OpenShift Container Platform 4.16

机器配置

在 OpenShift Container Platform 中管理和应用基本操作系统和容器运行时的配置和更新

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 机器配置

在 OpenShift Container Platform 中管理和应用基本操作系统和容器运行时的配置和更新

法律通告

摘要

本文档提供了有关使用 MachineConfig、Kubelet 和 ContainerRuntimeConfig 对象管理对 systemd、CRI-O、Kubelet、内核和其他系统功能的更改的说明。另外，镜像分层（image layering）允许您通过将额外镜像分层到任何集群 worker 节点的基础镜像中，轻松地自定义底层节点操作系统。

目录

第 1 章 机器配置概述

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。例如，您可以使用 Machine Config Operator (MCO) 和机器配置来管理对 systemd、CRI-O 和 kubelet、内核、Network Manager 和其他系统功能的更新。

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

重要

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

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

1.1. 关于 MACHINE CONFIG OPERATOR

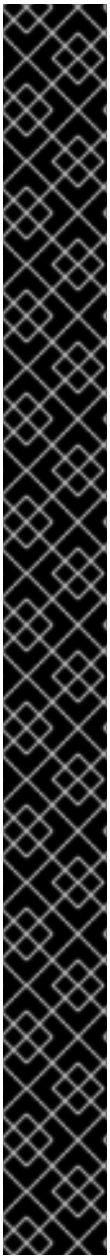
OpenShift Container Platform 4.16 集成了操作系统和集群管理。由于集群管理自己的更新，包括集群节点上 Red Hat Enterprise Linux CoreOS (RHCOS) 的更新，因此 OpenShift Container Platform 提供了可靠的生命周期管理体验，能够简化节点升级的编配。

OpenShift Container Platform 使用三个守护进程集和控制器来简化节点管理。这些守护进程集通过使用标准的 Kubernetes 式构造来编配操作系统更新和主机配置更改。它们包括：

- **machine-config-controller**，协调从 control plane 进行的机器升级。它监控所有集群节点并编配其配置更新。
- **machine-config-daemon** 守护进程集在集群中的每个节点上运行，并根据 MachineConfigController 的指示将机器更新为机器配置定义的配置。 当节点检测到更改时，它会排空其 pod，应用更新并重启。这些更改以 Ignition 配置文件的形式出现，这些文件应用指定的机器配置并控制 kubelet 配置。更新本身在容器中交付。此过程是成功管理 OpenShift Container Platform 和 RHCOS 更新的关键。
- **machine-config-server** 守护进程集，在加入集群时为 control plane 节点提供 Ignition 配置文件。

机器配置是 Ignition 配置的子集。**machine-config-daemon** 读取机器配置，以查看是否需要进行 OSTree 更新，或者是否必须应用一系列 systemd kubelet 文件更改、配置更改，或者对操作系统或 OpenShift Container Platform 配置的其他更改。

执行节点管理操作时，您可以创建或修改 **KubeletConfig** 自定义资源(CR)。



重要

当对机器配置进行修改时，Machine Config Operator（MCO） 会自动重启所有对应的节点，以使更改生效。

您可以使用节点中断策略缓解某些机器配置更改造成的中断。请参阅 *了解机器配置更改后节点重启行为*。

或者，您可以在进行更改前防止节点在机器配置更改后自动重启。通过在对应的机器配置池中将 **spec.paused** 字段设置为 **true** 来暂停自动引导过程。暂停后，机器配置更改不会生效，除非将 **spec.paused** 字段设置为 **false**，且节点已重启至新配置。

以下修改不会触发节点重新引导：

- 当 MCO 检测到以下任何更改时，它会在不排空或重启节点的情况下应用更新：
  - 在机器配置的 **spec.config.passwd.users.sshAuthorizedKeys** 参数中更改 SSH 密钥。
  - 在 **openshift-config** 命名空间中更改全局 pull secret 或 pull secret。
  - Kubernetes API Server Operator 自动轮转 **/etc/kubernetes/kubelet-ca.crt** 证书颁发机构（CA）。
- 当 MCO 检测到对 **/etc/containers/registries.conf** 文件的更改时，如添加或编辑 **ImageDigestMirrorSet**、**ImageTagMirrorSet** 或 **ImageContentSourcePolicy** 对象，它会排空对应的节点，应用更改并取消记录节点。对于以下更改，节点排空不会发生：
  - 增加了一个 registry，带有为每个镜像（mirror）设置了 **pull-from-mirror = "digest-only"** 参数。
  - 增加了一个镜像（mirror），带有在一个 registry 中设置的 **pull-from-mirror = "digest-only"** 参数。
  - 在 **unqualified-search-registries** 列表中添加项目。

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

其他资源

- 关于 OpenShift SDN 网络插件

1.2. 机器配置概述

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

- 机器配置按字母顺序处理，按字母顺序增加名称。呈现控制器使用列表中的第一个机器配置作为基础，并将剩余的附加到基本机器配置中。

- 机器配置可以对每个系统的操作系统上的文件或服务进行特定的更改，代表一个 OpenShift Container Platform 节点池。
- MCO 应用对机器池中的操作系统的更改。所有 OpenShift Container Platform 集群都以 worker 和 control plane 节点池开头。通过添加更多角色标签，您可以配置自定义节点池。例如，您可以设置一个自定义的 worker 节点池，其中包含应用程序所需的特定硬件功能。但是，本节中的示例着重介绍了对默认池类型的更改。

重要

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

- 机器配置更改后，MCO 根据 **topology.kubernetes.io/zone** 标签，按区字母更新受影响的节点。如果一个区域有多个节点，则首先更新最旧的节点。对于不使用区的节点，如裸机部署中的节点，节点会按使用的时间升级，首先更新最旧的节点。MCO 一次更新机器配置池中由 **maxUnavailable** 字段指定的节点数量。
- 在将 OpenShift Container Platform 安装到磁盘前，必须先进行一些机器配置。在大多数情况下，这可以通过创建直接注入 OpenShift Container Platform 安装程序进程中的机器配置来实现，而不必作为安装后机器配置运行。在其他情况下，您可能需要在 OpenShift Container Platform 安装程序启动时传递内核参数时进行裸机安装，以完成诸如设置每个节点的 IP 地址或高级磁盘分区等操作。
- MCO 管理机器配置中设置的项目。MCO 不会覆盖您对系统进行的手动更改，除非明确告知 MCO 管理冲突文件。换句话说，MCO 只提供您请求的特定更新，它不会声明对整个节点的控制。
- 强烈建议手动更改节点。如果您需要退出某个节点并启动一个新节点，则那些直接更改将会丢失。
- MCO 只支持写入 **/etc** 和 **/var** 目录里的文件，虽然有些目录的符号链接可以通过符号链接到那些区域之一来写入。例如 **/opt** 和 **/usr/local** 目录。
- Ignition 是 MachineConfig 中使用的配置格式。详情请参阅 Ignition 配置规格 v3.2.0。
- 虽然 Ignition 配置设置可以在 OpenShift Container Platform 安装时直接交付，且以 MCO 提供 Ignition 配置的方式格式化，但 MCO 无法查看这些原始 Ignition 配置是什么。因此，您应该在部署 Ignition 配置前将 Ignition 配置设置嵌套到机器配置中。
- 当由 MCO 管理的文件在 MCO 之外更改时，Machine Config Daemon（MCD）会将该节点设置为 **degraded**。然而，它不会覆盖这个错误的文件，它应该继续处于 **degraded（降级）**状态。
- 使用机器配置的一个关键原因是，当您为 OpenShift Container Platform 集群中的池添加新节点时，会应用它。**machine-api-operator** 置备一个新机器， MCO 配置它。

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

1.2.1. 机器配置可以更改什么？

MCO 可更改的组件类型包括：

- **config**：创建 Ignition 配置对象（请参阅 Ignition 配置规格），以完成修改 OpenShift Container Platform 机器上的文件、systemd 服务和其他功能，包括：

- **Configuration files**：创建或覆盖 **/var** 或 **/etc** 目录中的文件。
- **systemd units**：在附加设置中丢弃并设置 systemd 服务的状态，或者添加到现有 systemd 服务中。
- **用户和组**：在安装后更改 passwd 部分中的 SSH 密钥。

重要

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

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

重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。

当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

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

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

可以在安装后进行的 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。默认情况下，在 MCO 中进行的许多更改都需要重启。

以下修改不会触发节点重新引导：

- 当 MCO 检测到以下任何更改时，它会在不排空或重启节点的情况下应用更新：
  - 在机器配置的 **spec.config.passwd.users.sshAuthorizedKeys** 参数中更改 SSH 密钥。
  - 在 **openshift-config** 命名空间中更改全局 pull secret 或 pull secret。

- Kubernetes API Server Operator 自动轮转 **/etc/kubernetes/kubelet-ca.crt** 证书颁发机构（CA）。

- 当 MCO 检测到对 **/etc/containers/registries.conf** 文件的更改时，如添加或编辑 **ImageDigestMirrorSet**、**ImageTagMirrorSet** 或 **ImageContentSourcePolicy** 对象，它会排空对应的节点，应用更改并取消记录节点。对于以下更改，节点排空不会发生：

  - 增加了一个 registry，带有为每个镜像（mirror）设置了 **pull-from-mirror = "digest-only"** 参数。

  - 增加了一个镜像（mirror），带有在一个 registry 中设置的 **pull-from-mirror = "digest-only"** 参数。

  - 在 **unqualified-search-registries** 列表中添加项目。

在其他情况下，您可以使用 *节点中断策略*来减少 MCO 更改时造成工作负载中断的情况。如需更多信息，请参阅 *了解在机器配置更改后节点重启的行为*。

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

1.3. 了解 MACHINE CONFIG OPERATOR 节点排空行为

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

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

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

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

如果 MCO 在 master 节点上排空 pod，请注意以下条件：

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

注意

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

使用节点中断策略或禁用 control plane 重启，可以缓解排空和重启周期造成的中断。如需更多信息，请参阅"了解机器配置更改后节点重启的行为"和"禁用 Machine Config Operator 自动重新引导"。

其他资源

- 关于 Machine Config Operator
- 使用节点中断策略最小化对机器配置更改的中断
- 禁用 Machine Config Operator 自动重新引导

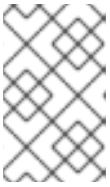
1.4. 了解配置偏移检测

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

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

MCD 在出现任何以下条件时执行配置偏移检测：

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



注意

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

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

如果 MCD 检测到配置偏移，MCD 执行以下任务：

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

您可以通过列出 MCP 检查是否有降级的节点：

```
$ oc get mcp worker
```

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

输出示例

```
NAME    CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
```

```
DEGRADEDMACHINECOUNT   AGE
worker   rendered-worker-404caf3180818d8ac1f50c32f14b57c3   False     True      True       2
1                 1                   1                   5h51m
```

您可以通过检查机器配置池来确定问题是否由配置偏移导致：

```
$ oc describe mcp worker
```

输出示例

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

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

2 节点的状态为 **NodeDegraded**。

或者，如果您知道哪个节点已降级，请检查该节点：

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

输出示例

```
 ...

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

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

2 节点的状态为 **Degraded**。

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

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

注意

在节点上生成强制文件会导致该节点重新引导。

1.5. 检查机器配置池状态

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

流程

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

```
$ oc get machineconfigpool
```

输出示例

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

其中：

UPDATED

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

UPDATING

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

DEGRADED

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

MACHINECOUNT

表示该 MCP 中的机器总数。

READYMACHINECOUNT

指明 MCP 中准备进行调度的机器总数。

UPDATEDMACHINECOUNT

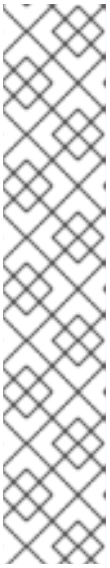
指明 MCP 中有当前机器配置的机器总数。

DEGRADEDMACHINECOUNT

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

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

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



注意

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

输出示例

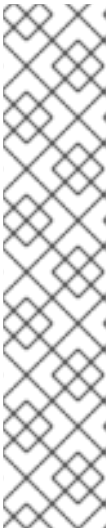
```
NAME     CONFIG                 UPDATED  UPDATING   DEGRADED
MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
master   rendered-master-06c9c4…  True     False      False    3          3
3                  0                  4h42m
worker   rendered-worker-c1b41a…  False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

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

```
$ oc describe mcp worker
```

输出示例

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



注意

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

输出示例

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

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

```
$ oc get machineconfigs
```

输出示例

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

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

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

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

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

输出示例

```
Name:         01-master-kubelet
...
Spec:
  Config:
    Ignition:
      Version:  3.2.0
    Storage:
      Files:
        Contents:
          Source:   data:,
        Mode:       420
        Overwrite:  true
        Path:       /etc/kubernetes/cloud.conf
```

```
        Contents:
          Source:
data:,kind%3A%20KubeletConfiguration%0AapiVersion%3A%20kubelet.config.k8s.io%2Fv1beta1%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
```

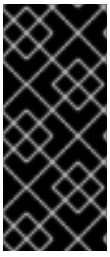
如果这是唯一的问题，则受影响池中的节点应返回非降级状态。这会导致呈现的配置回滚到其之前更改的状态。

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。

1.6. 检查机器配置节点状态

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

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



重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

流程

1. 运行以下命令，获取所有机器配置池中所有节点的更新状态概述：

```
$ oc get machineconfignodes
```

输出示例

```
NAME                    UPDATED   UPDATEPREPARED   UPDATEEXECUTED   UPDATEPOSTACTIONCOMPLETED   UPDATECOMPLETED   RESUMED
```

```
ip-10-0-12-194.ec2.internal   True     False      False      False      False
False
ip-10-0-17-102.ec2.internal   False    True       False      False      False
False
ip-10-0-2-232.ec2.internal    False    False      True       False      False
False
ip-10-0-59-251.ec2.internal   False    False      False      True       False
False
ip-10-0-59-56.ec2.internal    False    False      False      False      True
True
ip-10-0-6-214.ec2.internal    False    False      Unknown    False      False
False
```

其中：

UPDATED

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

UPDATEPREPARED

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

UPDATEEXECUTED

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

UPDATEPOSTACTIONCOMPLETED

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

UPDATECOMPLETED

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

RESUMED

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

注意

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

2. 运行以下命令，检查特定机器配置池中节点的更新状态：

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

1 池的名称是 **MachineConfigPool** 对象名称。

输出示例

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

3. 运行以下命令，检查单个节点的更新状态：

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

1 节点的名称是 **MachineConfigNode** 对象名称。

输出示例

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

```
Health:              Healthy
Most Recent Error:
Observed Generation:  3
```

1. 在节点上检测到新配置时，**spec.configversion.desired** 字段指定所需的配置会立即更新。
2. 只有在 Machine Config Daemon (MCD)验证新配置时，**status.configversion.desired** 字段中指定的配置才会更新。MCD 通过检查更新的当前阶段来执行验证。如果更新成功通过 **UPDATEPREPARED** 阶段，则状态会添加新配置。

1.7. 了解 MACHINE CONFIG OPERATOR 证书

Machine Config Operator 证书用于保护 Red Hat Enterprise Linux CoreOS (RHCOS) 节点和 Machine Config Server 之间的连接。如需更多信息，请参阅 Machine Config Operator 证书。

1.7.1. 查看证书并与其交互

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

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

MCC 还处理镜像 registry 证书及其关联的用户捆绑包证书。

您可以获取有关列出的证书的信息，包括减少证书的捆绑包，以及签名和主题数据。

流程

- 运行以下命令来获取详细的证书信息：

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

输出示例

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

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

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

输出示例

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

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

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

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

输出示例

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

第 2 章 使用机器配置对象配置节点

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

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

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

提示

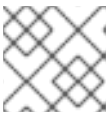
使用 "Configuring chrony time service" 部分作为如何将其他配置文件添加到 OpenShift Container Platform 节点的模型。

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: 4.16.0
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

2.2. 禁用 CHRONY 时间服务

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

先决条件

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

流程

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role> 1
  name: disable-chronyd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
```

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

1 要禁用 **chronyd** 的节点角色，如 **master**。

b. 运行以下命令来创建 **MachineConfig** CR：

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

2.3. 为节点添加内核参数

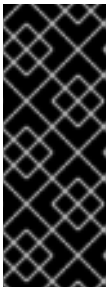
在一些特殊情况下，您可能需要为集群中的一组节点添加内核参数。进行此操作时应小心谨慎，而且您必须先清楚了解所设参数的影响。

警告

不当使用内核参数会导致系统变得无法引导。

您可以设置的内核参数示例包括：

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



重要

cgroup v1 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

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。

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

警告

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

如需内核参数的列表和描述，请参阅 Kernel.org 内核参数。

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

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

先决条件

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

流程

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

```
$ oc get MachineConfig
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
```

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

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

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

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

3. 创建新机器配置：

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

4. 检查机器配置以查看是否添加了新配置：

```
$ oc get MachineConfig
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
```

```
01-master-container-runtime                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-master-kubelet                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-container-runtime                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-kubelet                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
05-worker-kernelarg-selinuxpermissive                                  3.2.0          105s
99-master-generated-registries             52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-master-ssh                                                          3.2.0          40m
99-worker-generated-registries             52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-worker-ssh                                                          3.2.0          40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
```

5. 检查节点：

```
$ oc get nodes
```

输出示例

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.29.4
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.29.4
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.29.4
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.29.4
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.29.4
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.29.4
```

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
```

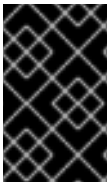
您应看到 **enforcing=0** 参数已添加至其他内核参数。

2.4. 在 RHCOS 上启用带有内核参数的多路径

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"启用多路径安装后"。

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

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

重要

当在配置了多路径的 IBM Power® 上的带有 "vSCSI" 存储的单个 VIOS 主机中安装或配置了 OpenShift Container Platform 4.16 集群作为安装后任务时，启用了多路径的 CoreOS 节点无法引导。此行为是正常的，因为只有一个路径可用于节点。

先决条件

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

流程

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

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

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

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

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

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

4. 检查机器配置以查看是否添加了新配置：

```
$ oc get MachineConfig
```

输出示例

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

5. 检查节点：

```
$ oc get nodes
```

输出示例

-

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.29.4
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.29.4
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.29.4
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.29.4
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.29.4
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.29.4
```

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。

2.5. 在节点中添加实时内核

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

如果您的 OpenShift Container Platform 工作负载需要这些实时特征，您可以将机器切换到 Linux 实时内核。对于 OpenShift Container Platform，4.16 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  v1.29.4
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.29.4
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.29.4
```

```
$ 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
```

2.6. 配置 JOURNALD 设置

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

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

先决条件

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

流程

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

注意

有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
variant: openshift
version: 4.16.0
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

2.7. 为 RHCOS 添加扩展

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

目前，有以下扩展可用：

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

以下流程描述了如何使用机器配置为 RHCOS 节点添加一个或多个扩展。

先决条件

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

流程

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

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

2. 将机器配置添加到集群。键入以下内容将机器配置添加到集群中：

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

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

3. 检查是否应用了扩展：

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

输出示例

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

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

```
$ oc get machineconfigpool
```

输出示例

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

5. 检查扩展。要检查是否应用了扩展，请运行：

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

输出示例

```
NAME                                         STATUS  ROLES    AGE   VERSION
ip-10-0-169-2.us-east-2.compute.internal     Ready   worker   102m  v1.29.4
```

```
$ 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
```

2.8. 在机器配置清单中载入自定义固件 BLOB

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

流程

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

注意

有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

自定义固件 blob 的 Butane 配置文件

```
variant: openshift
version: 4.16.0
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

2.9. 更改节点访问的核心用户密码

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

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

注意

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

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

流程

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

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

输出示例

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

2. 创建包含 **core** 用户名和散列密码的机器配置文件：

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

(1) 这必须是 **core**。

(2) 与 **core** 帐户一起使用的散列密码。

3. 运行以下命令来创建机器配置：

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

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

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

验证

1. 节点返回 **UPDATED=True** 状态后，运行以下命令为节点启动 debug 会话：

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

2. 运行以下命令，将 **/host** 设置为 debug shell 中的根目录：

```
sh-4.4# chroot /host
```

3. 检查 **/etc/shadow** 文件的内容：

输出示例

```
...
core:$6$2sE/010goDuRSxxv$o18K52wor.wlwZp:19418:0:99999:7:::
...
```

哈希密码分配给 **core** 用户。

第 3 章 使用节点中断策略最小化对机器配置更改的中断

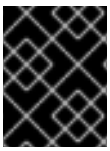
默认情况下，当您对 **MachineConfig** 对象中的字段进行某些更改时，Machine Config Operator (MCO) 会排空并重启与该机器配置关联的节点。但是，您可以创建一个 *节点中断策略*，在策略中定义一组对某些 Ignition 配置对象的更改，它们对负载只有非常少的（或没有）中断影响。

节点中断策略允许您定义哪些配置改变会对集群造成中断，哪些改变不会造成中断。这可让您对在集群中进行了小的集群配置变化时减少节点停机的时间。要配置策略，您可以修改位于 **openshift-machine-config-operator** 命名空间中的 **MachineConfiguration** 对象。请参阅以下 **MachineConfiguration** 对象中的节点中断策略示例。

注意

无论任何节点中断策略是什么，对机器配置的更改始终需要重启。如需更多信息，请参阅 *关于 Machine Config Operator* 。

创建节点中断策略后，MCO 会验证策略来搜索文件中潜在的问题，如格式化问题。然后，MCO 将策略与集群默认值合并，并为机器配置中的 **status.nodeDisruptionPolicyStatus** 字段填充在将来的机器配置中要执行的操作。策略中的配置总是覆盖集群默认值。



重要

MCO 不会验证节点中断策略是否能够成功应用更改。因此，您需要确保节点中断策略的准确性。

例如，您可以配置节点中断策略，以便 sudo 配置不需要节点排空和重新引导。或者，您可以配置集群，以便对 **sshd** 进行的更新仅会重新加载该服务。

重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

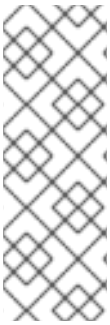
您可以在对以下 Ignition 配置对象进行更改时控制 MCO 的行为：

- **配置文件** ：在 **/var** 或 **/etc** 目录中添加了文件或更新了其中的文件。
- **systemd 单元** ：创建和设置 systemd 服务的状态，或修改现有的 systemd 服务。
- **用户和组** ：修改了 postinstallation 中的 **passwd** 部分中的 SSH 密钥。
- **ICSP,ITMS,IDMS** 对象：您可以从 **ImageContentSourcePolicy** (ICSP)、**ImageTagMirrorSet** (ITMS) 和 **ImageDigestMirrorSet** (IDMS) 对象中删除镜像规则。

当您进行任何这样的更改时，节点中断策略将决定在 MCO 实现更改时，需要进行以下的哪些操作：

- **Reboot**: MCO 排空并重启节点。这是默认的行为。
- **None**: MCO 不排空或重启节点。MCO 在不进行其他操作的情况下应用更改。
- **Drain** ：MCO 会封锁并排空其工作负载的节点。工作负载使用新配置重启。

- **Reload**: 对于服务，MCO 会在不重启该服务的情况下重新载入指定的服务。
- **Restart** ：对于服务，MCO 会完全重启指定的服务。
- **DaemonReload** ：MCO 重新加载 systemd 管理器配置。
- **Special** ：这是一个内部的 MCO 操作，用户无法设置。



注意

- **Reboot** 和 **None** 操作不能用于任何其他操作，因为 **Reboot** 和 **None** 操作会覆盖其他操作。
- 操作会按照节点中断策略列表中设置的顺序应用。
- 如果您进行了其他机器配置更改，它们需要重新引导节点或对节点有其他的中断影响，则重新引导会取代节点中断策略操作。

3.1. 节点中断策略示例

以下示例 **MachineConfiguration** 对象包含了一个节点中断策略。

提示

MachineConfiguration 对象和 **MachineConfig** 对象是不同的对象。**MachineConfiguration** 对象是 MCO 命名空间中的一个单个对象，其中包含 MCO operator 的配置参数。**MachineConfig** 对象定义了要应用到机器配置池的更改。

以下示例 **MachineConfiguration** 对象显示没有用户定义的策略。默认节点中断策略值显示在 **status** 小节中。

默认节点中断策略

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
  name: cluster
spec:
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
status:
  nodeDisruptionPolicyStatus:
    clusterPolicies:
      files:
      - actions:
        - type: None
        path: /etc/mco/internal-registry-pull-secret.json
      - actions:
        - type: None
        path: /var/lib/kubelet/config.json
      - actions:
        - reload:
            serviceName: crio.service
          type: Reload
```

```
      path: /etc/machine-config-daemon/no-reboot/containers-gpg.pub
    - actions:
      - reload:
          serviceName: crio.service
        type: Reload
      path: /etc/containers/policy.json
    - actions:
      - type: Special
      path: /etc/containers/registries.conf
    sshkey:
      actions:
      - type: None
  readyReplicas: 0
```

在以下示例中，当对 SSH 密钥进行了更改时，MCO 会排空集群节点，重新载入 **crio.service**，重新载入 systemd 配置，并重启 **crio-service**。

SSH 密钥更改的节点中断策略示例

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    sshkey:
      actions:
      - type: Drain
      - reload:
          serviceName: crio.service
        type: Reload
      - type: DaemonReload
      - restart:
          serviceName: crio.service
        type: Restart
# ...
```

在以下示例中，当对 **/etc/chrony.conf** 目录中的文件进行了更改时，MCO 会重新载入集群节点上的 **chronyd.service**。

配置文件更改的节点中断策略示例

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    files:
    - actions:
      - reload:
```

```
        serviceName: chronyd.service
      type: Reload
    path: /etc/chrony.conf
```

在以下示例中，当对 **auditd.service** systemd 单元进行了更改时，MCO 会排空集群节点，重新载入 **crio.service**，重新载入 systemd 管理器配置，并重启 **crio.service**。

配置文件更改的节点中断策略示例

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    units:
      - name: auditd.service
        actions:
          - type: Drain
          - type: Reload
            reload:
              serviceName: crio.service
          - type: DaemonReload
          - type: Restart
            restart:
              serviceName: crio.service
```

在以下示例中，当对 **registry.conf** 目录中的文件进行修改时，MCO 不会排空或重启节点，并应用没有进一步操作的更改。

配置文件更改的节点中断策略示例

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
      - actions:
        - type: None
        path: /etc/containers/registries.conf
```

3.2. 在机器配置更改时配置节点重启行为

您可以创建一个节点中断策略来定义，哪些机器配置改变会对集群造成中断，哪些改变不会造成中断。

您可以控制节点如何响应对 **/var** 或 **/etc** 目录中的文件、systemd 单元、SSH 密钥和 **registry.conf** 文件的更改。

当您进行任何这样的更改时，节点中断策略将决定在 MCO 实现更改时，需要进行以下的哪些操作：

- **Reboot**: MCO 排空并重启节点。这是默认的行为。
- **None**: MCO 不排空或重启节点。MCO 在不进行其他操作的情况下应用更改。
- **Drain** ：MCO 会封锁并排空其工作负载的节点。工作负载使用新配置重启。
- **Reload**: 对于服务，MCO 会在不重启该服务的情况下重新载入指定的服务。
- **Restart** ：对于服务，MCO 会完全重启指定的服务。
- **DaemonReload** ：MCO 重新加载 systemd 管理器配置。
- **Special** ：这是一个内部的 MCO 操作，用户无法设置。

注意

- **Reboot** 和 **None** 操作不能用于任何其他操作，因为 **Reboot** 和 **None** 操作会覆盖其他操作。
- 操作会按照节点中断策略列表中设置的顺序应用。
- 如果您进行了其他机器配置更改，它们需要重新引导节点或对节点有其他的中断影响，则重新引导会取代节点中断策略操作。

先决条件

- 已使用功能门启用 **TechPreviewNoUpgrade** 功能集。如需更多信息，请参阅“使用功能门启用功能”。

警告

在集群中启用 **TechPreviewNoUpgrade** 功能集可防止次版本更新。**TechPreviewNoUpgrade** 功能集无法被禁用。不要在生产环境集群中启用此功能。

流程

1. 编辑 **machineconfigurations.operator.openshift.io** 对象以定义节点中断策略：

```
$ oc edit MachineConfiguration cluster -n openshift-machine-config-operator
```

2. 添加类似如下的节点中断策略：

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
# ...
spec:
  nodeDisruptionPolicy: 1
```

```
    files: 2
    - actions: 3
      - reload: 4
          serviceName: chronyd.service 5
        type: Reload
      path: /etc/chrony.conf 6
    sshkey: 7
      actions:
      - type: Drain
      - reload:
          serviceName: crio.service
        type: Reload
      - type: DaemonReload
      - restart:
          serviceName: crio.service
        type: Restart
    units: 8
    - actions:
      - type: Drain
      - reload:
          serviceName: crio.service
        type: Reload
      - type: DaemonReload
      - restart:
          serviceName: crio.service
        type: Restart
      name: test.service
```

1. 指定节点中断策略。
2. 指定对这些路径进行修改的机器配置文件定义和操作列表。此列表支持最多 50 个条目。
3. 指定在更改指定文件时要执行的一系列操作。操作按此列表中设置的顺序应用。此列表支持最多 10 个条目。
4. 指定在对指定文件更改时，重新加载这里列出的服务。
5. 指定要执行操作的服务的完整名称。
6. 指定由机器配置管理的文件位置。当对 **path** 中的文件进行了更改时，策略中的操作将应用。
7. 指定在对集群中的 SSH 密钥更改时要执行的操作列表。
8. 指定在更改这些单元时要执行的 systemd 单元名称和操作的列表。

验证

- 查看您创建的 **MachineConfiguration** 对象文件：

```
$ oc get MachineConfiguration/cluster -o yaml
```

输出示例

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: cluster
# ...
status:
  nodeDisruptionPolicyStatus: 1
    clusterPolicies:
      files:
# ...
      - actions:
        - reload:
            serviceName: chronyd.service
          type: Reload
        path: /etc/chrony.conf
      sshkey:
        actions:
        - type: Drain
        - reload:
            serviceName: crio.service
          type: Reload
        - type: DaemonReload
        - restart:
            serviceName: crio.service
          type: Restart
      units:
      - actions:
        - type: Drain
        - reload:
            serviceName: crio.service
          type: Reload
        - type: DaemonReload
        - restart:
            serviceName: crio.service
          type: Restart
        name: test.se
# ...
```

1 指定当前的 cluster-validated 策略。

第 4 章 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，会影响 kubelet 和 CRI-O 容器运行时服务的行为。

4.1. 创建 KUBELETCONFIG CRD 来编辑 KUBELET 参数

kubelet 配置目前被序列化为 Ignition 配置，因此可以直接编辑。但是，在 Machine Config Controller (MCC) 中同时添加了新的 **kubelet-config-controller**。这可让您使用 **KubeletConfig** 自定义资源 (CR) 来编辑 kubelet 参数。

注意

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

请考虑以下指导：

- 编辑现有的 **KubeletConfig** CR 以修改现有设置或添加新设置，而不是为每个更改创建一个 CR。建议您仅创建一个 CR 来修改不同的机器配置池，或用于临时更改，以便您可以恢复更改。
- 为每个机器配置池创建一个 **KubeletConfig** CR，带有该池需要更改的所有配置。
- 根据需要，创建多个 **KubeletConfig** CR，每个集群限制为 10。对于第一个 **KubeletConfig** CR，Machine Config Operator (MCO) 会创建一个机器配置，并附带 **kubelet**。对于每个后续 CR，控制器会创建另一个带有数字后缀的 **kubelet** 机器配置。例如，如果您有一个带有 **-2** 后缀的 **kubelet** 机器配置，则下一个 **kubelet** 机器配置会附加 **-3**。

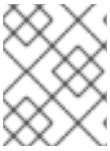
注意

如果要将 kubelet 或容器运行时配置应用到自定义机器配置池，则 **machineConfigSelector** 中的自定义角色必须与自定义机器配置池的名称匹配。

例如，由于以下自定义机器配置池名为 **infra**，因此自定义角色也必须是 **infra**：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
# ...
```

如果要删除机器配置，以相反的顺序删除它们，以避免超过限制。例如，在删除 **kubelet-2** 机器配置前删除 **kubelet-3** 机器配置。



注意

如果您有一个带有 **kubelet-9** 后缀的机器配置，并且创建了另一个 **KubeletConfig** CR，则不会创建新的机器配置，即使少于 10 个 **kubelet** 机器配置。

KubeletConfig CR 示例

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

显示 KubeletConfig 机器配置示例

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
26m
...
```

以下流程演示了如何配置 worker 节点上的每个节点的最大 pod 数量。

先决条件

1. 为您要配置的节点类型获取与静态 **MachineConfigPool** CR 关联的标签。执行以下步骤之一：
   a. 查看机器配置池：

   ```
   $ oc describe machineconfigpool <name>
   ```

   例如：

   ```
   $ oc describe machineconfigpool worker
   ```

   输出示例

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: set-max-pods 1
   ```

   1 如果添加了标签，它会出现在 **labels** 下。

   b. 如果标签不存在，则添加一个键/值对：

   ```
   $ oc label machineconfigpool worker custom-kubelet=set-max-pods
   ```

流程

1. 查看您可以选择的可用机器配置对象：

```
$ oc get machineconfig
```

默认情况下，与 kubelet 相关的配置为 **01-master-kubelet** 和 **01-worker-kubelet**。

2. 检查每个节点的最大 pod 的当前值：

```
$ oc describe node <node_name>
```

例如：

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

在 **Allocatable** 小节中找到 **value: pods: <value>**：

输出示例

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

3. 通过创建一个包含 kubelet 配置的自定义资源文件，设置 worker 节点上的每个节点的最大 pod：

重要

以特定机器配置池为目标的 kubelet 配置也会影响任何依赖的池。例如，为包含 worker 节点的池创建 kubelet 配置也适用于任何子集池，包括包含基础架构节点的池。要避免这种情况，您必须使用仅包含 worker 节点的选择表达式创建新的机器配置池，并让 kubelet 配置以这个新池为目标。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods 1
  kubeletConfig:
    maxPods: 500 2
```

1. 输入机器配置池中的标签。
2. 添加 kubelet 配置。在本例中，使用 **maxPods** 设置每个节点的最大 pod。

注意

kubelet 与 API 服务器进行交互的频率取决于每秒的查询数量 (QPS) 和 burst 值。如果每个节点上运行的 pod 数量有限，使用默认值（**kubeAPIQPS** 为 **50**，**kubeAPIBurst** 为 **100**）就可以。如果节点上有足够 CPU 和内存资源，则建议更新 kubelet QPS 和 burst 速率。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 为带有标签的 worker 更新机器配置池：

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

b. 创建 **KubeletConfig** 对象：

```
$ oc create -f change-maxPods-cr.yaml
```

c. 验证 **KubeletConfig** 对象是否已创建：

```
$ oc get kubeletconfig
```

输出示例

```
NAME              AGE
set-max-pods      15m
```

根据集群中的 worker 节点数量，等待每个 worker 节点被逐个重启。对于有 3 个 worker 节点的集群，这个过程可能需要大约 10 到 15 分钟。

4. 验证更改是否已应用到节点：

a. 在 worker 节点上检查 **maxPods** 值已更改：

```
$ oc describe node <node_name>
```

b. 找到 **Allocatable** 小节：

```
 ...
Allocatable:
  attachable-volumes-gce-pd:  127
  cpu:                        3500m
  ephemeral-storage:          123201474766
```

```
  hugepages-1Gi:     0
  hugepages-2Mi:     0
  memory:            14225400Ki
  pods:              500 (1)
 ...
```

(1) 在本例中，**pods** 参数应报告您在 **KubeletConfig** 对象中设置的值。

5. 验证 **KubeletConfig** 对象中的更改：

```
$ oc get kubeletconfigs set-max-pods -o yaml
```

这应该显示 **True** 状态和 **type:Success**，如下例所示：

```
spec:
  kubeletConfig:
    maxPods: 500
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
status:
  conditions:
  - lastTransitionTime: "2021-06-30T17:04:07Z"
    message: Success
    status: "True"
    type: Success
```

4.2. 创建 CONTAINERRUNTIMECONFIG CR 以编辑 CRI-O 参数

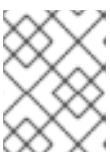
您可以为与特定机器配置池（MCP）关联的节点更改与 OpenShift Container Platform CRI-O 运行时关联的一些设置。通过使用 **ContainerRuntimeConfig** 自定义资源（CR），您可以设置配置值并添加一个标签以匹配 MCP。然后，MCO 会使用更新的值重建关联节点上的 **crio.conf** 和 **storage.conf** 配置文件。

注意

要使用 **ContainerRuntimeConfig** CR 恢复实现的更改，您必须删除 CR。从机器配置池中删除标签不会恢复更改。

您可以使用 **ContainerRuntimeConfig** CR 修改以下设置：

- **PIDs limit** ：在 **ContainerRuntimeConfig** 中设置 PID 限值将被弃用。如果需要 PIDs 限制，建议在 **KubeletConfig** CR 中使用 **podPidsLimit** 字段。**podPidsLimit** 字段的默认值为 **4096**。



注意

CRI-O 标志应用到容器的 cgroup 上，而 Kubelet 标志则在 pod 的 cgroup 中设置。请相应地调整 PID 限值。

- **日志级别**: **logLevel** 参数设置 CRI-O **log_level** 参数，即日志消息的详细程度。默认为 **info** (**log_level = info**)。其他选项包括 **fatal**、**panic**、**error**、**warn**、**debug** 和 **trace**。

- **Overlay 大小**：**overlaySize** 参数设置 CRI-O Overlay 存储驱动程序 **size** 参数，这是容器镜像的最大大小。
- **最大日志大小** ：在 **ContainerRuntimeConfig** 中设置最大日志大小被弃用。如果需要最大日志大小，建议在 **KubeletConfig** CR 中使用 **containerLogMaxSize** 字段。
- **容器运行时** ： **defaultRuntime** 参数将容器运行时设置为 **runc** 或 **crun**。默认为 **runc**。

您应该为每个机器配置池有一个**ContainerRuntimeConfig** CR，并为该池分配所有配置更改。如果要将相同的内容应用到所有池，则所有池只需要 **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-overlay  15m
ctr-level    5m45s
```

显示多个 containerruntime 机器配置示例

```
$ oc get mc | grep container
```

输出示例

```
...
01-master-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
57m
...
01-worker-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
57m
...
99-worker-generated-containerruntime        b5c5119de007945b6fe6fb215db3b8e2ceb12511
```

```
3.2.0             26m
99-worker-generated-containerruntime-1       b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0             17m
99-worker-generated-containerruntime-2       b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0             7m26s
...
```

以下示例将 **log_level** 字段设置为 **debug**，并将覆盖大小设置为 8 GB：

ContainerRuntimeConfig CR 示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: '' (1)
 containerRuntimeConfig:
   logLevel: debug (2)
   overlaySize: 8G (3)
   defaultRuntime: "crun" (4)
```

1. 指定机器配置池标签。对于容器运行时配置，角色必须与关联的机器配置池的名称匹配。
2. 可选：指定日志消息的详细程度。
3. 可选：指定容器镜像的最大大小。
4. 可选：指定部署到新容器的容器运行时。默认值为 **runc**。

流程

使用 **ContainerRuntimeConfig** CR 更改 CRI-O 设置：

1. 为 **ContainerRuntimeConfig** CR 创建 YAML 文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: '' (1)
 containerRuntimeConfig: (2)
   logLevel: debug
   overlaySize: 8G
```

1. 为您要修改的机器配置池指定一个标签。
2. 根据需要设置参数。

2. 创建 **ContainerRuntimeConfig** CR：

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

3. 验证是否已创建 CR：

```
$ oc get ContainerRuntimeConfig
```

输出示例

```
NAME          AGE
overlay-size  3m19s
```

4. 检查是否创建了新的 **containerruntime** 机器配置：

```
$ oc get machineconfigs | grep containerrun
```

输出示例

```
99-worker-generated-containerruntime  2c9371fbb673b97a6fe8b1c52691999ed3a1bfc2
3.2.0  31s
```

5. 监控机器配置池，直到所有系统都显示为 ready 状态：

```
$ oc get mcp worker
```

输出示例

```
NAME    CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
worker  rendered-worker-169  False    True      False     3          1                1                  0
9h
```

6. 验证设置是否在 CRI-O 中应用：

   a. 打开到机器配置池中节点的 **oc debug** 会话，并运行 **chroot /host**。

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

```
sh-4.4# chroot /host
```

   b. 验证 **crio.conf** 文件中的更改：

```
sh-4.4# crio config | grep 'log_level'
```

输出示例

```
log_level = "debug"
```

   c. 验证 'storage.conf' 文件中的更改：

```
sh-4.4# head -n 7 /etc/containers/storage.conf
```

输出示例

```
[storage]
  driver = "overlay"
  runroot = "/var/run/containers/storage"
  graphroot = "/var/lib/containers/storage"
  [storage.options]
    additionalimagestores = []
    size = "8G"
```

4.3. 使用 CRI-O 为 OVERLAY 设置默认的最大容器根分区大小

每个容器的根分区显示底层主机的所有可用磁盘空间。按照以下说明，为所有容器的 root 磁盘设置最大分区大小。

要配置最大 Overlay 大小以及其他 CRI-O 选项，您可以创建以下 **ContainerRuntimeConfig** 自定义资源定义 (CRD)：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     custom-crio: overlay-size
 containerRuntimeConfig:
   logLevel: debug
   overlaySize: 8G
```

流程

1. 创建配置对象：

   ```
   $ oc apply -f overlaysize.yml
   ```

2. 要将新的 CRI-O 配置应用到 worker 节点，请编辑 worker 机器配置池：

   ```
   $ oc edit machineconfigpool worker
   ```

3. 根据在 **ContainerRuntimeConfig** CRD 中设置的 **matchLabels** 名称添加 **custom-crio** 标签：

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: "2020-07-09T15:46:34Z"
     generation: 3
     labels:
       custom-crio: overlay-size
       machineconfiguration.openshift.io/mco-built-in: ""
   ```

4. 保存更改，然后查看机器配置：

```
$ oc get machineconfigs
```

新的 **99-worker-generated-containerruntime** 和 **rendered-worker-xyz** 对象被创建：

输出示例

```
99-worker-generated-containerruntime  4173030d89fbf4a7a0976d1665491a4d9a6e54f1
3.2.0             7m42s
rendered-worker-xyz                   4173030d89fbf4a7a0976d1665491a4d9a6e54f1   3.2.0
7m36s
```

5. 创建这些对象后，监控机器配置池以了解要应用的更改：

```
$ oc get mcp worker
```

worker 节点将 **UPDATING** 显示为 **True**，以及机器数量、更新的数字和其他详情：

输出示例

```
NAME   CONFIG            UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz False True False     3           2                2                0
20h
```

完成后，worker 节点会从 **UPDATING** 转换回 **False**，**UPDATEDMACHINECOUNT** 数与 **MACHINECOUNT** 数匹配：

输出示例

```
NAME   CONFIG            UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker   rendered-worker-xyz   True      False      False      3       3          3          0
20h
```

查看 worker 机器，您会看到新的 8 GB 最大大小配置适用于所有 worker：

输出示例

```
head -n 7 /etc/containers/storage.conf
[storage]
  driver = "overlay"
  runroot = "/var/run/containers/storage"
  graphroot = "/var/lib/containers/storage"
  [storage.options]
    additionalimagestores = []
    size = "8G"
```

在容器内，您会看到 root 分区现在为 8 GB：

输出示例

```
~ $ df -h
Filesystem              Size      Used Available Use% Mounted on
overlay                 8.0G      8.0K      8.0G   0% /
```

第 5 章 更新了引导镜像

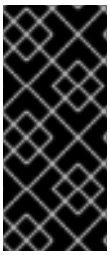
Machine Config Operator (MCO)使用引导镜像来启动 Red Hat Enterprise Linux CoreOS (RHCOS) 节点。默认情况下，OpenShift Container Platform 不管理引导镜像。

这意味着集群中的引导镜像不会随集群一起更新。例如，如果您的集群最初使用 OpenShift Container Platform 4.12 创建，集群用来创建节点的引导镜像是相同的 4.12 版本，即使集群是更新的版本。如果以后将集群升级到 4.13 或更高版本，新节点将继续使用相同的 4.12 镜像扩展。

这个过程可能会导致以下问题：

- 启动节点的额外时间
- 证书过期问题
- 版本偏移问题

要避免这些问题，您可以将集群配置为在更新集群时更新引导镜像。通过修改 **MachineConfiguration** 对象，您可以启用此功能。目前，更新引导镜像的功能仅适用于 Google Cloud Platform (GCP) 集群，且不支持由 Cluster API 管理的集群。



重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

要查看集群中使用的当前引导镜像，请检查机器集：

使用引导镜像引用的机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: ci-ln-hmy310k-72292-5f87z-worker-a
  namespace: openshift-machine-api
spec:
# ...
  template:
# ...
    spec:
# ...
      providerSpec:
# ...
        value:
          disks:
          - autoDelete: true
            boot: true
            image: projects/rhcos-cloud/global/images/rhcos-412-85-202203181601-0-gcp-x86-64 (1)
# ...
```

1 此引导镜像与最初安装的 OpenShift Container Platform 版本相同，在本例中为 OpenShift Container Platform 4.12，无论集群的当前版本是什么。在机器集中表示引导镜像的方式取决于平台，因为 **providerSpec** 字段的结构与平台不同。

如果您将集群配置为更新引导镜像，机器集中引用的引导镜像与集群的当前版本匹配。

5.1. 配置更新的引导镜像

默认情况下，OpenShift Container Platform 不管理引导镜像。您可以通过修改 **MachineConfiguration** 对象，将集群配置为在更新集群时更新引导镜像。

先决条件

- 已使用功能门启用 **TechPreviewNoUpgrade** 功能集。如需更多信息，请参阅*附加资源*部分中的"使用功能门启用功能"。

流程

1. 运行以下命令，编辑名为 **cluster** 的 **MachineConfiguration** 对象，以启用引导镜像的更新：

```
$ oc edit MachineConfiguration cluster
```

a. 可选：为所有机器集配置引导镜像更新功能：

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
# ...
  managedBootImages: 1
    machineManagers:
    - resource: machinesets
      apiGroup: machine.openshift.io
      selection:
        mode: All 2
```

1 激活引导镜像更新功能。

2 指定集群中的所有机器集都会被更新。

b. 可选：为特定机器集配置引导镜像更新功能：

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
# ...
  managedBootImages: 1
    machineManagers:
```

```
    - resource: machinesets
      apiGroup: machine.openshift.io
      selection:
        mode: Partial
          partial:
            machineResourceSelector:
              matchLabels:
                update-boot-image: "true" (2)
```

(1) 激活引导镜像更新功能。

(2) 指定具有此标签的任何机器集都会被更新。

提示

如果机器集中没有适当的标签，请运行以下命令来添加键/值对：

```
$ oc label machineset.machine ci-ln-hmy310k-72292-5f87z-worker-a update-boot-image=true -n openshift-machine-api
```

验证

1. 运行以下命令来获取引导镜像版本：

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

使用引导镜像引用的机器集示例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: ci-ln-77hmkpt-72292-d4pxp
    update-boot-image: "true"
  name: ci-ln-77hmkpt-72292-d4pxp-worker-a
  namespace: openshift-machine-api
spec:
# ...
  template:
# ...
    spec:
# ...
      providerSpec:
# ...
        value:
          disks:
          - autoDelete: true
            boot: true
            image: projects/rhcos-cloud/global/images/rhcos-416-92-202402201450-0-gcp-x86-64 (1)
# ...
```

1 此引导镜像与当前的 OpenShift Container Platform 版本相同。

其他资源

- 使用功能门启用功能

5.2. 禁用更新的引导镜像

要禁用更新的引导镜像功能，请编辑 **MachineConfiguration** 对象以删除 **managedBootImages** 小节。

如果您在使用新的引导镜像版本创建了某些节点后禁用此功能，则任何现有节点会保留其当前引导镜像。关闭这个功能不会将节点或机器集回滚到原始安装的引导镜像。机器集会保留启用该功能时出现的引导镜像版本，并在将来升级到新的 OpenShift Container Platform 版本时不会再次更新。

流程

1. 通过编辑 **MachineConfiguration** 对象禁用更新的引导镜像：

```
$ oc edit MachineConfiguration cluster
```

2. 删除 **managedBootImages** 小节：

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
# ...
  managedBootImages: 1
    machineManagers:
    - resource: machinesets
      apiGroup: machine.openshift.io
      selection:
        mode: All
```

1 删除整个小节以禁用更新的引导镜像。

第 6 章 管理未使用的渲染机器配置

Machine Config Operator (MCO) 不执行任何垃圾回收活动。这意味着所有呈现的机器配置都保留在集群中。每次用户或控制器应用新机器配置时，MCO 会为每个受影响的机器配置池创建新的配置。随着时间的推移，这可能会导致大量渲染的机器配置，这可能会造成使用机器配置混淆。有大量渲染的机器配置也可以会导致 etcd 磁盘空间问题和性能问题。

您可以使用 **oc adm prune renderedmachineconfigs** 命令和-- **confirm** 标志来删除旧的、未使用的机器配置。使用这个命令，您可以删除所有未使用的机器配置，或者只删除特定机器配置池中的机器配置。您还可以删除指定数量的未使用的配置，以便保留一些旧的机器配置，以便检查旧的配置。

您可以使用 **oc adm prune renderedmachineconfigs** 命令，而无需使用-- **confirm** 标记来查看哪些渲染的机器配置会被删除。

使用 **list** 子命令显示集群中的所有呈现的机器配置，或显示特定机器配置池。

注意

oc adm prune renderedmachineconfigs 命令只删除没有使用的机器配置。如果机器配置池正在使用渲染的机器配置，则渲染的机器配置不会被删除。在这种情况下，命令输出指定了渲染的机器配置不会被删除的原因。

6.1. 查看呈现的机器配置

您可以使用 **oc adm prune renderedmachineconfigs** 命令和 **list** 子命令来查看呈现的机器配置列表。

例如，以下流程中的命令会列出 **worker** 机器配置池的所有呈现的机器配置。

流程

- 可选：使用以下命令列出呈现的机器配置：

```
$ oc adm prune renderedmachineconfigs list --in-use=false --pool-name=worker
```

其中：

list

显示集群中呈现的机器配置列表。

--in-use

可选：指定是否只显示指定池中的机器配置或所有机器配置。如果为 **true**，输出会列出机器配置池使用的渲染机器配置。如果为 **false**，输出会列出集群中的所有呈现的机器配置。默认值为 **false**。

--pool-name

可选：指定显示机器配置的机器配置池。

输出示例

```
worker
status: rendered-worker-ae115e2b5e6ae05e0e6e5d62c7d0dd81
spec: rendered-worker-ae115e2b5e6ae05e0e6e5d62c7d0dd81
```

- 运行以下命令，列出您可以自动删除的渲染机器配置。任何在命令输出中被标记为 **as it's currently in use** 的渲染的机器配置都不会被删除。

```
$ oc adm prune renderedmachineconfigs --pool-name=worker
```

命令以空运行模式运行，不会删除机器配置。

其中：

--pool-name

可选：显示指定机器配置池中的机器配置。

输出示例

```
Dry run enabled - no modifications will be made. Add --confirm to remove rendered machine
configs.
DRY RUN: Deleted rendered MachineConfig rendered-worker-
23d7322831a57f02998e7e1600a0865f
DRY RUN: Deleted rendered MachineConfig rendered-worker-
fc94397dc7c43808c7014683c208956e
DRY RUN: Skipping deletion of rendered MachineConfig rendered-worker-
ad5a3cad36303c363cf458ab0524e7c0 as it's currently in use
```

6.2. 删除未使用的渲染机器配置

您可以使用 **oc adm prune renderedmachineconfigs** 命令删除未使用的呈现的机器配置，使用 **--confirm** 命令。如果有没有删除的机器配置，命令输出会显示哪个没有被删除，并列出没有删除的原因。

流程

1. 可选：运行以下命令，列出您可以自动删除的渲染机器配置。任何在命令输出中被标记为 **as it's currently in use** 的渲染的机器配置都不会被删除。

```
$ oc adm prune renderedmachineconfigs --pool-name=worker
```

输出示例

```
Dry run enabled - no modifications will be made. Add --confirm to remove rendered machine
configs.
DRY RUN: Deleted rendered MachineConfig rendered-worker-
23d7322831a57f02998e7e1600a0865f
DRY RUN: Deleted rendered MachineConfig rendered-worker-
fc94397dc7c43808c7014683c208956e
DRY RUN: Skipping deletion of rendered MachineConfig rendered-worker-
ad5a3cad36303c363cf458ab0524e7c0 as it's currently in use
```

其中：

pool-name

可选：指定您要从中删除机器配置的机器配置池。

2. 运行以下命令来删除未使用的渲染机器配置。以下流程中的命令将删除 **worker** 机器配置池中两个最早未使用的机器配置。

```
$ oc adm prune renderedmachineconfigs --pool-name=worker --count=2 --confirm
```

其中：

--count

可选：指定您要删除的未使用呈现的机器配置的最大数量，从最旧的开始。

--confirm

可选：指定您要删除的未使用呈现的机器配置的最大数量，从最旧的开始。

--pool-name

可选：指定您要从中删除机器的机器配置池。如果没有指定，则会评估所有池。

第 7 章 RHCOS 镜像分层

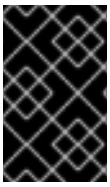
Red Hat Enterprise Linux CoreOS (RHCOS) 镜像分层允许您通过将额外镜像*分层*到基础镜像来轻松地扩展基本 RHCOS 镜像的功能。此分层不会修改基本 RHCOS 镜像。相反，它会创建一个*自定义层次镜像*，其中包含所有 RHCOS 功能，并为集群中的特定节点添加额外的功能。

7.1. 关于 RHCOS 镜像分层

镜像分层（Image layering）允许您在任何集群 worker 节点上自定义底层节点操作系统。这有助于保持一切最新状态，包括节点操作系统和添加的任何自定义，如特殊软件。

您可以使用 Containerfile 创建自定义分层镜像，并使用自定义对象将其应用到节点。在任何时候，您可以通过删除该自定义对象来删除自定义分层镜像。

使用 RHCOS 镜像分层时，您可以在基础镜像中安装 RPM，自定义内容将与 RHCOS 一起引导。Machine Config Operator (MCO) 可以推出这些自定义分层镜像，并像默认 RHCOS 镜像一样监控这些自定义容器。RHCOS 镜像分层为管理 RHCOS 节点的方式提供了更大的灵活性。



重要

不建议将实时内核和扩展 RPM 作为自定义分层内容安装。这是因为这些 RPM 可能会与使用机器配置安装的 RPM 冲突。如果存在冲突，MCO 会在尝试安装机器配置 RPM 时进入 **degraded** 状态。在继续操作前，您需要从机器配置中删除冲突扩展。

将自定义分层镜像应用到集群后，您可以有效地 *获取自定义分层镜像和这些节点的所有权*。虽然红帽仍负责维护和更新标准节点上的基础 RHCOS 镜像，但在使用自定义分层镜像的节点中维护和更新镜像。假定您对自定义分层镜像应用的软件包及软件包可能出现的问题负责。

在节点上部署自定义分层镜像的方法有两种：

On-cluster layering

通过 on-cluster layering，您可以创建一个 **MachineOSConfig** 对象，在其中包含 Containerfile 和其他参数。构建会在集群中执行，生成的自定义分层镜像会自动推送到您的存储库，并应用到您在 **MachineOSConfig** 对象中指定的机器配置池。整个过程会完全在集群中执行。

重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Out-of-cluster layering

通过 out-of-cluster layering，您可以创建一个 Containerfile，引用 OpenShift Container Platform 镜像和您要应用的 RPM，在您自己的环境中构建分层镜像，并将镜像推送到您的存储库。然后，在集群中，为指向新镜像的目标节点池创建一个 **MachineConfig** 对象。Machine Config Operator 覆盖基础 RHCOS 镜像，由关联的机器配置中的 **osImageURL** 值指定，并引导新镜像。

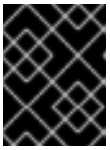
重要

对于这两种方法，使用在集群的其余部分上安装的同一基本 RHCOS 镜像。使用 **oc adm release info --image-for rhel-coreos** 命令获取集群中使用的基础镜像。

7.2. CONTAINERFILES 示例

RHCOS 镜像分层允许您使用以下类型的镜像来创建自定义分层镜像：

- **OpenShift 容器平台 Hotfixes**.您可以使用客户体验和参与(CEE)在 RHCOS 镜像之上获取并应用 Hotfix 软件包。在某些情况下，您可能需要在一个官方的 OpenShift Container Platform 发行版本中包括程序错误修复或功能增强。RHCOS 镜像分层允许您在正式发布前轻松添加 Hotfix，并在底层 RHCOS 镜像包含修复时删除 Hotfix。

  重要

  有些 Hotfixes 需要红帽支持例外，且不在 OpenShift Container Platform 支持覆盖范围或生命周期政策之外。

  根据 Red Hat Hotfix 策略为您提供修补程序。将它应用到基础镜像的顶部，并测试在非生产环境中的新的自定义分层镜像。当您满足自定义分层镜像在生产环境中安全使用时，您可以将其按您自己的计划部署到特定的节点池。因此，您可以轻松地回滚自定义分层镜像，并使用默认 RHCOS 返回。

  应用 Hotfix 的 on-cluster Containerfile 示例

  ```
  # Using a 4.12.0 image
  containerfileArch: noarch
  content: |-
    FROM configs AS final
    #Install hotfix rpm
    RUN rpm-ostree override replace https://example.com/myrepo/haproxy-1.0.16-5.el8.src.rpm
  && \
       rpm-ostree cleanup -m && \
       ostree container commit
  ```

  用于应用 Hotfix 的 out-of-cluster Containerfile 示例

  ```
  # Using a 4.12.0 image
  FROM quay.io/openshift-release-dev/ocp-release@sha256...
  #Install hotfix rpm
  RUN rpm-ostree override replace https://example.com/myrepo/haproxy-1.0.16-5.el8.src.rpm
  && \
     rpm-ostree cleanup -m && \
     ostree container commit
  ```

- **RHEL 软件包**.您可以从红帽客户门户网站下载 Red Hat Enterprise Linux (RHEL) 软件包，如 chrony, firewalld, and iputils。

  使用 libreswan 程序应用 of-cluster Containerfile 示例

  ```
  # Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
  # hadolint ignore=DL3006
  FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

  # Install our config file
  COPY my-host-to-host.conf /etc/ipsec.d/

  # RHEL entitled host is needed here to access RHEL packages
  ```

```
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
   systemctl enable ipsec && \
   ostree container commit
```

因为 libreswan 需要额外的 RHEL 软件包，所以必须在授权的 RHEL 主机上构建镜像。要使 RHEL 权利正常工作，您必须将 **etc-pki-entitlement** secret 复制到 **openshift-machine-api** 命名空间中。

- **第三方软件包**。您可以从第三方机构下载并安装 RPM，如以下类型的软件包：
  - 增强边缘驱动程序和内核增强，以提高性能或添加功能。
  - 用于调查可能和实际分类的客户端工具。
  - 安全代理。
  - 提供整个集群一致的视图的清单代理。
  - SSH 密钥管理软件包。

使用 EPEL 应用第三方软件包的集群 Containerfile 示例

```
FROM configs AS final

#Enable EPEL (more info at https://docs.fedoraproject.org/en-US/epel/ ) and install htop
RUN rpm-ostree install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
&& \
   rpm-ostree install htop && \
   ostree container commit
```

集群外文件示例应用来自 EPEL 的第三方软件包

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
   systemctl enable ipsec && \
   ostree container commit
```

这个 Containerfile 安装 RHEL fish 程序。由于 fish 需要额外的 RHEL 软件包，所以必须在授权的 RHEL 主机上构建镜像。要使 RHEL 权利正常工作，您必须将 **etc-pki-entitlement** secret 复制到 **openshift-machine-api** 命名空间中。

cluster Containerfile 示例，以应用具有 RHEL 依赖项的第三方软件包

```
FROM configs AS final

# RHEL entitled host is needed here to access RHEL packages
```

```
# Install fish as third party package from EPEL
RUN rpm-ostree install
https://dl.fedoraproject.org/pub/epel/9/Everything/x86_64/Packages/f/fish-3.3.1-
3.el9.x86_64.rpm && \
    ostree container commit
```

集群外的 Containerfile 示例，以应用具有 RHEL 依赖项的第三方软件包

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

创建机器配置后，Machine Config Operator (MCO) 执行以下步骤：

1. 为指定池呈现新机器配置。
2. 对池中的节点执行 cordon 和 drain 操作。
3. 将其余机器配置参数写入节点。
4. 将自定义分层镜像应用到节点。
5. 使用新镜像重启节点。

重要

强烈建议您在推出集群前测试生产环境中的镜像。

7.3. 使用 ON-CLUSTER 层应用自定义分层镜像

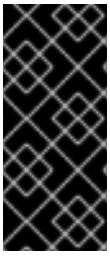
要使用集群构建过程将自定义分层镜像应用到集群，请创建一个 **MachineOSConfig** 自定义资源，其中包含 Containerfile、机器配置池引用、存储库推送和 pull secret 和其他参数，如先决条件中所述。

在创建对象时，Machine Config Operator (MCO) 会创建一个 **MachineOSBuild** 对象和 **machine-os-builder** pod。构建过程还会创建临时对象，如配置映射，这些对象会在构建完成后进行清理。

构建完成后，MCO 将新的自定义分层镜像推送到您的存储库中，以便在部署新节点时使用。您可以在 **MachineOSBuild** 对象和 **machine-os-builder** pod 中看到新自定义分层镜像的摘要镜像拉取 spec。

您不需要与这些新对象或 **machine-os-builder** pod 交互。但是，如果需要，您可以使用所有这些资源进行故障排除。

您要使用自定义分层镜像的每个机器配置池需要单独的 **MachineOSConfig** CR。



重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

先决条件

- 已使用功能门启用 **TechPreviewNoUpgrade** 功能集。如需更多信息，请参阅"使用功能门启用功能"。
- 在 MCO 需要拉取基本操作系统镜像的 **openshift-machine-config-operator** 命名空间中有 pull secret。
- 具有 MCO 需要将新的自定义分层镜像推送到 registry 所需的推送 secret。
- 您有一个 pull secret，节点需要从 registry 中拉取新的自定义分层镜像。这应该与用于将镜像推送到存储库的 secret 不同。
- 您熟悉如何配置 Containerfile。有关如何创建 Containerfile 的说明超出了本文档的范围。
- 可选：为要应用自定义分层镜像的节点有单独的机器配置池。

流程

1. 创建 **machineOSconfig** 对象：
   a. 创建一个类似以下示例的 YAML 文件：

```
apiVersion: machineconfiguration.openshift.io/v1alpha1
kind: MachineOSConfig
metadata:
  name: layered
spec:
  machineConfigPool:
    name: <mcp_name> 1
  buildInputs:
    containerFile: 2
    - containerfileArch: noarch
      content: |-
        FROM configs AS final
        RUN rpm-ostree install cowsay && \
          ostree container commit
    imageBuilder: 3
      imageBuilderType: PodImageBuilder
    baseImagePullSecret: 4
      name: global-pull-secret-copy
    renderedImagePushspec: image-registry.openshift-image-
registry.svc:5000/openshift/os-image:latest 5
    renderedImagePushSecret: 6
      name: builder-dockercfg-7lzwl
```

```
  buildOutputs: 7
   currentImagePullSecret:
     name: builder-dockercfg-7lzwl
```

1. 指定与要部署自定义分层镜像的节点关联的机器配置池的名称。
2. 指定用于配置自定义分层镜像的 Containerfile。
3. 指定要使用的镜像构建器的名称。这必须是 **PodImageBuilder**。
4. 指定 MCO 从 registry 中拉取基础操作系统镜像的 pull secret 名称。
5. 指定要将新构建的自定义分层镜像推送到的镜像 registry。这可以是集群可访问的任何 registry。本例使用内部 OpenShift Container Platform registry。
6. 指定 MCO 将新构建的自定义分层镜像推送到该 registry 的推送 secret 名称。
7. 指定镜像 registry 所需的 secret，节点需要拉取新构建的自定义分层镜像。这应该与用于将镜像推送到存储库的 secret 不同。

b. 创建 **MachineOSConfig** 对象：

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

2. 如果需要，当 **MachineOSBuild** 对象被创建且处于 **READY** 状态时，修改您要使用新自定义分层镜像的节点 spec：

a. 检查 **MachineOSBuild** 对象是否为 **READY**。当 **SUCCEEDED** 值为 **True** 时，构建已完成。

```
$ oc get machineosbuild
```

显示 MachineOSBuild 对象已就绪的输出示例

```
NAME                                                                PREPARED   BUILDING   SUCCEEDED
INTERRUPTED   FAILED
layered-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0-builder   False
False      True       False       False
```

b. 通过为 **MachineOSConfig** 对象中指定的机器配置池添加标签来编辑您要部署自定义分层镜像的节点：

```
$ oc label node <node_name> 'node-role.kubernetes.io/<mcp_name>='
```

其中：

node-role.kubernetes.io/<mcp_name>=

指定用于标识用于部署自定义分层镜像的节点选择器。

当您保存更改时，MCO 会排空、封锁并重启节点。重新引导后，该节点将使用新的自定义分层镜像。

验证

1. 使用以下命令验证新 pod 是否正在运行：

```
$ oc get pods -n <machineosbuilds_namespace>
```

```
NAME                                                READY   STATUS    RESTARTS   AGE
build-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0   2/2     Running   0
2m40s (1)
# ...
machine-os-builder-6fb66cfb99-zcpvq                 1/1     Running   0          2m42s (2)
```

(1) 这是自定义分层镜像构建的构建 pod。

(2) 此 pod 可用于故障排除。

2. 验证 **MachineOSConfig** 对象是否包含对新自定义分层镜像的引用：

```
$ oc describe MachineOSConfig <object_name>
```

```
apiVersion: machineconfiguration.openshift.io/v1alpha1
kind: MachineOSConfig
metadata:
  name: layered
spec:
  buildInputs:
    baseImagePullSecret:
      name: global-pull-secret-copy
    containerFile:
    - containerfileArch: noarch
      content: ""
    imageBuilder:
      imageBuilderType: PodImageBuilder
    renderedImagePushSecret:
      name: builder-dockercfg-ng82t-canonical
    renderedImagePushspec: image-registry.openshift-image-registry.svc:5000/openshift-
machine-config-operator/os-image:latest
  buildOutputs:
    currentImagePullSecret:
      name: global-pull-secret-copy
  machineConfigPool:
    name: layered
status:
  currentImagePullspec: image-registry.openshift-image-registry.svc:5000/openshift-machine-
config-operator/os-
image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11cd
(1)
```

(1) 新自定义分层镜像的摘要镜像拉取 spec。

3. 验证 **MachineOSBuild** 对象是否包含对新自定义分层镜像的引用。

```
$ oc describe machineosbuild <object_name>
```

```
apiVersion: machineconfiguration.openshift.io/v1alpha1
kind: MachineOSBuild
metadata:
  name: layered-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0-builder
spec:
  desiredConfig:
    name: rendered-layered-ad5a3cad36303c363cf458ab0524e7c0
  machineOSConfig:
    name: layered
  renderedImagePushspec: image-registry.openshift-image-registry.svc:5000/openshift-machine-config-operator/os-image:latest
# ...
status:
  conditions:
    - lastTransitionTime: "2024-05-21T20:25:06Z"
      message: Build Ready
      reason: Ready
      status: "True"
      type: Succeeded
  finalImagePullspec: image-registry.openshift-image-registry.svc:5000/openshift-machine-config-operator/os-image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11cd
```
(1)

(1) 新自定义分层镜像的摘要镜像拉取 spec。

4. 验证适当的节点是否使用新的自定义分层镜像：

   a. 以 root 用户身份为 control plane 节点启动一个 debug 会话：

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

   b. 将 **/host** 设置为 debug shell 中的根目录：

   ```
   sh-4.4# chroot /host
   ```

   c. 运行 **rpm-ostree status** 命令，以查看自定义分层镜像正在使用：

   ```
   sh-5.1# rpm-ostree status
   ```

   输出示例

   ```
   # ...
   Deployments:
   * ostree-unverified-registry:quay.io/openshift-release-dev/os-image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11cd
   ```
   (1)
   ```
                    Digest: sha256:bcea2546295b2a55e0a9bf6dd4789433a9867e378661093b6fdee0031ed1e8a4
                 Version: 416.94.202405141654-0 (2024-05-14T16:58:43Z)
   ```

   (1) 新自定义分层镜像的摘要镜像拉取 spec。

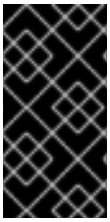
其他资源

- 使用功能门启用功能

7.4. 使用集群外分层应用自定义分层镜像

您可以在特定机器配置池中的节点上轻松配置 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像层。Machine Config Operator (MCO) 使用新的自定义分层镜像重启这些节点，覆盖基本 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像。

要将自定义分层镜像应用到集群，您必须在集群可访问的存储库中具有自定义层次镜像。然后，创建一个指向自定义分层镜像的 **MachineConfig** 对象。对于每个需要配置的集群配置池，都需要一个独立的 **MachineConfig** 对象。

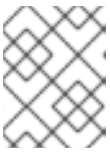


重要

当您配置自定义分层镜像时，OpenShift Container Platform 不再自动更新任何使用自定义分层镜像的节点。根据需要手动进行节点更新是您自己的责任。如果您回滚自定义层，OpenShift Container Platform 将再次自动更新该节点。有关更新使用自定义分层镜像的节点的重要信息，请参阅以下附加资源部分。

先决条件

- 您必须创建一个基于 OpenShift Container Platform 镜像摘要的自定义层次镜像，而不是标签。



注意

您应该使用与集群的其余部分上安装相同的基本 RHCOS 镜像。使用 **oc adm release info --image-for rhel-coreos** 命令获取集群中使用的基础镜像。

例如，以下 Containerfile 从 OpenShift Container Platform 4.16 镜像创建一个自定义层次镜像，并使用 CentOS 9 Stream 中的内核软件包覆盖内核软件包：

自定义层镜像的 Containerfile 示例

```
# Using a 4.16.0 image
FROM quay.io/openshift-release/ocp-release@sha256... 1
#Install hotfix rpm
RUN rpm-ostree override replace http://mirror.stream.centos.org/9-stream/BaseOS/x86_64/os/Packages/kernel-{,core-,modules-,modules-core-,modules-extra-}5.14.0-295.el9.x86_64.rpm && \ 2
    rpm-ostree cleanup -m && \
    ostree container commit
```

1 指定集群的 RHCOS 基础镜像。

2 替换内核软件包。

注意

有关如何创建 Containerfile 的说明超出了本文档的范围。

- 由于构建自定义分层镜像的过程是在集群之外执行，所以您必须在 Podman 或 Buildah 中使用 **--authfile /path/to/pull-secret** 选项。或者，要自动读取这些工具的 pull secret，您可以将其添加到默认文件位置之一：**~/.docker/config.json**, **$XDG_RUNTIME_DIR/containers/auth.json**, **~/.docker/config.json**, 或 **~/.dockercfg**。如需更多信息，请参阅 **containers-auth.json** 手册页。
- 您必须将自定义分层镜像推送到集群可访问的存储库。

流程

1. 创建机器配置文件。
   a. 创建一个类似以下示例的 YAML 文件：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: os-layer-custom
spec:
  osImageURL: quay.io/my-registry/custom-image@sha256... 2
```

   1 指定要应用自定义分层镜像的机器配置池。

   2 指定存储库中自定义分层镜像的路径。

   b. 创建 **MachineConfig** 对象：

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

重要

强烈建议您在推出集群前测试生产环境中的镜像。

验证

您可以通过执行以下任一方式来验证是否应用了自定义层次镜像：

1. 检查 worker 机器配置池已使用新机器配置推出：
   a. 检查是否创建了新机器配置：

```
$ oc get mc
```

输出示例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
00-worker                                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0             95m
```

```
01-master-container-runtime
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
01-master-kubelet                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0          95m
01-worker-container-runtime
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
01-worker-kubelet                          5bdb57489b720096ef912f738b46330a8f577803
3.2.0          95m
99-master-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
99-master-ssh                                              3.2.0          98m
99-worker-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
99-worker-ssh                                              3.2.0          98m
os-layer-custom                                                           10s (1)
rendered-master-15961f1da260f7be141006404d17d39b
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5aff604cb1381a4fe07feaf1595a797e
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5de4837625b1cbc237de6b22bc0bc873
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          4s (2)
```

1. 新机器配置
2. 新的渲染机器配置

b. 检查新机器配置中的 **osImageURL** 值是否指向预期的镜像：

```
$ oc describe mc rendered-master-4e8be63aef68b843b546827b6ebe0913
```

输出示例

```
Name:         rendered-master-4e8be63aef68b843b546827b6ebe0913
Namespace:
Labels:       <none>
Annotations:  machineconfiguration.openshift.io/generated-by-controller-version:
8276d9c1f574481043d3661a1ace1f36cd8c3b62
              machineconfiguration.openshift.io/release-image-version: 4.16.0-ec.3
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfig
...
  Os Image URL: quay.io/my-registry/custom-image@sha256...
```

c. 检查关联的机器配置池是否使用新机器配置更新：

```
$ oc get mcp
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-6faecdfa1b25c114a58cf178fbaa45e2   True      False      False
```

```
3          3          3          0          39m
worker   rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b   False     True
False    3          0          0          0          39m   1
```

1 当 **UPDATING** 字段为 **True** 时，机器配置池会使用新机器配置进行更新。当字段变为 **False** 时，代表 worker 机器配置池已应用到新机器配置。

d. 检查节点以查看是否禁用了在节点上调度。这表示要应用更改：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE
VERSION
ip-10-0-148-79.us-west-1.compute.internal    Ready                      worker                 32m
v1.29.4
ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
35m   v1.29.4
ip-10-0-170-47.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.29.4
ip-10-0-174-77.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.29.4
ip-10-0-211-49.us-west-1.compute.internal    Ready                      control-plane,master
42m   v1.29.4
ip-10-0-218-151.us-west-1.compute.internal   Ready                      worker                 31m
v1.29.4
```

2. 当节点重新处于 **Ready** 状态时，检查该节点是否使用自定义分层镜像：

a. 打开节点的 **oc debug** 会话。例如：

```
$ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
```

b. 将 **/host** 设置为 debug shell 中的根目录：

```
sh-4.4# chroot /host
```

c. 运行 **rpm-ostree status** 命令，以查看自定义分层镜像正在使用：

```
sh-4.4# sudo rpm-ostree status
```

输出示例

```
State: idle
Deployments:
* ostree-unverified-registry:quay.io/my-registry/...
                   Digest: sha256:...
```

其他资源

使用 RHCOS 自定义分层镜像进行更新

7.5. 删除 RHCOS 自定义层次镜像

您可以从特定机器配置池中的节点轻松恢复 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像层。Machine Config Operator (MCO) 使用集群基本 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像重启这些节点，覆盖自定义分层镜像。

要从集群中删除 Red Hat Enterprise Linux CoreOS (RHCOS) 自定义分层镜像，您需要删除应用镜像的机器配置。

流程

1. 删除应用自定义分层镜像的机器配置。

```
$ oc delete mc os-layer-custom
```

删除机器配置后，节点将重新引导。

验证

您可以通过执行以下任一方式来验证自定义层次镜像是否已被删除：

1. 检查 worker 机器配置池是否使用以前的机器配置更新：

```
$ oc get mcp
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-6faecdfa1b25c114a58cf178fbaa45e2   True      False      False
3              3                   3                   0                      39m
worker   rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b   False     True       False
3              0                   0                   0                      39m   (1)
```

(1) 当 **UPDATING** 字段为 **True** 时，机器配置池会使用以前的机器配置进行更新。当字段变为 **False** 时，worker 机器配置池已应用到以前的机器配置。

2. 检查节点以查看是否禁用了在节点上调度。这表示要应用更改：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-148-79.us-west-1.compute.internal    Ready                      worker                 32m
v1.29.4
ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
35m   v1.29.4
ip-10-0-170-47.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.29.4
ip-10-0-174-77.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.29.4
```

```
ip-10-0-211-49.us-west-1.compute.internal    Ready                      control-plane,master   42m
v1.29.4
ip-10-0-218-151.us-west-1.compute.internal   Ready                      worker                 31m
v1.29.4
```

3. 当节点重新处于 **Ready** 状态时，检查该节点是否使用基础镜像：

   a. 打开节点的 **oc debug** 会话。例如：

   ```
   $ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
   ```

   b. 将 **/host** 设置为 debug shell 中的根目录：

   ```
   sh-4.4# chroot /host
   ```

   c. 运行 **rpm-ostree status** 命令，以查看自定义分层镜像正在使用：

   ```
   sh-4.4# sudo rpm-ostree status
   ```

   输出示例

   ```
   State: idle
   Deployments:
   * ostree-unverified-registry:podman pull quay.io/openshift-release-dev/ocp-
   release@sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd
   73
                      Digest:
   sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd73
   ```

7.6. 使用 RHCOS 自定义分层镜像进行更新

当您配置 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像分层时，OpenShift Container Platform 不再自动更新使用自定义分层镜像的节点池。您负责根据需要手动更新节点。

要更新使用自定义分层镜像的节点，请按照以下步骤操作：

1. 集群会自动升级到 x.y.z+1 版本，但使用自定义分层镜像的节点除外。
2. 然后，您可以创建一个引用更新的 OpenShift Container Platform 镜像和之前应用的 RPM 的新 Containerfile。
3. 创建指向更新的自定义分层镜像的新机器配置。

不需要使用自定义分层镜像更新节点。但是，如果该节点早于当前的 OpenShift Container Platform 版本太多，您可能会遇到意外的结果。

第 8 章 机器配置守护进程指标概述

Machine Config Daemon 是 Machine Config Operator 的一部分。它可在集群的每个节点中运行。Machine Config Daemon 管理每个节点上的配置更改和更新。

8.1. 了解机器配置守护进程指标

从 OpenShift Container Platform 4.3 开始，Machine Config Daemon 提供了一组指标。这些指标可以使用 Prometheus Cluster Monitoring 来访问。

下表介绍了这些指标。有些条目包含获取特定日志的命令。使用 **oc adm must-gather** 命令提供了最全面的日志集合。

注意

在 **Name** 和 **Description** 栏中的被标记为 * 的指标数据代表了可能会造成性能问题的严重错误。这些问题可能会阻止更新和升级操作。

表 8.1. MCO 指标

名称	格式	描述	备注
mcd_host_os_and_version	**[]string{"os", "version"}**	显示运行 MCD 的操作系统，如 RHCOS 或 RHEL。如果是 RHCOS，则会提供版本信息。	
mcd_drain_err*		在排空失败时出现的错误。*	虽然排空可能需要多次尝试方可成功，但最终失败的排空会操作会阻止更新进行。**drain_time** 指标显示排空操作所用的时间，这可帮助进行故障排除。 如需进一步调查，请运行以下命令查看日志： **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_pivot_err*	**[]string{"err", "node", "pivot_target"}**	pivot 过程中遇到的日志错误。*	pivot 错误可能会导致 OS 升级无法进行。 要进一步调查，请运行以下命令查看 **machine-config-daemon** 容器中的日志： **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

名称	格式	描述	备注
mcd_state	**[]string{"state", "reason"}**	指定节点的 Machine Config Daemon 状态。可能的状态是 "Done"、"Working" 和 "Degraded"。如果是 "Degraded"，则会包括原因。	如需进一步调查，请运行以下命令查看日志： **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_kubelet_state*		日志 kubelet 健康失败。*	这应该为空，故障计数为 0。如果失败数超过 2，则代表超过了阈值。这表示 kubelet 健康可能存在问题。 要进行进一步调查，请运行这个命令访问该节点并查看其所有日志： **$ oc debug node/<node> — chroot /host journalctl -u kubelet**
mcd_reboot_err*	**[]string{"message", "err", "node"}**	重启失败以及相应错误的日志。*	这应该为空，代表重启成功。 如需进一步调查，请运行以下命令查看日志： **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_update_state	**[]string{"config", "err"}**	记录配置更新的成功或失败以及相应的错误。	预期的值为 **rendered-master/rendered-worker-XXXX**。如果更新失败，则会出现错误。 如需进一步调查，请运行以下命令查看日志： **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

其他资源

- 监控概述
- 收集集群数据