OpenShift Container Platform 4.8

安装后配置

OpenShift Container Platform 的第二天操作

Last Updated: 2023-07-28

OpenShift Container Platform 4.8 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
- 集群
- 节点
- Network
- 存储
- 用户
- 警报和通知

1.1. 安装后要执行的配置任务

集群管理员可执行以下安装后配置任务：

- 配置操作系统功能 ：Machine Config Operator(MCO)管理 **MachineConfig** 对象。通过使用 MCO，您可以在 OpenShift Container Platform 集群中执行以下任务：
  - 使用 **MachineConfig** 对象配置节点
  - 配置 MCO 相关的自定义资源
- 配置集群功能：作为集群管理员，您可以修改 OpenShift Container Platform 集群的主要功能的配置资源。这些功能包括：
  - 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
- 将集群组件配置为私有 ：默认情况下，安装程序使用公开的 DNS 和端点置备 OpenShift Container Platform。如果您希望集群只能从内部网络内部访问，请将以下组件配置为私有：
  - DNS
  - Ingress Controller
  - API Server

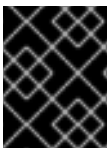
- 执行节点操作 ：默认情况下，OpenShift Container Platform 使用 Red Hat Enterprise Linux CoreOS(RHCOS)计算机器。作为集群管理员，您可以使用 OpenShift Container Platform 集群中的机器执行以下操作：
  - 添加和删除计算机器
  - 为节点添加和删除污点和容限
  - 配置每个节点的最大 pod 数量
  - 启用设备管理器
- 配置网络 ：安装 OpenShift Container Platform 后，您可以配置以下内容：
  - 入口集群流量
  - 节点端口服务范围
  - 网络策略
  - 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。TO 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。作为集群管理员，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
- 管理警报和通知 ：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 4.8 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。您可以将默认 Ingress Controller 替换为内部控制器。

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

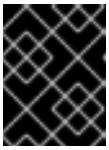
- 安装 OpenShift CLI（**oc**）。
- 使用具有 **admin** 权限的用户登陆到 web 控制台。

流程

1. 在 AWS 或 Azure 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，删除负载均衡器的 **api-internal** 规则。

b. 在公共区中删除 **api.$clustername.$yourdomain** DNS 条目。

2. 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。

a. 在终端中列出集群机器：

```
$ 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
```

在以下步骤中，修改 control plane 机器（名称中包含 **master** 的机器）。

b. 从每台 control plane 机器移除外部负载均衡器。

i. 编辑 control plane **Machine** 对象以移除对外部负载均衡器的引用：

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

1 指定要修改的 control plane 或 master **Machine** 对象的名称。

ii. 删除描述外部负载均衡器的行（在以下示例中已被标记），保存并退出对象规格：

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

1 2 删除这一行。

iii. 对名称中包含 **master** 的每个机器重复这个过程。

第 3 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。

3.1. 了解 MACHINE CONFIG OPERATOR

3.1.1. 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。

project
openshift-machine-config-operator

3.1.2. 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（也称为 master 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。

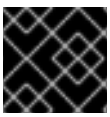
- 在将 OpenShift Container Platform 安装到磁盘前，必须先进行一些机器配置。在大多数情况下，这可以通过创建直接注入 OpenShift Container Platform 安装程序进程中的机器配置来实现，而不必作为安装后机器配置运行。在其他情况下，您可能需要在 OpenShift Container Platform 安装程序启动时传递内核参数时进行裸机安装，以完成诸如设置每个节点的 IP 地址或高级磁盘分区等操作。
- MCO 管理机器配置中设置的项目。MCO 不会覆盖您对系统进行的手动更改，除非明确告知 MCO 管理冲突文件。换句话说，MCO 只提供您请求的特定更新，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。

3.1.2.1. 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** 用户才支持通过机器配置更改 SSH 密钥。

- **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。
- **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

只有在 **x86_64** 架构中的 OpenShift Container Platform 部署支持 FIPS 验证的/Modules in Process 加密库。

- **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。

3.1.2.2. project

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

3.1.3. 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。

3.2. 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。

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。

3.2.1. 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: 4.8.0
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

3.2.2. 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：

- **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。
- **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。

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:
  config:
    ignition:
      version: 3.2.0
  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   v1.21.0
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.21.0
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.21.0
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.21.0
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.21.0
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.21.0
```

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。

3.2.3. 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 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 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

- 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。
- 您以具有管理特权的用户身份登录集群。

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   v1.20.0
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.20.0
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.20.0
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.20.0
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.20.0
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.20.0
```

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。

3.2.4. 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 4.8，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  v1.21.0
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.21.0
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.21.0
```

```
$ 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
```

3.2.5. 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。

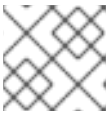
此流程描述了如何修改 **/etc/systemd/journald.conf** 文件中的 **journald** 限制设置并将其应用到 worker 节点。有关如何使用该文件的详情，请查看 **journald.conf** 手册页。

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: 4.8.0
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

3.2.6. 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，提供了以下扩展程序：

- **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。

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  v1.18.3
```

```
$ 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
```

3.2.7. 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: 4.9.0
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

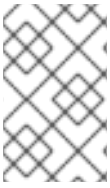
- 使用 Butane 创建机器配置

3.3. 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。

3.3.1. 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

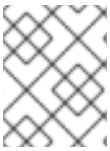
因为 **kubeletConfig** 对象中的字段直接从上游 Kubernetes 传递给 kubelet，kubelet 会直接验证这些值。**kubeletConfig** 对象中的无效值可能会导致集群节点不可用。有关有效值，请参阅 Kubernetes 文档。

请考虑以下指导：

- 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。如果要将相同的内容应用到所有池，则所有池仅需要一个 **KubeletConfig** CR。
- 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。

- 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**。

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：

```
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=large-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
```

这应该会显示 **status: "True"** 和 **type:Success**：

```
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
```

3.3.2. 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**：**pidsLimit** 参数设置 CRI-O **pids_limit** 参数，这是容器中允许的最大进程数。默认为 1024（**pids_limit = 1024**）。
- **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**：**logSizeMax** 参数设置 CRI-O **log_size_max** 参数，这是容器日志文件允许的最大值。默认为没有限制（**log_size_max = -1**）。如果设置为正数，则必须至少小于 ConMon 读取缓冲的 8192。conMon 是一个监控单个容器管理器（如 Podman 或 CRI-O）与 OCI 运行时

（如 runc 或 crun）之间的通信的程序。

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 **oneContainerRuntimeConfig** 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-pid     24m
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 **pids_limit** 增加到 2048，将 **log_level** 设置为 **debug**，将覆盖大小设置为 8 GB，并将 **log_size_max** 设置为无限：

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:
   pidsLimit: 2048 (2)
   logLevel: debug (3)
   overlaySize: 8G (4)
   logSizeMax: "-1" (5)
```

1. 指定机器配置池标签。
2. 可选：指定容器中允许的最大进程数。
3. 可选：指定日志消息的详细程度。
4. 可选：指定容器镜像的最大大小。
5. 可选：指定容器日志文件允许的最大大小。如果设置为正数，则必须至少为 8192。

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)
   pidsLimit: 2048
   logLevel: debug
   overlaySize: 8G
   logSizeMax: "-1"
```

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 | egrep 'log_level|pids_limit|log_size_max'
```

输出示例

```
pids_limit = 2048
log_size_max = -1
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"
```

3.3.3. 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，如日志级别和 PID 限制，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:
   pidsLimit: 2048
   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% /
```

第 4 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。

4.1. 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**

4.1.1. 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，如策略和默认节点选择器。

4.1.2. 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	配置内部镜像 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。

4.1.3. 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。

4.1.4. 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 4.8 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，请按照流程在安装过程中自定义网络。

4.2. 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。

4.3. 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，可以通过创建一个或多个新机器集来调整它们，先对它们进行扩展，并缩小原始的机器集，然后再删除它们。

4.3.1. 了解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。

4.3.2. 手动扩展机器集

要在机器集中添加或删除机器实例，您可以手动扩展机器集。

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. 查看集群中的机器集：

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

机器集以 **<clusterid>-worker-<aws-region-az>** 的形式列出。

2. 查看集群中的机器：

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

3. 在您要删除的机器上设置注解：

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

4. 进行 cordon 操作，排空您要删除的节点：

```
$ oc adm cordon <node_name>
$ oc adm drain <node_name>
```

5. 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
```

提示

您还可以应用以下 YAML 来扩展机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

您可以扩展或缩减机器集。需要过几分钟以后新机器才可用。

验证

- 验证删除预期的机器：

```
$ oc get machines
```

4.3.3. 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**，表示在扩展机器时随机选择并删除机器。通过修改特定机器集，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/cluster-api-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 机器集扩展为 **0**。

注意

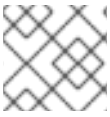
当用户需要特定的服务必须运行在特定节点，在 worker 机器集进行缩减时需要忽略这些服务时，可以使用自定义机器集。这可防止服务被中断。

4.3.4. 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。您可为节点、机器集或机器配置添加标签。将标签添加到机器集可确保节点或机器停机时，新节点具有标签。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
policy:
  name: ""
```

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. 通过使用机器集或直接编辑节点，为节点添加标签：

- 在创建节点时，使用机器集向由机器集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   v1.18.3+002a51f
```

- 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   v1.18.3+002a51f
```

4.4. 为生产环境创建基础架构机器集

您可以创建一个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，建议您至少部署三个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。这样的配置需要三个不同的机器集，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、post_installation_configuration/cluster-tasks.adoc#creating-an-infra-node_post-install-cluster-tasks[assign a label to the nodes]，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。

4.4.1. 创建机器集

除了安装程序创建的机器集之外，还可创建自己的机器集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。

   a. 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机器集：

   ```
   $ 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. 检查特定机器集的值：

   ```
   $ oc get machineset <machineset_name> -n \
     openshift-machine-api -o yaml
   ```

   输出示例

   ```
   ...
   template:
     metadata:
       labels:
         machine.openshift.io/cluster-api-cluster: agl030519-vplxk 1
         machine.openshift.io/cluster-api-machine-role: worker 2
         machine.openshift.io/cluster-api-machine-type: worker
         machine.openshift.io/cluster-api-machineset: agl030519-vplxk-worker-us-east-1a
   ```

   1 集群 ID。

   2 默认节点标签。

2. 创建新的 **MachineSet** CR:

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

3. 查看机器集列表：

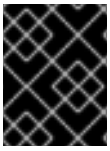
   ```
   $ oc get machineset -n openshift-machine-api
   ```

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
```

当新机器集可用时，**DESIRED** 和 **CURRENT** 的值会匹配。如果机器集不可用，请等待几分钟，然后再次运行命令。

4.4.2. 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 control plane 节点（也称为 master 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=""
```

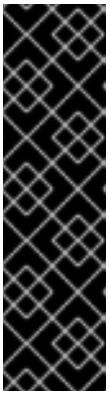
2. 向您要充当基础架构节点的 worker 节点添加标签：

```
$ 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: topology.kubernetes.io/region=us-east-1 1
...
```

1 默认情况下，此节点选择器示例将容器集部署到 **us-east-1** 区域的节点。

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。

4.4.3. 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。

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。

4.5. 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。

4.5.1. 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>:<effect>
   ```

   例如：

   ```
   $ oc adm taint nodes node1 node-role.kubernetes.io/infra: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
     ...
   ```

   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

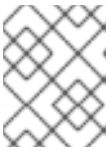
   如果使用 descheduler，则违反了节点污点的 pod 可能会从集群驱除。

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)
    operator: Exists (3)
```

1. 指定添加到节点的效果。
2. 指定添加到节点的键。
3. 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。

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。

4.6. 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。

4.6.1. 移动路由器

您可以将路由器 pod 部署到不同的机器集中。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   v1.21.0
```

由于角色列表包含 **infra**，因此 Pod 在正确的节点上运行。

4.6.2. 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。

4.6.3. 移动监控解决方案

监控堆栈包含多个组件，包括 Prometheus、Grafana 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
    grafana:
      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
```

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。

4.6.4. 移动 OpenShift Logging 资源

您可以配置 Cluster Logging Operator，将 OpenShift Logging 组件（如 Elasticsearch 和 Kibana）的 pod 部署到不同的节点上。您无法将 Cluster Logging Operator Pod 从其安装位置移走。

例如，您可以因为 CPU、内存和磁盘要求较高而将 Elasticsearch Pod 移到一个单独的节点上。

先决条件

- 必须安装 OpenShift Logging 和 Elasticsearch。默认情况下没有安装这些功能。

流程

1. 编辑 **openshift-logging** 项目中的 **ClusterLogging** 自定义资源（CR）：

```
$ oc edit ClusterLogging instance
```

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:
  collection:
    logs:
      fluentd:
        resources: null
      type: fluentd
  logStore:
    elasticsearch:
      nodeCount: 3
      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
      redundancyPolicy: SingleRedundancy
      resources:
        limits:
          cpu: 500m
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
```

```
      storage: {}
    type: elasticsearch
  managementState: Managed
  visualization:
    kibana:
      nodeSelector: 2
        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
      proxy:
        resources: null
      replicas: 1
      resources: null
    type: kibana

...
```

1 2 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。

验证

要验证组件是否已移动，您可以使用 **oc get pod -o wide** 命令。

例如：

- 您需要移动来自 **ip-10-0-147-79.us-east-2.compute.internal** 节点上的 Kibana pod：

```
$ oc get pod kibana-5b8bdf44f9-ccpq9 -o wide
```

输出示例

```
NAME                     READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
kibana-5b8bdf44f9-ccpq9  2/2     Running   0          27s   10.129.2.18   ip-10-0-147-79.us-
east-2.compute.internal   <none>           <none>
```

- 您需要将 Kibana pod 移到 **ip-10-0-139-48.us-east-2.compute.internal** 节点，该节点是一个专用的基础架构节点：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS   ROLES        AGE   VERSION
ip-10-0-133-216.us-east-2.compute.internal   Ready    master       60m   v1.21.0
ip-10-0-139-146.us-east-2.compute.internal   Ready    master       60m   v1.21.0
ip-10-0-139-192.us-east-2.compute.internal   Ready    worker       51m   v1.21.0
```

```
ip-10-0-139-241.us-east-2.compute.internal   Ready    worker       51m   v1.21.0
ip-10-0-147-79.us-east-2.compute.internal    Ready    worker       51m   v1.21.0
ip-10-0-152-241.us-east-2.compute.internal   Ready    master       60m   v1.21.0
ip-10-0-139-48.us-east-2.compute.internal    Ready    infra        51m   v1.21.0
```

请注意，该节点具有 **node-role.kubernetes.io/infra: ''** label:

```
$ oc get node ip-10-0-139-48.us-east-2.compute.internal -o yaml
```

输出示例

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-139-48.us-east-2.compute.internal
  selfLink: /api/v1/nodes/ip-10-0-139-48.us-east-2.compute.internal
  uid: 62038aa9-661f-41d7-ba93-b5f1b6ef8751
  resourceVersion: '39083'
  creationTimestamp: '2020-04-13T19:07:55Z'
  labels:
    node-role.kubernetes.io/infra: ''
...
```

- 要移动 Kibana pod，编辑 **ClusterLogging** CR 以添加节点选择器：

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:

...

  visualization:
    kibana:
      nodeSelector: 1
        node-role.kubernetes.io/infra: ''
      proxy:
        resources: null
      replicas: 1
      resources: null
    type: kibana
```

1 添加节点选择器以匹配节点规格中的 label。

- 保存 CR 后，当前 Kibana Pod 将被终止，新的 Pod 会被部署：

```
$ oc get pods
```

输出示例

```
NAME                                  READY   STATUS      RESTARTS   AGE
```

```
cluster-logging-operator-84d98649c4-zb9g7       1/1     Running       0          29m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg   2/2     Running       0          28m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj   2/2     Running       0          28m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78    2/2     Running       0          28m
fluentd-42dzz                                   1/1     Running       0          28m
fluentd-d74rq                                   1/1     Running       0          28m
fluentd-m5vr9                                   1/1     Running       0          28m
fluentd-nkxl7                                   1/1     Running       0          28m
fluentd-pdvqb                                   1/1     Running       0          28m
fluentd-tflh6                                   1/1     Running       0          28m
kibana-5b8bdf44f9-ccpq9                         2/2     Terminating   0          4m11s
kibana-7d85dcffc8-bfpfp                         2/2     Running       0          33s
```

- 新 pod 位于 **ip-10-0-139-48.us-east-2.compute.internal** 节点上 :

```
$ oc get pod kibana-7d85dcffc8-bfpfp -o wide
```

输出示例

```
NAME                     READY   STATUS       RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
kibana-7d85dcffc8-bfpfp   2/2     Running      0          43s   10.131.0.22   ip-10-0-139-48.us-
east-2.compute.internal   <none>           <none>
```

- 片刻后，原始 Kibana Pod 将被删除。

```
$ oc get pods
```

输出示例

```
NAME                                            READY   STATUS    RESTARTS   AGE
cluster-logging-operator-84d98649c4-zb9g7       1/1     Running   0          30m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg   2/2     Running   0          29m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj   2/2     Running   0          29m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78    2/2     Running   0          29m
fluentd-42dzz                                   1/1     Running   0          29m
fluentd-d74rq                                   1/1     Running   0          29m
fluentd-m5vr9                                   1/1     Running   0          29m
fluentd-nkxl7                                   1/1     Running   0          29m
fluentd-pdvqb                                   1/1     Running   0          29m
fluentd-tflh6                                   1/1     Running   0          29m
kibana-7d85dcffc8-bfpfp                         2/2     Running   0          62s
```

4.7. 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：

- 节点上运行的所有 pod 的 CPU 和内存请求总和小于节点上分配资源的 50%。
- 集群自动扩展可以将节点上运行的所有 pod 移到其他节点。
- 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。

4.7.1. ClusterAutoscaler 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
  scaleDown: (10)
    enabled: true (11)
    delayAfterAdd: 10m (12)
    delayAfterDelete: 5m (13)
    delayAfterFailure: 30s (14)
    unneededTime: 5m (15)
```

(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 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**。

11 指定集群自动扩展是否可以删除不必要的节点。

12 （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**。

13 指定在最近*删除*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 **10s**。

14 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**。

15 指定要经过多长时间之后，不需要的节点才符合删除条件。如果不指定值，则使用默认值 **10m**。

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。

4.7.2. 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 **ClusterAutoscaler** 资源创建一个 YAML 文件，其中包含自定义的资源定义。
2. 在集群中创建资源：

```
$ oc create -f <filename>.yaml 1
```

1 **<filename>** 是您自定义的资源文件的名称。

4.8. 关于机器自动扩展

机器自动扩展会调整您在 OpenShift Container Platform 集群中部署的机器集中的 Machine 数量。您可以扩展默认 **worker** 机器集，以及您创建的其他机器集。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机器设置中。

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。

4.8.1. MachineAutoscaler 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机器集，请指定或注明要扩展的机器集的名称。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，集群自动扩展会将机器集缩减为零。

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，提供用于描述要扩展的现有机器集的值。

(5) **kind** 参数值始终为 **MachineSet**。

(6) **name** 值必须与现有机器集的名称匹配，如 **metadata.name** 参数值所示。

4.8.2. 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，其中包含自定义的资源定义。
2. 在集群中创建资源：

```
$ oc create -f <filename>.yaml 1
```

1 **<filename>** 是您自定义的资源文件的名称。

4.9. 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。

4.9.1. 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 激活以下任何功能集：

- **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。启用此功能集无法撤消并阻止次版本更新。不建议在生产环境集群中使用此功能集。

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。

此功能集启用了以下技术预览功能：

  - Azure Disk CSI Driver Operator。使用 Microsoft Azure Disk Storage 的 Container Storage Interface(CSI)驱动程序启用持久性卷(PV)置备。
  - VMware vSphere CSI Driver Operator。使用 Virtual Machine Disk(VMDK)卷的 Container Storage Interface(CSI)VMware vSphere 驱动程序启用持久性卷(PV)。
  - CSI 自动迁移。为支持的树内卷插件启用自动迁移，并将其对应的 Container Storage Interface(CSI)驱动程序。作为技术预览提供：
    - Amazon Web Services（AWS）Elastic Block Storage（EBS）
    - OpenStack Cinder

4.9.2. 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。

4.9.3. 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。

4.10. ETCD 任务

备份 etcd、启用或禁用 etcd 加密或清除 etcd 数据。

4.10.1. 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。这些密钥会每周进行轮转。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。

4.10.2. 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。

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. 把 **encryption** 项类型设置为 **aescbc**：

```
spec:
  encryption:
    type: aescbc (1)
```

(1) **aescbc** 类型表示 AES-CBC 使用 PKCS#7 padding 和 32 字节密钥来执行加密。

3. 保存文件以使改变生效。
   加密过程开始。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 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。

4.10.3. 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。

4.10.4. 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

仅保存单一 control plane 主机（也称为 master 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. 为 control plane 节点启动一个 debug 会话：

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

2. 将您的根目录改为 **/host** ：

```
sh-4.2# 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. 运行 **cluster-backup.sh** 脚本，输入保存备份的位置。

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
```

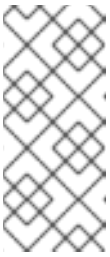
脚本输出示例

```
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。

4.10.5. 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_debugging_mvcc_db_total_size_in_bytes** 会显示数据库大小，包括等待碎片整理的空闲空间

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。

因为 etcd 将数据写入磁盘，所以其性能主要取决于磁盘性能。根据您的集群的具体情况，考虑每个月清理一次 etcd 碎片，或每个月清理两次。您还可以监控 **etcd_db_total_size_in_bytes** 指标，以确定是否需要进行碎片操作。

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 成员不会响应。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 get pods -n openshift-etcd -o wide | grep -v quorum-guard | grep etcd
```

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 |   3.4.9 |  104 MB |     false |      false |         7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |  104 MB |     false |      false |         7 |      91624 |              91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  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 |   3.4.9 |  104 MB |     false |      false |
7 |     91624 |           91624 |       |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |   41 MB |     false |      false |
7 |     91624 |           91624 |       | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  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

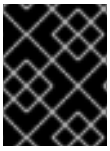
```
memberID:12345678912345678912 alarm:NOSPACE
```

b. 清除警告：

```
sh-4.4# etcdctl alarm disarm
```

4.10.6. 恢复到一个以前的集群状态

您可以使用已保存的 etcd 备份来恢复以前的集群状态，或者恢复丢失了大多数 control plane 主机（也称为 master 主机）的集群。

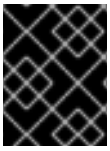


重要

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

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 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，Kubernetes API 服务器将无法访问，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

   不需要手动停止恢复主机上的 pod。恢复脚本将停止恢复主机上的 pod。

   a. 访问不是恢复主机的 control plane 主机。

   b. 将现有 etcd pod 文件从 Kubelet 清单目录中移出：

   ```
   $ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
   ```

   c. 验证 etcd pod 是否已停止。

   ```
   $ sudo crictl ps | grep etcd | grep -v operator
   ```

   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。

   d. 将现有 Kubernetes API 服务器 pod 文件移出 kubelet 清单目录中：

   ```
   $ sudo mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
   ```

   e. 验证 Kubernetes API 服务器 pod 是否已停止。

   ```
   $ sudo crictl ps | grep kube-apiserver | grep -v operator
   ```

   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。

   f. 将 etcd 数据目录移到不同的位置：

   ```
   $ sudo mv /var/lib/etcd/ /tmp
   ```

g. 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

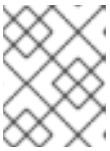
您可以通过查看 **oc get proxy cluster -o yaml** 的输出来检查代理是否已启用。如果 **httpProxy**、**httpsProxy**和 **noProxy** 字段设置了值，则会启用代理。

7. 在恢复 control plane 主机上运行恢复脚本，提供到 etcd 备份目录的路径：

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/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
```



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。

8. 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   v1.23.3+e419edf
host-172-25-75-38   Ready   infra,worker  3d20h   v1.23.3+e419edf
host-172-25-75-40   Ready   master        3d20h   v1.23.3+e419edf
host-172-25-75-65   Ready   master        3d20h   v1.23.3+e419edf
host-172-25-75-74   Ready   infra,worker  3d20h   v1.23.3+e419edf
host-172-25-75-79   Ready   worker        3d20h   v1.23.3+e419edf
host-172-25-75-86   Ready   worker        3d20h   v1.23.3+e419edf
host-172-25-75-98   Ready   infra,worker  3d20h   v1.23.3+e419edf
```

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
```

9. 在所有 control plane 主机上重启 kubelet 服务。

a. 在恢复主机中运行以下命令：

```
$ sudo systemctl restart kubelet.service
```

b. 在所有其他 control plane 主机上重复此步骤。

10. 批准待处理的 CSR：

a. 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（用于用户置备的安装）。

3 4 一个待处理的 **node-bootstrapper** CSR。

b. 查看一个 CSR 的详细信息以验证其是否有效：

```
$ oc describe csr <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

c. 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，请批准每个有效的 kubelet 服务 CSR：

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

11. 确认单个成员 control plane 已被成功启动。

a. 从恢复主机上，验证 etcd 容器是否正在运行。

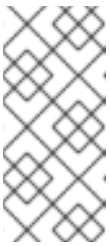
```
$ sudo crictl ps | grep etcd | grep -v operator
```

输出示例

```
3ad41b7908e32
36f86e2eeaaffe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running             etcd                                    0
7c05f8af362f0
```

b. 从恢复主机上，验证 etcd pod 是否正在运行。

```
$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep etcd
```



注意

如果您试图在运行这个命令前运行 **oc login** 并接收以下错误，请等待一些时间以便身份验证控制器启动并再次尝试。

```
Unable to connect to the server: EOF
```

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。

c. 对不是恢复主机的每个已丢失的 control plane 主机重复此步骤。

12. 逐个删除并重新创建其他非恢复 control plane 机器。重新创建此机器后，会强制一个新修订版本并自动扩展 etcd。
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。否则，您必须使用最初创建控制平面节点时使用的相同方法创建新的控制平面。

警告

不要为恢复主机删除和重新创建计算机。

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. 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. 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. 将 **metadata.name** 字段更改为新名称。
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. 删除 **spec.providerID** 字段：

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

iv. 删除 **metadata.annotations** 和 **metadata.generation** 字段：

```
annotations:
  machine.openshift.io/instance-state: running
```

```
...
generation: 2
```

v. 删除 **metadata.resourceVersion** 和 **metadata.uid** 字段：

```
resourceVersion: "13291"
uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

d. 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. 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. 使用 **new-master-machine.yaml** 文件创建新机器：

```
$ oc apply -f new-master-machine.yaml
```

g. 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。当机器或节点返回一个健康状态时，etcd cluster 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。

13. 在一个单独的终端窗口中，使用以下命令以具有 **cluster-admin** 角色的用户身份登录到集群：

```
$ oc login -u <cluster_admin> (1)
```

(1) 对于 **<cluster_admin>**，使用 **cluster-admin** 角色指定一个用户名。

14. 强制 etcd 重新部署。
在一个终端中使用 **cluster-admin** 用户连接到集群，运行以下命令：

```
$ 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 cluster 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。

15. 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，运行以下命令：

```
$ 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。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。

16. 在重新部署 etcd 后，为 control plane 强制进行新的 rollout。由于 kubelet 使用内部负载平衡器连接到 API 服务器，因此 Kubernetes API 将在其他节点上重新安装自己。
在一个终端中使用 **cluster-admin** 用户连接到集群，运行以下命令。

    a. 为 Kubernetes API 服务器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. 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
    ```

    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. 为 Kubernetes 调度程序强制进行新的推出部署：

```
$ oc patch kubescheduler 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 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。

17. 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 get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep etcd
```

输出示例

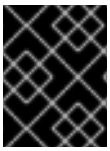
```
etcd-ip-10-0-143-125.ec2.internal        2/2     Running     0      9h
etcd-ip-10-0-154-194.ec2.internal        2/2     Running     0      9h
etcd-ip-10-0-173-171.ec2.internal        2/2     Running     0      9h
```

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 Kubernetes API 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。

请注意，在完成这个过程后，可能需要几分钟才能恢复所有服务。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。

4.10.7. 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*）。

4.11. POD 中断预算

了解并配置 pod 中断预算。

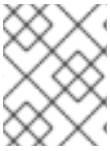
4.11.1. 了解如何使用 pod 中断预算来指定必须在线的 pod 数量

*pod 中断预算*是 Kubernetes API 的一部分，可以像其他对象类型一样通过 **oc** 命令进行管理。它们允许在操作过程中指定 pod 的安全约束，比如为维护而清空节点。

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

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。

您可以使用以下命令来检查所有项目的 pod 中断预算：

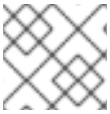
-

```
$ oc get poddisruptionbudget --all-namespaces
```

输出示例

```
NAMESPACE         NAME          MIN-AVAILABLE   SELECTOR
another-project   another-pdb   4               bar=foo
test-project      my-pdb        2               foo=bar
```

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。

4.11.2. 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:
      foo: bar
```

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。将此参数留空，如 **selector {}**，以选择项目中的所有 pod。

或者：

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
```

```
  selector: 3
    matchLabels:
      foo: bar
```

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。将此参数留空，如 **selector {}**，以选择项目中的所有 pod。

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>
```

4.12. 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。

4.12.1. 手动轮转云供应商凭证

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

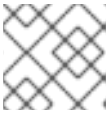
- 您的集群会在支持使用您要使用的 CCO 模式手动轮转云凭证的平台上安装：
  - 对于 mint 模式，支持 Amazon Web Services (AWS)和 Google Cloud Platform (GCP)。
  - 对于 passthrough 模式，支持 Amazon Web Services (AWS)、Microsoft Azure、Google Cloud Platform (GCP)、Red Hat OpenStack Platform (RHOSP)、Red Hat Virtualization (RHV)和 VMware vSphere。
- 您已更改了用于与云供应商接口的凭证。
- 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**
RHV	**ovirt-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
```

7. 如果集群的 CCO 配置为使用 mint 模式，请删除各个 **CredentialsRequest** 对象引用的每个组件 secret。
   a. 以具有 **cluster-admin** 角色的用户身份登录 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：

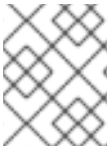
1. 在 web 控制台的 **Administrator** 视角中，导航到 **Workloads → Secrets**。
2. 验证 **Value** 字段的内容已改变。

其他资源

- vSphere CSI Driver Operator

4.12.2. 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

- 关于 Cloud Credential Operator
- Amazon Web Services(AWS)secret 格式
- Microsoft Azure secret 格式
- Google Cloud Platform(GCP)secret 格式

4.13. 为断开连接的集群配置镜像流

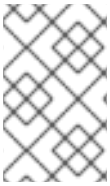
在断开连接的环境中安装 OpenShift Container Platform 后，为 Cluster Samples Operator 和 **must-gather** 镜像流配置镜像流。

4.13.1. 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。

4.13.2. 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。

重要

jenkins、**jenkins-agent-maven** 和 **jenkins-agent-nodejs** 镜像流的确来自安装有效负载，并由 Samples Operator 管理，因此这些镜像流不需要进一步的镜像操作。

将 Sample Operator 配置文件中的 **samplesRegistry** 字段设置为 registry.redhat.io 有很多冗余，因为它已经定向到 registry.redhat.io，只用于 Jenkins 镜像和镜像流。

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
```

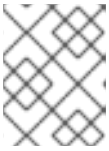
5. 更新 Cluster Samples Operator 配置对象中的 **samplesRegistry** 字段，使其包含镜像配置中定义的镜像位置的 **hostname** 部分：

```
$ 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。

4.13.3. 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)
```

4.14. 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 5 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。

5.1. 在 OPENSHIFT CONTAINER PLATFORM 集群中添加 RHEL 计算机器

理解并使用 RHEL 计算节点。

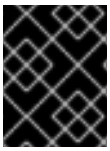
5.1.1. 关于在集群中添加 RHEL 计算节点

在 OpenShift Container Platform 4.8 中，如果使用用户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（也称为 worker）。集群中的 control plane 或主控机器（master）必须使用 Red Hat Enterprise Linux CoreOS (RHCOS)。

与所有使用用户置备的基础架构的安装一样，如果选择在集群中使用 RHEL 计算机器，您将需要负责所有操作系统生命周期的管理和维护任务，包括执行系统更新、应用补丁以及完成所有其他必要的任务。

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。



重要

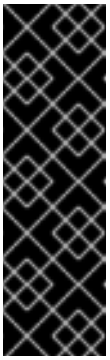
添加到 OpenShift Container Platform 集群的所有 RHEL 机器上都禁用内存交换功能。您无法在这些机器上启用交换内存。

您必须在初始化 control plane 之后将所有 RHEL 计算机器添加到集群。

5.1.2. RHEL 计算节点的系统要求

OpenShift Container Platform 环境中的 Red Hat Enterprise Linux (RHEL) 计算或 worker 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 7.9。



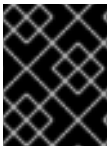
重要

在 OpenShift Container Platform 集群中添加 RHEL 7 计算机器已被弃用。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。

另外，您不能将计算机器升级到 RHEL 8，因为本发行版本中没有相关支持。

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 7 文档中的启用 FIPS 模式 。



重要

只有在 **x86_64** 架构中的 OpenShift Container Platform 部署支持 FIPS 验证的/Modules in Process 加密库。

- 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。

5.1.2.1. 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。

5.1.3. 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 4.8 集群之前，必须准备一个 RHEL 7 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. 确保机器上具有集群的 **kubeconfig** 文件，以及用于安装集群的安装程序。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 4.8 所需的存储库：

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ansible-2.9-rpms" \
    --enable="rhel-7-server-ose-4.8-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。

5.1.4. 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 4.8 需要的存储库：

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-fast-datapath-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-optional-rpms" \
    --enable="rhel-7-server-ose-4.8-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。

5.1.5. 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 4.8 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-rhel7-0.example.com
mycluster-rhel7-1.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。

5.1.6. 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。

5.1.7. 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。

5.2. 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。

5.2.1. 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，需要按照安装过程的说明获得这些文件。

5.2.2. 使用 ISO 镜像创建更多 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。

流程

1. 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。
2. 实例启动后，按 **TAB** 或 **E** 键编辑内核命令行。
3. 将参数添加到内核命令行：

```
coreos.inst.install_dev=sda 1
coreos.inst.ignition_url=http://example.com/worker.ign 2
```

1 指定要安装到的系统块设备。

2 指定计算 Ignition 配置文件的 URL。只支持 HTTP 和 HTTPS 协议。

4. 按 **Enter** 键完成安装。安装 RHCOS 后，系统会重启。系统重启后，它会应用您指定的 Ignition 配置文件。

5. 继续为集群创建更多计算机器。

5.2.3. 通过 PXE 或 iPXE 启动来创建更多 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。

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：

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
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
(1)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.<architecture>.img
(2)
```

(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.inst.ignition_url** 参数值是 worker Ignition 配置文件的位置，**coreos.live.rootfs_url** 参数值则是 live **rootfs** 文件的位置。**coreos.inst.ignition_url** 和 **coreos.live.rootfs_url** 参数仅支持 HTTP 和 HTTPS。

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

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 **内核参数** 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？

1. 使用 PXE 或 iPXE 基础架构为集群创建所需的计算机器。

5.2.4. 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

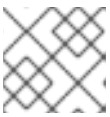
1. 确认集群可以识别这些机器：

```
$ oc get nodes
```

输出示例

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.21.0
master-1 Ready    master  63m  v1.21.0
master-2 Ready    master  64m  v1.21.0
```

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：

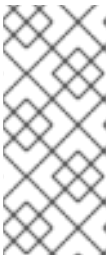
```
$ oc get csr
```

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  v1.21.0
master-1  Ready     master  73m  v1.21.0
master-2  Ready     master  74m  v1.21.0
worker-0  Ready     worker  11m  v1.21.0
worker-1  Ready     worker  11m  v1.21.0
```

注意

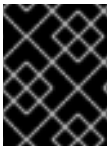
批准服务器 CSR 后可能需要几分钟时间让机器过渡到 **Ready 状态**。

其他信息

- 如需有关 CSR 的更多信息，请参阅证书签名请求。

5.3. 部署机器健康检查

理解并部署机器健康检查。



重要

此过程不适用于使用手动置备的机器的集群。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。

5.3.1. 关于机器健康检查

机器健康检查自动修复特定机器池中不健康的机器。

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。

注意

您不能对具有 master 角色的机器进行机器健康检查。

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。

例如，您应该在升级过程中停止检查，因为集群中的节点可能会临时不可用。**MachineHealthCheck** 可能会识别不健康的节点，并重新引导它们。为避免重新引导这样的节点，请在更新集群前删除您部署的任何 **MachineHealthCheck** 资源。但是，默认部署的 **MachineHealthCheck** 资源（如 **machine-api-termination-handler**）不能被删除并重新创建。

5.3.1.1. 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：

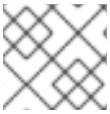
- 只有机器集拥有的机器才可以由机器健康检查修复。
- 目前不支持 control plane 机器，如果不健康，则不会被修复。
- 如果机器的节点从集群中移除，机器健康检查会认为机器不健康，并立即修复机器。
- 如果机器对应的节点在 **nodeStartupTimeout** 之后没有加入集群，则会修复机器。
- 如果 **Machine** 资源阶段为 **Failed**，则会立即修复机器。

5.3.2. 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。

5.3.2.1. 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机器集，以便在丢失整个区时，**maxUnhealthy** 设置可以在集群中阻止进一步补救。

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。

5.3.2.1.1. 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。

5.3.2.1.2. 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。

5.3.3. 创建 MachineHealthCheck 资源

您可以为集群中的所有 **MachineSet** 创建 **MachineHealthCheck** 资源。您不应该创建针对 control plane 机器的 **MachineHealthCheck** 资源。

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
```

5.3.4. 手动扩展机器集

要在机器集中添加或删除机器实例，您可以手动扩展机器集。

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. 查看集群中的机器集：

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

   机器集以 **<clusterid>-worker-<aws-region-az>** 的形式列出。

2. 查看集群中的机器：

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

3. 在您要删除的机器上设置注解：

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

4. 进行 cordon 操作，排空您要删除的节点：

   ```
   $ oc adm cordon <node_name>
   $ oc adm drain <node_name>
   ```

5. 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
   ```

提示

您还可以应用以下 YAML 来扩展机器集：

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

您可以扩展或缩减机器集。需要过几分钟以后新机器才可用。

验证

- 验证删除预期的机器：

```
$ oc get machines
```

5.3.5. 了解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。

5.4. 推荐的节点主机实践

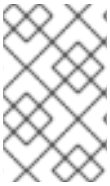
OpenShift Container Platform 节点配置文件包含重要的选项。例如，控制可以为节点调度的最大 pod 数量的两个参数: **podsPerCore** 和 **maxPods**。

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 根据节点中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。默认为 **0**。**podsPerCore** 不能超过 **maxPods**。

maxPods 把节点可以运行的 pod 数量设置为一个固定值，而不需要考虑节点的属性。

```
kubeletConfig:
  maxPods: 250
```

5.4.1. 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：

- 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。如果要将相同的内容应用到所有池，则所有池仅需要一个 **KubeletConfig** CR。
- 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。
- 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**。

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：

```
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=large-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
```

这应该会显示 **status: "True"** 和 **type:Success**：

```
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
```

5.4.2. 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。

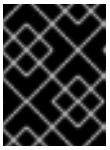
流程

1. 编辑 **worker** 机器配置池：

```
$ oc edit machineconfigpool worker
```

2. 将 **maxUnavailable** 设置为您需要的值：

```
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。

5.4.3. Control plane 节点大小

control plane 节点对资源的要求取决于集群中的节点数量。以下推荐的 control plane 节点大小是基于 control plane 密度测试的结果。control plane 测试会根据节点数在每个命名空间中在集群中创建以下对象：

- 12 个镜像流
- 3 个构建配置
- 6 个构建
- 1 个部署，带有 2 个 pod 副本，每个都挂载两个 secret
- 2 个部署，带有 1 个 pod 副本，挂载了两个 secret
- 3 个指向以前部署的服务
- 3 个指向之前部署的路由
- 10 个 secret，其中 2 个由以前的部署挂载
- 10 个配置映射，其中 2 个由以前的部署挂载

worker 节点数量	集群负载（命名空间）	CPU 内核	内存 (GB)
25	500	4	16
100	1000	8	32
250	4000	16	96

在具有三个 master 或 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、网络或底层基础架构出现意外问题，除了在关闭集群后重启集群以节约成本的情况下。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 master 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。

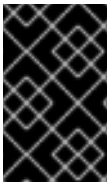
重要

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。

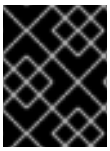
命名空间数量	处于空闲状态的 OLM 内存（GB）	安装了 5 个用户 operator 的 OLM 内存（GB）
500	0.823	1.7

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）
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
9000	8	17.7
10,000	9.9	21.6



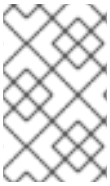
重要

如果使用安装程序置备的基础架构安装方法，则无法修改正在运行的 OpenShift Container Platform 4.8 集群中的 control plane 节点大小。反之，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 4.8 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。

5.4.4. 设置 CPU Manager

流程

1. 可选：标记节点：

```
# oc label node perf-node.example.com cpumanager=true
```

2. 编辑启用 CPU Manager 的节点的 **MachineConfigPool** 。在这个示例中，所有 worker 都启用了 CPU Manager：

```
# oc edit machineconfigpool worker
```

3. 为 worker 机器配置池添加标签：

```
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。
- **static**。此策略允许具有某些资源特征的 pod 获得提高 CPU 关联性和节点上专用的 pod。

2 可选。指定 CPU Manager 协调频率。默认值为 **5s**。

5. 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. 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. 检查 worker 是否有更新的 **kubelet.conf**：

```
# 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 2 当创建 **KubeletConfig** CR 时会定义这些设置。

8. 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:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
  nodeSelector:
    cpumanager: "true"
```

9. 创建 pod：

```
# oc create -f cpumanager-pod.yaml
```

10. 确定为您标记的节点调度了 pod：

```
# 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
```

11. 确认正确配置了 **cgroups**。获取 **pause** 进程的进程 ID（PID）：

```
# ├─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
```

服务质量（QoS）等级为 **Guaranteed** 的 pod 被放置到 **kubepods.slice** 中。其它 QoS 等级的 pod 会位于 **kubepods** 的子 **cgroups** 中：

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done
```

输出示例

```
cpuset.cpus 1
tasks 32706
```

12. 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
```

13. 确认系统中的另一个 pod（在这个示例中，QoS 等级为 **burstable** 的 pod）不能在为等级为**Guaranteed** 的 pod 分配的内核中运行：

```
-
```

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus

0
# 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
```

5.5. 巨页

了解并配置巨页。

5.5.1. 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。

5.5.2. 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。

5.5.3. 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。

5.5.3.1. 在引导时

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。

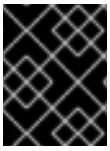
```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```

警告

目前，这个功能只在 Red Hat Enterprise Linux CoreOS（RHCOS）8.x worker 节点上被支持。在 Red Hat Enterprise Linux (RHEL) 7.x worker 节点上，目前不支持 TuneD **[bootloader]** 插件。

5.6. 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 reseting 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

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*。

5.6.1. 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。

5.6.2. 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。

5.6.3. 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。

5.7. 污点和容限

理解并使用污点和容限。

5.7.1. 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

```
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
....
```

Pod 规格中的容限示例

```
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 5.1. 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。

参数	描述	
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. 如果为 control plane 节点（也称为 master 节点）添加了一个 **NoSchedule** 污点，则节点必须具有 **node-role.kubernetes.io/master=:NoSchedule** 污点，该污点会被默认添加。
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
...
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**。

5.7.1.1. 了解如何使用容限秒数来延迟 pod 驱除

您可以通过在 **Pod** 规格或 **MachineSet** 对象中指定 **tolerationSeconds** 参数，指定 pod 在被驱除前可以保持与节点绑定的时长。如果将具有 **NoExecute effect** 的污点添加到节点，则容限污点（包含 **tolerationSeconds** 参数）的 pod，在此期限内 pod 不会被驱除。

输出示例

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

在这里，如果此 pod 正在运行但没有匹配的容限，pod 保持与节点绑定 3600 秒，然后被驱除。如果污点在这个时间之前移除，pod 就不会被驱除。

5.7.1.2. 了解如何使用多个污点

您可以在同一个节点中放入多个污点，并在同一 pod 中放入多个容限。OpenShift Container Platform 按照如下所述处理多个污点和容限：

1. 处理 pod 具有匹配容限的污点。
2. 其余的不匹配污点在 pod 上有指示的 effect：
   - 如果至少有一个不匹配污点具有 **NoSchedule** effect，则 OpenShift Container Platform 无法将 pod 调度到该节点上。
   - 如果没有不匹配污点具有 **NoSchedule** effect，但至少有一个不匹配污点具有 **PreferNoSchedule** effect，则 OpenShift Container Platform 尝试不将 pod 调度到该节点上。

- 如果至少有一个未匹配污点具有 **NoExecute** effect，OpenShift Container Platform 会将 pod 从该节点驱除（如果它已在该节点上运行），或者不将 pod 调度到该节点上（如果还没有在该节点上运行）。
  - 不容许污点的 Pod 会立即被驱除。
  - 如果 Pod 容许污点而没有在 **Pod** 规格中指定 **tolerationSeconds**，则会永久保持绑定。
  - 如果 Pod 容许污点，且指定了 **tolerationSeconds**，则会在指定的时间里保持绑定。

例如：

- 向节点添加以下污点：

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- pod 具有以下容限：

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
```

在本例中，pod 无法调度到节点上，因为没有与第三个污点匹配的容限。如果在添加污点时 pod 已在节点上运行，pod 会继续运行，因为第三个污点是三个污点中 pod 唯一不容许的污点。

5.7.1.3. 了解 pod 调度和节点状况（根据状况保留节点）

Taint Nodes By Condition （默认启用）可自动污点报告状况的节点，如内存压力和磁盘压力。如果某个节点报告一个状况，则添加一个污点，直到状况被清除为止。这些污点具有 **NoSchedule** effect；即，pod 无法调度到该节点上，除非 pod 有匹配的容限。

在调度 pod 前，调度程序会检查节点上是否有这些污点。如果污点存在，则将 pod 调度到另一个节点。由于调度程序检查的是污点而非实际的节点状况，因此您可以通过添加适当的 pod 容限，将调度程序配置为忽略其中一些节点状况。

为确保向后兼容，守护进程会自动将下列容限添加到所有守护进程中：

- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/unschedulable（1.10 或更高版本）
- node.kubernetes.io/network-unavailable（仅限主机网络）

您还可以在守护进程集中添加任意容限。

注意

control plane 还会在具有 QoS 类的 pod 中添加 **node.kubernetes.io/memory-pressure** 容限。这是因为 Kubernetes 在 **Guaranteed** 或 **Burstable** QoS 类中管理 pod。新的 **BestEffort** pod 不会调度到受影响的节点上。

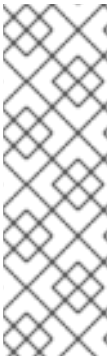
5.7.1.4. 了解根据状况驱除 pod（基于垃圾的驱除）

Taint-Based Evictions 功能默认是启用的，可以从遇到特定状况（如 **not-ready** 和 **unreachable**）的节点驱除 pod。当节点遇到其中一个状况时，OpenShift Container Platform 会自动给节点添加污点，并开始驱除 pod 以及将 pod 重新调度到其他节点。

Taint Based Evictions 具有 **NoExecute** 效果，不容许污点的 pod 都被立即驱除，容许污点的 pod 不会被驱除，除非 pod 使用 **tolerationSeconds** 参数。

tolerationSeconds 参数允许您指定 pod 保持与具有节点状况的节点绑定的时长。如果在 **tolerationSections** 到期后状况仍然存在，则污点会保持在节点上，并且具有匹配容限的 pod 将被驱除。如果状况在 **tolerationSeconds** 到期前清除，则不会删除具有匹配容限的 pod。

如果使用没有值的 **tolerationSeconds** 参数，则 pod 不会因为未就绪和不可访问的节点状况而被驱除。



注意

OpenShift Container Platform 会以限速方式驱除 pod，从而防止在主控机从节点分离等情形中发生大量 pod 驱除。

默认情况下，如果给定区域中的节点超过 55% 的节点不健康，节点生命周期控制器会将该区域的状态改为 **PartialDisruption**，并且 pod 驱除率会减少。对于此状态的小型集群（默认为 50 个节点或更少），这个区中的节点不会污点，驱除会被停止。

如需更多信息，请参阅 Kubernetes 文档中的 有关驱除率限制。

OpenShift Container Platform 会自动为 **node.kubernetes.io/not-ready** 和 **node.kubernetes.io/unreachable** 添加容限并设置 **tolerationSeconds=300**，除非 **Pod** 配置中指定了其中任一种容限。

```
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300 (1)
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```

(1) 这些容限确保了在默认情况下，pod 在检测到这些节点条件问题中的任何一个时，会保持绑定五分钟。

您可以根据需要配置这些容限。例如，如果您有一个具有许多本地状态的应用程序，您可能希望在发生网络分区时让 pod 与节点保持绑定更久一些，以等待分区恢复并避免 pod 驱除行为的发生。

由守护进程集生成的 pod 在创建时会带有以下污点的 **NoExecute** 容限，且没有 **tolerationSeconds**:

- **node.kubernetes.io/unreachable**
- **node.kubernetes.io/not-ready**

因此，守护进程集 pod 不会被驱除。

5.7.1.5. 容限所有污点

您可以通过添加 **operator: "Exists"** 容限而无需 **key** 和 **value** 参数，将节点配置为容许所有污点。具有此容限的 Pod 不会从具有污点的节点中删除。

用于容忍所有污点的Pod 规格

```
spec:
  tolerations:
  - operator: "Exists"
```

5.7.2. 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

   ```
   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

   ```
   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

如果为 control plane 节点（也称为 master 节点）添加了一个 **NoSchedule** 污点，则节点必须具有 **node-role.kubernetes.io/master=:NoSchedule** 污点，该污点会被默认添加。

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
...
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。

5.7.3. 使用机器集添加污点和容限

您可以使用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。容限对由机器集添加的污点的处理方式与直接添加到节点的污点的处理方式相同。

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

```
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

```
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：

   机器集规格中的污点示例

   ```
   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. 将机器缩减为 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. 根据需要扩展机器设置：

```
$ 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。

5.7.4. 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: <node_name>
  labels:
    ...
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。

5.7.5. 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：

```
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: <node_name>
  labels:
    ...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
```

5.7.6. 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：

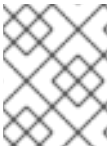
```
spec:
  tolerations:
  - key: "key2"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

5.8. 拓扑管理器

理解并使用拓扑管理器。

5.8.1. 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。



注意

要将 CPU 资源与 **Pod** 规格中的其他请求资源匹配，必须使用 **static** CPU Manager 策略启用 CPU Manager。

拓扑管理器支持四个分配策略，这些策略在 **cpumanager-enabled** 自定义资源（CR）中定义：

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。

5.8.2. 设置拓扑管理器

要使用拓扑管理器，您必须在 **cpumanager-enabled** 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**。请参考扩展和性能文档中的使用 CPU Manager 部分 。

流程

激活 Topolgy Manager:

1. 在 **cpumanager-enabled** 自定义资源（CR）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 此参数必须是 **static**。

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**。

5.8.3. 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 的静态策略，该策略可返回可用 CPU 的拓扑结构。拓扑管理器还参考设备管理器来发现可用设备的拓扑结构，如 example.com/device。

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。

5.9. 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%。

5.10. 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。

5.10.1. 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 Instance**。

   b. 在 **Create ClusterResourceOverride 页面**中，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/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1 引用 **ClusterResourceOverride** 准入Webhook。

5.10.2. 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: "4.8"
  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/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1 引用 **ClusterResourceOverride** 准入Webhook。

5.10.3. 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。

5.11. 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。

5.11.1. 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。

5.11.1.1. 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。

5.11.1.2. 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。

5.11.2. 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 5.2. 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。

5.11.2.1. 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。

5.11.3. 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。

5.11.4. 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 = 1
```

```
$ 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

5.11.5. 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

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)。

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: 3
      - "false"
```

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
```

5.11.6. 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”。

5.11.7. 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
```

5.12. 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。

5.12.1. 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. 编辑项目对象文件

2. 添加以下注解：

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

3. 创建项目对象：

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

5.13. 使用垃圾回收释放节点资源

理解并使用垃圾回收。

5.13.1. 了解如何通过垃圾回收移除已终止的容器

可使用驱除阈值来执行容器垃圾回收。

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 5.3. 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。

5.13.2. 了解如何通过垃圾回收移除镜像

镜像垃圾回收依靠节点上 cAdvisor 报告的磁盘用量来决定从节点中移除哪些镜像。

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 5.4. 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。

5.13.3. 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

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。

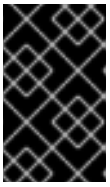
提示

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. 驱除类型： **evictionSoft** 或 **evictionHard**。
4. 基于特定驱除触发信号的驱除阈值。
5. 软驱除宽限期。此参数不适用于 **eviction-hard**。
6. 基于特定驱除触发信号的驱除阈值。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. 摆脱驱除压力状况前等待的时间。
8. 在通过垃圾收集移除镜像前，未用镜像的最小年龄。
9. 触发镜像垃圾回收的磁盘用量百分比（以整数表示）。
10. 镜像垃圾回收试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
```

5.14. 使用 NODE TUNING OPERATOR

理解并使用 Node Tuning Operator。

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。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。

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。

5.14.1. 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

1. 运行：

```
$ oc get Tuned/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 版本中可能会弃用 pod 标识功能。

5.14.2. 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)
```

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**。

<match> 是一个递归定义的可选数组，如下所示：

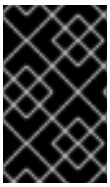
```
- label: <label_name> (1)
  value: <label_value> (2)
  type: <label_type> (3)
    <match> (4)
```

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。

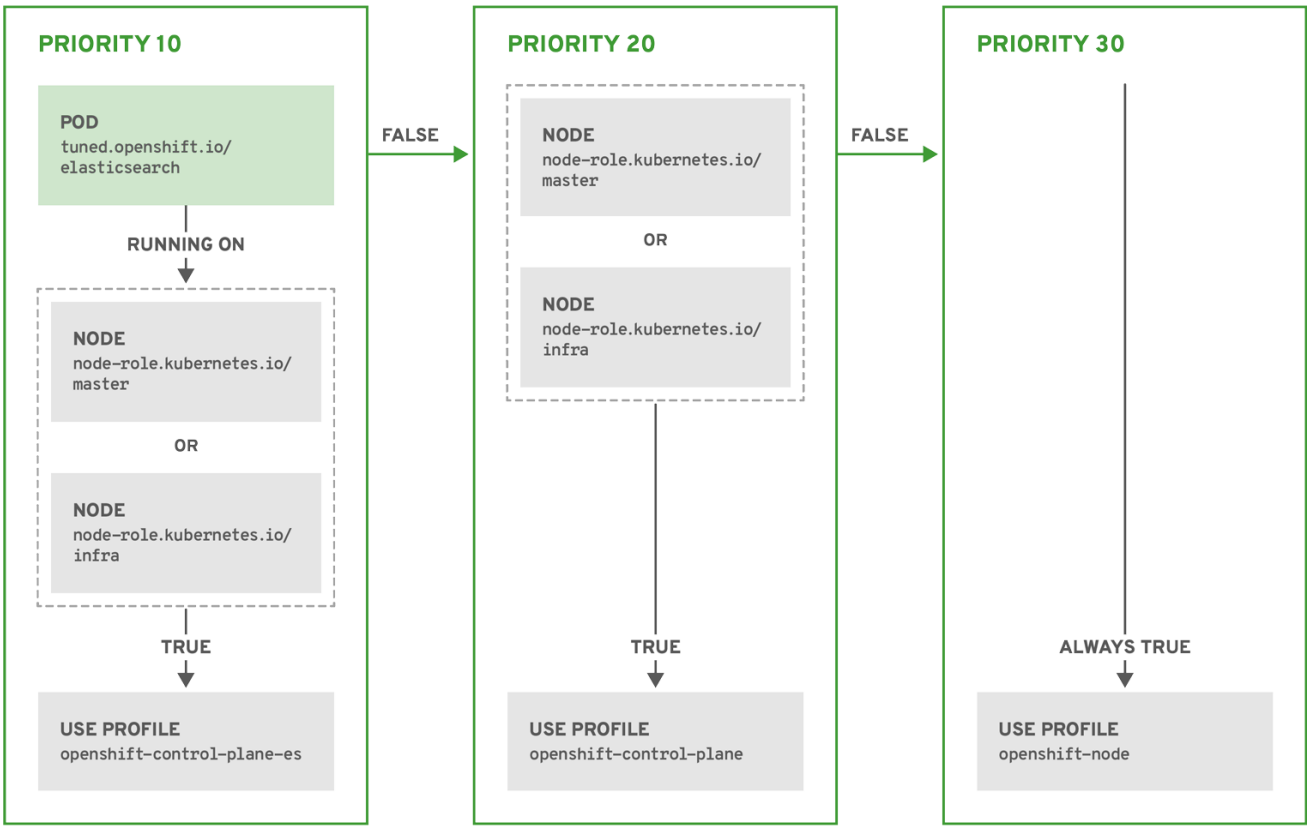
示例：基于节点或 pod 标签的匹配

```
- 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。

5.14.3. 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:
  - name: "openshift"
    data: |
      [main]
      summary=Optimize systems running OpenShift (parent profile)
      include=${f:virt_check:virtual-guest:throughput-performance}

      [selinux]
      avc_cache_threshold=8192

      [net]
      nf_conntrack_hashsize=131072

      [sysctl]
      net.ipv4.ip_forward=1
      kernel.pid_max=>4194304
      net.netfilter.nf_conntrack_max=1048576
      net.ipv4.conf.all.arp_announce=2
      net.ipv4.neigh.default.gc_thresh1=8192
      net.ipv4.neigh.default.gc_thresh2=32768
      net.ipv4.neigh.default.gc_thresh3=65536
      net.ipv6.neigh.default.gc_thresh1=8192
      net.ipv6.neigh.default.gc_thresh2=32768
      net.ipv6.neigh.default.gc_thresh3=65536
      vm.max_map_count=262144

      [sysfs]
      /sys/module/nvme_core/parameters/io_timeout=4294967295
      /sys/module/nvme_core/parameters/max_retries=10

  - name: "openshift-control-plane"
    data: |
      [main]
      summary=Optimize systems running OpenShift control plane
      include=openshift

      [sysctl]
      # ktune sysctl settings, maximizing i/o throughput
      #
      # Minimal preemption granularity for CPU-bound tasks:
      # (default: 1 msec#  (1 + ilog(ncpus)), units: nanoseconds)
      kernel.sched_min_granularity_ns=10000000
      # The total time the scheduler will consider a migrated process
      # "cache hot" and thus less likely to be re-migrated
      # (system default is 500000, i.e. 0.5 ms)
      kernel.sched_migration_cost_ns=5000000
      # SCHED_OTHER wake-up granularity.
      #
```

```
    # Preemption granularity when tasks wake up.  Lower the value to
    # improve wake-up latency and throughput for latency critical tasks.
    kernel.sched_wakeup_granularity_ns=4000000

- name: "openshift-node"
  data: |
    [main]
    summary=Optimize systems running OpenShift nodes
    include=openshift

    [sysctl]
    net.ipv4.tcp_fastopen=3
    fs.inotify.max_user_watches=65536
    fs.inotify.max_user_instances=8192

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
```

5.14.4. 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

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：

- bootloader
- script
- systemd

如需更多信息，请参阅 Available TuneD Plug-ins 和 Getting Started with TuneD 。

5.15. 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
```

第 6 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。

6.1. 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，且这些字段无法更改：

clusterNetwork

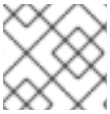
从中分配 Pod IP 地址的 IP 地址池。

serviceNetwork

服务的 IP 地址池.

defaultNetwork.type

集群网络供应商，如 OpenShift SDN 或 OVN-Kubernetes。



注意

在集群安装后，您无法修改上一节中列出的字段。

6.2. 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。

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。

6.3. 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: {}
```

6.4. 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）。

方法	用途
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。
配置一个 **NodePort**	在集群中的所有节点上公开某一服务。

6.5. 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。

6.5.1. 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。

6.5.1.1. 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

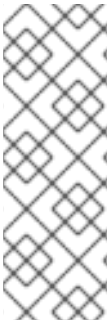
```
"service-node-port-range":["30000-33000"]
```

6.6. 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。

6.6.1. 关于网络策略

在使用支持 Kubernetes 网络策略的 Kubernetes Container Network Interface（CNI）插件的集群中，网络隔离完全由 **NetworkPolicy** 对象控制。在 OpenShift Container Platform 4.8 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。



注意

在使用 OpenShift SDN 集群网络供应商时，网络策略会有以下限制：

- 不支持由 **egress** 字段指定的出口网络策略。
- 网络策略支持 IPBlock，但不支持 **except**。如果创建的策略带有一个有 **except** 的 IPBlock 项，SDN pod 的日志中会出现警告，策略中的整个 IPBlock 项都会被忽略。

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。

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。

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。

6.6.2. 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。

6.6.3. 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** 对象的集群网络供应商，如 OVN-Kubernetes 网络供应商或设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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

   ```
   kind: NetworkPolicy
   ```

```
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  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: {}
```

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/default-deny created
```

6.6.4. 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。

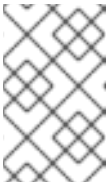
先决条件

- 集群使用支持 **NetworkPolicy** 对象的集群网络供应商，如 OVN-Kubernetes 网络供应商或设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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
```

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
```

6.6.5. 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。

6.6.6. 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：

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. 点击 **Global Configuration**，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。

6.6.6.1. 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 网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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
...
```

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
   ```

6.7. 支持的配置

Red Hat OpenShift Service Mesh 当前发行版本支持以下配置。

6.7.1. 支持的平台

Red Hat OpenShift Service Mesh Operator 支持 **ServiceMeshControlPlane** 资源的多个版本。在以下平台版本中支持以下 2.2 Service Mesh control plane：

- Red Hat OpenShift Container Platform 版本 4.9 或更高版本。
- Red Hat OpenShift Dedicated 版本 4。
- Azure Red Hat OpenShift (ARO) 版本 4。
- Red Hat OpenShift Service on AWS (ROSA)。

6.7.2. 不支持的配置

明确不支持的情形包括：

- OpenShift Online 不支持 Red Hat OpenShift Service Mesh。
- Red Hat OpenShift Service Mesh 不支持在 Service Mesh 集群外部管理微服务。

6.7.3. 支持的网络配置

Red Hat OpenShift Service Mesh 支持以下网络配置。

- OpenShift-SDN
- OpenShift Container Platform 4.7.32+、OpenShift Container Platform 4.8.12+ 和 OpenShift Container Platform 4.9+ 支持 OVN-Kubernetes。
- 在 OpenShift Container Platform 上认证并通过 Service Mesh 一致性测试的第三方 Container Network Interface(CNI)插件。如需更多信息，请参阅认证的 OpenShift CNI 插件。

6.7.4. Service Mesh 支持的配置

- 此 Red Hat OpenShift Service Mesh 发行版本仅适用于 OpenShift Container Platform x86_64、IBM Z 和 IBM Power Systems。
  - IBM Z 只在 OpenShift Container Platform 4.6 及更新的版本上被支持。
  - IBM Power Systems 只在 OpenShift Container Platform 4.6 及更新的版本上被支持。
- 将所有 Service Mesh 组件包含在单个 OpenShift Container Platform 集群中的配置。
- 不集成外部服务的配置，如虚拟机。
- Red Hat OpenShift Service Mesh 不支持 **EnvoyFilter** 配置，除非明确记录。

6.7.5. Kiali 支持的配置

- Kiali 控制台只支持 Chrome 、Edge 、Firefox 或 SDomain 浏览器的最新的两个版本。

6.7.6. 分布式追踪支持的配置

- Jaeger 代理是 Jaeger 唯一支持的配置。多租户安装或 OpenShift Dedicated 不支持 Jaeger 作为 daemonset。

6.7.7. 支持的 WebAssembly 模块

- 3scale WebAssembly 是唯一提供 WebAssembly 模块。您可以创建自定义 WebAssembly 模块。

6.7.8. Operator 概述

Red Hat OpenShift Service Mesh 需要以下四个 Operator：

- **OpenShift Elasticsearch** -（可选）为使用分布式追踪平台进行追踪和日志记录提供数据库存储。它基于开源 Elasticsearch 项目。
- **Red Hat OpenShift distributed tracing 平台** - 提供分布式追踪以监控复杂分布式系统中的事务并进行故障排除。它基于开源 Jaeger 项目。
- **Kiali** - 为您的服务网格提供可观察性。允许您在单个控制台中查看配置、监控流量和分析追踪。它基于开源 Kiali 项目。
- **Red Hat OpenShift Service Mesh** - 允许您连接、保护、控制和观察组成应用程序的微服务。Service Mesh Operator 定义并监控管理 **ServiceMeshControlPlane** 资源，这个资源用来管理 Service Mesh 组件的部署、更新和删除操作。它基于开源 Istio 项目。

后续步骤

- 在OpenShift Container Platform 环境中安装 Red Hat OpenShift Service Mesh 。

6.8. 优化路由

OpenShift Container Platform HAProxy 路由器扩展以优化性能。

6.8.1. Ingress Controller（router）性能的基线

OpenShift Container Platform Ingress Controller，或称为路由器，是所有用于 OpenShift Container Platform 服务的外部流量的入站点。

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 的公共云实例中进行测试。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 配置使用 **ROUTER_THREADS=4**，并测试了两个不同的端点发布策略 (LoadBalancerService/hostnetwork) 。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共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 5 到 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。

6.8.2. Ingress Controller（路由器）性能优化

OpenShift Container Platform 不再支持通过设置以下环境变量来修改 Ingress Controller 的部署：**ROUTER_THREADS**、**ROUTER_DEFAULT_TUNNEL_TIMEOUT**、**ROUTER_DEFAULT_CLIENT_TIMEOUT**、**ROUTER_DEFAULT_SERVER_TIMEOUT** 和 **RELOAD_INTERVAL**。

您可以修改 Ingress Controller 的部署，但当 Ingress Operator 被启用时，其配置会被覆盖。

6.9. 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。

6.9.1. 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> grafana-openshift-monitoring.apps.<cluster name>.<base domain>
<apps_FIP> <app name>.apps.<cluster name>.<base domain>
```

6.9.2. Kuryr 端口池

Kuryr 端口池在待机时维护多个端口，用于创建 pod。

将端口保留在待机上可最大程度缩短 pod 创建时间。如果没有端口池，Kuryr 必须明确请求在创建或删除 pod 时创建或删除端口。

Kuryr 使用的 Neutron 端口是在绑定到命名空间的子网中创建的。这些 pod 端口也作为子端口添加到 OpenShift Container Platform 集群节点的主端口。

因为 Kuryr 将每个命名空间保留在单独的子网中，所以为每个命名空间 worker 对维护一个单独的端口池。

在安装集群前，您可以在 **cluster-network-03-config.yml** 清单文件中设置以下参数来配置端口池行为：

- **enablePortPoolsPrepopulation 参数** 控制池预填充，它会强制 Kuryr 在创建时（如添加新主机或创建新命名空间时）将端口添加到池中。默认值为 **false**。
- **poolMinPorts** 参数是池中保留的最少可用端口的数量。默认值为 **1**。
- **poolMaxPorts** 参数是池中保留的最大可用端口数。值 **0 可** 禁用此上限。这是默认设置。
  如果您的 OpenStack 端口配额较低，或者 pod 网络上的 IP 地址有限，请考虑设置此选项以确保删除不需要的端口。
- **poolBatchPorts** 参数定义一次可以创建的 Neutron 端口的最大数量。默认值为 **3**。

6.9.3. 在 RHOSP 上的活跃部署中调整 Kuryr 端口池设置

您可以使用自定义资源 (CR) 配置 Kuryr 如何管理 Red Hat OpenStack Platform (RHOSP) Neutron 端口，以控制在部署的集群上创建 pod 的速度和效率。

流程

1. 在命令行中，打开 Cluster Network Operator (CNO) CR 进行编辑：

```
$ oc edit networks.operator.openshift.io cluster
```

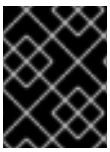
2. 编辑设置以满足您的要求。以下示例提供了以下文件：

```
apiVersion: operator.openshift.io/v1
kind: Network
```

```
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  defaultNetwork:
    type: Kuryr
    kuryrConfig:
      enablePortPoolsPrepopulation: false (1)
      poolMinPorts: 1 (2)
      poolBatchPorts: 3 (3)
      poolMaxPorts: 5 (4)
```

1. 将 **enablePortPoolsPrepopulation** 设置为 **true** 以使 Kuryr 在创建命名空间或在集群中添加新节点后创建新 Neutron 端口。此设置引发 Neutron 端口配额，但可以缩短生成容器集所需的时间。默认值为 **false**。
2. 如果池中的可用端口数量低于 **poolMinPorts** 的值，Kuryr 会为池创建新端口。默认值为 **1**。
3. **poolBatchPorts** 控制在可用端口数量低于 **poolMinPorts** 值时创建的新端口数量。默认值为 **3**。
4. 如果池中的可用端口数量大于 **poolMaxPorts** 的值，Kuryr 会删除它们，直到数量与这个值匹配为止。将值设为 **0** 可禁用此上限，防止池被缩小。默认值为 **0**。

3. 保存您的更改，再退出文本编辑器以提交更改。



重要

在正在运行的集群中修改这些选项会强制 kuryr-controller 和 kuryr-cni pod 重启。因此，创建新 pod 和服务会延迟。

6.9.4. 为负载均衡器服务启用 RHOSP Octavia

您可以通过在 Red Hat OpenStack Platform (RHOSP)上使用 Octavia 创建负载均衡器服务类型和入口控制器。

注意

依赖 Octavia 的服务和控制器有以下限制：

- 仅支持 TCP 流量。
- 在集群删除操作过程中不会删除活跃的 Octavia 负载均衡器以及附加到它们的浮动 IP 地址。您必须先删除这些项目，然后才能执行操作。
- 云供应商配置中的 **manage-security-groups** 属性只适用于具有管理特权的 RHOSP 租户。
- 不支持负载均衡器服务的 **loadBalancerSourceRanges** 属性。
- 不支持负载均衡器服务的 **loadBalancerIP** 属性。

先决条件

- 您有一个活跃的集群。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 在命令行中，打开云供应商配置进行编辑：

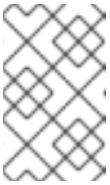
```
$ oc edit configmap -n openshift-config cloud-provider-config
```

2. 编辑驱动程序类型的配置：
   - 如果您使用 Amphora 驱动程序，请在云供应商配置中添加以下部分：

```
[LoadBalancer]
use-octavia = true
lb-provider = amphora
```

   - 如果使用 OVN 驱动程序，请在云供应商配置中添加以下部分：

```
[LoadBalancer]
use-octavia = true
lb-provider = ovn
lb-method = SOURCE_IP_PORT
```



注意

如果您对 Octavia 使用 OVN 驱动程序，还必须修改主和 worker 安全组的 TCP ingress 安全组规则，以允许来自 0.0.0.0/0 的 IPv4 流量发送到端口 30000 到 32767。

3. 如果您有多个外部网络，请将云供应商配置中的 **floating-network-id** 参数的值设置为在其中创建浮动 IP 地址的外部网络 UUID。例如：

```
[LoadBalancer]
use-octavia = true
```

```
lb-provider = amphora
floating-network-id = <network_UUID>
```

4. 保存对配置的更改。

第 7 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。

7.1. 动态置备

7.1.1. 关于动态置备

StorageClass 资源对象描述并分类了可请求的存储，并提供了根据需要为动态置备存储传递参数的方法。**StorageClass** 也可以作为控制不同级别的存储和访问存储的管理机制。集群管理员(**cluster-admin**)或存储管理员(**storage-admin**)可以在无需了解底层存储卷资源的情况下，定义并创建用户可以请求的 **StorageClass** 对象。

OpenShift Container Platform 的持久性卷框架启用了这个功能，并允许管理员为集群提供持久性存储。该框架还可让用户在不了解底层存储架构的情况下请求这些资源。

很多存储类型都可用于 OpenShift Container Platform 中的持久性卷。虽然它们都可以由管理员静态置备，但有些类型的存储是使用内置供应商和插件 API 动态创建的。

7.1.2. 可用的动态置备插件

OpenShift Container Platform 提供了以下置备程序插件，用于使用集群配置的供应商 API 创建新存储资源的动态部署：

存储类型	provisioner 插件名称	备注
Red Hat OpenStack Platform（RHOSP）Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface（CSI）	**manila.csi.openstack.org**	安装后, OpenStack Manila CSI Driver Operator 和 ManilaDriver 会自动为所有可用 Manila 共享类型创建动态置备所需的存储类。
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	当在不同的区中使用多个集群进行动态置备时，使用 **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>**（每个集群的**<cluster_name>** 和 **<cluster_id>** 是唯一的）来标记（tag）每个节点。
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** 服务帐户需要相应的权限，以创建并获取 Secret 来存储 Azure 存储帐户和密钥。

存储类型	provisioner 插件名称	备注
GCE 持久性磁盘 (gcePD)	**kubernetes.io/gce-pd**	在多区（multi-zone）配置中，建议在每个 GCE 项目中运行一个 OpenShift Container Platform 集群，以避免在当前集群没有节点的区域中创建 PV。
VMware vSphere	**kubernetes.io/vsphere-volume**	

重要

任何选择的置备程序插件还需要根据相关文档为相关的云、主机或者第三方供应商配置。

7.2. 定义存储类

StorageClass 对象目前是一个全局范围的对象，必须由 **cluster-admin** 或 **storage-admin** 用户创建。

重要

根据使用的平台，Cluster Storage Operator 可能会安装一个默认的存储类。这个存储类由 Operator 拥有和控制。不能在定义注解和标签之外将其删除或修改。如果需要实现不同的行为，则必须定义自定义存储类。

以下小节描述了 **StorageClass** 对象的基本定义，以及每个支持的插件类型的具体示例。

7.2.1. 基本 StorageClass 对象定义

以下资源显示了用来配置存储类的参数和默认值。这个示例使用 AWS ElasticBlockStore (EBS) 对象定义。

StorageClass 定义示例

```
kind: StorageClass (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: gp2 (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs (5)
parameters: (6)
  type: gp2
...
```

(1) （必需）API 对象类型。

2 （必需）当前的 apiVersion。

3 （必需）存储类的名称。

4 （可选）存储类的注解。

5 （必需）与这个存储类关联的置备程序类型。

6 （可选）特定置备程序所需的参数，这将根据插件的不同而有所不同。

7.2.2. 存储类注解

要将存储类设置为集群范围的默认值，请在存储类元数据中添加以下注解：

```
storageclass.kubernetes.io/is-default-class: "true"
```

例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

这允许任何没有指定特定存储类的持久性卷声明（PVC）通过默认存储类自动置备。但是，您的集群可以有多个存储类，但只有其中一个是默认存储类。

注意

beta 注解 **storageclass.beta.kubernetes.io/is-default-class** 当前仍然可用，但将在以后的版本中被删除。

要设置存储类描述，请在存储类元数据中添加以下注解：

```
kubernetes.io/description: My Storage Class Description
```

例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

7.2.3. RHOSP Cinder 对象定义

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast (1)
  availability: nova (2)
  fsType: ext4 (3)
```

1. 在 Cinder 中创建的卷类型。默认为空。
2. 可用区。如果没有指定可用区，则通常会在所有 OpenShift Container Platform 集群有节点的所有活跃区域间轮换选择。
3. 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

7.2.4. AWS Elastic Block Store (EBS) 对象定义

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 (1)
  iopsPerGB: "10" (2)
  encrypted: "true" (3)
  kmsKeyId: keyvalue (4)
  fsType: ext4 (5)
```

1. （必需）选择 **io1**、 **gp2**、 **sc1**、 **st1**。默认为 **gp2**。可用的 Amazon 资源名 (ARN) 值请查看 AWS 文档 。
2. （可选）只适用于 **io1** 卷。每个 GiB 每秒一次 I/O 操作。AWS 卷插件乘以这个值，再乘以请求卷的大小以计算卷的 IOPS。数值上限为 20,000 IOPS，这是 AWS 支持的最大值。详情请查看 AWS 文档 。
3. （可选）是否加密 EBS 卷。有效值为 **true** 或者 **false**。
4. （可选）加密卷时使用的密钥的完整 ARN。如果没有提供任何信息，但 **encypted** 被设置为 **true**，则 AWS 会生成一个密钥。有效 ARN 值请查看 AWS 文档 。
5. （可选）在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

7.2.5. Azure Disk 对象定义

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer (1)
allowVolumeExpansion: true
parameters:
  kind: Managed (2)
  storageaccounttype: Premium_LRS (3)
reclaimPolicy: Delete
```

1. 强烈建议使用 **WaitForFirstConsumer**。这会置备卷，同时允许有足够的存储空间从可用区将 pod 调度到空闲 worker 节点上。

2. 可能的值有 **Shared** （默认）、**Managed** 和 **Dedicated**。

重要

红帽仅在存储类中支持使用 **kind: Managed**。

使用 **Shared** 和 **Dedicated** 时，Azure 会创建非受管磁盘，而 OpenShift Container Platform 为机器 OS（root）磁盘创建一个受管磁盘。但是，因为 Azure Disk 不允许在节点上同时使用受管和非受管磁盘，所以使用 **Shared** 或 **Dedicated** 创建的非受管磁盘无法附加到 OpenShift Container Platform 节点。

3. Azure 存储帐户 SKU 层。默认为空。请注意，高级虚拟机可以同时附加 **Standard_LRS** 和 **Premium_LRS** 磁盘，标准虚拟机只能附加 **Standard_LRS** 磁盘，受管虚拟机只能附加受管磁盘，非受管虚拟机则只能附加非受管磁盘。

    a. 如果 **kind** 设为 **Shared**，Azure 会在与集群相同的资源组中的几个共享存储帐户下创建所有未受管磁盘。

    b. 如果 **kind** 设为 **Managed**，Azure 会创建新的受管磁盘。

    c. 如果 **kind** 设为 **Dedicated**，并且指定了 **StorageAccount**，Azure 会将指定的存储帐户用于与集群相同的资源组中新的非受管磁盘。为此，请确保：

    - 指定的存储帐户必须位于同一区域。
    - Azure Cloud Provider 必须具有存储帐户的写入权限。

    d. 如果 **kind** 设为 **Dedicated**，并且未指定 **StorageAccount**，Azure 会在与集群相同的资源组中为新的非受管磁盘创建一个新的专用存储帐户。

7.2.6. Azure File 对象定义

Azure File 存储类使用 secret 来存储创建 Azure File 共享所需的 Azure 存储帐户名称和存储帐户密钥。这些权限是在以下流程中创建的。

流程

1. 定义允许创建和查看 secret 的 **ClusterRole** 对象：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
#  name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: ['']
  resources: ['secrets']
  verbs:     ['get','create']
```

1 要查看并创建 secret 的集群角色名称。

2. 将集群角色添加到服务帐户：

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
```

输出示例

```
 system:serviceaccount:kube-system:persistent-volume-binder
```

3. 创建 Azure File **StorageClass** 对象：

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> 1
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus 2
  skuName: Standard_LRS 3
  storageAccount: <storage-account> 4
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。

2 Azure 存储帐户的位置，如 **eastus**。默认为空，表示将在 OpenShift Container Platform 集群的位置创建新的 Azure 存储帐户。

3 Azure 存储帐户的 SKU 层，如 **Standard_LRS**。默认为空，表示将使用 **Standard_LRS** SKU 创建新的 Azure 存储帐户。

4 Azure 存储帐户的名称。如果提供了存储帐户，则忽略 **skuName** 和 **location**。如果没有提供存储帐户，则存储类会为任何与定义的 **skuName** 和 **location** 匹配的帐户搜索与资源组关联的存储帐户。

7.2.6.1. 使用 Azure File 时的注意事项

默认 Azure File 存储类不支持以下文件系统功能：

- 符号链接

- 硬链接
- 扩展属性
- 稀疏文件
- 命名管道

另外，Azure File 挂载目录的所有者用户标识符 (UID) 与容器的进程 UID 不同。可在 **StorageClass** 对象中指定 **uid** 挂载选项来定义用于挂载的目录的特定用户标识符。

以下 **StorageClass** 对象演示了修改用户和组标识符，以及为挂载的目录启用符号链接。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1. 指定用于挂载的目录的用户标识符。
2. 指定用于挂载的目录的组标识符。
3. 启用符号链接。

7.2.7. GCE PersistentDisk (gcePD) 对象定义

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 1
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1. 选择 **pd-standard** 或 **pd-ssd**。默认为 **pd-standard**。

7.2.8. VMware vSphere 对象定义

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume (1)
parameters:
  diskformat: thin (2)
```

(1) 有关在 OpenShift Container Platform 中使用 VMware vSphere 的详情，请参阅 VMware vSphere 文档。

(2) **diskformat**：**thin**、**zeroedthick** 和 **eagerzeroedthick** 都是有效的磁盘格式。如需有关磁盘格式类型的更多详情，请参阅 vSphere 文档。默认值为 **thin**。

7.3. 更改默认存储类

使用以下流程更改默认存储类。例如，您有两个定义的存储类 **gp2** 和 **standard**，您想要将默认存储类从 **gp2** 改为 **standard**。

1. 列出存储类：

```
$ oc get storageclass
```

输出示例

```
NAME                TYPE
gp2 (default)       kubernetes.io/aws-ebs (1)
standard            kubernetes.io/aws-ebs
```

(1) **（默认）** 表示默认存储类。

2. 将默认存储类的 **storageclass.kubernetes.io/is-default-class** 注解的值改为 **false** ：

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 通过将 **storageclass.kubernetes.io/is-default-class** 注解设置为 **true** 来使另一个存储类成为默认值：

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 确认更改：

```
$ oc get storageclass
```

输出示例

```
NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```

7.4. 优化存储

优化存储有助于最小化所有资源中的存储使用。通过优化存储，管理员可帮助确保现有存储资源以高效的方式工作。

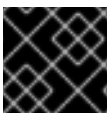
7.5. 可用的持久性存储选项

了解持久性存储选项，以便可以优化 OpenShift Container Platform 环境。

表 7.1. 可用存储选项

存储类型	描述	例子
Block	• 在操作系统 (OS) 中作为块设备 • 适用于需要完全控制存储，并绕过文件系统在低层直接操作文件的应用程序 • 也称为存储区域网络 (SAN) • 不可共享，这意味着，每次只有一个客户端可以挂载这种类型的端点	AWS EBS 和 VMware vSphere 支持在 OpenShift Container Platform 中的原生动态持久性卷 (PV)置备 。
File	• 在 OS 中作为要挂载的文件系统导出 • 也称为网络附加存储（Network Attached Storage，NAS） • 取决于不同的协议、实现、厂商及范围，其并行性、延迟、文件锁定机制和其它功能可能会有很大不同。	RHEL NFS、NetApp NFS [1] 和供应商 NFS
对象	• 通过 REST API 端点访问 • 可配置用于 OpenShift Container Platform Registry • 应用程序必须在应用程序和（/或）容器中构建其驱动程序。	AWS S3

1. NetApp NFS 在使用 Trident 插件时支持动态 PV 置备。



重要

目前，OpenShift Container Platform 4.8 不支持 CNS。

7.6. 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 7.2. 推荐的、可配置的存储技术

存储类型	ROX[1]	RWX[2]	Registry	扩展的 registry	Metrics[3]	日志记录	Apps
Block	Yes[4]	否	可配置	无法配置	推荐的	推荐的	推荐的
File	Yes[4]	是	可配置	可配置	Configurable[5]	Configurable[6]	推荐的
对象	是	是	推荐的	推荐的	无法配置	无法配置	Not configurable[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

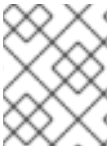
[3] **Prometheus 是用于指标数据的底层技术。**

[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，使用任何共享存储都将是一个反模式。每个 elasticsearch 都需要一个卷。**

[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 Container Platform 容器镜像仓库，它有两个或更多个 pod 运行副本。

7.6.1. 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。

市场上的其他 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。

7.6.1.1. Registry

在一个非扩展的/高可用性 (HA) OpenShift Container Platform 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 Container Platform Registry 集群部署，我们不推荐使用文件存储。

7.6.1.2. 扩展的 registry

在扩展的/HA OpenShift Container Platform 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 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。

7.6.1.3. 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。

7.6.1.4. 日志记录

在 OpenShift Container Platform 托管的日志集群部署中：

- 首选存储技术是块存储。
- 对象存储是不可配置的。

7.6.1.5. 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。

7.6.2. 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*。

其他资源

- 推荐的 etcd 实践

7.7. 部署 OPENSHIFT CONTAINER STORAGE

Red Hat OpenShift Container Storage 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 Container Storage 与 OpenShift Container Platform 完全集成，用于部署、管理和监控。

如果您要寻找 Red Hat OpenShift Container Storage 的相关信息	请参阅以下 Red Hat OpenShift Container Storage 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 Container Storage 4.7 发行注记
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 Container Storage 4.5 部署
有关在您的环境没有直接连接到互联网时准备部署的说明	准备在断开连接的环境中部署 OpenShift Container Storage 4.5
部署 OpenShift Container Storage 以使用外部 Red Hat Ceph Storage 集群的说明	以外部模式部署 OpenShift Container Storage 4.5
有关将 OpenShift Container Storage 部署到裸机基础架构上的本地存储的说明	使用裸机基础架构部署 OpenShift Container Storage 4.5
在 Red Hat OpenShift Container Platform VMware vSphere 集群上部署 OpenShift Container Storage 的说明	在 VMware vSphere 上部署 OpenShift Container Storage 4.5

如果您要寻找 Red Hat OpenShift Container Storage 的相关信息	请参阅以下 Red Hat OpenShift Container Storage 文档：
有关使用 Amazon Web Services 进行本地或云存储的部署 OpenShift Container Storage 的说明	使用 Amazon Web Services 部署 OpenShift Container Storage 4.5
在现有 Red Hat OpenShift Container Platform Google Cloud 集群上部署和管理 OpenShift Container Storage 的说明	使用 Google Cloud 部署和管理 OpenShift Container Storage 4.5
在现有 Red Hat OpenShift Container Platform Azure 集群上部署和管理 OpenShift Container Storage 的说明	使用 Microsoft Azure 部署和管理 OpenShift Container Storage 4.5
管理 Red Hat OpenShift Container Storage 4.5 集群	管理 OpenShift Container Storage 4.5
监控 Red Hat OpenShift Container Storage 4.5 集群	监控 Red Hat OpenShift Container Storage 4.5
解决操作过程中遇到的问题	OpenShift Container Storage 4.5 故障排除
将 OpenShift Container Platform 集群从版本 3 迁移到版本 4	迁移

第 8 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。

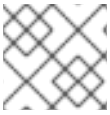
8.1. 了解身份提供程序配置

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。

8.1.1. 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 %。

8.1.2. 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。

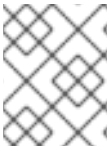
用户身份提供程序	描述
Google	配置 **google** 身份提供程序，使用 Google 的 OpenID Connect 集成。
OpenID Connect	配置 **oidc** 身份提供程序，使用授权代码流与 OpenID Connect 身份提供程序集成。

定义了身份提供程序后，您可以使用 RBAC 定义并应用权限。

8.1.3. 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。 **generate** 使用身份的首选用户名置备用户。如果拥有首选用户名的用户已映射到现有的身份，则生成一个唯一用户名。例如：**myuser2**。此方法不应与需要在 OpenShift Container Platform 用户名和身份提供程序用户名（如 LDAP 组同步）之间完全匹配的外部流程一同使用。 **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。

8.1.4. 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。

8.2. 使用 RBAC 定义和应用权限

理解并应用基于角色的访问控制。

8.2.1. 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。

8.2.1.1. 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。

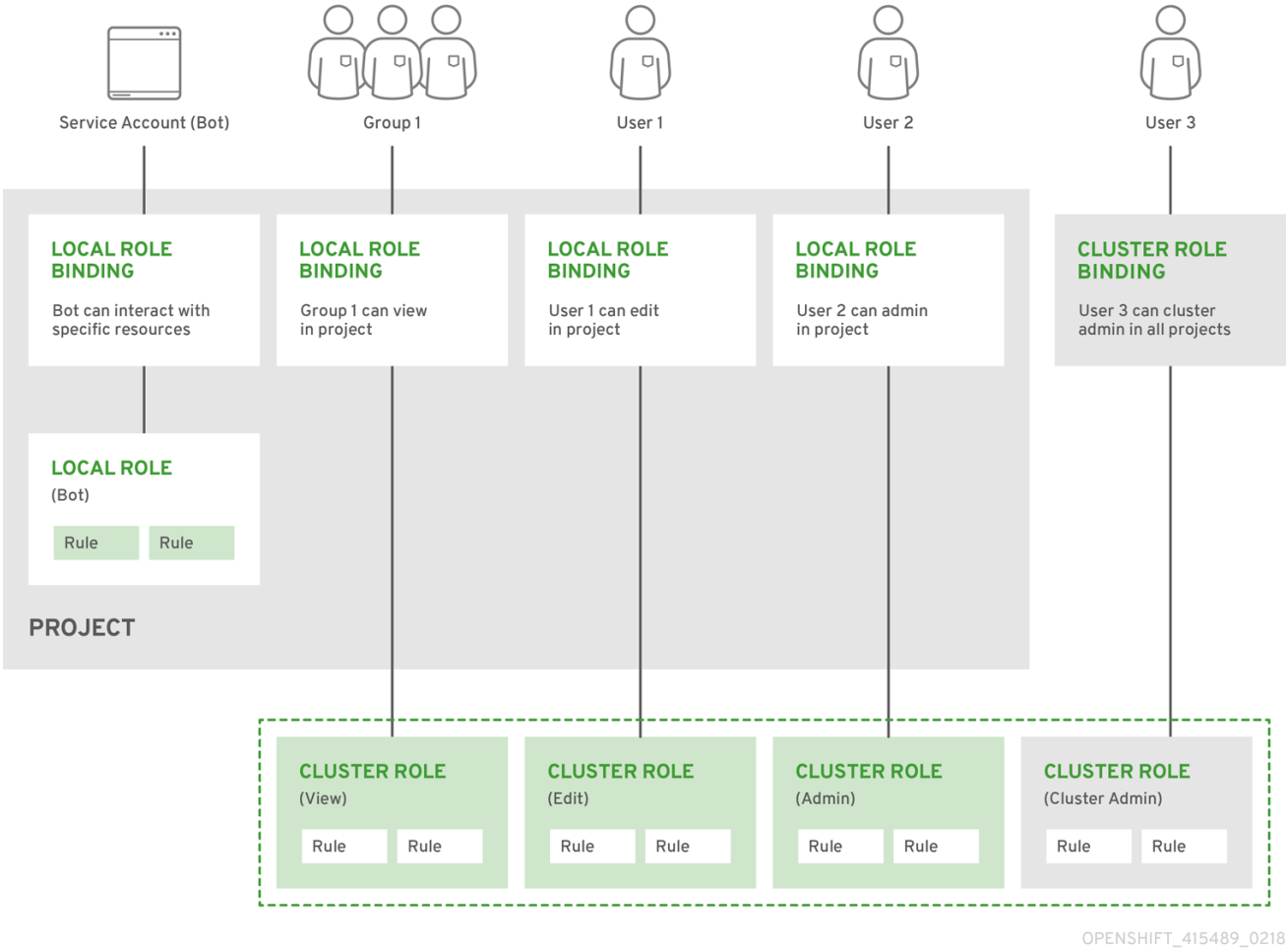
重要

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。



8.2.1.2. 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：

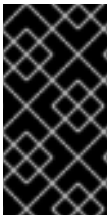
1. 使用身份和项目范围操作来查找应用到用户或所属组的所有绑定。
2. 使用绑定来查找应用的所有角色。

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。

8.2.1.2.1. 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。

8.2.2. 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。

8.2.3. 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”。

注意

您无法将 SCC 分配给在以下某一默认命名空间中创建的 Pod: **default**、**kube-system**、**kube-public**、**openshift-node**、**openshift-infra**、**openshift**。您不能使用这些命名空间用来运行 pod 或服务。

8.2.4. 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

...
```

8.2.5. 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
```

8.2.6. 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**。

8.2.7. 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
```

8.2.8. 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
```

8.2.9. 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 8.1. 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。

8.2.10. 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 8.2. 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。
$ 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。

8.2.11. 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>
```

8.3. 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>
```

8.3.1. 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
```

8.4. 镜像配置

了解并配置镜像 registry 设置。

8.4.1. 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，则端口还需要包含在域名中。 **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。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。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。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**）以及默认的内部镜像 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。它为默认内部镜像 registry 设置主机名。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。

8.4.2. 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。Machine Config Operator（MCO）会监控 **image.config.openshift.io/cluster** CR 是否有对 registry 的更改，并在检测到更改时重启节点。

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：

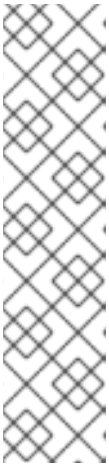
```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
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。

(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



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 和默认的内部镜像 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
ci-ln-j5cd0qt-f76d1-vfj5x-master-0       Ready                         master   98m
v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-master-1       Ready,SchedulingDisabled      master   99m
v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-master-2       Ready                         master   98m
v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-b-nsnd4   Ready                       worker   90m
v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-c-5z2gz   NotReady,SchedulingDisabled   worker   90m
v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-d-stsjv   Ready                       worker   90m
v1.19.0+7070803
```

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。

8.4.2.1. 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，配置映射键是带有信任此 CA 的端口的 registry 的主机名，而使用 base64 编码的证书是它的值。

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。

1. 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
```

8.4.2.2. 配置镜像 registry 存储库镜像

通过设置容器 registry 存储库镜像，您可以进行以下操作：

- 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 中存储库镜像的属性包括：

- 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：

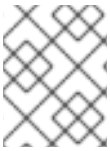
- 在 OpenShift Container Platform 安装中：
  通过拉取（pull） OpenShift Container Platform 所需的容器镜像，然后将这些镜像放至公司防火墙后，即可将 OpenShift Container Platform 安装到受限网络中的数据中心。

- 安装 OpenShift Container Platform 后：
  即使没有在 OpenShift Container Platform 安装期间配置镜像 (mirror)，之后您仍可使用 **ImageContentSourcePolicy** 对象进行配置。

以下流程提供安装后镜像配置，您可在此处创建 **ImageContentSourcePolicy** 对象来识别：

- 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。



注意

您只能为具有 **ImageContentSourcePolicy** 对象的集群配置全局 pull secret。您不能在项目中添加 pull secret。

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目录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/ubi8/ubi-minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455ebaa6 \
     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，您希望将 **ubi8/ubi-minimal** 镜像从 **registry.access.redhat.com** 复制到此镜像存储库。创建该 registry 后，您可将 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. 创建 **ImageContentSourcePolicy** 文件（如：**registryrepomirror.yaml**），将源和镜像 (mirror) 替换为您自己的 registry、存储库对和镜像中的源和镜像：

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal (1)
    source: registry.access.redhat.com/ubi8/ubi-minimal (2)
  - mirrors:
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 (3)
```

(1) 指明镜像 registry 和存储库的名称。

(2) 指明包含所镜像内容的 registry 和存储库。

(3) 您可以在 registry 中配置命名空间以使用该命名空间中的任何镜像。如果您使用 registry 域作为源，**ImageContentSourcePolicy** 资源将应用到 registry 中的所有存储库。

4. 创建新的 **ImageContentSourcePolicy** 对象：

```
$ oc create -f registryrepomirror.yaml
```

创建 **ImageContentSourcePolicy** 对象后，新的设置将部署到每个节点，集群开始使用已镜像的存储库来响应源存储库请求。

5. 要检查是否应用了已镜像的配置设置，在其中一个节点上执行以下内容。

   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   v1.21.0
ip-10-0-138-148.ec2.internal  Ready                      master   11m  v1.21.0
ip-10-0-139-122.ec2.internal  Ready                      master   11m  v1.21.0
ip-10-0-147-35.ec2.internal  Ready,SchedulingDisabled   worker   7m   v1.21.0
ip-10-0-153-12.ec2.internal  Ready                      worker   7m   v1.21.0
ip-10-0-154-10.ec2.internal  Ready                      master   11m  v1.21.0
```

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。

   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
```

输出示例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
[[registry]]
  location = "registry.access.redhat.com/ubi8/"
  insecure = false
  blocked = false
  mirror-by-digest-only = true
  prefix = ""

  [[registry.mirror]]
    location = "example.io/example/ubi8-minimal"
    insecure = false

  [[registry.mirror]]
    location = "example.com/example/ubi8-minimal"
    insecure = false
```

e. 将镜像摘要从源拉取到节点，并检查是否通过镜像解析。**ImageContentSourcePolicy** 对象仅支持镜像摘要，不支持镜像标签。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6
```

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。

8.5. 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。

8.5.1. 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. 选择一个**更新频道**（如有多个可用）。

c. 如前面所述，选择**自动**或**手动**批准策略。

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。

8.5.2. 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。
- 在您的本地系统安装 **oc** 命令。

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
...
```

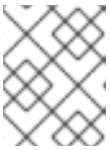
记录下所需 Operator 的目录。

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。如果您要使用 **AllNamespaces** 安装 Operator，则 **openshift-operators** 命名空间已有适当的 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。

   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 **AllNamespaces** 安装模式的使用，指定 **openshift-operators** 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 VolumeMounts 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. 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 9 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。

9.1. 将通知发送到外部系统

在 OpenShift Container Platform 4.8 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

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。

9.1.1. 配置警报接收器

您可以配置警报接收器，以确保了解集群出现的重要问题。

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. 在 **Administrator** 视角中，导航到 **Administration → Cluster Settings → Global Configuration → Alertmanager**。

注意

或者，您还可以通过 notification drawer 访问同一页面。选择 OpenShift Container Platform Web 控制台右上角的铃铛图标，并在 **AlertmanagerReceiverNotConfigured** 警报中选择 **Configure**。

2. 在该页面的 **Receivers** 部分中选择 **Create Receiver**。
3. 在 **Create Receiver** 表单中，添加 **Receiver Name**，然后从列表中选择 **Receiver Type**。
4. 编辑接收器配置：
    - 对于 PagerDuty 接收器：

a. 选择集成类型并添加 PagerDuty 集成密钥。

b. 添加 PagerDuty 安装的 URL。

c. 如果要编辑客户端和事件详情或严重性规格，请选择 **Show advanced configuration**。

- 对于 Webhook 接收器：

a. 添加将 HTTP POST 请求发送到的端点。

b. 如果要编辑将已解析的警报发送给接收器的默认选项，请选择 **Show advanced configuration**。

- 对于电子邮件接收器：

a. 添加要将通知发送到的电子邮件地址。

b. 添加 SMTP 配置详情，包括发送通知的地址、用来发送电子邮件的智能主机和端口号、SMTP 服务器的主机名以及验证详情。

c. 选择是否需要 TLS。

d. 如果要将默认选项编辑为不向接收器发送已解析的警报，或编辑电子邮件通知正文配置，请选择 **Show advanced configuration**。

- 对于 Slack 接收器：

a. 添加 Slack Webhook 的 URL。

b. 添加要将通知发送到的 Slack 频道或用户名。

c. 如果要将默认选项编辑为不向接收器发送已解析的警报，或编辑图标和用户名配置，请选择 **Show advanced configuration**。您还可以选择是否查找并链接频道名称和用户名。

5. 默认情况下，如果触发的警报带有与所有选择器匹配的标签，将被发送到接收器。如果您希望触发的警报在发送到接收器之前具有完全匹配的标签值：

a. 在表单的 **Routing Labels** 部分中添加路由标签名称和值。

b. 如果想要使用正则表达式，请选择 **Regular Expression**。

c. 选择 **Add Label** 来添加更多路由标签。

6. 选择 **Create** 来创建接收器。

9.2. 其他资源

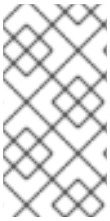
- 监控概述
- 管理警报

第 10 章 在 IBM Z 或 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 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 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

10.1. 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 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: ""
```

10.1.1. 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。

10.1.2. 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。

10.1.3. 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。

10.1.4. 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

10.2. 手动配置附加设备

本节中的任务描述了如何在 IBM Z 或 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 0.0.8000
  sudo chzdev -e 1000-1002
  sude chzdev -e 4444
  sudo chzdev -e 0.0.8000:0x500507680d760026:0x00bc000000000000
```

其他资源

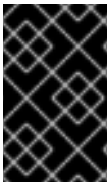
请参阅 IBM 文档中的持久性设备配置。

10.3. 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。

10.4. 为 FCP LUN 启用多路径

本节中的任务描述了如何在 IBM Z 或 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 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 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