Error:
  Observed Generation:  3
```

(1) 在节点上检测到新配置时，**spec.configversion.desired** 字段指定所需的配置会立即更新。

(2) 只有在 Machine Config Daemon (MCD)验证新配置时，**status.configversion.desired** 字段中指定的配置才会更新。MCD 通过检查更新的当前阶段来执行验证。如果更新成功通过 **UPDATEPREPARED** 阶段，则状态会添加新配置。

7.1.7. 查看证书并与其交互

以下证书由 Machine Config Controller (MCC) 在集群中处理，并可在 **ControllerConfig** 资源中找到：

- **/etc/kubernetes/kubelet-ca.crt**
- **/etc/kubernetes/static-pod-resources/configmaps/cloud-config/ca-bundle.pem**
- **/etc/pki/ca-trust/source/anchors/openshift-config-user-ca-bundle.crt**

MCC 还处理镜像 registry 证书及其关联的用户捆绑包证书。

您可以获取有关列出的证书的信息，包括减少证书的捆绑包，以及签名和主题数据。

流程

- 运行以下命令来获取详细的证书信息：

```
$ oc get controllerconfig/machine-config-controller -o yaml | yq -y
'.status.controllerCertificates'
```

输**出示例**

```
"controllerCertificates": [
                {
                    "bundleFile": "KubeAPIServerServingCAData",
                    "signer": "<signer_data1>",
                    "subject": "CN=openshift-kube-apiserver-operator_node-system-admin-
signer@168909215"
                },
                {
                    "bundleFile": "RootCAData",
                    "signer": "<signer_data2>",
                    "subject": "CN=root-ca,OU=openshift"
                }
            ]
```

- 使用以下命令检查机器配置池状态，获取 ControllerConfig 中找到信息的更简单版本：

```
$ oc get mcp master -o yaml | yq -y '.status.certExpirys'
```

输**出示例**

```
status:
  certExpirys:
  - bundle: KubeAPIServerServingCAData
    subject: CN=admin-kubeconfig-signer,OU=openshift
  - bundle: KubeAPIServerServingCAData
    subject: CN=kube-csr-signer_@1689585558
  - bundle: KubeAPIServerServingCAData
    subject: CN=kubelet-signer,OU=openshift
  - bundle: KubeAPIServerServingCAData
    subject: CN=kube-apiserver-to-kubelet-signer,OU=openshift
  - bundle: KubeAPIServerServingCAData
    subject: CN=kube-control-plane-signer,OU=openshift
```

此方法适用于已经消耗机器配置池信息的 OpenShift Container Platform 应用程序。

- 通过查看 **/etc/docker/cert.d** 目录的内容来检查节点上哪些镜像 registry 证书：

```
# ls /etc/docker/certs.d
```

输**出示例**

```
image-registry.openshift-image-registry.svc.cluster.local:5000 image-registry.openshift-image-registry.svc:5000
```

7.2. 使用 MACHINECONFIG 对象配置节点

您可以使用本节中的任务创建 **MachineConfig** 对象，修改 OpenShift Container Platform 节点上运行的文件、systemd 单元文件和其他操作系统功能。有关使用机器配置的更多信息，请参阅有关 更新 SSH 授权密钥、验证镜像签名、启用 SCTP 的内容，并为 OpenShift Container Platform 配置 iSCSI initiatorname。

OpenShift Container Platform 支持 Ignition 规格版本 3.2。您创建的所有新机器配置都应该基于 Ignition 规格版本 3.2。如果要升级 OpenShift Container Platform 集群，任何现有的 Ignition 规格版本 2.x 机器配置将自动转换为规格版本 3.2。

在某些情况下，节点上的配置与当前应用的机器配置指定不完全匹配。这个状态被称为 *配置偏移*。Machine Config Daemon(MCD)定期检查节点是否有配置偏移。如果 MCD 检测到配置偏移，MCO 会将节点标记为 **降级(degraded)**，直到管理员更正节点配置。降级的节点在线且可操作，但无法更新。有关配置偏移的更多信息，请参阅了解配置偏移检测。

提示

使用 "Configuring chrony time service" 部分作为如何将其他配置文件添加到 OpenShift Container Platform 节点的模型。

7.2.1. 配置 chrony 时间服务

您可以通过修改 chrony **.conf 文件的内容，并将这些内容作为机器配置传递给节点，从而设置 chrony** 时间服务(**chronyd**)使用的时间服务器和相关设置。

流程

1. 创建一个 Butane 配置，包括 **chrony.conf** 文件的内容。例如，要在 worker 节点上配置 chrony，请创建一个 **99-worker-chrony.bu** 文件。

注意

如需有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
variant: openshift
version: 4.15.0
metadata:
  name: 99-worker-chrony (1)
  labels:
    machineconfiguration.openshift.io/role: worker (2)
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644 (3)
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst (4)
```

```
    driftfile /var/lib/chrony/drift
    makestep 1.0 3
    rtcsync
    logdir /var/log/chrony
```

1 2 在 control plane 节点上，在这两个位置中将 **master** 替换为 **worker**。

3 为机器配置文件的 **mode** 字段指定数值模式。在创建文件并应用更改后，**模式** 将转换为十进制值。您可以使用 **oc get mc <mc-name> -o yaml** 命令来检查 YAML 文件。

4 指定任何有效的、可访问的时间源，如 DHCP 服务器提供的源。或者，您可以指定以下 NTP 服务器：**1.rhel.pool.ntp.org**, **2.rhel.pool.ntp.org**, 或 **3.rhel.pool.ntp.org**。

2. 使用 Butane 生成 **MachineConfig** 对象文件 **99-worker-chrony.yaml**，其中包含要交付至节点的配置：

```
$ butane 99-worker-chrony.bu -o 99-worker-chrony.yaml
```

3. 使用以下两种方式之一应用配置：

- 如果集群还没有运行，在生成清单文件后，将 **MachineConfig** 对象文件添加到 **<installation_directory>/openshift** 目录中，然后继续创建集群。
- 如果集群已在运行，请应用该文件：

```
$ oc apply -f ./99-worker-chrony.yaml
```

其他资源

- 使用 Butane 创建机器配置

7.2.2. 禁用 chrony 时间服务

您可以使用 **MachineConfig** 自定义资源 (CR) 为具有特定角色的节点禁用 chrony 时间服务 (**chronyd**)。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建 **MachineConfig** CR，为指定节点角色禁用 **chronyd**。
   a. 在 **disable-chronyd.yaml** 文件中保存以下 YAML：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role> 1
  name: disable-chronyd
spec:
```

```
config:
  ignition:
    version: 3.2.0
  systemd:
    units:
      - contents: |
          [Unit]
          Description=NTP client/server
          Documentation=man:chronyd(8) man:chrony.conf(5)
          After=ntpdate.service sntp.service ntpd.service
          Conflicts=ntpd.service systemd-timesyncd.service
          ConditionCapability=CAP_SYS_TIME
          [Service]
          Type=forking
          PIDFile=/run/chrony/chronyd.pid
          EnvironmentFile=-/etc/sysconfig/chronyd
          ExecStart=/usr/sbin/chronyd $OPTIONS
          ExecStartPost=/usr/libexec/chrony-helper update-daemon
          PrivateTmp=yes
          ProtectHome=yes
          ProtectSystem=full
          [Install]
          WantedBy=multi-user.target
        enabled: false
        name: "chronyd.service"
```

1 要禁用 **chronyd** 的节点角色，如 **master**。

b. 运行以下命令来创建 **MachineConfig** CR：

```
$ oc create -f disable-chronyd.yaml
```

7.2.3. 为节点添加内核参数

在一些特殊情况下，您可能需要为集群中的一组节点添加内核参数。进行此操作时应小心谨慎，而且您必须先清楚了解所设参数的影响。

警告

不当使用内核参数会导致系统变得无法引导。

您可以设置的内核参数示例包括：

- **nosmt**：在内核中禁用对称多线程 (SMT)。多线程允许每个 CPU 有多个逻辑线程。您可以在多租户环境中考虑使用 **nosmt**，以减少潜在的跨线程攻击风险。禁用 SMT 在本质上相当于选择安全性而非性能。
- **systemd.unified_cgroup_hierarchy**：启用 Linux 控制组版本 2 (cgroup v2)。cgroup v2 是内核控制组的下一个版本，它包括了多个改进。

- **Enforcing=0**：将 Security Enhanced Linux（SELinux）配置为以 permissive 模式运行。在 permissive 模式中，系统会象 enforcing 模式一样加载安全策略，包括标记对象并在日志中记录访问拒绝条目，但它并不会拒绝任何操作。虽然不建议在生产环境系统中使用 permissive 模式，但 permissive 模式会有助于调试。

警告

不支持在生产环境中禁用 RHCOS 上的 SELinux。在节点上禁用 SELinux 后，必须在生产集群中重新设置前重新置备它。

如需内核参数的列表和描述，请参阅 Kernel.org 内核参数。

在以下流程中，您要创建一个用于标识以下内容的 **MachineConfig** 对象：

- 您要添加内核参数的一组机器。本例中为具有 worker 角色的机器。
- 附加到现有内核参数末尾的内核参数。
- 指示机器配置列表中应用更改的位置的标签。

先决条件

- 具有正常运行的 OpenShift Container Platform 集群的管理特权。

流程

1. 列出 OpenShift Container Platform 集群的现有 **MachineConfig** 对象，以确定如何标记您的机器配置：

```
$ oc get MachineConfig
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                       3.2.0             40m
```

```
99-worker-generated-registries                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-worker-ssh                                                    3.2.0          40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
```

2. 创建一个用于标识内核参数的 **MachineConfig** 对象文件（例如 **05-worker-kernelarg-selinuxpermissive.yaml**）

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 05-worker-kernelarg-selinuxpermissive 2
spec:
  kernelArguments:
    - enforcing=0 3
```

1. 仅将新内核参数应用到 worker 节点。
2. 用于标识它插入到机器配置中的什么位置（05）以及发挥什么作用（添加一个内核参数来配置 SELinux permissive 模式）。
3. 将确切的内核参数标识为 **enforcing=0**。

3. 创建新机器配置：

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. 检查机器配置以查看是否添加了新配置：

```
$ oc get MachineConfig
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-master-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
05-worker-kernelarg-selinuxpermissive                                                3.2.0          105s
```

```
99-master-generated-registries                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-master-ssh                                                        3.2.0          40m
99-worker-generated-registries                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-worker-ssh                                                        3.2.0          40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
```

5. 检查节点：

```
$ oc get nodes
```

输出示例

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                        worker   28m   v1.28.5
ip-10-0-136-243.ec2.internal   Ready                        master   34m   v1.28.5
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.28.5
ip-10-0-142-249.ec2.internal   Ready                        master   34m   v1.28.5
ip-10-0-153-11.ec2.internal    Ready                       worker   28m   v1.28.5
ip-10-0-153-150.ec2.internal   Ready                        master   34m   v1.28.5
```

您可以发现，在应用更改时每个 worker 节点上的调度都会被禁用。

6. 前往其中一个 worker 节点并列出内核命令行参数（主机上的 **/proc/cmdline** 中），以检查内核参数确实已发挥作用：

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

输出示例

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

您应看到 **enforcing=0** 参数已添加至其他内核参数。

7.2.4. 在 RHCOS 上启用带有内核参数的多路径

Red Hat Enterprise Linux CoreOS (RHCOS) 支持主磁盘上的多路径，允许对硬件故障进行更强大的弹性，以实现更高的主机可用性。通过机器配置激活多路径，提供安装后支持。

重要

对于在 OpenShift Container Platform 4.8 或更高版本中置备的节点，推荐在安装过程中启用多路径。在任何 I/O 到未优化路径会导致 I/O 系统错误的设置中，您必须在安装时启用多路径。有关在安装过程中启用多路径的更多信息，请参阅*在裸机上安装*中的 "使用 RHCOS 上内核参数启用多路径"。

重要

在 IBM Z® 和 IBM® LinuxONE 中，您只能在在安装过程中为它配置集群时启用多路径。如需更多信息，请参阅在 *IBM Z® 和 IBM® LinuxONE 上安装使用 z/VM 的集群*"安装 RHCOS 并启动 OpenShift Container Platform bootstrap 过程"。

重要

当在配置了多路径的 IBM Power® 上的带有 "vSCSI" 存储的单个 VIOS 主机中安装或配置了 OpenShift Container Platform 4.15 集群作为安装后任务时，启用了多路径的 CoreOS 节点无法引导。此行为是正常的，因为只有一个路径可用于节点。

先决条件

- 您有一个正在运行的 OpenShift Container Platform 集群，它使用版本 4.7 或更高版本。
- 以具有管理特权的用户身份登录集群。
- 您已确认为多路径启用了磁盘。只有通过 HBA 适配器连接到 SAN 的主机上才支持多路径。

流程

1. 要在 control plane 节点上启用多路径安装后：
   - 创建机器配置文件，如 **99-master-kargs-mpath.yaml**，该文件指示集群添加 **master** 标签并标识多路径内核参数，例如：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "master"
  name: 99-master-kargs-mpath
spec:
  kernelArguments:
    - 'rd.multipath=default'
    - 'root=/dev/disk/by-label/dm-mpath-root'
```

2. 在 worker 节点上启用多路径安装后：
   - 创建机器配置文件，如 **99-worker-kargs-mpath.yaml**，该文件指示集群添加 **worker** 标签并标识多路径内核参数，例如：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
```

```
  name: 99-worker-kargs-mpath
spec:
  kernelArguments:
    - 'rd.multipath=default'
    - 'root=/dev/disk/by-label/dm-mpath-root'
```

3. 使用之前创建的 master 或 worker YAML 文件创建新机器配置：

```
$ oc create -f ./99-worker-kargs-mpath.yaml
```

4. 检查机器配置以查看是否添加了新配置：

```
$ oc get MachineConfig
```

输**出示例**

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                       3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-kargs-mpath                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             105s
99-worker-ssh                                                                       3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

5. 检查节点：

```
$ oc get nodes
```

输**出示例**

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.28.5
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.28.5
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.28.5
```

```
ip-10-0-142-249.ec2.internal   Ready                 master   34m   v1.28.5
ip-10-0-153-11.ec2.internal    Ready                 worker   28m   v1.28.5
ip-10-0-153-150.ec2.internal   Ready                 master   34m   v1.28.5
```

您可以发现，在应用更改时每个 worker 节点上的调度都会被禁用。

6. 前往其中一个 worker 节点并列出内核命令行参数（主机上的 **/proc/cmdline** 中），以检查内核参数确实已发挥作用：

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

输**出示例**

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
...
rd.multipath=default root=/dev/disk/by-label/dm-mpath-root
...

sh-4.2# exit
```

您应看到添加的内核参数。

其他资源

- 有关在安装过程中启用多路径的更多信息，请参阅在 RHCOS 上启用使用内核参数的多路径。

7.2.5. 在节点中添加实时内核

一些 OpenShift Container Platform 工作负载需要高度确定性。虽然 Linux 不是实时操作系统，但 Linux 实时内核包含一个抢占调度程序，它为操作系统提供实时特征。

如果您的 OpenShift Container Platform 工作负载需要这些实时特征，您可以将机器切换到 Linux 实时内核。对于 OpenShift Container Platform，4.15 您可以使用 **MachineConfig** 对象进行这个切换。虽然进行这个切换非常简单（只需要把机器配置的 **kernelType** 设置为 **realtime**），但进行更改前需要注意：

- 目前，实时内核只支持在 worker 节点上运行，且只支持无线电访问网络（RAN）使用。
- 使用为 Red Hat Enterprise Linux for Real Time 8 认证系统的裸机安装完全支持以下步骤。
- OpenShift Container Platform 中的实时支持仅限于特定的订阅。
- 以下流程也支持与 Google Cloud Platform 搭配使用。

先决条件

- 有一个正在运行的 OpenShift Container Platform 集群（版本 4.4 或更高版本）。
- 以具有管理特权的用户身份登录集群。

流程

1. 为实时内核创建一个机器配置：创建一个 YAML 文件（例如，**99-worker-realtime.yaml**），其中包含一个 **realtime** 内核类型的 **MachineConfig** 对象。本例告诉集群在所有 worker 节点中使用实时内核：

```
$ cat << EOF > 99-worker-realtime.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
  name: 99-worker-realtime
spec:
  kernelType: realtime
EOF
```

2. 将机器配置添加到集群。键入以下内容将机器配置添加到集群中：

```
$ oc create -f 99-worker-realtime.yaml
```

3. 检查实时内核：每当受影响节点重新引导后，登录到集群，并运行以下命令来确保您配置的节点组中使用实时内核替换了常规内核：

```
$ oc get nodes
```

输**出示例**

```
NAME                                         STATUS  ROLES    AGE   VERSION
ip-10-0-143-147.us-east-2.compute.internal  Ready   worker   103m  v1.28.5
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.28.5
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.28.5
```

```
$ oc debug node/ip-10-0-143-147.us-east-2.compute.internal
```

输**出示例**

```
Starting pod/ip-10-0-143-147us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`

sh-4.4# uname -a
Linux <worker_node> 4.18.0-147.3.1.rt24.96.el8_1.x86_64 #1 SMP PREEMPT RT
        Wed Nov 27 18:29:55 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

内核名称包含 **rt** 和 "PREMPT RT" 来表示这是一个实时内核。

4. 要返回常规内核，请删除 **MachineConfig** 对象：

```
$ oc delete -f 99-worker-realtime.yaml
```

7.2.6. 配置 journald 设置

如果您需要在 OpenShift Container Platform 节点上配置 **journald** 服务设置，您可以修改适当的配置文件并将该文件作为机器配置传递给适当的节点池。

此流程描述了如何修改 **/etc/systemd/journald.conf** 文件中的 **journald** 限制设置并将其应用到 worker 节点。有关如何使用该文件的详情，请查看 **journald.conf** 手册页。

先决条件

- 有一个正在运行的 OpenShift Container Platform 集群。
- 以具有管理特权的用户身份登录集群。

流程

1. 创建一个 Butane 配置文件 **40-worker-custom-journald.bu**，其中包含带有所需设置的 **/etc/systemd/journald.conf** 文件。

注意

有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
variant: openshift
version: 4.15.0
metadata:
  name: 40-worker-custom-journald
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/systemd/journald.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        # Disable rate limiting
        RateLimitInterval=1s
        RateLimitBurst=10000
        Storage=volatile
        Compress=no
        MaxRetentionSec=30s
```

2. 使用 Butane 生成 **MachineConfig** 对象文件 **40-worker-custom-journald.yaml**，包含要发送到 worker 节点的配置：

```
$ butane 40-worker-custom-journald.bu -o 40-worker-custom-journald.yaml
```

3. 将机器配置应用到池：

```
$ oc apply -f 40-worker-custom-journald.yaml
```

4. 检查是否应用新机器配置，并且节点是否处于降级状态。它可能需要几分钟时间。worker 池将显示更新进行中，每个节点都成功应用了新的机器配置：

```
$ oc get machineconfigpool
NAME   CONFIG              UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
```

```
AGE
master rendered-master-35 True    False    False    3         3         3         0
34m
worker rendered-worker-d8 False   True     False    3         1         1         0
34m
```

5. 要检查是否应用了更改，您可以登录到 worker 节点：

```
$ oc get node | grep worker
ip-10-0-0-1.us-east-2.compute.internal   Ready    worker   39m   v0.0.0-master+$Format:%h$
$ oc debug node/ip-10-0-0-1.us-east-2.compute.internal
Starting pod/ip-10-0-141-142us-east-2computeinternal-debug ...
...
sh-4.2# chroot /host
sh-4.4# cat /etc/systemd/journald.conf
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
sh-4.4# exit
```

其他资源

- 使用 Butane 创建机器配置

7.2.7. 为 RHCOS 添加扩展

RHCOS 是基于容器的最小 RHEL 操作系统,旨在为所有平台的 OpenShift Container Platform 集群提供一组通用的功能。通常不建议在 RHCOS 系统中添加软件软件包，但 MCO 提供了一个 **extensions（扩展）** 功能，您可以使用 MCO 为 RHCOS 节点添加一组最小的功能。

目前，有以下扩展可用：

- **usbguard**：添加 **usbguard** 扩展可保护 RHCOS 系统不受入侵 USB 设备的攻击。详情请查看 USBGuard。
- **Kerberos**：添加 **kerberos** 扩展提供了一种机制，允许用户和机器标识自身对网络进行定义的定义、限制对管理员配置的区域和服务的访问权限。请参阅使用 Kerberos 的详情，包括如何设置 Kerberos 客户端并挂载 Kerberized NFS 共享。

以下流程描述了如何使用机器配置为 RHCOS 节点添加一个或多个扩展。

先决条件

- 有一个正在运行的 OpenShift Container Platform 集群（版本 4.6 或更高版本）。
- 以具有管理特权的用户身份登录集群。

流程

1. 为扩展创建机器配置：创建一个 YAML 文件（如 **80-extensions.yaml**），其中包含 **MachineConfig extensions** 对象。本例告诉集群添加 **usbguard** 扩展。

```
$ cat << EOF > 80-extensions.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 80-worker-extensions
spec:
  config:
    ignition:
      version: 3.2.0
  extensions:
    - usbguard
EOF
```

2. 将机器配置添加到集群。键入以下内容将机器配置添加到集群中：

```
$ oc create -f 80-extensions.yaml
```

这会将所有 worker 节点设置为安装 **usbguard** 的 rpm 软件包。

3. 检查是否应用了扩展：

```
$ oc get machineconfig 80-worker-extensions
```

输**出示例**

```
NAME                 GENERATEDBYCONTROLLER IGNITIONVERSION AGE
80-worker-extensions                       3.2.0           57s
```

4. 检查是否应用新机器配置，并且节点是否处于降级状态。它可能需要几分钟时间。worker 池将显示更新进行中，每台机器都成功应用了新机器配置：

```
$ oc get machineconfigpool
```

输**出示例**

```
NAME   CONFIG              UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True    False    False    3          3              3                0
34m
worker rendered-worker-d8 False   True     False    3          1              1                0
34m
```

5. 检查扩展。要检查是否应用了扩展，请运行：

```
$ oc get node | grep worker
```

输**出示例**

```
NAME                                         STATUS  ROLES    AGE   VERSION
ip-10-0-169-2.us-east-2.compute.internal     Ready   worker   102m  v1.28.5
```

```
$ oc debug node/ip-10-0-169-2.us-east-2.compute.internal
```

输出示例

```
...
To use host binaries, run `chroot /host`
sh-4.4# chroot /host
sh-4.4# rpm -q usbguard
usbguard-0.7.4-4.el8.x86_64.rpm
```

7.2.8. 在机器配置清单中载入自定义固件 Blob

因为 **/usr/lib** 中固件 Blob 的默认位置是只读的，所以您可以通过更新搜索路径来查找自定义固件 Blob。这可让您在 RHCOS 不管理 blob 时载入机器配置清单中的本地固件 Blob。

流程

1. 创建 Butane 配置文件 **98-worker-firmware-blob.bu**，它会更新搜索路径，以便其为 root 所有且对本地存储可写。以下示例将本地工作站的自定义 blob 文件放在 **/var/lib/firmware** 下的节点上。

注意

有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

自定义固件 blob 的 Butane 配置文件

```
variant: openshift
version: 4.15.0
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 98-worker-firmware-blob
storage:
  files:
  - path: /var/lib/firmware/<package_name> 1
    contents:
      local: <package_name> 2
    mode: 0644 3
openshift:
  kernel_arguments:
    - 'firmware_class.path=/var/lib/firmware' 4
```

1. 设置将固件软件包复制到的节点上的路径。
2. 指定包含从运行 Butane 的系统上本地文件目录中读取的内容的文件。本地文件的路径相对于 **files-dir** 目录，必须在下一步中使用 **--files-dir** 选项指定它。
3. 为 RHCOS 节点上的文件设置权限。建议把选项设置为 **0644**。

4 **firmware_class.path** 参数自定义内核搜索路径，在其中查找从本地工作站复制到节点的根文件系统的自定义固件 Blob。这个示例使用 **/var/lib/firmware** 作为自定义路径。

2. 运行 Butane 生成 **MachineConfig** 对象文件，该文件使用名为 **98-worker-firmware-blob.yaml** 的本地工作站中的固件 blob 副本。固件 blob 包含要传送到节点的配置。以下示例使用 **--files-dir** 选项指定工作站上本地文件或目录所在的目录：

```
$ butane 98-worker-firmware-blob.bu -o 98-worker-firmware-blob.yaml --files-dir
<directory_including_package_name>
```

3. 通过两种方式之一将配置应用到节点：

- 如果集群还没有运行，在生成清单文件后，将 **MachineConfig** 对象文件添加到 **<installation_directory>/openshift** 目录中，然后继续创建集群。
- 如果集群已在运行，请应用该文件：

```
$ oc apply -f 98-worker-firmware-blob.yaml
```

已为您创建一个 **MachineConfig** 对象 YAML 文件，以完成机器的配置。

4. 如果将来需要更新 **MachineConfig** 对象，请保存 Butane 配置。

其他资源

- 使用 Butane 创建机器配置

7.2.9. 更改节点访问的核心用户密码

默认情况下，Red Hat Enterprise Linux CoreOS (RHCOS) 在集群的节点上创建一个名为 **core** 的用户。您可以使用 **core** 用户通过云供应商串口控制台或裸机基板管理控制器管理器 (BMC) 访问节点。例如，如果节点停机且您无法使用 SSH 或 **oc debug node** 命令访问该节点，这非常有用。但是，默认情况下，此用户没有密码，因此您无法在不创建密码的情况下登录。

您可以使用机器配置为 **core** 用户创建密码。Machine Config Operator (MCO) 分配密码并将密码注入 **/etc/shadow** 文件中，允许您使用 **core** 用户登录。MCO 不会检查密码哈希。因此，如果密码出现问题，MCO 无法报告。

注意

- 密码只能通过云供应商串口控制台或 BMC 正常工作。它不适用于 SSH。
- 如果您有包含 **/etc/shadow** 文件或 systemd 单元的机器配置，则优先于密码哈希。

您可以通过编辑用于创建密码的机器配置来更改密码。另外，您可以通过删除机器配置来删除密码。删除机器配置不会删除用户帐户。

流程

1. 使用您的操作系统支持的工具，创建一个哈希密码。例如，通过运行以下命令，使用 **mkpasswd** 创建哈希密码：

```
$ mkpasswd -m SHA-512 testpass
```

输**出示例**

```
$
$6$CBZwA6s6AVFOtiZe$aUKDWpthhJEyR3nnhM02NM1sKCpHn9XN.NPrJNQ3HYewioaorp
wL3mKGLxvW0AOb4pJxqoqP4nFX77y0p00.8.
```

2. 创建包含 **core** 用户名和散列密码的机器配置文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: set-core-user-password
spec:
  config:
    ignition:
      version: 3.2.0
    passwd:
      users:
      - name: core (1)
        passwordHash: <password> (2)
```

(1) 这必须是 **core**。

(2) 与 **core** 帐户一起使用的散列密码。

3. 运行以下命令来创建机器配置：

```
$ oc create -f <file-name>.yaml
```

节点不会重启，并在几分钟内可用。您可以使用 **oc get mcp** 来监控要更新的机器配置池，如下例所示：

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-d686a3ffc8fdec47280afec446fce8dd   True      False      False      3
3              3                   0                   64m
worker   rendered-worker-4605605a5b1f9de1d061e9d350f251e5   False     True       False
3              0                   0                   0                   64m
```

验证

1. 节点返回 **UPDATED=True** 状态后，运行以下命令为节点启动 debug 会话：

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

2. 运行以下命令，将 **/host** 设置为 debug shell 中的根目录：

```
sh-4.4# chroot /host
```

3. 检查 **/etc/shadow** 文件的内容：

   输**出示例**

```
...
core:$6$2sE/010goDuRSxxv$o18K52wor.wIwZp:19418:0:99999:7:::
...
```

   哈希密码分配给 **core** 用户。

7.3. 配置 MCO 相关的自定义资源

除了管理 **MachineConfig** 对象外，MCO 管理两个自定义资源（CR）：**KubeletConfig** 和 **ContainerRuntimeConfig**。这些 CR 可让您更改节点级别的设置，这会影响到 Kubelet 和 CRI-O 容器运行时服务的行为。

7.3.1. 创建 KubeletConfig CRD 来编辑 kubelet 参数

kubelet 配置目前被序列化为 Ignition 配置，因此可以直接编辑。但是，在 Machine Config Controller (MCC) 中同时添加了新的 **kubelet-config-controller**。这可让您使用 **KubeletConfig** 自定义资源 (CR) 来编辑 kubelet 参数。

注意

因为 **kubeletConfig** 对象中的字段直接从上游 Kubernetes 传递给 kubelet，kubelet 会直接验证这些值。**kubeletConfig** 对象中的无效值可能会导致集群节点不可用。有关有效值，请参阅 Kubernetes 文档。

请考虑以下指导：

- 编辑现有的 **KubeletConfig** CR 以修改现有设置或添加新设置，而不是为每个更改创建一个 CR。建议您仅创建一个 CR 来修改不同的机器配置池，或用于临时更改，以便您可以恢复更改。
- 为每个机器配置池创建一个 **KubeletConfig** CR，带有该池需要更改的所有配置。
- 根据需要，创建多个 **KubeletConfig** CR，每个集群限制为 10。对于第一个 **KubeletConfig** CR，Machine Config Operator (MCO) 会创建一个机器配置，并附带 **kubelet**。对于每个后续 CR，控制器会创建另一个带有数字后缀的 **kubelet** 机器配置。例如，如果您有一个带有 **-2** 后缀的 **kubelet** 机器配置，则下一个 **kubelet** 机器配置会附加 **-3**。

注意

如果要将 kubelet 或容器运行时配置应用到自定义机器配置池，则 **machineConfigSelector** 中的自定义角色必须与自定义机器配置池的名称匹配。

例如，由于以下自定义机器配置池名为 **infra**，因此自定义角色也必须是 **infra** ：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
# ...
```

如果要删除机器配置，以相反的顺序删除它们，以避免超过限制。例如，在删除 **kubelet-2** 机器配置前删除 **kubelet-3** 机器配置。

注意

如果您有一个带有 **kubelet-9** 后缀的机器配置，并且创建了另一个 **KubeletConfig** CR，则不会创建新的机器配置，即使少于 10 个 **kubelet** 机器配置。

KubeletConfig CR 示例

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

显示 **KubeletConfig 机器配置示例**

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                  b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
26m
...
```

以下流程演示了如何配置 worker 节点上的每个节点的最大 pod 数量。

先决条件

1. 为您要配置的节点类型获取与静态 **MachineConfigPool** CR 关联的标签。执行以下步骤之一：
   a. 查看机器配置池：

      ```
      $ oc describe machineconfigpool <name>
      ```

      例如：

```
$ oc describe machineconfigpool worker
```

输**出示例**

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods 1
```

1 如果添加了标签，它会出现在 **labels** 下。

b. 如果标签不存在，则添加一个键/值对：

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

流程

1. 查看您可以选择的可用机器配置对象：

```
$ oc get machineconfig
```

默认情况下，与 kubelet 相关的配置为 **01-master-kubelet** 和 **01-worker-kubelet**。

2. 检查每个节点的最大 pod 的当前值：

```
$ oc describe node <node_name>
```

例如：

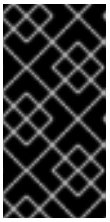
```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

在 **Allocatable** 小节中找到 **value: pods: <value>**：

输**出示例**

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

3. 通过创建一个包含 kubelet 配置的自定义资源文件，设置 worker 节点上的每个节点的最大 pod：



重要

以特定机器配置池为目标的 kubelet 配置也会影响任何依赖的池。例如，为包含 worker 节点的池创建 kubelet 配置也适用于任何子集池，包括包含基础架构节点的池。要避免这种情况，您必须使用仅包含 worker 节点的选择表达式创建新的机器配置池，并让 kubelet 配置以这个新池为目标。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods (1)
  kubeletConfig:
    maxPods: 500 (2)
```

1. 输入机器配置池中的标签。
2. 添加 kubelet 配置。在本例中，使用 **maxPods** 设置每个节点的最大 pod。

注意

kubelet 与 API 服务器进行交互的频率取决于每秒的查询数量 (QPS) 和 burst 值。如果每个节点上运行的 pod 数量有限，使用默认值（**kubeAPIQPS** 为 **50**，**kubeAPIBurst** 为 **100**）就可以。如果节点上有足够 CPU 和内存资源，则建议更新 kubelet QPS 和 burst 速率。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 为带有标签的 worker 更新机器配置池：

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

b. 创建 **KubeletConfig** 对象：

```
$ oc create -f change-maxPods-cr.yaml
```

c. 验证 **KubeletConfig** 对象是否已创建：

```
$ oc get kubeletconfig
```

输出示例

```
NAME                AGE
set-max-pods        15m
```

根据集群中的 worker 节点数量，等待每个 worker 节点被逐个重启。对于有 3 个 worker 节点的集群，这个过程可能需要大约 10 到 15 分钟。

4. 验证更改是否已应用到节点：

   a. 在 worker 节点上检查 **maxPods** 值已更改：

   ```
   $ oc describe node <node_name>
   ```

   b. 找到 **Allocatable** 小节：

   ```
    ...
   Allocatable:
     attachable-volumes-gce-pd:  127
     cpu:                        3500m
     ephemeral-storage:          123201474766
     hugepages-1Gi:              0
     hugepages-2Mi:              0
     memory:                     14225400Ki
     pods:                       500 (1)
    ...
   ```

   (1) 在本例中，**pods** 参数应报告您在 **KubeletConfig** 对象中设置的值。

5. 验证 **KubeletConfig** 对象中的更改：

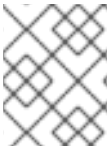
   ```
   $ oc get kubeletconfigs set-max-pods -o yaml
   ```

   这应该显示 **True** 状态和 **type:Success**，如下例所示：

   ```
   spec:
     kubeletConfig:
       maxPods: 500
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: set-max-pods
   status:
     conditions:
     - lastTransitionTime: "2021-06-30T17:04:07Z"
       message: Success
       status: "True"
       type: Success
   ```

7.3.2. 创建 ContainerRuntimeConfig CR 以编辑 CRI-O 参数

您可以为与特定机器配置池（MCP）关联的节点更改与 OpenShift Container Platform CRI-O 运行时关联的一些设置。通过使用 **ContainerRuntimeConfig** 自定义资源（CR），您可以设置配置值并添加一个标签以匹配 MCP。然后，MCO 会使用更新的值重建关联节点上的 **crio.conf** 和 **storage.conf** 配置文件。



注意

要使用 **ContainerRuntimeConfig** CR 恢复实现的更改，您必须删除 CR。从机器配置池中删除标签不会恢复更改。

您可以使用 **ContainerRuntimeConfig** CR 修改以下设置：

- **PIDs limit** ：在 **ContainerRuntimeConfig** 中设置 PID 限值将被弃用。如果需要 PIDs 限制，建议在 **KubeletConfig** CR 中使用 **podPidsLimit** 字段。**podPidsLimit** 字段的默认值为 **4096**。

  注意

  CRI-O 标志应用到容器的 cgroup 上，而 Kubelet 标志则在 pod 的 cgroup 中设置。请相应地调整 PID 限值。

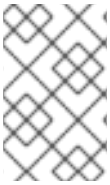
- **日志级别**: **logLevel** 参数设置 CRI-O **log_level** 参数，即日志消息的详细程度。默认为 **info** (**log_level = info**)。其他选项包括 **fatal**、**panic**、**error**、**warn**、**debug** 和 **trace**。
- **Overlay 大小**：**overlaySize** 参数设置 CRI-O Overlay 存储驱动程序 **size** 参数，这是容器镜像的最大大小。
- **最大日志大小** ：在 **ContainerRuntimeConfig** 中设置最大日志大小被弃用。如果需要最大日志大小，建议在 **KubeletConfig** CR 中使用 **containerLogMaxSize** 字段。
- **容器运行时** ： **defaultRuntime** 参数将容器运行时设置为 **runc** 或 **crun**。默认为 **runc**。

您应该为每个机器配置池有一个**ContainerRuntimeConfig** CR，并为该池分配所有配置更改。如果要将相同的内容应用到所有池，则所有池只需要 **one****ContainerRuntimeConfig** CR。

您应该编辑现有的 **ContainerRuntimeConfig** CR，以修改现有设置或添加新设置，而不是为每个更改创建新 CR。建议您只创建一个新的 **ContainerRuntimeConfig** CR 来修改不同的机器配置池，或者用于临时的更改，以便您可以恢复更改。

您可以根据需要创建多个 **ContainerRuntimeConfig** CR，每个集群的限制为 10。对于第一个 **ContainerRuntimeConfig** CR，MCO 会创建一个机器配置并附加 **containerruntime**。对于每个后续 CR，控制器会创建一个带有数字后缀的新 **containerruntime** 机器配置。例如，如果您有一个带有 **-2** 后缀的 **containerruntime** 机器配置，则下一个 **containerruntime** 机器配置会附加 **-3**。

如果要删除机器配置，应该以相反的顺序删除它们，以避免超过限制。例如，您应该在删除 **containerruntime-2** 机器配置前删除 **containerruntime-3** 机器配置。



注意

如果您的机器配置带有 **containerruntime-9** 后缀，并且创建了 **anotherContainerRuntimeConfig** CR，则不会创建新的机器配置，即使少于 10 个 **containerruntime** 机器配置。

显**示多个 ContainerRuntimeConfig CR 示例**

```
$ oc get ctrcfg
```

输出示例

```
NAME         AGE
ctr-overlay  15m
ctr-level    5m45s
```

显示多个 containerruntime 机器配置示例

```
$ oc get mc | grep container
```

输出示例

```
...
01-master-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
57m
...
01-worker-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
57m
...
99-worker-generated-containerruntime            b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0            26m
99-worker-generated-containerruntime-1          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0            17m
99-worker-generated-containerruntime-2          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0            7m26s
...
```

以下示例将 **log_level** 字段设置为 **debug**，并将覆盖大小设置为 8 GB：

ContainerRuntimeConfig CR 示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: " 1
 containerRuntimeConfig:
   logLevel: debug 2
   overlaySize: 8G 3
   defaultRuntime: "crun" 4
```

1 指定机器配置池标签。对于容器运行时配置，角色必须与关联的机器配置池的名称匹配。

2 可选：指定日志消息的详细程度。

3 可选：指定容器镜像的最大大小。

4 可选：指定部署到新容器的容器运行时。默认值为 **runc**。

流程

使用 **ContainerRuntimeConfig** CR 更改 CRI-O 设置：

1. 为 **ContainerRuntimeConfig** CR 创建 YAML 文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: " 1
 containerRuntimeConfig: 2
   logLevel: debug
   overlaySize: 8G
```

1 为您要修改的机器配置池指定一个标签。

2 根据需要设置参数。

2. 创建 **ContainerRuntimeConfig** CR：

```
$ oc create -f <file_name>.yaml
```

3. 验证是否已创建 CR：

```
$ oc get ContainerRuntimeConfig
```

输**出示例**

```
NAME           AGE
overlay-size   3m19s
```

4. 检查是否创建了新的 **containerruntime** 机器配置：

```
$ oc get machineconfigs | grep containerrun
```

输**出示例**

```
99-worker-generated-containerruntime   2c9371fbb673b97a6fe8b1c52691999ed3a1bfc2
3.2.0  31s
```

5. 监控机器配置池，直到所有系统都显示为 ready 状态：

```
$ oc get mcp worker
```

输**出示例**

```
NAME    CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
worker  rendered-worker-169  False    True      False     3          1                  1                  0
9h
```

6. 验证设置是否在 CRI-O 中应用：

   a. 打开到机器配置池中节点的 **oc debug** 会话，并运行 **chroot /host**。

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

   ```
   sh-4.4# chroot /host
   ```

   b. 验证 **crio.conf** 文件中的更改：

   ```
   sh-4.4# crio config | grep 'log_level'
   ```

   输**出示例**

   ```
   log_level = "debug"
   ```

   c. 验证 'storage.conf' 文件中的更改：

   ```
   sh-4.4# head -n 7 /etc/containers/storage.conf
   ```

   输**出示例**

   ```
   [storage]
     driver = "overlay"
     runroot = "/var/run/containers/storage"
     graphroot = "/var/lib/containers/storage"
     [storage.options]
       additionalimagestores = []
       size = "8G"
   ```

7.3.3. 使用 CRI-O 为 Overlay 设置默认的最大容器根分区大小

每个容器的根分区显示底层主机的所有可用磁盘空间。按照以下说明，为所有容器的 root 磁盘设置最大分区大小。

要配置最大 Overlay 大小以及其他 CRI-O 选项，您可以创建以下 **ContainerRuntimeConfig** 自定义资源定义 (CRD)：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     custom-crio: overlay-size
```

```
containerRuntimeConfig:
  logLevel: debug
  overlaySize: 8G
```

流程

1. 创建配置对象：

```
$ oc apply -f overlaysize.yml
```

2. 要将新的 CRI-O 配置应用到 worker 节点，请编辑 worker 机器配置池：

```
$ oc edit machineconfigpool worker
```

3. 根据在 **ContainerRuntimeConfig** CRD 中设置的 **matchLabels** 名称添加 **custom-crio** 标签：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2020-07-09T15:46:34Z"
  generation: 3
  labels:
    custom-crio: overlay-size
    machineconfiguration.openshift.io/mco-built-in: ""
```

4. 保存更改，然后查看机器配置：

```
$ oc get machineconfigs
```

新的 **99-worker-generated-containerruntime** 和 **rendered-worker-xyz** 对象被创建：

输**出示例**

```
99-worker-generated-containerruntime  4173030d89fbf4a7a0976d1665491a4d9a6e54f1
3.2.0             7m42s
rendered-worker-xyz                   4173030d89fbf4a7a0976d1665491a4d9a6e54f1   3.2.0
7m36s
```

5. 创建这些对象后，监控机器配置池以了解要应用的更改：

```
$ oc get mcp worker
```

worker 节点将 **UPDATING** 显示为 **True**，以及机器数量、更新的数字和其他详情：

输**出示例**

```
NAME   CONFIG              UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz False True False     3           2                2                0
20h
```

完成后，worker 节点会从 **UPDATING** 转换回 **False**，**UPDATEDMACHINECOUNT** 数与 **MACHINECOUNT** 数匹配：

输**出示例**

```
NAME   CONFIG            UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker   rendered-worker-xyz   True      False      False      3       3          3          0
20h
```

查看 worker 机器，您会看到新的 8 GB 最大大小配置适用于所有 worker：

输**出示例**

```
head -n 7 /etc/containers/storage.conf
[storage]
  driver = "overlay"
  runroot = "/var/run/containers/storage"
  graphroot = "/var/lib/containers/storage"
  [storage.options]
    additionalimagestores = []
    size = "8G"
```

在容器内，您会看到 root 分区现在为 8 GB：

输**出示例**

```
~ $ df -h
Filesystem                Size      Used Available Use% Mounted on
overlay                   8.0G      8.0K      8.0G   0% /
```

7.3.4. 为 CRI-O 的默认功能创建一个置入文件

您可以为与特定机器配置池（MCP）关联的节点更改与 OpenShift Container Platform CRI-O 运行时关联的一些设置。通过使用控制器自定义资源(CR)，您可以设置配置值并添加标签以匹配 MCP。然后，MCO 会使用更新的值重建关联节点上的 **crio.conf** 和 **default.conf** 配置文件。

默认情况下，OpenShift Container Platform 的早期版本包含特定的机器配置。如果您升级到更新的 OpenShift Container Platform 版本，则会保留这些机器配置，以确保在同一 OpenShift Container Platform 版本上运行的集群具有相同的机器配置。

您可以根据需要创建多个 **ContainerRuntimeConfig** CR，每个集群的限制为 10。对于第一个 **ContainerRuntimeConfig** CR，MCO 会创建一个机器配置并附加 **containerruntime**。对于每个后续 CR，控制器会创建一个带有数字后缀的 **containerruntime** 机器配置。例如，如果您有一个带有 **-2** 后缀的 **containerruntime** 机器配置，则下一个 **containerruntime** 机器配置会附加 **-3**。

如果要删除机器配置，以相反的顺序删除它们，以避免超过限制。例如，在删除 **containerruntime-2** 机器配置前删除 **containerruntime-3** 机器配置。

注意

如果您的机器配置带有 **containerruntime-9** 后缀并创建 another **ContainerRuntimeConfig** CR，则不会创建新的机器配置，即使少于 10 个 **containerruntime** 机器配置。

多个 ContainerRuntimeConfig CR 示例

```
$ oc get ctrcfg
```

输**出示例**

```
NAME         AGE
ctr-overlay  15m
ctr-level    5m45s
```

多个 containerruntime 相关系统配置示例

```
$ cat /proc/1/status | grep Cap
$ capsh --decode=<decode_CapBnd_value> 1
```

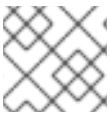
1 将 **<decode_CapBnd_value** > 替换为您要解码的特定值。

第 8 章 安装后集群任务

安装 OpenShift Container Platform 后，您可以按照自己的要求进一步扩展和自定义集群。

8.1. 可用的集群自定义

大多数集群配置和自定义在 OpenShift Container Platform 集群部署后完成。有若干*配置资源*可用。



注意

如果在 IBM Z® 上安装集群，则不是所有功能都可用。

您可以修改配置资源来配置集群的主要功能，如镜像 registry、网络配置、镜像构建操作以及用户身份供应商。

如需设置这些资源的当前信息，请使用 **oc explain** 命令，如 **oc explain builds --api-version=config.openshift.io/v1**

8.1.1. 集群配置资源

所有集群配置资源都作用于全局范围（而非命名空间），且命名为 **cluster**。

资源名称	描述
apiserver.config.openshift.io	提供 API 服务器配置，如证书和证书颁发机构。
authentication.config.openshift.io	控制集群的身份提供程序和身份验证配置。
build.config.openshift.io	控制集群中所有构建的默认和强制配置。
console.config.openshift.io	配置 Web 控制台界面的行为，包括注销行为。
featuregate.config.openshift.io	启用 FeatureGates，以便您能使用技术预览功能。
image.config.openshift.io	配置应如何对待特定的镜像 registry（允许、禁用、不安全、CA 详情）。
ingress.config.openshift.io	与路由相关的配置详情,如路由 的默认域。
oauth.config.openshift.io	配置用户身份供应商，以及与内部 OAuth 服务器流程相关的其他行为。

资源名称	描述
project.config.openshift.io	配置项目的创建方式，包括项目模板。
proxy.config.openshift.io	定义需要外部网络访问的组件要使用的代理。注意：目前不是所有组件都会消耗这个值。
scheduler.config.openshift.io	配置 调度程序行为，如配置集和默认节点选择器。

8.1.2. Operator 配置资源

这些配置资源是集群范围的实例，即 **cluster**，控制归特定 Operator 所有的特定组件的行为。

资源名称	描述
consoles.operator.openshift.io	控制控制台外观，如品牌定制
config.imageregistry.operator.openshift.io	配置 OpenShift 镜像 registry 设置，如公共路由、日志级别、代理设置、资源约束、副本数和存储类型。
config.samples.operator.openshift.io	配置 Samples Operator，以控制在集群上安装哪些镜像流和模板示例。

8.1.3. 其他配置资源

这些配置资源代表一个特定组件的单一实例。在有些情况下，您可以通过创建多个资源实例来请求多个实例。在其他情况下，Operator 只消耗指定命名空间中的特定资源实例名称。如需有关如何和何时创建其他资源实例的详情，请参考具体组件的文档。

资源名称	实例名称	命名空间	描述
alertmanager.monitoring.coreos.com	**main**	**openshift-monitoring**	控制 Alertmanager 部署参数。
ingresscontroller.operator.openshift.io	**default**	**openshift-ingress-operator**	配置 Ingress Operator 行为，如域、副本数、证书和控制器放置。

8.1.4. 信息资源

可以使用这些资源检索集群信息。有些配置可能需要您直接编辑这些资源。

资源名称	实例名称	描述
clusterversion.config.openshift.io	**version**	在 OpenShift Container Platform 4.15 中，不得自定义生产集群的 **ClusterVersion** 资源。相反，请按照流程更新集群。
dns.config.openshift.io	**cluster**	无法修改集群的 DNS 设置。您可以查看 DNS Operator 状态。
infrastructure.config.openshift.io	**cluster**	允许集群与其云供应商交互的配置详情。
network.config.openshift.io	**cluster**	无法在安装后修改集群网络。要自定义您的网络，请遵循相关的流程在安装过程中自定义网络。

8.2. 更新全局集群 PULL SECRET

您可以通过替换当前的 pull secret 或附加新的 pull secret 来更新集群的全局 pull secret。

当用户使用单独的 registry 存储镜像而不使用安装过程中的 registry时，需要这个过程。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 可选：要将新的 pull secret 附加到现有 pull secret 中，请完成以下步骤：
   a. 输入以下命令下载 pull secret：

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> 1
```

   1 提供 pull secret 文件的路径。

   b. 输入以下命令来添加新 pull secret：

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

   1 提供新的 registry。您可以在同一个 registry 中包含多个软件仓库，例如：**--registry="<registry/my-namespace/my-repository>"**。

   2 提供新 registry 的凭据。

3 提供 pull secret 文件的路径。

另外，您可以对 pull secret 文件执行手动更新。

2. 输入以下命令为您的集群更新全局 pull secret：

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

1 提供新 pull secret 文件的路径。

该更新将推广至所有节点，可能需要一些时间，具体取决于集群大小。

注意

从 OpenShift Container Platform 4.7.4 开始，对全局 pull secret 的更改不再触发节点排空或重启。

8.3. 添加 WORKER 节点

部署 OpenShift Container Platform 集群后，您可以添加 worker 节点来扩展集群资源。您可以根据安装方法和集群的环境，添加 worker 节点的不同方法。

8.3.1. 在安装程序置备的基础架构集群中添加 worker 节点

对于安装程序置备的基础架构集群，您可以手动或自动扩展 **MachineSet** 对象以匹配可用的裸机主机数量。

要添加裸机主机，您必须配置所有网络先决条件，配置关联的 **baremetalhost** 对象，然后为集群置备 worker 节点。您可以手动添加裸机主机，或使用 Web 控制台。

- 使用 Web 控制台添加 worker 节点
- 在 web 控制台中使用 YAML 添加 worker 节点
- 手动将 worker 节点添加到安装程序置备的基础架构集群中

8.3.2. 在用户置备的基础架构集群中添加 worker 节点

对于用户置备的基础架构集群，您可以使用 RHEL 或 RHCOS ISO 镜像添加 worker 节点，并使用集群 Ignition 配置文件将其连接到集群。对于 RHEL worker 节点，以下示例使用 Ansible playbook 在集群中添加 worker 节点。对于 RHCOS worker 节点，以下示例使用 ISO 镜像和网络引导来在集群中添加 worker 节点。

- 将 RHCOS worker 节点添加到用户置备的基础架构集群中
- 将 RHEL worker 节点添加到用户置备的基础架构集群中

8.3.3. 将 worker 节点添加到由 Assisted Installer 管理的集群

对于由 Assisted Installer 管理的集群，您可以使用 Red Hat OpenShift Cluster Manager 控制台（辅助安装程序 REST API）添加 worker 节点，也可以使用 ISO 镜像和集群 Ignition 配置文件手动添加 worker 节点。

- 使用 OpenShift Cluster Manager 添加 worker 节点
- 使用 Assisted Installer REST API 添加 worker 节点
- 手动将 worker 节点添加到 SNO 集群

8.3.4. 将 worker 节点添加到由 Kubernetes 的多集群引擎管理的集群

对于由 Kubernetes 多集群引擎管理的集群，您可以使用专用多集群引擎控制台添加 worker 节点。

- 使用控制台创建集群

8.4. 调整 WORKER 节点

如果您在部署过程中错误地定义了 worker 节点的大小，请通过创建一个或多个新计算机器集来调整它们，扩展它们，然后扩展原始的计算机器集，然后再删除它们。

8.4.1. 了解计算机器集和机器配置池之间的区别

MachineSet 对象描述了与云或机器供应商相关的 OpenShift Container Platform 节点。

MachineConfigPool 对象允许 **MachineConfigController** 组件在升级过程中定义并提供机器的状态。

MachineConfigPool 对象允许用户配置如何将升级应用到机器配置池中的 OpenShift Container Platform 节点。

NodeSelector 对象可以被一个到 **MachineSet** 对象的引用替换。

8.4.2. 手动扩展计算机器集

要在计算机器集中添加或删除机器实例，您可以手动扩展计算机器集。

这个指南与全自动的、安装程序置备的基础架构安装相关。自定义的、用户置备的基础架构安装没有计算机器集。

先决条件

- 安装 OpenShift Container Platform 集群和 **oc** 命令行。
- 以具有 **cluster-admin** 权限的用户身份登录 **oc**。

流程

1. 运行以下命令，查看集群中的计算机器：

```
$ oc get machinesets -n openshift-machine-api
```

   计算机器集以 **<clusterid>-worker-<aws-region-az>** 的形式列出。

2. 运行以下命令，查看集群中的计算机器：

```
$ oc get machine -n openshift-machine-api
```

3. 运行以下命令，在要删除的计算机器上设置注解：

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/delete-machine="true"
```

4. 运行以下命令来扩展计算机器集：

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

或者：

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

提示

您还可以应用以下 YAML 来扩展计算机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

您可以扩展或缩减计算机器。需要过几分钟以后新机器才可用。

重要

默认情况下，机器控制器会尝试排空在机器上运行的节点，直到成功为止。在某些情况下，如错误配置了 pod 中断预算，排空操作可能无法成功。如果排空操作失败，机器控制器无法继续删除机器。

您可以通过在特定机器上注解 **machine.openshift.io/exclude-node-draining** 来跳过排空节点。

验证

- 运行以下命令，验证删除所需的机器：

```
$ oc get machines
```

8.4.3. 计算机器集删除策略

Random、**Newest** 和 **Oldest** 是三个支持的删除选项。默认值为 **Random**，表示在扩展计算机器时随机选择并删除机器。通过修改特定的计算机器集，可以根据用例设置删除策略：

```
spec:
  deletePolicy: <delete_policy>
  replicas: <desired_replica_count>
```

无论删除策略是什么，都可通过在相关机器上添加 **machine.openshift.io/delete-machine=true** 注解来指定机器删除的优先级。

重要

默认情况下，OpenShift Container Platform 路由器 Pod 部署在 worker 上。由于路由器需要访问某些集群资源（包括 Web 控制台），除非先重新放置了路由器 Pod，否则请不要将 worker 计算机器集扩展为 **0**。

注意

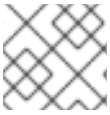
对于需要特定节点运行的用例，可以使用自定义计算机器集，在 worker 计算机器集缩减时，控制器会忽略这些服务。这可防止服务被中断。

8.4.4. 创建默认的集群范围节点选择器

您可以组合使用 pod 上的默认集群范围节点选择器和节点上的标签，将集群中创建的所有 pod 限制到特定节点。

使用集群范围节点选择器时，如果您在集群中创建 pod,OpenShift Container Platform 会将默认节点选择器添加到 pod，并将该 pod 调度到具有匹配标签的节点。

您可以通过编辑调度程序 Operator 自定义资源（CR）来配置集群范围节点选择器。您可向节点、计算机器集或机器配置添加标签。将标签添加到计算机器集可确保节点或机器停机时，新节点具有该标签。如果节点或机器停机，添加到节点或机器配置的标签不会保留。



注意

您可以向 pod 添加额外的键/值对。但是，您无法为一个默认的键添加不同的值。

流程

添加默认的集群范围节点选择器：

1. 编辑调度程序 Operator CR 以添加默认的集群范围节点选择器：

```
$ oc edit scheduler cluster
```

使用节点选择器的调度程序 Operator CR 示例

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: type=user-node,region=east (1)
  mastersSchedulable: false
```

(1) 使用适当的 **<key>:<value>** 对添加节点选择器。

完成此更改后，请等待重新部署 **openshift-kube-apiserver** 项目中的 pod。这可能需要几分钟。只有重新部署 pod 后，默认的集群范围节点选择器才会生效。

2. 使用计算机器集或直接编辑节点，为节点添加标签：

   - 在创建节点时，使用计算机器集向由计算机器设置管理的节点添加标签：

a. 运行以下命令，将标签添加到 **MachineSet** 对象中：

```
$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>"="
<value>","<key>"="<value>"}}]'  -n openshift-machine-api (1)
```

(1) 为每个标识添加 **<key>/<value>** 对。

例如：

```
$ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
node","region":"east"}}]'  -n openshift-machine-api
```

提示

您还可以应用以下 YAML 来向计算机器集中添加标签：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b. 使用 **oc edit** 命令验证标签是否已添加到 **MachineSet** 对象中：
例如：

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api
```

MachineSet 对象示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
  ...
spec:
  ...
  template:
    metadata:
  ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
  ...
```

c. 通过缩减至 **0** 并扩展节点来重新部署与该计算机器集关联的节点：
例如：

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

d. 当节点就绪并可用时，使用 **oc get** 命令验证该标签是否已添加到节点：

```
$ oc get nodes -l <key>=<value>
```

例如：

```
$ oc get nodes -l type=user-node
```

输**出示例**

```
NAME                                         STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp   Ready    worker   61s   v1.28.5
```

- 直接向节点添加标签：

a. 为节点编辑 **Node** 对象：

```
$ oc label nodes <name> <key>=<value>
```

例如，若要为以下节点添加标签：

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node region=east
```

提示

您还可以应用以下 YAML 来向节点添加标签：

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

b. 使用 **oc get** 命令验证标签是否已添加到节点：

```
$ oc get nodes -l <key>=<value>,<key>=<value>
```

例如：

```
$ oc get nodes -l type=user-node,region=east
```

输**出示例**

```
NAME                                         STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.28.5
```

8.5. 使用 WORKER 延迟配置集提高高延迟环境中的集群稳定性

如果集群管理员为平台验证执行了延迟测试，他们可以发现需要调整集群的操作，以确保高延迟的情况的稳定性。集群管理员只需要更改一个参数，该参数记录在一个文件中，它控制了 Supervisory 进程读取状态并解释集群的运行状况的四个参数。仅更改一个参数可以以方便、可支持的方式提供集群调整。

Kubelet 进程提供监控集群运行状况的起点。**Kubelet** 为 OpenShift Container Platform 集群中的所有节点设置状态值。Kubernetes Controller Manager (**kube controller**) 默认每 10 秒读取状态值。如果 **kube 控制器**无法读取节点状态值，它会在配置的时间后丢失与该节点联系。默认行为是：

1. control plane 上的节点控制器将节点健康状况更新为 **Unhealthy**，并奖节点 **Ready** 的条件标记为 'Unknown'。
2. 因此，调度程序会停止将 pod 调度到该节点。
3. Node Lifecycle Controller 添加了一个 **node.kubernetes.io/unreachable** 污点，对节点具有 **NoExecute** 效果，默认在五分钟后调度节点上的任何 pod 进行驱除。

如果您的网络容易出现延迟问题，尤其是在网络边缘中有节点时，此行为可能会造成问题。在某些情况下，Kubernetes Controller Manager 可能会因为网络延迟而从健康的节点接收更新。**Kubelet** 会从节点中驱除 pod，即使节点处于健康状态。

要避免这个问题，您可以使用 *worker 延迟配置集*调整 **kubelet** 和 Kubernetes Controller Manager 在执行操作前等待状态更新的频率。如果在控制平面和 worker 节点间存在网络延迟，worker 节点没有处于最近状态，这个调整有助于集群可以正常工作。

这些 worker 延迟配置集包含预定义的三组参数，它们带有经过仔细调优的值，以控制集群对增加的延迟进行适当地响应。用户不需要手动进行实验以查找最佳值。

您可在安装集群时配置 worker 延迟配置集，或当您发现集群网络中的延迟增加时。

8.5.1. 了解 worker 延迟配置集

worker 延迟配置集带有四个不同的、包括经过仔细调优的参数的类别。实现这些值的四个参数是 **node-status-update-frequency**、**node-monitor-grace-period**、**default-not-ready-toleration-seconds** 和 **default-unreachable-toleration-seconds**。这些参数可让您使用这些值来控制集群对延迟问题的响应，而无需手动确定最佳值。

重要

不支持手动设置这些参数。参数设置不正确会影响集群的稳定性。

所有 worker 延迟配置集配置以下参数：

node-status-update-frequency

指定 kubelet 将节点状态发布到 API 服务器的频率。

node-monitor-grace-period

指定 Kubernetes Controller Manager 在节点不健康前等待更新的时间（以秒为单位），并将 **node.kubernetes.io/not-ready** 或 **node.kubernetes.io/unreachable** 污点添加到节点。

default-not-ready-toleration-seconds

指定在标记节点不健康后，Kube API Server Operator 在从该节点驱除 pod 前等待的时间（以秒为单位）。

default-unreachable-toleration-seconds

指定在节点无法访问后，Kube API Server Operator 在从该节点驱除 pod 前等待的时间（以秒为单位）。

以下 Operator 监控 worker 延迟配置集的更改并相应地响应：

- Machine Config Operator (MCO) 更新 worker 节点上的 **node-status-update-frequency** 参数。
- Kubernetes Controller Manager 更新 control plane 节点上的 **node-monitor-grace-period** 参数。
- Kubernetes API Server Operator 更新 control plane 节点上的 **default-not-ready-toleration-seconds** 和 **default-unreachable-toleration-seconds** 参数。

虽然默认配置在大多数情况下可以正常工作，但 OpenShift Container Platform 会为网络遇到比通常更高的延迟的情况提供两个其他 worker 延迟配置集。以下部分描述了三个 worker 延迟配置集：

默认 worker 延迟配置集

使用 **Default** 配置集时，每个 **Kubelet** 每 10 秒更新其状态(**node-status-update-frequency**)。**Kube Controller Manager** 每 5 秒检查 **Kubelet** 的状态(**node-monitor-grace-period**)。
在认为 **Kubelet** 不健康前，Kubernetes Controller Manager 会等待 40 秒以获取来自 **Kubelet** 的状态更新。如果没有可用于 Kubernetes Controller Manager 的使用状态，它会使用 **node.kubernetes.io/not-ready** 或 **node.kubernetes.io/unreachable** 污点标记节点，并驱除该节点上的 pod。

如果该节点上的 pod 具有 **NoExecute** 污点，则 pod 会根据 **tolerationSeconds** 运行。如果 pod 没有污点，它将在 300 秒内被驱除(**default-not-ready-toleration-seconds** 和 **Kube API Server**的 **default-unreachable-toleration-seconds** 设置)。

profile	组件	参数	值
Default（默认）	kubelet	**node-status-update-frequency**	10s
	kubelet Controller Manager	**node-monitor-grace-period**	40s
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

中型 worker 延迟配置集

如果网络延迟比通常稍高，则使用 **MediumUpdateAverageReaction** 配置集。**MediumUpdateAverageReaction** 配置集减少了 kubelet 更新频率为 20 秒，并将 Kubernetes Controller Manager 等待这些更新的时间更改为 2 分钟。该节点上的 pod 驱除周期会减少到 60 秒。如果 pod 具有 **tolerationSeconds** 参数，则驱除会等待该参数指定的周期。

Kubernetes Controller Manager 会先等待 2 分钟时间，才会认为节点不健康。另一分钟后，驱除过程会启动。

profile	组件	参数	值
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	kubelet Controller Manager	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

低 worker 延迟配置集

如果网络延迟非常高，请使用 **LowUpdateSlowReaction** 配置集。**LowUpdateSlowReaction** 配置集将 kubelet 更新频率减少为 1 分钟，并将 Kubernetes Controller Manager 等待这些更新的时间更改为 5 分钟。该节点上的 pod 驱除周期会减少到 60 秒。如果 pod 具有 **tolerationSeconds** 参数，则驱除会等待该参数指定的周期。

Kubernetes Controller Manager 在认为节点不健康前会等待 5 分钟。另一分钟后，驱除过程会启动。

profile	组件	参数	值
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	kubelet Controller Manager	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s

profile	组件	参数	值
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

8.5.2. 使用和更改 worker 延迟配置集

要更改 worker 延迟配置集以处理网络延迟，请编辑 **node.config** 对象以添加配置集的名称。当延迟增加或减少时，您可以随时更改配置集。

您必须一次移动一个 worker 延迟配置集。例如，您无法直接从 **Default** 配置集移到 **LowUpdateSlowReaction** worker 延迟配置集。您必须首先从 **Default** worker 延迟配置集移到 **MediumUpdateAverageReaction** 配置集，然后再移到 **LowUpdateSlowReaction**。同样，当返回到 **Default** 配置集时，您必须首先从低配置集移到中配置集，然后移到 **Default**。

注意

您还可以在安装 OpenShift Container Platform 集群时配置 worker 延迟配置集。

流程

将默认的 worker 延迟配置集改为：

1. 中 worke worker 延迟配置集：
   a. 编辑 **node.config** 对象：

```
$ oc edit nodes.config/cluster
```

   b. 添加 **spec.workerLatencyProfile: MediumUpdateAverageReaction** ：

   node.config 对象示例

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
```

```
spec:
  workerLatencyProfile: MediumUpdateAverageReaction 1

# ...
```

1 指定中 worker 延迟策略。

随着更改被应用，每个 worker 节点上的调度都会被禁用。

2. 可选：改为低 worker 延迟配置集：

   a. 编辑 **node.config** 对象：

   ```
   $ oc edit nodes.config/cluster
   ```

   b. 将 **spec.workerLatencyProfile** 值更改为 **LowUpdateSlowReaction** ：

   node.config 对象示例

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction 1

   # ...
   ```

   1 指定使用低 worker 延迟策略。

随着更改被应用，每个 worker 节点上的调度都会被禁用。

验证

- 当所有节点都返回到 **Ready** 条件时，您可以使用以下命令查看 Kubernetes Controller Manager 以确保应用它：

  ```
  $ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
  ```

输出示例

```
# ...
    - lastTransitionTime: "2022-07-11T19:47:10Z"
      reason: ProfileUpdated
      status: "False"
      type: WorkerLatencyProfileProgressing
    - lastTransitionTime: "2022-07-11T19:47:10Z" (1)
      message: all static pod revision(s) have updated latency profile
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
# ...
```

(1) 指定配置集被应用并激活。

要将中配置集改为默认，或将默认改为中，编辑 **node.config** 对象，并将 **spec.workerLatencyProfile** 参数设置为适当的值。

8.6. 管理 CONTROL PLANE 机器

control plane 机器集为 control plane 机器提供管理功能，与为计算机器提供的计算机器集类似。集群上的 control plane 机器集的可用性和初始状态取决于您的云供应商和您安装的 OpenShift Container Platform 版本。如需更多信息，请参阅开始使用 control plane 机器集。

8.7. 为生产环境创建基础架构机器集

您可以创建一个计算机器集来创建仅托管基础架构组件的机器，如默认路由器、集成的容器镜像 registry 和组件用于集群指标和监控。这些基础架构机器不会被计算为运行环境所需的订阅总数。

在生产部署中，建议您部署至少三个计算机器集来容纳基础架构组件。OpenShift Logging 和 Red Hat OpenShift Service Mesh 部署 Elasticsearch，这需要三个实例安装到不同的节点上。这些节点都可以部署到不同的可用区以实现高可用性。类似的配置需要三个不同的计算机器集，每个可用区都有一个。在没有多个可用区的全局 Azure 区域，您可以使用可用性集来确保高可用性。

有关基础架构节点以及可在基础架构节点上运行的组件的详情，请参考 创建基础架构机器集。

要创建基础架构节点，您可以使用机器集，为节点分配标签，或使用机器配置池。

有关可用于这些流程的机器集示例，请参阅为不同的云创建机器集。

将特定节点选择器应用到所有基础架构组件会导致 OpenShift Container Platform 使用 该标签将这些工作负载调度到具有该标签的节点。

8.7.1. 创建计算机器集

除了安装程序创建的计算机器集外，您还可以创建自己的来动态管理您选择的特定工作负载的机器计算资源。

先决条件

- 部署一个 OpenShift Container Platform 集群。
- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 权限的用户身份登录 **oc**。

流程

1. 创建一个包含计算机器集自定义资源（CR）示例的新 YAML 文件，并将其命名为 **<file_name>.yaml**。
   确保设置 **<clusterID>** 和 **<role>** 参数值。
2. 可选：如果您不确定要为特定字段设置哪个值，您可以从集群中检查现有计算机器集：
   a. 要列出集群中的计算机器集，请运行以下命令：

```
$ oc get machinesets -n openshift-machine-api
```

输**出示例**

```
NAME                          DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d   0         0                             55m
agl030519-vplxk-worker-us-east-1e   0         0                             55m
agl030519-vplxk-worker-us-east-1f   0         0                             55m
```

   b. 要查看特定计算机器集自定义资源 (CR) 的值，请运行以下命令：

```
$ oc get machineset <machineset_name> \
  -n openshift-machine-api -o yaml
```

输**出示例**

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-<role> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
```

```
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machine-role: <role>
    machine.openshift.io/cluster-api-machine-type: <role>
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
spec:
  providerSpec: (3)
    ...
```

(1) 集群基础架构 ID。

(2) 默认节点标签。

注意

对于具有用户置备的基础架构的集群，计算机器集只能创建 **worker** 和 **infra** 类型机器。

(3) 计算机器设置 CR 的 **<providerSpec>** 部分中的值是特定于平台的。有关 CR 中的 **<providerSpec>** 参数的更多信息，请参阅您的供应商计算机器设置 CR 配置示例。

3. 运行以下命令来创建 **MachineSet** CR：

```
$ oc create -f <file_name>.yaml
```

验证

- 运行以下命令，查看计算机器集列表：

```
$ oc get machineset -n openshift-machine-api
```

输出示例

```
NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-infra-us-east-1a    1         1         1       1           11m
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d   0         0                             55m
agl030519-vplxk-worker-us-east-1e   0         0                             55m
agl030519-vplxk-worker-us-east-1f   0         0                             55m
```

当新的计算机器集可用时，**DESIRED** 和 **CURRENT** 的值会匹配。如果 compute 机器集不可用，请等待几分钟，然后再次运行命令。

8.7.2. 创建基础架构节点

重要

请参阅为安装程序置备的基础架构环境创建基础架构机器集，或为其 control plane 节点由机器 API 管理的任何集群创建基础架构机器集。

集群的基础架构系统（也称为 **infra 节点**）的要求已被置备。安装程序只为 control plane 和 worker 节点提供置备。Worker 节点可以通过标记来指定为基础架构节点或应用程序（也称为 **app** ）。

流程

1. 向您要充当应用程序节点的 worker 节点添加标签：

```
$ oc label node <node-name> node-role.kubernetes.io/app=""
```

2. 向您要充当基础架构节点的 worker 节点添加标签：

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

3. 检查相关节点现在是否具有 **infra** 角色或 **app** 角色：

```
$ oc get nodes
```

4. 创建默认的集群范围节点选择器。默认节点选择器应用到在所有命名空间中创建的 pod。这会创建一个与 pod 上任何现有节点选择器交集的交集，这会额外限制 pod 的选择器。

重要

如果默认节点选择器键与 pod 标签的键冲突，则不会应用默认节点选择器。

但是，不要设置可能会导致 pod 变得不可调度的默认节点选择器。例如，当 pod 的标签被设置为不同的节点角色（如 **node-role.kubernetes.io/infra=""**）时，将默认节点选择器设置为特定的节点角色（如 **node-role.kubernetes.io/master=""**）可能会导致 pod 无法调度。因此，将默认节点选择器设置为特定节点角色时要小心。

您还可以使用项目节点选择器来避免集群范围节点选择器键冲突。

a. 编辑 **Scheduler** 对象：

```
$ oc edit scheduler cluster
```

b. 使用适当的节点选择器添加 **defaultNodeSelector** 字段：

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec:
  defaultNodeSelector: node-role.kubernetes.io/infra="" 1
# ...
```

1 这个示例节点选择器默认在基础架构节点上部署 pod。

c. 保存文件以使改变生效。

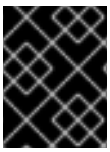
现在，您可以将基础架构资源移到新标记的 **infra** 节点。

其他资源

- 有关如何配置项目节点选择器以避免集群范围节点选择器冲突的详情，请参考项目节点选择器。

8.7.3. 为基础架构机器创建机器配置池

如果需要基础架构机器具有专用配置，则必须创建一个 infra 池。



重要

在创建一个默认自定义集群配置池时，如果它们引用同一文件或单元，则创建的自定义机器配置池会覆盖默认的 worker 池配置。

流程

1. 向您要分配为带有特定标签的 infra 节点的节点添加标签：

```
$ oc label node <node_name> <label>
```

```
$ oc label node ci-ln-n8mqwr2-f76d1-xscn2-worker-c-6fmtx node-role.kubernetes.io/infra=
```

2. 创建包含 worker 角色和自定义角色作为机器配置选择器的机器配置池：

```
$ cat infra.mcp.yaml
```

输出示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]} (1)
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: "" (2)
```

(1) 添加 worker 角色和自定义角色。

(2) 将您添加的标签作为 **nodeSelector** 添加到节点。

注意

自定义机器配置池从 worker 池中继承机器配置。自定义池使用任何针对 worker 池的机器配置，但增加了部署仅针对自定义池的更改的功能。由于自定义池从 worker 池中继承资源，对 worker 池的任何更改也会影响自定义池。

3. 具有 YAML 文件后，您可以创建机器配置池：

```
$ oc create -f infra.mcp.yaml
```

4. 检查机器配置，以确保基础架构配置成功：

```
$ oc get machineconfig
```

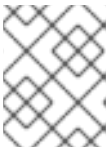
输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                          365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
00-worker                                          365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
01-master-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
01-master-kubelet                                  365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
01-worker-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
01-worker-kubelet                                  365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
99-master-ssh                                                 3.2.0             31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
99-worker-ssh                                                 3.2.0             31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             23m
rendered-master-072d4b2da7f88162636902b074e9e28e
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-master-3e88ec72aed3886dec061df60d16d1af
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0             31d
rendered-master-419bee7de96134963a15fdf9dd473b25
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             17d
rendered-master-53f5c91c7661708adce18739cc0f40fb
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             13d
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             7d3h
rendered-master-dc7f874ec77fc4b969674204332da037
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-2640531be11ba43c61d72e82dc634ce6
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             7d3h
rendered-worker-4f110718fe88e5f349987854a1147755
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             17d
rendered-worker-afc758e194d6188677eb837842d3b379
```

```
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0          31d
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          13d
```

您应该会看到一个新的机器配置，带有 **rendered-infra-*** 前缀。

5. 可选：要部署对自定义池的更改，请创建一个机器配置，该配置使用自定义池名称作为标签，如本例中的 **infra**。请注意，这不是必须的，在此包括仅用于指示目的。这样，您可以只应用特定于 infra 节点的任何自定义配置。



注意

创建新机器配置池后，MCO 会为该池生成一个新的呈现配置，以及该池重启的关联节点以应用新配置。

a. 创建机器配置：

```
$ cat infra.mc.yaml
```

输出示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 51-infra
  labels:
    machineconfiguration.openshift.io/role: infra (1)
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - path: /etc/infratest
        mode: 0644
        contents:
          source: data:,infra
```

(1) 将您添加的标签作为 **nodeSelector** 添加到节点。

b. 将机器配置应用到 infra-labeled 节点：

```
$ oc create -f infra.mc.yaml
```

6. 确认您的新机器配置池可用：

```
$ oc get mcp
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
```

```
DEGRADEDMACHINECOUNT   AGE
infra    rendered-infra-60e35c2e99f42d976e084fa94da4d0fc    True      False      False      1
1                1                0                   4m20s
master   rendered-master-9360fdb895d4c131c7c4bebbae099c90   True      False      False
3          3                3                   0                   91m
worker   rendered-worker-60e35c2e99f42d976e084fa94da4d0fc   True      False      False
2          2                2                   0                   91m
```

在本例中，worker 节点被改为一个 infra 节点。

其他资源

- 如需有关在自定义池中分组 infra 机器的更多信息，请参阅使用机器配置池进行节点配置管理。

8.8. 为基础架构节点分配机器设置资源

在创建了基础架构机器集后，**worker** 和 **infra** 角色将应用到新的 infra 节点。具有 **infra** 角色的节点不会计入运行环境所需的订阅总数中，即使也应用了 **worker** 角色。

但是，当为 infra 节点分配 worker 角色时，用户工作负载可能会意外地分配给 infra 节点。要避免这种情况，您可以将污点应用到 infra 节点，并为您要控制的 pod 应用容限。

8.8.1. 使用污点和容限绑定基础架构节点工作负载

如果您有一个分配了 **infra** 和 **worker** 角色的 infra 节点，您必须配置该节点，以便不为其分配用户工作负载。

重要

建议您保留为 infra 节点创建的双 **infra,worker** 标签，并使用污点和容限来管理用户工作负载调度到的节点。如果从节点中删除 **worker** 标签，您必须创建一个自定义池来管理它。没有自定义池的 MCO 不能识别具有 **master** 或 **worker** 以外的标签的节点。如果不存在选择自定义标签的自定义池，维护 **worker** 标签可允许默认 worker 机器配置池管理节点。**infra** 标签与集群通信，它不计算订阅总数。

先决条件

- 在 OpenShift Container Platform 集群中配置额外的 **MachineSet** 对象。

流程

1. 向 infra 节点添加污点以防止在其上调度用户工作负载：

   a. 确定节点是否具有污点：

   ```
   $ oc describe nodes <node_name>
   ```

   输出示例

   ```
   oc describe node ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Name:               ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Roles:              worker
   ```

```
 ...
Taints:             node-role.kubernetes.io/infra:NoSchedule
 ...
```

本例显示节点具有污点。您可以在下一步中继续向 pod 添加容限。

b. 如果您还没有配置污点以防止在其上调度用户工作负载：

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

例如：

```
$ oc adm taint nodes node1 node-role.kubernetes.io/infra=reserved:NoSchedule
```

提示

您还可以应用以下 YAML 来添加污点：

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      effect: NoSchedule
      value: reserved
  ...
```

本例在 **node1** 上放置一个键为 **node-role.kubernetes.io/infra** 的污点，污点是 **NoSchedule**。具有 **NoSchedule effect** 的节点仅调度容许该污点的 pod，但允许现有 pod 继续调度到该节点上。

注意

如果使用 descheduler，则违反了节点污点的 pod 可能会从集群驱除。

c. 添加带有 NoExecute Effect 的污点，以及上述带有 NoSchedule Effect 的污点：

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

例如：

```
$ oc adm taint nodes node1 node-role.kubernetes.io/infra=reserved:NoExecute
```

提示

您还可以应用以下 YAML 来添加污点：

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      effect: NoExecute
      value: reserved
  ...
```

本例在 **node1** 上放置一个键为 **node-role.kubernetes.io/infra** 的污点，污点是 **NoExecute**。带有 **NoExecute** 效果的节点仅调度容许该污点的 pod。该效果将从没有匹配容限的节点中删除任何现有 pod。

2. 为要在 infra 节点上调度的 pod 配置添加容限，如路由器、registry 和监控工作负载。在 **Pod** 对象规格中添加以下代码：

```
tolerations:
  - effect: NoSchedule (1)
    key: node-role.kubernetes.io/infra (2)
    value: reserved (3)
  - effect: NoExecute (4)
    key: node-role.kubernetes.io/infra (5)
    operator: Exists (6)
    value: reserved (7)
```

(1) 指定添加到节点的效果。

(2) 指定添加到节点的键。

(3) 指定添加到节点的键值对污点的值。

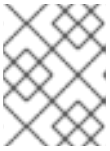
(4) 指定添加到节点的效果。

(5) 指定添加到节点的键。

(6) 指定 **Exists** Operator，以要求节点上存在一个带有键为 **node-role.kubernetes.io/infra** 的污点。

(7) 指定添加到节点的键值对污点的值。

此容忍度与 **oc adm taint** 命令创建的污点匹配。具有此容忍度的 pod 可以调度到 infra 节点上。



注意

并不总是能够将通过 OLM 安装的 Operator 的 Pod 移到 infra 节点。移动 Operator Pod 的能力取决于每个 Operator 的配置。

3. 使用调度程序将 pod 调度到 infra 节点。详情请参阅*控制节点上的 pod 放置*的文档。

其他资源

- 如需了解有关将 pod 调度到节点的信息，请参阅使用调度程序控制 pod 放置。

8.9. 将资源移到基础架构机器集

默认情况下，您的集群中已部署了某些基础架构资源。您可将它们移至您创建的基础架构机器集。

8.9.1. 移动路由器

您可以将路由器 Pod 部署到不同的计算机器集中。默认情况下，pod 部署到 worker 节点。

先决条件

- 在 OpenShift Container Platform 集群中配置额外的计算机器集。

流程

1. 查看路由器 Operator 的 **IngressController** 自定义资源：

```
$ oc get ingresscontroller default -n openshift-ingress-operator -o yaml
```

命令输出类似于以下文本：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: 2019-04-18T12:35:39Z
  finalizers:
  - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  generation: 1
  name: default
  namespace: openshift-ingress-operator
  resourceVersion: "11341"
  selfLink: /apis/operator.openshift.io/v1/namespaces/openshift-ingress-
operator/ingresscontrollers/default
  uid: 79509e05-61d6-11e9-bc55-02ce4781844a
spec: {}
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2019-04-18T12:36:15Z
    status: "True"
    type: Available
  domain: apps.<cluster>.example.com
```

```
  endpointPublishingStrategy:
    type: LoadBalancerService
  selector: ingresscontroller.operator.openshift.io/deployment-ingresscontroller=default
```

2. 编辑 **ingresscontroller** 资源，并更改 **nodeSelector** 以使用**infra** 标签：

```
$ oc edit ingresscontroller default -n openshift-ingress-operator
```

```
spec:
  nodePlacement:
    nodeSelector: 1
      matchLabels:
        node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

1 添加 **nodeSelector** 参数，并设为适用于您想要移动的组件的值。您可以根据为节点指定的值，按所示格式使用 **nodeSelector** 或使用 **<key>: <value>** 对。如果您在 infrasructure 节点中添加了污点，还要添加匹配的容限。

3. 确认路由器 Pod 在 **infra** 节点上运行。

   a. 查看路由器 Pod 列表，并记下正在运行的 Pod 的节点名称：

```
$ oc get pod -n openshift-ingress -o wide
```

   输出示例

```
NAME                              READY   STATUS        RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
router-default-86798b4b5d-bdlvd   1/1     Running       0          28s     10.130.2.4   ip-10-
0-217-226.ec2.internal   <none>           <none>
router-default-955d875f4-255g8    0/1     Terminating   0          19h     10.129.2.4   ip-10-
0-148-172.ec2.internal   <none>           <none>
```

   在本例中，正在运行的 Pod 位于 **ip-10-0-217-226.ec2.internal** 节点上。

   b. 查看正在运行的 Pod 的节点状态：

```
$ oc get node <node_name> 1
```

1 指定从 Pod 列表获得的 **<node_name>**。

   输出示例

```
NAME                           STATUS  ROLES        AGE   VERSION
ip-10-0-217-226.ec2.internal   Ready   infra,worker  17h   v1.28.5
```

-

由于角色列表包含 **infra**，因此 Pod 在正确的节点上运行。

8.9.2. 移动默认 registry

您需要配置 registry Operator，以便将其 Pod 部署到其他节点。

先决条件

- 在 OpenShift Container Platform 集群中配置额外的计算机器集。

流程

1. 查看 **config/instance** 对象：

```
$ oc get configs.imageregistry.operator.openshift.io/cluster -o yaml
```

输**出示例**

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12ffeee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
...
```

2. 编辑 **config/instance** 对象：

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
```

```
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
          topologyKey: kubernetes.io/hostname
        weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector: (1)
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
  - effect: NoExecute
    key: node-role.kubernetes.io/infra
    value: reserved
```

(1) 添加 **nodeSelector** 参数，并设为适用于您想要移动的组件的值。您可以根据为节点指定的值，按所示格式使用 **nodeSelector** 或使用 **<key>: <value>** 对。如果您在 infrasructure 节点中添加了污点，还要添加匹配的容限。

3. 验证 registry pod 已移至基础架构节点。

   a. 运行以下命令，以识别 registry pod 所在的节点：

   ```
   $ oc get pods -o wide -n openshift-image-registry
   ```

   b. 确认节点具有您指定的标签：

   ```
   $ oc describe node <node_name>
   ```

   查看命令输出，并确认 **node-role.kubernetes.io/infra** 列在 **LABELS** 列表中。

8.9.3. 移动监控解决方案

监控堆栈包含多个组件，包括 Prometheus、Thanos Querier 和 Alertmanager。Cluster Monitoring Operator 管理此堆栈。要将监控堆栈重新部署到基础架构节点，您可以创建并应用自定义配置映射。

流程

1. 编辑 **cluster-monitoring-config** 配置映射，并更改 **nodeSelector** 以使用 **infra** 标签：

```
$ oc edit configmap cluster-monitoring-config -n openshift-monitoring
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector: (1)
```

```
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  prometheusK8s:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  prometheusOperator:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  k8sPrometheusAdapter:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  kubeStateMetrics:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  telemeterClient:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
```

```
      value: reserved
      effect: NoExecute
  openshiftStateMetrics:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  thanosQuerier:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  monitoringPlugin:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
```

1 添加 **nodeSelector** 参数，并设为适用于您想要移动的组件的值。您可以根据为节点指定的值，按所示格式使用 **nodeSelector** 或使用 **<key>: <value>** 对。如果您在 infrasructure 节点中添加了污点，还要添加匹配的容限。

2. 观察监控 pod 移至新机器：

```
$ watch 'oc get pod -n openshift-monitoring -o wide'
```

3. 如果组件没有移到 **infra** 节点，请删除带有这个组件的 pod:

```
$ oc delete pod -n openshift-monitoring <pod>
```

已删除 pod 的组件在 **infra** 节点上重新创建。

8.9.4. 移动日志记录资源

有关移动日志记录资源的详情，请参考：

- 使用节点选择器移动日志记录资源
- 使用污点和容限来控制日志记录 pod 放置

8.10. 关于集群自动扩展

集群自动扩展会调整 OpenShift Container Platform 集群的大小，以满足其当前的部署需求。它使用 Kubernetes 样式的声明性参数来提供基础架构管理，而且这种管理不依赖于特定云提供商的对象。集群自动控制会在集群范围内有效，不与特定的命名空间相关联。

当由于资源不足而无法在任何当前 worker 节点上调度 pod 时，或者在需要另一个节点来满足部署需求时，集群自动扩展会增加集群的大小。集群自动扩展不会将集群资源增加到超过您指定的限制。

集群自动扩展会计算集群中所有节点上的内存、CPU 和 GPU，即使它不管理 control plane 节点。这些值不是单计算机导向型。它们是整个集群中所有资源的聚合。例如，如果您设置最大内存资源限制，集群自动扩展在计算当前内存用量时包括集群中的所有节点。然后，该计算用于确定集群自动扩展是否具有添加更多 worker 资源的容量。

重要

确保您所创建的 **ClusterAutoscaler** 资源定义中的 **maxNodesTotal** 值足够大，足以满足计算集群中可能的机器总数。此值必须包含 control plane 机器的数量以及可扩展至的机器数量。

每隔 10 秒，集群自动扩展会检查集群中不需要哪些节点，并移除它们。如果满足以下条件，集群自动扩展会考虑要删除的节点：

- 节点使用率低于集群的节点 *利用率级别* 阈值。节点使用率级别是请求的资源的总和，由分配给节点的资源划分。如果您没有在 **ClusterAutoscaler** 自定义资源中指定值，集群自动扩展会使用默认值 **0.5**，它对应于 50% 的利用率。
- 集群自动扩展可以将节点上运行的所有 pod 移到其他节点。Kubernetes 调度程序负责在节点上调度 pod。
- 集群自动扩展没有缩减禁用注解。

如果节点上存在以下类型的 pod，集群自动扩展不会删除该节点：

- 具有限制性 pod 中断预算（PDB）的 Pod。
- 默认不在节点上运行的 Kube 系统 Pod。
- 没有 PDB 或 PDB 限制性太强的 Kube 系统 pod。
- 不受控制器对象支持的 Pod,如部署、副本集或有状态集。
- 具有本地存储的 Pod。
- 因为缺乏资源、节点选择器或关联性不兼容或有匹配的反关联性等原因而无法移至其他位置的 Pod。
- 具有 **"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"** 注解的 Pod，除非同时也具有 **"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"** 注解。

例如，您可以将最大 CPU 限值设置为 64 个内核，并将集群自动扩展配置为每个创建具有 8 个内核的机器。如果您的集群从 30 个内核开始，集群自动扩展可最多添加具有 32 个内核的 4 个节点，共 62 个。

如果配置集群自动扩展，则需要额外的使用限制：

- 不要直接修改位于自动扩展节点组中的节点。同一节点组中的所有节点具有相同的容量和标签，并且运行相同的系统 Pod。
- 指定适合您的 Pod 的请求。
- 如果需要防止 Pod 被过快删除，请配置适当的 PDB。
- 确认您的云提供商配额足够大，能够支持您配置的最大节点池。
- 不要运行其他节点组自动扩展器，特别是云提供商提供的自动扩展器。

pod 横向自动扩展（HPA）和集群自动扩展以不同的方式修改集群资源。HPA 根据当前的 CPU 负载更改部署或副本集的副本数。如果负载增加，HPA 会创建新的副本，不论集群可用的资源量如何。如果没有足够的资源，集群自动扩展会添加资源，以便 HPA 创建的 pod 可以运行。如果负载减少，HPA 会停止一些副本。如果此操作导致某些节点利用率低下或完全为空，集群自动扩展会删除不必要的节点。

集群自动扩展会考虑 pod 优先级。如果集群没有足够的资源，则"Pod 优先级和抢占"功能可根据优先级调度 Pod，但集群自动扩展会确保集群具有运行所有 Pod 需要的资源。为满足这两个功能，集群自动扩展包含一个优先级截止函数。您可以使用此截止函数来调度"尽力而为"的 Pod，它们不会使集群自动扩展增加资源，而是仅在有可用备用资源时运行。

优先级低于截止值的 Pod 不会导致集群扩展或阻止集群缩减。系统不会添加新节点来运行 Pod，并且可能会删除运行这些 Pod 的节点来释放资源。

集群自动扩展支持在其上有机器 API 的平台。

8.10.1. 集群自动扩展资源定义

此 **ClusterAutoscaler** 资源定义显示了集群自动扩展的参数和示例值。

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10 (1)
  resourceLimits:
    maxNodesTotal: 24 (2)
    cores:
      min: 8 (3)
      max: 128 (4)
    memory:
      min: 4 (5)
      max: 256 (6)
    gpus:
      - type: nvidia.com/gpu (7)
        min: 0 (8)
        max: 16 (9)
      - type: amd.com/gpu
        min: 0
        max: 4
  logVerbosity: 4 (10)
  scaleDown: (11)
    enabled: true (12)
```

```
delayAfterAdd: 10m (13)
delayAfterDelete: 5m (14)
delayAfterFailure: 30s (15)
unneededTime: 5m (16)
utilizationThreshold: "0.4" (17)
```

(1) 指定 Pod 必须超过哪一优先级才能让机器自动扩展部署更多节点。输入一个 32 位整数值。**podPriorityThreshold** 值将与您分配给每个 Pod 的 **PriorityClass** 值进行比较。

(2) 指定要部署的最大节点数。这个值是集群中部署的机器总数，而不仅仅是自动扩展器控制的机器。确保这个值足够大，足以满足所有 control plane 和计算机器以及您在 **MachineAutoscaler** 资源中指定的副本总数。

(3) 指定在集群中部署的最小内核数。

(4) 指定集群中要部署的最大内核数。

(5) 指定集群中最小内存量（以 GiB 为单位）。

(6) 指定集群中的最大内存量（以 GiB 为单位）。

(7) 可选：指定要部署的 GPU 节点的类型。只有 **nvidia.com/gpu** 和 **amd.com/gpu** 是有效的类型。

(8) 指定在集群中部署的最小 GPU 数。

(9) 指定集群中要部署的最大 GPU 数量。

(10) 指定 **0** 到 **10** 之间的日志记录详细程度。为指导提供了以下日志级别阈值：

- **1**：（默认）有关更改的基本信息。
- **4**：用于对典型问题进行故障排除的详细程度。
- **9**：广泛的、协议级的故障排除信息。

如果没有指定值，则使用默认值 **1**。

(11) 在此部分中，您可以指定每个操作要等待的时长，可以使用任何有效的 ParseDuration 间隔，包括 **ns**、**us**、**ms**、**s**、**m** 和 **h**。

(12) 指定集群自动扩展是否可以删除不必要的节点。

(13) 可选：指定在最近*添加*节点之后要等待多久才能删除节点。如果不指定值，则使用默认值 **10m**。

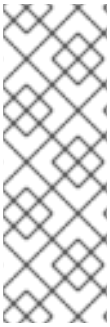
(14) 可选：指定在最近*删除*节点之后要等待多久才能删除节点。如果没有指定值，则使用默认值 **0s**。

(15) 可选：指定在发生缩减失败之后要等待多久才能删除节点。如果不指定值，则使用默认值 **3m**。

(16) 可选：指定不必要的节点有资格删除前的时间。如果不指定值，则使用默认值 **10m**。

(17) 可选：指定 *节点使用率级别*。此使用率级别下的节点可以被删除。

节点使用率是请求的资源的总和（由节点分配的资源划分），且值必须大于 **"0"**，但小于 **"1"**。如果没有指定值，集群自动扩展会使用默认值 **"0.5"**，它对应于 50% 的使用率。您必须以字符串形式表示这个值。



注意

执行扩展操作时，集群自动扩展会保持在 **ClusterAutoscaler** 资源定义中设置的范围，如要部署的最小和最大内核数，或集群中的内存量。但是，集群自动扩展无法将集群中的当前值修正为在这些范围内。

最小和最大 CPU、内存和 GPU 值是通过计算集群中所有节点上的这些资源来确定，即使集群自动扩展无法管理该节点。例如，control plane 节点在集群的总内存中考虑，即使集群自动扩展不管理 control plane 节点。

8.10.2. 部署集群自动扩展

要部署集群自动扩展，请创建一个 **ClusterAutoscaler** 资源实例。

流程

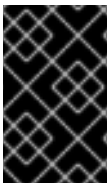
1. 为包含自定义资源定义的 **ClusterAutoscaler** 资源创建一个 YAML 文件。
2. 运行以下命令在集群中创建自定义资源：

```
$ oc create -f <filename>.yaml 1
```

1 **<filename>** 是自定义资源文件的名称。

8.11. 关于机器自动扩展

机器自动扩展会调整您在 OpenShift Container Platform 集群中部署的计算机器集中的机器数量。您可以扩展默认 **worker** 计算机器集，以及您创建的任何其他计算机器集。当集群没有足够资源来支持更多部署时，机器自动扩展会增加 Machine。对 **MachineAutoscaler** 资源中的值（如最小或最大实例数量）的任何更改都会立即应用到目标计算机器设置。



重要

您必须部署机器自动扩展才能使用集群自动扩展功能来扩展机器。集群自动扩展使用机器自动扩展集上的注解来确定可扩展的资源。如果您在没有定义机器自动扩展的情况下定义集群自动扩展，集群自动扩展永远不会扩展集群。

8.11.1. 机器自动扩展资源定义

此 **MachineAutoscaler** 资源定义显示了机器自动扩展器的参数和示例值。

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "worker-us-east-1a" 1
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1 2
  maxReplicas: 12 3
  scaleTargetRef: 4
```

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet 5
name: worker-us-east-1a 6
```

1 指定机器自动扩展名称。为了更容易识别此机器自动扩展会扩展哪些计算机器，请指定或包含要扩展的计算机器设置的名称。计算机器集名称采用以下形式： **<clusterid>-<machineset>-<region>**。

2 指定在机器自动扩展启动集群扩展后必须保留在指定区域中的指定类型的最小机器数量。如果在 AWS、GCP、Azure、RHOSP 或 vSphere 中运行，则此值可设为 **0**。对于其他供应商，请不要将此值设置为 **0**。

对于用于特殊工作负载的高价或有限使用硬件，或者扩展具有额外大型机器的计算机器集，您可以将此值设置为 **0** 来节约成本。如果机器没有使用，集群自动扩展会将计算机器设置缩减为零。

重要

对于安装程序置备的基础架构，请不要将 OpenShift Container Platform 安装过程中创建的三台计算机器集的 **spec.minReplicas** 值设置为 **0**。

3 指定集群自动扩展初始化集群扩展后可在指定类型区域中部署的指定类型的最大机器数量。确保 **ClusterAutoscaler** 资源定义的 **maxNodesTotal** 值足够大，以便机器自动扩展器可以部署这个数量的机器。

4 在本小节中，提供描述要扩展的现有计算机器设置的值。

5 **kind** 参数值始终为 **MachineSet**。

6 **name** 值必须与现有计算机器集的名称匹配，如 **metadata.name** 参数值中所示。

8.11.2. 部署机器自动扩展

要部署机器自动扩展，请创建一个 **MachineAutoscaler** 资源实例。

流程

1. 为 **MachineAutoscaler** 资源创建一个 YAML 文件，其中包含自定义资源定义。
2. 运行以下命令在集群中创建自定义资源：

```
$ oc create -f <filename>.yaml 1
```

1 **<filename>** 是自定义资源文件的名称。

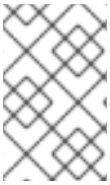
8.12. 配置 LINUX CGROUP

自 OpenShift Container Platform 4.14 起，OpenShift Container Platform 在集群中使用 Linux 控制组版本 2 (cgroup v2)。如果您在 OpenShift Container Platform 4.13 或更早版本中使用 cgroup v1，迁移到 OpenShift Container Platform 4.14 或更高版本不会自动将 cgroup 配置更新至版本 2。全新安装 OpenShift Container Platform 4.14 或更高版本默认使用 cgroup v2。但是，您可以在安装时启用 Linux 控制组群版本 1 (cgroup v1)。

cgroup v2 是 Linux cgroup API 的当前版本。cgroup v2 比 cgroup v1 提供多种改进，包括统一层次结构、

更安全的子树委派、新功能，如 Pressure Stall Information，以及增强的资源管理和隔离。但是，cgroup v2 与 cgroup v1 具有不同的 CPU、内存和 I/O 管理特征。因此，在运行 cgroup v2 的集群上，一些工作负载可能会遇到内存或 CPU 用量差异。

您可以根据需要在 cgroup v1 和 cgroup v2 之间更改。在 OpenShift Container Platform 中启用 cgroup v1 禁用集群中的所有 cgroup v2 控制器和层次结构。



注意

目前，cgroup v2 不支持禁用 CPU 负载均衡。因此，如果您启用了 cgroup v2，则可能无法从性能配置集中获取所需的行为。如果您使用 executeace 配置集，则不建议启用 cgroup v2。

先决条件

- 您有一个正在运行的 OpenShift Container Platform 集群，它使用版本 4.12 或更高版本。
- 以具有管理特权的用户身份登录集群。

流程

1. 在节点上启用 cgroup v1：
   a. 编辑 **node.config** 对象：

```
$ oc edit nodes.config/cluster
```

   b. 添加 **spec.cgroupMode: "v1"**:

node.config 对象示例

```
apiVersion: config.openshift.io/v2
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v2
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  cgroupMode: "v1" (1)
...
```

(1) 启用 cgroup v1。

验证

1. 检查机器配置以查看是否添加了新的机器配置：

```
$ oc get mc
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
97-master-generated-kubelet                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-generated-kubelet                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                       3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                                       3.2.0             40m
rendered-master-23d4317815a5f854bd3553d689cfe2e9
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             10s (1)
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-dcc7f1b92892d34db74d6832bcc9ccd4
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             10s
```

(1) 创建新机器配置，如预期一样。

2. 检查新的 **kernelArguments** 是否已添加到新机器配置中：

```
$ oc describe mc <name>
```

cgroup v1 的输出示例

```
apiVersion: machineconfiguration.openshift.io/v2
kind: MachineConfig
metadata:
  labels:
```

```
    machineconfiguration.openshift.io/role: worker
  name: 05-worker-kernelarg-selinuxpermissive
spec:
  kernelArguments:
    systemd.unified_cgroup_hierarchy=0 (1)
    systemd.legacy_systemd_cgroup_controller=1 (2)
```

1. 禁用 cgroup v2。
2. 在 systemd 中启用 cgroup v1。

3. 检查节点以查看是否禁用了在节点上调度。这表示要应用更改：

```
$ oc get nodes
```

输出示例

```
NAME                                STATUS                     ROLES    AGE   VERSION
ci-ln-fm1qnwt-72292-99kt6-master-0  Ready,SchedulingDisabled   master   58m
v1.28.5
ci-ln-fm1qnwt-72292-99kt6-master-1  Ready                      master   58m   v1.28.5
ci-ln-fm1qnwt-72292-99kt6-master-2  Ready                      master   58m   v1.28.5
ci-ln-fm1qnwt-72292-99kt6-worker-a-h5gt4  Ready,SchedulingDisabled   worker   48m
v1.28.5
ci-ln-fm1qnwt-72292-99kt6-worker-b-7vtmd  Ready                      worker   48m   v1.28.5
ci-ln-fm1qnwt-72292-99kt6-worker-c-rhzkv  Ready                      worker   48m   v1.28.5
```

4. 节点返回 **Ready** 状态后，为该节点启动 debug 会话：

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

5. 将 **/host** 设置为 debug shell 中的根目录：

```
sh-4.4# chroot /host
```

6. 检查您的节点上是否存在 **sys/fs/cgroup/cgroup2fs** 文件。此文件由 cgroup v1 创建：

```
$ stat -c %T -f /sys/fs/cgroup
```

输出示例

```
cgroup2fs
```

其他资源

- 在节点上配置 Linux cgroup 版本

8.13. 使用 FEATUREGATE 启用技术预览功能

您可以通过编辑 **FeatureGate** 自定义资源（CR）为集群中的所有节点开启当前技术预览功能的子集。

8.13.1. 了解功能门

您可以使用 **FeatureGate** 自定义资源（CR）在集群中启用特定的功能集。功能集是 OpenShift Container Platform 功能的集合，默认情况下不启用。

您可以使用 **FeatureGate** CR 激活以下功能集：

- **TechPreviewNoUpgrade**.这个功能集是当前技术预览功能的子集。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，同时保留生产集群中禁用的功能。

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。您不应该在生产环境集群中启用此功能。

此功能集启用了以下技术预览功能：

- 外部云供应商。为 vSphere、AWS、Azure 和 GCP 上的集群启用外部云供应商的支持。对 OpenStack 的支持是 GA。这是一个内部功能，大多数用户不需要与之交互。(**ExternalCloudProvider**)
- OpenShift 构建中的共享资源 CSI 驱动程序。启用 Container Storage Interface (CSI)。(**CSIDriverSharedResource**)
- 节点上的交换内存。根据每个节点为 OpenShift Container Platform 工作负载启用交换内存使用。(**NodeSwap**)
- OpenStack Machine API 提供程序。此最低要求无效，计划在以后的发行版本中从此功能集中删除。(**MachineAPIProviderOpenStack**)
- Insights Operator。启用 **InsightsDataGather** CRD，允许用户配置一些 Insights 数据收集选项。功能集还启用了 **DataGather** CRD，允许用户按需运行 Insights 数据收集。(**InsightsConfigAPI**)
- 递归默认存储类.如果 PVC 创建时没有默认存储类(**RetroactiveDefaultStorageClass**) 启用 OpenShift Container Platform 会主动地将默认存储类分配给 PVC。
- 动态资源分配 API。启用一个新的 API 在 pod 和容器间请求和共享资源。这是一个内部功能，大多数用户不需要与之交互。(**DynamicResourceAllocation**)
- Pod 安全准入强制。为 pod 安全准入启用受限强制模式。如果 pod 违反了 pod 安全标准，它们会被拒绝，而不是仅记录警告信息。(**OpenShiftPodSecurityAdmission**)
- StatefulSet pod 可用性升级限制。允许用户定义在更新过程中不可用的 statefulset pod 的最大数量，这可以减少应用程序停机时间。(**MaxUnavailableStatefulSet**)
- Admin Network 策略和 Baseline Admin Network 策略。在运行 OVN-Kubernetes CNI 插件的集群中，启用 **AdminNetworkPolicy** 和 **BaselineAdminNetworkPolicy** 资源，它们是 Network Policy V2 API 的一部分。集群管理员可以在创建命名空间前为整个集群应用集群范围的策略和保护。网络管理员可以通过强制无法覆盖的网络流量控制来保护集群。如果需要，网络管理员可以实施可选的基准网络流量控制，这些流量可以被集群中的用户覆盖。目前，这些 API 仅支持集群内流量的策略。(**AdminNetworkPolicy**)

- **MatchConditions** 是一个必须满足的条件列表，只要在满足这些条件时才向此 webhook 发送请求。匹配条件过滤请求，它们已与 rules、namespaceSelector 和 objectSelector 匹配。一个空的 **matchConditions** 列表，匹配所有请求。(**admissionWebhookMatchConditions**)
- 网关 API.要启用 OpenShift Container Platform Gateway API，在 **ServiceMeshControlPlane** 资源的 **techPreview.gatewayAPI** 规格中将 **enabled** 字段的值设置为 **true**。(**gateGatewayAPI**)
- **gcpLabelsTags**
- **vSphereStaticIPs**
- **routeExternalCertificate**
- **automatedEtcdBackup**
- **gcpClusterHostedDNS**
- **vSphereControlPlaneMachineset**
- **dnsNameResolver**
- **machineConfigNodes**
- **metricsServer**
- **installAlternateInfrastructureAWS**
- **sdnLiveMigration**
- **mixedCPUsAllocation**
- **managedBootImages**
- **onClusterBuild**
- **signatureStores**

8.13.2. 使用 Web 控制台启用功能集

您可以通过编辑 **FeatureGate** 自定义资源（CR）来使用 OpenShift Container Platform Web 控制台为集群中的所有节点启用功能集。

流程

启用功能集：

1. 在 OpenShift Container Platform web 控制台中，切换到 **Administration → Custom Resource Definitions** 页面。
2. 在 **Custom Resource Definitions** 页面中，点击 **FeatureGate**。
3. 在 **Custom Resource Definition Details** 页面中，点 **Instances** 选项卡。
4. 点 **集群** 功能门，然后点 **YAML** 选项卡。
5. 编辑**集群**实例以添加特定的功能集：

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。您不应该在生产环境集群中启用此功能。

功能门自定义资源示例

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster (1)
# ...
spec:
  featureSet: TechPreviewNoUpgrade (2)
```

1. **FeatureGate** CR 的名称必须是 **cluster**。
2. 添加要启用的功能集：
   - **TechPreviewNoUpgrade** 启用了特定的技术预览功能。

保存更改后，会创建新的机器配置，然后更新机器配置池，并在应用更改时在每个节点上调度。

验证

您可以在节点返回就绪状态后查看节点上的 **kubelet.conf** 文件来验证是否启用了功能门。

1. 从 Web 控制台中的 **Administrator** 视角，进入到 **Compute → Nodes**。
2. 选择一个节点。
3. 在 **Node 详情**页面中，点 **Terminal**。
4. 在终端窗口中，将根目录改为 **/host** ：

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

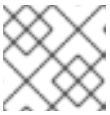
5. 查看 **kubelet.conf** 文件：

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

输出示例

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...
```

在集群中启用列为 **true** 的功能。



注意

列出的功能因 OpenShift Container Platform 版本的不同而有所不同。

8.13.3. 使用 CLI 启用功能集

您可以通过编辑 **FeatureGate** 自定义资源（CR）来使用 OpenShift CLI（**oc**）为集群中的所有节点启用功能集。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

启用功能集：

1. 编辑名为 **cluster** 的 **FeatureGate** CR：

```
$ oc edit featuregate cluster
```

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。您不应该在生产环境集群中启用此功能。

FeatureGate 自定义资源示例

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster (1)
# ...
spec:
  featureSet: TechPreviewNoUpgrade (2)
```

(1) **FeatureGate** CR 的名称必须是 **cluster**。

(2) 添加要启用的功能集：

- **TechPreviewNoUpgrade** 启用了特定的技术预览功能。

保存更改后，会创建新的机器配置，然后更新机器配置池，并在应用更改时在每个节点上调度。

验证

您可以在节点返回就绪状态后查看节点上的 **kubelet.conf** 文件来验证是否启用了功能门。

1. 从 Web 控制台中的 **Administrator** 视角，进入到 **Compute → Nodes**。
2. 选择一个节点。
3. 在 **Node 详情**页面中，点 **Terminal**。
4. 在终端窗口中，将根目录改为 **/host** ：

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

5. 查看 **kubelet.conf** 文件：

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

输**出示例**

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...
```

在集群中启用列为 **true** 的功能。

注意

列出的功能因 OpenShift Container Platform 版本的不同而有所不同。

8.14. ETCD 任务

备份 etcd、启用或禁用 etcd 加密或清除 etcd 数据。

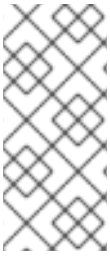
8.14.1. 关于 etcd 加密

默认情况下，OpenShift Container Platform 不加密 etcd 数据。在集群中启用对 etcd 进行加密的功能可为数据的安全性提供额外的保护层。例如，如果 etcd 备份暴露给不应该获得这个数据的人员，它会帮助保护敏感数据。

启用 etcd 加密时，以下 OpenShift API 服务器和 Kubernetes API 服务器资源将被加密：

- Secrets
- 配置映射
- Routes
- OAuth 访问令牌
- OAuth 授权令牌

当您启用 etcd 加密时，会创建加密密钥。您必须具有这些密钥才能从 etcd 备份中恢复。



注意

etcd 加密只加密值，而不加密键。资源类型、命名空间和对象名称是未加密的。

如果在备份过程中启用了 etcd 加密，***static_kuberesources_<datetimestamp>.tar.gz*** 文件包含 etcd 快照的加密密钥。为安全起见，请将此文件与 etcd 快照分开存储。但是，需要这个文件才能从相应的 etcd 快照恢复以前的 etcd 状态。

8.14.2. 支持的加密类型

在 OpenShift Container Platform 中，支持以下加密类型来加密 etcd 数据：

AES-CBC

使用带有 PKCS#7 padding 和 32 字节密钥的 AES-CBC 来执行加密。加密密钥每周轮转。

AES-GCM

使用带有随机 nonce 和 32 字节密钥的 AES-GCM 来执行加密。加密密钥每周轮转。

8.14.3. 启用 etcd 加密

您可以启用 etcd 加密来加密集群中的敏感资源。

警告

在初始加密过程完成前，不要备份 etcd 资源。如果加密过程还没有完成，则备份可能只被部分加密。

启用 etcd 加密后，可能会出现一些更改：

- etcd 加密可能会影响几个资源的内存消耗。
- 您可能会注意到对备份性能具有临时影响，因为领导必须提供备份服务。
- 磁盘 I/O 可能会影响接收备份状态的节点。

您可以在 AES-GCM 或 AES-CBC 加密中加密 etcd 数据库。

注意

要将 etcd 数据库从一个加密类型迁移到另一个加密类型，您可以修改 API 服务器的 **spec.encryption.type** 字段。将 etcd 数据迁移到新的加密类型会自动进行。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 修改 **APIServer** 对象：

```
$ oc edit apiserver
```

2. 将 **spec.encryption.type** 字段设置为 **aesgcm** 或 **aescbc** ：

```
spec:
  encryption:
    type: aesgcm (1)
```

(1) 设置为 **aesgcm** 用于 AES-GCM 加密，或设置为 **aescbc** 用于 AES-CBC 加密。

3. 保存文件以使改变生效。
加密过程开始。根据 etcd 数据库的大小，这个过程可能需要 20 分钟或更长时间才能完成。

4. 验证 etcd 加密是否成功。

   a. 查看 OpenShift API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

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

   在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

   如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

   b. 查看 Kubernetes API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

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

   在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

   如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

   c. 查看 OpenShift OAuth API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

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

   在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

   如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

8.14.4. 禁用 etcd 加密

您可以在集群中禁用 etcd 数据的加密。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 修改 **APIServer** 对象：

```
$ oc edit apiserver
```

2. 将 **encryption** 字段类型设置为 **identity**：

```
spec:
  encryption:
    type: identity (1)
```

(1) **identity** 类型是默认值，意味着没有执行任何加密。

3. 保存文件以使改变生效。
   解密过程开始。根据集群的大小，这个过程可能需要 20 分钟或更长的时间才能完成。

4. 验证 etcd 解密是否成功。

   a. 查看 OpenShift API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

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

   在成功解密后输出显示 **DecryptionCompleted**：

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

   b. 查看 Kubernetes API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

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

   在成功解密后输出显示 **DecryptionCompleted**：

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

   c. 查看 OpenShift OAuth API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

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

在成功解密后输出显示 **DecryptionCompleted**：

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

8.14.5. 备份 etcd 数据

按照以下步骤，通过创建 etcd 快照并备份静态 pod 的资源来备份 etcd 数据。这个备份可以被保存，并在以后需要时使用它来恢复 etcd 数据。

重要

只保存单一 control plane 主机的备份。不要从集群中的每个 control plane 主机进行备份。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您已检查是否启用了集群范围代理。

提示

您可以通过查看 **oc get proxy cluster -o yaml** 的输出来检查代理是否已启用。如果 **httpProxy**、**httpsProxy**和 **noProxy** 字段设置了值，则会启用代理。

流程

1. 以 root 用户身份为 control plane 节点启动一个 debug 会话：

```
$ oc debug --as-root node/<node_name>
```

2. 在 debug shell 中将根目录改为 **/host** ：

```
sh-4.4# chroot /host
```

3. 如果启用了集群范围的代理，请确定已导出了 **NO_PROXY**、**HTTP_PROXY**和 **HTTPS_PROXY** 环境变量。

4. 在 debug shell 中运行 **cluster-backup.sh** 脚本，并传递保存备份的位置。

提示

cluster-backup.sh 脚本作为 etcd Cluster Operator 的一个组件被维护，它是 **etcdctl snapshot save** 命令的包装程序（wrapper）。

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

脚本输出示例

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

在这个示例中，在 control plane 主机上的 **/home/core/assets/backup/** 目录中创建了两个文件：

- **snapshot_<datetimestamp>.db**：这个文件是 etcd 快照。**cluster-backup.sh** 脚本确认其有效。
- **static_kuberesources_<datetimestamp>.tar.gz**：此文件包含静态 pod 的资源。如果启用了 etcd 加密，它也包含 etcd 快照的加密密钥。

注意

如果启用了 etcd 加密，建议出于安全考虑，将第二个文件与 etcd 快照分开保存。但是，需要这个文件才能从 etcd 快照中进行恢复。

请记住，etcd 仅对值进行加密，而不对键进行加密。这意味着资源类型、命名空间和对象名称是不加密的。

8.14.6. 分离 etcd 数据

对于大型、高密度的集群，如果键空间增长过大并超过空间配额，etcd 的性能将会受到影响。定期维护并处理碎片化的 etcd，以释放数据存储中的空间。监控 Prometheus 以了解 etcd 指标数据，并在需要时对其进行碎片处理；否则，etcd 可能会引发一个集群范围的警报，使集群进入维护模式，仅能接受对键的读和删除操作。

监控这些关键指标：

- **etcd_server_quota_backend_bytes**，这是当前配额限制

- **etcd_mvcc_db_total_size_in_use_in_bytes**，表示历史压缩后实际数据库使用量
- **etcd_mvcc_db_total_size_in_bytes** 显示数据库大小，包括等待碎片整理的可用空间

在导致磁盘碎片的事件后（如 etcd 历史记录紧凑）对 etcd 数据进行清理以回收磁盘空间。

历史压缩将自动每五分钟执行一次，并在后端数据库中造成混乱。此碎片空间可供 etcd 使用，但主机文件系统不可用。您必须对碎片 etcd 进行碎片清除，才能使这个空间可供主机文件系统使用。

碎片清理会自动发生，但您也可以手动触发它。

注意

自动清理碎片非常适合大多数情况，因为 etcd operator 使用集群信息来确定用户最有效的操作。

8.14.6.1. 自动清理

etcd Operator 自动清理碎片磁盘。不需要人工干预。

查看以下日志之一来验证碎片整理过程是否成功：

- etcd 日志
- cluster-etcd-operator pod
- Operator 状态错误日志

警告

自动清除可能会导致各种 OpenShift 核心组件中的领导选举失败，如 Kubernetes 控制器管理器，这会触发重启失败的组件。重启会有危害，并会触发对下一个正在运行的实例的故障切换，或者组件在重启后再次恢复工作。

成功进行碎片处理的日志输出示例

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

进行碎片处理失败的日志输出示例

```
failed defrag on member: <member_name>, memberID: <member_id>: <error_message>
```

8.14.6.2. 手动清理

Prometheus 警报指示您需要手动进行碎片处理。该警报在两个情况下显示：

- 当 etcd 使用超过 50% 的可用空间超过了 10 分钟
- 当 etcd 活跃使用小于其数据库总大小的 50% 超过了 10 分钟

您还可以通过检查 etcd 数据库大小（MB）来决定是否需要进行碎片整理。通过 PromQL 表达 **(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024** 来释放空间。

警告

分离 etcd 是一个阻止性操作。在进行碎片处理完成前，etcd 成员将没有响应。因此，在每个下一个 pod 要进行碎片清理前，至少等待一分钟，以便集群可以恢复正常工作。

按照以下步骤对每个 etcd 成员上的 etcd 数据进行碎片处理。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 确定哪个 etcd 成员是领导成员，因为领导会进行最后的碎片处理。
   a. 获取 etcd pod 列表：

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

   输出示例

```
etcd-ip-10-0-159-225.example.redhat.com        3/3     Running     0          175m
10.0.159.225   ip-10-0-159-225.example.redhat.com   <none>           <none>
etcd-ip-10-0-191-37.example.redhat.com         3/3     Running     0          173m
10.0.191.37    ip-10-0-191-37.example.redhat.com    <none>           <none>
etcd-ip-10-0-199-170.example.redhat.com        3/3     Running     0          176m
10.0.199.170   ip-10-0-199-170.example.redhat.com   <none>           <none>
```

   b. 选择 pod 并运行以下命令来确定哪个 etcd 成员是领导：

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint
status --cluster -w table
```

   输出示例

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+--------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
|         ENDPOINT         |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
```

```
| https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |     91624 |            91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |     false |      false |
7 |     91624 |            91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |     91624 |            91624 |        |
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

基于此输出的 **IS LEADER** 列，**https://10.0.199.170:2379** 端点是领导。与上一步输出匹配此端点，领导的 pod 名称为 **etcd-ip-10-0-199-170.example.redhat.com**。

2. 清理 etcd 成员。

   a. 连接到正在运行的 etcd 容器，传递 *不是* 领导的 pod 的名称：

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
   ```

   b. 取消设置 **ETCDCTL_ENDPOINTS** 环境变量：

   ```
   sh-4.4# unset ETCDCTL_ENDPOINTS
   ```

   c. 清理 etcd 成员：

   ```
   sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
   ```

   输出示例

   ```
   Finished defragmenting etcd member[https://localhost:2379]
   ```

   如果发生超时错误，增加 **--command-timeout** 的值，直到命令成功为止。

   d. 验证数据库大小是否已缩小：

   ```
   sh-4.4# etcdctl endpoint status -w table --cluster
   ```

   输出示例

   ```
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+-------------------+--------+
   |        ENDPOINT           |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
   RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+-------------------+--------+
   | https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
   7 |     91624 |            91624 |        |
   | https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |   41 MB |     false |      false |
   7 |     91624 |            91624 |        | (1)
   | https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
   7 |     91624 |            91624 |        |
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+-------------------+--------+
   ```

本例显示这个 etcd 成员的数据库大小现在为 41 MB，而起始大小为 104 MB。

e. 重复这些步骤以连接到其他 etcd 成员并进行碎片处理。最后才对领导进行碎片清除。至少要在碎片处理操作之间等待一分钟，以便 etcd pod 可以恢复。在 etcd pod 恢复前，etcd 成员不会响应。

3. 如果因为超过空间配额而触发任何 **NOSPACE** 警告，请清除它们。

a. 检查是否有 **NOSPACE** 警告：

```
sh-4.4# etcdctl alarm list
```

输**出示例**

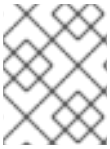
```
memberID:12345678912345678912 alarm:NOSPACE
```

b. 清除警告：

```
sh-4.4# etcdctl alarm disarm
```

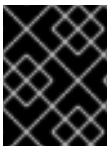
8.14.7. 恢复到一个以前的集群状态

您可以使用保存的 **etcd** 备份来恢复以前的集群状态，或恢复丢失了大多数 control plane 主机的集群。



注意

如果您的集群使用 control plane 机器集，请参阅 "Troubleshooting control plane 机器集"来了解有关 **etcd** 恢复的过程。



重要

恢复集群时，必须使用同一 z-stream 发行版本中获取的 **etcd** 备份。例如，OpenShift Container Platform 4.7.2 集群必须使用从 4.7.2 开始的 **etcd** 备份。

先决条件

- 通过一个基于证书的 **kubeconfig** 使用具有 **cluster-admin** 角色的用户访问集群，如安装期间的情况。
- 用作恢复主机的健康 control plane 主机。
- SSH 对 control plane 主机的访问。
- 包含从同一备份中获取的 etcd 快照和静态 **pod** 资源的备份目录。该目录中的文件名必须采用以下格式: **snapshot_<datetimestamp>.db** 和 **static_kuberesources_<datetimestamp>.tar.gz**。

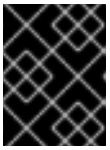
重要

对于非恢复 control plane 节点，不需要建立 SSH 连接或停止静态 pod。您可以逐个删除并重新创建其他非恢复 control plane 机器。

流程

1. 选择一个要用作恢复主机的 control plane 主机。这是您要在其中运行恢复操作的主机。

2. 建立到每个 control plane 节点（包括恢复主机）的 SSH 连接。
恢复过程启动后，**kube-apiserver** 将无法访问，因此您无法访问 control plane 节点。因此，建议在一个单独的终端中建立到每个control plane 主机的 SSH 连接。



重要

如果没有完成这个步骤，将无法访问 control plane 主机来完成恢复过程，您将无法从这个状态恢复集群。

3. 将 **etcd** 备份目录复制到恢复 control plane 主机上。
此流程假设您将 **backup** 目录（其中包含 **etcd** 快照和静态 pod 资源）复制到恢复 control plane 主机的 **/home/core/** 目录中。

4. 在任何其他 control plane 节点上停止静态 pod。

注意

您不需要停止恢复主机上的静态 pod。

a. 访问不是恢复主机的 control plane 主机。

b. 运行以下命令，将现有 etcd pod 文件从 kubelet 清单目录中移出：

```
$ sudo mv -v /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

c. 使用以下命令验证 **etcd** pod 是否已停止：

```
$ sudo crictl ps | grep etcd | egrep -v "operator|etcd-guard"
```

如果这个命令的输出不为空，请等待几分钟，然后再次检查。

d. 运行以下命令，将现有 **kube-apiserver** 文件从 kubelet 清单目录中移出：

```
$ sudo mv -v /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

e. 运行以下命令验证 **kube-apiserver** 容器是否已停止：

```
$ sudo crictl ps | grep kube-apiserver | egrep -v "operator|guard"
```

如果这个命令的输出不为空，请等待几分钟，然后再次检查。

f. 使用以下方法将现有 **kube-controller-manager** 文件从 kubelet 清单目录中移出：

```
$ sudo mv -v /etc/kubernetes/manifests/kube-controller-manager-pod.yaml /tmp
```

g. 运行以下命令验证 **kube-controller-manager** 容器是否已停止：

```
$ sudo crictl ps | grep kube-controller-manager | egrep -v "operator|guard"
```

如果这个命令的输出不为空，请等待几分钟，然后再次检查。

h. 使用以下方法将现有 **kube-scheduler** 文件从 kubelet 清单目录中移出：

```
$ sudo mv -v /etc/kubernetes/manifests/kube-scheduler-pod.yaml /tmp
```

i. 使用以下命令验证 **kube-scheduler** 容器是否已停止：

```
$ sudo crictl ps | grep kube-scheduler | egrep -v "operator|guard"
```

如果这个命令的输出不为空，请等待几分钟，然后再次检查。

j. 使用以下示例将 **etcd** 数据目录移到不同的位置：

```
$ sudo mv -v /var/lib/etcd/ /tmp
```

k. 如果 **/etc/kubernetes/manifests/keepalived.yaml** 文件存在，且节点被删除，请按照以下步骤执行：

i. 将 **/etc/kubernetes/manifests/keepalived.yaml** 文件从 kubelet 清单目录中移出：

```
$ sudo mv -v /etc/kubernetes/manifests/keepalived.yaml /tmp
```

ii. 容器验证由 **keepalived** 守护进程管理的任何容器是否已停止：

```
$ sudo crictl ps --name keepalived
```

命令输出应该为空。如果它不是空的，请等待几分钟后再重新检查。

iii. 检查 control plane 是否已分配任何 Virtual IP (VIP)：

```
$ ip -o address | egrep '<api_vip>|<ingress_vip>'
```

iv. 对于每个报告的 VIP，运行以下命令将其删除：

```
$ sudo ip address del <reported_vip> dev <reported_vip_device>
```

l. 在其他不是恢复主机的 control plane 主机上重复此步骤。

5. 访问恢复 control plane 主机。

6. 如果使用 **keepalived** 守护进程，请验证恢复 control plane 节点是否拥有 VIP：

```
$ ip -o address | grep <api_vip>
```

如果存在 VIP 的地址（如果存在）。如果 VIP 没有设置或配置不正确，这个命令会返回一个空字符串。

7. 如果启用了集群范围的代理，请确定已导出了 **NO_PROXY**、**HTTP_PROXY**和 **HTTPS_PROXY** 环境变量。

提示

您可以通过查看 **oc get proxy cluster -o yaml** 的输出来检查代理是否已启用。如果 **httpProxy**、**httpsProxy**和 **noProxy** 字段设置了值，则会启用代理。

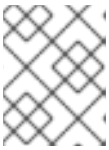
8. 在恢复 control plane 主机上运行恢复脚本，并传递到 **etcd** 备份目录的路径：

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/assets/backup
```

脚本输出示例

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

cluster-restore.sh 脚本必须显示 **etcd**、**kube-apiserver**、**kube-controller-manager** 和 **kube-scheduler** pod 已停止，然后在恢复过程结束时启动。



注意

如果在上次 **etcd** 备份后更新了节点，则恢复过程可能会导致节点进入 **NotReady** 状态。

9. 检查节点以确保它们处于 **Ready** 状态。

   a. 运行以下命令:

```
$ oc get nodes -w
```

输出示例

```
NAME              STATUS  ROLES         AGE    VERSION
host-172-25-75-28   Ready   master       3d20h   v1.28.5
host-172-25-75-38   Ready   infra,worker   3d20h   v1.28.5
host-172-25-75-40   Ready   master       3d20h   v1.28.5
host-172-25-75-65   Ready   master       3d20h   v1.28.5
host-172-25-75-74   Ready   infra,worker   3d20h   v1.28.5
```

```
host-172-25-75-79   Ready   worker        3d20h   v1.28.5
host-172-25-75-86   Ready   worker        3d20h   v1.28.5
host-172-25-75-98   Ready   infra,worker  3d20h   v1.28.5
```

所有节点都可能需要几分钟时间报告其状态。

b. 如果有任何节点处于 **NotReady** 状态，登录到节点，并从每个节点上的 **/var/lib/kubelet/pki** 目录中删除所有 PEM 文件。您可以 SSH 到节点，或使用 web 控制台中的终端窗口。

```
$  ssh -i <ssh-key-path> core@<master-hostname>
```

pki 目录示例

```
sh-4.4# pwd
/var/lib/kubelet/pki
sh-4.4# ls
kubelet-client-2022-04-28-11-24-09.pem  kubelet-server-2022-04-28-11-24-15.pem
kubelet-client-current.pem              kubelet-server-current.pem
```

10. 在所有 control plane 主机上重启 kubelet 服务。

a. 在恢复主机中运行：

```
$ sudo systemctl restart kubelet.service
```

b. 在所有其他 control plane 主机上重复此步骤。

11. 批准待处理的证书签名请求 (CSR)：

注意

没有 worker 节点的集群（如单节点集群或由三个可调度的 control plane 节点组成的集群）不会批准任何待处理的 CSR。您可以跳过此步骤中列出的所有命令。

a. 运行以下命令获取当前 CSR 列表：

```
$ oc get csr
```

输出示例

```
NAME        AGE    SIGNERNAME                                    REQUESTOR
CONDITION
csr-2s94x   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
Pending 1
csr-4bd6t   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
Pending 2
csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
3
csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
4
...
```

■

1 1 2 一个待处理的 kubelet 服务 CSR，由 kubelet 服务端点请求。

3 4 一个待处理的 kubelet 客户端 CSR，使用 **node-bootstrapper** 节点 bootstrap 凭证请求。

b. 运行以下命令，查看 CSR 的详情以验证其是否有效：

```
$ oc describe csr <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

c. 运行以下命令来批准每个有效的 **node-bootstrapper** CSR：

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

d. 对于用户置备的安装，运行以下命令批准每个有效的 kubelet 服务 CSR：

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

12. 确认单个成员 control plane 已被成功启动。

a. 在恢复主机上，使用以下命令验证 **etcd** 容器是否正在运行：

```
$ sudo crictl ps | grep etcd | egrep -v "operator|etcd-guard"
```

输**出示例**

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

b. 在恢复主机上，使用以下命令验证 **etcd** pod 是否正在运行：

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

输**出示例**

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

如果状态是 **Pending**，或者输出中列出了多个正在运行的 **etcd** pod，请等待几分钟，然后再次检查。

13. 如果使用 **OVNKubernetes** 网络插件，您必须重启 **ovnkube-controlplane** pod。

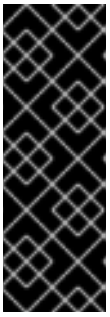
a. 运行以下命令删除所有 **ovnkube-controlplane** pod：

```
$ oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-control-plane
```

b. 使用以下命令验证所有 **ovnkube-controlplane** pod 是否已重新部署：

```
$ oc -n openshift-ovn-kubernetes get pod -l app=ovnkube-control-plane
```

14. 如果使用 OVN-Kubernetes 网络插件，请逐个重启所有节点上的 Open Virtual Network (OVN) Kubernetes pod。使用以下步骤重启每个节点上的 OVN-Kubernetes pod：



重要

按照以下顺序重启 OVN-Kubernetes pod：

1. 恢复控制平面主机
2. 其他控制平面主机（如果可用）
3. 其他节点

注意

验证和变异准入 Webhook 可能会拒绝 pod。如果您添加了额外的 Webhook，其 **failurePolicy** 被设置为 **Fail** 的，则它们可能会拒绝 pod，恢复过程可能会失败。您可以通过在恢复集群状态时保存和删除 Webhook 来避免这种情况。成功恢复集群状态后，您可以再次启用 Webhook。

另外，您可以在恢复集群状态时临时将 **failurePolicy** 设置为 **Ignore**。成功恢复集群状态后，您可以将 **failurePolicy** 设置为 **Fail**。

a. 删除北向数据库 (nbdb) 和南向数据库 (sbdb)。使用 Secure Shell (SSH) 访问恢复主机和剩余的 control plane 节点，并运行：

```
$ sudo rm -f /var/lib/ovn-ic/etc/*.db
```

b. 重新启动 OpenVSwitch 服务。使用 Secure Shell (SSH) 访问节点，并运行以下命令：

```
$ sudo systemctl restart ovs-vswitchd ovsdb-server
```

c. 运行以下命令删除节点上的 **ovnkube-node** pod，将 **<node>** 替换为您要重启的节点的名称：

```
$ oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-node --field-
selector=spec.nodeName==<node>
```

d. 使用以下命令验证 **ovnkube-node** pod 已再次运行：

```
$ oc -n openshift-ovn-kubernetes get pod -l app=ovnkube-node --field-
selector=spec.nodeName==<node>
```

注意

pod 可能需要几分钟时间来重启。

15. 逐个删除并重新创建其他非恢复 control plane 机器。重新创建机器后，会强制一个新修订版

15. 逐个删除并重新创建其他非恢复 control plane 机器。重新创建机器后，会强制一个新修订版本，**etcd** 会自动扩展。

- 如果使用用户置备的裸机安装，您可以使用最初创建它时使用的相同方法重新创建 control plane 机器。如需更多信息，请参阅"在裸机上安装用户置备的集群"。

> **警告**
>
> 不要为恢复主机删除并重新创建机器。

- 如果您正在运行安装程序置备的基础架构，或者您使用 Machine API 创建机器，请按照以下步骤执行：

> **警告**
>
> 不要为恢复主机删除并重新创建机器。
>
> 对于安装程序置备的基础架构上的裸机安装，不会重新创建 control plane 机器。如需更多信息，请参阅"替换裸机控制平面节点"。

a. 为丢失的 control plane 主机之一获取机器。
在一个终端中使用 cluster-admin 用户连接到集群，运行以下命令：

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

输出示例：

```
NAME                                PHASE     TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-0          Running   m4.xlarge   us-east-1   us-east-1a
3h37m   ip-10-0-131-183.ec2.internal   aws:///us-east-1a/i-0ec2782f8287dfb7e
stopped 1
clustername-8qw5l-master-1          Running   m4.xlarge   us-east-1   us-east-1b
3h37m   ip-10-0-143-125.ec2.internal   aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2          Running   m4.xlarge   us-east-1   us-east-1c
3h37m   ip-10-0-154-194.ec2.internal   aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running   m4.large   us-east-1   us-
east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-
010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running   m4.large   us-east-1   us-
east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-
0cb45ac45a166173b   running
```

```
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-06861c00007751b0a  running
```

1 这是用于丢失的 control plane 主机 **ip-10-0-131-183.ec2.internal** 的 control plane 机器。

b. 运行以下命令，将机器配置保存到文件系统中的一个文件中：

```
$ oc get machine clustername-8qw5l-master-0 \ 1
    -n openshift-machine-api \
    -o yaml \
    > new-master-machine.yaml
```

1 为丢失的 control plane 主机指定 control plane 机器的名称。

c. 编辑上一步中创建的 **new-master-machine.yaml** 文件，以分配新名称并删除不必要的字段。

i. 运行以下命令删除整个 **status** 部分：

```
status:
  addresses:
  - address: 10.0.131.183
    type: InternalIP
  - address: ip-10-0-131-183.ec2.internal
    type: InternalDNS
  - address: ip-10-0-131-183.ec2.internal
    type: Hostname
  lastUpdated: "2020-04-20T17:44:29Z"
  nodeRef:
    kind: Node
    name: ip-10-0-131-183.ec2.internal
    uid: acca4411-af0d-4387-b73e-52b2484295ad
  phase: Running
  providerStatus:
    apiVersion: awsproviderconfig.openshift.io/v1beta1
    conditions:
    - lastProbeTime: "2020-04-20T16:53:50Z"
      lastTransitionTime: "2020-04-20T16:53:50Z"
      message: machine successfully created
      reason: MachineCreationSucceeded
      status: "True"
      type: MachineCreation
    instanceId: i-0fdb85790d76d0c3f
    instanceState: stopped
    kind: AWSMachineProviderStatus
```

ii. 运行以下命令，将 **metadata.name** 字段改为新名称：
建议您保留与旧机器相同的基础名称，并将结束号码改为下一个可用数字。在本例中，**clustername-8qw5l-master-0** 被改为 **clustername-8qw5l-master-3** ：

```
apiVersion: machine.openshift.io/v1beta1
```

```
kind: Machine
metadata:
  ...
  name: clustername-8qw5l-master-3
  ...
```

iii. 运行以下命令来删除 **spec.providerID** 字段：

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

iv. 运行以下命令，删除 **metadata.annotations** 和 **metadata.generation** 字段：

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

v. 运行以下命令，删除 **metadata.resourceVersion** 和 **metadata.uid** 字段：

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

d. 运行以下命令，删除丢失的 control plane 主机的机器：

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

1 为丢失的 control plane 主机指定 control plane 机器的名称。

e. 运行以下命令验证机器是否已删除：

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

输出示例：

```
NAME                                PHASE   TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1          Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-143-125.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2          Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-154-194.ec2.internal   aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-worker-us-east-1a-wbtgd   Running m4.large   us-east-1   us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running m4.large   us-east-1   us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26   Running m4.large   us-east-1   us-east-1c   3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-06861c00007751b0a   running
```

f. 运行以下命令，使用 **new-master-machine.yaml** 文件创建机器：

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

g. 运行以下命令验证新机器是否已创建：

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

输出示例：

```
NAME                                PHASE         TYPE        REGION      ZONE
AGE   NODE                          PROVIDERID                          STATE
clustername-8qw5l-master-1          Running       m4.xlarge   us-east-1   us-east-
1b    3h37m   ip-10-0-143-125.ec2.internal   aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2          Running       m4.xlarge   us-east-1   us-east-
1c    3h37m   ip-10-0-154-194.ec2.internal   aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-master-3          Provisioning  m4.xlarge   us-east-1   us-east-
1a    85s     ip-10-0-173-171.ec2.internal   aws:///us-east-1a/i-015b0888fe17bc2c8
running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd   Running       m4.large    us-east-1
us-east-1a    3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-
010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running       m4.large    us-east-1   us-
east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-
0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26   Running       m4.large    us-east-1
us-east-1c    3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-
06861c00007751b0a   running
```

(1) 新机器 **clustername-8qw5l-master-3** 会被创建，并在阶段从 **Provisioning** 变为 **Running** 后就绪。

创建新机器可能需要几分钟时间。当机器或节点返回到健康状态时，**etcd** 集群 Operator 将自动同步。

h. 对不是恢复主机的每个已丢失的 control plane 主机重复此步骤。

16. 输入以下内容关闭仲裁保护：

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

此命令可确保您可以成功重新创建机密并推出静态 pod。

17. 在恢复主机中的一个单独的终端窗口中，运行以下命令导出恢复 **kubeconfig** 文件：

```
$ export KUBECONFIG=/etc/kubernetes/static-pod-resources/kube-apiserver-
certs/secrets/node-kubeconfigs/localhost-recovery.kubeconfig
```

18. 强制 **etcd** 重新部署。
在导出恢复 **kubeconfig** 文件的同一终端窗口中，运行：

-

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$( date --rfc-3339=ns )"'"}}' --type=merge
```
(1)

(1) **forceRedeploymentReason** 值必须是唯一的，这就是为什么附加时间戳的原因。

当 **etcd** 集群 Operator 执行重新部署时，现有节点开始使用与初始 bootstrap 扩展类似的新 pod。

19. 输入以下内容重新打开仲裁保护：

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

20. 您可以运行以下命令来验证 **unsupportedConfigOverrides** 部分是否已从对象中删除：

```
$ oc get etcd/cluster -oyaml
```

21. 验证所有节点是否已更新至最新的修订版本。
在一个终端中使用 **cluster-admin** 用户连接到集群，请运行：

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

查看 **etcd** 的 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新的修订版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

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

(1) 在本例中，最新的修订版本号是 **7**。

如果输出包含多个修订号，如 **2 个节点为修订版本 6；1 个节点为修订版本 7**，这意味着更新仍在进行中。等待几分钟后重试。

22. 重新部署 **etcd** 后，为 control plane 强制进行新的推出部署。 **kube-apiserver** 将在其他节点上重新安装自己，因为 kubelet 使用内部负载均衡器连接到 API 服务器。
在一个终端中使用 **cluster-admin** 用户连接到集群，请运行：

    a. 为 **kube-apiserver** 强制进行新的推出部署：

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

    验证所有节点是否已更新至最新的修订版本。

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

    查看 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

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

1 在本例中，最新的修订版本号是 **7**。

如果输出包含多个修订号，如 **2 个节点为修订版本 6；1 个节点为修订版本 7**，这意味着更新仍在进行中。等待几分钟后重试。

b. 运行以下命令，为 Kubernetes 控制器管理器强制进行新的推出部署：

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

运行以下命令，验证所有节点是否已更新至最新的修订版本：

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

查看 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

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

1 在本例中，最新的修订版本号是 **7**。

如果输出包含多个修订号，如 **2 个节点为修订版本 6；1 个节点为修订版本 7**，这意味着更新仍在进行中。等待几分钟后重试。

c. 运行以下命令，为 **kube-scheduler** 强制进行新的推出部署：

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

使用以下命令验证所有节点是否已更新至最新的修订版本：

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

查看 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

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

1 在本例中，最新的修订版本号是 **7**。

如果输出包含多个修订号，如 **2 个节点为修订版本 6；1 个节点为修订版本 7**，这意味着更新仍在进行中。等待几分钟后重试。

23. 验证所有 control plane 主机是否已启动并加入集群。

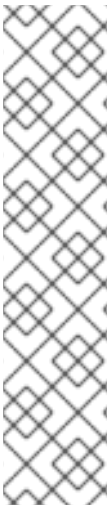
在一个终端中使用 **cluster-admin** 用户连接到集群，运行以下命令：

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

输**出示例**

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

为确保所有工作负载在恢复过程后返回到正常操作，请重启存储 **kube-apiserver** 信息的每个 pod。这包括 OpenShift Container Platform 组件，如路由器、Operator 和第三方组件。



注意

完成前面的流程步骤后，您可能需要等待几分钟，让所有服务返回到恢复的状态。例如，在重启 OAuth 服务器 pod 前，使用 **oc login** 进行身份验证可能无法立即正常工作。

考虑使用 **system:admin kubeconfig** 文件立即进行身份验证。这个方法基于 SSL/TLS 客户端证书作为 OAuth 令牌的身份验证。您可以发出以下命令来使用此文件进行身份验证：

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

发出以下命令以显示您的验证的用户名：

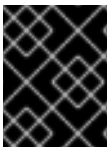
```
$ oc whoami
```

其他资源

- 推荐的 etcd 实践
- 在裸机上安装用户置备的集群
- 替换裸机 control plane 节点

8.14.8. 恢复持久性存储状态的问题和解决方法

如果您的 OpenShift Container Platform 集群使用任何形式的持久性存储，集群的状态通常存储在 etcd 外部。它可能是在 pod 中运行的 Elasticsearch 集群，或者在 **StatefulSet** 对象中运行的数据库。从 etcd 备份中恢复时，还会恢复 OpenShift Container Platform 中工作负载的状态。但是，如果 etcd 快照是旧的，其状态可能无效或过期。



重要

持久性卷（PV）的内容绝不会属于 etcd 快照的一部分。从 etcd 快照恢复 OpenShift Container Platform 集群时，非关键工作负载可能会访问关键数据，反之亦然。

以下是生成过时状态的一些示例情况：

- MySQL 数据库在由 PV 对象支持的 pod 中运行。从 etcd 快照恢复 OpenShift Container Platform 不会使卷恢复到存储供应商上，且不会生成正在运行的 MySQL pod，尽管 pod 会重复尝试启动。您必须通过在存储供应商中恢复卷，然后编辑 PV 以指向新卷来手动恢复这个 pod。

- Pod P1 使用卷 A，它附加到节点 X。如果另一个 pod 在节点 Y 上使用相同的卷，则执行 etcd 恢复时，pod P1 可能无法正确启动，因为卷仍然被附加到节点 Y。OpenShift Container Platform 并不知道附加，且不会自动分离它。发生这种情况时，卷必须从节点 Y 手动分离，以便卷可以在节点 X 上附加，然后 pod P1 才可以启动。

- 在执行 etcd 快照后，云供应商或存储供应商凭证会被更新。这会导致任何依赖于这些凭证的 CSI 驱动程序或 Operator 无法正常工作。您可能需要手动更新这些驱动程序或 Operator 所需的凭证。

- 在生成 etcd 快照后，会从 OpenShift Container Platform 节点中删除或重命名设备。Local Storage Operator 会为从 **/dev/disk/by-id** 或 **/dev** 目录中管理的每个 PV 创建符号链接。这种情况可能会导致本地 PV 引用不再存在的设备。
  要解决这个问题，管理员必须：

  1. 手动删除带有无效设备的 PV。
  2. 从对应节点中删除符号链接。
  3. 删除 **LocalVolume** 或 **LocalVolumeSet** 对象（请参阅 *Storage → Configuring persistent storage → Persistent storage → Persistent storage → Deleting the Local Storage Operator Resources*）。

8.15. POD 中断预算

了解并配置 pod 中断预算。

8.15.1. 了解如何使用 pod 中断预算来指定必须在线的 pod 数量

*pod 中断预算*允许在操作过程中指定 pod 的安全限制，如排空节点以进行维护。

PodDisruptionBudget 是一个 API 对象，用于指定在某一时间必须保持在线的副本的最小数量或百分比。在项目中进行这些设置对节点维护（比如缩减集群或升级集群）有益，而且仅在自愿驱除（而非节点失败）时遵从这些设置。

PodDisruptionBudget 对象的配置由以下关键部分组成：

- 标签选择器，即一组 pod 的标签查询。
- 可用性级别，用来指定必须同时可用的最少 pod 的数量：
  - **minAvailable** 是必须始终可用的 pod 的数量，即使在中断期间也是如此。
  - **maxUnavailable** 是中断期间可以无法使用的 pod 的数量。

注意

Available 指的是具有 **Ready=True** 的 pod 数量。**ready=True** 指的是能够服务请求的 pod，并应添加到所有匹配服务的负载平衡池中。

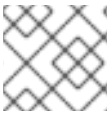
允许 **maxUnavailable** 为 **0%** 或 **0**，**minAvailable** 为 **100%** 或等于副本数，但这样设置可能会阻止节点排空操作。

警告

对于 OpenShift Container Platform 中的所有机器配置池，**maxUnavailable** 的默认设置是 **1**。建议您不要更改这个值，且一次只更新一个 control plane 节点。对于 control plane 池，请不要将这个值改为 **3**。

您可以使用以下命令来检查所有项目的 pod 中断预算：

```
$ oc get poddisruptionbudget --all-namespaces
```



注意

以下示例包含特定于 AWS 上的 OpenShift Container Platform 的一些值。

输出示例

```
NAMESPACE                              NAME                                MIN AVAILABLE   MAX UNAVAILABLE
ALLOWED DISRUPTIONS   AGE
openshift-apiserver                    openshift-apiserver-pdb             N/A             1               1
121m
openshift-cloud-controller-manager     aws-cloud-controller-manager        1               N/A             1
125m
openshift-cloud-credential-operator    pod-identity-webhook                1               N/A             1
117m
openshift-cluster-csi-drivers          aws-ebs-csi-driver-controller-pdb   N/A             1               1
121m
openshift-cluster-storage-operator     csi-snapshot-controller-pdb         N/A             1               1
122m
openshift-cluster-storage-operator     csi-snapshot-webhook-pdb            N/A             1               1
122m
openshift-console                      console                             N/A             1               1
116m
#...
```

如果系统中至少有 **minAvailable** 个 pod 正在运行，则 **PodDisruptionBudget** 被视为是健康的。超过这一限制的每个 pod 都可被驱除。

注意

根据您的 pod 优先级与抢占设置，可能会无视 pod 中断预算要求而移除较低优先级 pod。

8.15.2. 使用 pod 中断预算指定必须在线的 pod 数量

您可以使用 **PodDisruptionBudget** 对象来指定某一时间必须保持在线的副本的最小数量或百分比。

流程

配置 pod 中断预算：

1. 使用类似以下示例的对象定义来创建 YAML 文件：

```
apiVersion: policy/v1 (1)
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 (2)
  selector: (3)
    matchLabels:
      name: my-pod
```

(1) **PodDisruptionBudget** 是 **policy/v1** API 组的一部分。

(2) 必须同时可用的最小 pod 数量。这可以是整数，也可以是指定百分比的字符串（如 **20%**）。

(3) 对一组资源进行的标签查询。**matchLabels** 和 **matchExpressions** 的结果在逻辑上是联合的。要选择项目中的所有 pod，将此参数设置为空，如 **selector {}**。

或者：

```
apiVersion: policy/v1 (1)
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% (2)
  selector: (3)
    matchLabels:
      name: my-pod
```

(1) **PodDisruptionBudget** 是 **policy/v1** API 组的一部分。

(2) 同时不能使用的最多的 pod 数量。这可以是整数，也可以是指定百分比的字符串（如 **20%**）。

(3) 对一组资源进行的标签查询。**matchLabels** 和 **matchExpressions** 的结果在逻辑上是联合的。要选择项目中的所有 pod，将此参数设置为空，如 **selector {}**。

2. 运行以下命令，将对象添加到项目中：

```
$ oc create -f </path/to/file> -n <project_name>
```

8.15.3. 为不健康的 pod 指定驱除策略

当您使用 pod 中断预算 (PDB) 来指定必须同时有多少 pod 可用时，您还可以定义驱除不健康 pod 的条件。

您可以选择以下策略之一：

IfHealthyBudget

只有在保护的应用程序没有被中断时，运行的还没有处于健康状态的 pod 才能被驱除。

AlwaysAllow

无论是否满足 pod 中断预算中的条件，运行的还没有处于健康状态的 pod 都可以被驱除。此策略可帮助驱除出现故障的应用程序，如 pod 处于 **CrashLoopBackOff** 状态或无法报告 **Ready** 状态的应用程序。

注意

建议您在 **PodDisruptionBudget** 对象中将 **unhealthyPodEvictionPolicy** 字段设置为 **AlwaysAllow**，以便在节点排空期间支持收集错误的应用程序。默认行为是等待应用程序 pod 处于健康状态，然后才能排空操作。

流程

1. 创建定义 **PodDisruptionBudget** 对象的 YAML 文件，并指定不健康的 pod 驱除策略：

   pod-disruption-budget.yaml 文件示例

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      name: my-pod
  unhealthyPodEvictionPolicy: AlwaysAllow 1
```

   1 选择 **IfHealthyBudget** 或 **AlwaysAllow** 作为不健康 pod 的驱除策略。当 **unhealthyPodEvictionPolicy** 字段为空时，默认为 **IfHealthyBudget**。

2. 运行以下命令来创建 **PodDisruptionBudget** 对象：

```
$ oc create -f pod-disruption-budget.yaml
```

现在，设置了 **AlwaysAllow** 不健康 pod 驱除策略的 PDB，您可以排空节点并驱除受此 PDB 保护的应用程序的 pod。

其他资源

- 使用功能门启用功能
- Kubernetes 文档中的不健康 Pod 驱除策略

8.16. 为断开连接的集群配置镜像流

在断开连接的环境中安装 OpenShift Container Platform 后，为 Cluster Samples Operator 和 **must-gather** 镜像流配置镜像流。

8.16.1. 协助镜像的 Cluster Samples Operator

在安装过程中，OpenShift Container Platform 在 **openshift-cluster-samples-operator** 命名空间中创建一个名为 **imagestreamtag-to-image** 的配置映射。**imagestreamtag-to-image** 配置映射包含每个镜像流标签的条目（填充镜像）。

配置映射中 data 字段中每个条目的键格式为 **<image_stream_name>_<image_stream_tag_name>**。

在断开连接的 OpenShift Container Platform 安装过程中，Cluster Samples Operator 的状态被设置为 **Removed**。如果您将其改为 **Managed**，它会安装示例。

注意

在网络限制或断开连接的环境中使用示例可能需要通过网络访问服务。某些示例服务包括：Github、Maven Central、npm、RubyGems、PyPi 等。这可能需要执行额外的步骤，让集群 samples operator 对象能够访问它们所需的服务。

您可以使用此配置映射作为导入镜像流所需的镜像的引用。

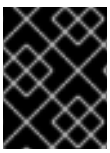
- 在 Cluster Samples Operator 被设置为 **Removed** 时，您可以创建镜像的 registry，或决定您要使用哪些现有镜像 registry。
- 使用新的配置映射作为指南来镜像您要镜像的 registry 的示例。
- 将没有镜像的任何镜像流添加到 Cluster Samples Operator 配置对象的 **skippedImagestreams** 列表中。
- 将 Cluster Samples Operator 配置对象的 **samplesRegistry** 设置为已镜像的 registry。
- 然后，将 Cluster Samples Operator 设置为 **Managed** 来安装您已镜像的镜像流。

8.16.2. 使用带有备用或镜像 registry 的 Cluster Samples Operator 镜像流

openshift 命名空间中大多数由 Cluster Samples Operator 管理的镜像流指向位于 registry.redhat.io 上红帽容器镜像仓库中的镜像。镜像功能不适用于这些镜像流。

注意

cli、**installer**、**must-gather** 和 **test** 镜像流虽然属于安装有效负载的一部分，但不由 Cluster Samples Operator 管理。此流程中不涉及这些镜像流。



重要

Cluster Samples Operator 必须在断开连接的环境中设置为 **Managed**。要安装镜像流，您需要有一个镜像的 registry。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 为您的镜像 registry 创建 pull secret。

流程

1. 访问被镜像（mirror）的特定镜像流的镜像，例如：

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep
registry.redhat.io
```

2. 将 registry.redhat.io 中与您在受限网络环境中需要的任何镜像流关联的镜像，镜像 (mirror) 成一个定义的镜像 (mirror)：

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. 创建集群的镜像配置对象：

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. 在集群的镜像配置对象中，为镜像添加所需的可信 CA：

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

5. 更新 Cluster Samples Operator 配置对象中的 **samplesRegistry** 字段，使其包含镜像配置中定义的镜像位置的 **hostname** 部分：

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

注意

这是必要的，因为镜像流导入过程在此刻不使用镜像（mirro）或搜索机制。

6. 将所有未镜像的镜像流添加到 Cluster Samples Operator 配置对象的 **skippedImagestreams** 字段。或者，如果您不想支持任何示例镜像流，在 Cluster Samples Operator 配置对象中将 Cluster Samples Operator 设置为 **Removed**。

注意

如果镜像流导入失败，Cluster Samples Operator 会发出警告，但 Cluster Samples Operator 会定期重试，或看起来并没有重试它们。

openshift 命名空间中的许多模板都引用镜像流。因此，使用 **Removed** 清除镜像流和模板，将避免在因为缺少镜像流而导致镜像流和模板无法正常工作时使用它们。

8.16.3. 准备集群以收集支持数据

使用受限网络的集群必须导入默认的 must-gather 镜像，以便为红帽支持收集调试数据。在默认情况下，must-gather 镜像不会被导入，受限网络中的集群无法访问互联网来从远程存储库拉取最新的镜像。

流程

1. 如果您还没有将镜像 registry 的可信 CA 添加到集群的镜像配置对象中，作为 Cluster Samples Operator 配置的一部分，请执行以下步骤：

a. 创建集群的镜像配置对象：

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

b. 在集群的镜像配置对象中，为镜像添加所需的可信 CA：

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

2. 从您的安装有效负载中导入默认 must-gather 镜像：

```
$ oc import-image is/must-gather -n openshift
```

在运行 **oc adm must-gather** 命令时，请使用 **--image** 标志并指向有效负载镜像，如下例所示：

```
$ oc adm must-gather --image=$(oc adm release info --image-for must-gather)
```

8.17. 配置定期导入 CLUSTER SAMPLE OPERATOR 镜像流标签

您可以在新版本可用时定期导入镜像流标签，以确保始终可以访问 Cluster Sample Operator 镜像的最新版本。

流程

1. 运行以下命令，获取 **openshift** 命名空间中的所有镜像流：

```
oc get imagestreams -nopenshift
```

2. 运行以下命令，获取 **openshift** 命名空间中的每个镜像流的标签：

```
$ oc get is <image-stream-name> -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}
{'\n'}{end}" -nopenshift
```

例如：

```
$ oc get is ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}{'\n'}
{end}" -nopenshift
```

输**出示例**

```
1.11 registry.access.redhat.com/ubi8/openjdk-17:1.11
1.12 registry.access.redhat.com/ubi8/openjdk-17:1.12
```

3. 运行以下命令，调度镜像流中存在的每个标签的镜像定期导入：

```
$ oc tag <repository/image> <image-stream-name:tag> --scheduled -nopenshift
```

例如：

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.11 ubi8-openjdk-17:1.11 --scheduled -
```

```
nopenshift
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.12 ubi8-openjdk-17:1.12 --scheduled -
nopenshift
```

该命令可使 OpenShift Container Platform 定期更新该特定镜像流标签。此周期是集群范围的设置，默认设为 15 分钟。

4. 运行以下命令，验证定期导入的调度状态：

```
oc get imagestream <image-stream-name> -o jsonpath="{range .spec.tags[*]}Tag: {.name}
{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

例如：

```
oc get imagestream ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}Tag: {.name}
{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

输**出示例**

```
Tag: 1.11 Scheduled: true
Tag: 1.12 Scheduled: true
```

第 9 章 安装后的节点任务

安装 OpenShift Container Platform 后，您可以通过某些节点任务进一步根据要求扩展和自定义集群。

9.1. 在 OPENSHIFT CONTAINER PLATFORM 集群中添加 RHEL 计算机器

理解并使用 RHEL 计算节点。

9.1.1. 关于在集群中添加 RHEL 计算节点

在 OpenShift Container Platform 4.15 中，如果 **x86_64** 架构中使用用户置备或安装程序置备的基础架构安装，您可以选择将 Red Hat Enterprise Linux(RHEL)机器用作集群中的计算机器。您必须使用 Red Hat Enterprise Linux CoreOS (RHCOS) 机器作为集群中的控制平面机器。

如果您在集群中使用 RHEL 计算机器，则需要自己负责所有操作系统生命周期的管理和维护任务。您必须执行系统更新、应用补丁并完成所有其他必要的任务。

对于安装程序置备的基础架构集群，您必须手动添加 RHEL 计算机器，因为安装程序置备的基础架构集群中的自动扩展会默认添加 Red Hat Enterprise Linux CoreOS (RHCOS) 计算机器。

重要

- 由于从集群中的机器上删除 OpenShift Container Platform 需要破坏操作系统，因此您必须对添加到集群中的所有 RHEL 机器使用专用的硬件。
- 添加到 OpenShift Container Platform 集群的所有 RHEL 机器上都禁用内存交换功能。您无法在这些机器上启用交换内存。

您必须在初始化 control plane 之后将所有 RHEL 计算机器添加到集群。

9.1.2. RHEL 计算节点的系统要求

OpenShift Container Platform 环境中的 Red Hat Enterprise Linux(RHEL)计算机器主机必须满足以下最低硬件规格和系统级别要求：

- 您的红帽帐户必须具有有效的 OpenShift Container Platform 订阅。如果没有，请与您的销售代表联系以了解更多信息。
- 生产环境必须提供能支持您的预期工作负载的计算机器。作为集群管理员，您必须计算预期的工作负载，再加上大约 10% 的开销。对于生产环境，请分配足够的资源，以防止节点主机故障影响您的最大容量。
- 所有系统都必须满足以下硬件要求：
  - 物理或虚拟系统，或在公有或私有 IaaS 上运行的实例。
  - 基础操作系统：使用 "Minimal" 安装选项的 RHEL 8.6 及之后的版本。

重要

不支持将 RHEL 7 计算机器添加到 OpenShift Container Platform 集群。

如果您有在以前的 OpenShift Container Platform 版本中支持的 RHEL 7 计算机器，则无法将其升级到 RHEL 8。您必须部署新的 RHEL 8 主机，并且应该删除旧的 RHEL 7 主机。如需更多信息，请参阅"删除节点"部分。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中*已弃用和删除的功能*部分。

- 如果以 FIPS 模式部署 OpenShift Container Platform，则需要在 RHEL 机器上启用 FIPS，然后才能引导它。请参阅 RHEL 8 文档中的启用 FIPS 模式安装 RHEL 8 系统 。

重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

- NetworkManager 1.0 或更高版本。
- 1 个 vCPU。
- 最小 8 GB RAM。
- 最小15 GB 硬盘空间，用于包含 **/var/** 的文件系统。
- 最小1 GB 硬盘空间，用于包含 **/usr/local/bin/** 的文件系统。
- 最小 1 GB 硬盘空间，用于包含其临时目录的文件系统。临时系统目录根据 Python 标准库中 tempfile 模块中定义的规则确定。
  - 每个系统都必须满足您的系统提供商的任何其他要求。例如，如果在 VMware vSphere 上安装了集群，必须根据其存储准则配置磁盘，而且必须设置 **disk.enableUUID=true** 属性。
  - 每个系统都必须能够使用 DNS 可解析的主机名访问集群的 API 端点。任何现有的网络安全访问控制都必须允许系统访问集群的 API 服务端点。

其他资源

- 删除节点

9.1.2.1. 证书签名请求管理

在使用您置备的基础架构时，集群只能有限地访问自动机器管理，因此您必须提供一种在安装后批准集群证书签名请求 (CSR) 的机制。**kube-controller-manager** 只能批准 kubelet 客户端 CSR。 **machine-approver** 无法保证使用 kubelet 凭证请求的提供证书的有效性，因为它不能确认是正确的机器发出了该请求。您必须决定并实施一种方法，以验证 kubelet 提供证书请求的有效性并进行批准。

9.1.3. 准备机器以运行 Playbook

在将使用 Red Hat Enterprise Linux(RHEL)作为操作系统的计算机器添加到 OpenShift Container Platform 4.15 集群之前，必须准备一个 RHEL 8 机器，以运行向集群添加新节点的 Ansible playbook。这台机器不是集群的一部分，但必须能够访问集群。

先决条件

- 在运行 playbook 的机器上安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 确保用于安装集群的 **kubeconfig** 文件和用于安装集群的安装程序位于 RHEL 8 机器中。若要实现这一目标，一种方法是使用安装集群时所用的同一台机器。
2. 配置机器，以访问您计划用作计算机器的所有 RHEL 主机。您可以使用公司允许的任何方法，包括使用 SSH 代理或 VPN 的堡垒主机。
3. 在运行 playbook 的机器上配置一个用户，该用户对所有 RHEL 主机具有 SSH 访问权限。

重要

如果使用基于 SSH 密钥的身份验证，您必须使用 SSH 代理来管理密钥。

4. 如果还没有这样做，请使用 RHSM 注册机器，并为它附加一个带有 **OpenShift** 订阅的池：
   a. 使用 RHSM 注册机器：

```
# subscription-manager register --username=<user_name> --password=<password>
```

   b. 从 RHSM 获取最新的订阅数据：

```
# subscription-manager refresh
```

   c. 列出可用的订阅：

```
# subscription-manager list --available --matches '*OpenShift*'
```

   d. 在上一命令的输出中，找到 OpenShift Container Platform 订阅的池 ID 并附加该池：

```
# subscription-manager attach --pool=<pool_id>
```

5. 启用 OpenShift Container Platform 4.15 所需的存储库:

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.15-for-rhel-8-x86_64-rpms"
```

6. 安装所需的软件包，包括 **openshift-ansible**:

```
# yum install openshift-ansible openshift-clients jq
```

openshift-ansible 软件包提供了安装实用程序，并且会拉取将 RHEL 计算节点添加到集群所需要的其他软件包，如 Ansible、playbook 和相关的配置文件。**openshift-clients** 提供 **oc** CLI，**jq** 软件包则可改善命令行中 JSON 输出的显示。

9.1.4. 准备 RHEL 计算节点

在将 Red Hat Enterprise Linux (RHEL) 机器添加到 OpenShift Container Platform 集群之前，您必须将每台主机注册到 Red Hat Subscription Manager (RHSM)，为其附加有效的 OpenShift Container Platform 订阅，并且启用所需的存储库。

1. 在每一主机上进行 RHSM 注册：

```
# subscription-manager register --username=<user_name> --password=<password>
```

2. 从 RHSM 获取最新的订阅数据：

```
# subscription-manager refresh
```

3. 列出可用的订阅：

```
# subscription-manager list --available --matches '*OpenShift*'
```

4. 在上一命令的输出中，找到 OpenShift Container Platform 订阅的池 ID 并附加该池：

```
# subscription-manager attach --pool=<pool_id>
```

5. 禁用所有 yum 存储库：

   a. 禁用所有已启用的 RHSM 存储库：

   ```
   # subscription-manager repos --disable="*"
   ```

   b. 列出剩余的 yum 存储库，并记录它们在 **repo id** 下的名称（若有）：

   ```
   # yum repolist
   ```

   c. 使用 **yum-config-manager** 禁用剩余的 yum 存储库：

   ```
   # yum-config-manager --disable <repo_id>
   ```

   或者，禁用所有存储库：

   ```
   # yum-config-manager --disable \*
   ```

   请注意，有大量可用存储库时可能需要花费几分钟

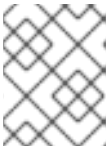
6. 仅启用 OpenShift Container Platform 4.15 需要的存储库：

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.15-for-rhel-8-x86_64-rpms" \
    --enable="fast-datapath-for-rhel-8-x86_64-rpms"
```

-

7. 停止并禁用主机上的防火墙：

```
# systemctl disable --now firewalld.service
```



注意

请不要在以后启用防火墙。如果这样做，则无法访问 worker 上的 OpenShift Container Platform 日志。

9.1.5. 在集群中添加 RHEL 计算机器

您可以将使用 Red Hat Enterprise Linux 作为操作系统的计算机器添加到 OpenShift Container Platform 4.15 集群中。

先决条件

- 运行 playbook 的机器上已安装必需的软件包并且执行了必要的配置。
- RHEL 主机已做好安装准备。

流程

在为运行 playbook 而准备的机器上执行以下步骤：

1. 创建一个名为 **/<path>/inventory/hosts** 的 Ansible 清单文件，以定义您的计算机器主机和必要的变量：

```
[all:vars]
ansible_user=root (1)
#ansible_become=True (2)

openshift_kubeconfig_path="~/.kube/config" (3)

[new_workers] (4)
mycluster-rhel8-0.example.com
mycluster-rhel8-1.example.com
```

(1) 指定要在远程计算机器上运行 Ansible 任务的用户名。

(2) 如果不将 **root** 指定为 **ansible_user**，您必须将 **ansible_become** 设置为 **True**，并为该用户分配 sudo 权限。

(3) 指定集群的 **kubeconfig** 文件的路径和文件名。

(4) 列出要添加到集群中的每台 RHEL 机器。必须为每个主机提供完全限定域名。此名称是集群用来访问机器的主机名，因此请设置用于访问机器的正确公共或私有名称。

2. 进入到 Ansible playbook 目录：

```
$ cd /usr/share/ansible/openshift-ansible
```

3. 运行 playbook：

-

```
$ ansible-playbook -i /<path>/inventory/hosts playbooks/scaleup.yml 1
```

1 对于**<path>**，指定您创建的Ansible库存文件的路径。

9.1.6. Ansible hosts 文件的必要参数

在将 Red Hat Enterprise Linux (RHEL) 计算机器添加到集群之前，必须在 Ansible hosts 文件中定义以下参数。

参数	描述	值
ansible_user	能够以免密码方式进行 SSH 身份验证的 SSH 用户。如果使用基于 SSH 密钥的身份验证，则必须使用 SSH 代理来管理密钥。	系统上的用户名。默认值为 **root**。
ansible_become	如果 **ansible_user** 的值不是 root，您必须将 **ansible_become** 设置为 **True**，并且您指定为 **ansible_user** 的用户必须配置有免密码 sudo 访问权限。	**True**。如果值不是 **True**，请不要指定和定义此参数。
openshift_kubeconfig_path	指定包含集群的 **kubeconfig** 文件的本地目录的路径和文件名。	配置文件的路径和名称。

9.1.7. 可选：从集群中删除 RHCOS 计算机器

将 Red Hat Enterprise Linux（RHEL）计算机器添加到集群后，您可以选择性地删除 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器来释放资源。

先决条件

- 您已将 RHEL 计算机器添加到集群中。

流程

1. 查看机器列表并记录 RHCOS 计算机器的节点名称：

```
$ oc get nodes -o wide
```

2. 对于每一台 RHCOS 计算机器，删除其节点：
   a. 通过运行 **oc adm cordon** 命令，将节点标记为不可调度：

```
$ oc adm cordon <node_name> 1
```

1 指定其中一台 RHCOS 计算机器的节点名称。

   b. 清空节点中的所有 Pod：

```
$ oc adm drain <node_name> --force --delete-emptydir-data --ignore-daemonsets 1
```

1 指定您隔离的 RHCOS 计算机器的节点名称。

c. 删除节点：

```
$ oc delete nodes <node_name> 1
```

1 指定您清空的 RHCOS 计算机器的节点名称。

3. 查看计算机器的列表，以确保仅保留 RHEL 节点：

```
$ oc get nodes -o wide
```

4. 从集群的计算机器的负载均衡器中删除 RHCOS 机器。您可以删除虚拟机或重新制作 RHCOS 计算机器物理硬件的镜像。

9.2. 将 RHCOS 计算机器添加到 OPENSHIFT CONTAINER PLATFORM 集群

您可以在裸机上的 OpenShift Container Platform 集群中添加更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器。

在将更多计算机器添加到在裸机基础架构上安装的集群之前，必须先创建 RHCOS 机器供其使用。您可以使用 ISO 镜像或网络 PXE 引导来创建机器。

9.2.1. 先决条件

- 您在裸机上安装了集群。
- 您有用来创建集群的安装介质和 Red Hat Enterprise Linux CoreOS（RHCOS）镜像。如果您没有这些文件，可以按照安装过程的说明获得这些文件。

9.2.2. 使用 ISO 镜像创建 RHCOS 机器

您可以使用 ISO 镜像为裸机集群创建更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器，以创建机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 已安装 OpenShift CLI (**oc**)。

流程

1. 运行以下命令，从集群中删除 Ignition 配置文件：

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. 将您从集群导出的 **worker.ign** Ignition 配置文件上传到 HTTP 服务器。注意这些文件的 URL。

3. 您可以验证 ignition 文件是否在 URL 上可用。以下示例获取计算节点的 Ignition 配置文件：

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. 您可以运行以下命令来访问 ISO 镜像以引导新机器：

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 使用 ISO 文件在更多计算机器上安装 RHCOS。在安装集群前，使用创建机器时使用的相同方法：

   - 将 ISO 镜像刻录到磁盘并直接启动。
   - 在 LOM 接口中使用 ISO 重定向。

6. 在不指定任何选项或中断实时引导序列的情况下引导 RHCOS ISO 镜像。等待安装程序在 RHCOS live 环境中引导进入 shell 提示符。

注意

您可以中断 RHCOS 安装引导过程来添加内核参数。但是，在这个 ISO 过程中，您应该使用以下步骤中所述的 **coreos-installer** 命令，而不是添加内核参数。

7. 运行 **coreos-installer** 命令并指定满足您的安装要求的选项。您至少必须指定指向节点类型的 Ignition 配置文件的 URL，以及您要安装到的设备：

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

1. 您必须使用 **sudo** 运行 **coreos-installer** 命令，因为 **core** 用户没有执行安装所需的 root 权限。
2. 当 Ignition 配置文件通过 HTTP URL 获取时，需要 **--ignition-hash** 选项来验证集群节点上 Ignition 配置文件的真实性。**<digest>** 是上一步中获取的 Ignition 配置文件 SHA512 摘要。

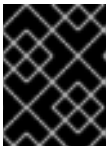
注意

如果要通过使用 TLS 的 HTTPS 服务器提供 Ignition 配置文件，您可以在运行 **coreos-installer** 前将内部证书颁发机构(CA)添加到系统信任存储中。

以下示例将引导节点安装初始化到 **/dev/sda** 设备。bootstrap 节点的 Ignition 配置文件从 IP 地址 192.168.1.2 的 HTTP Web 服务器获取：

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. 在机器的控制台上监控 RHCOS 安装的进度。



重要

在开始安装 OpenShift Container Platform 之前，确保每个节点中安装成功。观察安装过程可以帮助确定可能会出现 RHCOS 安装问题的原因。

9. 继续为集群创建更多计算机器。

9.2.3. 通过 PXE 或 iPXE 启动来创建 RHCOS 机器

您可以使用 PXE 或 iPXE 引导为裸机集群创建更多 Red Hat Enterprise Linux CoreOS（RHCOS）计算机器。

先决条件

- 获取集群计算机器的 Ignition 配置文件的 URL。在安装过程中将该文件上传到 HTTP 服务器。
- 获取您在集群安装过程中上传到 HTTP 服务器的 RHCOS ISO 镜像、压缩的裸机 BIOS、**kernel** 和 **initramfs** 文件的 URL。
- 您可以访问在安装过程中为 OpenShift Container Platform 集群创建机器时使用的 PXE 引导基础架构。机器必须在安装 RHCOS 后从本地磁盘启动。
- 如果使用 UEFI，您可以访问在 OpenShift Container Platform 安装过程中修改的 **grub.conf** 文件。

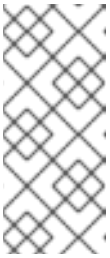
流程

1. 确认 RHCOS 镜像的 PXE 或 iPXE 安装正确。
   - 对于 PXE：

```
DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> (1)
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img (2)
```

(1) 指定上传到 HTTP 服务器的 live **kernel** 文件位置。

(2) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**initrd** 参数值是 live **initramfs** 文件的位置，**coreos.inst.ignition_url** 参数值是 worker Ignition 配置文件的位置，**coreos.live.rootfs_url** 参数值是 live **rootfs** 文件的位置。**coreos.inst.ignition_url** 和 **coreos.live.rootfs_url** 参数仅支持 HTTP 和 HTTPS。



注意

此配置不会在图形控制台的机器上启用串行控制台访问。要配置不同的控制台，请在 **APPEND** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅 如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？

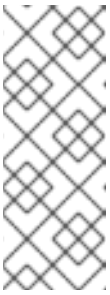
- 对于 iPXE (**x86_64** + **aarch64**)：

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img (3)
boot
```

(1) 指定上传到 HTTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 **kernel** 文件的位置，在 UEFI 系统中引导时需要 **initrd=main** 参数，**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位置，**coreos.inst.ignition_url** 参数值则是 worker Ignition 配置文件的位置。

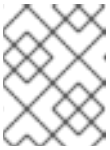
(2) 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。

(3) 指定上传到 HTTP 服务器的 **initramfs** 文件的位置。



注意

此配置不会在使用图形控制台配置不同的控制台的机器上启用串行控制台访问，请在 **kernel** 行中添加一个或多个 **console=** 参数。例如，添加 **console=tty0 console=ttyS0** 以将第一个 PC 串口设置为主控制台，并将图形控制台设置为二级控制台。如需更多信息，请参阅如何在 Red Hat Enterprise Linux 中设置串行终端和/或控制台？ 和"启用 PXE 和 ISO 安装的串行控制台"部分。



注意

要在 **aarch64** 架构中网络引导 CoreOS **内核**，您需要使用启用了 **IMAGE_GZIP** 选项的 iPXE 构建版本。请参阅 iPXE 中的**IMAGE_GZIP** 选项。

- 对于 **aarch64** 中的 PXE（使用 UEFI 和 GRUB 作为第二阶段）：

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
    initrd rhcos-<version>-live-initramfs.<architecture>.img (3)
}
```

1. 指定上传到 HTTP/TFTP 服务器的 RHCOS 文件的位置。**kernel** 参数值是 TFTP 服务器中的 **kernel** 文件的位置。**coreos.live.rootfs_url** 参数值是 **rootfs** 文件的位
2. 如果您使用多个 NIC，请在 **ip** 选项中指定一个接口。例如，要在名为 **eno1** 的 NIC 上使用 DHCP，请设置 **ip=eno1:dhcp**。
3. 指定上传到 TFTP 服务器的 **initramfs** 文件的位置。

2. 使用 PXE 或 iPXE 基础架构为集群创建所需的计算机器。

9.2.4. 批准机器的证书签名请求

当您将机器添加到集群时，会为您添加的每台机器生成两个待处理证书签名请求(CSR)。您必须确认这些 CSR 已获得批准，或根据需要自行批准。必须首先批准客户端请求，然后批准服务器请求。

先决条件

- 您已将机器添加到集群中。

流程

1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输**出示例**

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.28.5
master-1 Ready    master  63m  v1.28.5
master-2 Ready    master  64m  v1.28.5
```

输出中列出了您创建的所有机器。

注意

在有些 CSR 被批准前，前面的输出可能不包括计算节点（也称为 worker 节点）。

2. 检查待处理的 CSR，并确保添加到集群中的每台机器都有 **Pending** 或 **Approved** 状态的客户端请求：

```
$ oc get csr
```

输**出示例**

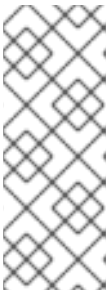
```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper   Pending
...
```

在本例中，两台机器加入集群。您可能会在列表中看到更多已批准的 CSR。

3. 如果 CSR 没有获得批准，在您添加的机器的所有待处理 CSR 都处于 **Pending 状态** 后，请批准集群机器的 CSR：

注意

由于 CSR 会自动轮转，因此请在将机器添加到集群后一小时内批准您的 CSR。如果没有在一小时内批准它们，证书将会轮转，每个节点会存在多个证书。您必须批准所有这些证书。批准客户端 CSR 后，Kubelet 为服务证书创建一个二级 CSR，这需要手动批准。然后，如果 Kubelet 请求具有相同参数的新证书，则后续提供证书续订请求由 **machine-approver** 自动批准。



注意

对于在未启用机器 API 的平台上运行的集群，如裸机和其他用户置备的基础架构，您必须实施一种方法来自动批准 kubelet 提供证书请求(CSR)。如果没有批准请求，则 **oc exec**、oc**rsh** 和 **oc logs** 命令将无法成功，因为 API 服务器连接到 kubelet 时需要服务证书。与 Kubelet 端点联系的任何操作都需要此证书批准。该方法必须监视新的 CSR，确认 CSR 由 system: **node** 或 **system:admin** 组中的 **node-bootstrapper** 服务帐户提交，并确认节点的身份。

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

注意

在有些 CSR 被批准前，一些 Operator 可能无法使用。

4. 现在，您的客户端请求已被批准，您必须查看添加到集群中的每台机器的服务器请求：

```
$ oc get csr
```

输**出示例**

```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-bfd72   5m26s  system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv   5m26s  system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. 如果剩余的 CSR 没有被批准，且处于 **Pending** 状态，请批准集群机器的 CSR：

- 要单独批准，请对每个有效的 CSR 运行以下命令：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 要批准所有待处理的 CSR，请运行以下命令：

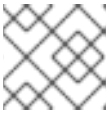
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 批准所有客户端和服务器 CSR 后，机器将 处于 **Ready 状态**。运行以下命令验证：

```
$ oc get nodes
```

输**出示例**

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  73m  v1.28.5
master-1  Ready     master  73m  v1.28.5
master-2  Ready     master  74m  v1.28.5
worker-0  Ready     worker  11m  v1.28.5
worker-1  Ready     worker  11m  v1.28.5
```



注意

批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready 状态**。

其他信息

- 如需有关 CSR 的更多信息，请参阅 证书签名请求。

9.2.5. 在 AWS 中使用自定义 /var 分区添加新的 RHCOS worker 节点

OpenShift Container Platform 使用 bootstrap 过程中完成的机器配置支持在安装过程中分区设备。但是，如果您使用 **/var** 分区，该设备名称必须在安装时确定，且不可更改。如果节点具有不同的设备命名模式，则无法将不同的实例类型添加为节点。例如，如果您将 **/var** 分区配置为 **m4.large** 实例的默认 AWS 设备名称，**dev/xvdb**，则无法直接添加 AWS **m5.large** 实例，因为 **m5.large** 实例默认使用 **/dev/nvme1n1** 设备。由于不同的命名模式，设备可能无法分区。

本节中的步骤演示了如何使用与安装时配置的不同设备名称的实例添加新的 Red Hat Enterprise Linux CoreOS(RHCOS)计算节点。您可以创建自定义用户数据 secret 并配置新的计算机器集。这些步骤特定于 AWS 集群。其原则也适用于其他云部署。但是，针对其他部署的设备命名模式会有所不同，应该根据具体情况进行决定。

流程

1. 在命令行中更改为 **openshift-machine-api** 命名空间：

```
$ oc project openshift-machine-api
```

2. 从 **worker-user-data** secret 创建新 secret：

a. 将 secret 的 **userData** 部分导出到文本文件：

```
$ oc get secret worker-user-data --template='{{index .data.userData | base64decode}}' |
jq > userData.txt
```

b. 编辑文本文件，为要用于新节点的分区添加 **storage**, **filesystems**, 和 **systemd** 小节。您可以根据需要指定任何 Ignition 配置参数。

注意

不要更改 **ignition** 小节中的值。

```
{
  "ignition": {
    "config": {
      "merge": [
        {
          "source": "https:...."
        }
      ]
    },
    "security": {
      "tls": {
        "certificateAuthorities": [
          {
            "source": "data:text/plain;charset=utf-8;base64,.....=="
          }
        ]
      }
    },
    "version": "3.2.0"
  },
  "storage": {
    "disks": [
      {
        "device": "/dev/nvme1n1", (1)
        "partitions": [
          {
            "label": "var",
            "sizeMiB": 50000, (2)
            "startMiB": 0 (3)
          }
        ]
      }
    ],
    "filesystems": [
      {
        "device": "/dev/disk/by-partlabel/var", (4)
        "format": "xfs", (5)
        "path": "/var" (6)
      }
    ]
  },
  "systemd": {
```

```
    "units": [ 7
      {
        "contents": "[Unit]\nBefore=local-
fs.target\n[Mount]\nWhere=/var\nWhat=/dev/disk/by-
partlabel/var\nOptions=defaults,pquota\n[Install]\nWantedBy=local-fs.target\n",
        "enabled": true,
        "name": "var.mount"
      }
    ]
  }
}
```

1. 指定 AWS 块设备的绝对路径。
2. 指定以兆字节为单位的数据分区大小。
3. 指定以兆字节为单位的分区的开头。当在引导磁盘中添加数据分区时，推荐最少使用 25000 MB(Mebibytes)。root 文件系统会自动调整大小以填充所有可用空间（最多到指定的偏移值）。如果没有指定值，或者指定的值小于推荐的最小值，则生成的 root 文件系统会太小，而在以后进行的 RHCOS 重新安装可能会覆盖数据分区的开始部分。
4. 指定到 **/var** 分区的绝对路径。
5. 指定文件系统格式。
6. 指定文件系统的挂载点，而 Ignition 运行时相对于根文件系统挂载的位置。这不一定与应在真实 root 中挂载的位置相同，但鼓励使其相同的位置。
7. 定义将 **/dev/disk/by-partlabel/var** 设备挂载到 **/var** 分区的 systemd 挂载单元。

c. 将 **disableTemplating** 部分从 **work-user-data** secret 提取到文本文件：

```
$ oc get secret worker-user-data --template='{{index .data.disableTemplating |
base64decode}}' | jq > disableTemplating.txt
```

d. 从两个文本文件创建新用户数据机密文件。此用户数据 secret 将 **userData.txt** 文件中的额外节点分区信息传递给新创建的节点。

```
$ oc create secret generic worker-user-data-x5 --from-file=userData=userData.txt --
from-file=disableTemplating=disableTemplating.txt
```

3. 为新节点创建新计算机器集：

a. 创建新的计算机器集 YAML 文件，类似于为 AWS 配置的文件。添加所需分区和新创建的用户数据 secret：

提示

使用现有计算机器集作为模板，并根据需要更改新节点的参数。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
```

```
    machine.openshift.io/cluster-api-cluster: auto-52-92tf4
  name: worker-us-east-2-nvme1n1 (1)
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: auto-52-92tf4
      machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: auto-52-92tf4
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
    spec:
      metadata: {}
      providerSpec:
        value:
          ami:
            id: ami-0c2dbd95931a
          apiVersion: awsproviderconfig.openshift.io/v1beta1
          blockDevices:
          - DeviceName: /dev/nvme1n1 (2)
            ebs:
              encrypted: true
              iops: 0
              volumeSize: 120
              volumeType: gp2
          - DeviceName: /dev/nvme1n2 (3)
            ebs:
              encrypted: true
              iops: 0
              volumeSize: 50
              volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: auto-52-92tf4-worker-profile
          instanceType: m6i.large
          kind: AWSMachineProviderConfig
          metadata:
            creationTimestamp: null
          placement:
            availabilityZone: us-east-2b
            region: us-east-2
          securityGroups:
          - filters:
            - name: tag:Name
              values:
              - auto-52-92tf4-worker-sg
          subnet:
            id: subnet-07a90e5db1
          tags:
```

```
      - name: kubernetes.io/cluster/auto-52-92tf4
        value: owned
      userDataSecret:
        name: worker-user-data-x5 4
```

1. 为新节点指定名称。
2. 指定到 AWS 块设备的绝对路径，这里是一个加密的 EBS 卷。
3. 可选。指定附加 EBS 卷。
4. 指定用户数据 secret 文件。

b. 创建计算机器集：

```
$ oc create -f <file-name>.yaml
```

机器可能需要一些时间才可用。

4. 验证新分区和节点是否已创建：

a. 验证是否已创建计算机器设置：

```
$ oc get machineset
```

输出示例

```
NAME                                 DESIRED   CURRENT   READY   AVAILABLE   AGE
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1a   1         1         1       1           124m
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1b   2         2         2       2           124m
worker-us-east-2-nvme1n1             1         1         1       1           2m35s 1
```

1. 这是新的计算机器集。

b. 验证新节点是否已创建：

```
$ oc get nodes
```

输出示例

```
NAME                         STATUS   ROLES    AGE     VERSION
ip-10-0-128-78.ec2.internal    Ready    worker   117m    v1.28.5
ip-10-0-146-113.ec2.internal   Ready    master   127m    v1.28.5
ip-10-0-153-35.ec2.internal    Ready    worker   118m    v1.28.5
ip-10-0-176-58.ec2.internal    Ready    master   126m    v1.28.5
ip-10-0-217-135.ec2.internal   Ready    worker   2m57s   v1.28.5 1
ip-10-0-225-248.ec2.internal   Ready    master   127m    v1.28.5
ip-10-0-245-59.ec2.internal    Ready    worker   116m    v1.28.5
```

1. 这是新节点。

c. 验证新节点上是否创建了自定义 **/var** 分区：

```
$ oc debug node/<node-name> -- chroot /host lsblk
```

例如：

```
$ oc debug node/ip-10-0-217-135.ec2.internal -- chroot /host lsblk
```

输**出示例**

```
NAME        MAJ:MIN  RM  SIZE RO TYPE MOUNTPOINT
nvme0n1     202:0    0   120G  0 disk
|-nvme0n1p1 202:1    0     1M  0 part
|-nvme0n1p2 202:2    0   127M  0 part
|-nvme0n1p3 202:3    0   384M  0 part /boot
`-nvme0n1p4 202:4    0 119.5G  0 part /sysroot
nvme1n1     202:16   0    50G  0 disk
`-nvme1n1p1 202:17   0  48.8G  0 part /var (1)
```

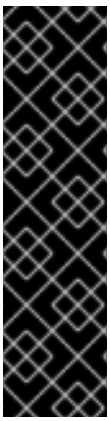
(1) **nvme1n1** 设备被挂载到 **/var** 分区。

其他资源

- 如需有关 OpenShift Container Platform 如何使用磁盘分区的更多信息，请参阅 磁盘分区。

9.3. 部署机器健康检查

理解并部署机器健康检查。



重要

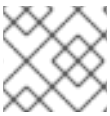
您只能在 Machine API 操作的集群中使用高级机器管理和扩展功能。具有用户置备的基础架构的集群需要额外的验证和配置才能使用 Machine API。

具有基础架构平台类型 **none** 的集群无法使用 Machine API。即使附加到集群的计算机器安装在支持该功能的平台上，也会应用这个限制。在安装后无法更改此参数。

要查看集群的平台类型，请运行以下命令：

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

9.3.1. 关于机器健康检查



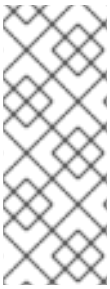
注意

您只能对由计算机器集或 control plane 机器集管理的机器应用机器健康检查。

要监控机器的健康状况，创建资源来定义控制器的配置。设置要检查的条件（例如，处于 **NotReady** 状态达到五分钟或 node-problem-detector 中显示了持久性状况），以及用于要监控的机器集合的标签。

监控 **MachineHealthCheck** 资源的控制器会检查定义的条件。如果机器无法进行健康检查，则会自动删除机器并创建一个机器来代替它。删除机器之后，您会看到**机器被删除**事件。

为限制删除机器造成的破坏性影响，控制器一次仅清空并删除一个节点。如果目标机器池中不健康的机器池中不健康的机器数量大于 **maxUnhealthy** 的值，则补救会停止，需要启用手动干预。



注意

请根据工作负载和要求仔细考虑超时。

- 超时时间较长可能会导致不健康的机器上的工作负载长时间停机。
- 超时时间太短可能会导致补救循环。例如，检查 **NotReady** 状态的超时时间必须足够长，以便机器能够完成启动过程。

要停止检查，请删除资源。

9.3.1.1. 部署机器健康检查时的限制

部署机器健康检查前需要考虑以下限制：

- 只有机器集拥有的机器才可以由机器健康检查修复。
- 如果机器的节点从集群中移除，机器健康检查会认为机器不健康，并立即修复机器。
- 如果机器对应的节点在 **nodeStartupTimeout** 之后没有加入集群，则会修复机器。
- 如果 **Machine** 资源阶段为 **Failed**，则会立即修复机器。

其他资源

- 关于 control plane 机器集

9.3.2. MachineHealthCheck 资源示例

所有基于云的安装类型的 **MachineHealthCheck** 资源，以及裸机以外的资源，类似以下 YAML 文件：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role> 3
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" 5
    status: "False"
  - type:    "Ready"
    timeout: "300s" 6
```

```
  status: "Unknown"
maxUnhealthy: "40%" 7
nodeStartupTimeout: "10m" 8
```

1 指定要部署的机器健康检查的名称。

2 3 为要检查的机器池指定一个标签。

4 以 **<cluster_name>-<label>-<zone>** 格式 指定要跟踪的机器集。例如， **prod-node-us-east-1a**。

5 6 指定节点条件的超时持续时间。如果在超时时间内满足了条件，则会修复机器。超时时间较长可能会导致不健康的机器上的工作负载长时间停机。

7 指定目标池中允许同时修复的机器数量。这可设为一个百分比或一个整数。如果不健康的机器数量超过 **maxUnhealthy** 设定的限制，则不会执行补救。

8 指定机器健康检查在决定机器不健康前必须等待节点加入集群的超时持续时间。

注意

matchLabels 只是示例; 您必须根据具体需要映射您的机器组。

9.3.2.1. 短路机器健康检查补救

短路可确保仅在集群健康时机器健康检查修复机器。通过 **MachineHealthCheck** 资源中的 **maxUnhealthy** 字段配置短路。

如果用户在修复任何机器前为 **maxUnhealthy** 字段定义了一个值，**MachineHealthCheck** 会将 **maxUnhealthy** 的值与它决定不健康的目标池中的机器数量进行比较。如果不健康的机器数量超过 **maxUnhealthy** 限制，则不会执行补救。

重要

如果没有设置 **maxUnhealthy**，则默认值为 **100%**，无论集群状态如何，机器都会被修复。

适当的 **maxUnhealthy** 值取决于您部署的集群规模以及 **MachineHealthCheck** 覆盖的机器数量。例如，您可以使用 **maxUnhealthy** 值覆盖多个可用区间的多个计算机器集，以便在丢失整个区时，**maxUnhealthy** 设置可以在集群中防止进一步补救。在没有多个可用区的全局 Azure 区域，您可以使用可用性集来确保高可用性。

重要

如果您为 control plane 配置 **MachineHealthCheck** 资源，请将 **maxUnhealthy** 的值设置为 **1**。

此配置可确保当多个 control plane 机器显示为不健康时，机器健康检查不会采取任何操作。多个不健康的 control plane 机器可能会表示 etcd 集群已降级或扩展操作来替换失败的机器。

如果 etcd 集群降级，可能需要手动干预。如果扩展操作正在进行，机器健康检查应该允许它完成。

maxUnhealthy 字段可以设置为整数或百分比。根据 **maxUnhealthy** 值，有不同的补救实现。

9.3.2.1.1. 使用绝对值设置 maxUnhealthy

如果将 **maxUnhealthy** 设为 **2**:

- 如果 2 个或更少节点不健康，则可执行补救
- 如果 3 个或更多节点不健康，则不会执行补救

这些值与机器健康检查要检查的机器数量无关。

9.3.2.1.2. 使用百分比设置 maxUnhealthy

如果 **maxUnhealthy** 被设置为 **40%**，有 25 个机器被检查：

- 如果有 10 个或更少节点处于不健康状态，则可执行补救
- 如果 11 个或多个节点不健康，则不会执行补救

如果 **maxUnhealthy** 被设置为 **40%**，有 6 个机器被检查：

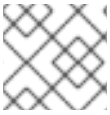
- 如果 2 个或更少节点不健康，则可执行补救
- 如果 3 个或更多节点不健康，则不会执行补救

注意

当被检查的 **maxUnhealthy** 机器的百分比不是一个整数时，允许的机器数量会被舍入到一个小的整数。

9.3.3. 创建机器健康检查资源

您可以为集群中的机器集创建 **MachineHealthCheck** 资源。



注意

您只能对由计算机器集或 control plane 机器集管理的机器应用机器健康检查。

先决条件

- 安装 **oc** 命令行界面。

流程

1. 创建一个 **healthcheck.yml** 文件，其中包含您的机器健康检查的定义。
2. 将 **healthcheck.yml** 文件应用到您的集群：

```
$ oc apply -f healthcheck.yml
```

9.3.4. 手动扩展计算机器集

要在计算机器集中添加或删除机器实例，您可以手动扩展计算机器集。

这个指南与全自动的、安装程序置备的基础架构安装相关。自定义的、用户置备的基础架构安装没有计算机器集。

先决条件

- 安装 OpenShift Container Platform 集群和 **oc** 命令行。
- 以具有 **cluster-admin** 权限的用户身份登录 **oc**。

流程

1. 运行以下命令，查看集群中的计算机器：

```
$ oc get machinesets -n openshift-machine-api
```

计算机器集以 **<clusterid>-worker-<aws-region-az>** 的形式列出。

2. 运行以下命令，查看集群中的计算机器：

```
$ oc get machine -n openshift-machine-api
```

3. 运行以下命令，在要删除的计算机器上设置注解：

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/delete-machine="true"
```

4. 运行以下命令来扩展计算机器集：

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

或者：

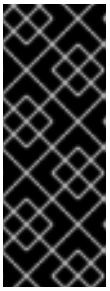
```
$ oc edit machineset <machineset> -n openshift-machine-api
```

提示

您还可以应用以下 YAML 来扩展计算机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

您可以扩展或缩减计算机器。需要过几分钟以后新机器才可用。



重要

默认情况下，机器控制器会尝试排空在机器上运行的节点，直到成功为止。在某些情况下，如错误配置了 pod 中断预算，排空操作可能无法成功。如果排空操作失败，机器控制器无法继续删除机器。

您可以通过在特定机器上注解 **machine.openshift.io/exclude-node-draining** 来跳过排空节点。

验证

- 运行以下命令，验证删除所需的机器：

```
$ oc get machines
```

9.3.5. 了解计算机器集和机器配置池之间的区别

MachineSet 对象描述了与云或机器供应商相关的 OpenShift Container Platform 节点。

MachineConfigPool 对象允许 **MachineConfigController** 组件在升级过程中定义并提供机器的状态。

MachineConfigPool 对象允许用户配置如何将升级应用到机器配置池中的 OpenShift Container Platform 节点。

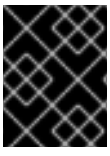
NodeSelector 对象可以被一个到 **MachineSet** 对象的引用替换。

9.4. 推荐的节点主机实践

OpenShift Container Platform 节点配置文件包含重要的选项。例如，控制可以为节点调度的最大 pod 数量的两个参数: **podsPerCore** 和 **maxPods**。

当两个参数都被设置时，其中较小的值限制了节点上的 pod 数量。超过这些值可导致：

- CPU 使用率增加。
- 减慢 pod 调度的速度。
- 根据节点中的内存数量，可能出现内存耗尽的问题。
- 耗尽 IP 地址池。
- 资源过量使用，导致用户应用程序性能变差。



重要

在 Kubernetes 中，包含单个容器的 pod 实际使用两个容器。第二个容器用来在实际容器启动前设置联网。因此，运行 10 个 pod 的系统实际上会运行 20 个容器。

注意

云供应商的磁盘 IOPS 节流可能会对 CRI-O 和 kubelet 产生影响。当节点上运行大量 I/O 高负载的 pod 时，可能会出现超载的问题。建议您监控节点上的磁盘 I/O，并使用有足够吞吐量的卷。

podsPerCore 参数根据节点上的处理器内核数来设置节点可运行的 pod 数量。例如：在一个有 4 个处理器内核的节点上将 **podsPerCore** 设为 **10**，则该节点上允许的最大 pod 数量为 **40**。

```
kubeletConfig:
  podsPerCore: 10
```

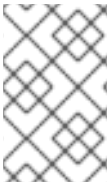
将 **podsPerCore** 设置为 **0** 可禁用这个限制。默认值为 **0**。**podsPerCore** 参数的值不能超过 **maxPods** 参数的值。

maxPods 参数将节点可以运行的 pod 数量设置为固定值，而不考虑节点的属性。

```
kubeletConfig:
  maxPods: 250
```

9.4.1. 创建 KubeletConfig CRD 来编辑 kubelet 参数

kubelet 配置目前被序列化为 Ignition 配置，因此可以直接编辑。但是，在 Machine Config Controller (MCC) 中同时添加了新的 **kubelet-config-controller**。这可让您使用 **KubeletConfig** 自定义资源 (CR) 来编辑 kubelet 参数。

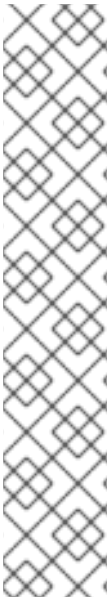


注意

因为 **kubeletConfig** 对象中的字段直接从上游 Kubernetes 传递给 kubelet，kubelet 会直接验证这些值。**kubeletConfig** 对象中的无效值可能会导致集群节点不可用。有关有效值，请参阅 Kubernetes 文档。

请考虑以下指导：

- 编辑现有的 **KubeletConfig** CR 以修改现有设置或添加新设置，而不是为每个更改创建一个 CR。建议您仅创建一个 CR 来修改不同的机器配置池，或用于临时更改，以便您可以恢复更改。
- 为每个机器配置池创建一个 **KubeletConfig** CR，带有该池需要更改的所有配置。
- 根据需要，创建多个 **KubeletConfig** CR，每个集群限制为 10。对于第一个 **KubeletConfig** CR，Machine Config Operator (MCO) 会创建一个机器配置，并附带 **kubelet**。对于每个后续 CR，控制器会创建另一个带有数字后缀的 **kubelet** 机器配置。例如，如果您有一个带有 **-2** 后缀的 **kubelet** 机器配置，则下一个 **kubelet** 机器配置会附加 **-3**。



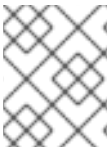
注意

如果要将 kubelet 或容器运行时配置应用到自定义机器配置池，则 **machineConfigSelector** 中的自定义角色必须与自定义机器配置池的名称匹配。

例如，由于以下自定义机器配置池名为 **infra**，因此自定义角色也必须是 **infra** ：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
# ...
```

如果要删除机器配置，以相反的顺序删除它们，以避免超过限制。例如，在删除 **kubelet-2** 机器配置前删除 **kubelet-3** 机器配置。



注意

如果您有一个带有 **kubelet-9** 后缀的机器配置，并且创建了另一个 **KubeletConfig** CR，则不会创建新的机器配置，即使少于 10 个 **kubelet** 机器配置。

KubeletConfig CR 示例

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

显示 **KubeletConfig 机器配置示例**

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                  b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
26m
...
```

以下流程演示了如何配置 worker 节点上的每个节点的最大 pod 数量。

先决条件

1. 为您要配置的节点类型获取与静态 **MachineConfigPool** CR 关联的标签。执行以下步骤之一：
   a. 查看机器配置池：

      ```
      $ oc describe machineconfigpool <name>
      ```

      例如：

```
$ oc describe machineconfigpool worker
```

输**出示例**

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods 1
```

1 如果添加了标签，它会出现在 **labels** 下。

b. 如果标签不存在，则添加一个键/值对：

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

流程

1. 查看您可以选择的可用机器配置对象：

```
$ oc get machineconfig
```

默认情况下，与 kubelet 相关的配置为 **01-master-kubelet** 和 **01-worker-kubelet**。

2. 检查每个节点的最大 pod 的当前值：

```
$ oc describe node <node_name>
```

例如：

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

在 **Allocatable** 小节中找到 **value: pods: <value>**：

输**出示例**

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

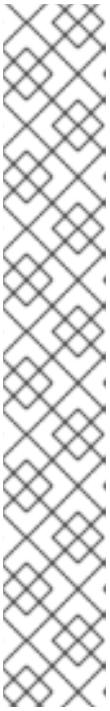
3. 通过创建一个包含 kubelet 配置的自定义资源文件，设置 worker 节点上的每个节点的最大 pod：

重要

以特定机器配置池为目标的 kubelet 配置也会影响任何依赖的池。例如，为包含 worker 节点的池创建 kubelet 配置也适用于任何子集池，包括包含基础架构节点的池。要避免这种情况，您必须使用仅包含 worker 节点的选择表达式创建新的机器配置池，并让 kubelet 配置以这个新池为目标。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods (1)
  kubeletConfig:
    maxPods: 500 (2)
```

1. 输入机器配置池中的标签。
2. 添加 kubelet 配置。在本例中，使用 **maxPods** 设置每个节点的最大 pod。



注意

kubelet 与 API 服务器进行交互的频率取决于每秒的查询数量 (QPS) 和 burst 值。如果每个节点上运行的 pod 数量有限，使用默认值（**kubeAPIQPS** 为 **50**，**kubeAPIBurst** 为 **100**）就可以。如果节点上有足够 CPU 和内存资源，则建议更新 kubelet QPS 和 burst 速率。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 为带有标签的 worker 更新机器配置池：

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

b. 创建 **KubeletConfig** 对象：

```
$ oc create -f change-maxPods-cr.yaml
```

c. 验证 **KubeletConfig** 对象是否已创建：

```
$ oc get kubeletconfig
```

输出示例

```
NAME                AGE
set-max-pods        15m
```

根据集群中的 worker 节点数量，等待每个 worker 节点被逐个重启。对于有 3 个 worker 节点的集群，这个过程可能需要大约 10 到 15 分钟。

4. 验证更改是否已应用到节点：

   a. 在 worker 节点上检查 **maxPods** 值已更改：

   ```
   $ oc describe node <node_name>
   ```

   b. 找到 **Allocatable** 小节：

   ```
    ...
   Allocatable:
     attachable-volumes-gce-pd:  127
     cpu:                        3500m
     ephemeral-storage:          123201474766
     hugepages-1Gi:              0
     hugepages-2Mi:              0
     memory:                     14225400Ki
     pods:                       500 (1)
    ...
   ```

   (1) 在本例中，**pods** 参数应报告您在 **KubeletConfig** 对象中设置的值。

5. 验证 **KubeletConfig** 对象中的更改：

   ```
   $ oc get kubeletconfigs set-max-pods -o yaml
   ```

   这应该显示 **True** 状态和 **type:Success**，如下例所示：

   ```
   spec:
     kubeletConfig:
       maxPods: 500
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: set-max-pods
   status:
     conditions:
     - lastTransitionTime: "2021-06-30T17:04:07Z"
       message: Success
       status: "True"
       type: Success
   ```

9.4.2. 修改不可用 worker 节点的数量

默认情况下，在对可用的 worker 节点应用 kubelet 相关的配置时，只允许一台机器不可用。对于大型集群来说，它可能需要很长时间才可以反映出配置的更改。在任何时候，您可以调整更新的机器数量来加快进程速度。

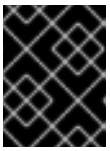
流程

1. 编辑 **worker** 机器配置池：

```
$ oc edit machineconfigpool worker
```

2. 添加 **maxUnavailable** 字段并设置值：

```
spec:
  maxUnavailable: <node_count>
```



重要

当设置该值时，请考虑无法使用的 worker 节点数量，而不影响在集群中运行的应用程序。

9.4.3. Control plane 节点大小

控制平面节点资源要求取决于集群中的节点和对象的数量和类型。以下控制平面节点大小是基于控制平面密度测试的结果，或 *Clusterdensity*。此测试会在给定很多命名空间中创建以下对象：

- 1 个镜像流
- 1 个构建
- 5 个部署，其中 2 个 pod 副本处于**睡眠**状态，每个状态都挂载 4 个 secret、4 个配置映射和 1 Downward API 卷
- 5 个服务，每个服务都指向前一个部署的 TCP/8080 和 TCP/8443 端口
- 1 个路由指向上一个服务的第一个路由
- 包含 2048 个随机字符串字符的 10 个 secret
- 10 个配置映射包含 2048 个随机字符串字符

worker 节点数量	集群密度（命名空间）	CPU 内核	内存 (GB)
24	500	4	16
120	1000	8	32
252	4000	16，但如果使用 OVN-Kubernetes 网络插件，则为 24	64，但在使用 OVN-Kubernetes 网络插件时为 128

worker 节点数量	集群密度（命名空间）	CPU 内核	内存 (GB)
501，但使用 OVN-Kubernetes 网络插件时未测试	4000	16	96

上表中的数据基于在 AWS 上运行的 OpenShift Container Platform，使用 r5.4xlarge 实例作为 control-plane 节点，m5.2xlarge 实例作为 worker 节点。

在具有三个 control plane 节点的大型高密度集群中，当其中一个节点停止、重启或失败时，CPU 和内存用量将会激增。故障可能是因为电源、网络、底层基础架构或意外情况造成意外问题，因为集群在关闭后重启，以节约成本。其余两个 control plane 节点必须处理负载才能高度可用，从而增加资源使用量。另外，在升级过程中还会有这个预期，因为 control plane 节点被封锁、排空并按顺序重新引导，以应用操作系统更新以及 control plane Operator 更新。为了避免级联失败，请将 control plane 节点上的总体 CPU 和内存资源使用量保留为最多 60% 的所有可用容量，以处理资源使用量激增。相应地增加 control plane 节点上的 CPU 和内存，以避免因为缺少资源而造成潜在的停机。

重要

节点大小取决于集群中的节点和对象数量。它还取决于集群上是否正在主动创建这些对象。在创建对象时，control plane 在资源使用量方面与对象处于**运行（running）**阶段的时间相比更活跃。

Operator Lifecycle Manager（OLM）在 control plane 节点上运行，其内存占用量取决于 OLM 在集群中管理的命名空间和用户安装的 operator 的数量。Control plane 节点需要相应地调整大小，以避免 OOM 终止。以下数据基于集群最大测试的结果。

命名空间数量	处于空闲状态的 OLM 内存（GB）	安装了 5 个用户 operator 的 OLM 内存（GB）
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8

命名空间数量	处于空闲状态的 OLM 内存（GB）	安装了 5 个用户 operator 的 OLM 内存（GB）
9000	8	17.7
10,000	9.9	21.6

重要

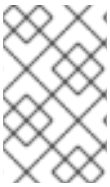
您只能为以下配置修改正在运行的 OpenShift Container Platform 4.15 集群中的 control plane 节点大小：

- 使用用户置备的安装方法安装的集群。
- 使用安装程序置备的基础架构安装方法安装的 AWS 集群。
- 使用 control plane 机器集管理 control plane 机器的集群。

对于所有其他配置，您必须估计节点总数并在安装过程中使用推荐的 control plane 节点大小。

重要

建议基于在带有 OpenShiftSDN 作为网络插件的 OpenShift Container Platform 集群上捕获的数据点。



注意

在 OpenShift Container Platform 4.15 中，与 OpenShift Container Platform 3.11 及之前的版本相比，系统现在默认保留半个 CPU 内核(500 millicore)。确定大小时应该考虑这一点。

9.4.4. 设置 CPU Manager

要配置 CPU Manager，请创建一个 KubeletConfig 自定义资源 (CR) 并将其应用到所需的一组节点。

流程

1. 运行以下命令来标记节点：

```
# oc label node perf-node.example.com cpumanager=true
```

2. 要为所有计算节点启用 CPU Manager，请运行以下命令来编辑 CR：

```
# oc edit machineconfigpool worker
```

3. 将 **custom-kubelet: cpumanager-enabled** 标签添加到 **metadata.labels** 部分。

```
metadata:
  creationTimestamp: 2020-xx-xxx
  generation: 3
```

```
  labels:
    custom-kubelet: cpumanager-enabled
```

4. 创建 **KubeletConfig**，**cpumanager-kubeletconfig.yaml**，自定义资源 (CR)。请参阅上一步中创建的标签，以便使用新的 kubelet 配置更新正确的节点。请参见 **MachineConfigPoolSelector** 部分：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s 2
```

1 指定一个策略：

- **none**.这个策略明确启用了现有的默认 CPU 关联性方案，从而不会出现超越调度程序自动进行的关联性。这是默认策略。
- **static**。此策略允许保证 pod 中的容器具有整数 CPU 请求。它还限制对节点上的专用 CPU 的访问。如果为 **static**，则需要使用一个小些 **s**。

2 可选。指定 CPU Manager 协调频率。默认值为 **5s**。

5. 运行以下命令来创建动态 kubelet 配置：

```
# oc create -f cpumanager-kubeletconfig.yaml
```

这会在 kubelet 配置中添加 CPU Manager 功能，如果需要，Machine Config Operator（MCO）将重启节点。要启用 CPU Manager，则不需要重启。

6. 运行以下命令，检查合并的 kubelet 配置：

```
# oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep ownerReference -A7
```

输出示例

```
    "ownerReferences": [
        {
            "apiVersion": "machineconfiguration.openshift.io/v1",
            "kind": "KubeletConfig",
            "name": "cpumanager-enabled",
            "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
        }
    ]
```

7. 运行以下命令，检查更新的 **kubelet.conf** 文件的计算节点：

-

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

输**出示例**

```
cpuManagerPolicy: static       1
cpuManagerReconcilePeriod: 5s  2
```

1. 在创建 **KubeletConfig** CR 时，会定义 **cpuManagerPolicy**。
2. 在创建 **KubeletConfig** CR 时，会定义 **cpuManagerReconcilePeriod**。

8. 运行以下命令来创建项目：

```
$ oc new-project <project_name>
```

9. 创建请求一个或多个内核的 pod。限制和请求都必须将其 CPU 值设置为一个整数。这是专用于此 pod 的内核数：

```
# cat cpumanager-pod.yaml
```

输**出示例**

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause:3.2
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    cpumanager: "true"
```

10. 创建 pod：

```
# oc create -f cpumanager-pod.yaml
```

验证

1. 运行以下命令，验证 pod 是否已调度到您标记的节点：

```
# oc describe pod cpumanager
```

输出示例

```
Name:               cpumanager-6cqz7
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:  perf-node.example.com/xxx.xx.xx.xxx
...
 Limits:
     cpu:     1
     memory:  1G
   Requests:
     cpu:        1
     memory:     1G
...
QoS Class:       Guaranteed
Node-Selectors:  cpumanager=true
```

2. 运行以下命令，验证 CPU 是否已完全分配给 pod：

```
# oc describe node --selector='cpumanager=true' | grep -i cpumanager- -B2
```

输出示例

```
NAMESPACE    NAME                CPU Requests  CPU Limits  Memory Requests  Memory Limits  Age
cpuman       cpumanager-mlrrz    1 (28%)       1 (28%)     1G (13%)         1G (13%)       27m
```

3. 确认正确配置了 **cgroups**。运行以下命令，获取 **cluster** 进程的进程 ID (PID)：

```
# oc debug node/perf-node.example.com
```

```
sh-4.2# systemctl status | grep -B5 pause
```

注意

如果输出返回多个暂停进程条目，您必须识别正确的暂停进程。

输出示例

```
# ├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
└─kubepods.slice
  ├─kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
  │ ├─crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
  │ └─32706 /pause
```

4. 运行以下命令，验证 pod 服务质量(QoS)等级 **Guaranteed** 是否在 **kubepods.slice** 子目录中：

```
# cd /sys/fs/cgroup/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
```

```
# for i in `ls cpuset.cpus cgroup.procs` ; do echo -n "$i "; cat $i ; done
```

注意

其他 QoS 等级的 Pod 会位于父 **kubepods** 的子 **cgroups** 中。

输**出示例**

```
cpuset.cpus 1
tasks 32706
```

5. 运行以下命令，检查任务允许的 CPU 列表：

```
# grep ^Cpus_allowed_list /proc/32706/status
```

输**出示例**

```
Cpus_allowed_list:    1
```

6. 验证系统中的另一个 pod 无法在为 **Guaranteed** pod 分配的内核中运行。例如，要验证 **besteffort** QoS 层中的 pod，请运行以下命令：

```
# cat /sys/fs/cgroup/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus
```

```
# oc describe node perf-node.example.com
```

输**出示例**

```
...
Capacity:
 attachable-volumes-aws-ebs:  39
 cpu:                         2
 ephemeral-storage:           124768236Ki
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      8162900Ki
 pods:                        250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                         1500m
 ephemeral-storage:           124768236Ki
 hugepages-1Gi:               0
 hugepages-2Mi:               0
```

```
 memory:                    7548500Ki
 pods:                      250
-------                         ----                    ------------  ----------  ---------------  -------------  --
-
 default                         cpumanager-6cqz7             1 (66%)       1 (66%)     1G (12%)
1G (12%)       29m

Allocated resources:
 (Total limits may be over 100 percent, i.e., overcommitted.)
 Resource                   Requests         Limits
 --------                   --------         ------
 cpu                        1440m (96%)      1 (66%)
```

这个 VM 有两个 CPU 内核。**system-reserved** 设置保留 500 millicores，这代表一个内核中的一半被从节点的总容量中减小，以达到 **Node Allocatable** 的数量。您可以看到 **Allocatable CPU** 是 1500 毫秒。这意味着您可以运行一个 CPU Manager pod，因为每个 pod 需要一个完整的内核。一个完整的内核等于 1000 毫秒。如果您尝试调度第二个 pod，系统将接受该 pod，但不会调度它：

```
NAME                    READY   STATUS    RESTARTS   AGE
cpumanager-6cqz7        1/1     Running   0          33m
cpumanager-7qc2t        0/1     Pending   0          11s
```

9.5. 巨页

了解并配置巨页。

9.5.1. 巨页的作用

内存在块（称为页）中进行管理。在大多数系统中，页的大小为 4Ki。1Mi 内存相当于 256 个页，1Gi 内存相当于 256,000 个页。CPU 有内置的内存管理单元，可在硬件中管理这些页的列表。Translation Lookaside Buffer (TLB) 是虚拟页到物理页映射的小型硬件缓存。如果在硬件指令中包括的虚拟地址可以在 TLB 中找到，则其映射信息可以被快速获得。如果没有包括在 TLN 中，则称为 TLB miss。系统将会使用基于软件的，速度较慢的地址转换机制，从而出现性能降低的问题。因为 TLB 的大小是固定的，因此降低 TLB miss 的唯一方法是增加页的大小。

巨页指一个大于 4Ki 的内存页。在 x86_64 构架中，有两个常见的巨页大小: 2Mi 和 1Gi。在其它构架上的大小会有所不同。要使用巨页，必须写相应的代码以便应用程序了解它们。Transparent Huge Pages（THP）试图在应用程序不需要了解的情况下自动管理巨页，但这个技术有一定的限制。特别是，它的页大小会被限为 2Mi。当有较高的内存使用率时，THP 可能会导致节点性能下降，或出现大量内存碎片（因为 THP 的碎片处理）导致内存页被锁定。因此，有些应用程序可能更适用于（或推荐）使用预先分配的巨页，而不是 THP。

9.5.2. 应用程序如何使用巨页

节点必须预先分配巨页以便节点报告其巨页容量。一个节点只能预先分配一个固定大小的巨页。

巨页可以使用名为 **hugepages-<size>** 的容器一级的资源需求被消耗。其中 size 是特定节点上支持的整数值的最精简的二进制标记。例如：如果某个节点支持 2048KiB 页大小，它将会有一个可调度的资源 **hugepages-2Mi**。与 CPU 或者内存不同，巨页不支持过量分配。

```
apiVersion: v1
kind: Pod
metadata:
```

```
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        hugepages-2Mi: 100Mi (1)
        memory: "1Gi"
        cpu: "1"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1) 为巨页指定要分配的准确内存数量。不要将这个值指定为巨页内存大小乘以页的大小。例如，巨页的大小为 2MB，如果应用程序需要使用由巨页组成的 100MB 的内存，则需要分配 50 个巨页。OpenShift Container Platform 会进行相应的计算。如上例所示，您可以直接指定 **100MB** 。

分配特定大小的巨页

有些平台支持多个巨页大小。要分配指定大小的巨页，在巨页引导命令参数前使用巨页大小选择参数**hugepagesz=<size>**。**<size>** 的值必须以字节为单位，并可以使用一个可选的后缀 [**kKmMgG**]。默认的巨页大小可使用 **default_hugepagesz=<size>** 引导参数定义。

巨页要求

- 巨页面请求必须等于限制。如果指定了限制，则它是默认的，但请求不是。
- 巨页在 pod 范围内被隔离。容器隔离功能计划在以后的版本中推出。
- 后端为巨页的 **EmptyDir** 卷不能消耗大于 pod 请求的巨页内存。
- 通过带有 **SHM_HUGETLB** 的 **shmget()** 来使用巨页的应用程序，需要运行一个匹配 ***proc/sys/vm/hugetlb_shm_group*** 的 supplemental 组。

9.5.3. 在引导时配置巨页

节点必须预先分配在 OpenShift Container Platform 集群中使用的巨页。保留巨页的方法有两种： 在引导时和在运行时。在引导时进行保留会增加成功的可能性，因为内存还没有很大的碎片。Node Tuning Operator 目前支持在特定节点上分配巨页。

流程

要减少节点重启的情况，请按照以下步骤顺序进行操作：

1. 通过标签标记所有需要相同巨页设置的节点。

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. 创建一个包含以下内容的文件，并把它命名为 **hugepages_tuning.yaml**：

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages (1)
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: (2)
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 (3)
    name: openshift-node-hugepages

  recommend:
  - machineConfigLabels: (4)
      machineconfiguration.openshift.io/role: "worker-hp"
    priority: 30
    profile: openshift-node-hugepages
```

(1) 将 Tuned 资源的 **name** 设置为 **hugepages**。

(2) 将 **profile** 部分设置为分配巨页。

(3) 请注意，参数顺序是非常重要的，因为有些平台支持各种大小的巨页。

(4) 启用基于机器配置池的匹配。

3. 创建 Tuned **hugepages** 对象

```
$ oc create -f hugepages-tuned-boottime.yaml
```

4. 创建一个带有以下内容的文件，并把它命名为 **hugepages-mcp.yaml**：

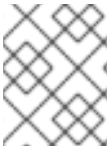
```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-hp
  labels:
    worker-hp: ""
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-hp]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-hp: ""
```

5. 创建机器配置池：

```
$ oc create -f hugepages-mcp.yaml
```

因为有足够的非碎片内存，**worker-hp** 机器配置池中的所有节点现在都应分配 50 个 2Mi 巨页。

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```



注意

TuneD bootloader 插件只支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。

9.6. 了解设备插件

设备插件提供一致并可移植的解决方案，以便跨集群消耗硬件设备。设备插件通过一种扩展机制为这些设备提供支持，从而使这些设备可供容器使用，提供这些设备的健康检查，并安全地共享它们。

重要

OpenShift Container Platform 支持设备插件 API，但设备插件容器由各个供应商提供支持。

设备插件是在节点（**kubelet** 的外部）上运行的 gRPC 服务，负责管理特定的硬件资源。任何设备插件都必须支持以下远程过程调用 (RPC)：

```
service DevicePlugin {
    // GetDevicePluginOptions returns options to be communicated with Device
    // Manager
    rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

    // ListAndWatch returns a stream of List of Devices
    // Whenever a Device state change or a Device disappears, ListAndWatch
    // returns the new list
    rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

    // Allocate is called during container creation so that the Device
    // Plug-in can run device specific operations and instruct Kubelet
    // of the steps to make the Device available in the container
    rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

    // PreStartcontainer is called, if indicated by Device Plug-in during
    // registration phase, before each container start. Device plug-in
    // can run device specific operations such as resetting the device
    // before making devices available to the container
    rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

设备插件示例

- 适用于 COS 型操作系统的 Nvidia GPU 设备插件

- Nvidia 官方 GPU 设备插件
- Solarflare 设备插件
- KubeVirt 设备插件：vfio 和 kvm
- 用于 IBM® Crypto Express (CEX) 卡的 Kubernetes 设备插件

注意

对于简单设备插件参考实现，设备管理器代码中有一个 stub 设备插件：*vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go*。

9.6.1. 设备插件部署方法

- 守护进程集是设备插件部署的推荐方法。
- 在启动时，设备插件会尝试在节点上 */var/lib/kubelet/device-plugin/* 创建一个 UNIX 域套接字，以便服务来自于设备管理器的 RPC。
- 由于设备插件必须管理硬件资源、主机文件系统的访问权以及套接字创建，它们必须在一个特权安全上下文中运行。
- 各种设备插件实现中提供了有关部署步骤的更多细节。

9.6.2. 了解设备管理器

设备管理器提供了一种机制，可借助称为“设备插件”的插件公告专用节点硬件资源。

您可以公告专用的硬件，而不必修改任何上游代码。

重要

OpenShift Container Platform 支持设备插件 API，但设备插件容器由各个供应商提供支持。

设备管理器将设备公告为**外部资源**。用户 pod 可以利用相同的**限制/请求**机制来使用设备管理器公告的设备，这一机制也用于请求任何其他**扩展资源**。

在启动时，设备插件会在 */var/lib/kubelet/device-plugins/kubelet.sock* 上调用 **Register** 将自身注册到设备管理器，并启动位于 */ var/lib/kubelet/device-plugins/<plugin>.sock* 的 gRPC 服务，以服务设备管理器请求。

在处理新的注册请求时，设备管理器会在设备插件服务中调用 **ListAndWatch** 远程过程调用 (RPC)。作为响应，设备管理器通过 gRPC 流从插件中获取设备对象的列表。设备管理器对流进行持续监控，以确认插件有没有新的更新。在插件一端，插件也会使流保持开放；只要任何设备的状态有所改变，就会通过相同的流传输连接将新设备列表发送到设备管理器。

在处理新的 pod 准入请求时，Kubelet 将请求的**扩展资源**传递给设备管理器以进行设备分配。设备管理器在其数据库中检查，以验证是否存在对应的插件。如果插件存在并且有可分配的设备及本地缓存，则在该特定设备插件上调用 **Allocate** RPC。

此外，设备插件也可以执行其他几个特定于设备的操作，如驱动程序安装、设备初始化和设备重置。这些功能视具体实现而异。

9.6.3. 启用设备管理器

启用设备管理器来实现设备插件，在不更改上游代码的前提下公告专用硬件。

设备管理器提供了一种机制，可借助称为“设备插件”的插件公告专用节点硬件资源。

1. 输入以下命令为您要配置的节点类型获取与静态 **MachineConfigPool** CRD 关联的标签。执行以下步骤之一：

   a. 查看机器配置：

   ```
   # oc describe machineconfig <name>
   ```

   例如：

   ```
   # oc describe machineconfig 00-worker
   ```

   输出示例

   ```
   Name:         00-worker
   Namespace:
   Labels:       machineconfiguration.openshift.io/role=worker (1)
   ```

   (1) 设备管理器所需标签。

流程

1. 为配置更改创建自定义资源 (CR)。

   设备管理器 CR 配置示例

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: devicemgr (1)
   spec:
     machineConfigPoolSelector:
       matchLabels:
         machineconfiguration.openshift.io: devicemgr (2)
     kubeletConfig:
       feature-gates:
         - DevicePlugins=true (3)
   ```

   (1) 为 CR 分配一个名称。

   (2) 输入来自机器配置池的标签。

   (3) 将 **DevicePlugins** 设为“true”。

2. 创建设备管理器：

   ```
   $ oc create -f devicemgr.yaml
   ```

输出示例

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. 通过确认节点上已创建了 */var/lib/kubelet/device-plugins/kubelet.sock*，确保已启用了设备管理器。这是设备管理器 gRPC 服务器在其上侦听新插件注册的 UNIX 域套接字。只有启用了设备管理器，才会在 Kubelet 启动时创建此 sock 文件。

9.7. 污点和容限

理解并使用污点和容限。

9.7.1. 了解污点和容限

通过使用*污点（taint）*，节点可以拒绝调度 pod，除非 pod 具有匹配的 *容限（toleration）*。

您可以通过节点规格（**NodeSpec**）将污点应用到节点，并通过 **Pod** 规格（**PodSpec**）将容限应用到 pod。当您应用污点时，调度程序无法将 pod 放置到该节点上，除非 pod 可以容限该污点。

节点规格中的污点示例

```
apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Pod 规格中的容限示例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

污点与容限由 key、value 和 effect 组成。

表 9.1. 污点和容限组件

参数	描述
key	**key** 是任意字符串，最多 253 个字符。key 必须以字母或数字开头，可以包含字母、数字、连字符、句点和下划线。
value	**value** 是任意字符串，最多 63 个字符。value 必须以字母或数字开头，可以包含字母、数字、连字符、句点和下划线。
effect	effect 的值包括： **NoSchedule** [1]： • 与污点不匹配的新 pod 不会调度到该节点上。 • 该节点上现有的 pod 会保留。 **PreferNoSchedule**： • 与污点不匹配的新 pod 可以调度到该节点上，但调度程序会尽量不这样调度。 • 该节点上现有的 pod 会保留。 **NoExecute**： • 与污点不匹配的新 pod 无法调度到该节点上。 • 节点上没有匹配容限的现有 pod 将被移除。
operator	**Equal**：**key**/**value**/**effect** 参数必须匹配。这是默认值。 **Exists**：**key**/**effect** 参数必须匹配。您必须保留一个空的 **value** 参数，这将匹配任何值。

1. 如果向 control plane 节点添加了一个 **NoSchedule** 污点，节点必须具有 **node-role.kubernetes.io/master=:NoSchedule** 污点，这默认会添加。
例如：

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
  name: my-node
#...
spec:
```

```
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
#...
```

容限与污点匹配：

- 如果 **operator** 参数设为 **Equal**：
  - **key** 参数相同；
  - **value** 参数相同；
  - **effect** 参数相同。
- 如果 **operator** 参数设为 **Exists**：
  - **key** 参数相同；
  - **effect** 参数相同。

OpenShift Container Platform 中内置了以下污点：

- **node.kubernetes.io/not-ready**：节点未就绪。这与节点状况 **Ready=False** 对应。
- **node.kubernetes.io/unreachable**：节点无法从节点控制器访问。这与节点状况 **Ready=Unknown** 对应。
- **node.kubernetes.io/memory-pressure**：节点存在内存压力问题。这与节点状况 **MemoryPressure=True** 对应。
- **node.kubernetes.io/disk-pressure**：节点存在磁盘压力问题。这与节点状况 **DiskPressure=True** 对应。
- **node.kubernetes.io/network-unavailable**：节点网络不可用。
- **node.kubernetes.io/unschedulable**：节点不可调度。
- **node.cloudprovider.kubernetes.io/uninitialized**：当节点控制器通过外部云提供商启动时，在节点上设置这个污点来将其标记为不可用。在云控制器管理器中的某个控制器初始化这个节点后，kubelet 会移除此污点。
- **node.kubernetes.io/pid-pressure** ：节点具有 pid 压力。这与节点状况 **PIDPressure=True** 对应。

重要

OpenShift Container Platform 不设置默认的 pid.available **evictionHard**。

9.7.2. 添加污点和容限

您可以为 pod 和污点添加容限，以便节点能够控制哪些 pod 应该或不应该调度到节点上。对于现有的 pod 和节点，您应首先将容限添加到 pod，然后将污点添加到节点，以避免在添加容限前从节点上移除 pod。

流程

1. 通过编辑 **Pod** spec 使其包含 **tolerations** 小节来向 pod 添加容限：

使用 Equal 运算符的 pod 配置文件示例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1" (1)
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 (2)
#...
```

(1) 容限参数，如 **Taint 和 toleration 组件**表中所述。

(2) **tolerationSeconds** 参数指定 pod 在被驱除前可以保持与节点绑定的时长。

例如：

使用 Exists 运算符的 pod 配置文件示例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1"
      operator: "Exists" (1)
      effect: "NoExecute"
      tolerationSeconds: 3600
#...
```

(1) **Exists** 运算符不会接受一个 **value**。

本例在 **node1** 上放置一个键为 **key1** 且值为 **value1** 的污点，污点效果是 **NoExecute**。

2. 通过以下命令，使用 **Taint 和 toleration 组件**表中描述的参数为节点添加污点：

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

例如：

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

此命令在 **node1** 上放置一个键为 **key1**，值为 **value1** 的污点，其效果是 **NoExecute**。

注意

如果向 control plane 节点添加了一个 **NoSchedule** 污点，节点必须具有 **node-role.kubernetes.io/master=:NoSchedule** 污点，这默认会添加。

例如：

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-cdc1ab7da414629332cc4c3926e6e59c
  name: my-node
#...
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
#...
```

pod 上的容限与节点上的污点匹配。具有任一容限的 pod 可以调度到 **node1** 上。

9.7.3. 使用计算机器集添加污点和容限

您可以使用计算机器集为节点添加污点。与 **MachineSet** 对象关联的所有节点都会使用污点更新。容限响应由计算机器设置添加的污点，其方式与直接添加到节点的污点相同。

流程

1. 通过编辑 **Pod** spec 使其包含 **tolerations** 小节来向 pod 添加容限：

   使用 Equal 运算符的 pod 配置文件示例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1" (1)
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 (2)
#...
```

(1) 容限参数，如 **Taint 和 toleration 组件**表中所述。

(2) **tolerationSeconds** 参数指定 pod 在被驱除前与节点绑定的时长。

例如：

使用 Exists 运算符的 pod 配置文件示例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

2. 将污点添加到 **MachineSet** 对象：

   a. 为您想要污点的节点编辑 **MachineSet** YAML，也可以创建新 **MachineSet** 对象：

   ```
   $ oc edit machineset <machineset>
   ```

   b. 将污点添加到 **spec.template.spec** 部分：

   计算机器设置规格中的污点示例

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     name: my-machineset
   #...
   spec:
   #...
     template:
   #...
       spec:
         taints:
         - effect: NoExecute
           key: key1
           value: value1
   #...
   ```

   本例在节点上放置一个键为 **key1**，值为 **value1** 的污点，污点效果是 **NoExecute**。

   c. 将计算机器设置为 0：

   ```
   $ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
   ```

提示

您还可以应用以下 YAML 来扩展计算机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

等待机器被删除。

d. 根据需要扩展计算机器：

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

或者：

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

等待机器启动。污点添加到与 **MachineSet** 对象关联的节点上。

9.7.4. 使用污点和容限将用户绑定到节点

如果要指定一组节点供特定用户独占使用，为 pod 添加容限。然后，在这些节点中添加对应的污点。具有容限的 pod 被允许使用污点节点，或集群中的任何其他节点。

如果您希望确保 pod 只调度到那些污点节点，还要将标签添加到同一组节点，并为 pod 添加节点关联性，以便 pod 只能调度到具有该标签的节点。

流程

配置节点以使用户只能使用该节点：

1. 为这些节点添加对应的污点：
   例如：

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

提示

您还可以应用以下 YAML 来添加污点：

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
#...
```

2. 通过编写自定义准入控制器，为 pod 添加容限。

9.7.5. 使用污点和容限控制具有特殊硬件的节点

如果集群中有少量节点具有特殊的硬件，您可以使用污点和容限让不需要特殊硬件的 pod 与这些节点保持距离，从而将这些节点保留给那些确实需要特殊硬件的 pod。您还可以要求需要特殊硬件的 pod 使用特定的节点。

您可以将容限添加到需要特殊硬件并污点具有特殊硬件的节点的 pod 中。

流程

确保为特定 pod 保留具有特殊硬件的节点：

1. 为需要特殊硬件的 pod 添加容限。
例如：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
#...
```

2. 使用以下命令之一，给拥有特殊硬件的节点添加污点：

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

或者：

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

提示

您还可以应用以下 YAML 来添加污点：

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
#...
```

9.7.6. 删除污点和容限

您可以根据需要，从节点移除污点并从 pod 移除容限。您应首先将容限添加到 pod，然后将污点添加到节点，以避免在添加容限前从节点上移除 pod。

流程

移除污点和容限：

1. 从节点移除污点：

```
$ oc adm taint nodes <node-name> <key>-
```

例如：

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

输**出示例**

```
node/ip-10-0-132-248.ec2.internal untainted
```

2. 要从 pod 移除某一容限，请编辑 **Pod** 规格来移除该容限：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key2"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

9.8. 拓扑管理器

理解并使用拓扑管理器。

9.8.1. 拓扑管理器策略

拓扑管理器通过从 Hint 提供者（如 CPU Manager 和设备管理器）收集拓扑提示来调整所有级别服务质量（QoS）的 **Pod** 资源，并使用收集的提示来匹配 **Pod** 资源。

拓扑管理器支持四个分配策略，这些策略在名为 **cpumanager-enabled** 的 **KubeletConfig** 自定义资源 (CR) 中分配：

none 策略

这是默认策略，不执行任何拓扑对齐调整。

best-effort 策略

对于带有 **best-effort** 拓扑管理策略的 pod 中的每个容器，kubelet 会调用每个 Hint 提供者来发现其资源的可用性。使用这些信息，拓扑管理器会保存那个容器的首选 NUMA 节点关联性设置。如果关联性没有被首选设置，则拓扑管理器会保存这个设置，并把 pod 分配给节点。

restricted 策略

对于带有 **restricted** 拓扑管理策略的 pod 中的每个容器，kubelet 会调用每个 Hint 提供者来发现其资源的可用性。使用这些信息，拓扑管理器会保存那个容器的首选 NUMA 节点关联性设置。如果关联性没有被首选，则拓扑管理器会从节点拒绝这个 pod，从而导致 pod 处于 **Terminated** 状态，且 pod 准入失败。

single-numa-node 策略

对于带有 **single-numa-node** 拓扑管理策略的 pod 中的每个容器，kubelet 会调用每个 Hint 提供者来发现其资源的可用性。使用这个信息，拓扑管理器会决定单个 NUMA 节点关联性是否可能。如果是，pod 将会分配给该节点。如果无法使用单一 NUMA 节点关联性，则拓扑管理器会拒绝来自节点的 pod。这会导致 pod 处于 Terminated 状态，且 pod 准入失败。

9.8.2. 设置拓扑管理器

要使用拓扑管理器，您必须在名为 **cpumanager-enabled** 的 **KubeletConfig** 自定义资源 (CR) 中配置分配策略。如果您设置了 CPU Manager，则该文件可能会存在。如果这个文件不存在，您可以创建该文件。

先决条件

- 将 CPU Manager 策略配置为 **static**。

流程

激活拓扑管理器：

1. 在自定义资源中配置拓扑管理器分配策略。

```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
```

```
kubeletConfig:
  cpuManagerPolicy: static 1
  cpuManagerReconcilePeriod: 5s
  topologyManagerPolicy: single-numa-node 2
```

1 这个参数必须是 **static**，**s** 为小写。

2 指定所选拓扑管理器分配策略。在这里，策略是 **single-numa-node**。有效值为：**default**、**best-effort**、**restricted**、**single-numa-node**。

9.8.3. Pod 与拓扑管理器策略的交互

以下的 **Pod** specs 示例演示了 Pod 与 Topology Manager 的交互。

因为没有指定资源请求或限制，以下 pod 以 **BestEffort** QoS 类运行。

```
spec:
  containers:
  - name: nginx
    image: nginx
```

因为请求小于限制，下一个 pod 以 **Burstable** QoS 类运行。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

如果所选策略不是 **none**，则拓扑管理器将不考虑其中任何一个 **Pod** 规格。

因为请求等于限制，最后一个 pod 以 Guaranteed QoS 类运行。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

拓扑管理器将考虑这个 pod。拓扑管理器会参考 CPU Manager 和设备管理器的 hint 供应商，以获取 pod 的拓扑提示。

拓扑管理器将使用此信息存储该容器的最佳拓扑。在本 pod 中，CPU Manager 和设备管理器将在资源分配阶段使用此存储的信息。

9.9. 资源请求和过量使用

对于每个计算资源，容器可以指定一个资源请求和限制。根据确保节点有足够可用容量以满足请求值的请求来做出调度决策。如果容器指定了限制，但忽略了请求，则请求会默认采用这些限制。容器无法超过节点上指定的限制。

限制的强制实施取决于计算资源类型。如果容器没有请求或限制，容器会调度到没有资源保障的节点。在实践中，容器可以在最低本地优先级适用的范围内消耗指定的资源。在资源较少的情况下，不指定资源请求的容器将获得最低的服务质量。

调度基于请求的资源，而配额和硬限制指的是资源限制，它们可以设置为高于请求的资源。请求和限制的差值决定了过量使用程度；例如，如果为容器赋予 1Gi 内存请求和 2Gi 内存限制，则根据 1Gi 请求将容器调度到节点上，但最多可使用 2Gi；因此过量使用为 200%。

9.10. 使用 CLUSTER RESOURCE OVERRIDE OPERATOR 的集群级别的过量使用

Cluster Resource Override Operator 是一个准入 Webhook，可让您控制过量使用的程度，并在集群中的所有节点上管理容器密度。Operator 控制特定项目中节点可以如何超过定义的内存和 CPU 限值。

您必须使用 OpenShift Container Platform 控制台或 CLI 安装 Cluster Resource override Operator，如下所示。在安装过程中，您会创建一个 **ClusterResourceOverride** 自定义资源 (CR)，其中设置过量使用级别，如下例所示：

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
   name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
# ...
```

(1) 名称必须是 **cluster**。

(2) 可选。如果指定或默认指定了容器内存限值，则该内存请求会覆盖到限值的这个百分比，从 1 到 100 之间。默认值为 50.

(3) 可选。如果指定或默认指定了容器 CPU 限值，则将 CPU 请求覆盖到限值的这个百分比，从 1 到 100 之间。默认值为 25。

(4) 可选。如果指定或默认指定了容器内存限值，则 CPU 限值将覆盖的内存限值的百分比（如果指定）。以 100% 扩展 1Gi RAM， 等于 1 个 CPU 内核。这会在覆盖 CPU 请求前进行（如果配置了）。默认值为 200。

注意

如果容器上没有设置限值，则 Cluster Resourceoverride Operator 覆盖无效。创建一个针对单独项目的带有默认限制的 **LimitRange** 对象，或在 **Pod** specs 中配置要应用的覆盖的限制。

配置后，可通过将以下标签应用到每个项目的命名空间对象来启用每个项目的覆盖：

```
apiVersion: v1
kind: Namespace
metadata:

# ...

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"

# ...
```

Operator 监视 **ClusterResourceOverride** CR， 并确保 **ClusterResourceOverride** 准入 Webhook 被安装到与 Operator 相同的命名空间。

9.10.1. 使用 Web 控制台安装 Cluster Resource Override Operator

您可以使用 OpenShift Container Platform Web 控制台来安装 Cluster Resource Override Operator，以帮助控制集群中的过量使用。

先决条件

- 如果容器上未设置限值，Cluster Resourceoverride Operator 将没有作用。您必须使用一个 **LimitRange** 对象为项目指定默认限值，或在 **Pod** spec 中配置要应用的覆盖的限制。

流程

使用 OpenShift Container Platform web 控制台安装 Cluster Resource Override Operator：

1. 在 OpenShift Container Platform web 控制台中进入 **Home → Projects**
   a. 点击 **Create Project**。
   b. 指定 **clusterresourceoverride-operator** 作为项目的名称。
   c. 点击 **Create**。
2. 进入 **Operators → OperatorHub**。
   a. 从可用 Operator 列表中选择 **ClusterResourceOverride Operator**，再点击 **Install**。
   b. 在 **Install Operator** 页面中，确保为 **Installation Mode** 选择了 **A specific Namespace on the cluster**。
   c. 确保为 **Installed Namespace** 选择了 **clusterresourceoverride-operator**。
   d. 指定**更新频道**和**批准策略**。
   e. 点击 **Install**。

3. 在 **Installed Operators** 页面中，点 **ClusterResourceOverride**。

   a. 在 **ClusterResourceOverride Operator** 详情页面中，点 **Create ClusterResourceOverride**。

   b. 在 **Create ClusterResourceOverride** 页面中，点 **YAML 视图**并编辑 YAML 模板，以根据需要设置过量使用值：

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
# ...
```

   (1) 名称必须是 **cluster**。

   (2) 可选。指定在 1-100 之间覆盖容器内存限值的百分比（如果使用的话）。默认值为 50.

   (3) 可选。指定在 1-100 之间覆盖容器 CPU 限值的百分比（如果使用的话）。默认值为 25。

   (4) 可选。如果使用，请指定覆盖容器内存限值的百分比。以 100% 扩展 1Gi RAM， 等于 1 个 CPU 内核。这会在覆盖 CPU 请求前进行处理（如果已配置）。默认值为 200。

   c. 点击 **Create**。

4. 通过检查集群自定义资源的状态来检查准入 Webhook 的当前状态：

   a. 在 **ClusterResourceOverride Operator** 页面中，点击 **cluster**。

   b. 在 **ClusterResourceOverride Details** 页中，点 **YAML**。当 webhook 被调用时，**mutatingWebhookConfigurationRef** 项会出现。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
```

```
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

    mutatingWebhookConfigurationRef: 1
      apiVersion: admissionregistration.k8s.io/v1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...
```

1 引用 **ClusterResourceOverride** 准入Webhook。

9.10.2. 使用 CLI 安装 Cluster Resource Override Operator

您可以使用 OpenShift Container Platform CLI 来安装 Cluster Resource Override Operator，以帮助控制集群中的过量使用。

先决条件

- 如果容器上未设置限值，Cluster Resourceoverride Operator 将没有作用。您必须使用一个 **LimitRange** 对象为项目指定默认限值，或在 **Pod** spec 中配置要应用的覆盖的限制。

流程

使用 CLI 安装 Cluster Resource Override Operator：

1. 为 Cluster Resource Override Operator 创建命名空间：
   a. 为 Cluster Resource Override Operator 创建一个 **Namespace** 空间对象 YAML 文件（如 **cro-namespace.yaml**）：

```
apiVersion: v1
kind: Namespace
metadata:
  name: clusterresourceoverride-operator
```

   b. 创建命名空间：

```
$ oc create -f <file-name>.yaml
```

   例如：

```
$ oc create -f cro-namespace.yaml
```

2. 创建一个 Operator 组：

a. 为 Cluster Resource Override Operator 创建一个 **OperatorGroup** 对象 YAML 文件（如 cro-og.yaml）：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: clusterresourceoverride-operator
  namespace: clusterresourceoverride-operator
spec:
  targetNamespaces:
    - clusterresourceoverride-operator
```

b. 创建 Operator 组：

```
$ oc create -f <file-name>.yaml
```

例如：

```
$ oc create -f cro-og.yaml
```

3. 创建一个订阅：

a. 为 Cluster Resourceoverride Operator 创建一个 **Subscription** 对象 YAML 文件（如 cro-sub.yaml):

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "4.15"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. 创建订阅：

```
$ oc create -f <file-name>.yaml
```

例如：

```
$ oc create -f cro-sub.yaml
```

4. 在 **clusterresourceoverride-operator** 命名空间中创建 **ClusterResourceOverride** 自定义资源（CR）对象：

a. 进入 **clusterresourceoverride-operator** 命名空间。

```
$ oc project clusterresourceoverride-operator
```

b. 为 Cluster Resourceoverride Operator 创建 **ClusterResourceOverride** 对象 YAML 文件（如 cro-cr.yaml):

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
```

(1) 名称必须是 **cluster**。

(2) 可选。指定在 1-100 之间覆盖容器内存限值的百分比（如果使用的话）。默认值为 50.

(3) 可选。指定在 1-100 之间覆盖容器 CPU 限值的百分比（如果使用的话）。默认值为 25。

(4) 可选。如果使用，请指定覆盖容器内存限值的百分比。以 100% 扩展 1Gi RAM， 等于 1 个 CPU 内核。这会在覆盖 CPU 请求前进行处理（如果已配置）。默认值为 200。

c. 创建 **ClusterResourceOverride** 对象：

```
$ oc create -f <file-name>.yaml
```

例如：

```
$ oc create -f cro-cr.yaml
```

5. 通过检查集群自定义资源的状态来验证准入 Webhook 的当前状态。

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

当 webhook 被调用时， **mutatingWebhookConfigurationRef** 项会出现。

输出示例

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
```

```
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

    mutatingWebhookConfigurationRef: 1
      apiVersion: admissionregistration.k8s.io/v1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...
```

1 引用 **ClusterResourceOverride** 准入Webhook。

9.10.3. 配置集群级别的过量使用

Cluster Resource Override Operator 需要一个 **ClusterResourceOverride** 自定义资源 (CR)，以及您希望 Operator 来控制过量使用的每个项目的标识。

先决条件

- 如果容器上未设置限值，Cluster Resourceoverride Operator 将没有作用。您必须使用一个 **LimitRange** 对象为项目指定默认限值，或在 **Pod** spec 中配置要应用的覆盖的限制。

流程

修改集群级别的过量使用：

1. 编辑 **ClusterResourceOverride** CR:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
   name: cluster
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 1
      cpuRequestToLimitPercent: 25 2
      limitCPUToMemoryPercent: 200 3
# ...
```

1 可选。指定在 1-100 之间覆盖容器内存限值的百分比（如果使用的话）。默认值为 50.

2 可选。指定在 1-100 之间覆盖容器 CPU 限值的百分比（如果使用的话）。默认值为 25。

3 可选。如果使用，请指定覆盖容器内存限值的百分比。以 100% 扩展 1Gi RAM， 等于 1 个 CPU 内核。这会在覆盖 CPU 请求前进行处理（如果已配置）。默认值为 200。

2. 确保在每个您希望 Cluster Resourceoverride Operator 来控制过量使用的项目中都添加了以下标识：

```
apiVersion: v1
kind: Namespace
metadata:

# ...

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" (1)

# ...
```

(1) 把这个标识添加到每个项目。

9.11. 节点级别的过量使用

您可以使用各种方法来控制特定节点上的过量使用，如服务质量 (QOS) 保障、CPU 限值或保留资源。您还可以为特定节点和特定项目禁用过量使用功能。

9.11.1. 了解计算资源和容器

计算资源的节点强制行为特定于资源类型。

9.11.1.1. 了解容器 CPU 请求

容器可以保证获得其请求的 CPU 量，还可额外消耗节点上提供的超额 CPU，但不会超过容器指定的限制。如果多个容器试图使用超额 CPU，则会根据每个容器请求的 CPU 数量来分配 CPU 时间。

例如，如果一个容器请求了 500m CPU 时间，另一个容器请求了 250m CPU 时间，那么该节点上提供的额外 CPU 时间以 2:1 比例在这两个容器之间分配。如果容器指定了一个限制，它将被限速，无法使用超过指定限制的 CPU。使用 Linux 内核中的 CFS 共享支持强制实施 CPU 请求。默认情况下，使用 Linux 内核中的 CFS 配额支持以 100ms 测量间隔强制实施 CPU 限制，但这可以禁用。

9.11.1.2. 了解容器内存请求

容器可以保证获得其请求的内存量。容器可以使用高于请求量的内存，但一旦超过请求量，就有可能在节点上遇到内存不足情形时被终止。如果容器使用的内存少于请求量，它不会被终止，除非系统任务或守护进程需要的内存量超过了节点资源保留考虑在内的内存量。如果容器指定了内存限制，则超过限制数量时会立即被终止。

9.11.2. 了解过量使用和服务质量类

当节点上调度了没有发出请求的 pod，或者节点上所有 pod 的限制总和超过了机器可用容量时，该节点处于*过量使用*状态。

在过量使用环境中，节点上的 pod 可能会在任意给定时间点尝试使用超过可用量的计算资源。发生这种情况时，节点必须为 pod 赋予不同的优先级。有助于做出此决策的工具称为服务质量 (QoS) 类。

pod 被指定为三个 QoS 类中的一个，带有降序排列：

表 9.2. 服务质量类

优先级	类名称	描述
1（最高）	Guaranteed	如果为所有资源设置了限制和可选请求（不等于 0）并且它们相等，则 pod 被归类为 **Guaranteed**。
2	Burstable	如果为所有资源设置了请求和可选限制（不等于 0）并且它们不相等，则 pod 被归类为 **Burstable**。
3（最低）	BestEffort	如果没有为任何资源设置请求和限制，则 pod 被归类为 **BestEffort**。

内存是一种不可压缩的资源，因此在内存量较低的情况下，优先级最低的容器首先被终止：

- **Guaranteed** 容器优先级最高，并且保证只有在它们超过限制或者系统遇到内存压力且没有优先级更低的容器可被驱除时，才会被终止。
- 在遇到系统内存压力时，**Burstable** 容器如果超过其请求量并且不存在其他 **BestEffort** 容器，则有较大的可能会被终止。
- **BestEffort** 容器被视为优先级最低。系统内存不足时，这些容器中的进程最先被终止。

9.11.2.1. 了解如何为不同的服务质量层级保留内存

您可以使用 **qos-reserved** 参数指定在特定 QoS 级别上 pod 要保留的内存百分比。此功能尝试保留请求的资源，阻止较低 QoS 类中的 pod 使用较高 QoS 类中 pod 所请求的资源。

OpenShift Container Platform 按照如下所示使用 **qos-reserved** 参数：

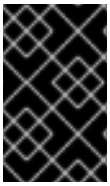
- 值为 **qos-reserved=memory=100%** 时，阻止 **Burstable** 和 **BestEffort** QoS 类消耗较高 QoS 类所请求的内存。这会增加 **BestEffort** 和 **Burstable** 工作负载上为了提高 **Guaranteed** 和 **Burstable** 工作负载的内存资源保障而遭遇 OOM 的风险。
- 值为 **qos-reserved=memory=50%** 时，允许 **Burstable** 和 **BestEffort** QoS 类消耗较高 QoS 类所请求的内存的一半。
- 值为 **qos-reserved=memory=0%** 时，允许 **Burstable** 和 **BestEffort** QoS 类最多消耗节点的所有可分配数量（若可用），但会增加 **Guaranteed** 工作负载不能访问所请求内存的风险。此条件等同于禁用这项功能。

9.11.3. 了解交换内存和 QoS

您可以在节点上默认禁用交换，以便保持服务质量 (QoS) 保障。否则，节点上的物理资源会超额订阅，从而影响 Kubernetes 调度程序在 pod 放置过程中所做的资源保障。

例如，如果两个有保障 pod 达到其内存限制，各个容器可以开始使用交换内存。最终，如果没有足够的交换空间，pod 中的进程可能会因为系统被超额订阅而被终止。

如果不禁用交换，会导致节点无法意识到它们正在经历 **MemoryPressure**，从而造成 pod 无法获得它们在调度请求中索取的内存。这样节点上就会放置更多 pod，进一步增大内存压力，最终增加遭遇系统内存不足 (OOM) 事件的风险。



重要

如果启用了交换，则对于可用内存的资源不足处理驱除阈值将无法正常发挥作用。利用资源不足处理，允许在遇到内存压力时从节点中驱除 pod，并且重新调度到没有此类压力的备选节点上。

9.11.4. 了解节点过量使用

在过量使用的环境中，务必要正确配置节点，以提供最佳的系统行为。

当节点启动时，它会确保为内存管理正确设置内核可微调标识。除非物理内存不足，否则内核应该永不会在内存分配时失败。

为确保这一行为，OpenShift Container Platform 通过将 **vm.overcommit_memory** 参数设置为 **1** 来覆盖默认操作系统设置，从而将内核配置为始终过量使用内存。

OpenShift Container Platform 还通过将 **vm.panic_on_oom** 参数设置为 **0**，将内核配置为不会在内存不足时崩溃。设置为 0 可告知内核在内存不足 (OOM) 情况下调用 oom_killer，以根据优先级终止进程

您可以通过对节点运行以下命令来查看当前的设置：

```
$ sysctl -a |grep commit
```

输**出示例**

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

输**出示例**

```
#...
vm.panic_on_oom = 0
#...
```

注意

节点上应该已设置了上述标记，不需要进一步操作。

您还可以为每个节点执行以下配置：

- 使用 CPU CFS 配额禁用或强制实施 CPU 限制
- 为系统进程保留资源
- 为不同的服务质量等级保留内存

9.11.5. 使用 CPU CFS 配额禁用或强制实施 CPU 限制

默认情况下，节点使用 Linux 内核中的完全公平调度程序 (CFS) 配额支持来强制实施指定的 CPU 限制。

如果禁用了 CPU 限制强制实施，了解其对节点的影响非常重要：

- 如果容器有 CPU 请求，则请求仍由 Linux 内核中的 CFS 共享来实施。
- 如果容器没有 CPU 请求，但没有 CPU 限制，则 CPU 请求默认为指定的 CPU 限值，并由 Linux 内核中的 CFS 共享强制。
- 如果容器同时具有 CPU 请求和限制，则 CPU 请求由 Linux 内核中的 CFS 共享强制实施，且 CPU 限制不会对节点产生影响。

先决条件

- 输入以下命令为您要配置的节点类型获取与静态 **MachineConfigPool** CRD 关联的标签：

```
$ oc edit machineconfigpool <name>
```

例如：

```
$ oc edit machineconfigpool worker
```

输出示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" (1)
  name: worker
```

(1) 标签会出现在 Labels 下。

提示

如果标签不存在，请添加键/值对，例如：

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

流程

1. 为配置更改创建自定义资源 (CR)。

禁用 CPU 限制的示例配置

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
```

```
    pools.operator.machineconfiguration.openshift.io/worker: "" 2
kubeletConfig:
  cpuCfsQuota: false 3
```

1 为 CR 分配一个名称。

2 指定机器配置池中的标签。

3 将 **cpuCfsQuota** 参数设置为 **false**。

2. 运行以下命令来创建 CR：

```
$ oc create -f <file_name>.yaml
```

9.11.6. 为系统进程保留资源

为提供更可靠的调度并且最大程度减少节点资源过量使用，每个节点都可以保留一部分资源供系统守护进程使用（节点上必须运行这些守护进程才能使集群正常工作）。特别是，建议您为内存等不可压缩的资源保留资源。

流程

要明确为非 pod 进程保留资源，请通过指定可用于调度的资源来分配节点资源。如需了解更多详细信息，请参阅“为节点分配资源”。

9.11.7. 禁用节点过量使用

启用之后，可以在每个节点上禁用过量使用。

流程

要在节点中禁用过量使用，请在该节点上运行以下命令：

```
$ sysctl -w vm.overcommit_memory=0
```

9.12. 项目级别限值

为帮助控制过量使用，您可以设置每个项目的资源限值范围，为过量使用无法超过的项目指定内存和 CPU 限值，以及默认值。

如需有关项目级别资源限值的信息，请参阅附加资源。

另外，您可以为特定项目禁用过量使用。

9.12.1. 禁用项目过量使用

启用之后，可以按项目禁用过量使用。例如，您可以允许独立于过量使用配置基础架构组件。

流程

在某个项目中禁用过量使用：

1. 创建或编辑命名空间对象文件。

2. 添加以下注解：

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    quota.openshift.io/cluster-resource-override-enabled: "false" (1)
# ...
```

(1) 将此注解设置为 **false** 可禁用这个命名空间的过量使用。

9.13. 使用垃圾回收释放节点资源

理解并使用垃圾回收。

9.13.1. 了解如何通过垃圾回收移除已终止的容器

容器垃圾回收使用驱除阈值移除已终止的容器。

为垃圾回收设定了驱除阈值时，节点会尝试为任何可从 API 访问的 pod 保留容器。如果 pod 已被删除，则容器也会被删除。只要 pod 没有被删除且没有达到驱除阈值，容器就会保留。如果节点遭遇磁盘压力，它会移除容器，并且无法再通过 **oc logs** 访问其日志。

- **eviction-soft** - 软驱除阈值将驱除阈值与一个由管理员指定的必要宽限期配对。
- **removal-hard** - 硬驱除阈值没有宽限期，如果观察到，OpenShift Container Platform 就会立即采取行动。

下表列出了驱除阈值：

表 9.3. 用于配置容器垃圾回收的变量

节点状况	驱除信号	描述
MemoryPressure	**memory.available**	节点上的可用内存。
DiskPressure	• **nodefs.available** • **nodefs.inodesFree** • **imagefs.available** • **imagefs.inodesFree**	节点根文件系统、**nodefs** 或镜像文件系统 **imagefs** 上的可用磁盘空间或索引节点。

注意

对于 **evictionHard**，您必须指定所有这些参数。如果没有指定所有参数，则只应用指定的参数，垃圾回收将无法正常工作。

如果节点在软驱除阈值上下浮动，但没有超过其关联的宽限期，则对应的节点将持续在 **true** 和 **false** 之间振荡。因此，调度程序可能会做出不当的调度决策。

要防止这种情况的出现，请使用 **remove-pressure-transition-period** 标记来控制 OpenShift Container Platform 在摆脱压力状况前必须等待的时间。OpenShift Container Platform 不会设置在状况切换回 false 前，在指定期限内针对指定压力状况满足的驱除阈值。

9.13.2. 了解如何通过垃圾回收移除镜像

镜像垃圾回收会删除未被任何正在运行的 pod 引用的镜像。

OpenShift Container Platform 根据 **cAdvisor** 报告的磁盘用量决定要从节点中删除哪些镜像。

镜像垃圾收集策略基于两个条件：

- 触发镜像垃圾回收的磁盘用量百分比（以整数表示）。默认值为 85。
- 镜像垃圾回收试尝试释放的磁盘用量百分比（以整数表示）。默认值为 80。

对于镜像垃圾回收，您可以使用自定义资源修改以下任意变量。

表 9.4. 用于配置镜像垃圾回收的变量

设置	描述
imageMinimumGCAge	在通过垃圾收集移除镜像前，未用镜像的最小年龄。默认值为 2m。
imageGCHighThresholdPercent	触发镜像垃圾回收的磁盘用量百分比，以整数表示。默认值为 85。
imageGCLowThresholdPercent	镜像垃圾回收试尝试释放的磁盘用量百分比，以整数表示。默认值为 80。

每次运行垃圾收集器都会检索两个镜像列表：

1. 目前在至少一个 pod 中运行的镜像的列表。
2. 主机上可用镜像的列表。

随着新容器运行，新镜像即会出现。所有镜像都标有时间戳。如果镜像正在运行（上方第一个列表）或者刚被检测到（上方第二个列表），它将标上当前的时间。其余镜像的标记来自于以前的运行。然后，所有镜像都根据时间戳进行排序。

一旦开始回收，首先删除最旧的镜像，直到满足停止条件。

9.13.3. 为容器和镜像配置垃圾回收

作为管理员，您可以通过为各个机器配置池创建 **kubeletConfig** 对象来配置 OpenShift Container Platform 执行垃圾回收的方式。

注意

OpenShift Container Platform 只支持每个机器配置池的一个 **kubeletConfig** 对象。

您可以配置以下几项的任意组合：

- 容器软驱除
- 容器硬驱除
- 镜像驱除

容器垃圾回收会移除已终止的容器。镜像垃圾回收会删除未被任何正在运行的 pod 引用的镜像。

先决条件

1. 输入以下命令为您要配置的节点类型获取与静态 **MachineConfigPool** CRD 关联的标签：

```
$ oc edit machineconfigpool <name>
```

例如：

```
$ oc edit machineconfigpool worker
```

输**出示例**

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

1 标签会出现在 Labels 下。

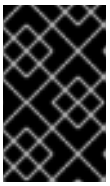
提示

如果标签不存在，请添加键/值对，例如：

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

流程

1. 为配置更改创建自定义资源 (CR)。



重要

如果只有一个文件系统，或者 **/var/lib/kubelet** 和 **/var/lib/containers/** 位于同一文件系统中，则具有最高值的设置会触发驱除操作，因为它们被首先满足。文件系统会触发驱除。

容器垃圾回收 CR 的配置示例：

```
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: KubeletConfig
metadata:
  name: worker-kubeconfig (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    evictionSoft: (3)
      memory.available: "500Mi" (4)
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: (5)
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard: (6)
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s (7)
    imageMinimumGCAge: 5m (8)
    imageGCHighThresholdPercent: 80 (9)
    imageGCLowThresholdPercent: 75 (10)
#...
```

(1) 对象的名称。

(2) 指定机器配置池中的标签。

(3) 对于容器垃圾回收：**eviction Soft** 或 **evictionHard** 的驱除类型。

(4) 对于容器垃圾回收：根据特定驱除触发器信号驱除阈值。

(5) 对于容器垃圾回收：软驱除周期。此参数不适用于 **eviction-hard**。

(6) 对于容器垃圾回收：根据特定驱除触发器信号驱除阈值。对于 **evictionHard**，您必须指定所有这些参数。如果没有指定所有参数，则只应用指定的参数，垃圾回收将无法正常工作。

(7) 对于容器垃圾回收： 摆脱驱除压力状况前等待的持续时间。

(8) 对于镜像垃圾回收：在镜像被垃圾回收移除前，未使用的镜像的最小期限。

(9) 对于镜像垃圾回收：触发镜像垃圾回收的磁盘用量百分比（以整数表示）。

(10) 对于镜像垃圾回收：镜像垃圾回收尝试释放的磁盘用量百分比（以整数表示）。

2. 运行以下命令来创建 CR：

```
$ oc create -f <file_name>.yaml
```

例如：

```
$ oc create -f gc-container.yaml
```

输**出示例**

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

验证

1. 输入以下命令验证垃圾回收是否活跃。您在自定义资源中指定的 Machine Config Pool 会将 **UPDATING** 显示为"true"，直到更改完全实施为止：

```
$ oc get machineconfigpool
```

输**出示例**

```
NAME     CONFIG                                         UPDATED   UPDATING
master   rendered-master-546383f80705bd5aeaba93   True      False
worker   rendered-worker-b4c51bb33ccaae6fc4a6a5   False     True
```

9.14. 使用 NODE TUNING OPERATOR

理解并使用 Node Tuning Operator。

用途
Node Tuning Operator 可以帮助您通过编排 TuneD 守护进程来管理节点级别的性能优化，并使用 Performance Profile 控制器获得低延迟性能。大多数高性能应用程序都需要一定程度的内核级性能优化。Node Tuning Operator 为用户提供了一个统一的、节点一级的 sysctl 管理接口，并可以根据具体用户的需要灵活地添加自定义性能优化设置。

Operator 将为 OpenShift Container Platform 容器化 TuneD 守护进程作为一个 Kubernetes 守护进程集进行管理。它保证了自定义性能优化设置以可被守护进程支持的格式传递到在集群中运行的所有容器化的 TuneD 守护进程中。相应的守护进程会在集群的所有节点上运行，每个节点上运行一个。

在发生触发配置集更改的事件时，或通过接收和处理终止信号安全终止容器化 TuneD 守护进程时，容器化 TuneD 守护进程所应用的节点级设置将被回滚。

Node Tuning Operator 使用 Performance Profile 控制器来实现自动性能优化，从而实现 OpenShift Container Platform 应用程序的低延迟性能。

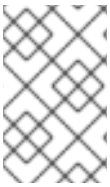
集群管理员配置了性能配置集以定义节点级别的设置，例如：

- 将内核更新至 kernel-rt。
- 为内务选择 CPU。
- 为运行工作负载选择 CPU。

注意

目前，cgroup v2 不支持禁用 CPU 负载均衡。因此，如果您启用了 cgroup v2，则可能无法从性能配置集中获取所需的行为。如果您使用 executeace 配置集，则不建议启用 cgroup v2。

在版本 4.1 及更高版本中，OpenShift Container Platform 标准安装中包含了 Node Tuning Operator。



注意

在早期版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

9.14.1. 访问 Node Tuning Operator 示例规格

使用此流程来访问 Node Tuning Operator 的示例规格。

流程

- 运行以下命令以访问 Node Tuning Operator 示例规格：

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

默认 CR 旨在为 OpenShift Container Platform 平台提供标准的节点级性能优化，它只能被修改来设置 Operator Management 状态。Operator 将覆盖对默认 CR 的任何其他自定义更改。若进行自定义性能优化，请创建自己的 Tuned CR。新创建的 CR 将与默认的 CR 合并，并基于节点或 pod 标识和配置文件优先级对节点应用自定义调整。

警告

虽然在某些情况下，对 pod 标识的支持可以作为自动交付所需调整的一个便捷方式，但我们不鼓励使用这种方法，特别是在大型集群中。默认 Tuned CR 并不带有 pod 标识匹配。如果创建了带有 pod 标识匹配的自定义配置集，则该功能将在此时启用。在以后的 Node Tuning Operator 版本中将弃用 pod 标识功能。

9.14.2. 自定义调整规格

Operator 的自定义资源 (CR) 包含两个主要部分。第一部分是 **profile:**，这是 TuneD 配置集及其名称的列表。第二部分是 **recommend:**，用来定义配置集选择逻辑。

多个自定义调优规格可以共存，作为 Operator 命名空间中的多个 CR。Operator 会检测到是否存在新 CR 或删除了旧 CR。所有现有的自定义性能优化设置都会合并，同时更新容器化 TuneD 守护进程的适当对象。

管理状态

通过调整默认的 Tuned CR 来设置 Operator Management 状态。默认情况下，Operator 处于 Managed 状态，默认的 Tuned CR 中没有 **spec.managementState** 字段。Operator Management 状态的有效值如下：

- Managed: Operator 会在配置资源更新时更新其操作对象
- Unmanaged: Operator 将忽略配置资源的更改
- Removed: Operator 将移除 Operator 置备的操作对象和资源

配置集数据

profile: 部分列出了 TuneD 配置集及其名称。

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

建议的配置集

profile: 选择逻辑通过 CR 的 **recommend:** 部分来定义。**recommend:** 部分是根据选择标准推荐配置集的项目列表。

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

列表中的独立项：

```
- machineConfigLabels: 1
    <mcLabels> 2
  match: 3
    <match> 4
  priority: <priority> 5
  profile: <tuned_profile_name> 6
  operand: 7
```

```
    debug: <bool> 8
    tunedConfig:
      reapply_sysctl: <bool> 9
```

1. 可选。
2. **MachineConfig** 标签的键/值字典。键必须是唯一的。
3. 如果省略，则会假设配置集匹配，除非设置了优先级更高的配置集，或设置了 **machineConfigLabels**。
4. 可选列表。
5. 配置集排序优先级。较低数字表示优先级更高（**0** 是最高优先级）。
6. 在匹配项中应用的 TuneD 配置集。例如 **tuned_profile_1**。
7. 可选操作对象配置。
8. 为 TuneD 守护进程打开或关闭调试。**true** 为打开，**false** 为关闭。默认值为 **false**。
9. 为 TuneD 守护进程打开或关闭 **reapply_sysctl** 功能。选择 **true** 代表开启，**false** 代表关闭。

<match> 是一个递归定义的可选数组，如下所示：

```
- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
    <match> 4
```

1. 节点或 pod 标签名称。
2. 可选的节点或 pod 标签值。如果省略，**<label_name>** 足以匹配。
3. 可选的对象类型（**node** 或 **pod**）。如果省略，会使用 **node**。
4. 可选的 **<match>** 列表。

如果不省略 **<match>**，则所有嵌套的 **<match>** 部分也必须评估为 **true**。否则会假定 **false**，并且不会应用或建议具有对应 **<match>** 部分的配置集。因此，嵌套（子级 **<match>** 部分）会以逻辑 AND 运算来运作。反之，如果匹配 **<match>** 列表中任何一项，整个 **<match>** 列表评估为 **true**。因此，该列表以逻辑 OR 运算来运作。

如果定义了 **machineConfigLabels**，基于机器配置池的匹配会对给定的 **recommend:** 列表项打开。**<mcLabels>** 指定机器配置标签。机器配置会自动创建，以在配置集 **<tuned_profile_name>** 中应用主机设置，如内核引导参数。这包括使用与 **<mcLabels>** 匹配的机器配置选择器查找所有机器配置池，并在分配了找到的机器配置池的所有节点上设置配置集 **<tuned_profile_name>**。要针对同时具有 master 和 worker 角色的节点，您必须使用 master 角色。

列表项 **match** 和 **machineConfigLabels** 由逻辑 OR 操作符连接。**match** 项首先以短电路方式评估。因此，如果它被评估为 **true**，则不考虑 **MachineConfigLabels** 项。

重要

当使用基于机器配置池的匹配时，建议将具有相同硬件配置的节点分组到同一机器配置池中。不遵循这个原则可能会导致在共享同一机器配置池的两个或者多个节点中 TuneD 操作对象导致内核参数冲突。

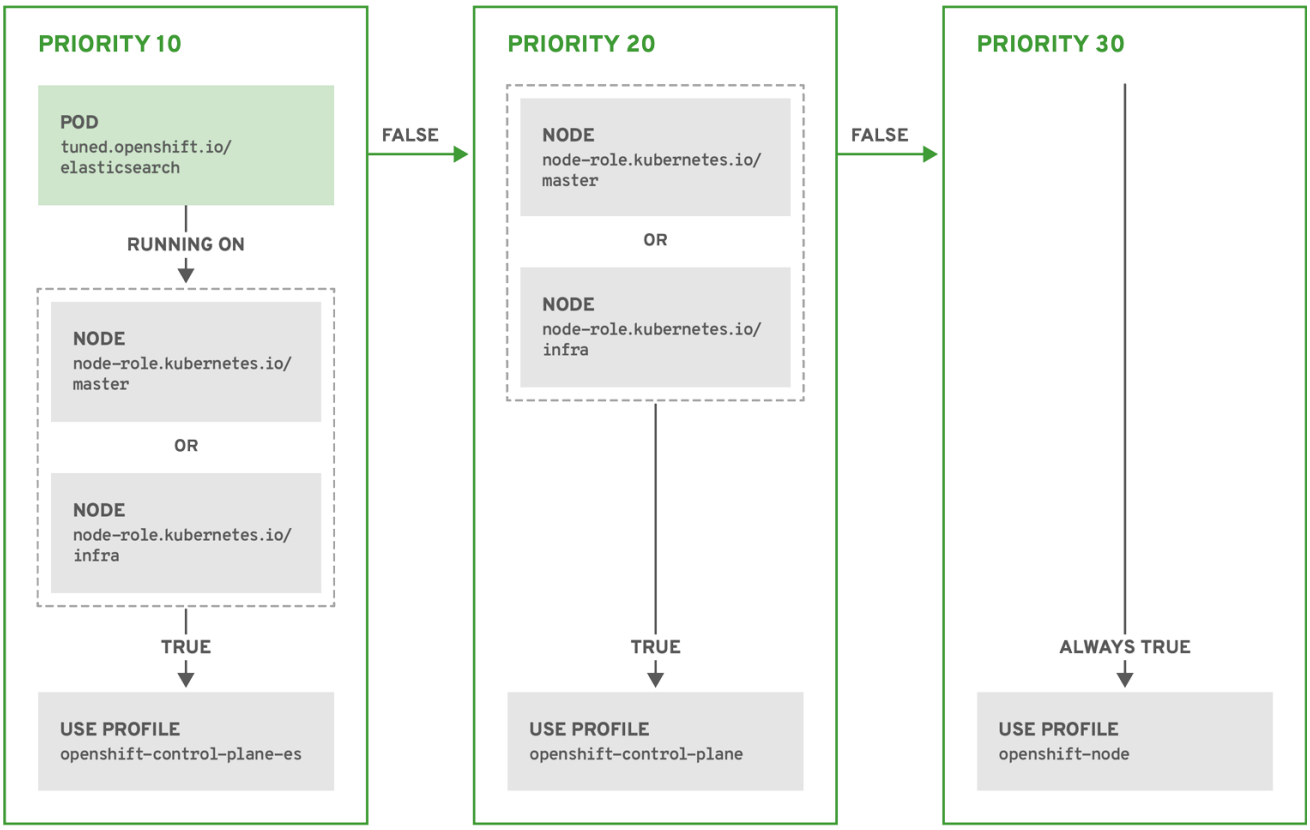
示例：基于节点或 pod 标签的匹配

```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

根据配置集优先级，以上 CR 针对容器化 TuneD 守护进程转换为 **recommend.conf** 文件。优先级最高 (**10**) 的配置集是 **openshift-control-plane-es**，因此会首先考虑它。在给定节点上运行的容器化 TuneD 守护进程会查看同一节点上是否在运行设有 **tuned.openshift.io/elasticsearch** 标签的 pod。如果没有，则整个 **<match>** 部分评估为 **false**。如果存在具有该标签的 pod，为了让 **<match>** 部分评估为 **true**，节点标签也需要是 **node-role.kubernetes.io/master** 或 **node-role.kubernetes.io/infra**。

如果这些标签对优先级为 **10** 的配置集而言匹配，则应用 **openshift-control-plane-es** 配置集，并且不考虑其他配置集。如果节点/pod 标签组合不匹配，则考虑优先级第二高的配置集 (**openshift-control-plane**)。如果容器化 TuneD Pod 在具有标签 **node-role.kubernetes.io/master** 或 **node-role.kubernetes.io/infra** 的节点上运行，则应用此配置集。

最后，配置集 **openshift-node** 的优先级最低 (**30**)。它没有 **<match>** 部分，因此始终匹配。如果给定节点上不匹配任何优先级更高的配置集，它会作为一个适用于所有节点的配置集来设置 **openshift-node** 配置集。



示例：基于机器配置池的匹配

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift node profile with an additional kernel parameter
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

  recommend:
  - machineConfigLabels:
      machineconfiguration.openshift.io/role: "worker-custom"
    priority: 20
    profile: openshift-node-custom
```

为尽量减少节点的重新引导情况，为目标节点添加机器配置池将匹配的节点选择器标签，然后创建上述 Tuned CR，最后创建自定义机器配置池。

特定于云供应商的 TuneD 配置集

使用此功能，所有针对于 OpenShift Container Platform 集群上的云供应商都可以方便地分配 TuneD 配置集。这可实现，而无需添加额外的节点标签或将节点分组到机器配置池中。

这个功能会利用 **spec.providerID** 节点对象值（格式为 **<cloud-provider>://<cloud-provider-specific-id>**），并在 NTO operand 容器中写带有 **<cloud-provider>** 值的文件 **/var/lib/tuned/provider**。然后，TuneD 会使用这个文件的内容来加载 **provider-<cloud-provider>** 配置集（如果这个配置集存在）。

openshift 配置集（**openshift-control-plane** 和 **openshift-node** 配置集都从其中继承设置）现在被更新来使用这个功能（通过使用条件配置集加载）。NTO 或 TuneD 目前不包含任何特定于云供应商的配置集。但是，您可以创建一个自定义配置集 **provider-<cloud-provider>**，它将适用于所有针对于所有云供应商的集群节点。

GCE 云供应商配置集示例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
    name: provider-gce
```

注意

由于配置集的继承，**provider-<cloud-provider>** 配置集中指定的任何设置都会被 **openshift** 配置集及其子配置集覆盖。

9.14.3. 在集群中设置默认配置集

以下是在集群中设置的默认配置集。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift
    name: openshift
  recommend:
  - profile: openshift-control-plane
    priority: 30
    match:
    - label: node-role.kubernetes.io/master
```

```
    - label: node-role.kubernetes.io/infra
  - profile: openshift-node
    priority: 40
```

从 OpenShift Container Platform 4.9 开始，所有 OpenShift TuneD 配置集都随 TuneD 软件包一起提供。您可以使用 **oc exec** 命令查看这些配置集的内容：

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{,-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

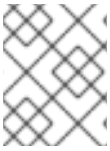
9.14.4. 支持的 TuneD 守护进程插件

在使用 Tuned CR 的 **profile:** 部分中定义的自定义配置集时，以下 TuneD 插件都受到支持，但 **[main]** 部分除外：

- audio
- cpu
- disk
- eeepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm
- bootloader

其中一些插件提供了不受支持的动态性能优化功能。目前不支持以下 TuneD 插件：

- script
- systemd



注意

TuneD bootloader 插件只支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。

其他资源

- 可用的 TuneD 插件
- TuneD 入门

9.15. 配置每个节点的最大 POD 数量

有两个参数控制可调度到节点的 pod 数量上限，分别为 **podsPerCore** 和 **maxPods**。如果您同时使用这两个选项，则取两者中较小的限制来限制节点上的 pod 数。

例如，如果将一个有 4 个处理器内核的节点上的 **podsPerCore** 设置为 **10**，则该节点上允许的 pod 数量上限为 40。

先决条件

1. 输入以下命令为您要配置的节点类型获取与静态 **MachineConfigPool** CRD 关联的标签：

```
$ oc edit machineconfigpool <name>
```

例如：

```
$ oc edit machineconfigpool worker
```

输**出示例**

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

1 标签会出现在 Labels 下。

提示

如果标签不存在，请添加键/值对，例如：

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

流程

1. 为配置更改创建自定义资源 (CR)。

max-pods CR 配置示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    podsPerCore: 10 (3)
    maxPods: 250 (4)
#...
```

1. 为 CR 分配一个名称。
2. 指定机器配置池中的标签。
3. 根据节点的处理器内核数限制节点上可运行的 pod 数量。
4. 将节点上可运行的 pod 数量指定为一个固定值，而不考虑节点的属性。

注意

将 **podsPerCore** 设置为 **0** 可禁用这个限制。

在上例中，**podsPerCore** 的默认值为 **10**，**maxPods** 的默认值则为 **250**。这意味着，除非节点有 25 个以上的内核，否则 **podsPerCore** 就是默认的限制因素。

2. 运行以下命令来创建 CR：

```
$ oc create -f <file_name>.yaml
```

验证

1. 列出 **MachineConfigPool** CRD 以查看是否应用了更改。如果 Machine Config Controller 抓取到更改，则 **UPDATING** 列会报告 **True**：

```
$ oc get machineconfigpools
```

输出示例

```
NAME     CONFIG                    UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     False      False
worker   worker-8cecd1236b33ee3f8a5e   False     True       False
```

更改完成后，**UPDATED** 列会报告 **True**。

```
$ oc get machineconfigpools
```

输出示例

```
NAME     CONFIG                 UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     True       False
worker   worker-8cecd1236b33ee3f8a5e   True      False      False
```

9.16. 使用静态 IP 地址进行机器扩展

在将集群部署到使用静态 IP 地址运行节点后，您可以扩展机器实例或机器集以使用这些静态 IP 地址之一。

其他资源

- vSphere 节点的静态 IP 地址

9.16.1. 扩展机器以使用静态 IP 地址

您可以扩展额外的机器集，以使用集群中的预定义静态 IP 地址。对于此配置，您需要创建机器资源 YAML 文件，然后在此文件中定义静态 IP 地址。

重要

vSphere 节点的静态 IP 地址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。

先决条件

- 您在 **install-config.yaml** 文件中包括 **featureSet:TechPreviewNoUpgrade** 作为初始条目。
- 您已部署了至少一个具有配置静态 IP 地址的节点运行的集群。

流程

1. 创建机器资源 YAML 文件，并在 **network** 参数中定义静态 IP 地址网络信息。

使用 network 参数中定义的静态 IP 地址信息的机器资源 YAML 文件示例。

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  creationTimestamp: null
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machine-role: <role>
    machine.openshift.io/cluster-api-machine-type: <role>
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  lifecycleHooks: {}
```

```
    metadata: {}
    providerSpec:
      value:
        apiVersion: machine.openshift.io/v1beta1
        credentialsSecret:
          name: vsphere-cloud-credentials
        diskGiB: 120
        kind: VSphereMachineProviderSpec
        memoryMiB: 8192
        metadata:
          creationTimestamp: null
        network:
          devices:
          - gateway: 192.168.204.1 (1)
            ipAddrs:
            - 192.168.204.8/24 (2)
            nameservers: (3)
            - 192.168.204.1
            networkName: qe-segment-204
        numCPUs: 4
        numCoresPerSocket: 2
        snapshot: ""
        template: <vm_template_name>
        userDataSecret:
          name: worker-user-data
        workspace:
          datacenter: <vcenter_datacenter_name>
          datastore: <vcenter_datastore_name>
          folder: <vcenter_vm_folder_path>
          resourcepool: <vsphere_resource_pool>
          server: <vcenter_server_ip>
    status: {}
```

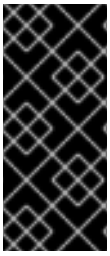
1. 网络接口默认网关的 IP 地址。
2. 列出安装程序传递给网络接口的 IPv4、IPv6 或两个 IP 地址。两个 IP 系列都必须为默认网络使用相同的网络接口。
3. 列出 DNS 名称服务器。您可以定义最多 3 个 DNS 名称服务器。如果一个 DNS 名称服务器变得不可访问，请考虑定义多个 DNS 名称服务器以利用 DNS 解析。

- 在终端中输入以下命令来创建 **machine** 自定义资源(CR)：

```
$ oc create -f <file_name>.yaml
```

9.16.2. 使用配置的静态 IP 地址的机器设置扩展

您可以使用机器集来扩展带有配置的静态 IP 地址的机器。

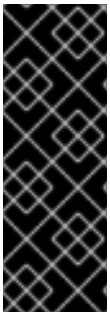


重要

vSphere 节点的静态 IP 地址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。

在将机器集配置为为机器请求静态 IP 地址后，机器控制器会在 **openshift-machine-api** 命名空间中创建 **IPAddressClaim** 资源。然后，外部控制器会创建一个 **IPAddress** 资源，并将任何静态 IP 地址绑定到 **IPAddressClaim** 资源。



重要

您的组织可能会使用许多类型的 IP 地址管理(IPAM)服务。如果要在 OpenShift Container Platform 上启用特定的 IPAM 服务，您可能需要在 YAML 定义中手动创建 **IPAddressClaim** 资源，然后在 **oc** CLI 中输入以下命令来将静态 IP 地址绑定到这个资源：

```
$ oc create -f <ipaddressclaim_filename>
```

以下示例演示了 **IPAddressClaim** 资源的示例：

```
kind: IPAddressClaim
metadata:
  finalizers:
  - machine.openshift.io/ip-claim-protection
  name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  namespace: openshift-machine-api
spec:
  poolRef:
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
status: {}
```

机器控制器更新状态为 **IPAddressClaimed** 的机器，以指示静态 IP 地址被成功绑定到 **IPAddressClaim** 资源。机器控制器将相同的状态应用到具有多个 **IPAddressClaim** 资源的机器，每个都包含一个绑定静态 IP 地址。机器控制器会创建一个虚拟机，并将静态 IP 地址应用到机器配置的 **providerSpec** 中列出的任何节点。

9.16.3. 使用机器集扩展带有配置的静态 IP 地址的机器

您可以使用机器集来扩展带有配置的静态 IP 地址的机器。

重要

vSphere 节点的静态 IP 地址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。

该流程中的示例演示了使用控制器在机器集中扩展机器。

先决条件

- 您在 **install-config.yaml** 文件中包括 **featureSet:TechPreviewNoUpgrade** 作为初始条目。
- 您已部署了至少一个具有配置静态 IP 地址的节点运行的集群。

流程

1. 通过在机器集的 YAML 文件的 **network.devices.addressesFromPools** schema 中指定 IP 池信息来配置机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  annotations:
    machine.openshift.io/memoryMb: "8192"
    machine.openshift.io/vCPU: "4"
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  replicas: 0
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        ipam: "true"
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      lifecycleHooks: {}
      metadata: {}
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1beta1
          credentialsSecret:
            name: vsphere-cloud-credentials
          diskGiB: 120
          kind: VSphereMachineProviderSpec
          memoryMiB: 8192
          metadata: {}
          network:
            devices:
            - addressesFromPools: (1)
              - group: ipamcontroller.example.io
                name: static-ci-pool
                resource: IPPool
```

```
      nameservers:
      - "192.168.204.1" 2
      networkName: qe-segment-204
  numCPUs: 4
  numCoresPerSocket: 2
  snapshot: ""
  template: rvanderp4-dev-9n5wg-rhcos-generated-region-generated-zone
  userDataSecret:
    name: worker-user-data
  workspace:
    datacenter: IBMCdatacenter
    datastore: /IBMCdatacenter/datastore/vsanDatastore
    folder: /IBMCdatacenter/vm/rvanderp4-dev-9n5wg
    resourcePool: /IBMCdatacenter/host/IBMCcluster//Resources
    server: vcenter.ibmc.devcluster.openshift.com
```

1 指定一个 IP 池，它列出了静态 IP 地址或静态 IP 地址的范围。IP 池可以是对自定义资源定义 (CRD) 或 **IPAddressClaims** 资源处理器支持的资源的引用。机器控制器访问机器集配置中列出的静态 IP 地址，然后为每个机器分配每个地址。

2 列出名称服务器.您必须为接收静态 IP 地址的节点指定名称服务器，因为动态主机配置协议 (DHCP) 网络配置不支持静态 IP 地址。

2. 在 **oc** CLI 中输入以下命令来扩展机器集：

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

或者：

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

扩展每台机器后，机器控制器会创建一个 **IPAddresssClaim** 资源。

3. 可选：输入以下命令检查 **openshift-machine-api** 命名空间中是否存在 **IPAddressClaim** 资源：

```
$ oc get ipaddressclaims.ipam.cluster.x-k8s.io -n openshift-machine-api
```

列出 openshift-machine-api 命名空间中列出两个 IP 池的 oc CLI 输出示例

```
NAME                                      POOL NAME       POOL KIND
cluster-dev-9n5wg-worker-0-m7529-claim-0-0   static-ci-pool   IPPool
cluster-dev-9n5wg-worker-0-wdqkt-claim-0-0   static-ci-pool   IPPool
```

4. 输入以下命令来创建 **IPAddress** 资源：

```
$ oc create -f ipaddress.yaml
```

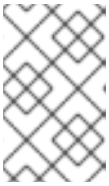
以下示例显示了带有定义网络配置信息的 **IPAddress** 资源，以及一个定义的静态 IP 地址：

```
apiVersion: ipam.cluster.x-k8s.io/v1alpha1
kind: IPAddress
metadata:
  name: cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0
```

```
  namespace: openshift-machine-api
spec:
  address: 192.168.204.129
  claimRef: 1
    name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  gateway: 192.168.204.1
  poolRef: 2
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
  prefix: 23
```

1 目标 **IPAddressClaim** 资源的名称。

2 有关节点的静态 IP 地址或地址的详情。



注意

默认情况下，外部控制器会自动扫描机器集中的任何资源，以了解可识别的地址池类型。当外部控制器找到 **IPAddress** 资源中定义的 **kind: IPPool** 时，控制器会将任何静态 IP 地址绑定到 **IPAddressClaim** 资源。

5. 使用对 **IPAddress** 资源的引用更新 **IPAddressClaim** 状态：

```
$ oc --type=merge patch IPAddressClaim cluster-dev-9n5wg-worker-0-m7529-claim-0-0 -
p='{"status":{"addressRef": {"name": "cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0"}}}' -
n openshift-machine-api --subresource=status
```

第 10 章 安装后的网络配置

安装 OpenShift Container Platform 后，您可以按照您的要求进一步扩展和自定义网络。

10.1. CLUSTER NETWORK OPERATOR 配置

集群网络的配置作为 Cluster Network Operator(CNO)配置的一部分指定，并存储在名为 **cluster** 的自定义资源(CR)对象中。CR 指定 **operator.openshift.io** API 组中的 **Network** API 的字段。

CNO 配置在集群安装过程中从 **Network.config.openshift.io** API 组中的 **Network** API 继承以下字段：

clusterNetwork

从中分配 Pod IP 地址的 IP 地址池。

serviceNetwork

服务的 IP 地址池.

defaultNetwork.type

集群网络插件。**OVNKubernetes** 是安装期间唯一支持的插件。

注意

在集群安装后，您只能修改 **clusterNetwork** IP 地址范围。默认网络类型只能通过迁移从 OpenShift SDN 改为 OVN-Kubernetes。

10.2. 启用集群范围代理

Proxy 对象用于管理集群范围出口代理。如果在安装或升级集群时没有配置代理，则 **Proxy** 对象仍会生成，但它会有一个空的 **spec**。例如：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

集群管理员可以通过修改这个 **cluster Proxy** 对象来配置 OpenShift Container Platform 的代理。

注意

只支持名为 **cluster** 的 **Proxy** 对象，且无法创建额外的代理。

警告

启用集群范围代理会导致 Machine Config Operator (MCO)触发节点重启。

先决条件

- 集群管理员权限
- 已安装 OpenShift Container Platform **oc** CLI 工具

流程

1. 创建包含代理 HTTPS 连接所需的额外 CA 证书的 ConfigMap。

注意

如果代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名，您可以跳过这一步。

a. 利用以下内容，创建一个名为 **user-ca-bundle.yaml** 的文件，并提供 PEM 编码证书的值：

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1. 这个数据键必须命名为 **ca-bundle.crt**。
2. 一个或多个 PEM 编码的 X.509 证书，用来为代理的身份证书签名。
3. 从 **Proxy** 对象引用的配置映射名称。
4. 配置映射必须位于 **openshift-config** 命名空间中。

b. 从此文件创建配置映射：

```
$ oc create -f user-ca-bundle.yaml
```

2. 使用 **oc edit** 命令修改 **Proxy** 对象：

```
$ oc edit proxy/cluster
```

3. 为代理配置所需的字段：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
```

```
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1. 用于创建集群外 HTTP 连接的代理 URL。URL 方案必须是 **http**。
2. 用于创建集群外 HTTPS 连接的代理 URL。URL 方案必须是 **http** 或 **https**。指定支持 URL 方案的代理的 URL。例如，如果大多数代理被配置为使用 **https**，则大多数代理都会报告错误，但它们只支持 **http**。此失败消息可能无法传播到日志，并可能显示为网络连接失败。如果使用侦听来自集群的 **https** 连接的代理，您可能需要配置集群以接受代理使用的 CA 和证书。
3. 要排除代理的目标域名、域、IP 地址或其他网络 CIDR 的逗号分隔列表。

   在域前面加 **.** 来仅匹配子域。例如： **.y.com** 匹配 **x.y.com**，但不匹配 **y.com**。使用 * 可对所有目的地绕过所有代理。如果您扩展了未包含在安装配置中 **networking.machineNetwork[].cidr** 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

   如果未设置 **httpProxy** 或 **httpsProxy** 字段，则此字段将被忽略。
4. 将 **httpProxy** 和 **httpsProxy** 值写进状态之前，执行就绪度检查时要使用的一个或多个集群外部 URL。
5. 引用 **openshift-config** 命名空间中的 ConfigMap，其包含代理 HTTPS 连接所需的额外 CA 证书。注意 ConfigMap 必须已经存在，然后才能在这里引用它。此字段是必需的，除非代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名。

4. 保存文件以使改变生效。

10.3. 将 DNS 设置为私有

部署集群后，您可以修改其 DNS 使其只使用私有区。

流程

1. 查看集群的 **DNS** 自定义资源：

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

输**出示例**

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
```

```
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

请注意，**spec** 部分包含一个私有区和一个公共区。

2. 修补 **DNS** 自定义资源以删除公共区：

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec": {"publicZone": null}}'
dns.config.openshift.io/cluster patched
```

因为 Ingress Controller 在创建 **Ingress** 对象时会参考 **DNS** 定义，因此当您创建或修改 **Ingress** 对象时，只会创建私有记录。

重要

在删除公共区时，现有 Ingress 对象的 DNS 记录不会修改。

3. 可选：查看集群的 **DNS** 自定义资源，并确认已删除公共区：

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

输**出示例**

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>-wfpg4: owned
status: {}
```

10.4. 配置集群入口流量

OpenShift Container Platform 提供了以下从集群外部与集群中运行的服务进行通信的方法：

- 如果您有 HTTP/HTTPS，请使用 Ingress Controller。

- 如果您有 HTTPS 之外的 TLS 加密协议，如使用 SNI 标头的 TLS，请使用 Ingress Controller。
- 否则，请使用负载均衡器、外部 IP 或节点端口。

方法	用途
使用 Ingress Controller	允许访问 HTTP/HTTPS 流量和 HTTPS 以外的 TLS 加密协议（例如，使用 SNI 标头的 TLS）。
使用负载均衡器服务自动分配外部 IP	允许流量通过从池分配的 IP 地址传到非标准端口。
手动将外部 IP 分配给服务	允许流量通过特定的 IP 地址传到非标准端口。
配置 **NodePort**	在集群中的所有节点上公开某一服务。

10.5. 配置节点端口服务范围

作为集群管理员，您可以扩展可用的节点端口范围。如果您的集群使用大量节点端口，可能需要增加可用端口的数量。

默认端口范围为 **30000-32767**。您永远不会缩小端口范围，即使您首先将其扩展超过默认范围。

10.5.1. 先决条件

- 集群基础架构必须允许访问您在扩展范围内指定的端口。例如，如果您将节点端口范围扩展到 **30000-32900**，防火墙或数据包过滤配置必须允许 **32768-32900** 端口范围。

10.5.1.1. 扩展节点端口范围

您可以扩展集群的节点端口范围。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 使用具有 **cluster-admin** 权限的用户登陆到集群。

流程

1. 要扩展节点端口范围，请输入以下命令。将 **<port>** 替换为新范围内的最大端口号码。

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
  '{
    "spec":
      { "serviceNodePortRange": "30000-<port>" }
  }'
```

提示

您还可以应用以下 YAML 来更新节点端口范围：

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

输**出示例**

```
network.config.openshift.io/cluster patched
```

2. 要确认配置是活跃的，请输入以下命令。应用更新可能需要几分钟。

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo '"service-node-port-range":["[[:digit:]]+-[[:digit:]]+"]'
```

输**出示例**

```
"service-node-port-range":["30000-33000"]
```

10.6. 配置 IPSEC 加密

启用 IPsec 后，OVN-Kubernetes 网络插件的节点之间的所有网络流量都通过加密的隧道传输。

默认禁用 IPsec。

10.6.1. 先决条件

- 集群必须使用 OVN-Kubernetes 网络插件。

10.6.1.1. 启**用 IPsec 加密**

作为集群管理员，您可以在集群和外部 IPsec 端点之间，启用 pod 到 pod IPsec 加密和 IPsec 加密。

您可以使用以下模式之一配置 IPsec：

- **Full** ：pod 到 pod 和外部流量的加密
- **External** ：对外部流量进行加密

如果您需要除 pod 到 pod 流量加密外，还需要对外部流量配置加密，则需要完成"为外部流量配置 IPsec 加密"流程。

先决条件

- 安装 OpenShift CLI (**oc**) 。

- 以具有 **cluster-admin** 权限的用户身份登录集群。
- 您已将集群 MTU 的大小减少为 **46** 字节，以允许 IPsec ESP 标头的开销。

流程

1. 要启用 IPsec 加密，请输入以下命令：

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "ipsecConfig":{
          "mode":<mode>
        }}}}}'
```

其中：

模式

指定 **External** 以加密到外部主机的流量，或者指定 **Full** 来加密 pod 到 pod 流量，以及可选的到外部主机的流量。默认情况下禁用 IPsec。

2. 可选：如果您需要加密到外部主机的流量，请完成"为外部流量配置 IPsec 加密"流程。

验证

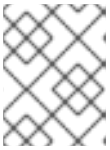
1. 要查找 OVN-Kubernetes data plane pod 的名称，请输入以下命令：

```
$ oc get pods -n openshift-ovn-kubernetes -l=app=ovnkube-node
```

输出示例

```
ovnkube-node-5xqbf                 8/8     Running   0          28m
ovnkube-node-6mwcx                 8/8     Running   0          29m
ovnkube-node-ck5fr                 8/8     Running   0          31m
ovnkube-node-fr4ld                 8/8     Running   0          26m
ovnkube-node-wgs4l                 8/8     Running   0          33m
ovnkube-node-zfvcl                 8/8     Running   0          34m
```

2. 运行以下命令，验证集群中是否启用了 IPsec：



注意

作为集群管理员，您可以在 IPsec 配置为 **Full** 模式时验证集群中 pod 之间是否启用了 IPsec。此步骤不会验证 IPsec 是否在集群和外部主机间工作。

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-node-<XXXXX> ovn-nbctl --no-leader-only get nb_global . ipsec
```

其中：

<XXXXX>

指定上一步中 pod 的随机字符序列。

输**出示例**

```
true
```

10.7. 配置网络策略

作为集群管理员或项目管理员，您可以为项目配置网络策略。

10.7.1. 关于网络策略

在使用支持 Kubernetes 网络策略的网络插件的集群中，网络隔离完全由 **NetworkPolicy** 对象控制。在 OpenShift Container Platform 4.15 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。

警告

网络策略不适用于主机网络命名空间。启用主机网络的 Pod 不受网络策略规则的影响。但是，连接到 host-networked pod 的 pod 会受到网络策略规则的影响。

网络策略无法阻止来自 localhost 或来自其驻留的节点的流量。

默认情况下，项目中的所有 pod 都可被其他 pod 和网络端点访问。要在一个项目中隔离一个或多个 Pod，您可以在该项目中创建 **NetworkPolicy** 对象来指示允许的入站连接。项目管理员可以在自己的项目中创建和删除 **NetworkPolicy** 对象。

如果一个 pod 由一个或多个 **NetworkPolicy** 对象中的选择器匹配，那么该 pod 将只接受至少被其中一个 **NetworkPolicy** 对象所允许的连接。未被任何 **NetworkPolicy** 对象选择的 pod 可以完全访问。

网络策略仅适用于 TCP、UDP、ICMP 和 SCTP 协议。其他协议不会受到影响。

以下示例 **NetworkPolicy** 对象演示了支持不同的情景：

- 拒绝所有流量：
  要使项目默认为拒绝流量，请添加一个匹配所有 pod 但不接受任何流量的 **NetworkPolicy** 对象：

  ```
  kind: NetworkPolicy
  apiVersion: networking.k8s.io/v1
  metadata:
    name: deny-by-default
  spec:
    podSelector: {}
    ingress: []
  ```

- 只允许 OpenShift Container Platform Ingress Controller 的连接：
  要使项目只允许 OpenShift Container Platform Ingress Controller 的连接，请添加以下 **NetworkPolicy** 对象。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

- 只接受项目中 pod 的连接：
  要使 pod 接受同一项目中其他 pod 的连接，但拒绝其他项目中所有 pod 的连接，请添加以下 **NetworkPolicy** 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
```

- 仅允许基于 pod 标签的 HTTP 和 HTTPS 流量：
  要对带有特定标签（以下示例中的 **role=frontend**）的 pod 仅启用 HTTP 和 HTTPS 访问，请添加类似如下的 **NetworkPolicy** 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- 使用命名空间和 pod 选择器接受连接：
  要通过组合使用命名空间和 pod 选择器来匹配网络流量,您可以使用类似如下的 **NetworkPolicy** 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
```

```
    name: allow-pod-and-namespace-both
  spec:
    podSelector:
      matchLabels:
        name: test-pods
    ingress:
      - from:
        - namespaceSelector:
            matchLabels:
              project: project_name
          podSelector:
            matchLabels:
              name: test-pods
```

NetworkPolicy 对象是可添加的；也就是说，您可以组合多个 **NetworkPolicy** 对象来满足复杂的网络要求。

例如，对于以上示例中定义的 **NetworkPolicy** 对象，您可以在同一个项目中定义 **allow-same-namespace** 和 **allow-http-and-https** 策略。因此，允许带有标签 **role=frontend** 的 pod 接受每一策略所允许的任何连接。即，任何端口上来自同一命名空间中的 pod 的连接，以及端口 **80** 和 **443** 上的来自任意命名空间中 pod 的连接。

10.7.1.1. 使用 allow-from-router 网络策略

使用以下 **NetworkPolicy** 来允许外部流量，而不考虑路由器配置：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" (1)
  podSelector: {}
  policyTypes:
  - Ingress
```

(1) **policy-group.network.openshift.io/ingress:""** 标签支持 OpenShift-SDN 和 OVN-Kubernetes。

10.7.1.2. 使用 allow-from-hostnetwork 网络策略

添加以下 **allow-from-hostnetwork NetworkPolicy** 对象来指示来自主机网络 pod 的流量：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
```

```
      matchLabels:
        policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

10.7.2. 示例 NetworkPolicy 对象

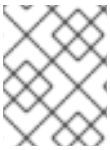
下文解释了示例 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy 对象的名称。
2. 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。
3. 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。
4. 接受流量的一个或多个目标端口的列表。

10.7.3. 使用 CLI 创建网络策略

要定义细致的规则来描述集群中命名空间允许的入口或出口网络流量，您可以创建一个网络策略。



注意

如果使用具有 **cluster-admin** 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。

- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略规则：
   a. 创建一个 **<policy_name>.yaml** 文件：

   ```
   $ touch <policy_name>.yaml
   ```

   其中：

   <policy_name>
   指定网络策略文件名。

   b. 在您刚才创建的文件中定义网络策略，如下例所示：

   拒绝来自所有命名空间中的所有 pod 的入口流量

   这是一个基本的策略，阻止配置其他网络策略所允许的跨 pod 流量以外的所有跨 pod 网络。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: deny-by-default
   spec:
     podSelector: {}
     policyTypes:
     - Ingress
     ingress: []
   ```

   允许来自所有命名空间中的所有 pod 的入口流量

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: allow-same-namespace
   spec:
     podSelector:
     ingress:
     - from:
       - podSelector: {}
   ```

   允许从特定命名空间中到一个 pod 的入口流量

   此策略允许流量从在 **namespace-y** 中运行的容器集到标记 **pod-a** 的 pod。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: allow-traffic-pod
   spec:
     podSelector:
   ```

```
  matchLabels:
    pod: pod-a
policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: namespace-y
```

2. 运行以下命令来创建网络策略对象：

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

其中：

<policy_name>

指定网络策略文件名。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

```
networkpolicy.networking.k8s.io/deny-by-default created
```

注意

如果您使用 **cluster-admin** 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式创建网络策略。

10.7.4. 使用网络策略配置多租户隔离

您可以配置项目，使其与其他项目命名空间中的 pod 和服务分离。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。

流程

1. 创建以下 **NetworkPolicy** 对象：

   a. 名为 **allow-from-openshift-ingress** 的策略。

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
```

```
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

注意

policy-group.network.openshift.io/ingress: "" 是 OpenShift SDN 的首选命名空间选择器标签。您可以使用 **network.openshift.io/policy-group: ingress** 命名空间选择器标签，但这是一个比较旧的用法。

b. 名为 **allow-from-openshift-monitoring** 的策略：

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c. 名为 **allow-same-namespace** 的策略：

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

d. 名为 **allow-from-kube-apiserver-operator** 的策略：

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
```

```
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
    policyTypes:
    - Ingress
  EOF
```

如需了解更多详细信息，请参阅 新的**kube-apiserver-operator** Webhook 控制器验证 Webhook 的健康状况。

2. 可选：要确认当前项目中存在网络策略，请输入以下命令：

```
$ oc describe networkpolicy
```

输**出示例**

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

10.7.5. 为新项目创建默认网络策略

作为集群管理员，您可以在创建新项目时修改新项目模板，使其自动包含 **NetworkPolicy** 对象。

10.7.6. 为新项目修改模板

作为集群管理员，您可以修改默认项目模板，以便使用自定义要求创建新项目。

创建自己的自定义项目模板：

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 以具有 **cluster-admin** 特权的用户身份登录。
2. 生成默认项目模板：

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 使用文本编辑器，通过添加对象或修改现有对象来修改生成的 **template.yaml** 文件。
4. 项目模板必须创建在 **openshift-config** 命名空间中。加载修改后的模板：

```
$ oc create -f template.yaml -n openshift-config
```

5. 使用 Web 控制台或 CLI 编辑项目配置资源。
   - 使用 Web 控制台：
     i. 导航至 **Administration → Cluster Settings** 页面。
     ii. 单击 **Configuration** 以查看所有配置资源。
     iii. 找到 **Project** 的条目，并点击 **Edit YAML**。
   - 使用 CLI：
     i. 编辑 **project.config.openshift.io/cluster** 资源：

```
$ oc edit project.config.openshift.io/cluster
```

6. 更新 **spec** 部分，使其包含 **projectRequestTemplate** 和 **name** 参数，再设置您上传的项目模板的名称。默认名称为 **project-request**。

   带有自定义项目模板的项目配置资源

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
# ...
spec:
```

```
  projectRequestTemplate:
    name: <template_name>
# ...
```

7. 保存更改后，创建一个新项目来验证是否成功应用了您的更改。

10.7.6.1. 在新项目模板中添加网络策略

作为集群管理员，您可以在新项目的默认模板中添加网络策略。OpenShift Container Platform 将自动创建项目中模板中指定的所有 **NetworkPolicy** 对象。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的默认 CNI 网络插件，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您需要使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您必须已为新项目创建了自定义的默认项目模板。

流程

1. 运行以下命令来编辑新项目的默认模板：

```
$ oc edit template <project_template> -n openshift-config
```

将 **<project_template>** 替换为您为集群配置的缺省模板的名称。默认模板名称为 **project-request**。

2. 在模板中，将每个 **NetworkPolicy** 对象作为一个元素添加到 **objects** 参数中。**objects** 参数可以是一个或多个对象的集合。
在以下示例中，**objects** 参数集合包括几个 **NetworkPolicy** 对象。

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
```

```
    podSelector: {}
    policyTypes:
    - Ingress
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
    policyTypes:
    - Ingress
...
```

3. 可选：通过运行以下命令创建一个新项目，来确认您的网络策略对象已被成功创建：

   a. 创建一个新项目：

   ```
   $ oc new-project <project> 1
   ```

   1 将 **<project>** 替换为您要创建的项目的名称。

   b. 确认新项目模板中的网络策略对象存在于新项目中：

   ```
   $ oc get networkpolicy
   NAME                           POD-SELECTOR   AGE
   allow-from-openshift-ingress   <none>         7s
   allow-from-same-namespace      <none>         7s
   ```

10.8. 优化路由

OpenShift Container Platform HAProxy 路由器可以扩展或配置以优化性能。

10.8.1. Ingress Controller（router）性能的基线

OpenShift Container Platform Ingress Controller 或路由器是使用路由和入口配置的应用程序和服务的入口流量的入站点。

当根据每秒处理的 HTTP 请求来评估单个 HAProxy 路由器性能时，其性能取决于多个因素。特别是：

- HTTP keep-alive/close 模式
- 路由类型
- 对 TLS 会话恢复客户端的支持
- 每个目标路由的并行连接数

- 目标路由数
- 后端服务器页面大小
- 底层基础结构（网络/SDN 解决方案、CPU 等）

具体环境中的性能会有所不同，红帽实验室在一个有 4 个 vCPU/16GB RAM 的公有云实例中进行测试。一个 HAProxy 路由器处理由后端终止的 100 个路由服务提供 1kB 静态页面，每秒处理以下传输数。

在 HTTP 的 keep-alive 模式下：

Encryption	LoadBalancerService	HostNetwork
none	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

在 HTTP 关闭（无 keep-alive）情境中：

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

默认 Ingress Controller 配置用于将 **spec.tuningOptions.threadCount** 字段设置为 **4**。测试了两个不同的端点发布策略：Load Balancer Service 和 Host Network。TLS 会话恢复用于加密路由。使用 HTTP keep-alive 设置，单个 HAProxy 路由器可在页面大小小到 8 kB 时充满 1 Gbit NIC。

当在使用现代处理器的裸机中运行时，性能可以期望达到以上公有云实例测试性能的大约两倍。这个开销是由公有云的虚拟化层造成的，基于私有云虚拟化的环境也会有类似的开销。下表是有关在路由器后面的应用程序数量的指导信息：

应用程序数量	应用程序类型
5-10	静态文件/web 服务器或者缓存代理
100-1000	生成动态内容的应用程序

取决于所使用的技术，HAProxy 通常可支持最多 1000 个程序的路由。Ingress Controller 性能可能会受其后面的应用程序的能力和性能的限制，如使用的语言，静态内容或动态内容。

如果有多个服务于应用程序的 Ingress 或路由器，则应该使用路由器分片（router sharding）以帮助横向扩展路由层。

10.8.2. 配置 Ingress Controller 存活度、就绪度和启动探测

集群管理员可为由 OpenShift Container Platform Ingress Controller（路由器）管理的路由器部署配置 kubelet 存活度、就绪度和启动探测的超时值。路由器的存活度和就绪度探测使用默认值 1 秒，这在网络或运行时性能严重降级时太短。探测超时可能会导致中断应用程序连接的路由器重启。设置较大的超时值可以降低不必要的和不需要的重启的风险。

您可以更新路由器容器的 **livenessProbe**、**readinessProbe** 和 **startProbe** 参数上的 **timeoutSeconds** 值。

参数	描述
livenessProbe	**livenessProbe** 会向 kubelet 报告 Pod 是否已死并需要重启。
readinessProbe	**readinessProbe** 报告容器集是健康还是不健康。当就绪度探测报告不健康的 pod 时，kubelet 会将 pod 标记为不接受流量。然后，该 pod 的端点标记为未就绪，这个状态会应用到 kube-proxy。在配置了负载均衡器的云平台中，kube-proxy 与云负载均衡器通信，不会将流量发送到该 pod 的节点。
startupProbe	**startupProbe** 为路由器 pod 提供最多 2 分钟的时间，以便 kubelet 开始发送路由器存活度和就绪度探测。这种初始化时间可以防止带有多个路由或端点的路由器会预先重启。

重要

timeout 配置选项是一个高级调优技术，可用于解决问题。但是，最终应该诊断这些问题，并可能为导致探测超时的所有问题打开支持问题单或 JIRA 问题。

以下示例演示了如何直接修补默认路由器部署来为存活度和就绪度探测设置 5 秒超时：

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":
{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5},"readinessProbe":
{"timeoutSeconds":5}}]}}}}'
```

验证

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
    Liveness:   http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
    Readiness:  http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1
#failure=3
```

10.8.3. 配置 HAProxy 重新加载间隔

当您更新路由或与路由关联的端点时，OpenShift Container Platform 路由器会更新 HAProxy 的配置。然后，HAProxy 重新加载更新后的配置以使这些更改生效。当 HAProxy 重新加载时，它会生成一个使用更新的配置来处理新连接的新进程。

HAProxy 保持旧进程正在运行，以处理现有连接，直到这些连接都关闭。当旧进程有长期连接时，这些进程可能会累积并消耗资源。

默认最小 HAProxy 重新加载间隔为 5 秒。您可以使用 **spec.tuningOptions.reloadInterval** 字段配置 Ingress Controller，以设置较长的重新载入间隔。

警告

为最低 HAProxy 重新加载间隔设置较大的值可能会导致观察路由及其端点更新产生延迟。要降低风险，请避免设置值超过更新可容忍的延迟。HAProxy 重新加载间隔的最大值为 120 秒。

流程

- 运行以下命令，将默认 Ingress Controller 的最小 HAProxy 重新加载间隔改为 15 秒：

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
```

10.9. 安装后 RHOSP 网络配置

您可在安装后在 Red Hat OpenStack Platform(RHOSP)集群中配置 OpenShift Container Platform 的一些方面。

10.9.1. 使用浮动 IP 地址配置应用程序访问

安装 OpenShift Container Platform 后，请配置 Red Hat OpenStack Platform (RHOSP) 以允许应用程序网络流量。

注意

如果您在 **install-config.yaml** 文件中为 **platform.openstack.apiFloatingIP** 和 **platform.openstack.ingressFloatingIP** 提供了值，或为 **inventory.yaml** playbook 中的 **os_api_fip** 和 **os_ingress_fip** 提供了值，在安装过程中不需要执行此步骤。已设置浮动 IP 地址。

先决条件

- 必须已安装 OpenShift Container Platform 集群
- 启用浮动 IP 地址，如 RHOSP 安装文档中的 OpenShift Container Platform 所述。

流程

在安装 OpenShift Container Platform 集群后，将浮动 IP 地址附加到入口端口：

1. 显示端口：

```
$ openstack port show <cluster_name>-<cluster_ID>-ingress-port
```

2. 将端口附加到 IP 地址：

```
$ openstack floating ip set --port <ingress_port_ID> <apps_FIP>
```

3. 在您的 DNS 文件中，为 ***apps.** 添加一条通配符 **A** 记录。

```
*.apps.<cluster_name>.<base_domain>  IN  A  <apps_FIP>
```

注意

如果您不控制 DNS 服务器，但希望为非生产用途启用应用程序访问，您可以将这些主机名添加到 **/etc/hosts**：

```
<apps_FIP> console-openshift-console.apps.<cluster name>.<base domain>
<apps_FIP> integrated-oauth-server-openshift-authentication.apps.<cluster name>.
<base domain>
<apps_FIP> oauth-openshift.apps.<cluster name>.<base domain>
<apps_FIP> prometheus-k8s-openshift-monitoring.apps.<cluster name>.<base
domain>
<apps_FIP> <app name>.apps.<cluster name>.<base domain>
```

10.9.2. 启用 OVS 硬件卸载

对于在 Red Hat OpenStack Platform(RHOSP)上运行的集群，您可以启用 Open vSwitch(OVS) 硬件卸载。

OVS 是一种多层虚拟交换机，能够启用大规模多服务器网络虚拟化。

先决条件

- 您在为单根输入/输出虚拟化(SR-IOV)配置的 RHOSP 上安装集群。
- 在集群中安装了 SR-IOV Network Operator。
- 您在集群中创建了两个 **hw-offload** 类型虚拟功能(VF)接口。

注意

应用程序层网关流在 OpenShift Container Platform 版本 4.10、4.11 和 4.12 中无法正常工作。另外，您无法卸载 OpenShift Container Platform 版本 4.13 的应用程序层网关流。

流程

1. 为集群中的两个 **hw-offload** 类型 VF 接口创建一个 **SriovNetworkNodePolicy** 策略：

第一个虚拟功能接口

```
apiVersion: sriovnetwork.openshift.io/v1
```

```
kind: SriovNetworkNodePolicy (1)
metadata:
  name: "hwoffload9"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: (2)
    - ens6
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload9"
```

1. 在此处插入 **SriovNetworkNodePolicy** 值。
2. 两个接口都必须包含物理功能(PF)名称。

第二个虚拟功能接口

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy (1)
metadata:
  name: "hwoffload10"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: (2)
    - ens5
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload10"
```

1. 在此处插入 **SriovNetworkNodePolicy** 值。
2. 两个接口都必须包含物理功能(PF)名称。

2. 为两个接口创建 **NetworkAttachmentDefinition** 资源：

第一接口的 NetworkAttachmentDefinition 资源

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload9
  name: hwoffload9
```

```
  namespace: default
spec:
    config: '{ "cniVersion":"0.3.1", "name":"hwoffload9","type":"host-device","device":"ens6"
    }'
```

第二个接口的 NetworkAttachmentDefinition 资源

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
  name: hwoffload10
  namespace: default
spec:
    config: '{ "cniVersion":"0.3.1", "name":"hwoffload10","type":"host-device","device":"ens5"
    }'
```

3. 使用通过 pod 创建的接口。例如：

使用两个 OVS 卸载接口的 Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-testpmd
  namespace: default
  annotations:
    irq-load-balancing.crio.io: disable
    cpu-quota.crio.io: disable
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload9
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
spec:
  restartPolicy: Never
  containers:
  - name: dpdk-testpmd
    image: quay.io/krister/centos8_nfv-container-dpdk-testpmd:latest
```

10.9.3. 附加 OVS 硬件卸载网络

您可以将 Open vSwitch (OVS) 硬件卸载网络添加到集群中。

先决条件

- 集群已安装并运行。
- 您已在 Red Hat OpenStack Platform (RHOSP) 上置备了 OVS 硬件卸载网络，用于您的集群。

流程

1. 从以下模板创建一个名为 **network.yaml** 的文件：

```
spec:
  additionalNetworks:
```

```
  - name: hwoffload1
    namespace: cnf
    rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "hwoffload1", "type": "host-
device","pciBusId": "0000:00:05.0", "ipam": {}}' 1
    type: Raw
```

其中：

pciBusId

指定连接到卸载网络的设备。如果没有它，您可以运行以下命令来查找这个值：

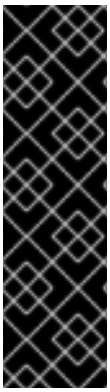
```
$ oc describe SriovNetworkNodeState -n openshift-sriov-network-operator
```

2. 在命令行中输入以下命令使用该文件来修补集群：

```
$ oc apply -f network.yaml
```

10.9.4. 启用与 RHOSP 上的 pod 的 IPv6 连接

要在具有不同节点上的额外网络的 pod 之间启用 IPv6 连接，请禁用服务器的 IPv6 端口的端口安全性。禁用端口安全性，从而需要为每个分配给 pod 的 IPv6 地址创建允许的地址对，并启用安全组上的流量。



重要

仅支持以下 IPv6 额外网络配置：

- SLAAC 和 host-device
- SLAAC 和 MACVLAN
- DHCP stateless 和 host-device
- DHCP stateless 和 MACVLAN

流程

- 在命令行中输入以下命令：

```
$ openstack port set --no-security-group --disable-port-security <compute_ipv6_port>
```

重要

这个命令从端口中删除安全组，并禁用端口安全性。流量限制完全从端口中删除。

其中：

<compute_ipv6_port>

指定计算服务器的 IPv6 端口。

10.9.5. 为 RHOSP 上的 pod 添加 IPv6 连接

在 pod 中启用 IPv6 连接后，使用 Container Network Interface (CNI) 配置添加与它们的连接。

流程

1. 要编辑 Cluster Network Operator (CNO)，请输入以下命令：

```
$ oc edit networks.operator.openshift.io cluster
```

2. 在 **spec** 字段中指定您的 CNI 配置。例如，以下配置使用带有 MACVLAN 的 192.168.1.0/24 地址模式：

```
...
spec:
  additionalNetworks:
  - name: ipv6
    namespace: ipv6 1
    rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "ipv6", "type": "macvlan", "master": "ens4"}'
2
    type: Raw
```

1 1 确保在同一命名空间中创建 pod。

2 当配置了更多网络或使用不同的内核驱动程序时，网络附加 **"master"** 字段中的接口可能与 **"ens4"** 不同。

注意

如果您使用有状态地址模式，请在 CNI 配置中包含 IP 地址管理 (IPAM)。

Multus 不支持 DHCPv6。

3. 保存您的更改，再退出文本编辑器以提交更改。

验证

- 在命令行中输入以下命令：

```
$ oc get network-attachment-definitions -A
```

输**出示例**

```
NAMESPACE      NAME          AGE
ipv6           ipv6          21h
```

现在，您可以创建具有辅助 IPv6 连接的 pod。

其他资源

- 配置额外网络附加

10.9.6. 在 RHOSP 上创建具有 IPv6 连接的 pod

为 pod 启用 IPv6 连接后，将其添加到其中，创建具有辅助 IPv6 连接的 pod。

流程

1. 定义使用 IPv6 命名空间和注解 **k8s.v1.cni.cncf.io/networks: <additional_network_name>** 的 pod，其中 **<additional_network_name** 是额外网络的名称。例如，作为 **Deployment** 对象的一部分：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
  namespace: ipv6
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - hello-openshift
  replicas: 2
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
      annotations:
        k8s.v1.cni.cncf.io/networks: ipv6
    spec:
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containers:
      - name: hello-openshift
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
        image: quay.io/openshift/origin-hello-openshift
        ports:
        - containerPort: 8080
```

2. 创建 pod。例如，在命令行中输入以下命令：

```
$ oc create -f <ipv6_enabled_resource>
```

其中：

<ipv6_enabled_resource>

指定包含资源定义的文件。

第 11 章 安装后存储配置

安装 OpenShift Container Platform 后，您可以按照自己的要求进一步扩展和自定义集群，包括存储配置。

11.1. 动态置备

通过动态置备，您可以按需创建存储卷，使集群管理员无需预置备存储。请参阅动态置备。

11.2. 推荐的可配置存储技术

下表总结了为给定的 OpenShift Container Platform 集群应用程序推荐的可配置存储技术。

表 11.1. 推荐的、可配置的存储技术

存储类型	Block	File	对象
ROX[1]	Yes[4]	Yes[4]	是
RWX[2]	否	是	是
Registry	可配置	可配置	推荐的
扩展的 registry	无法配置	可配置	推荐的
Metrics[3]	推荐的	Configurable[5]	无法配置
Elasticsearch Logging	推荐的	Configurable[6]	不支持[6]
Loki Logging	无法配置	无法配置	推荐的

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

[3] **Prometheus 是用于指标数据的底层技术。**

[4] **这不适用于物理磁盘、虚拟机物理磁盘、VMDK、NFS 回送、AWS EBS 和 Azure 磁盘。**

[5] **对于指标数据，使用 ReadWriteMany (RWX) 访问模式的文件存储是不可靠的。如果使用文件存储，请不要在配置用于指标数据的持久性卷声明（PVC）上配置 RWX 访问模式。**

[6] **用于日志记录，请参阅为日志存储配置持久性存储中的推荐存储解决方案。使用 NFS 存储作为持久性卷或通过 NAS （如 Gluster）可能会破坏数据。因此，OpenShift Container Platform Logging 中的 Elasticsearch 存储和 LokiStack 日志存储不支持 NFS。您必须为每个日志存储使用一个持久性卷类型。**

[7] **对象存储不会通过 OpenShift Container Platform 的 PV 或 PVC 使用。应用程序必须与对象存储 REST API 集成。**

存储类型	Block	File	对象
Apps	推荐的	推荐的	Not configurable[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

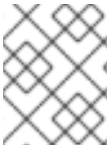
[3] Prometheus 是用于指标数据的底层技术。

[4] 这不适用于物理磁盘、虚拟机物理磁盘、VMDK、NFS 回送、AWS EBS 和 Azure 磁盘。

[5] 对于指标数据，使用 **ReadWriteMany** (RWX) 访问模式的文件存储是不可靠的。如果使用文件存储，请不要在配置用于指标数据的持久性卷声明（PVC）上配置 RWX 访问模式。

[6] 用于日志记录，请参阅为日志存储配置持久性存储中的推荐存储解决方案。使用 NFS 存储作为持久性卷或通过 NAS （如 Gluster）可能会破坏数据。因此，OpenShift Container Platform Logging 中的 Elasticsearch 存储和 LokiStack 日志存储不支持 NFS。您必须为每个日志存储使用一个持久性卷类型。

[7] 对象存储不会通过 OpenShift Container Platform 的 PV 或 PVC 使用。应用程序必须与对象存储 REST API 集成。



注意

扩展的容器镜像仓库（registry）是一个 OpenShift 镜像 registry，它有两个或更多个 pod 运行副本。

11.2.1. 特定应用程序存储建议

重要

测试显示在 Red Hat Enterprise Linux (RHEL)中使用 NFS 服务器作为核心服务的存储后端的问题。这包括 OpenShift Container Registry 和 Quay，Prometheus 用于监控存储，以及 Elasticsearch 用于日志存储。因此，不建议使用 RHEL NFS 作为 PV 后端用于核心服务。

市场中的其他 NFS 实现可能没有这些问题。如需了解更多与此问题相关的信息，请联络相关的 NFS 厂商。

11.2.1.1. Registry

在非扩展的/高可用性 (HA) OpenShift 镜像 registry 集群部署中：

- 存储技术不需要支持 RWX 访问模式。
- 存储技术必须保证读写一致性。
- 首选存储技术是对象存储，然后是块存储。
- 对于应用于生产环境工作负载的 OpenShift 镜像 Registry 集群部署，我们不推荐使用文件存储。

11.2.1.2. 扩展的 registry

在扩展的/HA OpenShift 镜像 registry 集群部署中：

- 存储技术必须支持 RWX 访问模式。
- 存储技术必须保证读写一致性。
- 首选存储技术是对象存储。
- 支持 Red Hat OpenShift Data Foundation (ODF), Amazon Simple Storage Service (Amazon S3), Google Cloud Storage (GCS), Microsoft Azure Blob Storage, 和 OpenStack Swift。
- 对象存储应该兼容 S3 或 Swift。
- 对于非云平台，如 vSphere 和裸机安装，唯一可配置的技术是文件存储。
- 块存储是不可配置的。
- 支持将网络文件系统(NFS)存储与 OpenShift Container Platform 搭配使用。但是，将 NFS 存储与扩展 registry 搭配使用可能会导致已知问题。如需更多信息，请参阅红帽知识库解决方案，是否在生产阶段中对 OpenShift 集群内部组件支持 NFS？

11.2.1.3. 指标

在 OpenShift Container Platform 托管的 metrics 集群部署中：

- 首选存储技术是块存储。
- 对象存储是不可配置的。

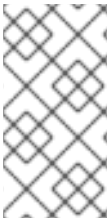
重要

在带有生产环境负载的托管 metrics 集群部署中不推荐使用文件存储。

11.2.1.4. 日志记录

在 OpenShift Container Platform 托管的日志集群部署中：

- Loki Operator：
  - 首选存储技术是 S3 兼容对象存储。
  - 块存储是不可配置的。
- OpenShift Elasticsearch Operator:
  - 首选存储技术是块存储。
  - 不支持对象存储。



注意

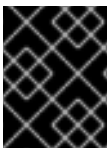
自日志记录版本 5.4.3 起，OpenShift Elasticsearch Operator 已被弃用，计划在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。您可以使用 Loki Operator 作为 OpenShift Elasticsearch Operator 的替代方案来管理默认日志存储。

11.2.1.5. 应用程序

应用程序的用例会根据不同应用程序而不同，如下例所示：

- 支持动态 PV 部署的存储技术的挂载时间延迟较低，且不与节点绑定来支持一个健康的集群。
- 应用程序开发人员需要了解应用程序对存储的要求，以及如何与所需的存储一起工作以确保应用程序扩展或者与存储层交互时不会出现问题。

11.2.2. 其他特定的应用程序存储建议



重要

不建议在 **Write** 密集型工作负载（如 **etcd** ）中使用 RAID 配置。如果您使用 RAID 配置运行 **etcd**，您可能会遇到工作负载性能问题的风险。

- Red Hat OpenStack Platform（RHOSP）Cinder: RHOSP Cinder 倾向于在 ROX 访问模式用例中使用。
- 数据库：数据库（RDBMS 、 nosql DBs 等等）倾向于使用专用块存储来获得最好的性能。
- etcd 数据库必须具有足够的存储和适当的性能容量才能启用大型集群。有关监控和基准测试工具的信息，以建立基本存储和高性能环境，请参阅 *推荐 etcd 实践*。

11.3. 部署 RED HAT OPENSHIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation 是 OpenShift Container Platform 支持的文件、块和对象存储的持久性存储供应商，可以在内部或混合云环境中使用。作为红帽存储解决方案，Red Hat OpenShift Data Foundation 与 OpenShift Container Platform 完全集成，用于部署、管理和监控。如需更多信息，请参阅 Red Hat OpenShift Data Foundation 文档 。

重要

虚拟化的 Red Hat Hyperconverged Infrastructure(RHHI)上的 OpenShift Data Foundation 不被支持。它使用超融合节点来托管 OpenShift Container Platform 安装的虚拟机。有关支持的平台的更多信息，请参阅 Red Hat OpenShift Data Foundation 支持性和互操作性指南。

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
新的、已知的问题、显著的程序错误修复以及技术预览	OpenShift Data Foundation 4.12 发行注记

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
支持的工作负载、布局、硬件和软件要求、调整和扩展建议	规划 OpenShift Data Foundation 4.12 部署
部署 OpenShift Data Foundation 以使用外部 Red Hat Ceph Storage 集群的说明	以外部模式部署 OpenShift Data Foundation 4.12
有关使用裸机基础架构上的本地存储部署 OpenShift Data Foundation 的说明	使用裸机基础架构部署 OpenShift Data Foundation 4.12
有关在 Red Hat OpenShift Container Platform VMware vSphere 集群上部署 OpenShift Data Foundation 的说明	在 VMware vSphere 上部署 OpenShift Data Foundation 4.12
有关使用 Amazon Web Services 进行本地或云存储部署 OpenShift Data Foundation 的说明	使用 Amazon Web Services 部署 OpenShift Data Foundation 4.12
在现有 Red Hat OpenShift Container Platform Google Cloud 集群上部署和管理 OpenShift Data Foundation 的说明	使用 Google Cloud 部署和管理 OpenShift Data Foundation 4.12
在现有 Red Hat OpenShift Container Platform Azure 集群上部署和管理 OpenShift Data Foundation 的说明	使用 Microsoft Azure 部署和管理 OpenShift Data Foundation 4.12
有关部署 OpenShift Data Foundation 在 IBM Power® 基础架构上使用本地存储的说明	在 IBM Power® 上部署 OpenShift Data Foundation。
有关部署 OpenShift Data Foundation 在 IBM Z® 基础架构上使用本地存储的说明	在 IBM Z® 基础架构上部署 OpenShift Data Foundation
将存储分配给 Red Hat OpenShift Data Foundation 中的核心服务和托管应用程序，包括快照和克隆	管理并分配资源
使用多云对象网关（NooBaa）在混合云或多云环境中管理存储资源	管理混合和多云资源
为 Red Hat OpenShift Data Foundation 安全替换存储设备	替换设备
安全替换 Red Hat OpenShift Data Foundation 集群中的节点	替换节点
在 Red Hat OpenShift Data Foundation 中扩展操作	扩展存储
监控 Red Hat OpenShift Data Foundation 4.12 集群	监控 Red Hat OpenShift Data Foundation 4.12

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
解决操作过程中遇到的问题	OpenShift Data Foundation 4.12 故障排除
将 OpenShift Container Platform 集群从版本 3 迁移到版本 4	Migration（迁移）

第 12 章 准备供用户使用

安装 OpenShift Container Platform 后，您可以按照您的要求进一步扩展和自定义集群，包括为用户准备步骤。

12.1. 了解身份提供程序配置

OpenShift Container Platform control plane 包含内置的 OAuth 服务器。开发人员和管理员获取 OAuth 访问令牌，以完成自身的 API 身份验证。

作为管理员，您可以在安装集群后通过配置 OAuth 来指定身份提供程序。

12.1.1. 关于 OpenShift Container Platform 中的身份提供程序

默认情况下，集群中只有 **kubeadmin** 用户。要指定身份提供程序，您必须创建一个自定义资源（CR）来描述该身份提供程序并把它添加到集群中。

注意

OpenShift Container Platform 用户名不能包括 /、: 和 %。

12.1.2. 支持的身份提供程序

您可以配置以下类型的身份提供程序：

用户身份提供程序	描述
htpasswd	配置 **htpasswd** 身份提供程序，针对使用 **htpasswd** 生成的文件验证用户名和密码。
Keystone	配置 **keystone** 身份提供程序，将 OpenShift Container Platform 集群与 Keystone 集成以启用共享身份验证，用配置的 OpenStack Keystone v3 服务器将用户存储到内部数据库中。
LDAP	配置 **ldap** 身份提供程序，使用简单绑定身份验证来针对 LDAPv3 服务器验证用户名和密码。
基本身份验证（Basic authentication）	配置 **basic-authentication** 身份提供程序，以便用户使用针对远程身份提供程序验证的凭证来登录 OpenShift Container Platform。基本身份验证是一种通用后端集成机制。
请求标头（Request header）	配置 **request-header** 身份提供程序，标识请求标头值中的用户，例如 **X-Remote-User**。它通常与设定请求标头值的身份验证代理一起使用。
Github 或 GitHub Enterprise	配置 **github** 身份提供程序，针对 GitHub 或 GitHub Enterprise 的 OAuth 身份验证服务器验证用户名和密码。
GitLab	配置 **gitlab** 身份提供程序，使用 GitLab.com 或任何其他 GitLab 实例作为身份提供程序。

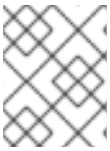
用户身份提供程序	描述
Google	配置 **google** 身份提供程序，使用 Google 的 OpenID Connect 集成。
OpenID Connect	配置 **oidc** 身份提供程序，使用授权代码流与 OpenID Connect 身份提供程序集成。

定义了身份提供程序后，您可以使用 RBAC 定义并应用权限。

12.1.3. 身份提供程序参数

以下是所有身份提供程序通用的参数：

参数	描述
name	此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
mappingMethod	定义在用户登录时如何将新身份映射到用户。输入以下值之一： **claim** 默认值。使用身份的首选用户名置备用户。如果具有该用户名的用户已映射到另一身份，则失败。 **lookup** 查找现有的身份、用户身份映射和用户，但不自动置备用户或身份。这允许集群管理员手动或使用外部流程设置身份和用户。使用此方法需要手动置备用户。 **add** 使用身份的首选用户名置备用户。如果已存在具有该用户名的用户，此身份将映射到现有用户，添加到该用户的现有身份映射中。如果配置了多个身份提供程序并且它们标识同一组用户并映射到相同的用户名，则需要进行此操作。



注意

在添加或更改身份提供程序时，您可以通过把 **mappingMethod** 参数设置为 **add**，将新提供程序中的身份映射到现有的用户。

12.1.4. 身份提供程序 CR 示例

以下自定义资源 (CR) 显示用来配置身份提供程序的参数和默认值。本例使用 htpasswd 身份提供程序。

身份提供程序 CR 示例

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_identity_provider 1
    mappingMethod: claim 2
    type: HTPasswd
```

```
  htpasswd:
    fileData:
      name: htpass-secret 3
```

1 此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。

2 控制如何在此提供程序的身份和 **User** 对象之间建立映射。

3 包含使用 **htpasswd** 生成的文件的现有 secret。

12.2. 使用 RBAC 定义和应用权限

理解并应用基于角色的访问控制。

12.2.1. RBAC 概述

基于角色的访问控制 (RBAC) 对象决定是否允许用户在项目内执行给定的操作。

集群管理员可以使用集群角色和绑定来控制谁对 OpenShift Container Platform 平台本身和所有项目具有各种访问权限等级。

开发人员可以使用本地角色和绑定来控制谁有权访问他们的项目。请注意，授权是与身份验证分开的一个步骤，身份验证更在于确定执行操作的人的身份。

授权通过使用以下几项来管理：

授权对象	描述
规则	一组对象上允许的操作集合。例如，用户或服务帐户能否**创建（create）** Pod。
角色	规则的集合。可以将用户和组关联或绑定到多个角色。
绑定	用户和/组与角色之间的关联。

控制授权的 RBAC 角色和绑定有两个级别：

RBAC 级别	描述
集群 RBAC	对所有项目均适用的角色和绑定。*集群角色*存在于集群范围，*集群角色绑定*只能引用集群角色。
本地 RBAC	作用于特定项目的角色和绑定。虽然*本地角色*只存在于单个项目中，但本地角色绑定可以*同时*引用集群和本地角色。

集群角色绑定是存在于集群级别的绑定。角色绑定存在于项目级别。集群角色 *view* 必须使用本地角色绑定来绑定到用户，以便该用户能够查看项目。只有集群角色不提供特定情形所需的权限集合时才应创建本地角色。

这种双级分级结构允许通过集群角色在多个项目间重复使用，同时允许通过本地角色在个别项目中自定义。

在评估过程中，同时使用集群角色绑定和本地角色绑定。例如：

1. 选中集群范围的“allow”规则。
2. 选中本地绑定的“allow”规则。
3. 默认为拒绝。

12.2.1.1. 默认集群角色

OpenShift Container Platform 包括了一组默认的集群角色，您可以在集群范围或本地将它们绑定到用户和组。

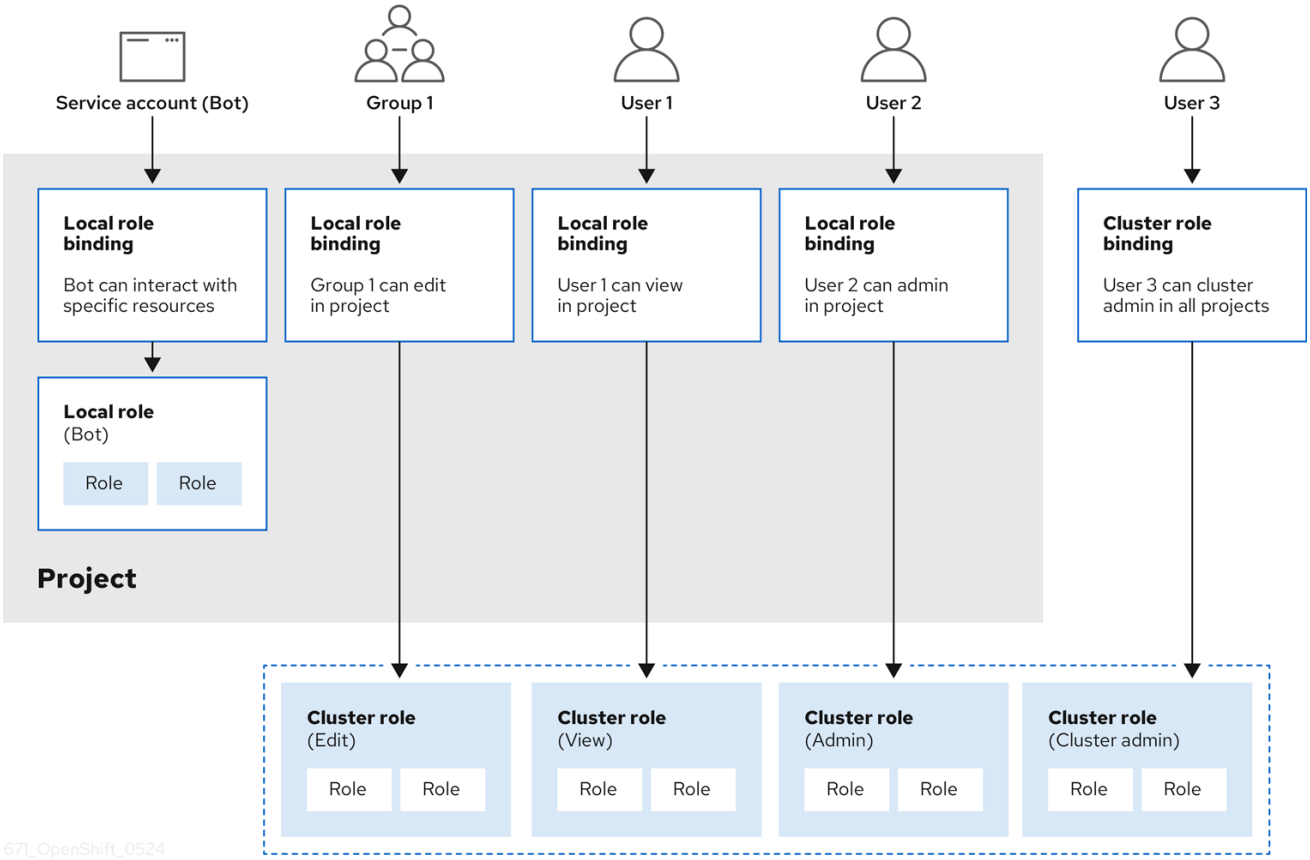
重要

不建议手动修改默认集群角色。对这些系统角色的修改可能会阻止集群正常工作。

默认集群角色	描述
admin	项目管理者。如果在本地绑定中使用，则 **admin** 有权查看项目中的任何资源，并且修改项目中除配额外的任何资源。
basic-user	此用户可以获取有关项目和用户的基本信息。
cluster-admin	此超级用户可以在任意项目中执行任何操作。当使用本地绑定来绑定一个用户时，这些用户可以完全控制项目中每一资源的配额和所有操作。
cluster-status	此用户可以获取基本的集群状态信息。
cluster-reader	用户可以获取或查看大多数对象，但不能修改它们。
edit	此用户可以修改项目中大多数对象，但无权查看或修改角色或绑定。
self-provisioner	此用户可以创建自己的项目。
view	此用户无法进行任何修改，但可以查看项目中的大多数对象。不能查看或修改角色或绑定。

请注意本地和集群绑定之间的区别。例如，如果使用本地角色绑定将 **cluster-admin** 角色绑定到一个用户，这可能看似该用户具有了集群管理员的特权。事实并非如此。将 **cluster-admin** 绑定到项目里的某一用户，仅会将该项目的超级管理员特权授予这一用户。该用户具有集群角色 **admin** 的权限，以及该项目的一些额外权限，例如能够编辑项目的速率限制。通过 web 控制台 UI 操作时此绑定可能会令人混淆，因为它不会列出绑定到真正集群管理员的集群角色绑定。然而，它会列出可用来本地绑定 **cluster-admin** 的本地角色绑定。

下方展示了集群角色、本地角色、集群角色绑定、本地角色绑定、用户、组和服务帐户之间的关系。



警告

当它们被应用到一个角色时，**get pods/exec**, **get pods/***, 和 **get *** 规则会授予执行权限。应用最小特权的原则，仅分配用户和代理所需的最小 RBAC 权限。如需更多信息，请参阅 RBAC 规则允许执行权限。

12.2.1.2. 评估授权

OpenShift Container Platform 使用以下几项来评估授权：

身份

用户名以及用户所属组的列表。

操作

您执行的操作。在大多数情况下，这由以下几项组成：

- **项目**：您访问的项目。项目是一种附带额外注解的 Kubernetes 命名空间，使一个社区的用户可以在与其他社区隔离的前提下组织和管理其内容。
- **操作动词**：操作本身：**get**、**list**、**create**、**update**、**delete**、**deletecollection** 或 **watch**。
- **资源名称**：您访问的 API 端点。

绑定

绑定的完整列表，用户或组与角色之间的关联。

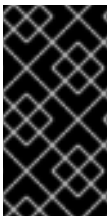
OpenShift Container Platform 通过以下几个步骤评估授权：

1. 使用身份和项目范围操作来查找应用到用户或所属组的所有绑定。
2. 使用绑定来查找应用的所有角色。
3. 使用角色来查找应用的所有规则。
4. 针对每一规则检查操作，以查找匹配项。
5. 如果未找到匹配的规则，则默认拒绝该操作。

提示

请记住，用户和组可以同时关联或绑定到多个角色。

项目管理员可以使用 CLI 查看本地角色和绑定信息，包括与每个角色关联的操作动词和资源的一览表。



重要

通过本地绑定来绑定到项目管理员的集群角色会限制在一个项目内。不会像授权给 **cluster-admin** 或 **system:admin** 的集群角色那样在集群范围绑定。

集群角色是在集群级别定义的角色，但可在集群级别或项目级别进行绑定。

12.2.1.2.1. 集群角色聚合

默认的 admin、edit、view 和 cluster-reader 集群角色支持集群角色聚合，其中每个角色的集群规则可在创建了新规则时动态更新。只有通过创建自定义资源扩展 Kubernetes API 时，此功能才有意义。

12.2.2. 项目和命名空间

Kubernetes *命名空间*提供设定集群中资源范围的机制。Kubernetes 文档中提供有关命名空间的更多信息。

命名空间为以下对象提供唯一范围：

- 指定名称的资源，以避免基本命名冲突。
- 委派给可信用户的管理授权。
- 限制社区资源消耗的能力。

系统中的大多数对象都通过命名空间来设定范围，但一些对象不在此列且没有命名空间，如节点和用户。

*项目*是附带额外注解的 Kubernetes 命名空间，是管理常规用户资源访问权限的集中载体。通过项目，一个社区的用户可以在与其他社区隔离的前提下组织和管理其内容。用户必须由管理员授予对项目的访问权限；或者，如果用户有权创建项目，则自动具有自己创建项目的访问权限。

项目可以有单独的 **name**、**displayName** 和 **description**。

- 其中必备的 **name** 是项目的唯一标识符，在使用 CLI 工具或 API 时最常见。名称长度最多为 63 个字符。

- 可选的 **displayName** 是项目在 web 控制台中的显示形式（默认为 **name**）。
- 可选的 **description** 可以为项目提供更加详细的描述，也可显示在 web 控制台中。

每个项目限制了自己的一组：

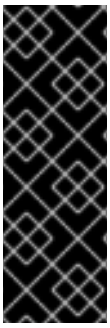
对象	描述
对象（object）	Pod、服务和复制控制器等。
策略（policy）	用户能否对对象执行操作的规则。
约束（constraint）	对各种对象进行限制的配额。
服务帐户	服务帐户自动使用项目中对象的指定访问权限进行操作。

集群管理员可以创建项目，并可将项目的管理权限委派给用户社区的任何成员。集群管理员也可以允许开发人员创建自己的项目。

开发人员和管理员可以使用 CLI 或 Web 控制台与项目交互。

12.2.3. 默认项目

OpenShift Container Platform 附带若干默认项目，名称以 **openshift--** 开头的项目对用户而言最为重要。这些项目托管作为 Pod 运行的主要组件和其他基础架构组件。在这些命名空间中创建的带有关键 (critical) Pod 注解的 Pod 是很重要的，它们可以保证被 kubelet 准入。在这些命名空间中为主要组件创建的 Pod 已标记为“critical”。



重要

不要在默认项目中运行工作负载或共享对默认项目的访问权限。为运行核心集群组件保留默认项目。

以下默认项目被视为具有高度特权：**default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**，其他系统创建的项目的标签 **openshift.io/run-level** 被设置为 **0** 或 **1**。依赖于准入插件（如 pod 安全准入、安全性上下文约束、集群资源配额和镜像引用解析）的功能无法在高特权项目中工作。

12.2.4. 查看集群角色和绑定

通过 **oc describe** 命令，可以使用 **oc** CLI 来查看集群角色和绑定。

先决条件

- 安装 **oc** CLI。
- 获取查看集群角色和绑定的权限。

在集群范围内绑定了 **cluster-admin** 默认集群角色的用户可以对任何资源执行任何操作，包括查看集群角色和绑定。

流程

1. 查看集群角色及其关联的规则集：

```
$ oc describe clusterrole.rbac
```

输出示例

```
Name:         admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names  Verbs
  ---------                                  -----------------  --------------  -----
  .packages.apps.redhat.com                  []                 []              [* create update
patch delete get list watch]
  imagestreams                               []                 []              [create delete
deletecollection get list patch update watch create get list watch]
  imagestreams.image.openshift.io            []                 []              [create delete
deletecollection get list patch update watch create get list watch]
  secrets                                    []                 []              [create delete deletecollection
get list patch update watch get list watch create delete deletecollection patch update]
  buildconfigs/webhooks                      []                 []              [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs                               []                 []              [create delete
deletecollection get list patch update watch get list watch]
  buildlogs                                  []                 []              [create delete deletecollection
get list patch update watch get list watch]
  deploymentconfigs/scale                    []                 []              [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs                          []                 []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreamimages                          []                 []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreammappings                        []                 []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreamtags                            []                 []              [create delete
deletecollection get list patch update watch get list watch]
  processedtemplates                         []                 []              [create delete
deletecollection get list patch update watch get list watch]
  routes                                     []                 []              [create delete deletecollection
get list patch update watch get list watch]
  templateconfigs                            []                 []              [create delete
deletecollection get list patch update watch get list watch]
  templateinstances                          []                 []              [create delete
deletecollection get list patch update watch get list watch]
  templates                                  []                 []              [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs.apps.openshift.io/scale  []                 []              [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs.apps.openshift.io        []                 []              [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs.build.openshift.io/webhooks   []                 []              [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs.build.openshift.io            []                 []              [create delete
deletecollection get list patch update watch get list watch]
```

```
  buildlogs.build.openshift.io                    []                []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreamimages.image.openshift.io            []                []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreammappings.image.openshift.io          []                []              [create delete
deletecollection get list patch update watch get list watch]
  imagestreamtags.image.openshift.io              []                []              [create delete
deletecollection get list patch update watch get list watch]
  routes.route.openshift.io                       []                []              [create delete
deletecollection get list patch update watch get list watch]
  processedtemplates.template.openshift.io        []                []              [create delete
deletecollection get list patch update watch get list watch]
  templateconfigs.template.openshift.io           []                []              [create delete
deletecollection get list patch update watch get list watch]
  templateinstances.template.openshift.io         []                []              [create delete
deletecollection get list patch update watch get list watch]
  templates.template.openshift.io                 []                []              [create delete
deletecollection get list patch update watch get list watch]
  serviceaccounts                                 []                []              [create delete
deletecollection get list patch update watch impersonate create delete deletecollection patch
update get list watch]
  imagestreams/secrets                            []                []              [create delete
deletecollection get list patch update watch]
  rolebindings                                    []                []              [create delete
deletecollection get list patch update watch]
  roles                                           []                []              [create delete deletecollection
get list patch update watch]
  rolebindings.authorization.openshift.io         []                []              [create delete
deletecollection get list patch update watch]
  roles.authorization.openshift.io                []                []              [create delete
deletecollection get list patch update watch]
  imagestreams.image.openshift.io/secrets         []                []              [create delete
deletecollection get list patch update watch]
  rolebindings.rbac.authorization.k8s.io          []                []              [create delete
deletecollection get list patch update watch]
  roles.rbac.authorization.k8s.io                 []                []              [create delete
deletecollection get list patch update watch]
  networkpolicies.extensions                      []                []              [create delete
deletecollection patch update create delete deletecollection get list patch update watch get
list watch]
  networkpolicies.networking.k8s.io               []                []              [create delete
deletecollection patch update create delete deletecollection get list patch update watch get
list watch]
  configmaps                                      []                []              [create delete
deletecollection patch update get list watch]
  endpoints                                       []                []              [create delete
deletecollection patch update get list watch]
  persistentvolumeclaims                          []                []              [create delete
deletecollection patch update get list watch]
  pods                                            []                []              [create delete deletecollection
patch update get list watch]
  replicationcontrollers/scale                    []                []              [create delete
deletecollection patch update get list watch]
  replicationcontrollers                          []                []              [create delete
deletecollection patch update get list watch]
  services                                        []                []              [create delete deletecollection
```

```
patch update get list watch]
  daemonsets.apps                                 []              []              [create delete
deletecollection patch update get list watch]
  deployments.apps/scale                          []              []              [create delete
deletecollection patch update get list watch]
  deployments.apps                                []              []              [create delete
deletecollection patch update get list watch]
  replicasets.apps/scale                          []              []              [create delete
deletecollection patch update get list watch]
  replicasets.apps                                []              []              [create delete
deletecollection patch update get list watch]
  statefulsets.apps/scale                         []              []              [create delete
deletecollection patch update get list watch]
  statefulsets.apps                               []              []              [create delete
deletecollection patch update get list watch]
  horizontalpodautoscalers.autoscaling            []              []              [create delete
deletecollection patch update get list watch]
  cronjobs.batch                                  []              []              [create delete
deletecollection patch update get list watch]
  jobs.batch                                      []              []              [create delete
deletecollection patch update get list watch]
  daemonsets.extensions                           []              []              [create delete
deletecollection patch update get list watch]
  deployments.extensions/scale                    []              []              [create delete
deletecollection patch update get list watch]
  deployments.extensions                          []              []              [create delete
deletecollection patch update get list watch]
  ingresses.extensions                            []              []              [create delete
deletecollection patch update get list watch]
  replicasets.extensions/scale                    []              []              [create delete
deletecollection patch update get list watch]
  replicasets.extensions                          []              []              [create delete
deletecollection patch update get list watch]
  replicationcontrollers.extensions/scale         []              []              [create delete
deletecollection patch update get list watch]
  poddisruptionbudgets.policy                     []              []              [create delete
deletecollection patch update get list watch]
  deployments.apps/rollback                       []              []              [create delete
deletecollection patch update]
  deployments.extensions/rollback                 []              []              [create delete
deletecollection patch update]
  catalogsources.operators.coreos.com             []              []              [create update
patch delete get list watch]
  clusterserviceversions.operators.coreos.com     []              []              [create update
patch delete get list watch]
  installplans.operators.coreos.com               []              []              [create update
patch delete get list watch]
  packagemanifests.operators.coreos.com           []              []              [create update
patch delete get list watch]
  subscriptions.operators.coreos.com              []              []              [create update
patch delete get list watch]
  buildconfigs/instantiate                        []              []              [create]
  buildconfigs/instantiatebinary                  []              []              [create]
  builds/clone                                    []              []              [create]
  deploymentconfigrollbacks                       []              []              [create]
  deploymentconfigs/instantiate                   []              []              [create]
```

```
  deploymentconfigs/rollback                                  []                []              [create]
  imagestreamimports                                          []                []              [create]
  localresourceaccessreviews                                  []                []              [create]
  localsubjectaccessreviews                                   []                []              [create]
  podsecuritypolicyreviews                                    []                []              [create]
  podsecuritypolicyselfsubjectreviews                         []                []              [create]
  podsecuritypolicysubjectreviews                             []                []              [create]
  resourceaccessreviews                                       []                []              [create]
  routes/custom-host                                          []                []              [create]
  subjectaccessreviews                                        []                []              [create]
  subjectrulesreviews                                         []                []              [create]
  deploymentconfigrollbacks.apps.openshift.io                 []                []              [create]
  deploymentconfigs.apps.openshift.io/instantiate             []                []              [create]
  deploymentconfigs.apps.openshift.io/rollback                []                []              [create]
  localsubjectaccessreviews.authorization.k8s.io              []                []              [create]
  localresourceaccessreviews.authorization.openshift.io       []                []              [create]
  localsubjectaccessreviews.authorization.openshift.io        []                []              [create]
  resourceaccessreviews.authorization.openshift.io            []                []              [create]
  subjectaccessreviews.authorization.openshift.io             []                []              [create]
  subjectrulesreviews.authorization.openshift.io              []                []              [create]
  buildconfigs.build.openshift.io/instantiate                 []                []              [create]
  buildconfigs.build.openshift.io/instantiatebinary           []                []              [create]
  builds.build.openshift.io/clone                             []                []              [create]
  imagestreamimports.image.openshift.io                       []                []              [create]
  routes.route.openshift.io/custom-host                       []                []              [create]
  podsecuritypolicyreviews.security.openshift.io              []                []              [create]
  podsecuritypolicyselfsubjectreviews.security.openshift.io   []                []              [create]
  podsecuritypolicysubjectreviews.security.openshift.io       []                []              [create]
  jenkins.build.openshift.io                                  []                []              [edit view view admin
edit view]
  builds                                                      []                []              [get create delete
deletecollection get list patch update watch get list watch]
  builds.build.openshift.io                                   []                []              [get create delete
deletecollection get list patch update watch get list watch]
  projects                                                    []                []              [get delete get delete get patch
update]
  projects.project.openshift.io                               []                []              [get delete get delete
get patch update]
  namespaces                                                  []                []              [get get list watch]
  pods/attach                                                 []                []              [get list watch create delete
deletecollection patch update]
  pods/exec                                                   []                []              [get list watch create delete
deletecollection patch update]
  pods/portforward                                            []                []              [get list watch create
delete deletecollection patch update]
  pods/proxy                                                  []                []              [get list watch create delete
deletecollection patch update]
  services/proxy                                              []                []              [get list watch create delete
deletecollection patch update]
  routes/status                                               []                []              [get list watch update]
  routes.route.openshift.io/status                            []                []              [get list watch update]
  appliedclusterresourcequotas                                []                []              [get list watch]
  bindings                                                    []                []              [get list watch]
  builds/log                                                  []                []              [get list watch]
  deploymentconfigs/log                                       []                []              [get list watch]
  deploymentconfigs/status                                    []                []              [get list watch]
```

```
  events                                          []                  []              [get list watch]
  imagestreams/status                             []                  []              [get list watch]
  limitranges                                     []                  []              [get list watch]
  namespaces/status                               []                  []              [get list watch]
  pods/log                                        []                  []              [get list watch]
  pods/status                                     []                  []              [get list watch]
  replicationcontrollers/status                   []                  []              [get list watch]
  resourcequotas/status                           []                  []              [get list watch]
  resourcequotas                                  []                  []              [get list watch]
  resourcequotausages                             []                  []              [get list watch]
  rolebindingrestrictions                         []                  []              [get list watch]
  deploymentconfigs.apps.openshift.io/log         []                  []              [get list watch]
  deploymentconfigs.apps.openshift.io/status      []                  []              [get list watch]
  controllerrevisions.apps                        []                  []              [get list watch]
  rolebindingrestrictions.authorization.openshift.io  []              []              [get list watch]
  builds.build.openshift.io/log                   []                  []              [get list watch]
  imagestreams.image.openshift.io/status          []                  []              [get list watch]
  appliedclusterresourcequotas.quota.openshift.io []                  []              [get list watch]
  imagestreams/layers                             []                  []              [get update get]
  imagestreams.image.openshift.io/layers          []                  []              [get update get]
  builds/details                                  []                  []              [update]
  builds.build.openshift.io/details               []                  []              [update]


Name:         basic-user
Labels:       <none>
Annotations:  openshift.io/description: A user that can get basic information about projects.
              rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
 Resources                                       Non-Resource URLs  Resource Names  Verbs
  ---------                                      -----------------  --------------  -----
  selfsubjectrulesreviews                        []                 []              [create]
  selfsubjectaccessreviews.authorization.k8s.io  []                 []              [create]
  selfsubjectrulesreviews.authorization.openshift.io  []             []              [create]
  clusterroles.rbac.authorization.k8s.io         []                 []              [get list watch]
  clusterroles                                   []                 []              [get list]
  clusterroles.authorization.openshift.io        []                 []              [get list]
  storageclasses.storage.k8s.io                  []                 []              [get list]
  users                                          []                 [~]             [get]
  users.user.openshift.io                        []                 [~]             [get]
  projects                                       []                 []              [list watch]
  projects.project.openshift.io                  []                 []              [list watch]
  projectrequests                                []                 []              [list]
  projectrequests.project.openshift.io           []                 []              [list]

Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
Resources  Non-Resource URLs  Resource Names  Verbs
---------  -----------------  --------------  -----
*.*        []                 []              [*]
           [*]                []              [*]

...
```

2. 查看当前的集群角色绑定集合，这显示绑定到不同角色的用户和组：

```
$ oc describe clusterrolebinding.rbac
```

输**出示例**

```
Name:         alertmanager-main
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  alertmanager-main
Subjects:
  Kind           Name              Namespace
  ----           ----              ---------
  ServiceAccount  alertmanager-main  openshift-monitoring


Name:         basic-users
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  basic-user
Subjects:
  Kind   Name                Namespace
  ----   ----                ---------
  Group  system:authenticated


Name:         cloud-credential-operator-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cloud-credential-operator-role
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  default  openshift-cloud-credential-operator


Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters


Name:         cluster-admins
```

```
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name                Namespace
  ----   ----                ---------
  Group  system:cluster-admins
  User   system:admin


Name:         cluster-api-manager-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cluster-api-manager-role
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  default  openshift-machine-api

...
```

12.2.5. 查看本地角色和绑定

使用 **oc describe** 命令通过 **oc** CLI 来查看本地角色和绑定。

先决条件

- 安装 **oc** CLI。
- 获取查看本地角色和绑定的权限：
  - 在集群范围内绑定了 **cluster-admin** 默认集群角色的用户可以对任何资源执行任何操作，包括查看本地角色和绑定。
  - 本地绑定了 **admin** 默认集群角色的用户可以查看并管理项目中的角色和绑定。

流程

1. 查看当前本地角色绑定集合，这显示绑定到当前项目的不同角色的用户和组：

```
$ oc describe rolebinding.rbac
```

2. 要查其他项目的本地角色绑定，请向命令中添加 **-n** 标志：

```
$ oc describe rolebinding.rbac -n joe-project
```

输**出示例**

```
Name:         admin
Labels:       <none>
```

```
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name       Namespace
  ----  ----       ---------
  User  kube:admin


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind           Name      Namespace
  ----           ----      ---------
  ServiceAccount  deployer  joe-project


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  builder  joe-project


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                          Namespace
  ----   ----                          ---------
  Group  system:serviceaccounts:joe-project
```

12.2.6. 向用户添加角色

可以使用 **oc adm** 管理员 CLI 管理角色和绑定。

将角色绑定或添加到用户或组可让用户或组具有该角色授予的访问权限。您可以使用 **oc adm policy** 命令向用户和组添加和移除角色。

您可以将任何默认集群角色绑定到项目中的本地用户或组。

流程

1. 向指定项目中的用户添加角色：

```
$ oc adm policy add-role-to-user <role> <user> -n <project>
```

例如，您可以运行以下命令，将 **admin** 角色添加到 **joe** 项目中的 **alice** 用户：

```
$ oc adm policy add-role-to-user admin alice -n joe
```

提示

您还可以应用以下 YAML 向用户添加角色：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: admin-0
  namespace: joe
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

2. 查看本地角色绑定，并在输出中验证添加情况：

```
$ oc describe rolebinding.rbac -n <project>
```

例如，查看 **joe** 项目的本地角色绑定：

```
$ oc describe rolebinding.rbac -n joe
```

输**出示例**

```
Name:         admin
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
```

```
  Kind  Name       Namespace
  ----  ----       ---------
  User  kube:admin


Name:         admin-0
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice  (1)


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind           Name      Namespace
  ----           ----      ---------
  ServiceAccount  deployer  joe


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  builder  joe


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
```

```
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:serviceaccounts:joe
```

1 **alice** 用户已添加到 **admins RoleBinding**。

12.2.7. 创建本地角色

您可以为项目创建本地角色，然后将其绑定到用户。

流程

1. 要为项目创建本地角色，请运行以下命令：

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
```

在此命令中，指定：

- **<name>**，本地角色的名称
- **<verb>**，以逗号分隔的、应用到角色的操作动词列表
- **<resource>**，角色应用到的资源
- **<project>**，项目名称

例如，要创建一个本地角色来允许用户查看 **blue** 项目中的 Pod，请运行以下命令：

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. 要将新角色绑定到用户，运行以下命令：

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
```

12.2.8. 创建集群角色

您可以创建集群角色。

流程

1. 要创建集群角色，请运行以下命令：

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

在此命令中，指定：

- **<name>**，本地角色的名称
- **<verb>**，以逗号分隔的、应用到角色的操作动词列表
- **<resource>**，角色应用到的资源

例如，要创建一个集群角色来允许用户查看 Pod，请运行以下命令：

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

12.2.9. 本地角色绑定命令

在使用以下操作为本地角色绑定管理用户或组的关联角色时，可以使用 **-n** 标志来指定项目。如果未指定，则使用当前项目。

您可以使用以下命令进行本地 RBAC 管理。

表 12.1. 本地角色绑定操作

命令	描述
$ oc adm policy who-can *<verb> <resource>*	指出哪些用户可以对某一资源执行某种操作。
$ oc adm policy add-role-to-user *<role> <username>*	将指定角色绑定到当前项目中的指定用户。
$ oc adm policy remove-role-from-user *<role> <username>*	从当前项目中的指定用户移除指定角色。
$ oc adm policy remove-user *<username>*	移除当前项目中的指定用户及其所有角色。
$ oc adm policy add-role-to-group *<role> <groupname>*	将给定角色绑定到当前项目中的指定组。
$ oc adm policy remove-role-from-group *<role> <groupname>*	从当前项目中的指定组移除给定角色。
$ oc adm policy remove-group *<groupname>*	移除当前项目中的指定组及其所有角色。

12.2.10. 集群角色绑定命令

您也可以使用以下操作管理集群角色绑定。因为集群角色绑定使用没有命名空间的资源，所以这些操作不使用 **-n** 标志。

表 12.2. 集群角色绑定操作

命令	描述
$ oc adm policy add-cluster-role-to-user *<role> <username>*	将给定角色绑定到集群中所有项目的指定用户。
$ oc adm policy remove-cluster-role-from-user *<role> <username>*	从集群中所有项目的指定用户移除给定角色。
$ oc adm policy add-cluster-role-to-group *<role> <groupname>*	将给定角色绑定到集群中所有项目的指定组。

命令	描述
$ oc adm policy remove-cluster-role-from-group *<role> <groupname>*	从集群中所有项目的指定组移除给定角色。

12.2.11. 创建集群管理员

需要具备 **cluster-admin** 角色才能在 OpenShift Container Platform 集群上执行管理员级别的任务，例如修改集群资源。

先决条件

- 您必须已创建了要定义为集群管理员的用户。

流程

- 将用户定义为集群管理员：

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user>
```

12.3. KUBEADMIN 用户

OpenShift Container Platform 在安装过程完成后会创建一个集群管理员 **kubeadmin**。

此用户自动具有 **cluster-admin** 角色，并视为集群的 root 用户。其密码是动态生成的，对 OpenShift Container Platform 环境中是唯一的。安装完成后，安装程序的输出中会包括这个密码。例如：

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

12.3.1. 移除 kubeadmin 用户

在定义了身份提供程序并创建新的 **cluster-admin** 用户后，您可以移除 **kubeadmin** 来提高集群安全性。

警告

如果在另一用户成为 **cluster-admin** 前按照这个步骤操作，则必须重新安装 OpenShift Container Platform。此命令无法撤销。

先决条件

- 必须至少配置了一个身份提供程序。

- 必须向用户添加了 **cluster-admin** 角色。
- 必须已经以管理员身份登录。

流程

- 移除 **kubeadmin** Secret：

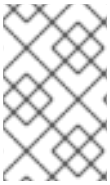
```
$ oc delete secrets kubeadmin -n kube-system
```

12.4. 镜像配置

了解并配置镜像 registry 设置。

12.4.1. 镜像控制器配置参数

Image.config.openshift.io/cluster 资源包含有关如何处理镜像的集群范围信息。规范且唯一有效的名称是 **cluster**。它的 **spec** 提供以下配置参数。



注意

参数，如 **DisableScheduledImport**, **MaxImagesBulkImportedPerRepository**, **MaxScheduledImportsPerMinute**, **ScheduledImageImportMinimumIntervalSeconds**, **InternalRegistryHostname** 不可配置。

参数	描述
allowedRegistriesForImport	限制普通用户可从中导入镜像的容器镜像 registry。将此列表设置为您信任包含有效镜像且希望应用程序能够从中导入的 registry。有权从 API 创建镜像或 **ImageStreamMappings** 的用户不受此策略的影响。通常只有集群管理员具有适当权限。 这个列表中的每个项包含由 registry 域名指定的 registry 的位置。 **domainname**：指定 registry 的域名。如果 registry 使用非标准的**80** 或 **443** 端口，则该端口也应包含在域名中。 **insecure**：不安全指示 registry 是否安全。默认情况下，如果未另行指定，registry 假定为安全。
additionalTrustedCA	对包含 **image stream import**、**pod image pull**、**openshift-image-registry pullthrough** 和构建期间应受信任的额外 CA 的配置映射的引用。 此配置映射的命名空间为 **openshift-config**。ConfigMap 的格式是使用 registry 主机名作为键，使用 PEM 编码证书作为值，用于每个要信任的额外 registry CA。
externalRegistryHostnames	提供默认外部镜像 registry 的主机名。只有在镜像 registry 对外公开时才应设置外部主机名。第一个值用于镜像流中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。

参数	描述
registrySources	包含用于决定容器运行时在访问构建和 pod 的镜像时应如何处理个别 registry 的配置。例如，是否允许不安全的访问。它不包含内部集群 registry 的配置。 **insecureRegistries**：无有效 TLS 证书或仅支持 HTTP 连接的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。 **blockedRegistries**：拒绝镜像拉取（pull）和推送（push）操作的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。允许所有其他 registry。 **allowedRegistries**：允许镜像拉取（pull）和推送（push）操作的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。阻止所有其他 registry。 **containerRuntimeSearchRegistries**：允许使用镜像短名称的镜像拉取（pull）和推送（push）操作的 registry。阻止所有其他 registry。 可以设置 **blockedRegistries** 或 **allowedRegistries**，但不能同时都被设置。

警告

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 **registry.redhat.io** 和 **quay.io** registry 和默认的 OpenShift 镜像 registry）都会被阻断。当使用参数时，为了避免 pod 失败，将所有 registry（包括 **registry.redhat.io** 和 **quay.io** registry）和 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。对于断开连接的集群，还应添加镜像的 registry。

image.config.openshift.io/cluster 资源的 **status** 项包括了从集群观察到的值。

参数	描述
internalRegistryHostname	由控制 **internalRegistryHostname** 的 Image Registry Operator 设置。它设置默认 OpenShift 镜像 registry 的主机名。该值必须采用 **hostname[:port]** 格式。为实现向后兼容，您仍可使用 **OPENSHIFT_DEFAULT_REGISTRY** 环境变量，但该设置会覆盖环境变量。
externalRegistryHostnames	由 Image Registry Operator 设置，在镜像 registry 通过外部公开时为它提供外部主机名。第一个值用于镜像流中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。

12.4.2. 配置镜像 registry 设置

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）来配置镜像 registry 设置。当对 registry 的更改应用到 **image.config.openshift.io/cluster** CR 时，Machine Config Operator (MCO) 执行以下顺序操作：

1. 对节点进行 cordon 操作
2. 通过重启 CRI-O 应用更改
3. 取消记录节点

注意

MCO 在检测到更改时不会重启节点。

流程

1. 编辑 **image.config.openshift.io/cluster** 自定义资源：

```
$ oc edit image.config.openshift.io/cluster
```

以下是 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: (2)
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: (3)
    name: myconfigmap
  registrySources: (4)
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

(1) **Image**：包含有关如何处理镜像的集群范围信息。规范且唯一有效的名称是 **cluster**。

2 **allowedRegistriesForImport**：限制普通用户可从中导入镜像的容器镜像 registry。将此列表设置为您信任包含有效镜像且希望应用程序能够从中导入的 registry。有权从 API 创建镜

3 **additionalTrustedCA**：引用包含镜像流导入、Pod 镜像拉取、**openshift-image-registry** pullthrough 和构建期间受信任的额外证书颁发机构（CA）的配置映射。此配置映射的命名空间为 **openshift-config**。ConfigMap 的格式是使用 registry 主机名作为键，使用 PEM 证书作为值，用于每个要信任的额外 registry CA。

4 **registrySources** ：包含用于决定容器运行时在访问构建和 pod 的镜像时是否允许或阻止个别 registry 的配置。可以设置 **allowedRegistries** 参数或 **blockedRegistries** 参数，但不能同时设置这两个参数。您还可以定义是否允许访问允许使用镜像短名称的不安全的 registry。本例使用 **allowedRegistries** 参数，该参数定义允许使用的 registry。不安全 registry **insecure.com** 也被允许。**registrySources** 参数不包含内部集群 registry 的配置。

注意

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 registry.redhat.io 和 quay.io registry 和默认的 OpenShift 镜像 registry）都会被阻断。如果使用参数，为了避免 pod 失败，您必须将 **registry.redhat.io** 和 **quay.io** registry 以及 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。不要将 **registry.redhat.io** 和 **quay.io** registry 添加到 **blockedRegistries** 列表中。

使用 **allowedRegistries**、**blockedRegistries** 或 **insecureRegistries** 参数时，您可以在 registry 中指定单独的存储库。例如： **reg1.io/myrepo/myapp:latest**。

应避免使用不安全的外部 registry，以减少可能的安全性风险。

2. 要检查是否应用了更改，请列出您的节点：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker                 65m   v1.28.5
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.28.5
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.28.5
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.28.5
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.28.5
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.28.5
```

如需有关允许、阻止和不安全的 registry 参数的更多信息，请参阅配置镜像 registry 设置。

12.4.3. 为镜像 registry 访问配置额外的信任存储

Image.config.openshift.io/cluster 自定资源可包含对配置映射的引用，该配置映射包含要在镜像 registry 访问期间被信任的额外证书颁发机构。

先决条件

- 证书颁发机构（CA）必须经过 PEM 编码。

流程

您可以在**openshift-config**命名空间中创建配置映射，并在 **image.config.openshift.io** 子定义资源中的 **AdditionalTrustedCA** 中使用其名称，以提供与外部 registry 联系时可以被信任的额外CA。

对于每个要信任的额外 registry CA，配置映射键是带有要信任此 CA 的端口的 registry 的主机名，而 PEM 证书内容是要信任的每个额外 registry CA。

镜像 registry CA 配置映射示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1 如果 registry 带有端口，如 **registry-with-port.example.com:5000**，**:** 需要被 **..** 替换。

您可以按照以下过程配置其他CA。

- 配置其他CA：

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

12.5. 了解镜像 REGISTRY 仓库镜像

通过设置容器 registry 存储库镜像，您可以执行以下任务：

- 配置 OpenShift Container Platform 集群，以便重定向从源镜像 registry 上的存储库拉取（pull）镜像的请求，并通过已镜像 (mirror) 的镜像 registry 上的存储库来解决该请求。

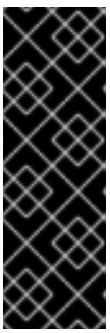
- 为每个目标存储库识别多个已镜像 (mirror)的存储库，以确保如果一个镜像停止运作，仍可使用其他镜像。

OpenShift Container Platform 中的存储库镜像包括以下属性：

- 镜像拉取（pull）可应对 registry 停机的问题。
- 在断开连接的环境中的集群可以从关键位置（如 quay.io）拉取镜像，并让公司防火墙后面的 registry 提供请求的镜像。
- 发出镜像拉取（pull）请求时尝试特定 registry 顺序，通常最后才会尝试持久性 registry。
- 您所输入的镜像信息会添加到 OpenShift Container Platform 集群中每个节点上的 **/etc/containers/registries.conf** 文件中。
- 当节点从源存储库中请求镜像时，它会依次尝试每个已镜像的存储库，直到找到所请求的内容。如果所有镜像均失败，集群则会尝试源存储库。如果成功，则镜像拉取至节点中。

可通过以下方式设置存储库镜像：

- 在 OpenShift Container Platform 安装中：
  通过拉取（pull） OpenShift Container Platform 所需的容器镜像，然后将这些镜像放至公司防火墙后，即可将 OpenShift Container Platform 安装到受限网络中的数据中心。
- 安装 OpenShift Container Platform 后：
  如果您没有在 OpenShift Container Platform 安装过程中配置镜像，您可以在安装后使用以下自定义资源 (CR) 对象之一进行配置：
  - **ImageDigestMirrorSet** (IDMS).此对象允许您使用摘要规格从镜像 registry 中拉取镜像。IDMS CR 可让您设置回退策略，在镜像拉取失败时继续尝试从源 registry 中拉取。
  - **ImageTagMirrorSet** (ITMS)。此对象允许您使用镜像标签从已镜像的 registry 中拉取镜像。ITMS CR 可让您设置回退策略，在镜像拉取失败时继续尝试从源 registry 中拉取。
  - **ImageContentSourcePolicy** (ICSP)。此对象允许您使用摘要规格从镜像 registry 中拉取镜像。如果镜像无法正常工作，ICSP CR 始终回退到源 registry。



重要

使用 **ImageContentSourcePolicy** (ICSP)对象配置存储库镜像是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。如果您有用于创建 **ImageContentSourcePolicy** 对象的 YAML 文件，您可以使用 **oc adm migrate icsp** 命令将这些文件转换为 **ImageDigestMirrorSet** YAML 文件。如需更多信息，请参阅以下部分"协调 ImageContentSourcePolicy (ICSP)文件以进行镜像 registry 存储库镜像"。

每个自定义资源对象都标识以下信息：

- 您希望镜像 (mirror) 的容器镜像存储库的源。
- 您希望为其提供从源存储库请求的内容的每个镜像存储库的单独条目。

对于新集群，您可以根据需要使用 IDMS、ITMS 和 ICSP CR 对象。但是，建议使用 IDMS 和 ITMS。

如果您升级了集群，则任何现有 ICSP 对象都会保持稳定，并且支持 IDMS 和 ICSP 对象。使用 ICSP 对象

的工作负载可以按预期工作。但是，如果要利用 IDMS CR 中引入的回退策略，您可以使用 **oc adm migrate icsp** 命令将当前工作负载迁移到 IDMS 对象，如后面的**镜像 registry 存储库镜像**部分所示。迁移到 IDMS 对象不需要重启集群。

注意

如果您的集群使用 **ImageDigestMirrorSet**、**ImageTagMirrorSet** 或 **ImageContentSourcePolicy** 对象来配置存储库镜像，则只能使用镜像的 registry 的全局 pull secret。您不能在项目中添加 pull secret。

12.5.1. 配置镜像 registry 存储库镜像

您可以创建安装后镜像配置自定义资源 (CR)，将源镜像 registry 中的镜像拉取请求重定向到镜像 registry。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 通过以下方法配置已镜像的存储库：
   - 按照 Red Hat Quay 存储库镜像 中所述，使用 Red Hat Quay 来设置已镜像的存储库。使用 Red Hat Quay 有助于您将镜像从一个存储库复制到另一存储库，并可随着时间的推移重复自动同步这些存储库。
   - 使用 **skopeo** 等工具手动将镜像从源存储库复制到已镜像的存储库。
     例如：在 Red Hat Enterprise Linux（RHEL 7 或 RHEL 8）系统上安装 skopeo RPM 软件包后，使用 **skopeo** 命令，如下例所示：

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi9/ubi-minimal:latest@sha256:5cf... \
     docker://example.io/example/ubi-minimal
     ```

     在本例中，您有一个名为 **example.io** 的容器镜像 registry，其中包含一个名为 **example** 的镜像存储库，您要将 **ubi9/ubi-minimal** 镜像从 **registry.access.redhat.com** 复制到其中。创建已镜像的 registry 后，您可以将 OpenShift Container Platform 集群配置为将源存储库的请求重定向到已镜像的存储库。
2. 登录您的 OpenShift Container Platform 集群。
3. 使用以下示例之一创建安装后镜像配置 CR：
   - 根据需要，创建一个 **ImageDigestMirrorSet** 或 **ImageTagMirrorSet** CR，将源和镜像 (mirror)替换为您自己的 registry、存储库对和镜像：

     ```
     apiVersion: config.openshift.io/v1 (1)
     kind: ImageDigestMirrorSet (2)
     metadata:
       name: ubi9repo
     spec:
       imageDigestMirrors: (3)
       - mirrors:
     ```

```
  - example.io/example/ubi-minimal (4)
  - example.com/example/ubi-minimal (5)
  source: registry.access.redhat.com/ubi9/ubi-minimal (6)
  mirrorSourcePolicy: AllowContactingSource (7)
- mirrors:
  - mirror.example.com/redhat
  source: registry.example.com/redhat (8)
  mirrorSourcePolicy: AllowContactingSource
- mirrors:
  - mirror.example.com
  source: registry.example.com (9)
  mirrorSourcePolicy: AllowContactingSource
- mirrors:
  - mirror.example.net/image
  source: registry.example.com/example/myimage (10)
  mirrorSourcePolicy: AllowContactingSource
- mirrors:
  - mirror.example.net
  source: registry.example.com/example (11)
  mirrorSourcePolicy: AllowContactingSource
- mirrors:
  - mirror.example.net/registry-example-com
  source: registry.example.com (12)
  mirrorSourcePolicy: AllowContactingSource
```

(1) 指明此 CR 要使用的 API。这必须是 **config.openshift.io/v1**。

(2) 根据 pull 类型指示对象类型：

- **ImageDigestMirrorSet** ：提取摘要引用镜像。
- **ImageTagMirrorSet** ：提取标签引用镜像。

(3) 表示镜像拉取方法的类型，请执行以下任一方法：

- **imageDigestMirrors** ：用于 **ImageDigestMirrorSet** CR。
- **imageTagMirrors** ：用于 **ImageTagMirrorSet** CR。

(4) 指明镜像 registry 和存储库的名称。

(5) 可选：指定每个目标仓库的二级镜像存储库。如果一个镜像停机，则目标仓库可以使用另一个镜像。

(6) 指明 registry 和存储库源，这是在镜像拉取规格中引用的存储库。

(7) 可选：如果镜像拉取失败，则指示回退策略：

- **AllowContactingSource** ：允许继续尝试从源存储库拉取镜像。这是默认值。
- **NeverContactSource**: 防止继续尝试从源存储库拉取镜像。

(8) 可选：指示 registry 中的命名空间，它允许您使用该命名空间中的任何镜像。如果您使用 registry 域作为源，则对象将应用到 registry 中的所有存储库。

(9) 可选：指示一个 registry，它允许您使用该 registry 中的任何镜像。如果指定了 registry

名称，对象将应用到源 registry 中的所有存储库到镜像 registry。

10 从 mirror **mirror.example.net/image@sha256:..** 拉取镜像 **registry.example.com/example/myimage@sha256:...**。

11 从 mirror **mirror.example.net/image@sha256:...** 的源 registry 命名空间中拉取镜像 **registry.example.com/example/image@sha256:...**。

12 从 mirror registry **example.net/registry-example-com/myimage@sha256:...** 中拉取镜像 **registry.example.com/myimage@sha256**。

- 创建 **ImageContentSourcePolicy** 自定义资源，将源和镜像替换为您自己的 registry、存储库对和镜像：

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: mirror-ocp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - mirror.registry.com:443/ocp/release 1
    source: quay.io/openshift-release-dev/ocp-release 2
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1 指定镜像 registry 和存储库的名称。

2 指定包含所镜像内容的在线 registry 和存储库。

4. 创建新对象：

```
$ oc create -f registryrepomirror.yaml
```

创建对象后，Machine Config Operator (MCO) 只会排空 **ImageTagMirrorSet** 对象的节点。MCO 不会排空 **ImageDigestMirrorSet** 和 **ImageContentSourcePolicy** 对象的节点。

5. 要检查是否应用了镜像的配置设置，请在其中一个节点上执行以下操作。

   a. 列出您的节点：

```
$ oc get node
```

输出示例

```
NAME                         STATUS                     ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal   Ready                         worker   7m   v1.28.5
ip-10-0-138-148.ec2.internal   Ready                          master   11m  v1.28.5
ip-10-0-139-122.ec2.internal   Ready                          master   11m  v1.28.5
ip-10-0-147-35.ec2.internal   Ready                         worker   7m   v1.28.5
ip-10-0-153-12.ec2.internal   Ready                         worker   7m   v1.28.5
ip-10-0-154-10.ec2.internal   Ready                         master   11m  v1.28.5
```

b. 启动调试过程以访问节点：

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

输出示例

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

c. 将您的根目录改为 **/host** ：

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

d. 检查 **/etc/containers/registries.conf** 文件，确保已完成更改：

```
sh-4.2# cat /etc/containers/registries.conf
```

以下输出代表了应用安装后镜像配置 CR 的 **registry.conf** 文件。最后的两个条目分别标记为 **digest-only** 和 **tag-only**。

输出示例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal" (1)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal" (2)
    pull-from-mirror = "digest-only" (3)

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"

  [[registry.mirror]]
    location = "mirror.example.net"
    pull-from-mirror = "digest-only"

[[registry]]
```

```
  prefix = ""
  location = "registry.example.com/example/myimage"

  [[registry.mirror]]
    location = "mirror.example.net/image"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/redhat"

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
    pull-from-mirror = "digest-only"
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal"
  blocked = true (4)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal-tag"
    pull-from-mirror = "tag-only" (5)
```

1. 指明在 pull spec 中引用的存储库。
2. 指明该存储库的镜像。
3. 表示从 mirror 的镜像拉取是一个摘要引用镜像。
4. 表示为此存储库设置了 **NeverContactSource** 参数。
5. 表示从 mirror 的镜像拉取是一个标签引用镜像。

e. 从源拉取镜像到节点，并检查是否通过 mirror 解析。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi9/ubi-
minimal@sha256:5cf...
```

存储库镜像故障排除

如果存储库镜像流程未按规定工作，请使用以下有关存储库镜像如何工作的信息协助排查问题。

- 首个工作镜像用于提供拉取（pull）的镜像。
- 只有在无其他镜像工作时，才会使用主 registry。
- 从系统上下文，**Insecure** 标志用作回退。

- 最近更改了 **/etc/containers/registries.conf** 文件的格式。现在它是第 2 版，采用 TOML 格式。

12.5.2. 为镜像 registry 存储库镜像转换 ImageContentSourcePolicy (ICSP) 文件

使用 **ImageContentSourcePolicy** (ICSP)对象配置存储库镜像是一个已弃用的功能。此功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

ICSP 对象被 **ImageDigestMirrorSet** 和 **ImageTagMirrorSet** 对象替代，以配置存储库镜像。如果您有用于创建 **ImageContentSourcePolicy** 对象的 YAML 文件，您可以使用 **oc adm migrate icsp** 命令将这些文件转换为 **ImageDigestMirrorSet** YAML 文件。命令将 API 更新至当前版本，将 **kind** 值更改为 **ImageDigestMirrorSet**，并将 **spec.repositoryDigestMirrors** 更改为 **spec.imageDigestMirrors**。文件的其余部分不会改变。

因为迁移不会更改 **registry.conf** 文件，所以集群不需要重启。

有关 **ImageDigestMirrorSet** 或 **ImageTagMirrorSet** 对象的更多信息，请参阅上一节中的"配置镜像 registry 存储库镜像"。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 确保集群中具有 **ImageContentSourcePolicy** 对象。

流程

1. 使用以下命令，将一个或多个 **ImageContentSourcePolicy** YAML 文件转换为 **ImageDigestMirrorSet** YAML 文件：

```
$ oc adm migrate icsp <file_name>.yaml <file_name>.yaml <file_name>.yaml --dest-dir
<path_to_the_directory>
```

其中：

<file_name>

指定源 **ImageContentSourcePolicy** YAML 的名称。您可以列出多个文件名。

--dest-dir

可选：指定输出 **ImageDigestMirrorSet** YAML 的目录。如果未设置，则会将该文件写入当前目录中。

例如，以下命令可将 **icsp.yaml** 和 **icsp-2.yaml** 文件转换，并将新的 YAML 文件保存到 **idms-files** 目录中。

```
$ oc adm migrate icsp icsp.yaml icsp-2.yaml --dest-dir idms-files
```

输**出示例**

```
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi8repo.5911620242173376087.yaml
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi9repo.6456931852378115011.yaml
```

2. 运行以下命令来创建 CR 对象：

```
$ oc create -f <path_to_the_directory>/<file-name>.yaml
```

其中：

<path_to_the_directory>
如果使用 **--dest-dir** 标志，请指定目录的路径。

<file_name>
指定 **ImageDigestMirrorSet** YAML 的名称。

3. 在推出 IDMS 对象后，删除 ICSP 对象。

12.6. 从镜像的 OPERATOR 目录填充 OPERATORHUB

如果用于断开连接的集群的 Operator 目录的镜像 (mirror)，您可以在镜像目录中为 OperatorHub 填充 Operator。您可以使用镜像过程中生成的清单来创建所需的 **ImageContentSourcePolicy** 和 **CatalogSource** 对象。

12.6.1. 先决条件

- 镜像用于断开连接的集群的 Operator 目录

12.6.2. 创建 ImageContentSourcePolicy 对象

将 Operator 目录内容镜像到镜像 registry 后，创建所需的 **ImageContentSourcePolicy** (ICSP)对象。ICSP 对象配置节点，以在 Operator 清单中存储的镜像引用和镜像的 registry 间进行转换。

流程

- 在可以访问断开连接的集群的主机上，运行以下命令指定 manifests 目录中的 **imageContentSourcePolicy.yaml** 文件来创建 ICSP：

```
$ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
```

其中 **<path/to/manifests/dir>** 是镜像内容的 manifests 目录的路径。

现在，您可以创建一个 **CatalogSource** 来引用您的镜像索引镜像和 Operator 内容。

12.6.3. 在集群中添加目录源

将目录源添加到 OpenShift Container Platform 集群可为用户发现和安装 Operator。集群管理员可以创建一个 **CatalogSource** 对象来引用索引镜像。OperatorHub 使用目录源来填充用户界面。

提示

或者，您可以使用 Web 控制台管理目录源。在 **Administration → Cluster Settings → Configuration → OperatorHub** 页面中，点 **Sources** 选项卡，您可以在其中创建、更新、删除、禁用和启用单独的源。

先决条件

- 构建并推送索引镜像到 registry。

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建一个 **CatalogSource** 对象来引用索引镜像。如果使用 **oc adm catalog mirror** 命令将目录镜像到目标 registry，您可以使用 manifests 目录中生成的 **catalogSource.yaml** 文件作为起点。

   a. 根据您的规格修改以下内容，并将它保存为 **catalogSource.yaml** 文件：

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog 1
  namespace: openshift-marketplace 2
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 3
  image: <registry>/<namespace>/redhat-operator-index:v4.15 4
  displayName: My Operator Catalog
  publisher: <publisher_name> 5
  updateStrategy:
    registryPoll: 6
      interval: 30m
```

   1. 如果您在上传到 registry 前将内容镜像到本地文件，请从 **metadata.name** 字段中删除任何反斜杠(/)字符，以避免在创建对象时出现 "invalid resource name" 错误。

   2. 如果您希望目录源对所有命名空间中的用户全局可用，请指定 **openshift-marketplace** 命名空间。否则，您可以指定一个不同的命名空间来对目录进行作用域并只对该命名空间可用。

   3. 指定 **legacy** 或 **restricted** 的值。如果没有设置该字段，则默认值为 **legacy**。在以后的 OpenShift Container Platform 发行版本中，计划默认值为 **restricted**。如果您的目录无法使用 **restricted** 权限运行，建议您手动将此字段设置为 **legacy**。

   4. 指定索引镜像。如果您在镜像名称后指定了标签，如 **:v4.15**，则目录源 Pod 会使用镜像 pull 策略 **Always**，这意味着 pod 始终在启动容器前拉取镜像。如果您指定了摘要，如 **@sha256:<id>**，则镜像拉取策略为 **IfNotPresent**，这意味着仅在节点上不存在的镜像时才拉取镜像。

   5. 指定发布目录的名称或机构名称。

   6. 目录源可以自动检查新版本以保持最新。

   b. 使用该文件创建 **CatalogSource** 对象：

```
$ oc apply -f catalogSource.yaml
```

2. 确定成功创建以下资源。

   a. 检查 pod：

```
$ oc get pods -n openshift-marketplace
```

输出示例

```
NAME                                READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6           1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. 检查目录源：

```
$ oc get catalogsource -n openshift-marketplace
```

输出示例

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc           5s
```

c. 检查软件包清单：

```
$ oc get packagemanifest -n openshift-marketplace
```

输出示例

```
NAME                     CATALOG               AGE
jaeger-product           My Operator Catalog   93s
```

现在，您可以在 OpenShift Container Platform Web 控制台中通过 **OperatorHub** 安装 Operator。

其他资源

- 从私有 registry 访问 Operator 的镜像
- 自定义目录源的镜像模板
- 镜像拉取（pull）策略

12.7. 关于使用 OPERATORHUB 安装 OPERATOR

OperatorHub 是一个发现 Operator 的用户界面，它与 Operator Lifecycle Manager（OLM）一起工作，后者在集群中安装和管理 Operator。

作为集群管理员，您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装来自 OperatorHub 的 Operator。将 Operator 订阅到一个或多个命名空间，供集群上的开发人员使用。

安装过程中，您必须为 Operator 确定以下初始设置：

安装模式

选择 **All namespaces on the cluster (default)**将 Operator 安装至所有命名空间；或选择单个命名空间（如果可用），仅在选定命名空间中安装 Operator。本例选择 **All namespaces...** 使 Operator 可供所有用户和项目使用。

更新频道

如果某个 Operator 可通过多个频道获得，则可任选您想要订阅的频道。例如，要通过 **stable** 频道部署（如果可用），则从列表中选择这个选项。

批准策略

您可以选择自动或者手动更新。
如果选择自动更新某个已安装的 Operator，则当所选频道中有该 Operator 的新版本时，Operator Lifecycle Manager（OLM）将自动升级 Operator 的运行实例，而无需人为干预。

如果选择手动更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。

12.7.1. 使用 Web 控制台从 OperatorHub 安装

您可以使用 OpenShift Container Platform Web 控制台从 OperatorHub 安装并订阅 Operator。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

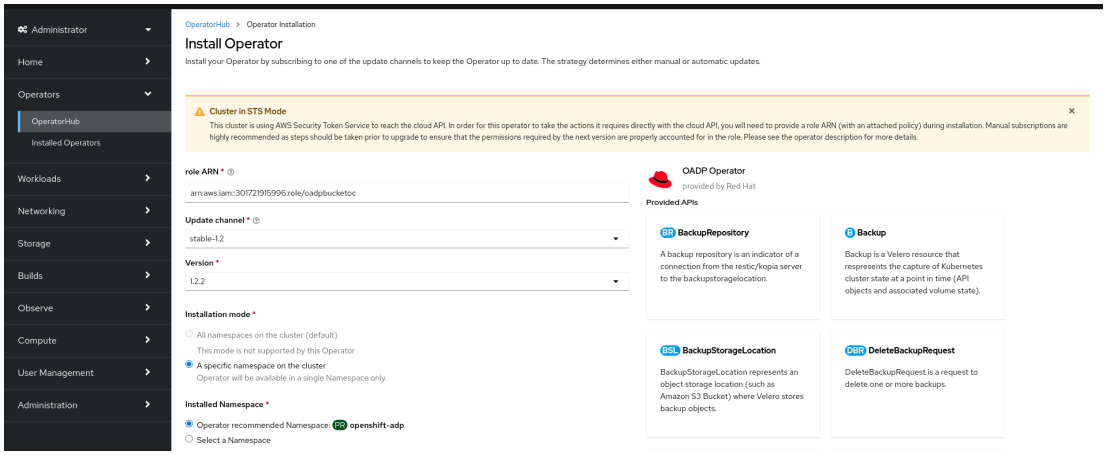
流程

1. 在 Web 控制台中导航至 **Operators → OperatorHub** 页面。
2. 找到您需要的 Operator（滚动页面会在 **Filter by keyword** 框中输入查找关键字）。例如，键入 **jaeger** 来查找 Jaeger Operator。
   您还可以根据**基础架构功能**过滤选项。例如，如果您希望 Operator 在断开连接的环境中工作，请选择 **Disconnected**。
3. 选择要显示更多信息的 Operator。

注意

选择 Community Operator 会警告红帽没有认证社区 Operator；您必须确认该警告方可继续。

4. 阅读 Operator 信息并单击 **Install**。
5. 在 **Install Operator** 页面中：
   a. 任选以下一项：
      - **All namespaces on the cluster (default)** 选择该项会将 Operator 安装至默认 **openshift-operators** 命名空间，以便供集群中的所有命名空间监视和使用。该选项并非始终可用。
      - **A specific namespace on the cluster**，该项支持您选择单一特定命名空间来安装 Operator。该 Operator 仅限在该单一命名空间中监视和使用。
   b. 对于启用了令牌身份验证的云供应商上的集群：
      - 如果集群使用 AWS STS (Web 控制台中的**STS 模式**)，在 **role ARN** 字段中输入服务帐户的 AWS IAM 角色的 Amazon Resource Name (ARN)。



要创建角色的 ARN，请按照 准备 AWS 帐户 中所述的步骤进行操作。

- 如果集群使用 Microsoft Entra Workload ID (Web 控制台中的**Workload Identity / Federated Identity Mode**)，请在适当的项中添加客户端 ID、租户 ID 和订阅 ID。

c. 如果有多个更新频道可用，请选择一个 **更新频道**。

d. 如前面所述，选择**自动**或**手动**批准策略。

重要

如果 web 控制台显示集群使用 AWS STS 或 Microsoft Entra Workload ID，您必须将 **Update approval** 设置为 **Manual**。

不建议使用具有自动更新批准的订阅，因为更新前可能会有权限更改。使用手动批准的订阅可确保管理员有机会验证更新版本的权限，并在更新前采取必要的操作。

6. 点击 **Install** 使 Operator 可供 OpenShift Container Platform 集群上的所选命名空间使用。

a. 如果选择了**手动**批准策略，订阅的升级状态将保持在 **Upgrading** 状态，直至您审核并批准安装计划。
在 **Install Plan** 页面批准后，订阅的升级状态将变为 **Up to date**。

b. 如果选择了 **Automatic** 批准策略，升级状态会在不用人工参与的情况下变为 **Up to date**。

7. 在订阅的升级状态成为 **Up to date** 后，选择 **Operators → Installed Operators** 来验证已安装 Operator 的 ClusterServiceVersion（CSV）是否最终出现了。**状态**最终会在相关命名空间中变为 **InstallSucceeded**。

注意

对于 **All namespaces...** 安装模式，状态在 **openshift-operators** 命名空间中解析为 **InstallSucceeded**，但如果检查其他命名空间，则状态为 **Copied**。

如果没有：

a. 检查 **openshift-operators** 项目（如果选择了 **A specific namespace...** 安装模式）中的 openshift-operators 项目中的 pod 的日志，这会在 **Workloads → Pods** 页面中报告问题以便进一步排除故障。

12.7.2. 使用 CLI 从 OperatorHub 安装

您可以使用 CLI 从 OperatorHub 安装 Operator，而不必使用 OpenShift Container Platform Web 控制台。使用 **oc** 命令来创建或更新一个**订阅**对象。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 查看 OperatorHub 中集群可用的 Operator 列表：

```
$ oc get packagemanifests -n openshift-marketplace
```

输**出示例**

```
NAME                               CATALOG               AGE
3scale-operator                    Red Hat Operators     91m
advanced-cluster-management        Red Hat Operators     91m
amq7-cert-manager                  Red Hat Operators     91m
...
couchbase-enterprise-certified     Certified Operators   91m
crunchy-postgres-operator          Certified Operators   91m
mongodb-enterprise                 Certified Operators   91m
...
etcd                               Community Operators   91m
jaeger                             Community Operators   91m
kubefed                            Community Operators   91m
...
```

记录下所需 Operator 的目录。

2. 检查所需 Operator，以验证其支持的安装模式和可用频道：

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

3. 一个 Operator 组（由 **OperatorGroup** 对象定义），在其中选择目标命名空间，在其中为与 Operator 组相同的命名空间中的所有 Operator 生成所需的 RBAC 访问权限。
订阅 Operator 的命名空间必须具有与 Operator 的安装模式相匹配的 Operator 组，可采用 **AllNamespaces** 模式，也可采用 **SingleNamespace** 模式。如果要安装的 Operator 使用 **AllNamespaces** 模式，**openshift-operators** 命名空间已有适当的 **global-operators** Operator 组。

   如果要安装的 Operator 采用 **SingleNamespace** 模式，而您没有适当的 Operator 组，则必须创建一个。

注意

- 在选择 **SingleNamespace** 模式时，该流程的 Web 控制台版本会在后台自动为您处理 **OperatorGroup** 和 **Subscription** 对象的创建。
- 每个命名空间只能有一个 Operator 组。如需更多信息，请参阅 "Operator 组"。

a. 创建 **OperatorGroup** 对象 YAML 文件，如 **operatorgroup.yaml**：

OperatorGroup 对象示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace>
spec:
  targetNamespaces:
  - <namespace>
```

b. 创建 **OperatorGroup** 对象：

```
$ oc apply -f operatorgroup.yaml
```

4. 创建一个 **Subscription** 对象 YAML 文件，以便为 Operator 订阅一个命名空间，如 **sub.yaml**：

Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators (1)
spec:
  channel: <channel_name> (2)
  name: <operator_name> (3)
  source: redhat-operators (4)
  sourceNamespace: openshift-marketplace (5)
  config:
    env: (6)
    - name: ARGS
      value: "-v=10"
    envFrom: (7)
    - secretRef:
        name: license-secret
    volumes: (8)
    - name: <volume_name>
      configMap:
        name: <configmap_name>
    volumeMounts: (9)
    - mountPath: <directory_name>
      name: <volume_name>
```

```
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1. 对于默认的 **AllNamespaces** 安装模式用法，请指定 **openshift-operators** 命名空间。另外，如果创建了自定义全局命名空间，您可以指定一个自定义全局命名空间。否则，为 **SingleNamespace** 安装模式使用指定相关单一命名空间。
2. 要订阅的频道的名称。
3. 要订阅的 Operator 的名称。
4. 提供 Operator 的目录源的名称。
5. 目录源的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub 目录源。
6. **env** 参数定义必须存在于由 OLM 创建的 pod 中所有容器中的环境变量列表。
7. **envFrom** 参数定义要在容器中填充环境变量的源列表。
8. **volumes** 参数定义 OLM 创建的 pod 上必须存在的卷列表。
9. **volumeMounts** 参数定义由 OLM 创建的 pod 中必须存在的卷挂载列表。如果 **volumeMount** 引用不存在的 **卷**，OLM 无法部署 Operator。
10. **tolerations** 参数为 OLM 创建的 pod 定义 Tolerations 列表。
11. **resources** 参数为 OLM 创建的 pod 中所有容器定义资源限制。
12. **nodeSelector** 参数为 OLM 创建的 pod 定义 **NodeSelector**。

5. 对于启用了令牌身份验证的云供应商上的集群：

   a. 确保 **Subscription** 对象被设置为手动更新批准：

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual 1
```

   1. 不建议使用具有自动更新批准的订阅，因为更新前可能会有权限更改。使用手动批准的订阅可确保管理员有机会验证更新版本的权限，并在更新前采取必要的操作。

   b. 在 **Subscription** 对象的 **config** 部分包括相关的云供应商相关字段：

   - 如果集群处于 AWS STS 模式，请包含以下字段：
     -

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" (1)
```

(1) 包含角色 ARN 详情。

- 如果集群处于 Microsoft Entra Workload ID 模式，请包括以下字段：

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" (1)
   - name: TENANTID
     value: "<tenant_id>" (2)
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" (3)
```

(1) 包含客户端 ID。

(2) 包含租户 ID。

(3) 包括订阅 ID。

6. 创建 **Subscription** 对象：

```
$ oc apply -f sub.yaml
```

此时，OLM 已了解所选的 Operator。Operator 的集群服务版本（CSV）应出现在目标命名空间中，由 Operator 提供的 API 应可用于创建。

其他资源

- 关于 OperatorGroup

第 13 章 更改云供应商凭证配置

对于支持的配置，您可以更改 OpenShift Container Platform 与云供应商进行身份验证的方式。

要确定集群使用哪个云凭证策略，请参阅 确定 Cloud Credential Operator 模式。

13.1. 轮转或删除云供应商凭证

安装 OpenShift Container Platform 后，一些机构需要轮转或删除初始安装过程中使用的云供应商凭证。

要允许集群使用新凭证，您必须更新 Cloud Credential Operator（CCO）用来管理云供应商凭证的 secret。

13.1.1. 使用 Cloud Credential Operator 实用程序轮转云供应商凭证

Cloud Credential Operator (CCO) 实用程序 **ccoctl** 支持为 IBM Cloud® 上安装的集群更新 secret。

13.1.1.1. 轮转 API 密钥

您可以为现有服务 ID 轮转 API 密钥，并更新对应的 secret。

先决条件

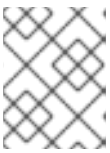
- 您已配置了 **ccoctl** 二进制文件。
- 已安装 live OpenShift Container Platform 集群中现有的服务 ID。

流程

- 使用 **ccoctl** 实用程序轮转服务 ID 的 API 密钥并更新 secret：

```
$ ccoctl <provider_name> refresh-keys \ 1
  --kubeconfig <openshift_kubeconfig_file> \ 2
  --credentials-requests-dir <path_to_credential_requests_directory> \ 3
  --name <name> 4
```

1. 供应商的名称。例如：**ibmcloud** 或 **powervs**。
2. 与集群关联的 **kubeconfig** 文件。例如，**<installation_directory>/auth/kubeconfig**.
3. 存储了凭据请求的目录。
4. OpenShift Container Platform 集群的名称。



注意

如果您的集群使用 **TechPreviewNoUpgrade** 功能集启用的技术预览功能，则必须包含 **--enable-tech-preview** 参数。

13.1.2. 维护云供应商凭证

如果因为某种原因更改了云供应商凭证，您必须手动更新 Cloud Credential Operator（CCO）用来管理云供应商凭证的 secret。

轮转云凭证的过程取决于 CCO 配置使用的模式。在为使用 mint 模式的集群轮转凭证后，您必须手动删除由删除凭证创建的组件凭证。

先决条件

- 您的集群会在支持使用您要使用的 CCO 模式手动轮转云凭证的平台上安装：
  - 对于 mint 模式，支持 Amazon Web Services (AWS)和 Google Cloud Platform (GCP)。
  - 对于 passthrough 模式，支持 Amazon Web Services (AWS)、Microsoft Azure、Google Cloud Platform (GCP)、Red Hat OpenStack Platform (RHOSP)和 VMware vSphere。
- 您已更改了用于与云供应商接口的凭证。
- 新凭证有足够的权限来在集群中使用 CCO 模式。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Workloads → Secrets**。
2. 在 **Secrets** 页面的表中，找到您的云供应商的 root secret。

平台	Secret 名称
AWS	**aws-creds**
Azure	**azure-credentials**
GCP	**gcp-credentials**
RHOSP	**openstack-credentials**
VMware vSphere	**vsphere-creds**

3. 点击与 secret 相同的行  中的 **Options** 菜单，然后选择 **Edit Secret**。
4. 记录 **Value** 字段的内容。您可以使用这些信息验证在更新凭证后该值是否不同。
5. 使用云供应商的新身份验证信息更新 **Value** 字段的文本，然后点 **Save**。
6. 如果您要为没有启用 vSphere CSI Driver Operator 的 vSphere 集群更新凭证，您必须强制推出 Kubernetes 控制器管理器以应用更新的凭证。

注意

如果启用了 vSphere CSI Driver Operator，则不需要这一步。

要应用更新的 vSphere 凭证，请以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform CLI，并运行以下命令：

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

当凭证被推出时，Kubernetes Controller Manager Operator 的状态会报告 **Progressing=true**。要查看状态，请运行以下命令：

```
$ oc get co kube-controller-manager
```

a. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform CLI。

b. 获取所有引用的组件 secret 的名称和命名空间：

```
$ oc -n openshift-cloud-credential-operator get CredentialsRequest \
  -o json | jq -r '.items[] | select (.spec.providerSpec.kind=="<provider_spec>") |
.spec.secretRef'
```

其中 **<provider_spec>** 是您的云供应商的对应值：

- AWS: **AWSProviderSpec**
- GCP: **GCPProviderSpec**

AWS 输出的部分示例

```
{
  "name": "ebs-cloud-credentials",
  "namespace": "openshift-cluster-csi-drivers"
}
{
  "name": "cloud-credential-operator-iam-ro-creds",
  "namespace": "openshift-cloud-credential-operator"
}
```

c. 删除每个引用的组件 secret：

```
$ oc delete secret <secret_name> \ 1
  -n <secret_namespace> 2
```

1 指定 secret 的名称。

2 指定包含 secret 的命名空间。

删除 AWS secret 示例

```
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

您不需要从供应商控制台手动删除凭证。删除引用的组件 secret 将导致 CCO 从平台中删除现有凭证并创建新凭证。

验证

验证凭证是否已更改：

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Workloads → Secrets**。
2. 验证 **Value** 字段的内容已改变。

其他资源

- vSphere CSI Driver Operator

13.1.3. 删除云供应商凭证

在以 mint 模式使用 Cloud Credential Operator（CCO）安装 OpenShift Container Platform 集群后，您可以从集群中的 **kube-system** 命名空间中删除管理员级别的凭证 secret。只有在进行更改时（需要提高的权限，如升级），才需要管理员级别的凭证。

注意

在非 z-stream 升级前，您必须使用管理员级别的凭证重新恢复凭证 secret。如果没有凭证，则可能会阻止升级。

先决条件

- 集群安装在支持从 CCO 中删除云凭证的平台上。支持的平台是 AWS 和 GCP。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Workloads → Secrets**。
2. 在 **Secrets** 页面的表中，找到您的云供应商的 root secret。

平台	Secret 名称
AWS	**aws-creds**
GCP	**gcp-credentials**

3. 点击与 secret 相同的行  中的 **Options** 菜单，然后选择 **Delete Secret**。

其他资源

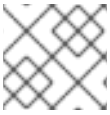
- 管理凭证 root secret 格式

13.2. 启用基于令牌的身份验证

安装 Microsoft Azure OpenShift Container Platform 集群后，您可以启用 Microsoft Entra Workload ID 来使用简短凭证。

13.2.1. 配置 Cloud Credential Operator 工具

当 Cloud Credential Operator(CCO)以手动模式运行时，要从集群外部创建和管理云凭证，提取并准备 CCO 实用程序(**ccoctl**)二进制文件。



注意

ccoctl 工具是在 Linux 环境中运行的 Linux 二进制文件。

先决条件

- 您可以访问具有集群管理员权限的 OpenShift Container Platform 帐户。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 运行以下命令，为 OpenShift Container Platform 发行镜像设置变量：

```
$ RELEASE_IMAGE=$(./openshift-install version | awk '/release image/ {print $3}')
```

2. 运行以下命令，从 OpenShift Container Platform 发行镜像获取 CCO 容器镜像：

```
$ CCO_IMAGE=$(oc adm release info --image-for='cloud-credential-operator'
$RELEASE_IMAGE -a ~/.pull-secret)
```

注意

确保 **$RELEASE_IMAGE** 的架构与将使用 **ccoctl** 工具的环境架构相匹配。

3. 运行以下命令，将 CCO 容器镜像中的 **ccoctl** 二进制文件提取到 OpenShift Container Platform 发行镜像中：

```
$ oc image extract $CCO_IMAGE --file="/usr/bin/ccoctl" -a ~/.pull-secret
```

4. 运行以下命令更改权限以使 **ccoctl** 可执行：

```
$ chmod 775 ccoctl
```

验证

- 要验证 **ccoctl** 是否准备就绪，可以尝试显示帮助文件。运行命令时使用相对文件名，例如：

```
$ ./ccoctl.rhel9
```

输出示例

```
OpenShift credentials provisioning tool

Usage:
  ccoctl [command]

Available Commands:
  alibabacloud Manage credentials objects for alibaba cloud
  aws          Manage credentials objects for AWS cloud
  azure        Manage credentials objects for Azure
  gcp          Manage credentials objects for Google cloud
```

```
  help        Help about any command
  ibmcloud    Manage credentials objects for IBM Cloud
  nutanix     Manage credentials objects for Nutanix

Flags:
  -h, --help   help for ccoctl

Use "ccoctl [command] --help" for more information about a command.
```

13.2.2. 在现有集群中启用 Microsoft Entra Workload ID

如果您没有将 Microsoft Azure OpenShift Container Platform 集群配置为在安装过程中使用 Microsoft Entra Workload ID，您可以在现有集群上启用此验证方法。

重要

在现有集群中启用工作负载 ID 的过程具有破坏性，需要花费大量时间。在继续操作前，请观察以下注意事项：

- 阅读以下步骤，并确保您了解并接受时间要求。具体的时间要求因单个集群而异，但可能需要至少一个小时。
- 在此过程中，您必须刷新所有服务帐户并重启集群中的所有 pod。这些操作对工作负载具有破坏性。要缓解这种影响，您可以临时停止这些服务，然后在集群就绪时重新部署它们。
- 启动此过程后，在完成前不要尝试更新集群。如果触发更新，在现有集群中启用 Workload ID 的过程会失败。

先决条件

- 您已在 Microsoft Azure 上安装了 OpenShift Container Platform 集群。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 您已提取并准备 Cloud Credential Operator 实用程序 (**ccoctl**) 二进制文件。
- 您可以使用 Azure CLI (**az**) 访问 Azure 帐户。

流程

1. 为 **ccoctl** 工具生成的清单创建一个输出目录。此流程使用 **./output_dir** 作为示例。
2. 运行以下命令，将集群的服务帐户公钥密钥提取到输出目录中：

```
$ oc get configmap \
  --namespace openshift-kube-apiserver bound-sa-token-signing-certs \
  --output 'go-template={{index .data "service-account-001.pub"}}' >
./output_dir/serviceaccount-signer.public 1
```

1 此流程使用名为 **serviceaccount-signer.public** 的文件作为示例。

3. 运行以下命令，使用提取的服务帐户公钥创建 OpenID Connect (OIDC) 签发者和 Azure blob 存储容器：

```
$ ./ccoctl azure create-oidc-issuer \
  --name <azure_infra_name> \ 1
  --output-dir ./output_dir \
  --region <azure_region> \ 2
  --subscription-id <azure_subscription_id> \ 3
  --tenant-id <azure_tenant_id> \
  --public-key-file ./output_dir/serviceaccount-signer.public 4
```

1. **name** 参数的值用于创建 Azure 资源组。要使用现有的 Azure 资源组而不是创建新 Azure 资源组，请使用现有组名称指定 **--oidc-resource-group-name** 参数作为其值。
2. 指定现有集群的区域。
3. 指定现有集群的订阅 ID。
4. 指定包含集群服务帐户公钥的文件。

4. 运行以下命令，验证 Azure pod 身份 Webhook 的配置文件是否已创建：

```
$ ll ./output_dir/manifests
```

输**出示例**

```
total 8
-rw-------. 1 cloud-user cloud-user 193 May 22 02:29 azure-ad-pod-identity-webhook-
config.yaml 1
-rw-------. 1 cloud-user cloud-user 165 May 22 02:29 cluster-authentication-02-config.yaml
```

1. 文件 **azure-ad-pod-identity-webhook-config.yaml** 包含 Azure pod 身份 Webhook 配置。

5. 运行以下命令，使用输出目录中生成的清单中的 **OIDC_ISSUER_URL** 变量设置 OIDC_ISSUER_URL 变量：

```
$ OIDC_ISSUER_URL=`awk '/serviceAccountIssuer/ { print $2 }'
./output_dir/manifests/cluster-authentication-02-config.yaml`
```

6. 运行以下命令，更新集群 **身份验证** 配置的 **spec.serviceAccountIssuer** 参数：

```
$ oc patch authentication cluster \
  --type=merge \
  -p "{\"spec\":{\"serviceAccountIssuer\":\"${OIDC_ISSUER_URL}\"}}"
```

7. 运行以下命令监控配置更新进度：

```
$ oc adm wait-for-stable-cluster
```

这个过程可能需要 15 分钟或更长时间。以下输出表示进程已完成：

```
All clusteroperators are stable
```

8. 运行以下命令重启集群中的所有 pod：

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

重启 pod 会更新 **serviceAccountIssuer** 字段，并刷新服务帐户公钥。

9. 运行以下命令监控重启和更新进程：

```
$ oc adm wait-for-node-reboot nodes --all
```

这个过程可能需要 15 分钟或更长时间。以下输出表示进程已完成：

```
All nodes rebooted
```

10. 运行以下命令，将 Cloud Credential Operator **spec.credentialsMode** 参数更新为 **Manual** ：

```
$ oc patch cloudcredential cluster \
  --type=merge \
  --patch '{"spec":{"credentialsMode":"Manual"}}'
```

11. 运行以下命令，从 OpenShift Container Platform 发行镜像中提取 **CredentialsRequest** 对象列表：

```
$ oc adm release extract \
  --credentials-requests \
  --included \
  --to <path_to_directory_for_credentials_requests> \
  --registry-config ~/.pull-secret
```

注意

此命令可能需要一些时间才能运行。

12. 运行以下命令，使用 Azure 资源组名称设置 **AZURE_INSTALL_RG** 变量：

```
$ AZURE_INSTALL_RG=`oc get infrastructure cluster -o jsonpath --template '{
.status.platformStatus.azure.resourceGroupName }'`
```

13. 运行以下命令，使用 **ccoctl** 工具为所有 **CredentialsRequest** 对象创建受管身份：

```
$ ccoctl azure create-managed-identities \
  --name <azure_infra_name> \
  --output-dir ./output_dir \
  --region <azure_region> \
  --subscription-id <azure_subscription_id> \
  --credentials-requests-dir <path_to_directory_for_credentials_requests> \
  --issuer-url "${OIDC_ISSUER_URL}" \
  --dnszone-resource-group-name <azure_dns_zone_resourcegroup_name> \ 1
  --installation-resource-group-name "${AZURE_INSTALL_RG}"
```

1 指定包含 DNS 区的资源组的名称。

14. 运行以下命令，为 Workload ID 应用 Azure pod 身份 Webhook 配置：

```
$ oc apply -f ./output_dir/manifests/azure-ad-pod-identity-webhook-config.yaml
```

15. 运行以下命令应用 **ccoctl** 工具生成的 secret：

```
$ find ./output_dir/manifests -iname "openshift*yaml" -print0 | xargs -I {} -0 -t oc replace -f {}
```

这可能需要几分钟。

16. 运行以下命令重启集群中的所有 pod：

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

重启 pod 会更新 **serviceAccountIssuer** 字段，并刷新服务帐户公钥。

17. 运行以下命令监控重启和更新进程：

```
$ oc adm wait-for-node-reboot nodes --all
```

这个过程可能需要 15 分钟或更长时间。以下输出表示进程已完成：

```
All nodes rebooted
```

18. 运行以下命令监控配置更新进度：

```
$ oc adm wait-for-stable-cluster
```

这个过程可能需要 15 分钟或更长时间。以下输出表示进程已完成：

```
All clusteroperators are stable
```

19. 可选：运行以下命令来删除 Azure root 凭证 secret：

```
$ oc delete secret -n kube-system azure-credentials
```

其他资源

- Microsoft Entra Workload ID
- 配置 Azure 集群以使用短期凭证

13.2.3. 验证集群是否使用短期凭证

您可以通过检查集群中的 Cloud Credential Operator (CCO) 配置和其他值来验证集群是否对各个组件使用简短安全凭证。

先决条件

- 已使用 Cloud Credential Operator 实用程序(**ccoctl**)部署了 OpenShift Container Platform 集群来实现短期凭证。
- 已安装 OpenShift CLI（**oc**）。
- 您以具有 **cluster-admin** 权限的用户身份登录。

流程

- 运行以下命令，验证 CCO 是否配置为以手动模式运行：

```
$ oc get cloudcredentials cluster \
  -o=jsonpath={.spec.credentialsMode}
```

以下输出确认 CCO 以手动模式运行：

输出示例

```
Manual
```

- 运行以下命令，验证集群没有 **root** 凭证：

```
$ oc get secrets \
  -n kube-system <secret_name>
```

其中 **<secret_name>** 是云供应商的 root secret 的名称。

平台	Secret 名称
Amazon Web Services (AWS)	**aws-creds**
Microsoft Azure	**azure-credentials**
Google Cloud Platform (GCP)	**gcp-credentials**

一个错误确认集群中不存在 root secret。

AWS 集群的输出示例

```
Error from server (NotFound): secrets "aws-creds" not found
```

- 运行以下命令，验证组件是否在单个组件中使用短期安全凭证：

```
$ oc get authentication cluster \
  -o jsonpath \
  --template='{ .spec.serviceAccountIssuer }'
```

此命令显示集群 **Authentication** 对象中 **.spec.serviceAccountIssuer** 参数的值。与云供应商关联的 URL 的输出表示集群使用从集群外部创建和管理的简短凭证的手动模式。

- Azure 集群：通过运行以下命令，验证组件假定 secret 清单中指定的 Azure 客户端 ID：

```
$ oc get secrets \
  -n openshift-image-registry installer-cloud-credentials \
  -o jsonpath='{.data}'
```

输出中包含了 **azure_client_id** 和 **azure_federated_token_file** 字段代表组件假定 Azure 客户端 ID。

- Azure 集群 ：运行以下命令来验证 pod 身份 Webhook 是否正在运行：

```
$ oc get pods \
  -n openshift-cloud-credential-operator
```

输**出示例**

```
NAME                                READY   STATUS    RESTARTS   AGE
cloud-credential-operator-59cf744f78-r8pbq   2/2     Running   2          71m
pod-identity-webhook-548f977b4c-859lz        1/1     Running   1          70m
```

13.3. 其他资源

- 关于 Cloud Credential Operator

第 14 章 配置警报通知

在 OpenShift Container Platform 中，当警报规则中定义的条件满足时会触发警报。警报提供一条通知，表示集群中存在一组情况。默认情况下，触发的警报可在 OpenShift Container Platform Web 控制台中的 Alerting UI 中查看。安装后，您可以配置 OpenShift Container Platform，将警报通知发送到外部系统。

14.1. 将通知发送到外部系统

在 OpenShift Container Platform 4.15 中，可在 Alerting UI 中查看触发警报。默认不会将警报配置为发送到任何通知系统。您可以将 OpenShift Container Platform 配置为将警报发送到以下类型的接收器：

- PagerDuty
- Webhook
- 电子邮件
- Slack
- Microsoft Teams

通过将警报路由到接收器，您可在出现故障时及时向适当的团队发送通知。例如，关键警报需要立即关注，通常会传给个人或关键响应团队。相反，提供非关键警告通知的警报可能会被路由到一个问题单系统进行非即时的审阅。

使用 watchdog 警报检查警报是否工作正常

OpenShift Container Platform 监控功能包含持续触发的 watchdog 警报。Alertmanager 重复向已配置的通知提供程序发送 watchdog 警报通知。此提供程序通常会配置为在其停止收到 watchdog 警报时通知管理员。这种机制可帮助您快速识别 Alertmanager 和通知提供程序之间的任何通信问题。

14.2. 其他资源

- 监控概述
- 配置警报接收器

第 15 章 将连接的集群转换为断开连接的集群

在某些情况下，您可能需要将 OpenShift Container Platform 集群从连接的集群转换为断开连接的集群。

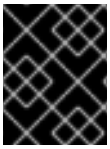
断开连接的集群（也称为受限集群）没有与互联网的活跃连接。因此，您必须镜像 registry 和安装介质的内容。您可以在可以访问互联网和您的封闭网络的主机上创建此镜像 registry，或者将镜像复制到可跨网络界限的设备中。

本主题描述了将现有连接的集群转换为断开连接的集群的一般过程。

15.1. 关于镜像 REGISTRY

您可以镜像 OpenShift Container Platform 安装所需的镜像，以及容器镜像 registry 的后续产品更新，如 Red Hat Quay、JFrog Artifactory、Sonatype Nexus Repository 或 Harbor。如果您无法访问大型容器 registry，可以使用 *mirror registry for Red Hat OpenShift*，它是包括在 OpenShift Container Platform 订阅中的一个小型容器 registry。

您可以使用支持 Docker v2-2 的任何容器 registry，如 Red Hat Quay，*mirror registry for Red Hat OpenShift*, Artifactory, Sonatype Nexus Repository, 或 Harbor。无论您所选 registry 是什么，都会将互联网上红帽托管站点的内容镜像到隔离的镜像 registry 相同。镜像内容后，您要将每个集群配置为从镜像 registry 中检索此内容。



重要

OpenShift 镜像 registry 不能用作目标 registry，因为它不支持没有标签的推送，在镜像过程中需要这个推送。

如果选择的容器 registry 不是 *mirror registry for Red Hat OpenShift*，则需要集群中置备的每台机器都可以访问它。如果 registry 无法访问，安装、更新或常规操作（如工作负载重新定位）可能会失败。因此，您必须以高度可用的方式运行镜像 registry，镜像 registry 至少必须与 OpenShift Container Platform 集群的生产环境可用性相匹配。

使用 OpenShift Container Platform 镜像填充镜像 registry 时，可以遵循以下两种情况。如果您的主机可以同时访问互联网和您的镜像 registry，而不能访问您的集群节点，您可以直接从该机器中镜像该内容。这个过程被称为 *连接的镜像(mirror)*。如果没有这样的主机，则必须将该镜像文件镜像到文件系统中，然后将该主机或者可移动介质放入受限环境中。这个过程被称为 *断开连接的镜像*。

对于已镜像的 registry，若要查看拉取镜像的来源，您必须查看 **Trying 以访问** CRI-O 日志中的日志条目。查看镜像拉取源的其他方法（如在节点上使用 **crictl images** 命令）显示非镜像镜像名称，即使镜像是从镜像位置拉取的。

注意

红帽没有针对 OpenShift Container Platform 测试第三方 registry。

15.2. 先决条件

- 已安装 **oc** 客户端。
- 一个正在运行的集群。
- 安装的镜像 registry，这是支持托管 OpenShift Container Platform 集群的位置的 Docker v2-2 的容器镜像 registry，如以下 registry 之一：
  - Red Hat Quay

- JFrog Artifactory
- Sonatype Nexus 仓库
- Harbor

如果您有 Red Hat Quay 订阅，请参阅有关部署 Red Hat Quay 用于概念验证 的文档，或使用 Quay Operator。

- 必须将镜像存储库配置为共享镜像。例如，Red Hat Quay 仓库需要 机构 才能共享镜像。
- 访问互联网来获取所需的容器镜像。

15.3. 为镜像准备集群

在断开集群的连接前，您必须镜像(mirror)或复制(mirror)或复制(mirror)或复制镜像(mirror) registry。要镜像镜像，您必须通过以下方法准备集群：

- 将镜像 registry 证书添加到主机上的可信 CA 列表中。
- 创建包含您的镜像 pull secret 的 **.dockerconfigjson** 文件，该文件来自 **cloud.openshift.com** 令牌。

流程

1. 配置允许镜像镜像的凭证：
   a. 将镜像 registry 的 CA 证书（采用简单 PEM 或 DER 文件格式）添加到可信 CA 列表中。例如：

   ```
   $ cp </path/to/cert.crt> /usr/share/pki/ca-trust-source/anchors/
   ```

   其中, </path/to/cert.crt>
   指定本地文件系统中的证书路径。

   b. 更新 CA 信任。例如，在 Linux 中：

   ```
   $ update-ca-trust
   ```

   c. 从全局 pull secret 中提取 **.dockerconfigjson** 文件：

   ```
   $ oc extract secret/pull-secret -n openshift-config --confirm --to=.
   ```

   输出示例

   ```
   .dockerconfigjson
   ```

   d. 编辑 **.dockerconfigjson** 文件以添加您的镜像 registry 和身份验证凭证，并将它保存为新文件：

   ```
   {"auths":{"<local_registry>": {"auth": "<credentials>","email": "you@example.com"}}},"
   <registry>:<port>/<namespace>/":{"auth":"<token>"}}}
   ```

其中：

<local_registry>

指定 registry 域名，以及您的镜像 registry 用来提供内容的可选端口。

auth

为您的镜像 registry 指定 base64 编码的用户名和密码。

<registry>:<port>/<namespace>

指定镜像 registry 详情。

<token>

为您的镜像 registry 指定 base64 编码的 **username:password**。
例如：

```
$ {"auths":{"cloud.openshift.com":
{"auth":"b3BlbnNoaWZ0Y3UjhGOVZPT0lOMEFaUjdPUzRGTA==","email":"user@exa
mple.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UtZGOVZPT0lOMEFaUGSTd4VGVGVUjdPUzR
GTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VHkxOSTd4VGVGVU1MdTpleUpoYkdjaUail
A==","email":"user@example.com"},
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VH3BGSTd4VGVGVU1MdTpleUpoYkdjaU9
fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwWVFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1
ZXeTRV"},"my-registry:5000/my-namespace/":
{"auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}
```

15.4. 对容器镜像进行镜像处理(MIRROR)

正确配置集群后，您可以将外部存储库中的镜像镜像到镜像存储库。

流程

1. 镜像 Operator Lifecycle Manager(OLM)镜像：

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v{product-version}
<mirror_registry>:<port>/olm -a <reg_creds>
```

其中：

product-version

指定要与 OpenShift Container Platform 版本对应的标签，如 **4.8**。

mirror_registry

指定用于镜像 Operator 内容的目标 registry 和命名空间的完全限定域名(FQDN)，其中 **<namespace>** 是 registry 上的任何现有命名空间。

reg_creds

指定您修改的 **.dockerconfigjson** 文件的位置。

例如：

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

2. 为任何其他红帽提供的 Operator 镜像内容：

```
$ oc adm catalog mirror <index_image> <mirror_registry>:<port>/<namespace> -a
<reg_creds>
```

其中：

index_image

指定您要镜像的目录的索引镜像。

mirror_registry

指定要将 Operator 内容镜像到的目标 registry 和命名空间的 FQDN，其中 **<namespace>** 是 registry 上的任何现有命名空间。

reg_creds

可选：如果需要，指定 registry 凭证文件的位置。

例如：

```
$ oc adm catalog mirror registry.redhat.io/redhat/community-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

3. 镜像 OpenShift Container Platform 镜像存储库：

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-
release:v<product-version>-<architecture> --to=<local_registry>/<local_repository> --to-
release-image=<local_registry>/<local_repository>:v<product-version>-<architecture>
```

其中：

product-version

指定与要安装的 OpenShift Container Platform 版本对应的标签，如 **4.8.15-x86_64**。

架构

为您的服务器指定构架类型，如 **x86_64**。

local_registry

指定镜像存储库的 registry 域名。

local_repository

指定要在 registry 中创建的存储库名称，如 **ocp4/openshift4**。

例如：

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-
release:4.8.15-x86_64 --to=mirror.registry.com:443/ocp/release --to-release-
image=mirror.registry.com:443/ocp/release:4.8.15-x86_64
```

输**出示例**

■

```
info: Mirroring 109 images to mirror.registry.com/ocp/release ...
mirror.registry.com:443/
  ocp/release
 manifests:
   sha256:086224cadce475029065a0efc5244923f43fb9bb3bb47637e0aaf1f32b9cad47 ->
4.8.15-x86_64-thanos
   sha256:0a214f12737cb1cfbec473cc301aa2c289d4837224c9603e99d1e90fc00328db ->
4.8.15-x86_64-kuryr-controller
   sha256:0cf5fd36ac4b95f9de506623b902118a90ff17a07b663aad5d57c425ca44038c ->
4.8.15-x86_64-pod
   sha256:0d1c356c26d6e5945a488ab2b050b75a8b838fc948a75c0fa13a9084974680cb ->
4.8.15-x86_64-kube-client-agent

…..
sha256:66e37d2532607e6c91eedf23b9600b4db904ce68e92b43c43d5b417ca6c8e63c
mirror.registry.com:443/ocp/release:4.5.41-multus-admission-controller
sha256:d36efdbf8d5b2cbc4dcdbd64297107d88a31ef6b0ec4a39695915c10db4973f1
mirror.registry.com:443/ocp/release:4.5.41-cluster-kube-scheduler-operator
sha256:bd1baa5c8239b23ecdf76819ddb63cd1cd6091119fecdbf1a0db1fb3760321a2
mirror.registry.com:443/ocp/release:4.5.41-aws-machine-controllers
info: Mirroring completed in 2.02s (0B/s)

Success
Update image:  mirror.registry.com:443/ocp/release:4.5.41-x86_64
Mirror prefix: mirror.registry.com:443/ocp/release
```

4. 根据需要镜像任何其他 registry：

```
$ oc image mirror <online_registry>/my/image:latest <mirror_registry>
```

附加信息

- 如需有关镜像 Operator 目录的更多信息，请参阅 镜像 Operator 目录。
- 如需有关 **oc adm catalog mirror** 命令的更多信息，请参阅 OpenShift CLI 管理员命令参考。

15.5. 为镜像 REGISTRY 配置集群

在将镜像创建并镜像(mirror)到镜像 registry 后，您必须修改集群，以便 pod 可以从镜像 registry 中拉取镜像。

您必须：

- 将镜像 registry 凭证添加到全局 pull secret 中。
- 在集群中添加镜像 registry 服务器证书。
- 创建 **ImageContentSourcePolicy** 自定义资源(ICSP)，它将镜像 registry 与源 registry 关联。

1. 将镜像 registry 凭证添加到集群全局 pull-secret 中：

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

1 提供新 pull secret 文件的路径。

例如：

```
$ oc set data secret/pull-secret -n openshift-config --from-
file=.dockerconfigjson=.mirrorsecretconfigjson
```

2. 将 CA 签名的镜像 registry 服务器证书添加到集群中的节点：

   a. 创建包含镜像 registry 的服务器证书的配置映射

   ```
   $ oc create configmap <config_map_name> --from-file=<mirror_address_host>..
   <port>=$path/ca.crt -n openshift-config
   ```

   例如：

   ```
   S oc create configmap registry-config --from-
   file=mirror.registry.com..443=/root/certs/ca-chain.cert.pem -n openshift-config
   ```

   b. 使用配置映射更新 **image.config.openshift.io/cluster** 自定义资源(CR)。OpenShift Container Platform 将对此 CR 的更改应用到集群中的所有节点：

   ```
   $ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
   {"name":"<config_map_name>"}}}' --type=merge
   ```

   例如：

   ```
   $ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
   {"name":"registry-config"}}}' --type=merge
   ```

3. 创建一个 ICSP，将在线 registry 中的容器拉取请求重定向到镜像 registry：

   a. 创建 **ImageContentSourcePolicy** 自定义资源：

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: ImageContentSourcePolicy
   metadata:
     name: mirror-ocp
   spec:
     repositoryDigestMirrors:
     - mirrors:
       - mirror.registry.com:443/ocp/release (1)
       source: quay.io/openshift-release-dev/ocp-release (2)
     - mirrors:
       - mirror.registry.com:443/ocp/release
       source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
   ```

   (1) 指定镜像 registry 和存储库的名称。

   (2) 指定包含所镜像内容的在线 registry 和存储库。

   b. 创建 ICSP 对象：

   ```
   $ oc create -f registryrepomirror.yaml
   ```

输**出示例**

```
imagecontentsourcepolicy.operator.openshift.io/mirror-ocp created
```

OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

4. 验证已添加了镜像 registry 的凭证、CA 和 ICSP：

   a. 登录到节点：

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

   b. 将 **/host** 设置为 debug shell 中的根目录：

   ```
   sh-4.4# chroot /host
   ```

   c. 检查凭证的 **config.json** 文件：

   ```
   sh-4.4# cat /var/lib/kubelet/config.json
   ```

   输**出示例**

   ```
   {"auths":{"brew.registry.redhat.io":{"xx=="},"brewregistry.stage.redhat.io":
   {"auth":"xxx=="},"mirror.registry.com:443":{"auth":"xx="}}} (1)
   ```

   (1) 确保存在镜像 registry 和凭证。

   d. 进入 **certs.d** 目录

   ```
   sh-4.4# cd /etc/docker/certs.d/
   ```

   e. 列出 **certs.d** 目录中的证书：

   ```
   sh-4.4# ls
   ```

   输**出示例**

   ```
   image-registry.openshift-image-registry.svc.cluster.local:5000
   image-registry.openshift-image-registry.svc:5000
   mirror.registry.com:443 (1)
   ```

   (1) 确保镜像 registry 位于列表中。

   f. 检查 ICSP 是否将镜像 registry 添加到 **registry.conf** 文件中：

   ```
   sh-4.4# cat /etc/containers/registries.conf
   ```

   输**出示例**

   ```
   unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
   ```

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-release"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"

[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-v4.0-art-dev"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"
```

registry.mirror 参数表示在原始 registry 前搜索镜像 registry。

g. 退出节点。

```
sh-4.4# exit
```

15.6. 确保应用程序继续工作

在断开集群与网络的连接前，请确保集群按预期运行，并且所有应用程序都按预期工作。

流程

使用以下命令检查集群的状态：

- 确定您的 pod 正在运行：

```
$ oc get pods --all-namespaces
```

输**出示例**

```
NAMESPACE                                       NAME                                                READY
STATUS     RESTARTS   AGE
kube-system                                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-0
1/1     Running     0          39m
kube-system                                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-1
1/1     Running     0          39m
kube-system                                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-2
1/1     Running     0          39m
openshift-apiserver-operator                    openshift-apiserver-operator-79c7c646fd-5rvr5
1/1     Running     3          45m
openshift-apiserver                             apiserver-b944c4645-q694g                           2/2
Running     0          29m
openshift-apiserver                             apiserver-b944c4645-shdxb                           2/2
Running     0          31m
openshift-apiserver                             apiserver-b944c4645-x7rf2                           2/2
Running     0          33m
...
```

- 确定您的节点处于 READY 状态：

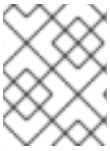
```
$ oc get nodes
```

输出示例

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-47ltxtb-f76d1-mrffg-master-0       Ready    master   42m   v1.28.5
ci-ln-47ltxtb-f76d1-mrffg-master-1       Ready    master   42m   v1.28.5
ci-ln-47ltxtb-f76d1-mrffg-master-2       Ready    master   42m   v1.28.5
ci-ln-47ltxtb-f76d1-mrffg-worker-a-gsxbz   Ready    worker   35m   v1.28.5
ci-ln-47ltxtb-f76d1-mrffg-worker-b-5qqdx   Ready    worker   35m   v1.28.5
ci-ln-47ltxtb-f76d1-mrffg-worker-c-rjkpq   Ready    worker   34m   v1.28.5
```

15.7. 断开集群与网络的连接

镜像所有所需的软件仓库并配置集群以作为断开连接的集群后，您可以断开集群与网络的连接。



注意

当集群丢失了互联网连接时，Insights Operator 会降级。您可以通过临时禁用 Insights Operator 来避免这个问题，直到您可以恢复它。

15.8. 恢复降级的 INSIGHTS OPERATOR

从网络断开集群的连接会导致集群丢失互联网连接。Insights Operator 会降级，因为它需要访问 Red Hat Insights。

本节描述了如何从降级的 Insights Operator 恢复。

流程

1. 编辑 **.dockerconfigjson** 文件以删除 **cloud.openshift.com** 条目，例如：

```
"cloud.openshift.com":{"auth":"<hash>","email":"user@example.com"}
```

2. 保存该文件。

3. 使用编辑的 **.dockerconfigjson** 文件更新集群 secret：

```
$ oc set data secret/pull-secret -n openshift-config --from-
file=.dockerconfigjson=./.dockerconfigjson
```

4. 验证 Insights Operator 不再降级：

```
$ oc get co insights
```

输出示例

```
NAME       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
insights   4.5.41    True        False         False      3d
```

15.9. 恢复网络

如果要重新连接断开连接的集群并从在线 registry 中拉取镜像，请删除集群的 ImageContentSourcePolicy(ICSP)对象。如果没有 ICSP，对外部 registry 的拉取请求不再重定向到镜像 registry。

流程

1. 查看集群中的 ICSP 对象：

```
$ oc get imagecontentsourcepolicy
```

输**出示例**

```
NAME              AGE
mirror-ocp        6d20h
ocp4-index-0      6d18h
qe45-index-0      6d15h
```

2. 删除断开集群时创建的所有 ICSP 对象：

```
$ oc delete imagecontentsourcepolicy <icsp_name> <icsp_name> <icsp_name>
```

例如：

```
$ oc delete imagecontentsourcepolicy mirror-ocp ocp4-index-0 qe45-index-0
```

输**出示例**

```
imagecontentsourcepolicy.operator.openshift.io "mirror-ocp" deleted
imagecontentsourcepolicy.operator.openshift.io "ocp4-index-0" deleted
imagecontentsourcepolicy.operator.openshift.io "qe45-index-0" deleted
```

3. 等待所有节点重启并返回到 READY 状态，并验证 **registry.conf** 文件是否指向原始 registry，而不是镜像 registry：

   a. 登录到节点：

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

   b. 将 **/host** 设置为 debug shell 中的根目录：

   ```
   sh-4.4# chroot /host
   ```

   c. 检查 **registry.conf** 文件：

   ```
   sh-4.4# cat /etc/containers/registries.conf
   ```

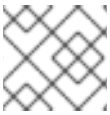
   输**出示例**

   ```
   unqualified-search-registries = ["registry.access.redhat.com", "docker.io"] 1
   ```

1 您删除的、由 ICSP 创建的 **registry** 和 **registry.mirror** 条目已被删除。

第 16 章 启用集群功能

集群管理员可以启用在安装前禁用的集群功能。



注意

集群管理员无法在启用集群后禁用集群功能。

16.1. 查看集群功能

作为集群管理员，您可以使用 **clusterversion** 资源状态来查看功能。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

- 要查看集群功能的状态，请运行以下命令：

```
$ oc get clusterversion version -o jsonpath='{.spec.capabilities}{"\n"}{.status.capabilities}{"\n"}'
```

输**出示例**

```
{"additionalEnabledCapabilities":["openshift-samples"],"baselineCapabilitySet":"None"}
{"enabledCapabilities":["openshift-samples"],"knownCapabilities":
["CSISnapshot","Console","Insights","Storage","baremetal","marketplace","openshift-
samples"]}
```

16.2. 通过设置基准功能集启用集群功能

作为集群管理员，您可以通过设置 **baselineCapabilitySet** 来启用功能。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

- 要设置 **baselineCapabilitySet**，请运行以下命令：

```
$ oc patch clusterversion version --type merge -p '{"spec":{"capabilities":
{"baselineCapabilitySet":"vCurrent"}}}' 1
```

1 对于 **baselineCapabilitySet**，您可以指定 **vCurrent**、**v4.15** 或 **None**。

下表描述了 **baselineCapabilitySet** 值。

表 16.1. 集群功能 baselineCapabilitySet 值描述

值	描述
vCurrent	当您要自动添加新版本中引入的新功能时，指定这个选项。
v4.11	当要为 OpenShift Container Platform 4.11 启用默认功能时指定这个选项。通过指定 **v4.11**，不会启用较新版本的 OpenShift Container Platform 中引入的功能。OpenShift Container Platform 4.11 中的默认功能是 **baremetal**、**MachineAPI**、**marketplace** 和 **openshift-samples**。
v4.12	当您要为 OpenShift Container Platform 4.12 启用默认功能时指定这个选项。通过指定 **v4.12**，不会启用较新版本的 OpenShift Container Platform 中引入的功能。OpenShift Container Platform 4.12 中的默认功能是 **baremetal**,**MachineAPI**,**marketplace**,**openshift-samples**,**Console**,**Insights**,**Storage**, 和 **CSISnapshot**。
v4.13	当要为 OpenShift Container Platform 4.13 启用默认功能时，指定这个选项。通过指定 **v4.13**，不会启用较新版本的 OpenShift Container Platform 中引入的功能。OpenShift Container Platform 4.13 中的默认功能是 **baremetal**,**MachineAPI**,**marketplace**,**openshift-samples**,**Console**,**Insights**,**Storage**,**CSISnapshot**, 和 **NodeTuning**。
v4.14	当要为 OpenShift Container Platform 4.14 启用默认功能时指定这个选项。通过指定 **v4.14**，不会启用较新版本的 OpenShift Container Platform 中引入的功能。OpenShift Container Platform 4.14 中的默认功能是 **baremetal**,**MachineAPI**,**marketplace**,**openshift-samples**,**Console**,**Insights**,**Storage**,**CSISnapshot**,**NodeTuning**,**ImageRegistry**,**Build**, 和 **DeploymentConfig**。
v4.15	当要为 OpenShift Container Platform 4.15 启用默认功能时指定这个选项。通过指定 **v4.15**，不会启用较新版本的 OpenShift Container Platform 中引入的功能。OpenShift Container Platform 4.15 中的默认功能是 **baremetal**,**MachineAPI**,**marketplace**,**OperatorLifecycleManager**,**openshift-samples**,**Console**,**Insights**,**Storage**,**CSISnapshot**,**NodeTuning**,**ImageRegistry**,**Build**,**CloudCredential**, 和 **DeploymentConfig**。
None	指定其他集合太大，您不需要任何功能，或者想要 **通过额外的 EnabledCapabilities** 进行微调。

16.3. 通过设置其他启用的功能来启用集群功能

作为集群管理员，您可以通过设置 **additionalEnabledCapabilities** 来启用集群功能。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

1. 运行以下命令查看附加启用的功能：

```
$ oc get clusterversion version -o jsonpath='{.spec.capabilities.additionalEnabledCapabilities}
{"\n"}'
```

输**出示例**

```
["openshift-samples"]
```

2. 要设置 **additionalEnabledCapabilities**，请运行以下命令：

```
$ oc patch clusterversion/version --type merge -p '{"spec":{"capabilities":
{"additionalEnabledCapabilities":["openshift-samples", "marketplace"]}}}'
```

重要

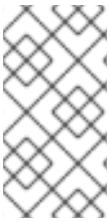
无法禁用集群中已经启用的功能。集群版本 Operator (CVO) 继续协调集群中已经启用的功能。

如果您尝试禁用某个功能，则 CVO 会显示相关的 spec:

```
$ oc get clusterversion version -o jsonpath='{.status.conditions[?
(@.type=="ImplicitlyEnabledCapabilities")]}{"\n"}'
```

输**出示例**

```
{"lastTransitionTime":"2022-07-22T03:14:35Z","message":"The following capabilities could not be
disabled: openshift-
samples","reason":"CapabilitiesImplicitlyEnabled","status":"True","type":"ImplicitlyEnabledCapabilities"}
```



注意

在集群升级过程中，可以隐式启用给定功能。如果在升级前已在集群上运行资源，那么将启用属于资源的任何功能。例如，在集群升级过程中，已在集群中运行的资源已更改为系统已作为 **marketplace** 功能的一部分。即使集群管理员没有明确启用了 **marketplace** 功能，它也会被系统隐式启用。

16.4. 其他资源

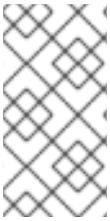
- 集群功能

第 17 章 在 IBM Z 或 IBM LINUXONE 环境中配置附加设备

安装 OpenShift Container Platform 后，您可以在使用 z/VM 安装的 IBM Z® 或 IBM® LinuxONE 环境中为集群配置额外的设备。可以配置以下设备：

- 光纤通道协议 (FCP) 主机
- FCP LUN
- DASD
- qeth

您可以使用 Machine Config Operator (MCO) 添加 udev 规则来配置设备，也可以手动配置设备。



注意

此处描述的步骤只适用于 z/VM 安装。如果您在 IBM Z® 或 IBM® LinuxONE 基础架构中使用 RHEL KVM 安装集群，在设备添加到 KVM 客户端后，在 KVM 客户端中不需要额外的配置。但是，在 z/VM 和 RHEL KVM 环境中，需要应用以下步骤来配置 Local Storage Operator 和 Kubernetes NMState Operator。

其他资源

- 安装后机器配置任务

17.1. 使用 MACHINE CONFIG OPERATOR (MCO) 配置附加设备

本节中的任务描述了如何使用 Machine Config Operator (MCO) 在 IBM Z® 或 IBM® LinuxONE 环境中配置附加设备的功能。使用 MCO 配置设备是持久性的，但只允许计算节点的特定配置。MCO 不允许 control plane 节点有不同的配置。

先决条件

- 以具有管理特权的用户身份登录集群。
- 该设备必须可供 z/VM 客户机使用。
- 该设备已经附加。
- 该设备不包含在 **cio_ignore** 列表中，可在内核参数中设置。
- 已使用以下 YAML 创建 **MachineConfig** 对象文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker0
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker0]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker0: ""
```

17.1.1. 配置光纤通道协议 (FCP) 主机

以下是如何通过添加 udev 规则来使用 N_Port 标识符虚拟化 (NPIV) 配置 FCP 主机适配器的示例。

流程

1. 取以下 udev 规则 **441-zfcp-host-0.0.8000.rules** 示例：

```
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.8000", DRIVER=="zfcp",
GOTO="cfg_zfcp_host_0.0.8000"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="zfcp", TEST=="[ccw/0.0.8000]",
GOTO="cfg_zfcp_host_0.0.8000"
GOTO="end_zfcp_host_0.0.8000"

LABEL="cfg_zfcp_host_0.0.8000"
ATTR{[ccw/0.0.8000]online}="1"

LABEL="end_zfcp_host_0.0.8000"
```

2. 运行以下命令，将规则转换为 Base64 编码：

```
$ base64 /path/to/file/
```

3. 将以下 MCO 示例配置集复制到 YAML 文件中：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-zfcp-host-0.0.8000.rules (3)
```

(1) 您在机器配置文件中定义的角色。

(2) 您在上一步中生成的 Base64 编码字符串。

(3) udev 规则所在的路径。

17.1.2. 配置 FCP LUN

以下是如何通过添加 udev 规则来配置 FCP LUN 的示例。您可以添加新的 FCP LUN，或为已使用多路径配置的 LUN 添加附加路径。

流程

1. 以下 udev 规则示例 **41-zfcp-lun-0.0.8000:0x500507680d760026:0x00bc000000000000.rules**:

```
ACTION=="add", SUBSYSTEMS=="ccw", KERNELS=="0.0.8000",
GOTO="start_zfcp_lun_0.0.8207"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="start_zfcp_lun_0.0.8000"
SUBSYSTEM=="fc_remote_ports", ATTR{port_name}=="0x500507680d760026",
GOTO="cfg_fc_0.0.8000_0x500507680d760026"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="cfg_fc_0.0.8000_0x500507680d760026"
ATTR{[ccw/0.0.8000]0x500507680d760026/unit_add}="0x00bc000000000000"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="end_zfcp_lun_0.0.8000"
```

2. 运行以下命令，将规则转换为 Base64 编码：

```
$ base64 /path/to/file/
```

3. 将以下 MCO 示例配置集复制到 YAML 文件中：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-zfcp-lun-
0.0.8000:0x500507680d760026:0x00bc000000000000.rules (3)
```

(1) 您在机器配置文件中定义的角色。

(2) 您在上一步中生成的 Base64 编码字符串。

(3) udev 规则所在的路径。

17.1.3. 配置 DASD

以下是如何通过添加 udev 规则来配置 DASD 设备的示例。

流程

1. 以下 udev 规则 **41-dasd-eckd-0.0.4444.rules** 示例：

```
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.4444", DRIVER=="dasd-eckd",
GOTO="cfg_dasd_eckd_0.0.4444"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="dasd-eckd", TEST=="
[ccw/0.0.4444]", GOTO="cfg_dasd_eckd_0.0.4444"
GOTO="end_dasd_eckd_0.0.4444"

LABEL="cfg_dasd_eckd_0.0.4444"
ATTR{[ccw/0.0.4444]online}="1"

LABEL="end_dasd_eckd_0.0.4444"
```

2. 运行以下命令，将规则转换为 Base64 编码：

```
$ base64 /path/to/file/
```

3. 将以下 MCO 示例配置集复制到 YAML 文件中：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules (3)
```

(1) 您在机器配置文件中定义的角色。

(2) 您在上一步中生成的 Base64 编码字符串。

(3) udev 规则所在的路径。

17.1.4. 配置 qeth

以下是如何通过添加 udev 规则来配置 qeth 设备的示例。

流程

1. 取以下 udev 规则 **41-qeth-0.0.1000.rules** 示例：

```
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1001", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1002", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="cfg_qeth_0.0.1000"
GOTO="end_qeth_0.0.1000"

LABEL="group_qeth_0.0.1000"
TEST=="[ccwgroup/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1001]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1002]", GOTO="end_qeth_0.0.1000"
ATTR{[drivers/ccwgroup:qeth]group}="0.0.1000,0.0.1001,0.0.1002"
GOTO="end_qeth_0.0.1000"

LABEL="cfg_qeth_0.0.1000"
ATTR{[ccwgroup/0.0.1000]online}="1"

LABEL="end_qeth_0.0.1000"
```

2. 运行以下命令，将规则转换为 Base64 编码：

```
$ base64 /path/to/file/
```

3. 将以下 MCO 示例配置集复制到 YAML 文件中：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules (3)
```

(1) 您在机器配置文件中定义的角色。

(2) 您在上一步中生成的 Base64 编码字符串。

(3) udev 规则所在的路径。

后续步骤

- 安装和配置 Local Storage Operator
- 更新节点网络配置

17.2. 手动配置附加设备

本节中的任务描述了如何在 IBM Z® 或 IBM® LinuxONE 环境中手动配置附加设备。此配置方法在节点重启后保留，但不是 OpenShift Container Platform 原生的，如果您替换该节点，则需要重新执行这些步骤。

先决条件

- 以具有管理特权的用户身份登录集群。
- 该设备必须可供节点使用。
- 在 z/VM 环境中，设备必须附加到 z/VM 客户机。

流程

1. 运行以下命令，通过 SSH 连接到节点：

```
$ ssh <user>@<node_ip_address>
```

   您还可以运行以下命令为节点启动 debug 会话：

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

2. 要使用 **chzdev** 命令启用设备，请输入以下命令：

```
$ sudo chzdev -e <device>
```

其他资源

- chzdev - 配置 IBM Z® 设备 (IBM® 文档)
- 持久设备配置 (IBM® 文档)

17.3. ROCE 网卡

不需要启用 RoCE （通过融合以太网）网卡，并在节点上可用时使用 Kubernetes NMState Operator 配置其接口。例如，如果 RoCE 网卡在 z/VM 环境中附加，或者在 RHEL KVM 环境中通过。

17.4. 为 FCP LUN 启用多路径

本节中的任务描述了如何在 IBM Z® 或 IBM® LinuxONE 环境中手动配置附加设备。此配置方法在节点重启后保留，但不是 OpenShift Container Platform 原生的，如果您替换该节点，则需要重新执行这些步骤。

重要

在 IBM Z® 和 IBM® LinuxONE 中，您只能在在安装过程中为它配置集群时启用多路径。如需更多信息，请参阅在 *IBM Z® 和 IBM® LinuxONE 上安装使用 z/VM 的集群*"安装 RHCOS 并启动 OpenShift Container Platform bootstrap 过程"。

先决条件

- 以具有管理特权的用户身份登录集群。
- 您已配置了多个 LUN 路径，它带有上述任一方法。

流程

1. 运行以下命令，通过 SSH 连接到节点：

```
$ ssh <user>@<node_ip_address>
```

   您还可以运行以下命令为节点启动 debug 会话：

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

2. 要启用多路径，请运行以下命令：

```
$ sudo /sbin/mpathconf --enable
```

3. 要启动 **multipathd** 守护进程，请运行以下命令：

```
$ sudo multipath
```

4. 可选： 要使用 fdisk 格式化多路径设备，请运行以下命令：

```
$ sudo fdisk /dev/mapper/mpatha
```

验证

- 要验证设备是否已分组，请运行以下命令：

```
$ sudo multipath -ll
```

   输**出示例**

```
mpatha (20017380030290197) dm-1 IBM,2810XIV
  size=512G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
 -+- policy='service-time 0' prio=50 status=enabled
  |- 1:0:0:6  sde 68:16  active ready running
  |- 1:0:1:6  sdf 69:24  active ready running
  |- 0:0:0:6  sdg  8:80  active ready running
  `- 0:0:1:6  sdh 66:48  active ready running
```

后续步骤

- 安装和配置 Local Storage Operator
- 更新节点网络配置

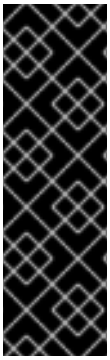
第 18 章 VSPHERE 上集群的多个区域和区域配置

作为管理员，您可以为在 VMware vSphere 实例上运行的 OpenShift Container Platform 集群指定多个区域和区域。此配置降低了硬件故障或网络中断的风险，从而导致集群失败。

故障域配置列出了创建拓扑的参数。以下列表指出其中一些参数：

- **computeCluster**
- **datacenter**
- **datastore**
- **networks**
- **resourcePool**

为 OpenShift Container Platform 集群定义了多个区域和区域后，您可以创建或将节点迁移到另一个故障域。



重要

如果要将预先存在的 OpenShift Container Platform 集群计算节点迁移到故障域，您必须为计算节点定义一个新的计算机器集。此新机器集可以根据故障域的拓扑扩展计算节点，并缩减预先存在的计算节点。

云供应商将 **topology.kubernetes.io/zone** 和 **topology.kubernetes.io/region** 标签添加到机器集资源置备的任何计算节点。

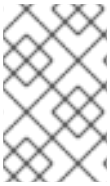
如需更多信息，请参阅创建计算机器集。

18.1. 在 VSPHERE 上为集群指定多个区域和区域

您可以配置 **infrastructure.config.openshift.io** 配置资源，为在 VMware vSphere 实例上运行的 OpenShift Container Platform 集群指定多个区域和区域。

云控制器管理器和 vSphere Container Storage Interface (CSI) Operator 驱动程序的拓扑感知功能需要有关托管 OpenShift Container Platform 集群的 vSphere 拓扑的信息。此拓扑信息存在于 **infrastructure.config.openshift.io** 配置资源中。

在为集群指定区域和区域前，您必须确保所有数据中心和计算集群都包含标签，以便云供应商可以向节点添加标签。例如，如果 **datacenter-1** 代表 **region-a**，**compute-cluster-1** 代表 **zone-1**，则云供应商会将值为 **region-a** 的 **openshift-region** 目录标签添加到 **datacenter-1**。另外，云供应商将值为 **zone-1** 的 **openshift-zone** 目录标签添加到 **compute-cluster-1**。



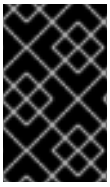
注意

您可以将具有 vMotion 功能的 control plane 节点迁移到故障域。将这些节点添加到故障域后，云供应商会为这些节点添加 **topology.kubernetes.io/zone** 和 **topology.kubernetes.io/region** 标签。

先决条件

- 您在 vCenter 服务器上创建了 **openshift-region** 和 **openshift-zone** 标签类别。

- 您可以确保每个数据中心和计算集群都包含代表相关区域或区域的名称的标签，或两者。
- 可选：如果您为安装程序定义了 **API** 和 **Ingress** 静态 IP 地址，您必须确保所有区域和区共享一个通用的第 2 层网络。此配置可确保 API 和 Ingress 虚拟 IP (VIP) 地址可以与集群交互。



重要

如果您在创建节点或迁移节点前没有向所有数据中心和计算机器提供标签，则云供应商无法将 **topology.kubernetes.io/zone** 和 **topology.kubernetes.io/region** 标签添加到节点。这意味着服务无法将流量路由到节点。

流程

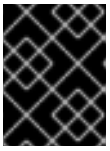
1. 运行以下命令，编辑集群的 **infrastructure.config.openshift.io** 自定义资源定义(CRD)，以指定资源的 **failureDomains** 部分中的多个区域和区域：

```
$ oc edit infrastructures.config.openshift.io cluster
```

名为 cluster 的实例的 infrastructure.config.openshift.io CRD 示例，其配置中定义了多个区域和区域

```
spec:
  cloudConfig:
    key: config
    name: cloud-provider-config
  platformSpec:
    type: vSphere
    vsphere:
      vcenters:
        - datacenters:
            - <region_a_datacenter>
            - <region_b_datacenter>
          port: 443
          server: <your_vcenter_server>
      failureDomains:
        - name: <failure_domain_1>
          region: <region_a>
          zone: <zone_a>
          server: <your_vcenter_server>
          topology:
            datacenter: <region_a_dc>
            computeCluster: "</region_a_dc/host/zone_a_cluster>"
            resourcePool: "</region_a_dc/host/zone_a_cluster/Resources/resource_pool>"
            datastore: "</region_a_dc/datastore/datastore_a>"
            networks:
            - port-group
        - name: <failure_domain_2>
          region: <region_a>
          zone: <zone_b>
          server: <your_vcenter_server>
          topology:
            computeCluster: </region_a_dc/host/zone_b_cluster>
            datacenter: <region_a_dc>
            datastore: </region_a_dc/datastore/datastore_a>
            networks:
```

```
      - port-group
    - name: <failure_domain_3>
      region: <region_b>
      zone: <zone_a>
      server: <your_vcenter_server>
      topology:
        computeCluster: </region_b_dc/host/zone_a_cluster>
        datacenter: <region_b_dc>
        datastore: </region_b_dc/datastore/datastore_b>
        networks:
        - port-group
  nodeNetworking:
    external: {}
    internal: {}
```



重要

创建故障域并在 VMware vSphere 集群的 CRD 中定义后，不得修改或删除故障域。对此配置执行任何操作可能会影响 control plane 机器的可用性和容错。

2. 保存资源文件以应用更改。

其他资源

- 集群范围的基础架构 CRD 的参数

18.2. 为集群启用多个第 2 层网络

您可以将集群配置为使用多个第 2 层网络配置，以便节点间的数据传输可以跨越多个网络。

先决条件

- 您在机器间配置了网络连接，以便集群组件可以相互通信。

流程

- 如果使用安装程序置备的基础架构安装集群，您必须确保所有 control plane 节点共享一个通用的第 2 层网络。另外，确保为 Ingress pod 调度配置的计算节点共享一个通用的第 2 层网络。
  - 如果需要计算节点跨越多个第 2 层网络，您可以创建可托管 Ingress pod 的基础架构节点。
  - 如果您需要在额外第 2 层网络间置备工作负载，您可以在 vSphere 上创建计算机器集，然后将这些工作负载移到目标第 2 层网络。
- 如果在您提供的基础架构上安装集群（定义为用户置备的基础架构），请完成以下步骤以满足您的需要：
  - 配置 API 负载均衡器和网络，以便负载均衡器可以访问 control plane 节点上的 API 和 Machine Config Server。
  - 配置 Ingress 负载均衡器和网络，以便负载均衡器可以访问计算或基础架构节点上的 Ingress pod。

其他资源

- 网络连接要求
- 为生产环境创建基础架构机器集
- 创建计算机器集

18.3. 集群范围的基础架构 CRD 的参数

您必须在集群范围的基础架构、infrastructure. **config.openshift.io**、自定义资源定义(CRD) 中为特定参数设置值，以定义在 VMware vSphere 实例上运行的 OpenShift Container Platform 集群的多个区域和区域。

下表列出了为 OpenShift Container Platform 集群定义多个区域和区域的必要参数：

参数	描述
vcenters	OpenShift Container Platform 集群的 vCenter 服务器。您只能为集群指定一个 vCenter。
datacenters	与 OpenShift Container Platform 集群关联的 vCenter 数据中心。
port	vCenter 服务器的 TCP 端口。
server	vCenter 服务器的完全限定域名 (FQDN)。
failureDomains	故障域列表。
name	故障域的名称。
region	分配给故障域拓扑的 **openshift-region** 标签的值。
zone	分配给故障域的拓扑的 **openshift-zone** 标签的值。
topology	与故障域关联的 vCenter reources。
datacenter	与故障域关联的数据中心。
computeCluster	与故障域关联的计算机器的完整路径。
resourcePool	与故障域关联的资源池的完整路径。
datastore	与故障域关联的数据存储的完整路径。
networks	与故障域关联的端口组列表。只能定义一个 portgroup。

其他资源

- 在 vSphere 上为集群指定多个区域和区域

第 19 章 RHCOS 镜像分层

Red Hat Enterprise Linux CoreOS (RHCOS) 镜像分层允许您通过将额外镜像*分层*到基础镜像来轻松地扩展基本 RHCOS 镜像的功能。此分层不会修改基本 RHCOS 镜像。相反，它会创建一个*自定义层次镜像*，其中包含所有 RHCOS 功能，并为集群中的特定节点添加额外的功能。

您可以使用 Containerfile 创建自定义分层镜像，并使用 **MachineConfig** 对象将其应用到节点。Machine Config Operator 覆盖基础 RHCOS 镜像，由关联的机器配置中的 **osImageURL** 值指定，并引导新镜像。您可以通过删除机器配置来删除自定义层次镜像，MCO 会将节点重启回基础 RHCOS 镜像。

使用 RHCOS 镜像分层时，您可以在基础镜像中安装 RPM，自定义内容将与 RHCOS 一起引导。Machine Config Operator (MCO) 可以推出这些自定义分层镜像，并像默认 RHCOS 镜像一样监控这些自定义容器。RHCOS 镜像分层为管理 RHCOS 节点的方式提供了更大的灵活性。

重要

不建议将实时内核和扩展 RPM 作为自定义分层内容安装。这是因为这些 RPM 可能会与使用机器配置安装的 RPM 冲突。如果存在冲突，MCO 会在尝试安装机器配置 RPM 时进入 **degraded** 状态。在继续操作前，您需要从机器配置中删除冲突扩展。

将自定义分层镜像应用到集群后，您可以有效地 *获取自定义分层镜像和这些节点的所有权*。虽然红帽仍负责维护和更新标准节点上的基础 RHCOS 镜像，但在使用自定义分层镜像的节点中维护和更新镜像。假定您对自定义分层镜像应用的软件包及软件包可能出现的问题负责。

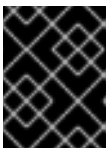
要应用自定义分层镜像，您可以创建一个 Containerfile 来引用 OpenShift Container Platform 镜像和您要应用的 RPM。然后，您将生成的自定义分层镜像推送到镜像 registry。在非生产环境的 OpenShift Container Platform 集群中，为指向新镜像的目标节点池创建一个 **MachineConfig** 对象。

注意

使用在集群的其余部分上安装的同一基本 RHCOS 镜像。使用 **oc adm release info --image-for rhel-coreos** 命令获取集群中使用的基础镜像。

RHCOS 镜像分层允许您使用以下类型的镜像来创建自定义分层镜像：

- **OpenShift 容器平台 Hotfixes.**您可以使用客户体验和参与(CEE)在 RHCOS 镜像之上获取并应用 Hotfix 软件包。在某些情况下，您可能需要在一个官方的 OpenShift Container Platform 发行版本中包括程序错误修复或功能增强。RHCOS 镜像分层允许您在正式发布前轻松添加 Hotfix，并在底层 RHCOS 镜像包含修复时删除 Hotfix。

  重要

  有些 Hotfixes 需要红帽支持例外，且不在 OpenShift Container Platform 支持覆盖范围或生命周期政策之外。

  如果您需要热修复，它将遵循红帽 Hotfix 策略提供给您。将它应用到基础镜像的顶部，并测试在非生产环境中的新的自定义分层镜像。当您满足自定义分层镜像在生产环境中安全使用时，您可以将其按您自己的计划部署到特定的节点池。因此，您可以轻松地回滚自定义分层镜像，并使用默认 RHCOS 返回。

  应用 Hotfix 的 Containerfile 示例

  ```
  # Using a 4.12.0 image
  ```

```
FROM quay.io/openshift-release-dev/ocp-release@sha256...
#Install hotfix rpm
RUN rpm-ostree override replace https://example.com/myrepo/haproxy-1.0.16-5.el8.src.rpm
&& \
    rpm-ostree cleanup -m && \
    ostree container commit
```

- **RHEL 软件包**.您可以从红帽客户门户网站下载 Red Hat Enterprise Linux (RHEL) 软件包，如 chrony, firewalld, and iputils。

  应用 firewalld 工具的 Containerfile 示例

```
FROM quay.io/openshift-release-dev/ocp-release@sha256...
ADD configure-firewall-playbook.yml .
RUN rpm-ostree install firewalld ansible && \
    ansible-playbook configure-firewall-playbook.yml && \
    rpm -e ansible && \
    ostree container commit
```

  应用 libreswan 工具的 Containerfile 示例

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

  因为 libreswan 需要额外的 RHEL 软件包，所以必须在授权的 RHEL 主机上构建镜像。

- **第三方软件包**。您可以从第三方机构下载并安装 RPM，如以下类型的软件包：
  - 增强边缘驱动程序和内核增强，以提高性能或添加功能。
  - 用于调查可能和实际分类的客户端工具。
  - 安全代理。
  - 提供整个集群一致的视图的清单代理。
  - SSH 密钥管理软件包。

  从 EPEL 应用第三方软件包的 Containerfile 示例

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
```

```
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

应用具有 RHEL 依赖项的第三方软件包的 Containerfile 示例

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

这个 Containerfile 安装 Linux fish 程序。由于 fish 需要额外的 RHEL 软件包，所以必须在授权的 RHEL 主机上构建镜像。

创建机器配置后，Machine Config Operator (MCO) 执行以下步骤：

1. 为指定池呈现新机器配置。
2. 对池中的节点执行 cordon 和 drain 操作。
3. 将其余机器配置参数写入节点。
4. 将自定义分层镜像应用到节点。
5. 使用新镜像重启节点。

重要

强烈建议您在推出集群前测试生产环境中的镜像。

19.1. 应用 RHCOS 自定义层次镜像

您可以在特定机器配置池中的节点上轻松配置 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像层。Machine Config Operator (MCO) 使用新的自定义分层镜像重启这些节点，覆盖基本 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像。

要将自定义分层镜像应用到集群，您必须在集群可访问的存储库中具有自定义层次镜像。然后，创建一个指向自定义分层镜像的 **MachineConfig** 对象。对于每个需要配置的集群配置池，都需要一个独立的 **MachineConfig** 对象。

重要

当您配置自定义分层镜像时，OpenShift Container Platform 不再自动更新任何使用自定义分层镜像的节点。根据需要手动进行节点更新是您自己的责任。如果您回滚自定义层，OpenShift Container Platform 将再次自动更新该节点。有关更新使用自定义分层镜像的节点的重要信息，请参阅以下附加资源部分。

先决条件

- 您必须创建一个基于 OpenShift Container Platform 镜像摘要的自定义层次镜像，而不是标签。

注意

您应该使用与集群的其余部分上安装相同的基本 RHCOS 镜像。使用 **oc adm release info --image-for rhel-coreos** 命令获取集群中使用的基础镜像。

例如，以下 Containerfile 从 OpenShift Container Platform 4.15 镜像创建一个自定义分层镜像，并使用 CentOS 9 Stream 之一覆盖内核软件包：

自定义层镜像的 Containerfile 示例

```
# Using a 4.15.0 image
FROM quay.io/openshift-release/ocp-release@sha256... 1
#Install hotfix rpm
RUN rpm-ostree cliwrap install-to-root / && \ 2
    rpm-ostree override replace http://mirror.stream.centos.org/9-
stream/BaseOS/x86_64/os/Packages/kernel-{,core-,modules-,modules-core-,modules-extra-
}5.14.0-295.el9.x86_64.rpm && \ 3
    rpm-ostree cleanup -m && \
    ostree container commit
```

1. 指定集群的 RHCOS 基础镜像。
2. 启用 **cliwrap**。目前需要截获从内核脚本进行的一些命令调用。
3. 替换内核软件包。

注意

有关如何创建 Containerfile 的说明超出了本文档的范围。

- 由于构建自定义分层镜像的过程是在集群之外执行，所以您必须在 Podman 或 Buildah 中使用 **--authfile /path/to/pull-secret** 选项。或者，要自动读取这些工具的 pull secret，您可以将其添加到默认文件位置之一：**~/.docker/config.json**, **$XDG_RUNTIME_DIR/containers/auth.json**, **~/.docker/config.json**, 或 **~/.dockercfg**。如需更多信息，请参阅 **containers-auth.json** 手册页。
- 您必须将自定义分层镜像推送到集群可访问的存储库。

流程

1. 创建机器配置文件。

a. 创建一个类似以下示例的 YAML 文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: os-layer-custom
spec:
  osImageURL: quay.io/my-registry/custom-image@sha256... (2)
```

1. 指定要应用自定义分层镜像的机器配置池。
2. 指定存储库中自定义分层镜像的路径。

b. 创建 **MachineConfig** 对象：

```
$ oc create -f <file_name>.yaml
```

重要

强烈建议您在推出集群前测试生产环境中的镜像。

验证

您可以通过执行以下任一方式来验证是否应用了自定义层次镜像：

1. 检查 worker 机器配置池已使用新机器配置推出：

a. 检查是否创建了新机器配置：

```
$ oc get mc
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
00-worker                                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
01-master-container-runtime
5bdb57489b720096ef912f738b46330a8f577803   3.2.0             95m
01-master-kubelet                                  5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
01-worker-container-runtime
5bdb57489b720096ef912f738b46330a8f577803   3.2.0             95m
01-worker-kubelet                                  5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
99-master-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0             95m
99-master-ssh                                                3.2.0             98m
99-worker-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0             95m
```

```
99-worker-ssh                                              3.2.0        98m
os-layer-custom                                                         10s (1)
rendered-master-15961f1da260f7be141006404d17d39b
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5aff604cb1381a4fe07feaf1595a797e
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5de4837625b1cbc237de6b22bc0bc873
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          4s (2)
```

1. 新机器配置
2. 新的渲染机器配置

b. 检查新机器配置中的 **osImageURL** 值是否指向预期的镜像：

```
$ oc describe mc rendered-master-4e8be63aef68b843b546827b6ebe0913
```

输**出示例**

```
Name:         rendered-master-4e8be63aef68b843b546827b6ebe0913
Namespace:
Labels:       <none>
Annotations:  machineconfiguration.openshift.io/generated-by-controller-version:
8276d9c1f574481043d3661a1ace1f36cd8c3b62
              machineconfiguration.openshift.io/release-image-version: 4.15.0-ec.3
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfig
...
  Os Image URL: quay.io/my-registry/custom-image@sha256...
```

c. 检查关联的机器配置池是否使用新机器配置更新：

```
$ oc get mcp
```

输**出示例**

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-6faecdfa1b25c114a58cf178fbaa45e2   True      False      False
3              3                   3                   0                      39m
worker   rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b   False     True
False    3              0                   0                   0                      39m (1)
```

1. 当 **UPDATING** 字段为 **True** 时，机器配置池会使用新机器配置进行更新。当字段变为 **False** 时，代表 worker 机器配置池已应用到新机器配置。

d. 检查节点以查看是否禁用了在节点上调度。这表示要应用更改：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE
VERSION
ip-10-0-148-79.us-west-1.compute.internal    Ready                      worker                 32m
v1.28.5
ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
35m   v1.28.5
ip-10-0-170-47.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.28.5
ip-10-0-174-77.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.28.5
ip-10-0-211-49.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.28.5
ip-10-0-218-151.us-west-1.compute.internal   Ready                      worker                 31m
v1.28.5
```

2. 当节点重新处于 **Ready** 状态时，检查该节点是否使用自定义分层镜像：

   a. 打开节点的 **oc debug** 会话。例如：

   ```
   $ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
   ```

   b. 将 **/host** 设置为 debug shell 中的根目录：

   ```
   sh-4.4# chroot /host
   ```

   c. 运行 **rpm-ostree status** 命令，以查看自定义分层镜像正在使用：

   ```
   sh-4.4# sudo rpm-ostree status
   ```

   输出示例

   ```
   State: idle
   Deployments:
   * ostree-unverified-registry:quay.io/my-registry/...
                      Digest: sha256:...
   ```

其他资源

使用 RHCOS 自定义分层镜像进行更新

19.2. 删除 RHCOS 自定义层次镜像

您可以从特定机器配置池中的节点轻松恢复 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像层。Machine Config Operator (MCO) 使用集群基本 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像重启这些节点，覆盖自定义分层镜像。

要从集群中删除 Red Hat Enterprise Linux CoreOS (RHCOS) 自定义分层镜像，您需要删除应用镜像的机器配置。

流程

1. 删除应用自定义分层镜像的机器配置。

```
$ oc delete mc os-layer-custom
```

删除机器配置后，节点将重新引导。

验证

您可以通过执行以下任一方式来验证自定义层次镜像是否已被删除：

1. 检查 worker 机器配置池是否使用以前的机器配置更新：

```
$ oc get mcp
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-6faecdfa1b25c114a58cf178fbaa45e2   True      False      False
3              3                   3                   0                      39m
worker   rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b   False     True       False
3              0                   0                   0                      39m   1
```

1 当 **UPDATING** 字段为 **True** 时，机器配置池会使用以前的机器配置进行更新。当字段变为 **False** 时，worker 机器配置池已应用到以前的机器配置。

2. 检查节点以查看是否禁用了在节点上调度。这表示要应用更改：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-148-79.us-west-1.compute.internal    Ready                      worker                 32m
v1.28.5
ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
35m   v1.28.5
ip-10-0-170-47.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.28.5
ip-10-0-174-77.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.28.5
ip-10-0-211-49.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.28.5
ip-10-0-218-151.us-west-1.compute.internal   Ready                      worker                 31m
v1.28.5
```

3. 当节点重新处于 **Ready** 状态时，检查该节点是否使用基础镜像：

   a. 打开节点的 **oc debug** 会话。例如：

```
$ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
```

b. 将 **/host** 设置为 debug shell 中的根目录：

```
sh-4.4# chroot /host
```

c. 运行 **rpm-ostree status** 命令，以查看自定义分层镜像正在使用：

```
sh-4.4# sudo rpm-ostree status
```

输出示例

```
State: idle
Deployments:
* ostree-unverified-registry:podman pull quay.io/openshift-release-dev/ocp-
release@sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd
73
                   Digest:
sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd73
```

19.3. 使用 RHCOS 自定义分层镜像进行更新

当您配置 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像分层时，OpenShift Container Platform 不再自动更新使用自定义分层镜像的节点池。您负责根据需要手动更新节点。

要更新使用自定义分层镜像的节点，请按照以下步骤操作：

1. 集群会自动升级到 x.y.z+1 版本，但使用自定义分层镜像的节点除外。
2. 然后，您可以创建一个引用更新的 OpenShift Container Platform 镜像和之前应用的 RPM 的新 Containerfile。
3. 创建指向更新的自定义分层镜像的新机器配置。

不需要使用自定义分层镜像更新节点。但是，如果该节点早于当前的 OpenShift Container Platform 版本太多，您可能会遇到意外的结果。

第 20 章 在现有的 NUTANIX 集群中添加故障域

默认情况下，安装程序会将 control plane 和计算机器安装到单个 Nutanix Prism Element (集群) 中。部署 OpenShift Container Platform 集群后，您可以使用故障域在部署中添加额外的 Prism Element 实例来提高其容错功能。

故障域代表单个 Prism Element 实例，它：

- 可以部署新的 control plane 和计算机器。
- 可以分发现有的 control plane 和计算机器。

20.1. 故障域要求

在计划使用故障域时，请考虑以下要求：

- 所有 Nutanix Prism Element 实例都必须由同一 Prism Central 实例管理。不支持由多个 Prism Central 实例组成的部署。
- 组成 Prism Element 集群的机器必须位于同一以太网网络中，以便故障域能够相互通信。
- 每个 Prism Element 中都需要一个子网，将用作 OpenShift Container Platform 集群中的故障域。在定义这些子网时，它们必须共享相同的 IP 地址前缀 (CIDR)，并且应包含 OpenShift Container Platform 集群使用的虚拟 IP 地址。

20.2. 在 INFRASTRUCTURE CR 中添加故障域

您可以通过修改其 Infrastructure 自定义资源 (CR) (**infrastructures.config.openshift.io**)，将故障域添加到现有 Nutanix 集群。

提示

建议您配置三个故障域以确保高可用性。

流程

1. 运行以下命令来编辑 Infrastructure CR：

```
$ oc edit infrastructures.config.openshift.io cluster
```

2. 配置故障域。

带有 Nutanix 故障域的基础架构 CR 示例

```
spec:
  cloudConfig:
    key: config
    name: cloud-provider-config
#...
  platformSpec:
    nutanix:
      failureDomains:
      - cluster:
          type: UUID
```

```
        uuid: <uuid>
      name: <failure_domain_name>
      subnets:
      - type: UUID
        uuid: <network_uuid>
    - cluster:
        type: UUID
        uuid: <uuid>
      name: <failure_domain_name>
      subnets:
      - type: UUID
        uuid: <network_uuid>
    - cluster:
        type: UUID
        uuid: <uuid>
      name: <failure_domain_name>
      subnets:
      - type: UUID
        uuid: <network_uuid>
# ...
```

其中：

<uuid>

指定 Prism Element 的通用唯一标识符 (UUID)。

<failure_domain_name>

指定故障域的唯一名称。名称的长度不能超过 64 个字符，可以包括小写字母、数字和短划线 (**-**)。短划线不能是名称的第一个或最后一个。

<network_uuid>

指定 Prism Element 子网对象的 UUID。子网的 IP 地址前缀 (CIDR) 应包含 OpenShift Container Platform 集群使用的虚拟 IP 地址。在 OpenShift Container Platform 集群中，只支持每个故障域 (Prism Element) 一个子网。

3. 保存 CR 以应用更改。

20.3. 在故障域间分布 CONTROL PLANE

您可以通过修改 control plane 机器集自定义资源 (CR) 在 Nutanix 故障域之间分发 control plane。

先决条件

- 您已在集群的 Infrastructure 自定义资源 (CR) 中配置了故障域。
- control plane 机器集自定义资源 (CR) 处于 active 状态。

有关检查 control plane 机器集自定义资源状态的更多信息，请参阅"添加资源"。

流程

1. 运行以下命令来编辑 control plane 机器集 CR：

```
$ oc edit controlplanemachineset.machine.openshift.io cluster -n openshift-machine-api
```

2. 通过添加 **spec.template.machines_v1beta1_machine_openshift_io.failureDomains** 小节，将 control plane 机器集配置为使用故障域。

使用 Nutanix 故障域的 control plane 机器集示例

```
apiVersion: machine.openshift.io/v1
kind: ControlPlaneMachineSet
  metadata:
    creationTimestamp: null
    labels:
      machine.openshift.io/cluster-api-cluster: <cluster_name>
    name: cluster
    namespace: openshift-machine-api
spec:
# ...
  template:
    machineType: machines_v1beta1_machine_openshift_io
    machines_v1beta1_machine_openshift_io:
      failureDomains:
        platform: Nutanix
        nutanix:
        - name: <failure_domain_name_1>
        - name: <failure_domain_name_2>
        - name: <failure_domain_name_3>
# ...
```

3. 保存您的更改。

默认情况下，control plane 机器集会自动将更改传播到 control plane 配置。如果集群被配置为使用 **OnDelete** 更新策略，您必须手动替换 control plane。如需更多信息，请参阅"附加资源"。

其他资源

- 检查 control plane 机器设置自定义资源状态
- 替换 control plane 机器

20.4. 在故障域间分布计算机器

您可以使用以下方法之一在 Nutanix 故障域间分发计算机器：

- 通过 编辑现有的计算机器集，您可以在 Nutanix 故障域间分发计算机器，作为最小配置更新。
- 替换现有计算机器集 可确保规格不可变，且所有机器都相同。

20.4.1. 编辑计算机器集以实施故障域

要使用现有计算机器集在 Nutanix 故障域之间分发计算机器，您可以使用您的配置更新计算机器集，然后使用扩展来替换现有的计算机器。

先决条件

- 您已在集群的 Infrastructure 自定义资源 (CR) 中配置了故障域。

流程

1. 运行以下命令，以查看集群的 Infrastructure CR。

```
$ oc describe infrastructures.config.openshift.io cluster
```

2. 对于每个故障域 (**platformSpec.nutanix.failureDomains**)，请注意集群的 UUID、名称和子网对象 UUID。这些值需要将故障域添加到计算机器集中。

3. 运行以下命令列出集群中的计算机器集：

```
$ oc get machinesets -n openshift-machine-api
```

输**出示例**

```
NAME                DESIRED   CURRENT   READY   AVAILABLE   AGE
<machine_set_name_1>  1         1         1       1           55m
<machine_set_name_2>  1         1         1       1           55m
```

4. 运行以下命令来编辑第一个计算机器集：

```
$ oc edit machineset <machine_set_name_1> -n openshift-machine-api
```

5. 通过将以下内容添加到 **spec.template.spec.providerSpec.value** 小节中，将计算机器设置配置为使用第一个故障域：

注意

确保为 **cluster** he **subnets** 字段指定的值与集群的 Infrastructure CR 中的 **failureDomains** 小节中配置的值匹配。

使用 Nutanix 故障域的计算机器集示例

```
apiVersion: machine.openshift.io/v1
kind: MachineSet
metadata:
  creationTimestamp: null
  labels:
    machine.openshift.io/cluster-api-cluster: <cluster_name>
  name: <machine_set_name_1>
  namespace: openshift-machine-api
spec:
  replicas: 2
# ...
  template:
    spec:
# ...
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1
          failureDomain:
            name: <failure_domain_name_1>
          cluster:
```

```
          type: uuid
          uuid: <prism_element_uuid_1>
        subnets:
        - type: uuid
          uuid: <prism_element_network_uuid_1>
# ...
```

6. 注意 **spec.replicas** 的值，因为在扩展计算机器时需要它来应用更改。
7. 保存您的更改。
8. 运行以下命令，列出由更新的计算机器集管理的机器：

```
$ oc get -n openshift-machine-api machines \
  -l machine.openshift.io/cluster-api-machineset=<machine_set_name_1>
```

输**出示例**

```
NAME                      PHASE     TYPE   REGION    ZONE            AGE
<machine_name_original_1>  Running   AHV    Unnamed   Development-STS   4h
<machine_name_original_2>  Running   AHV    Unnamed   Development-STS   4h
```

9. 对于由更新的计算机器集管理的每台机器，请运行以下命令设置 **delete** 注解：

```
$ oc annotate machine/<machine_name_original_1> \
  -n openshift-machine-api \
  machine.openshift.io/delete-machine="true"
```

10. 要使用新配置创建替换机器，请运行以下命令将计算机器设置为两倍：

```
$ oc scale --replicas=<twice_the_number_of_replicas> \ 1
  machineset <machine_set_name_1> \
  -n openshift-machine-api
```

1 例如，如果计算机器设置中的原始副本数为 **2**，请将副本扩展到 **4**。

11. 运行以下命令，列出由更新的计算机器集管理的机器：

```
$ oc get -n openshift-machine-api machines -l machine.openshift.io/cluster-api-machineset=
<machine_set_name_1>
```

当新机器处于 **Running** 阶段时，您可以将计算机器设置为原始副本数。

12. 要删除使用旧配置创建的机器，请运行以下命令将计算机器设置为原始副本数：

```
$ oc scale --replicas=<original_number_of_replicas> \ 1
  machineset <machine_set_name_1> \
  -n openshift-machine-api
```

1 例如，如果计算机器设置中的原始副本数为 **2**，请将副本扩展到 **2**。

13. 根据需要，继续修改机器集以引用可用于部署的额外故障域。

其他资源

- 修改计算机器集

20.4.2. 替换计算机器集以实施故障域

要通过替换计算机器集来跨 Nutanix 故障域分发计算机器，您可以使用您的配置创建新的计算机器集，等待它创建的机器启动，然后删除旧的计算机器集。

先决条件

- 您已在集群的 Infrastructure 自定义资源 (CR) 中配置了故障域。

流程

1. 运行以下命令，以查看集群的 Infrastructure CR。

```
$ oc describe infrastructures.config.openshift.io cluster
```

2. 对于每个故障域 (**platformSpec.nutanix.failureDomains**)，请注意集群的 UUID、名称和子网对象 UUID。这些值需要将故障域添加到计算机器集中。

3. 运行以下命令列出集群中的计算机器集：

```
$ oc get machinesets -n openshift-machine-api
```

输**出示例**

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<original_machine_set_name_1>   1         1         1       1           55m
<original_machine_set_name_2>   1         1         1       1           55m
```

4. 请注意现有计算机器集的名称。

5. 使用以下方法之一创建一个包含新计算机器设置自定义资源 (CR) 的 YAML 文件：

   - 运行以下命令，将现有计算机器集配置复制到新文件中：

```
$ oc get machineset <original_machine_set_name_1> \
  -n openshift-machine-api -o yaml > <new_machine_set_name_1>.yaml
```

     您可以使用首选文本编辑器编辑此 YAML 文件。

   - 使用您的首选文本编辑器创建名为 **<new_machine_set_name_1>.yaml** 的空白 YAML 文件，并包含新计算机器集所需的值。
     如果您不确定为特定字段设置哪个值，您可以通过运行以下命令来查看现有计算机器集 CR 的值：

```
$ oc get machineset <original_machine_set_name_1> \
  -n openshift-machine-api -o yaml
```

     输**出示例**

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-<role> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      providerSpec: 3
        ...
```

1. 集群基础架构 ID。
2. 默认节点标签。

注意

对于具有用户置备的基础架构的集群，计算机器集只能使用 **worker** 或 **infra** 角色创建机器。

3. 计算机器设置 CR 的 **<providerSpec>** 部分中的值是特定于平台的。有关 CR 中的 **<providerSpec>** 参数的更多信息，请参阅您的供应商计算机器设置 CR 配置示例。

6. 通过更新或将以下内容添加到 **<new_machine_set_name_1>.yaml** 文件中的 **spec.template.spec.providerSpec.value** 小节中，将新的计算机器集配置为使用第一个故障域。

注意

确保为 **cluster** he **subnets** 字段指定的值与集群的 Infrastructure CR 中的 **failureDomains** 小节中配置的值匹配。

使用 Nutanix 故障域的计算机器集示例

```
apiVersion: machine.openshift.io/v1
kind: MachineSet
metadata:
  creationTimestamp: null
  labels:
```

```
    machine.openshift.io/cluster-api-cluster: <cluster_name>
  name: <new_machine_set_name_1>
  namespace: openshift-machine-api
spec:
  replicas: 2
# ...
  template:
    spec:
# ...
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1
          failureDomain:
            name: <failure_domain_name_1>
          cluster:
            type: uuid
            uuid: <prism_element_uuid_1>
          subnets:
          - type: uuid
            uuid: <prism_element_network_uuid_1>
# ...
```

7. 保存您的更改。
8. 运行以下命令来创建计算机器设置 CR：

```
$ oc create -f <new_machine_set_name_1>.yaml
```

9. 根据需要，继续创建计算机器集以引用可用于部署的额外故障域。
10. 通过为每个新计算机器集运行以下命令来列出由新计算机器集管理的机器：

```
$ oc get -n openshift-machine-api machines -l machine.openshift.io/cluster-api-machineset=
<new_machine_set_name_1>
```

输出示例

```
NAME                        PHASE         TYPE   REGION    ZONE              AGE
<machine_from_new_1>        Provisioned   AHV    Unnamed   Development-STS   25s
<machine_from_new_2>        Provisioning  AHV    Unnamed   Development-STS   25s
```

当新机器处于 **Running** 阶段时，您可以删除不包含故障域配置的旧计算机器集。

11. 当您确认新机器处于 **Running** 阶段时，通过为每个机器运行以下命令来删除旧计算机器集：

```
$ oc delete machineset <original_machine_set_name_1> -n openshift-machine-api
```

验证

- 要验证没有更新的配置的计算机器集已被删除，请运行以下命令列出集群中的计算机器集：

```
$ oc get machinesets -n openshift-machine-api
```

输出示例

```
NAME                 DESIRED   CURRENT   READY   AVAILABLE   AGE
<new_machine_set_name_1>  1       1       1       1         4m12s
<new_machine_set_name_2>  1       1       1       1         4m12s
```

- 要验证没有更新配置的计算机器是否已删除，请运行以下命令列出集群中的机器：

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

删除过程中的输出示例

```
NAME                 PHASE         TYPE    REGION    ZONE            AGE
<machine_from_new_1>      Running       AHV     Unnamed   Development-STS   5m41s
<machine_from_new_2>      Running       AHV     Unnamed   Development-STS   5m41s
<machine_from_original_1> Deleting      AHV     Unnamed   Development-STS   4h
<machine_from_original_2> Deleting      AHV     Unnamed   Development-STS   4h
```

删除完成后的输出示例

```
NAME                 PHASE         TYPE    REGION    ZONE            AGE
<machine_from_new_1>      Running       AHV     Unnamed   Development-STS   6m30s
<machine_from_new_2>      Running       AHV     Unnamed   Development-STS   6m30s
```

- 要验证由新计算机器集创建的机器是否具有正确的配置，请运行以下命令检查 CR 中的相关字段是否有新机器：

```
$ oc describe machine <machine_from_new_1> -n openshift-machine-api
```

其他资源

- 在 Nutanix 上创建计算机器集

第 21 章 AWS LOCAL ZONE 或 WAVELENGTH ZONE 任务

在 Amazon Web Services (AWS) 上安装 OpenShift Container Platform 后，您可以进一步配置 AWS Local Zones 或 Wavelength Zones 和边缘计算池。

21.1. 将现有集群扩展为使用 AWS LOCAL ZONES 或 WAVELENGTH 区域

作为安装后任务，您可以在 Amazon Web Services (AWS) 上扩展现有 OpenShift Container Platform 集群，以使用 AWS Local Zones 或 Wavelength 区域。

将节点扩展到 Local Zones 或 Wavelength 区域位置包括以下步骤：

- 调整 cluster-network 最大传输单元 (MTU)。
- 在 Local Zones 或 Wavelength Zones 组中选择 AWS Local Zones 或 Wavelength 区域。
- 在现有 VPC 中为本地区域或 Wavelength 区域位置创建一个子网。

重要

在将 AWS 上的现有 OpenShift Container Platform 集群扩展为使用 Local Zones 或 Wavelength 区域前，请检查现有的 VPC 是否包含可用的无类别域间路由 (CIDR) 块。创建子网需要这些块。

- 创建机器集清单，然后在每个 Local Zone 或 Wavelength Zone 位置创建一个节点。
- 仅限本地区域：将权限 **ec2:ModifyAvailabilityZoneGroup** 添加到 Identity and Access Management (IAM) 用户或组，以便可以创建所需的网络资源。例如：

  AWS Local Zones 部署的额外 IAM 策略示例

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:ModifyAvailabilityZoneGroup"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- 仅限 Wavelength 区域：将权限 **ec2:ModifyAvailabilityZoneGroup**,**ec2:CreateCarrierGateway**, 和 **ec2:DeleteCarrierGateway** 添加到 Identity and Access Management (IAM) 用户或角色，以便可以创建所需的网络资源。例如：

  AWS Wavelength Zones 部署的额外 IAM 策略示例

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DeleteCarrierGateway",
        "ec2:CreateCarrierGateway"
      ],
      "Resource": "*"
    },
    {
      "Action": [
        "ec2:ModifyAvailabilityZoneGroup"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

其他资源

- 如需有关 AWS Local Zones、支持的实例类型和服务的更多信息，请参阅 AWS 文档中的 AWS Local Zones 功能。
- 有关 AWS 本地区域、支持的实例类型和服务的更多信息，请参阅 AWS 文档中的 AWS Wavelength 功能。

21.1.1. 关于边缘计算池

边缘计算节点是在 AWS Local Zones 或 Wavelength 区域位置中运行的污点计算节点。

在部署使用 Local Zones 或 Wavelength 区域的集群时，请考虑以下点：

- 本地区域或 Wavelength 区域中的 Amazon EC2 实例比可用区中的 Amazon EC2 实例的成本更高。
- 在 AWS Local Zones 或 Wavelength 区域和最终用户中运行的应用程序间延迟较低。例如，在 Local Zones 或 Wavelength 区域和可用区之间混合了入口流量，则对一些工作负载存在延迟影响。

重要

通常，Local Zones 或 Wavelength Zones 中的 Amazon EC2 实例和 Region 中的 Amazon EC2 实例之间的最大传输单元 (MTU) 为 1300。对于开销，集群网络 MTU 必须总是小于 EC2 MTU。具体开销由网络插件决定。例如： OVN-Kubernetes 的开销为 **100 字节**。

网络插件可以提供额外的功能，如 IPsec，它们也会影响 MTU 大小。

您可以访问以下资源以了解更多有关相应区类型的信息：

- 请参阅 AWS 文档中的 Local Zones 的工作原理。
- 请参阅 AWS 文档中的 AWS Wavelength 的工作原理。

OpenShift Container Platform 4.12 引入了一个新的计算池 *edge*，用于在远程区中使用。边缘计算池配置在 AWS Local Zones 或 Wavelength 区域位置之间很常见。由于 Local Zones 或 Wavelength Zones 资源的 EC2 和 EBS 等资源的类型和大小限制，默认的实例类型可能与传统的计算池不同。

Local Zones 或 Wavelength 区域位置的默认 Elastic Block Store (EBS) 是 **gp2**，它与非边缘计算池不同。根据区域上的实例产品，边缘计算池中每个本地区域或 Wavelength 区域使用的实例类型可能与其他计算池不同。

边缘计算池创建新的标签，供开发人员用来将应用程序部署到 AWS Local Zones 或 Wavelength 区域节点上。新标签包括：

- **node-role.kubernetes.io/edge=''**
- 仅限 Local Zones：**machine.openshift.io/zone-type=local-zone**
- 仅限 Wavelength Zones: **machine.openshift.io/zone-type=wavelength-zone**
- **machine.openshift.io/zone-group=$ZONE_GROUP_NAME**

默认情况下，边缘计算池的机器集定义 **NoSchedule** 污点，以防止其他工作负载分散到 Local Zones 或 Wavelength Zones 实例。只有用户在 pod 规格中定义容限时，用户才能运行用户工作负载。

21.2. 更改集群网络 MTU 以支持本地区域或 WAVELENGTH 区域

您可能需要更改集群网络的最大传输单元 (MTU)值，以便集群基础架构可以支持 Local Zones 或 Wavelength 区域子网。

21.2.1. 关于集群 MTU

在安装集群网络的最大传输单元(MTU)期间，会根据集群中节点的主网络接口的 MTU 自动检测到。您通常不需要覆盖检测到的 MTU。

您可能希望因为以下原因更改集群网络的 MTU：

- 集群安装过程中检测到的 MTU 不正确。
- 集群基础架构现在需要不同的 MTU，如添加需要不同 MTU 的节点来获得最佳性能

只有 OVN-Kubernetes 集群网络插件支持更改 MTU 值。

21.2.1.1. 服务中断注意事项

当您为集群启动 MTU 更改时，以下效果可能会影响服务可用性：

- 至少需要两个滚动重启才能完成迁移到新的 MTU。在此过程中，一些节点在重启时不可用。
- 部署到集群的特定应用程序带有较短的超时间隔，超过绝对 TCP 超时间隔可能会在 MTU 更改过程中造成中断。

21.2.1.2. MTU 值选择

在规划 MTU 迁移时，需要考虑两个相关但不同的 MTU 值。

- **Hardware MTU** ：此 MTU 值根据您的网络基础架构的具体设置。

- **Cluster network MTU** ：此 MTU 值始终小于您的硬件 MTU，以考虑集群网络覆盖开销。具体开销由您的网络插件决定。对于 OVN-Kubernetes，开销为 **100** 字节。

如果您的集群为不同的节点需要不同的 MTU 值，则必须从集群中任何节点使用的最低 MTU 值中减去网络插件的开销值。例如，如果集群中的某些节点的 MTU 为 **9001**，而某些节点的 MTU 为 **1500**，则必须将此值设置为 **1400**。

重要

为了避免选择节点无法接受的 MTU 值，请使用 **ip -d link** 命令验证网络接口接受的最大 MTU 值 (**maxmtu**)。

21.2.1.3. 迁移过程如何工作

下表对迁移过程进行了概述，它分为操作中的用户发起的步骤，以及在响应过程中迁移过程要执行的操作。

表 21.1. 集群 MTU 的实时迁移

用户发起的步骤	OpenShift Container Platform 活动
在 Cluster Network Operator 配置中设置以下值： • **spec.migration.mtu.machine.to** • **spec.migration.mtu.network.from** • **spec.migration.mtu.network.to**	**Cluster Network Operator(CNO)** ：确认每个字段都设置为有效的值。 • 如果硬件的 MTU 没有改变，则 **mtu.machine.to** 必须设置为新硬件 MTU 或当前的硬件 MTU。这个值是临时的，被用作迁移过程的一部分。如果您指定了与现有硬件 MTU 值不同的硬件 MTU，您必须手动将 MTU 配置为持久，如机器配置、DHCP 设置或 Linux 内核命令行。 • **mtu.network.from** 字段必须等于 **network.status.clusterNetworkMTU** 字段，这是集群网络的当前 MTU。 • **mtu.network.to** 字段必须设置为目标集群网络 MTU，且必须小于硬件 MTU，以允许网络插件的覆盖开销。对于 OVN-Kubernetes，开销为 **100** 字节。 如果提供的值有效，CNO 会生成一个新的临时配置，它将集群网络集的 MTU 设置为 **mtu.network.to** 字段的值。 **Machine Config Operator(MCO)** ：执行集群中每个节点的滚动重启。
重新配置集群中节点的主网络接口 MTU。您可以使用各种方法完成此操作，包括： • 使用 MTU 更改部署新的 NetworkManager 连接配置集 • 通过 DHCP 服务器设置更改 MTU • 通过引导参数更改 MTU	N/A

用户发起的步骤	OpenShift Container Platform 活动
在网络插件的 CNO 配置中设置 **mtu** 值，并将 **spec.migration** 设置为 **null**。	**Machine Config Operator(MCO)** ：使用新的 MTU 配置执行集群中每个节点的滚动重启。

21.2.1.4. 更改集群网络 MTU

作为集群管理员，您可以增加或减少集群的最大传输单元 (MTU)。

重要

当 MTU 更新推出时，集群中的迁移具有破坏性且节点可能会临时不可用。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已为集群识别目标 MTU。OVN-Kubernetes 网络插件的 MTU 必须设置为比集群中的最低硬件 MTU 值小 **100**。

流程

1. 要获得集群网络的当前 MTU，请输入以下命令：

```
$ oc describe network.config cluster
```

输出示例

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. 要开始 MTU 迁移，请输入以下命令指定迁移配置。Machine Config Operator 在集群中执行节点的滚动重启，以准备 MTU 更改。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } ,
"machine": { "to" : <machine_to> } } } } }'
```

其中：

<overlay_from>

指定当前的集群网络 MTU 值。

<overlay_to>

指定集群网络的目标 MTU。这个值相对于 **<machine_to>** 的值设置。对于 OVN-Kubernetes，这个值必须比 **<machine_to>** 的值小 **100**。

<machine_to>

指定底层主机网络上的主网络接口的 MTU。

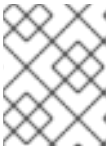
增加集群 MTU 的示例

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": { "to" :
9100} } } } }'
```

3. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。



注意

默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

4. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   输出示例

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   b. 验证以下语句是否正确：

   - **machineconfiguration.openshift.io/state** 字段的值为 **Done**。
   - **machineconfiguration.openshift.io/currentConfig** 字段的值等于 **machineconfiguration.openshift.io/desiredConfig** 字段的值。

   c. 要确认机器配置正确，请输入以下命令：

   ```
   $ oc get machineconfig <config_name> -o yaml | grep ExecStart
   ```

   这里的 **<config_name>** 是 **machineconfiguration.openshift.io/currentConfig** 字段中机器配置的名称。

机器配置必须包括以下对 systemd 配置的更新：

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

5. 要完成 MTU 迁移，请为 OVN-Kubernetes 网络插件输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

其中：

<mtu>
指定您使用 **<overlay_to>** 指定的新集群网络 MTU。

6. 最终调整 MTU 迁移后，每个机器配置池节点都会逐个重启一个。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。

验证

- 输入以下命令验证集群中的节点是否使用您指定的 MTU：

```
$ oc describe network.config cluster
```

21.2.2. 选择 AWS 本地区域或 Wavelength 区域

如果您计划在 AWS Local Zones 或 Wavelength 区域中创建子网，则必须单独选择每个 zone group。

先决条件

- 已安装 AWS CLI。
- 您已决定要部署 OpenShift Container Platform 集群的 AWS 区域。
- 您已将 permissive IAM 策略附加到选择 zone 组的用户或组帐户。

流程

1. 运行以下命令，列出 AWS 区域中可用的区域：

在 AWS 区域中列出可用 AWS 区域的命令示例

```
$ aws --region "<value_of_AWS_Region>" ec2 describe-availability-zones \
    --query 'AvailabilityZones[].[{ZoneName: ZoneName, GroupName: GroupName, Status:
OptInStatus}]' \
    --filters Name=zone-type,Values=local-zone \
    --all-availability-zones
```

在 AWS 区域中列出可用 AWS Wavelength 区域的命令示例

```
$ aws --region "<value_of_AWS_Region>" ec2 describe-availability-zones \
    --query 'AvailabilityZones[].[{ZoneName: ZoneName, GroupName: GroupName, Status:
OptInStatus}]' \
    --filters Name=zone-type,Values=wavelength-zone \
    --all-availability-zones
```

根据 AWS 区域，可用区列表可能比较长。该命令返回以下字段：

ZoneName

本地区域或 Wavelength 区域的名称。

GroupName

组成区域的组。要选择 Region，保存名称。

Status

Local Zones 或 Wavelength Zones 组的状态。如果状态为 **not-opted-in**，则需要选择 **GroupName**，如下一步所述。

2. 运行以下命令，选择 AWS 帐户上的 zone 组：

```
$ aws ec2 modify-availability-zone-group \
    --group-name "<value_of_GroupName>" \ 1
    --opt-in-status opted-in
```

1 将 **<value_of_GroupName>** 替换为您要创建子网的 Local Zones 或 Wavelength 区域的名称。

21.2.3. 在使用 AWS Local Zones 或 Wavelength 区域的现有 VPC 中创建网络要求

如果您希望 Machine API 在远程区位置创建 Amazon EC2 实例，您必须在 Local Zones 或 Wavelength Zones 位置创建一个子网。您可以使用任何置备工具（如 Ansible 或 Terraform）在现有的 Virtual Private Cloud (VPC) 中创建子网。

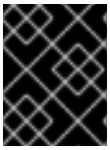
您可以配置 CloudFormation 模板以满足您的要求。以下小节包括使用 CloudFormation 模板创建网络要求的步骤，将现有的 VPC 扩展为使用 AWS Local Zones 或 Wavelength 区域。

将节点扩展到 Local Zones 要求您创建以下资源：

- 2 个 VPC 子网：公共和私有。共子网与 Region 中的普通可用域的公共路由表关联。专用子网与提供的路由表 ID 关联。

将节点扩展到 Wavelength 区域需要您创建以下资源：

- 1 个与提供的 VPC ID 关联的 VPC Carrier Gateway。
- 1 个 VPC 路由表用于 Wavelength 区域，带有到 VPC Carrier Gateway 的默认路由条目。
- 2 个 VPC 子网：公共和私有。公共子网与 AWS Wavelength Zone 的公共路由表关联。专用子网与提供的路由表 ID 关联。



重要

考虑 Wavelength 区域中的 NAT 网关限制，提供的 CloudFormation 模板支持仅将专用子网与提供的路由表 ID 关联。路由表 ID 附加到 AWS 区域中的有效 NAT 网关。

21.2.4. 仅限 Wavelength 区域：创建 VPC 载体网关

要在 Wavelength 区域上运行的 OpenShift Container Platform 集群中使用公共子网，您必须创建载体网关，并将载体网关关联到 VPC。子网可用于部署负载均衡器或边缘计算节点。

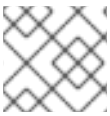
要在 OpenShift Container Platform 集群的 Wavelength 区域位置创建边缘节点或面向互联网的负载均衡器，您必须创建以下所需的网络组件：

- 与现有 VPC 关联的载体网关。
- 列出路由条目的载波路由表。
- 与载体路由表关联的子网。

对于仅包含 Wavelength 区中的子网的 VPC 存在载体网关。

以下列表解释了 AWS Wavelength 区域位置上下文中的载体网关的功能：

- 提供 Wavelength Zone 和 carrier 网络之间的连接，其中包括来自载体网络的任何可用设备。
- 执行网络地址转换(NAT)功能，如将作为网络边框组的公共 IP 地址（从 Wavelength 区域转换为载体 IP 地址）的 IP 地址转换。这些转换功能适用于入站和出站流量。
- 授权来自位于特定位置的载体网络的入站流量。
- 授权到载波网络和互联网的出站流量。

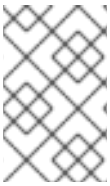


注意

互联网没有入站连接配置，通过载体网关到 Wavelength 区域。

您可以使用提供的 CloudFormation 模板来创建以下 AWS 资源堆栈：

- 一个载体网关，与模板中的 VPC ID 关联。
- 一个用于 Wavelength Zone 的公共路由表，名为 **<ClusterName>-public-carrier**。
- 以载体网关为目标的新路由表中的默认 IPv4 路由条目。
- AWS Simple Storage Service (S3) 的 VPC 网关端点。



注意

如果不使用提供的 CloudFormation 模板来创建 AWS 基础架构，您必须检查提供的信息并手动创建基础架构。如果集群没有正确初始化，您可能需要联系红帽支持并提供您的安装日志。

先决条件

- 已配置了一个 AWS 帐户。
- 您可以通过运行 **aws configure**，将 AWS 密钥和区域添加到本地 AWS 配置集中。

流程

1. 进入名为"CloudFormation template for the VPC Carrier Gateway"的文档的下一部分，然后复制 **VPC Carrier Gateway 模板的 CloudFormation**模板的语法。将复制的模板语法保存为本地系统中的 YAML 文件。此模板描述了集群所需的 VPC。

2. 运行以下命令来部署 CloudFormation 模板，它会创建一个代表 VPC 的 AWS 资源堆栈：

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \//
    ParameterKey=VpcId,ParameterValue="${VpcId}" \ 3
    ParameterKey=ClusterName,ParameterValue="${ClusterName}" 4
```

1. **<stack_name>** 是 CloudFormation 堆栈的名称，如 **clusterName-vpc-carrier-gw**。如果您删除集群，则需要此堆栈的名称。
2. **<template>** 是相对路径，以及保存的 CloudFormation 模板 YAML 文件的名称。
3. **<VpcId>** 是从名为"Creating a VPC in AWS"部分创建的 CloudFormation 堆栈输出中提取的 VPC ID。
4. **<clusterName>** 是一个自定义值，前缀为 CloudFormation 堆栈创建的资源的前缀。您可以使用 **install-config.yaml** 配置文件的 **metadata.name** 部分中定义的相同名称。

输出示例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-2fd3-11eb-820e-12a48460849f
```

验证

- 运行以下命令确认 CloudFormation 模板组件已存在：

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

在 **StackStatus** 显示 **CREATE_COMPLETE** 后，输出显示以下参数的值：确保为集群运行的其他 CloudFormation 模板提供参数值。

PublicRouteTableId	Carrier 基础架构中路由表 ID。

21.2.5. 仅限 Wavelength 区域：VPC Carrier Gateway 的 CloudFormation 模板

您可以使用以下 CloudFormation 模板，在 AWS Wavelength 基础架构上部署 Carrier Gateway。

例 21.1. VPC Carrier Gateway 的 CloudFormation 模板

```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Creating Wavelength Zone Gateway (Carrier Gateway).

Parameters:
  VpcId:
```

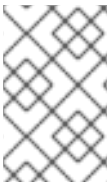
```
    Description: VPC ID to associate the Carrier Gateway.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
  ClusterName:
    Description: Cluster Name or Prefix name to prepend the tag Name for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.

Resources:
  CarrierGateway:
    Type: "AWS::EC2::CarrierGateway"
    Properties:
      VpcId: !Ref VpcId
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "cagw"]]

  PublicRouteTable:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VpcId
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "public-carrier"]]

  PublicRoute:
    Type: "AWS::EC2::Route"
    DependsOn: CarrierGateway
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      CarrierGatewayId: !Ref CarrierGateway

  S3Endpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      PolicyDocument:
        Version: 2012-10-17
        Statement:
        - Effect: Allow
          Principal: '*'
          Action:
          - '*'
          Resource:
          - '*'
      RouteTableIds:
      - !Ref PublicRouteTable
      ServiceName: !Join
      - ''
      - - com.amazonaws.
        - !Ref 'AWS::Region'
        - .s3
      VpcId: !Ref VpcId
```

```
Outputs:
  PublicRouteTableId:
    Description: Public Route table ID
    Value: !Ref PublicRouteTable
```

21.2.6. 为 AWS 边缘计算服务创建子网

在 OpenShift Container Platform 集群中为边缘计算节点配置机器集前，您必须在 Local Zones 或 Wavelength Zones 中创建子网。对您要将计算节点部署到的每个 Wavelength 区完成以下步骤。

您可以使用提供的 CloudFormation 模板并创建 CloudFormation 堆栈。然后，您可以使用此堆栈自定义置备子网。



注意

如果不使用提供的 CloudFormation 模板来创建 AWS 基础架构，您必须检查提供的信息并手动创建基础架构。如果集群没有正确初始化，您可能需要联系红帽支持并提供您的安装日志。

先决条件

- 已配置了一个 AWS 帐户。
- 您可以通过运行 **aws configure**，将 AWS 密钥和区域添加到本地 AWS 配置集中。
- 您可以选择 Local Zones 或 Wavelength Zones 组。

流程

1. 进入名为"CloudFormation template for the VPC 子网"的文档部分，并从模板中复制语法。将复制的模板语法保存为本地系统中的 YAML 文件。此模板描述了集群所需的 VPC。
2. 运行以下命令来部署 CloudFormation 模板，它会创建一个代表 VPC 的 AWS 资源堆栈：

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \
    ParameterKey=VpcId,ParameterValue="${VPC_ID}" \ 3
    ParameterKey=ClusterName,ParameterValue="${CLUSTER_NAME}" \ 4
    ParameterKey=ZoneName,ParameterValue="${ZONE_NAME}" \ 5
    ParameterKey=PublicRouteTableId,ParameterValue="${ROUTE_TABLE_PUB}" \ 6
    ParameterKey=PublicSubnetCidr,ParameterValue="${SUBNET_CIDR_PUB}" \ 7
    ParameterKey=PrivateRouteTableId,ParameterValue="${ROUTE_TABLE_PVT}" \ 8
    ParameterKey=PrivateSubnetCidr,ParameterValue="${SUBNET_CIDR_PVT}" 9
```

1. **<stack_name>** 是 CloudFormation 堆栈的名称，如用于 Local Zones 的 **cluster-wl-<local_zone_shortname>**，用于 Wavelength Zones 的 **cluster-wl-<wavelength_zone_shortname>**。如果您删除集群，则需要此堆栈的名称。
2. **<template>** 是相对路径，以及保存的 CloudFormation 模板 YAML 文件的名称。

3. **${VPC_ID}** 是 VPC ID，它是 VPC 模板输出中的 **VpcID** 值。
4. **${CLUSTER_NAME}** 是 **ClusterName** 的值，用作新 AWS 资源名称的前缀。
5. **${ZONE_NAME}** 是创建子网的 Local Zones 或 Wavelength 区域名称的值。
6. **${ROUTE_TABLE_PUB}** 是从 CloudFormation 模板中提取的公共路由表 Id。对于 Local Zones，从 VPC CloudFormation 堆栈中提取公共路由表。对于 Wavelength Zones，值必须从 VPC 的载体网关 CloudFormation 堆栈的输出中提取。
7. **${SUBNET_CIDR_PUB}** 是一个有效的 CIDR 块，用于创建公共子网。这个块必须是 VPC CIDR 块 **VpcCidr** 的一部分。
8. **${ROUTE_TABLE_PVT}** 是从 VPC 的 CloudFormation 堆栈的输出中提取的 **PrivateRouteTableId**。
9. **${SUBNET_CIDR_PVT}** 是一个有效的 CIDR 块，用于创建专用子网。这个块必须是 VPC CIDR 块 **VpcCidr** 的一部分。

输出示例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-820e-11eb-2fd3-12a48460849f
```

验证

- 运行以下命令确认模板组件已存在：

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

在 **StackStatus** 显示 **CREATE_COMPLETE** 后，输出会显示以下参数的值：

PublicSubnetId	由 CloudFormation 堆栈创建的公共子网的 ID。
PrivateSubnetId	由 CloudFormation 堆栈创建的专用子网的 ID。

确保将这些参数值提供给您为集群创建的其他 CloudFormation 模板。

21.2.7. VPC 子网的 CloudFormation 模板

您可以使用以下 CloudFormation 模板，在 Local Zones 或 Wavelength 区域基础架构上部署私有和公共子网。

例 21.2. VPC 子网的 CloudFormation 模板

```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Best Practice Subnets (Public and Private)

Parameters:
  VpcId:
```

```
    Description: VPC ID that comprises all the target subnets.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
  ClusterName:
    Description: Cluster name or prefix name to prepend the Name tag for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.
  ZoneName:
    Description: Zone Name to create the subnets, such as us-west-2-lax-1a.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ZoneName parameter must be specified.
  PublicRouteTableId:
    Description: Public Route Table ID to associate the public subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PublicRouteTableId parameter must be specified.
  PublicSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for public subnet.
    Type: String
  PrivateRouteTableId:
    Description: Private Route Table ID to associate the private subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PrivateRouteTableId parameter must be specified.
  PrivateSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for private subnet.
    Type: String


Resources:
  PublicSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PublicSubnetCidr
      AvailabilityZone: !Ref ZoneName
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "public", !Ref ZoneName]]

  PublicSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTableId
```

```
  PrivateSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PrivateSubnetCidr
      AvailabilityZone: !Ref ZoneName
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "private", !Ref ZoneName]]

  PrivateSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnet
      RouteTableId: !Ref PrivateRouteTableId

Outputs:
  PublicSubnetId:
    Description: Subnet ID of the public subnets.
    Value:
      !Join ["", [!Ref PublicSubnet]]

  PrivateSubnetId:
    Description: Subnet ID of the private subnets.
    Value:
      !Join ["", [!Ref PrivateSubnet]]
```

21.2.8. 为 AWS Local Zones 或 Wavelength 区域节点创建机器集清单

在 AWS Local Zones 或 Wavelength 区域中创建子网后，您可以创建机器集清单。

安装程序在集群安装过程中为 **edge** 机器池设置以下标签：

- **machine.openshift.io/parent-zone-name: <value_of_ParentZoneName>**
- **machine.openshift.io/zone-group: <value_of_ZoneGroup>**
- **machine.openshift.io/zone-type: <value_of_ZoneType>**

以下过程详细介绍了如何创建与 **edge** 计算池 配置匹配的机器集 configuraton。

先决条件

- 您已在 AWS Local Zones 或 Wavelength 区域中创建子网。

流程

- 通过收集 AWS API 创建机器集清单时，手动保留 **edge** 机器池标签。要完成此操作，请在命令行界面(CLI)中输入以下命令：

```
$ aws ec2 describe-availability-zones --region <value_of_Region> \ 1
    --query 'AvailabilityZones[].{
  ZoneName: ZoneName,
```

```
ParentZoneName: ParentZoneName,
GroupName: GroupName,
ZoneType: ZoneType}' \
  --filters Name=zone-name,Values=<value_of_ZoneName> \ 2
  --all-availability-zones
```

1 对于 **<value_of_Region>**，请指定区的区域名称。

2 对于 **<value_of_ZoneName>**，请指定 Local Zones 或 Wavelength Zones 的名称。

Local Zone us-east-1-nyc-1a 的输出示例

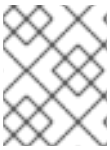
```
[
  {
    "ZoneName": "us-east-1-nyc-1a",
    "ParentZoneName": "us-east-1f",
    "GroupName": "us-east-1-nyc-1",
    "ZoneType": "local-zone"
  }
]
```

Wavelength Zone us-east-1-wl1 的输出示例

```
[
  {
    "ZoneName": "us-east-1-wl1-bos-wlz-1",
    "ParentZoneName": "us-east-1a",
    "GroupName": "us-east-1-wl1",
    "ZoneType": "wavelength-zone"
  }
]
```

21.2.8.1. AWS 上计算机器设置自定义资源的 YAML 示例

此 YAML 示例定义了一个在 **us-east-1-nyc-1a** Amazon Web Services (AWS)区域中运行的计算机器集，并创建带有 **node-role.kubernetes.io/edge: ""** 标记的节点。



注意

如果要在 Wavelength 区域上下文中引用示例 YAML 文件，请确保将 AWS Region 和 zone 信息替换为支持的 Wavelength Zone 值。

在本例中，**<infrastructure_id>** 是基础架构 ID 标签，该标签基于您在置备集群时设定的集群 ID，而 **<edge>** 则是要添加的节点标签。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-edge-<zone> 2
  namespace: openshift-machine-api
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-edge-<zone>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> 4
        machine.openshift.io/cluster-api-machine-role: edge 5
        machine.openshift.io/cluster-api-machine-type: edge 6
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-edge-<zone> 7
    spec:
      metadata:
        labels:
          machine.openshift.io/parent-zone-name: <value_of_ParentZoneName>
          machine.openshift.io/zone-group: <value_of_GroupName>
          machine.openshift.io/zone-type: <value_of_ZoneType>
          node-role.kubernetes.io/edge: "" 8
      providerSpec:
        value:
          ami:
            id: ami-046fe691f52a953f9 9
          apiVersion: machine.openshift.io/v1beta1
          blockDevices:
            - ebs:
                iops: 0
                volumeSize: 120
                volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: <infrastructure_id>-worker-profile 10
          instanceType: m6i.large
          kind: AWSMachineProviderConfig
          placement:
            availabilityZone: <zone> 11
            region: <region> 12
          securityGroups:
            - filters:
                - name: tag:Name
                  values:
                    - <infrastructure_id>-worker-sg 13
          subnet:
              id: <value_of_PublicSubnetIds> 14
          publicIp: true
          tags:
            - name: kubernetes.io/cluster/<infrastructure_id> 15
              value: owned
            - name: <custom_tag_name> 16
              value: <custom_tag_value> 17
          userDataSecret:
```

```
      name: worker-user-data
    taints: 18
      - key: node-role.kubernetes.io/edge
        effect: NoSchedule
```

1 3 4 10 13 15 指定基于置备集群时所设置的集群 ID 的基础架构 ID。如果已安装 OpenShift CLI，您可以通过运行以下命令来获取基础架构 ID：

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

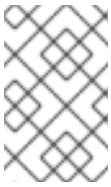
2 7 指定基础架构 ID、**edge** 角色节点标签和区域名称。

5 6 8 指定 **edge** 角色节点标签。

9 为 OpenShift Container Platform 节点的 AWS 区域指定有效的 Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Image (AMI)。如果要使用 AWS Marketplace 镜像，则必须从 AWS Marketplace 完成 OpenShift Container Platform 订阅来获取您所在地区的 AMI ID。

```
$ oc -n openshift-machine-api \
    -o jsonpath='{.spec.template.spec.providerSpec.value.ami.id}{"\n"}' \
    get machineset/<infrastructure_id>-<role>-<zone>
```

16 17 可选：为集群指定自定义标签数据。例如，您可以通过指定 **Email:admin-email@example.com** 的 **name:value** 对来添加管理员的电子邮件地址。



注意

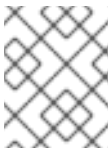
也可以在 **install-config.yml** 文件中在安装过程中指定自定义标签。如果 **install-config.yml** 文件和机器集包含具有相同 **name** 数据的标签，则机器集的标签值优先于 **install-config.yml** 文件中的标签值。

11 指定区域名称，如 **us-east-1-nyc-1a**。

12 指定区域，如 **us-east-1**。

14 您在 AWS Local Zones 或 Wavelength 区域中创建的公共子网的 ID。完成"在 AWS 区域中创建子网"的步骤后，创建了此公共子网 ID。

18 指定污点，以防止将用户工作负载调度到 **edge** 节点上。



注意

在基础架构节点上添加 **NoSchedule** 污点后，在该节点上运行的现有 DNS pod 被标记为 **misscheduled**。您必须删除或在 **misscheduled** DNS pod 中添加容限。

21.2.8.2. 创建计算机器集

除了安装程序创建的计算机器集外，您还可以创建自己的来动态管理您选择的特定工作负载的机器计算资源。

先决条件

- 部署一个 OpenShift Container Platform 集群。
- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 权限的用户身份登录 **oc**。

流程

1. 创建一个包含计算机器集自定义资源（CR）示例的新 YAML 文件，并将其命名为 **<file_name>.yaml**。
   确保设置 **<clusterID>** 和 **<role>** 参数值。
2. 可选：如果您不确定要为特定字段设置哪个值，您可以从集群中检查现有计算机器集：
   a. 要列出集群中的计算机器集，请运行以下命令：

```
$ oc get machinesets -n openshift-machine-api
```

输**出示例**

```
NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d   0         0                             55m
agl030519-vplxk-worker-us-east-1e   0         0                             55m
agl030519-vplxk-worker-us-east-1f   0         0                             55m
```

   b. 要查看特定计算机器集自定义资源 (CR) 的值，请运行以下命令：

```
$ oc get machineset <machineset_name> \
  -n openshift-machine-api -o yaml
```

输**出示例**

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-<role> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
```

```
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
spec:
  providerSpec: 3
    ...
```

1 集群基础架构 ID。

2 默认节点标签。

注意

对于具有用户置备的基础架构的集群，计算机器集只能创建 **worker** 和 **infra** 类型机器。

3 计算机器设置 CR 的 **<providerSpec>** 部分中的值是特定于平台的。有关 CR 中的 **<providerSpec>** 参数的更多信息，请参阅您的供应商计算机器设置 CR 配置示例。

3. 运行以下命令来创建 **MachineSet** CR：

```
$ oc create -f <file_name>.yaml
```

验证

- 运行以下命令，查看计算机器集列表：

```
$ oc get machineset -n openshift-machine-api
```

输出示例

```
NAME                                 DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-edge-us-east-1-nyc-1a   1         1         1       1           11m
agl030519-vplxk-worker-us-east-1a       1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b       1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c       1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d       0         0                             55m
agl030519-vplxk-worker-us-east-1e       0         0                             55m
agl030519-vplxk-worker-us-east-1f       0         0                             55m
```

当新的计算机器集可用时，**DESIRED** 和 **CURRENT** 的值会匹配。如果 compute 机器集不可用，请等待几分钟，然后再次运行命令。

- 可选：要检查边缘机器创建的节点，请运行以下命令：

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

输出示例

```
NAME                          STATUS   ROLES         AGE    VERSION
ip-10-0-207-188.ec2.internal   Ready    edge,worker   172m   v1.25.2+d2e245f
```

其他资源

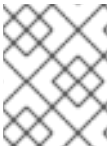
- 使用 AWS Local Zones 上的计算节点在 AWS 上安装集群
- 在 AWS 上使用计算节点在 AWS Wavelength 区域上安装集群

21.3. 在 AWS 本地区中或 WAVELENGTH 区域创建用户工作负载

创建 Amazon Web Service (AWS) Local Zones 或 Wavelength 区域基础架构并部署集群后，您可以使用边缘计算节点在 Local Zones 或 Wavelength 区域子网中创建用户工作负载。

当使用安装程序创建集群时，安装程序会自动为每个边缘计算节点指定 **NoSchedule** 的污点效果。这意味着，如果 pod 与污点的指定容限不匹配，调度程序不会向节点添加新 pod 或部署。您可以修改污点，以更好地控制节点如何在每个本地区域或 Wavelength 区域子网中创建工作负载。

安装程序创建计算机器集清单文件，该文件带有 **node-role.kubernetes.io/edge** 和 **node-role.kubernetes.io/worker** 标签，应用到位于 Local Zones 或 Wavelength Zones 子网中的每个边缘计算节点。



注意

该流程中的示例用于本地区域基础架构。如果您使用 Wavelength 区域基础架构，请确保将示例适应此基础架构支持的内容。

先决条件

- 您可以访问 OpenShift CLI(**oc**)。
- 您在带有定义的 Local Zones 或 Wavelength 区域子网中的虚拟私有云 (VPC) 中部署了集群。
- 确保 Local Zones 或 Wavelength Zones 子网上边缘计算节点的计算机器集指定了 **node-role.kubernetes.io/edge** 的污点。

流程

1. 为要在 Local Zones 子网中运行的边缘计算节点中部署的示例应用程序创建一个 **deployment** 资源 YAML 文件。确保指定与边缘计算节点污点匹配的正确容限。

   在 Local Zone 子网中运行的边缘计算节点配置的 deployment 资源示例

```
kind: Namespace
apiVersion: v1
metadata:
  name: <local_zone_application_namespace>
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc_name>
  namespace: <local_zone_application_namespace>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: gp2-csi (1)
```

```
  volumeMode: Filesystem
---
apiVersion: apps/v1
kind: Deployment (2)
metadata:
  name: <local_zone_application> (3)
  namespace: <local_zone_application_namespace> (4)
spec:
  selector:
    matchLabels:
      app: <local_zone_application>
  replicas: 1
  template:
    metadata:
      labels:
        app: <local_zone_application>
        zone-group: ${ZONE_GROUP_NAME} (5)
    spec:
      securityContext:
        seccompProfile:
          type: RuntimeDefault
      nodeSelector: (6)
        machine.openshift.io/zone-group: ${ZONE_GROUP_NAME}
      tolerations: (7)
      - key: "node-role.kubernetes.io/edge"
        operator: "Equal"
        value: ""
        effect: "NoSchedule"
      containers:
        - image: openshift/origin-node
          command:
           - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
          imagePullPolicy: Always
          name: echoserver
          ports:
            - containerPort: 8080
          volumeMounts:
            - mountPath: "/mnt/storage"
              name: data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: <pvc_name>
```

1. **storageClassName** ：对于 Local Zone 配置，您必须指定 **gp2-csi**。
2. **kind** ：定义 **deployment** 资源。
3. **name** ：指定 Local Zone 应用程序的名称。例如， **local-zone-demo-app-nyc-1**。
4. **namespace** ：定义您要运行用户工作负载的 AWS Local Zone 的命名空间。例如： **local-zone-app-nyc-1a**。

5 **zone-group** ：定义区域所属的组。例如，**us-east-1-iah-1**。

6 **nodeSelector** ：目标与指定标签匹配的边缘计算节点。

7 **tolerations** ：设置与 Local Zone 节点的 **MachineSet** 清单上定义的污点匹配的值。

2. 为节点创建 **service** 资源 YAML 文件。此资源将来自目标边缘计算节点的 pod 公开给在 Local Zone 网络中运行的服务。

在 Local Zone 子网中运行的边缘计算节点配置的 service 资源示例

```
apiVersion: v1
kind: Service 1
metadata:
  name:  <local_zone_application>
  namespace: <local_zone_application_namespace>
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector: 2
    app: <local_zone_application>
```

1 **kind** ：定义 **service** 资源。

2 **selector：** 指定应用到受管 pod 的标签类型。

其他资源

- 使用 AWS Local Zones 上的计算节点在 AWS 上安装集群
- 在 AWS 上使用计算节点在 AWS Wavelength 区域上安装集群
- 了解污点和容限

21.4. 后续步骤

- 可选： 使用 AWS Load Balancer (ALB) Operator 将目标边缘计算节点中的 pod 公开给在本地区域或公共网络中运行的服务。请参阅安装 AWS Load Balancer Operator。

第 22 章 将 AWS VPC 集群扩展到 AWS OUTPOST

在 Amazon Web Services (AWS)上安装集群到现有的 Amazon Virtual Private Cloud (VPC) 后，您可以创建一个在 AWS Outposts 中部署计算机器的计算机器集。AWS Outposts 是一个 AWS 边缘计算服务，允许使用基于云的 AWS 部署的许多功能，并减少内部环境的延迟。如需更多信息，请参阅 AWS Outposts 文档。

22.1. OPENSHIFT CONTAINER PLATFORM 要求和限制的 AWS OUTPOSTS

如果配置 OpenShift Container Platform 集群以适应以下要求和限制，您可以管理 AWS Outpost 上的资源，与基于云的 AWS 集群上的资源类似：

- 要将 AWS 上的 OpenShift Container Platform 集群扩展到 Outpost 中，您必须将集群安装到现有的 Amazon Virtual Private Cloud (VPC) 中。
- Outpost 的基础架构与 AWS 区域中的可用区关联，并使用专用子网。部署到 Outpost 中的边缘计算机器必须使用 Outpost 子网以及 Outpost 与之关联的可用区。
- 当 AWS Kubernetes 云控制器管理器发现 Outpost 子网时，它会尝试在 Outpost 子网中创建服务负载均衡器。AWS Outposts 不支持运行服务负载均衡器。要防止云控制器管理器在 Outpost 子网中创建不支持的服务，您必须在 Outpost 子网配置中包含 **kubernetes.io/cluster/unmanaged** 标签。这个要求是 OpenShift Container Platform 版本 4.15 的一个临时解决方案。如需更多信息，请参阅 OCPBUGS-30041。
- AWS 上的 OpenShift Container Platform 集群包括 **gp3-csi** 和 **gp2-csi** 存储类。这些类与 Amazon Elastic Block Store (EBS) gp3 和 gp2 卷对应。OpenShift Container Platform 集群默认使用 **gp3-csi** 存储类，但 AWS Outposts 不支持 EBS gp3 卷。
- 此实现使用 **node-role.kubernetes.io/outposts** 污点，以防止将常规集群工作负载分散到 Outpost 节点。要在 Outpost 中调度用户工作负载，您必须在应用程序的 **Deployment** 资源中指定对应的容限。为用户工作负载保留 AWS Outpost 基础架构可以避免额外的配置要求，如将默认 CSI 更新至 **gp2-csi**，使其兼容。
- 要在 Outpost 中创建卷，CSI 驱动程序需要 Outpost Amazon Resource Name (ARN)。驱动程序使用 **CSINode** 对象中存储的拓扑密钥来确定 Outpost ARN。为确保驱动程序使用正确的拓扑值，您必须将卷绑定模式设置为 **WaitForConsumer**，并避免在您创建的任何新存储类上设置允许的拓扑。
- 当您将 AWS VPC 集群扩展到 Outpost 时，您有两个计算资源。Outpost 具有边缘计算节点，而 VPC 具有基于云的计算节点。基于云的 AWS Elastic Block 卷无法附加到 Outpost 边缘计算节点，而 Outpost 卷无法附加到基于云的计算节点。
  因此，您无法使用 CSI 快照将使用持久性存储的应用程序从基于云的计算节点迁移到边缘计算节点，或直接使用原始持久性卷。要为应用程序迁移持久性存储数据，您必须执行手动备份和恢复操作。
- AWS Outposts 不支持 AWS Network Load Balancers 或 AWS Classic Load Balancers。您必须使用 AWS Application Load Balancers 在 AWS Outposts 环境中为边缘计算资源启用负载均衡。要置备 Application Load Balancer，您必须使用 Ingress 资源并安装 AWS Load Balancer Operator。如果您的集群包含共享工作负载的边缘和基于云的计算实例，则需要额外的配置。

  如需更多信息，请参阅"在 AWS VPC 集群中使用 AWS Load Balancer Operator 进入 Outpost"。

其他资源

- 在 AWS VPC 集群中使用 AWS Load Balancer Operator 扩展至 Outpost

22.2. 获取有关您的环境的信息

要将 AWS VPC 集群扩展到 Outpost，您必须提供有关 OpenShift Container Platform 集群和 Outpost 环境的信息。您可以使用此信息完成网络配置任务，并配置在 Outpost 中创建计算机器的计算机器集。您可以使用命令行工具收集所需的详情。

22.2.1. 从 OpenShift Container Platform 集群获取信息

您可以使用 OpenShift CLI (**oc**)从 OpenShift Container Platform 集群获取信息。

提示

您可以使用 **export** 命令，将部分或所有值存储为环境变量。

先决条件

- 您已将 OpenShift Container Platform 集群安装到 AWS 上的自定义 VPC 中。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 运行以下命令，列出集群的基础架构 ID。保留这个值。

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructures.config.openshift.io
cluster
```

2. 运行以下命令，获取安装程序创建的计算机器集的详情：

   a. 列出集群中的计算机器集：

```
$ oc get machinesets.machine.openshift.io -n openshift-machine-api
```

   输出示例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<compute_machine_set_name_1>  1         1         1       1           55m
<compute_machine_set_name_2>  1         1         1       1           55m
```

   b. 显示列出的计算机器集的 Amazon Machine Image (AMI) ID。保留这个值。

```
$ oc get machinesets.machine.openshift.io <compute_machine_set_name_1> \
  -n openshift-machine-api \
  -o jsonpath='{.spec.template.spec.providerSpec.value.ami.id}'
```

   c. 显示 AWS VPC 集群的子网 ID。保留这个值。

```
$ oc get machinesets.machine.openshift.io <compute_machine_set_name_1> \
  -n openshift-machine-api \
  -o jsonpath='{.spec.template.spec.providerSpec.value.subnet.id}'
```

22.2.2. 从 AWS 帐户获取信息

您可以使用 AWS CLI (**aws**) 从 AWS 帐户获取信息。

提示

您可以使用 **export** 命令，将部分或所有值存储为环境变量。

先决条件

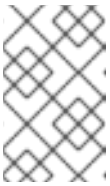
- 您有一个带有所需硬件设置的 AWS Outposts 站点。
- 您的 Outpost 连接到您的 AWS 帐户。
- 您可以使用 AWS CLI (**aws**) 作为具有执行所需任务权限的用户访问 AWS 帐户。

流程

1. 运行以下命令，列出连接到 AWS 帐户的 Outposts：

```
$ aws outposts list-outposts
```

2. 保留 **aws outposts list-outposts** 命令的输出中的以下值：
   - Outpost ID。
   - Outpost 的 Amazon 资源名称 (ARN)。
   - Outpost 可用区。



注意

aws outposts list-outposts 命令的输出包括两个与可用区相关的值：**AvailabilityZone** 和 **AvailabilityZoneId**。您可以使用 **AvailablilityZone** 值配置在 Outpost 中创建计算机器的计算机器集。

3. 使用 Outpost ID 的值，运行以下命令来显示 Outpost 中可用的实例类型。保留可用实例类型的值。

```
$ aws outposts get-outpost-instance-types \
  --outpost-id <outpost_id_value>
```

4. 使用 Outpost ARN 的值，运行以下命令来显示 Outpost 的子网 ID。保留这个值。

```
$ aws ec2 describe-subnets \
  --filters Name=outpost-arn,Values=<outpost_arn_value>
```

22.3. 为您的 OUTPOST 配置网络

要将 VPC 集群扩展到 Outpost 中，您必须完成以下网络配置任务：

- 更改集群网络 MTU。
- 在 Outpost 中创建子网。

22.3.1. 更改集群网络 MTU 以支持 AWS Outposts

在安装过程中，集群网络的最大传输单元 (MTU) 会根据集群中节点的主网络接口的 MTU 自动检测到。您可能需要减少集群网络的 MTU 值来支持 AWS Outposts 子网。

重要

当 MTU 更新推出时，集群中的迁移具有破坏性且节点可能会临时不可用。

有关迁移过程的详情，包括重要的服务中断注意事项，请参阅此流程的其他资源中的"为集群网络分配 MTU"。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已为集群识别目标 MTU。OVN-Kubernetes 网络插件的 MTU 必须设置为比集群中的最低硬件 MTU 值小 **100**。

流程

1. 要获得集群网络的当前 MTU，请输入以下命令：

```
$ oc describe network.config cluster
```

输出示例

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. 要开始 MTU 迁移，请输入以下命令指定迁移配置。Machine Config Operator 在集群中执行节点的滚动重启，以准备 MTU 更改。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } ,
"machine": { "to" : <machine_to> } } } } }'
```

其中：

<overlay_from>

指定当前的集群网络 MTU 值。

<overlay_to>

指定集群网络的目标 MTU。这个值相对于 **<machine_to>** 的值设置。对于 OVN-Kubernetes，这个值必须比 **<machine_to>** 的值小 **100**。

<machine_to>

指定底层主机网络上的主网络接口的 MTU。

减**少集群 MTU 的示例**

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 1000 } , "machine": { "to" :
1100} } } } }'
```

3. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。

注意

默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

4. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   输**出示例**

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   b. 验证以下语句是否正确：

   - **machineconfiguration.openshift.io/state** 字段的值为 **Done**。
   - **machineconfiguration.openshift.io/currentConfig** 字段的值等于 **machineconfiguration.openshift.io/desiredConfig** 字段的值。

   c. 要确认机器配置正确，请输入以下命令：

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

这里的 **<config_name>** 是 **machineconfiguration.openshift.io/currentConfig** 字段中机器配置的名称。

机器配置必须包括以下对 systemd 配置的更新：

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

5. 要完成 MTU 迁移，请为 OVN-Kubernetes 网络插件输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

其中：

<mtu>
指定您使用 **<overlay_to>** 指定的新集群网络 MTU。

6. 最终调整 MTU 迁移后，每个机器配置池节点都会逐个重启一个。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。

验证

- 输入以下命令验证集群中的节点是否使用您指定的 MTU：

```
$ oc describe network.config cluster
```

其他资源

- 更改集群网络的 MTU

22.3.2. 为 AWS 边缘计算服务创建子网

在 OpenShift Container Platform 集群中为边缘计算节点配置机器集前，您必须在 AWS Outposts 中创建子网。

您可以使用提供的 CloudFormation 模板并创建 CloudFormation 堆栈。然后，您可以使用此堆栈自定义置备子网。

注意

如果不使用提供的 CloudFormation 模板来创建 AWS 基础架构，您必须检查提供的信息并手动创建基础架构。如果集群没有正确初始化，您可能需要联系红帽支持并提供您的安装日志。

先决条件

- 已配置了一个 AWS 帐户。

- 您可以通过运行 **aws configure**，将 AWS 密钥和区域添加到本地 AWS 配置集中。
- 您已从 OpenShift Container Platform 集群、Outpost 和 AWS 帐户获取环境所需的信息。

流程

1. 进入名为"CloudFormation template for the VPC 子网"的文档部分，并从模板中复制语法。将复制的模板语法保存为本地系统中的 YAML 文件。此模板描述了集群所需的 VPC。
2. 运行以下命令来部署 CloudFormation 模板，它会创建一个代表 VPC 的 AWS 资源堆栈：

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \
    ParameterKey=VpcId,ParameterValue="${VPC_ID}" \ 3
    ParameterKey=ClusterName,ParameterValue="${CLUSTER_NAME}" \ 4
    ParameterKey=ZoneName,ParameterValue="${ZONE_NAME}" \ 5
    ParameterKey=PublicRouteTableId,ParameterValue="${ROUTE_TABLE_PUB}" \ 6
    ParameterKey=PublicSubnetCidr,ParameterValue="${SUBNET_CIDR_PUB}" \ 7
    ParameterKey=PrivateRouteTableId,ParameterValue="${ROUTE_TABLE_PVT}" \ 8
    ParameterKey=PrivateSubnetCidr,ParameterValue="${SUBNET_CIDR_PVT}" \ 9
    ParameterKey=PrivateSubnetLabel,ParameterValue="private-outpost" \
    ParameterKey=PublicSubnetLabel,ParameterValue="public-outpost" \
    ParameterKey=OutpostArn,ParameterValue="${OUTPOST_ARN}" 10
```

1. **<stack_name>** 是 CloudFormation 堆栈的名称，如 **cluster-<outpost_name>**。
2. **<template>** 是相对路径，以及保存的 CloudFormation 模板 YAML 文件的名称。
3. **${VPC_ID}** 是 VPC ID，它是 VPC 模板输出中的 **VpcID** 值。
4. **${CLUSTER_NAME}** 是 **ClusterName** 的值，用作新 AWS 资源名称的前缀。
5. **${ZONE_NAME}** 是创建子网的 AWS Outposts 名称的值。
6. **${ROUTE_TABLE_PUB}** 是 **${VPC_ID}** 中创建的公共路由表 ID，用于关联 Outposts 上的公共子网。指定用于关联此堆栈创建的 Outpost 子网的公共路由表。
7. **${SUBNET_CIDR_PUB}** 是一个有效的 CIDR 块，用于创建公共子网。这个块必须是 VPC CIDR 块 **VpcCidr** 的一部分。
8. **${ROUTE_TABLE_PVT}** 是 **${VPC_ID}** 中创建的私有路由表 ID，用于关联 Outposts 上的专用子网。指定用于关联此堆栈创建的 Outpost 子网的专用路由表。
9. **${SUBNET_CIDR_PVT}** 是一个有效的 CIDR 块，用于创建专用子网。这个块必须是 VPC CIDR 块 **VpcCidr** 的一部分。
10. **${OUTPOST_ARN}** 是 Outpost 的 Amazon 资源名称 (ARN)。

输出示例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-820e-11eb-2fd3-12a48460849f
```

-

验证

- 运行以下命令确认模板组件已存在：

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

在 **StackStatus** 显示 **CREATE_COMPLETE** 后，输出会显示以下参数的值：

PublicSubnetId	由 CloudFormation 堆栈创建的公共子网的 ID。
PrivateSubnetId	由 CloudFormation 堆栈创建的专用子网的 ID。

确保将这些参数值提供给您为集群创建的其他 CloudFormation 模板。

22.3.3. VPC 子网的 CloudFormation 模板

您可以使用以下 CloudFormation 模板来部署 Outpost 子网。

例 22.1. VPC 子网的 CloudFormation 模板

```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Best Practice Subnets (Public and Private)

Parameters:
  VpcId:
    Description: VPC ID that comprises all the target subnets.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
  ClusterName:
    Description: Cluster name or prefix name to prepend the Name tag for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.
  ZoneName:
    Description: Zone Name to create the subnets, such as us-west-2-lax-1a.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ZoneName parameter must be specified.
  PublicRouteTableId:
    Description: Public Route Table ID to associate the public subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PublicRouteTableId parameter must be specified.
  PublicSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for public subnet.
```

```
    Type: String
  PrivateRouteTableId:
    Description: Private Route Table ID to associate the private subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PrivateRouteTableId parameter must be specified.
  PrivateSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for private subnet.
    Type: String
  PrivateSubnetLabel:
    Default: "private"
    Description: Subnet label to be added when building the subnet name.
    Type: String
  PublicSubnetLabel:
    Default: "public"
    Description: Subnet label to be added when building the subnet name.
    Type: String
  OutpostArn:
    Default: ""
    Description: OutpostArn when creating subnets on AWS Outpost.
    Type: String

Conditions:
  OutpostEnabled: !Not [!Equals [!Ref "OutpostArn", ""]]

Resources:
  PublicSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PublicSubnetCidr
      AvailabilityZone: !Ref ZoneName
      OutpostArn: !If [ OutpostEnabled, !Ref OutpostArn, !Ref "AWS::NoValue"]
      Tags:
      - Key: Name
        Value: !Join ['-', [ !Ref ClusterName, !Ref PublicSubnetLabel, !Ref ZoneName]]
      - Key: kubernetes.io/cluster/unmanaged (1)
        Value: true

  PublicSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTableId

  PrivateSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PrivateSubnetCidr
      AvailabilityZone: !Ref ZoneName
      OutpostArn: !If [ OutpostEnabled, !Ref OutpostArn, !Ref "AWS::NoValue"]
```

```
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, !Ref PrivateSubnetLabel, !Ref ZoneName]]
      - Key: kubernetes.io/cluster/unmanaged 2
        Value: true

  PrivateSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnet
      RouteTableId: !Ref PrivateRouteTableId

Outputs:
  PublicSubnetId:
    Description: Subnet ID of the public subnets.
    Value:
      !Join ["", [!Ref PublicSubnet]]

  PrivateSubnetId:
    Description: Subnet ID of the private subnets.
    Value:
      !Join ["", [!Ref PrivateSubnet]]
```

1 您必须在 AWS Outposts 的公共子网配置中包含 **kubernetes.io/cluster/unmanaged** 标签。

2 您必须在 AWS Outposts 的专用子网配置中包含 **kubernetes.io/cluster/unmanaged** 标签。

22.4. 创建在 OUTPOST 上部署边缘计算机器的计算机器集

要在 AWS Outposts 上创建边缘计算机器，您必须使用兼容配置创建新的计算机器集。

先决条件

- 您有一个 AWS Outposts 站点。
- 您已将 OpenShift Container Platform 集群安装到 AWS 上的自定义 VPC 中。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 运行以下命令列出集群中的计算机器集：

```
$ oc get machinesets.machine.openshift.io -n openshift-machine-api
```

输**出示例**

```
NAME                        DESIRED  CURRENT  READY  AVAILABLE  AGE
<original_machine_set_name_1>  1        1        1      1          55m
<original_machine_set_name_2>  1        1        1      1          55m
```

2. 记录现有计算机器集的名称。

3. 使用以下方法之一创建一个包含新计算机器设置自定义资源 (CR) 的 YAML 文件：

   - 运行以下命令，将现有计算机器集配置复制到新文件中：

     ```
     $ oc get machinesets.machine.openshift.io <original_machine_set_name_1> \
       -n openshift-machine-api -o yaml > <new_machine_set_name_1>.yaml
     ```

     您可以使用首选文本编辑器编辑此 YAML 文件。

   - 使用您的首选文本编辑器创建名为 **<new_machine_set_name_1>.yaml** 的空白 YAML 文件，并包含新计算机器集所需的值。
     如果您不确定为特定字段设置哪个值，您可以通过运行以下命令来查看现有计算机器集 CR 的值：

     ```
     $ oc get machinesets.machine.openshift.io <original_machine_set_name_1> \
       -n openshift-machine-api -o yaml
     ```

     输出示例

     ```
     apiVersion: machine.openshift.io/v1beta1
     kind: MachineSet
     metadata:
       labels:
         machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
       name: <infrastructure_id>-<role>-<availability_zone> 2
       namespace: openshift-machine-api
     spec:
       replicas: 1
       selector:
         matchLabels:
           machine.openshift.io/cluster-api-cluster: <infrastructure_id>
           machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-
     <availability_zone>
       template:
         metadata:
           labels:
             machine.openshift.io/cluster-api-cluster: <infrastructure_id>
             machine.openshift.io/cluster-api-machine-role: <role>
             machine.openshift.io/cluster-api-machine-type: <role>
             machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-
     <availability_zone>
         spec:
           providerSpec: 3
     # ...
     ```

     1 集群基础架构 ID。

     2 默认节点标签。对于 AWS Outposts，您可以使用 **outposts** 角色。

     3 省略的 **providerSpec** 部分包括必须为 Outpost 配置的值。

4. 通过编辑 **<new_machine_set_name_1>.yaml** 文件，将新计算机器集配置为在 Outpost 中创建边缘计算机器：

AWS Outposts 的计算机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-outposts-<availability_zone> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-outposts-
<availability_zone>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: outposts
        machine.openshift.io/cluster-api-machine-type: outposts
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-outposts-
<availability_zone>
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/outposts: ""
          location: outposts
      providerSpec:
        value:
          ami:
            id: <ami_id> 3
          apiVersion: machine.openshift.io/v1beta1
          blockDevices:
            - ebs:
                volumeSize: 120
                volumeType: gp2 4
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: <infrastructure_id>-worker-profile
          instanceType: m5.xlarge 5
          kind: AWSMachineProviderConfig
          placement:
            availabilityZone: <availability_zone>
            region: <region> 6
          securityGroups:
            - filters:
                - name: tag:Name
                  values:
```

```
            - <infrastructure_id>-worker-sg
        subnet:
          id: <subnet_id> 7
        tags:
          - name: kubernetes.io/cluster/<infrastructure_id>
            value: owned
        userDataSecret:
          name: worker-user-data
      taints: 8
        - key: node-role.kubernetes.io/outposts
          effect: NoSchedule
```

1. 指定集群基础架构 ID。
2. 指定计算机器设置的名称。名称由集群基础架构 ID、**outposts** 角色名称和 Outpost 可用区组成。
3. 指定 Amazon Machine Image (AMI) ID。
4. 指定 EBS 卷类型。AWS Outposts 需要 gp2 卷。
5. 指定 AWS 实例类型。您必须使用 Outpost 中配置的实例类型。
6. 指定 Outpost 可用区存在的 AWS 区域。
7. 指定 Outpost 的专用子网。
8. 指定污点，以防止将工作负载调度到具有 **node-role.kubernetes.io/outposts** 标签的节点上。要在 Outpost 中调度用户工作负载，您必须在应用程序的 **Deployment** 资源中指定对应的容限。

5. 保存您的更改。
6. 运行以下命令来创建计算机器设置 CR：

```
$ oc create -f <new_machine_set_name_1>.yaml
```

验证

- 要验证是否已创建计算机器设置，请运行以下命令列出集群中的计算机器集：

```
$ oc get machinesets.machine.openshift.io -n openshift-machine-api
```

输出示例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<new_machine_set_name_1>    1         1         1       1           4m12s
<original_machine_set_name_1>  1      1         1       1           55m
<original_machine_set_name_2>  1      1         1       1           55m
```

- 要列出由新计算机器集管理的机器，请运行以下命令：

```
$ oc get -n openshift-machine-api machines.machine.openshift.io \
  -l machine.openshift.io/cluster-api-machineset=<new_machine_set_name_1>
```

输**出示例**

```
NAME                      PHASE        TYPE       REGION      ZONE         AGE
<machine_from_new_1>      Provisioned  m5.xlarge  us-east-1   us-east-1a   25s
<machine_from_new_2>      Provisioning m5.xlarge  us-east-1   us-east-1a   25s
```

- 要验证由新计算机器集创建的机器是否具有正确的配置，请运行以下命令检查 CR 中的相关字段是否有新机器：

```
$ oc describe machine <machine_from_new_1> -n openshift-machine-api
```

22.5. 在 OUTPOST 中创建用户工作负载

将 AWS VPC 集群中的 OpenShift Container Platform 扩展为 Outpost 后，您可以使用带有标签 **node-role.kubernetes.io/outposts** 的边缘计算节点在 Outpost 中创建用户工作负载。

先决条件

- 您已将 AWS VPC 集群扩展到 Outpost。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 您已创建了部署与 Outpost 环境兼容的边缘计算机器的计算机器集。

流程

1. 为您要部署到边缘子网中边缘计算节点的应用程序配置 **Deployment** 资源文件。

Deployment 清单示例

```
kind: Namespace
apiVersion: v1
metadata:
  name: <application_name> 1
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <application_name>
  namespace: <application_namespace> 2
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: gp2-csi 3
  volumeMode: Filesystem
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  selector:
    matchLabels:
      app: <application_name>
  replicas: 1
  template:
    metadata:
      labels:
        app: <application_name>
        location: outposts 4
    spec:
      securityContext:
        seccompProfile:
          type: RuntimeDefault
      nodeSelector: 5
        node-role.kubernetes.io/outpost: ''
      tolerations: 6
      - key: "node-role.kubernetes.io/outposts"
        operator: "Equal"
        value: ""
        effect: "NoSchedule"
      containers:
        - image: openshift/origin-node
          command:
           - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
          imagePullPolicy: Always
          name: <application_name>
          ports:
            - containerPort: 8080
          volumeMounts:
            - mountPath: "/mnt/storage"
              name: data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: <application_name>
```

1. 为应用程序指定一个名称。
2. 为应用程序指定一个命名空间。应用程序命名空间可以与应用程序名称相同。
3. 指定存储类名称。对于边缘计算配置，必须使用 **gp2-csi** 存储类。
4. 指定标签来标识 Outpost 中部署的工作负载。
5. 指定以边缘计算节点为目标的节点选择器标签。

6 指定与边缘计算机器计算机器集中的 **key** 和 **effects** 污点匹配的容限。设置 **value** 和 **operator**，如下所示。

2. 运行以下命令来创建 **Deployment** 资源：

```
$ oc create -f <application_deployment>.yaml
```

3. 配置 **Service** 对象，将 pod 从目标边缘计算节点公开到在边缘网络中运行的服务。

Service 清单示例

```
apiVersion: v1
kind: Service 1
metadata:
  name:  <application_name>
  namespace: <application_namespace>
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector: 2
    app: <application_name>
```

1 定义 **service** 资源。

2 指定要应用到受管 pod 的标签类型。

4. 运行以下命令来创建 **Service** CR：

```
$ oc create -f <application_service>.yaml
```

22.6. 在边缘和基于云的 AWS 计算资源上调度工作负载

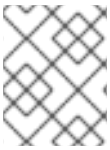
当您将 AWS VPC 集群扩展到 Outpost 时，Outpost 使用边缘计算节点，并且 VPC 使用基于云的计算节点。以下负载均衡器注意事项适用于扩展到 Outpost 中的 AWS VPC 集群：

- Outposts 无法运行 AWS Network Load Balancers 或 AWS Classic Load Balancers，但扩展到 Outpost 的 VPC 集群的 Classic Load Balancer 可以附加到 Outpost 边缘计算节点。如需更多信息，请参阅在 AWS VPC 集群中使用 AWS Classic Load Balancers 扩展到 Outpost 。
- 要在 Outpost 实例上运行负载均衡器，您必须使用 AWS Application Load Balancer。您可以使用 AWS Load Balancer Operator 部署 AWS Load Balancer Controller 的实例。控制器为 Kubernetes Ingress 资源置备 AWS Application Load Balancers。如需更多信息，请参阅在 AWS VPC 集群中使用 AWS Load Balancer Operator 扩展到 Outpost。

22.6.1. 在 AWS VPC 集群中使用 AWS Classic Load Balancers 扩展至 Outpost

AWS Outposts 基础架构无法运行 AWS Classic Load Balancers，但 AWS VPC 集群中的 Classic Load Balancers 可以在 Outpost 中目标边缘计算节点（如果边缘和基于云的子网位于同一可用区中）。因此，VPC 集群上的 Classic Load Balancers 可能会在其中一个节点类型上调度 pod。

在边缘计算节点和基于云的计算节点上调度工作负载可能会带来延迟。如果要防止 VPC 集群中的 Classic Load Balancer 针对 Outpost 边缘计算节点，您可以将标签应用到基于云的计算节点，并将 Classic Load Balancer 配置为仅在带有应用的标签的节点上调度。



注意

如果您不需要防止 VPC 集群中的 Classic Load Balancer 以 Outpost 边缘计算节点为目标，则不需要完成这些步骤。

先决条件

- 您已将 AWS VPC 集群扩展到 Outpost。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 您已在 Outpost 中创建用户工作负载，其容限与边缘计算机器的污点匹配。

流程

1. 可选：运行以下命令来验证边缘计算节点是否具有 **location=outposts** 标签，并验证输出是否只包含 Outpost 中的边缘计算节点：

```
$ oc get nodes -l location=outposts
```

2. 运行以下命令，使用键值对标记 VPC 集群中的基于云的计算节点：

```
$ for NODE in $(oc get node -l node-role.kubernetes.io/worker --no-headers | grep -v outposts | awk '{print$1}'); do oc label node $NODE <key_name>=<value>; done
```

其中 **<key_name>=<value>** 是您要用来区分基于云的计算节点的标签。

输**出示例**

```
node1.example.com labeled
node2.example.com labeled
node3.example.com labeled
```

3. 可选：运行以下命令来验证基于云的计算节点是否具有指定的标签，并确认输出是否包含 VPC 集群中的所有基于云的计算节点：

```
$ oc get nodes -l <key_name>=<value>
```

输**出示例**

```
NAME                STATUS   ROLES    AGE     VERSION
node1.example.com   Ready    worker   7h      v1.28.5
node2.example.com   Ready    worker   7h      v1.28.5
node3.example.com   Ready    worker   7h      v1.28.5
```

4. 通过在 **Service** 清单的 **annotations** 字段中添加基于云的子网信息来配置 Classic Load Balancer 服务：

服务配置示例

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: <application_name>
  name: <application_name>
  namespace: <application_namespace>
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-subnets: <aws_subnet> (1)
    service.beta.kubernetes.io/aws-load-balancer-target-node-labels: <key_name>=<value>
(2)
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: <application_name>
  type: LoadBalancer
```

(1) 指定 AWS VPC 集群的子网 ID。

(2) 在节点标签中指定与密钥对匹配的键值对。

5. 运行以下命令来创建 **Service** CR：

```
$ oc create -f <file_name>.yaml
```

验证

1. 运行以下命令，验证 **service** 资源的状态，以显示置备的 AWS Load Balancer(ALB)的主机：

```
$ HOST=$(oc get service <application_name> -n <application_namespace> --
template='{{(index .status.loadBalancer.ingress 0).hostname}}')
```

2. 运行以下命令，验证置备的 Classic Load Balancer 主机的状态：

```
$ curl $HOST
```

3. 在 AWS 控制台中，验证是否只有标记的实例显示为负载均衡器的目标实例。

22.6.2. 在 AWS VPC 集群中使用 AWS Load Balancer Operator 扩展至 Outpost

您可以配置 AWS Load Balancer Operator，以便在 AWS VPC 集群中置备 AWS Application Load Balancer。AWS Outposts 不支持 AWS Network Load Balancers。因此，AWS Load Balancer Operator 无法在 Outpost 中置备 Network Load Balancers。

您可以在云子网或 Outpost 子网中创建 AWS Application Load Balancer。云中的 Application Load Balancer 可以附加到基于云的计算节点，而 Outpost 中的 Application Load Balancer 可以附加到边缘计算节点。您必须使用 Outpost 子网或 VPC 子网来注解 Ingress 资源，但不能同时注解两者。

先决条件

- 您已将 AWS VPC 集群扩展到 Outpost。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 AWS Load Balancer Operator 并创建了 AWS Load Balancer Controller。

流程

- 将 **Ingress** 资源配置为使用指定的子网：

  Ingress 资源配置示例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <application_name>
  annotations:
    alb.ingress.kubernetes.io/subnets: <subnet_id> 1
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <application_name>
                port:
                  number: 80
```

  1 指定要使用的子网。

    - 要在 Outpost 中使用 Application Load Balancer，请指定 Outpost 子网 ID。
    - 要在云中使用 Application Load Balancer，您必须在不同的可用区中指定至少两个子网。

其他资源

- 使用 AWS Load Balancer Operator 创建 AWS Load Balancer Controller 实例

22.7. 其他资源

- 在 AWS 上将集群安装到现有的 VPC 中