OpenShift Container Platform 4.16

边缘计算

在网络边缘配置和部署 OpenShift Container Platform 集群

Last Updated: 2024-07-03

OpenShift Container Platform 4.16 边缘计算

在网络边缘配置和部署 OpenShift Container Platform 集群

法律通告

摘要

本文档论述了如何使用 GitOps ZTP 配置和部署 OpenShift Container Platform 集群，以便在网络边缘置备和管理站点。

目录

第 1 章 网络边缘的挑战

在地理位置管理多个站点时，边缘计算带来了复杂的挑战。使用 GitOps Zero Touch Provisioning (ZTP) 在网络边缘置备和管理站点。

1.1. 克服网络边缘的挑战

今天，服务提供商希望在网络边缘部署其基础架构。这带来了显著的挑战：

- 您怎样处理并行部署多个边缘站点的部署？
- 当您需要在断开连接的环境中部署站点时，会出现什么情况？
- 如何管理集群的生命周期？

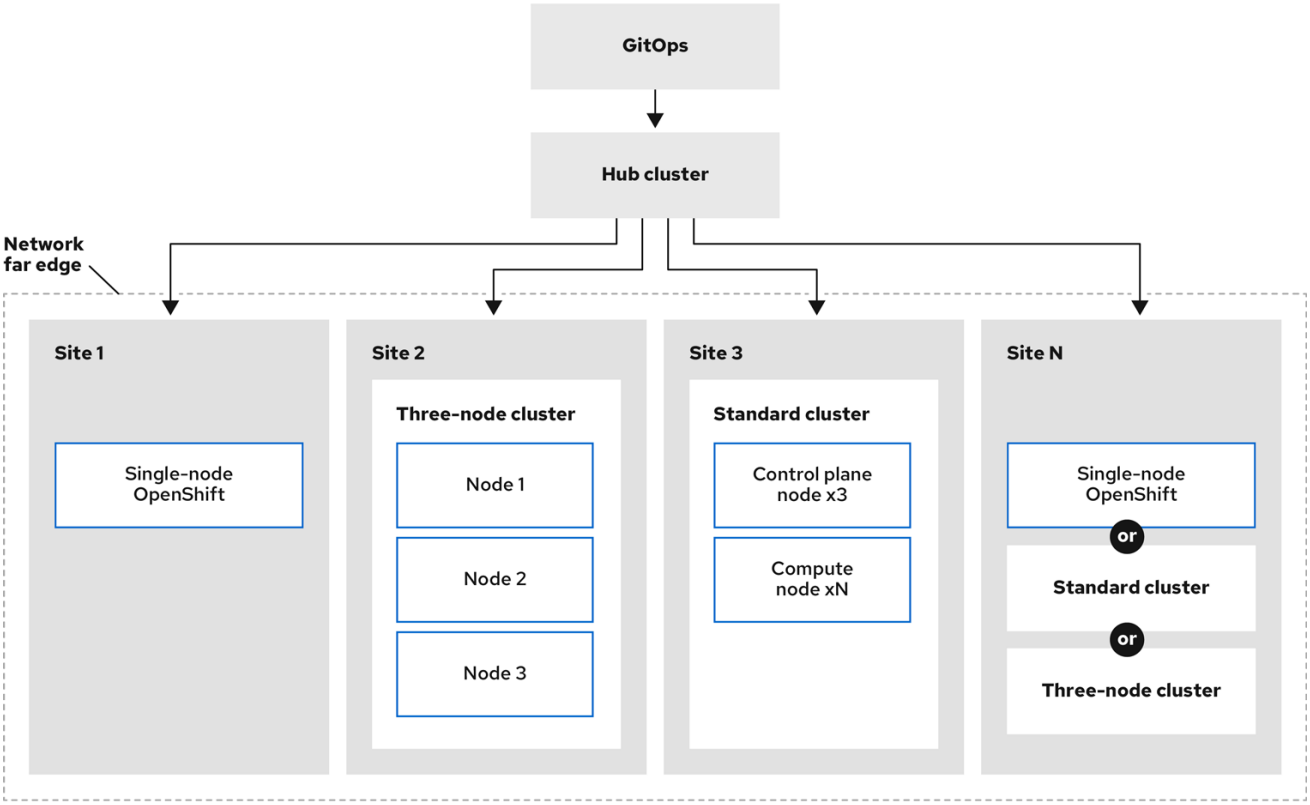
GitOps Zero Touch Provisioning (ZTP) 和 *GitOps* 通过允许您为裸机设备使用声明站点定义和配置大规模置备远程边缘站点。模板或覆盖配置安装 CNF 工作负载所需的 OpenShift Container Platform 功能。安装和升级的完整生命周期通过 GitOps ZTP 管道处理。

GitOps ZTP 使用 GitOps 进行基础架构部署。使用 GitOps，您可以使用声明 YAML 文件和其他存储在 Git 存储库中的其他定义模式。Red Hat Advanced Cluster Management (RHACM)使用 Git 存储库来驱动基础架构部署。

GitOps 提供可追溯性、基于角色的访问控制 (RBAC)，以及每个站点的所需状态的单一数据源。Git 方法可通过 webhook 解决可扩展性问题，以及事件驱动的操作。

您可以通过创建 GitOps ZTP 管道提供给边缘节点的声明站点定义和配置自定义资源 (CR) 来启动 GitOps ZTP 工作流。

下图显示了 GitOps ZTP 如何在最边缘框架内工作。



1.2. 使用 GITOPS ZTP 在网络边缘置备集群

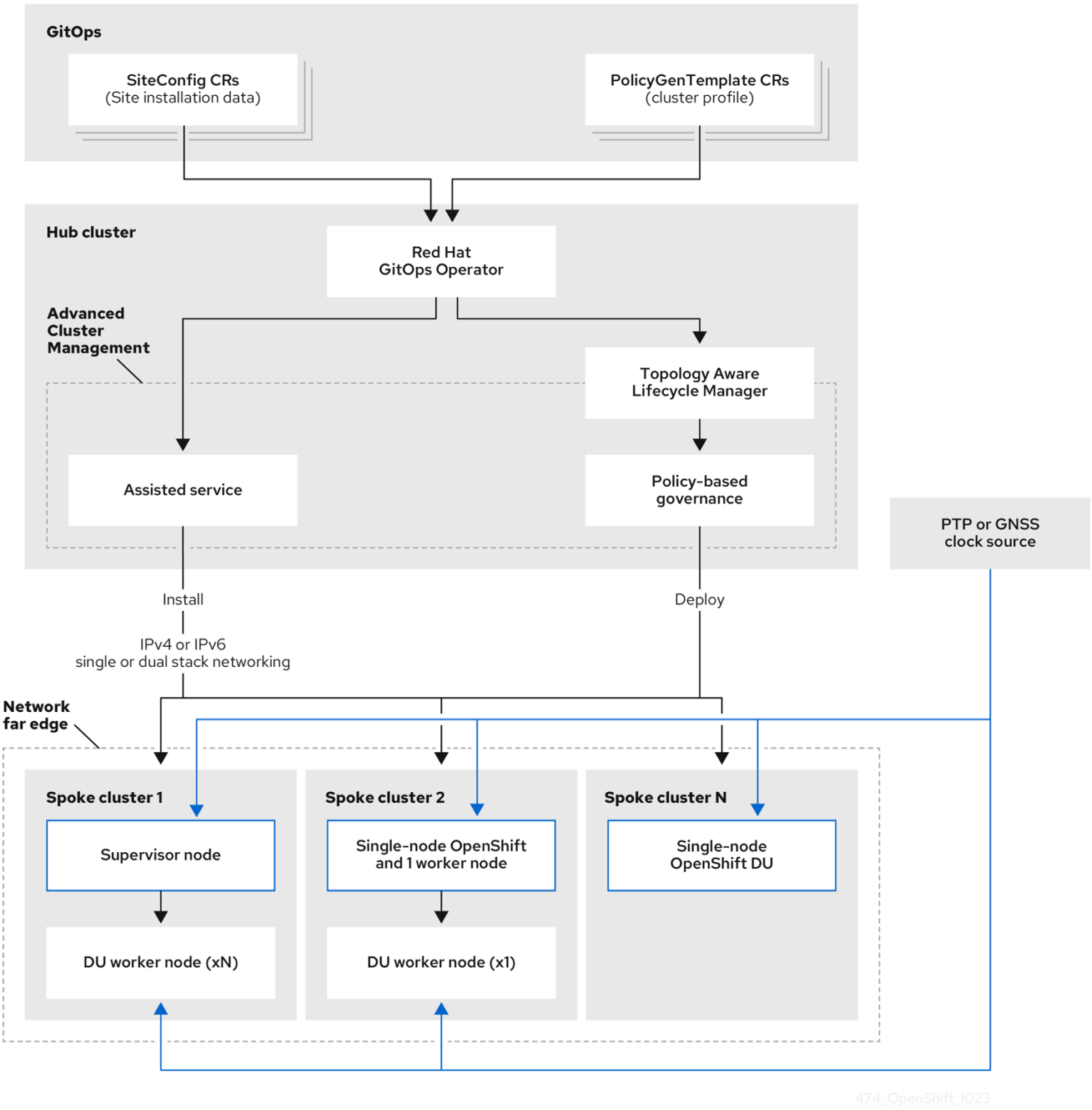
Red Hat Advanced Cluster Management (RHACM)在 hub 和 spoke 架构中管理集群，其中单个 hub 集群管理多个 spoke 集群。运行 RHACM 的 hub 集群使用 GitOps Zero Touch Provisioning (ZTP) 和安装 RHACM 时部署的辅助服务来置备和部署受管集群。

协助的服务处理在单一节点集群、三节点集群或裸机上运行的标准集群上 OpenShift Container Platform 置备。

使用 GitOps ZTP 的高级别概述来置备和维护使用 OpenShift Container Platform 的裸机主机，如下所示：

- 运行 RHACM 的 hub 集群管理一个 OpenShift 镜像 registry，用于镜像 OpenShift Container Platform 发行镜像。RHACM 使用 OpenShift 镜像 registry 来置备受管集群。
- 您以 YAML 格式清单文件管理裸机主机，并在 Git 存储库中版本。
- 您可以使主机准备好作为受管集群置备，并使用 RHACM 和辅助服务在站点上安装裸机主机。

安装和部署集群分为两个阶段，涉及初始安装阶段，以及后续配置和部署阶段。下图演示了这个工作流：



1.3. 使用 SITECONFIG 资源和 RHACM 安装受管集群

GitOps Zero Touch Provisioning (ZTP) 使用 Git 存储库中的 **SiteConfig** 自定义资源 (CR) 来管理安装 OpenShift Container Platform 集群的进程。**SiteConfig** CR 包含安装所需的特定于集群的参数。它有在安装过程中应用所选配置 CR 的选项，包括用户定义的额外清单。

GitOps ZTP 插件处理 **SiteConfig** CR，以便在 hub 集群上生成 CR 集合。这会在 Red Hat Advanced Cluster Management (RHACM) 中触发辅助服务，以便在裸机主机上安装 OpenShift Container Platform。您可以在 hub 集群上的这些 CR 中找到安装状态和错误消息。

您可以手动置备单个集群，或使用 GitOps ZTP 批量置备单个集群：

置备单个集群

为集群创建单一 **SiteConfig** CR 及相关的安装和配置 CR，并在 hub 集群中应用它们以开始集群置备。这是在大规模部署前测试 CR 的好方法。

置备多个集群

通过在 Git 仓库中定义 **SiteConfig** 和相关 CR，以最多 400 的批处理中安装受管集群。ArgoCD 使用 **SiteConfig** CR 来部署站点。RHACM 策略生成器创建清单，并将其应用到 hub 集群。这将启动集群置备过程。

1.4. 使用策略和 POLICYGENTEMPLATE 资源配置受管集群

GitOps Zero Touch Provisioning (ZTP) 使用 Red Hat Advanced Cluster Management (RHACM) 使用基于策略的监管方法应用配置配置。

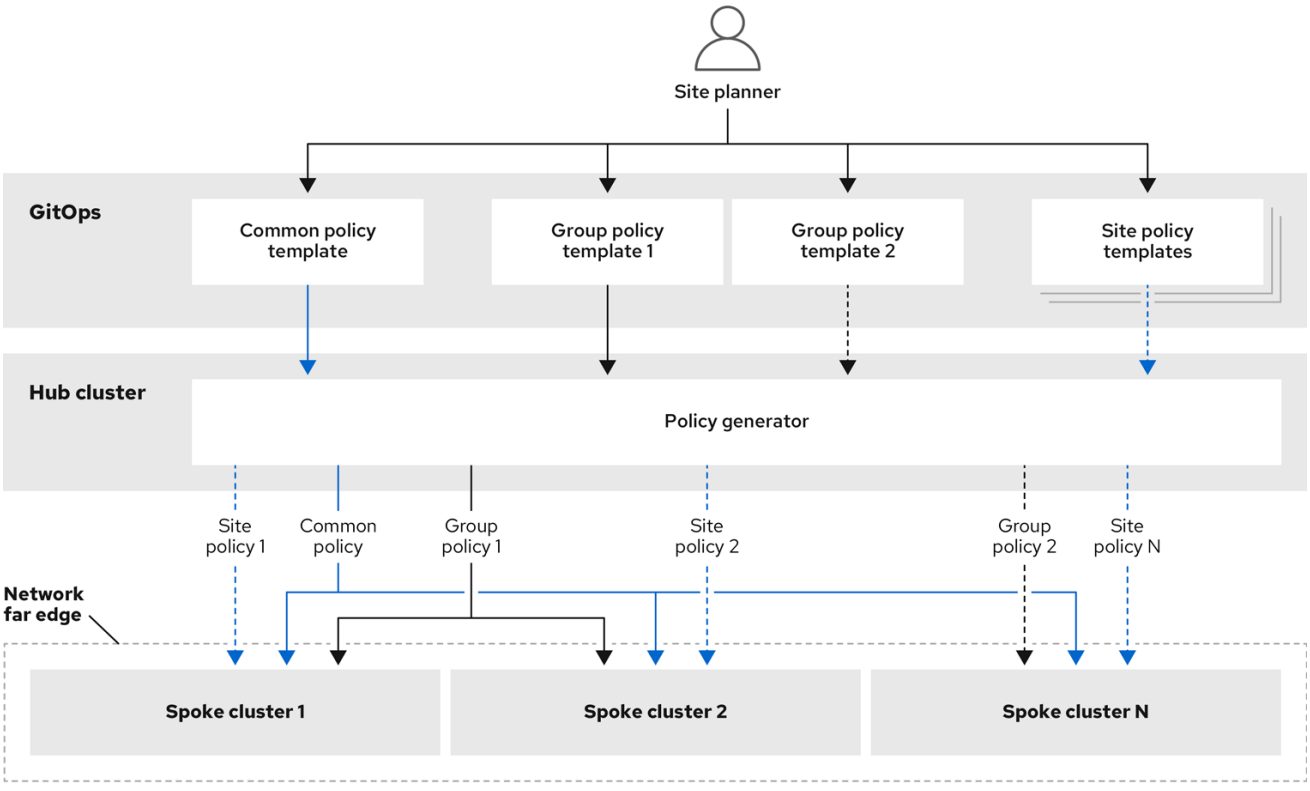
策略生成器或 **PolicyGen** 是 GitOps 操作器的一个插件，它允许从简洁的模板创建 RHACM 策略。该工具可将多个 CR 合并为一个策略，您可以生成多个策略应用到团队中集群的不同子集的策略。

注意

为了扩展并降低跨集群管理配置的复杂性，请尽可能使用配置 CR。

- 在可能的情况下，使用机范围的通用策略应用配置 CR。
- 下一个首选项是创建集群的逻辑分组，以在组策略下尽可能管理剩余的配置。
- 当配置对单个站点是唯一的时，请使用 hub 集群上的 RHACM 模板将特定于站点的数据注入通用或组策略。或者，为站点应用单个站点策略。

下图显示了在集群部署配置阶段策略生成器如何与 GitOps 和 RHACM 交互。



对于大型集群群，在配置这些集群时通常具有高级别的一致性。

以下推荐的策略结构组合了配置 CR，以满足几个目标：

- 描述一次通用配置，并应用到所有系统。

- 最小化维护和管理策略的数量。
- 支持集群变体的通用配置的灵活性。

表 1.1. 推荐的 PolicyGenTemplate 策略类别

策略类别	描述
Common	一个存在于 common 类别中的策略被应用到该团队中的所有集群。使用通用 **PolicyGenerator** CR 在所有集群类型中应用通用安装设置。
组	组类别中存在的策略应用到一组集群。使用 group **PolicyGenerator** CR 管理单节点、三节点和标准集群安装的特定方面。集群组也可以遵循区域、硬件变体等。
Sites	站点类别中存在的策略应用到特定的集群站点。任何集群都可以维护自己的特定策略。

重要

使用 **PolicyGenTemplate** CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 **PolicyGenerator** CR 提供了等效和改进的功能。

有关 **PolicyGenerator** 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁
- 准备 GitOps ZTP Git 存储库

第 2 章 为 GITOPS ZTP 准备 HUB 集群

要在断开连接的环境中使用 RHACM，请创建一个镜像 registry，镜像 OpenShift Container Platform 发行镜像和包含所需 Operator 镜像的 Operator Lifecycle Manager (OLM) 目录。OLM 在集群中管理、安装和升级 Operator 及其依赖项。您还可以使用断开连接的镜像主机来提供用于置备裸机主机的 RHCOS ISO 和 RootFS 磁盘镜像。

2.1. TELCO RAN DU 4.16 验证软件组件

Red Hatlecommunications RAN DU 4.16 解决方案已使用以下红帽软件产品用于 OpenShift Container Platform 受管集群和 hub 集群。

表 2.1. Telco RAN DU 受管集群验证的软件组件

组件	软件版本
受管集群版本	4.16
Cluster Logging Operator	5.9
Local Storage Operator	4.16
PTP Operator	4.16
SRIOV Operator	4.16
Node Tuning Operator	4.16
Logging Operator	4.16
SRIOV-FEC Operator	2.9

表 2.2. hub 集群验证的软件组件

组件	软件版本
hub 集群版本	4.16
GitOps ZTP 插件	4.16
Red Hat Advanced Cluster Management (RHACM)	2.11
Red Hat OpenShift GitOps	1.12
Topology Aware Lifecycle Manager (TALM)	4.16

2.2. GITOPS ZTP 推荐的 HUB 集群规格和受管集群限制

使用 GitOps Zero Touch Provisioning (ZTP)，您可以在地理分散的区域和网络中管理数千个集群。在实验室环境中，Red Hat Performance and Scale 实验室成功创建和管理了 3500 个虚拟单节点 OpenShift 集群，该集群减少了 DU 配置集。

在实际情况下，您可以管理的集群数量扩展限制将因影响 hub 集群的不同因素而有所不同。例如：

hub 集群资源

可用的 hub 集群主机资源(CPU、内存、存储)是决定 hub 集群可以管理的集群的一个重要因素。分配给 hub 集群的更多资源，它可以容纳更多受管集群。

hub 集群存储

hub 集群主机存储 IOPS 评级以及 hub 集群主机是否使用 NVMe 存储可能会影响 hub 集群性能及其可以管理的集群数量。

网络带宽和延迟

hub 集群和受管集群之间的延迟或高延迟网络连接可能会影响 hub 集群管理多个集群的方式。

受管集群大小和复杂性

受管集群的大小和复杂性也会影响 hub 集群的容量。具有更多节点、命名空间和资源的大型受管集群需要额外的处理和管理资源。同样，具有复杂配置（如 RAN DU 配置集或不同工作负载）的集群可以从 hub 集群中需要更多资源。

受管策略数量

由 hub 集群管理的策略数量通过绑定到这些策略的受管集群数量进行扩展，是一个重要的因素，决定了可以管理多少个集群。

监控和管理工作负载

RHACM 持续监控和管理受管集群。在 hub 集群中运行的监控和管理工作负载的数量和复杂性可能会影响其容量。密集型监控或频繁协调操作可能需要其他资源，从而可能会限制可管理的集群数量。

RHACM 版本和配置

RHACM 的不同版本可能具有不同的性能特性和资源要求。另外，RHACM 的配置设置（如并发协调的数量或健康检查的频率）可能会影响 hub 集群的受管集群容量。

使用以下代表配置和网络规格来开发您自己的 Hub 集群和网络规格。

重要

以下准则仅基于内部实验室基准测试，并不代表完整的实际主机规格。

表 2.3. 代表三节点集群机器规格

要求	描述
OpenShift Container Platform	版本 4.13
RHACM	版本 2.7
Topology Aware Lifecycle Manager (TALM)	版本 4.13
服务器硬件	3 x Dell PowerEdge R650 机架服务器

要求	描述
NVMe 硬盘	• 50 GB 磁盘用于 **/var/lib/etcd** • 2.9 TB 磁盘用于 **/var/lib/containers**
SSD 硬盘	• 1 个 SSD 被分成 15 200GB 的精简置备逻辑卷作为 **PV** CR • 1 个 SSD 作为额外的大型 **PV** 资源
应用的 DU 配置集策略数量	5

重要

以下网络规格是典型的实际 RAN 网络的代表，并在测试过程中应用于扩展实验室环境。

表 2.4. 模拟实验室环境网络规格

规格	描述
往返时间(RTT)延迟	50 ms
数据包丢失	0.02% 数据包丢失
网络带宽限制	20 Mbps

其他资源

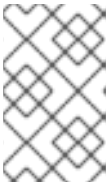
- 使用 RHACM 创建和管理单节点 OpenShift 集群

2.3. 在断开连接的环境中安装 GITOPS ZTP

在断开连接的环境中，使用 Red Hat Advanced Cluster Management (RHACM)、Red Hat OpenShift GitOps 和 Topology Aware Lifecycle Manager (TALM) 来管理多个受管集群的部署。

先决条件

- 已安装 OpenShift Container Platform CLI (**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 您已配置了断开连接的镜像 registry 以在集群中使用。



注意

您创建的断开连接的镜像 registry 必须包含 TALM backup 和 pre-cache 镜像的版本，该镜像与 hub 集群中运行的 TALM 版本匹配。spoke 集群必须能够在断开连接的镜像 registry 中解析这些镜像。

流程

- 在 hub 集群上安装 RHACM。请参阅 在断开连接的环境中安装 RHACM。
- 在 hub 集群中安装 GitOps 和 TALM。

其他资源

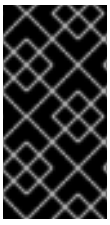
- 安装 OpenShift GitOps
- 安装 TALM
- 对 Operator 目录进行镜像(mirror)

2.4. 在断开连接的镜像主机中添加 RHCOS ISO 和 ROOTFS 镜像

在使用 Red Hat Advanced Cluster Management (RHACM) 在断开连接的环境中安装集群前，您必须首先托管 Red Hat Enterprise Linux CoreOS (RHCOS) 镜像供其使用。使用断开连接的镜像来托管 RHCOS 镜像。

先决条件

- 部署和配置 HTTP 服务器以托管网络上的 RHCOS 镜像资源。您必须能够从计算机以及您创建的机器访问 HTTP 服务器。



重要

RHCOS 镜像可能不会随着 OpenShift Container Platform 的每个发行版本而改变。您必须下载最高版本的镜像，其版本号应小于或等于您安装的版本。如果可用，请使用与 OpenShift Container Platform 版本匹配的镜像版本。您需要 ISO 和 RootFS 镜像在主机上安装 RHCOS。此安装类型不支持 RHCOS QCOW2 镜像。

流程

1. 登录到镜像主机。
2. 从 mirror.openshift.com 获取 RHCOS ISO 和 RootFS 镜像，例如：
   a. 将所需的镜像名称和 OpenShift Container Platform 版本导出为环境变量：

```
$ export ISO_IMAGE_NAME=<iso_image_name> 1
$ export ROOTFS_IMAGE_NAME=<rootfs_image_name> 1
$ export OCP_VERSION=<ocp_version> 1
```

1 ISO 镜像名称，如 **rhcos-4.16.1-x86_64-live.x86_64.iso**

1 rootfs 镜像名称，如 **rhcos-4.16.1-x86_64-live-rootfs.x86_64.img**

1 OpenShift Container Platform 版本，如 **4.16.1**

b. 下载所需的镜像：

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.16/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.16/${OCP_VERSION}/${ROOTFS_IMAGE_NAME} -O
/var/www/html/${ROOTFS_IMAGE_NAME}
```

验证步骤

- 验证下载的镜像是否成功，并在断开连接的镜像主机上提供，例如：

```
$ wget http://$(hostname)/${ISO_IMAGE_NAME}
```

输出示例

```
Saving to: rhcos-4.16.1-x86_64-live.x86_64.iso
rhcos-4.16.1-x86_64-live.x86_64.iso-  11%[====>    ]  10.01M  4.71MB/s
```

其他资源

- 创建镜像 registry
- 为断开连接的安装 mirror 镜像

2.5. 启用辅助服务

Red Hat Advanced Cluster Management (RHACM)使用辅助服务来部署 OpenShift Container Platform 集群。当您在 Red Hat Advanced Cluster Management (RHACM)上启用 MultiClusterHub Operator 时，辅助服务会自动部署。之后，您需要配置 **Provisioning** 资源以监视所有命名空间，并更新 **AgentServiceConfig** 自定义资源(CR)以引用托管在镜像 registry HTTP 服务器上的 ISO 和 RootFS 镜像。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已以具有 **cluster-admin** 权限的用户身份登录到 hub 集群。
- 启用了 MultiClusterHub 的 RHACM。

流程

1. 启用 **Provisioning** 资源，以监视所有命名空间并为断开连接的环境配置镜像。如需更多信息，请参阅启用中央基础架构管理服务。

2. 运行以下命令来更新 **AgentServiceConfig** CR：

```
$ oc edit AgentServiceConfig
```

3. 在 CR 的 **items.spec.osImages** 字段中添加以下条目：

```
- cpuArchitecture: x86_64
  openshiftVersion: "4.16"
  rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
  url: https://<mirror-registry>/<path>/rhcos-live.x86_64.iso
```

其中：

<host>

是目标镜像 registry HTTP 服务器的完全限定域名 (FQDN)。

<path>

是目标镜像 registry 上镜像的路径。

保存并退出编辑器以应用更改。

2.6. 将 HUB 集群配置为使用断开连接的镜像 REGISTRY

您可以将 hub 集群配置为使用断开连接的镜像 registry 作为断开连接的环境。

先决条件

- 已安装 Red Hat Advanced Cluster Management (RHACM) 2.10 的断开连接的 hub 集群安装。
- 您已在 HTTP 服务器中托管 **rootfs** 和 **iso** 镜像。有关 *Mirroring the OpenShift Container Platform image repository* 的信息，请参阅 *附加资源部分*。

警告

如果为 HTTP 服务器启用 TLS，您必须确认 root 证书由客户端信任的颁发机构签名，并验证 OpenShift Container Platform hub 和受管集群和 HTTP 服务器之间的可信证书链。使用配置了不受信任的证书的服务器可防止将镜像下载到创建镜像中。不支持使用不受信任的 HTTPS 服务器。

流程

1. 创建包含镜像 registry 配置的 **ConfigMap**：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: assisted-installer-mirror-config
  namespace: multicluster-engine (1)
  labels:
```

```
    app: assisted-service
data:
  ca-bundle.crt: | (2)
    -----BEGIN CERTIFICATE-----
    <certificate_contents>
    -----END CERTIFICATE-----

  registries.conf: | (3)
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

    [[registry]]
      prefix = ""
      location = "quay.io/example-repository" (4)
      mirror-by-digest-only = true

      [[registry.mirror]]
      location = "mirror1.registry.corp.com:5000/example-repository" (5)
```

1. **ConfigMap** 命名空间必须设置为 **multicluster-engine**。
2. 创建镜像 registry 时使用的镜像 registry 证书。
3. 镜像 registry 的配置文件。镜像 registry 配置将镜像信息添加到发现镜像的 **/etc/containers/registries.conf** 文件中。当信息传递给安装程序时，镜像信息会存储在 **install-config.yaml** 文件的 **imageContentSources** 部分中。在 hub 集群上运行的 Assisted Service pod 从配置的镜像 registry 获取容器镜像。
4. 镜像 registry 的 URL。在配置镜像 registry 时，您必须运行 **oc adm release mirror** 命令使用 **imageContentSources** 部分中的 URL。如需更多信息，请参阅 *Mirroring the OpenShift Container Platform image repository* 部分。
5. **registries.conf** 文件中定义的 registry 必须由存储库范围，而不是由 registry 范围。在本例中，**quay.io/example-repository** 和 **mirror1.registry.corp.com:5000/example-repository** 存储库都限定了 **example-repository** 存储库。

这会更新 **AgentServiceConfig** 自定义资源中的 **mirrorRegistryRef**，如下所示：

输出示例

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  namespace: multicluster-engine (1)
spec:
  databaseStorage:
    volumeName: <db_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <db_storage_size>
  filesystemStorage:
    volumeName: <fs_pv_name>
    accessModes:
```

```
    - ReadWriteOnce
    resources:
      requests:
        storage: <fs_storage_size>
  mirrorRegistryRef:
    name: assisted-installer-mirror-config (2)
  osImages:
    - openshiftVersion: <ocp_version>
      url: <iso_url> (3)
```

1. 将 **AgentServiceConfig** 命名空间设置为 **multicluster-engine**，以匹配 **ConfigMap** 命名空间
2. 将 **mirrorRegistryRef.name** 设置为与相关 **ConfigMap** CR 中指定的定义匹配
3. 设置在 **httpd** 服务器上托管的 ISO 的 URL

重要

集群安装过程中需要一个有效的 NTP 服务器。确保有合适的 NTP 服务器可用，并可通过断开连接的网络从安装的系统访问。

其他资源

- 镜像 OpenShift Container Platform 镜像存储库

2.7. 将 HUB 集群配置为使用未经身份验证的 REGISTRY

您可以将 hub 集群配置为使用未经身份验证的 registry。未经身份验证的 registry 不需要进行身份验证才能访问和下载镜像。

先决条件

- 您已在 hub 集群上安装并配置了 hub 集群，并安装了 Red Hat Advanced Cluster Management (RHACM)。
- 已安装 OpenShift Container Platform CLI (oc)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 已配置了一个未经身份验证的 registry 以用于 hub 集群。

流程

1. 运行以下命令来更新 **AgentServiceConfig** 自定义资源 (CR)：

```
$ oc edit AgentServiceConfig agent
```

2. 在 CR 中添加 **unauthenticatedRegistries** 字段：

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
```

```
  name: agent
spec:
  unauthenticatedRegistries:
  - example.registry.com
  - example.registry2.com
  ...
```

未经身份验证的 registry 在 **AgentServiceConfig** 资源的 **spec.unauthenticatedRegistries** 下列出。任何此列表中的 registry 都不需要在用于 spoke 集群安装的 pull secret 中有一个条目。**assisted-service** 通过确保包含用于安装的每个镜像 registry 的身份验证信息来验证 pull secret。

注意

镜像 registry 会自动添加到 ignore 列表中，不需要在 **spec.unauthenticatedRegistries** 下添加。在 **ConfigMap** 中指定 **PUBLIC_CONTAINER_REGISTRIES** 环境变量会用指定的值覆盖默认值。**PUBLIC_CONTAINER_REGISTRIES** 默认值是 quay.io 和 registry.svc.ci.openshift.org。

验证

运行以下命令，验证您可以从 hub 集群访问新添加的 registry：

1. 在 hub 集群中打开一个 debug shell 提示符：

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

2. 运行以下命令测试对未经身份验证的 registry 的访问：

```
sh-4.4# podman login -u kubeadmin -p $(oc whoami -t) <unauthenticated_registry>
```

其中：

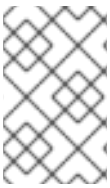
<unauthenticated_registry>
新的 registry，如 **unauthenticated-image-registry.openshift-image-registry.svc:5000**。

输出示例

```
Login Succeeded!
```

2.8. 使用 ARGOCD 配置 HUB 集群

您可以使用一组 ArgoCD 应用程序来配置 hub 集群，每个站点使用 GitOps 零接触置备 (ZTP) 生成所需的安装和策略自定义资源(CR)。



注意

Red Hat Advanced Cluster Management (RHACM) 使用 **SiteConfig** CR 为 ArgoCD 生成第 1 天受管集群安装 CR。每个 ArgoCD 应用程序都可以管理最多 300 个 **SiteConfig** CR。

先决条件

- 已安装 Red Hat Advanced Cluster Management (RHACM) 和 Red Hat OpenShift GitOps 的 OpenShift Container Platform hub 集群。
- 您已从 GitOps ZTP 插件容器中提取了引用部署，如 "Preparing the GitOps ZTP site configuration repository" 部分所述。提取引用部署会创建以下流程中引用的 **out/argocd/deployment** 目录。

流程

1. 准备 ArgoCD 管道配置：
   a. 创建 Git 存储库，其目录结构类似于 example 目录。如需更多信息，请参阅"准备 GitOps ZTP 站点配置存储库"。
   b. 使用 ArgoCD UI 配置对存储库的访问。在 **Settings** 下配置以下内容：
      - **Repositories** - 添加连接信息。URL 必须以 **.git** 结尾，例如 **https://repo.example.com/repo.git** 和凭证。
      - **证书** - 如果需要，为存储库添加公共证书。
   c. 根据您的 Git 仓库修改两个 ArgoCD 应用程序，**out/argocd/deployment/clusters-app.yaml** 和 **out/argocd/deployment/policies-app.yaml** ：
      - 更新 URL 以指向 Git 存储库。URL 以 **.git** 结尾，例如 **https://repo.example.com/repo.git**。
      - **targetRevision** 表示要监控的 Git 存储库分支。
      - **path** 指定到 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** CR 的路径。
2. 要安装 GitOps ZTP 插件，使用相关多集群引擎 (MCE) 订阅镜像对 hub 集群中的 ArgoCD 实例进行补丁。自定义您解压到环境的 **out/argocd/deployment/** 目录中的补丁文件。
   a. 选择与您的 RHACM 版本匹配的 **multicluster-operators-subscription** 镜像。

表 2.5. multicluster-operators-subscription 镜像版本

OpenShift Container Platform 版本	RHACM 版本	MCE 版本	MCE RHEL 版本	MCE 镜像
4.14, 4.15, 4.16	2.8, 2.9	2.8, 2.9	RHEL 8	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.8** **registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.9**
4.14, 4.15, 4.16	2.10	2.10	RHEL 9	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10**

重要

multicluster-operators-subscription 镜像的版本应与 RHACM 版本匹配。从 MCE 2.10 发行版本开始，RHEL 9 是 **multicluster-operators-subscription** 镜像的基础镜像。

b. 在 **out/argocd/deployment/argocd-openshift-gitops-patch.json** 文件中添加以下配置：

```
{
  "args": [
    "-c",
    "mkdir -p /.config/kustomize/plugin/policy.open-cluster-
management.io/v1/policygenerator && cp /policy-generator/PolicyGenerator-not-fips-
compliant /.config/kustomize/plugin/policy.open-cluster-
management.io/v1/policygenerator/PolicyGenerator" (1)
  ],
  "command": [
    "/bin/bash"
  ],
  "image": "registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10", (2)
(3)
  "name": "policy-generator-install",
  "imagePullPolicy": "Always",
  "volumeMounts": [
    {
      "mountPath": "/.config",
      "name": "kustomize"
    }
  ]
}
```

(1) 可选：对于 RHEL 9 镜像，在 ArgoCD 版本的 **/policy-generator/PolicyGenerator-not-fips-compliant** 文件夹中复制所需的通用可执行文件。

(2) 将 **multicluster-operators-subscription** 镜像与 RHACM 版本匹配。

(3) 在断开连接的环境中，将 **multicluster-operators-subscription** 镜像的 URL 替换为与您的环境中使用的断开连接的 registry。

c. 对 ArgoCD 实例进行补丁。运行以下命令:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

3. 在 RHACM 2.7 及更高版本中，多集群引擎默认启用 **cluster-proxy-addon** 功能。应用以下补丁来禁用 **cluster-proxy-addon** 功能，并删除负责此附加组件的相关 hub 集群和受管 pod。运行以下命令:

```
$ oc patch multiclusterengines.multicluster.openshift.io multiclusterengine --type=merge --
patch-file out/argocd/deployment/disable-cluster-proxy-addon.json
```

4. 运行以下命令，将管道配置应用到 hub 集群：

```
$ oc apply -k out/argocd/deployment
```

2.9. 准备 GITOPS ZTP 站点配置存储库

在使用 GitOps Zero Touch Provisioning (ZTP) 管道前，您需要准备 Git 存储库来托管站点配置数据。

先决条件

- 已配置了 hub 集群 GitOps 应用程序来生成所需的安装和策略自定义资源 (CR)。
- 已使用 GitOps ZTP 部署受管集群。

流程

1. 使用 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** CR 的单独路径创建一个目录结构。

注意

将 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** CR 保留在单独的目录中。**SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** 目录必须包含一个 **kustomization.yaml** 文件，该文件明确包含该目录中的文件。

2. 使用以下命令，从 **ztp-site-generate** 容器镜像导出 **argocd** 目录：

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16
```

```
$ mkdir -p ./out
```

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./out
```

3. 检查 **out** 目录是否包含以下子目录：
   - **out/extra-manifest** 包含 **SiteConfig** 用来生成额外清单 **configMap** 的源 CR 文件。
   - **out/source-crs** 包含 **PolicyGenerator** 用来生成 Red Hat Advanced Cluster Management (RHACM)策略的源 CR 文件。
   - **out/argocd/deployment** 包含补丁和 YAML 文件，可在 hub 集群中应用，以便在此过程的下一步中使用。
   - **out/argocd/example** 包含代表推荐的配置的 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** 文件的示例。
4. 将 **out/source-crs** 文件夹和内容复制到 **PolicyGenerator** 或 **PolicyGentemplate** 目录。
5. out/extra-manifests 目录包含 RAN DU 集群的参考清单。将 **out/extra-manifests** 目录复制到 **SiteConfig** 文件夹。此目录应包含来自 **ztp-site-generate** 容器的 CR。不要在此处添加用户提供的 CR。如果要使用用户提供的 CR，您必须为其内容创建另一个目录。例如：

```
example/
  ├── acmpolicygenerator
```

```
│   ├── kustomization.yaml
│   └── source-crs/
├── policygentemplates (1)
│   ├── kustomization.yaml
│   └── source-crs/
└── siteconfig
    ├── extra-manifests
    └── kustomization.yaml
```

1. 使用 **PolicyGenTemplate** CR 管理和部署策略来管理集群将在以后的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 **PolicyGenerator** CR 提供了等效和改进的功能。

6. 提交目录结构和 **kustomization.yaml** 文件并推送到 Git 存储库。初始推送到 Git 的推送应包含 **kustomization.yaml** 文件。

您可以使用 **out/argocd/example** 下的目录结构作为 Git 存储库结构和内容的参考。该结构包括用于单节点、三节点和标准集群的 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** 引用 CR。删除您对未使用集群类型的引用。

对于所有集群类型，您必须：

- 将 **source-crs** 子目录添加到 **acmpolicygenerator** 或 **policygentemplates** 目录中。
- 将 **extra-manifests** 目录添加到 **siteconfig** 目录中。

以下示例描述了单节点集群网络的一组 CR：

```
example/
├── acmpolicygenerator
│   ├── acm-common-ranGen.yaml
│   ├── acm-example-sno-site.yaml
│   ├── acm-group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   ├── kustomization.yaml
│   ├── source-crs/
│   └── ns.yaml
└── siteconfig
    ├── example-sno.yaml
    ├── extra-manifests/ (1)
    ├── custom-manifests/ (2)
    ├── KlusterletAddonConfigOverride.yaml
    └── kustomization.yaml
```

1. 包含 **ztp-container** 中的引用清单。
2. 包含自定义清单。

重要

使用 **PolicyGenTemplate** CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 **PolicyGenerator** CR 提供了等效和改进的功能。

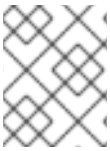
有关 **PolicyGenerator** 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

2.10. 为独立版本准备 GITOPS ZTP 站点配置存储库

您可以使用 GitOps ZTP 管理运行不同版本的 OpenShift Container Platform 的受管集群的源自定义资源 (CR)。这意味着，在 hub 集群中运行的 OpenShift Container Platform 版本可以独立于受管集群上运行的版本。



注意

以下流程假设您使用 **PolicyGenerator** 资源而不是 **PolicyGentemplate** 资源进行集群策略管理。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 使用 **SiteConfig** 和 **PolicyGenerator** CR 的单独路径创建一个目录结构。
2. 在 **PolicyGenerator** 目录中，为您要提供的每个 OpenShift Container Platform 版本创建一个目录。对于每个版本，创建以下资源：
   - 明确包含该目录中文件的 **kustomization.yaml** 文件
   - **source-crs** 目录，其中包含 **ztp-site-generate** 容器中的引用 CR 配置文件
     如果要使用用户提供的 CR，必须为它们创建一个单独的目录。
3. 在 **/siteconfig** 目录中，为您要提供的每个 OpenShift Container Platform 版本创建一个子目录。对于每个版本，至少创建一个目录，用于从容器中复制引用 CR。对目录命名或引用目录数量没有限制。如果要使用自定义清单，必须为它们创建单独的目录。
   以下示例描述了对不同 OpenShift Container Platform 版本使用用户提供的清单和 CR 的结构：

```
├── acmpolicygenerator
│   ├── kustomization.yaml (1)
│   ├── version_4.13 (2)
│   │   ├── common-ranGen.yaml
│   │   ├── group-du-sno-ranGen.yaml
│   │   ├── group-du-sno-validator-ranGen.yaml
│   │   ├── helix56-v413.yaml
```

```
│   │   ├── kustomization.yaml (3)
│   │   ├── ns.yaml
│   │   └── source-crs/ (4)
│   │       └── reference-crs/ (5)
│   │       └── custom-crs/ (6)
│   └── version_4.14 (7)
│       ├── common-ranGen.yaml
│       ├── group-du-sno-ranGen.yaml
│       ├── group-du-sno-validator-ranGen.yaml
│       ├── helix56-v414.yaml
│       ├── kustomization.yaml (8)
│       ├── ns.yaml
│       └── source-crs/ (9)
│           └── reference-crs/ (10)
│           └── custom-crs/ (11)
└── siteconfig
    ├── kustomization.yaml
    ├── version_4.13
    │   ├── helix56-v413.yaml
    │   ├── kustomization.yaml
    │   ├── extra-manifest/ (12)
    │   └── custom-manifest/ (13)
    └── version_4.14
        ├── helix57-v414.yaml
        ├── kustomization.yaml
        ├── extra-manifest/ (14)
        └── custom-manifest/ (15)
```

(1) 创建顶级 **kustomization** YAML 文件。

(2) (7) 在自定义 **/acmpolicygenerator** 目录中创建特定于版本的目录。

(3) (8) 为每个版本创建一个 **kustomization.yaml** 文件。

(4) (9) 为每个版本创建一个 **source-crs** 目录，使其包含 **ztp-site-generate** 容器中的引用 CR。

(5) (10) 为从 ZTP 容器中提取的策略 CR 创建 **reference-crs** 目录。

(6) (11) 可选：为用户提供的 CR 创建一个 **custom-crs** 目录。

(12) (14) 在自定义 **/siteconfig** 目录中创建一个目录，使其包含 **ztp-site-generate** 容器的额外清单。

(13) (15) 创建用于存放用户提供的清单的文件夹。

注意

在上例中，自定义 **/siteconfig** 目录中的每个版本子目录包含两个进一步的子目录，一个子目录包含从容器中复制的引用清单，另一个用于您提供的自定义清单。分配给这些目录的名称是示例。如果使用用户提供的 CR，则 **SiteConfig** CR 中的 **extraManifests.searchPaths** 下列出的最后一个目录必须是包含用户提供的 CR 的目录。

4. 编辑 **SiteConfig** CR，使其包含您创建的任何目录的搜索路径。**extraManifests.searchPaths** 下列出的第一个目录必须是包含引用清单的目录。考虑列出目录的顺序。如果目录包含相同名称的文件，则最终目录中的文件将具有优先权。

   SiteConfig CR 示例

```
extraManifests:
  searchPaths:
  - extra-manifest/ 1
  - custom-manifest/ 2
```

   1. 包含引用清单的目录必须首先列在 **extraManifests.searchPaths** 下。
   2. 如果您使用用户提供的 CR，则 **SiteConfig** CR 中的 **extraManifests.searchPaths** 下列出的最后一个目录必须是包含这些用户提供的 CR 的目录。

5. 编辑顶级 **kustomization.yaml** 文件，以控制哪些 OpenShift Container Platform 版本处于活跃状态。以下是顶层的 **kustomization.yaml** 文件示例：

```
resources:
- version_4.13 1
#- version_4.14 2
```

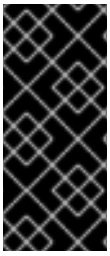
   1. 激活版本 4.13。
   2. 使用注释取消激活版本。

第 3 章 更新 GITOPS ZTP

您可以独立于 hub 集群、Red Hat Advanced Cluster Management (RHACM) 和受管 OpenShift Container Platform 集群更新 Gitops Zero Touch Provisioning (ZTP) 基础架构。

注意

当新版本可用时，您可以更新 Red Hat OpenShift GitOps Operator。更新 GitOps ZTP 插件时，请查看参考配置中的更新文件，并确保更改满足您的要求。



重要

使用 **PolicyGenTemplate** CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 **PolicyGenerator** CR 提供了等效和改进的功能。

有关 **PolicyGenerator** 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

3.1. GITOPS ZTP 更新过程概述

您可以为运行较早版本的 GitOps ZTP 集群更新 GitOps Zero Touch Provisioning (ZTP)。更新过程可避免对受管集群的影响。

注意

对策略设置的任何更改（包括添加推荐内容）都会生成要应用到受管集群并协调的更新策略。

在高级别上，更新 GitOps ZTP 基础架构的策略如下：

1. 使用 **ztp-done** 标签标记所有现有集群。
2. 停止 ArgoCD 应用程序。
3. 安装新的 GitOps ZTP 工具。
4. 更新 Git 存储库中的所需内容和可选更改。
5. 更新并重启应用程序配置。

3.2. 准备升级

使用以下步骤为 GitOps Zero Touch Provisioning (ZTP)升级准备您的站点。

流程

1. 获取具有用于配置 Red Hat OpenShift GitOps 的自定义资源 (CR) 的 GitOps ZTP 容器的最新版本，以用于 GitOps ZTP。

2. 使用以下命令提取 **argocd/deployment** 目录：

```
$ mkdir -p ./update
```

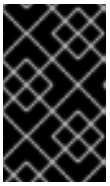
```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./update
```

/**update** 目录包含以下子目录：

- **update/extra-manifest**: 包含 **SiteConfig** CR 用来生成额外清单 **configMap** 的源 CR 文件。
- **update/source-crs** ：包含 **PolicyGenerator** 或 **PolicyGentemplate** CR 用于生成 Red Hat Advanced Cluster Management (RHACM)策略的源 CR 文件。
- **update/argocd/deployment**: 包含要在 hub 集群上应用的补丁和 YAML 文件，以便在此过程的下一步中使用。
- **update/argocd/example**: 包含代表推荐的配置的 **SiteConfig** 和 **PolicyGenerator** 或 **PolicyGentemplate** 文件示例。

3. 更新 **cluster-app.yaml** 和 **policies-app.yaml** 文件，以反映应用程序的名称以及 Git 仓库的 URL、分支和路径。
如果升级包含导致过时的策略的更改，则应该在执行升级前删除过时的策略。

4. 在 /**update** 文件夹和 Git 仓库（您管理团队站点 CR）中的配置和部署源 CR 之间的更改进行 diff 操作。应用所需的更改并将其推送到您的站点存储库。



重要

当您将 GitOps ZTP 更新至最新版本时，您必须将 **update/argocd/deployment** 目录中的更改应用到您的站点存储库。不要使用旧版本的 **argocd/deployment/** 文件。

3.3. 标记现有集群

为确保现有集群由工具更新保持不变，请使用 **ztp-done** 标签标记所有现有的受管集群。

注意

此流程仅在更新没有使用 Topology Aware Lifecycle Manager (TALM) 置备的集群时应用。使用 TALM 置备的集群会使用 **ztp-done** 自动标记。

流程

1. 找到列出使用 GitOps Zero Touch Provisioning (ZTP) 部署的受管集群的标签选择器，如 **local-cluster!=true** ：

```
$ oc get managedcluster -l 'local-cluster!=true'
```

2. 确保生成的列表中包含使用 GitOps ZTP 部署的所有受管集群，然后使用该选择器添加 **ztp-done** 标签：

```
$ oc label managedcluster -l 'local-cluster!=true' ztp-done=
```

3.4. 停止现有的 GITOPS ZTP 应用程序

删除现有的应用程序可确保在有新版本工具可用前，不会推出对 Git 存储库中现有内容的任何更改。

使用 **deployment** 目录中的应用文件。如果您为应用程序使用自定义名称，则首先更新这些文件中的名称。

流程

1. 在 **clusters** 应用程序上执行非级联删除以保留所有生成的资源：

```
$ oc delete -f update/argocd/deployment/clusters-app.yaml
```

2. 在 **policies** 应用程序上执行级联删除以删除所有之前的策略：

```
$ oc patch -f policies-app.yaml -p '{"metadata": {"finalizers": ["resources-finalizer.argocd.argoproj.io"]}}' --type merge
```

```
$ oc delete -f update/argocd/deployment/policies-app.yaml
```

3.5. 对 GIT 存储库进行所需的更改

当将 **ztp-site-generate** 容器从较早版本的 GitOps Zero Touch Provisioning (ZTP) 升级到 v4.10 或更高版本时，Git 仓库的内容需要额外的要求。存储库中的现有内容必须更新，以反映这些更改。

注意

以下流程假设您使用 **PolicyGenerator** 资源而不是 **PolicyGentemplate** 资源进行集群策略管理。

- 对 **PolicyGenerator** 文件进行必要的更改：
  所有 **PolicyGenerator** 文件都必须在前缀为 **ztp** 的命名空间中创建。这样可确保 GitOps ZTP 应用程序能够管理由 GitOps ZTP 生成的策略 CR，而无需与 Red Hat Advanced Cluster Management (RHACM) 在内部管理策略冲突。

- 将 **kustomization.yaml** 文件添加到存储库中：
  所有 **SiteConfig** 和 **PolicyGenerator** CR 必须包含在其各自目录树下的 **kustomization.yaml** 文件中。例如：

```
├── acmpolicygenerator
│   ├── site1-ns.yaml
│   ├── site1.yaml
│   ├── site2-ns.yaml
│   ├── site2.yaml
│   ├── common-ns.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen-ns.yaml
│   ├── group-du-sno-ranGen.yaml
│   └── kustomization.yaml
└── siteconfig
```

```
├── site1.yaml
├── site2.yaml
└── kustomization.yaml
```

注意

generator 部分中列出的文件只能包含 **SiteConfig** 或 **{policy-gen-cr}** CR。如果现有 YAML 文件包含其他 CR，如 **Namespace**，则这些其他 CR 必须拉取到单独的文件中，并在 **resources** 部分列出。

PolicyGenerator kustomization 文件必须包含 **generator** 部分中的所有 **PolicyGenerator** YAML 文件，以及 **resources** 部分中的 **Namespace** CR。例如：

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- acm-common-ranGen.yaml
- acm-group-du-sno-ranGen.yaml
- site1.yaml
- site2.yaml

resources:
- common-ns.yaml
- acm-group-du-sno-ranGen-ns.yaml
- site1-ns.yaml
- site2-ns.yaml
```

SiteConfig kustomization 文件必须包括 **generator** 部分中的所有 **SiteConfig** YAML 文件，以及资源中的任何其他 CR：

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- site1.yaml
- site2.yaml
```

- 删除 **pre-sync.yaml** 和 **post-sync.yaml** 文件。
  在 OpenShift Container Platform 4.10 及更新的版本中，不再需要 **pre-sync.yaml** 和 **post-sync.yaml** 文件。**update/deployment/kustomization.yaml** CR 管理 hub 集群上的策略部署。

注意

SiteConfig 和 **{policy-gen-cr}** 树都有一组 **pre-sync.yaml** 和 **post-sync.yaml** 文件。

- 检查并纳入推荐的更改
  每个发行版本可能会包括对应用到已部署集群的配置进行额外的推荐更改。通常，这些更改由 OpenShift 平台、额外功能或改进对平台的调整带来较低的 CPU 使用。

  查看适用于网络中集群类型的参考 **SiteConfig** 和 **PolicyGenerator** CR。这些示例可在从 GitOps ZTP 容器中提取的 **argocd/example** 目录中找到。

3.6. 安装新的 GITOPS ZTP 应用程序

使用提取的 **argocd/deployment** 目录，并在确保应用程序指向 Git 存储库后应用部署目录的所有内容。应用目录的内容可确保正确配置应用程序的所有必要资源。

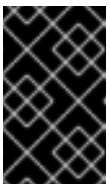
流程

1. 要安装 GitOps ZTP 插件，使用相关多集群引擎 (MCE) 订阅镜像对 hub 集群中的 ArgoCD 实例进行补丁。自定义您解压到环境的 **out/argocd/deployment/** 目录中的补丁文件。

   a. 选择与您的 RHACM 版本匹配的 **multicluster-operators-subscription** 镜像。

   表 3.1. multicluster-operators-subscription 镜像版本

OpenShift Container Platform 版本	RHACM 版本	MCE 版本	MCE RHEL 版本	MCE 镜像
4.14, 4.15, 4.16	2.8, 2.9	2.8, 2.9	RHEL 8	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.8** **registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.9**
4.14, 4.15, 4.16	2.10	2.10	RHEL 9	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10**



重要

multicluster-operators-subscription 镜像的版本应与 RHACM 版本匹配。从 MCE 2.10 发行版本开始，RHEL 9 是 **multicluster-operators-subscription** 镜像的基础镜像。

   b. 在 **out/argocd/deployment/argocd-openshift-gitops-patch.json** 文件中添加以下配置：

```
{
  "args": [
    "-c",
    "mkdir -p /.config/kustomize/plugin/policy.open-cluster-management.io/v1/policygenerator && cp /policy-generator/PolicyGenerator-not-fips-compliant /.config/kustomize/plugin/policy.open-cluster-management.io/v1/policygenerator/PolicyGenerator" (1)
  ],
  "command": [
    "/bin/bash"
  ],
  "image": "registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10", (2)
(3)
```

```
    "name": "policy-generator-install",
    "imagePullPolicy": "Always",
    "volumeMounts": [
      {
        "mountPath": "/.config",
        "name": "kustomize"
      }
    ]
  }
```

1. 可选：对于 RHEL 9 镜像，在 ArgoCD 版本的 **/policy-generator/PolicyGenerator-not-fips-compliant** 文件夹中复制所需的通用可执行文件。
2. 将 **multicluster-operators-subscription** 镜像与 RHACM 版本匹配。
3. 在断开连接的环境中，将 **multicluster-operators-subscription** 镜像的 URL 替换为与您的环境中使用的断开连接的 registry。

c. 对 ArgoCD 实例进行补丁。运行以下命令:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

2. 在 RHACM 2.7 及更高版本中，多集群引擎默认启用 **cluster-proxy-addon** 功能。应用以下补丁来禁用 **cluster-proxy-addon** 功能，并删除负责此附加组件的相关 hub 集群和受管 pod。运行以下命令:

```
$ oc patch multiclusterengines.multicluster.openshift.io multiclusterengine --type=merge --patch-file out/argocd/deployment/disable-cluster-proxy-addon.json
```

3. 运行以下命令，将管道配置应用到 hub 集群：

```
$ oc apply -k out/argocd/deployment
```

3.7. 推出 GITOPS ZTP 配置更改

如果因为实现推荐的更改而在升级过程中包括任何配置更改，升级过程会在 hub 集群上生成 **Non-Compliant** 状态的一组策略 CR。使用 GitOps Zero Touch Provisioning (ZTP) 版本 4.10 及更新的版本 **ztp-site-generate** 容器，这些策略被设置为 **inform** 模式，且不会由用户在没有额外步骤的情况下推送到受管集群。这样可保证在进行更改时可以管理对集群的破坏性更改，例如在维护窗口期间以及同时更新多少个集群。

要推出更改，请创建一个或多个 **ClusterGroupUpgrade** CR，如 TALM 文档所述。CR 必须包含您要推送到受管集群的 **Non-Compliant** 策略列表，以及应包含在更新中的集群的列表或选择器。

其他资源

- 有关 Topology Aware Lifecycle Manager(TALM)，请参阅关于 Topology Aware Lifecycle Manager 配置。
- 有关创建 **ClusterGroupUpgrade** CR 的信息，请参阅关于为 GitOps ZTP 自动创建的 ClusterGroupUpgrade CR。

第 4 章 使用 RHACM 和 SITECONFIG 资源安装受管集群

您可以使用辅助服务以及启用了 core-reduction 技术的 GitOps 插件策略生成器，以扩展 Red Hat Advanced Cluster Management (RHACM) 来大规模置备 OpenShift Container Platform 集群。GitOps Zero Touch Provisioning (ZTP) 管道执行集群安装。GitOps ZTP 可以在断开连接的环境中使用。

重要

使用 **PolicyGenTemplate** CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 **PolicyGenerator** CR 提供了等效和改进的功能。

有关 **PolicyGenerator** 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

4.1. GITOPS ZTP 和 TOPOLOGY AWARE LIFECYCLE MANAGER

GitOps Zero Touch Provisioning (ZTP) 从 Git 中存储的清单生成安装和配置 CR。这些工件应用到一个中央化的 hub 集群，其中 Red Hat Advanced Cluster Management (RHACM)、辅助服务和 Topology Aware Lifecycle Manager (TALM) 使用 CR 来安装和配置受管集群。GitOps ZTP 管道的配置阶段使用 TALM 将配置 CR 的应用程序编排到集群。GitOps ZTP 和 TALM 之间有几个关键集成点。

通知策略

默认情况下，GitOps ZTP 创建所有带有 **inform** 的补救操作的策略。这些策略会导致 RHACM 报告与策略相关的集群合规性状态，但不会应用所需的配置。在 GitOps ZTP 过程中，在 OpenShift 安装后，TALM 步骤通过创建的 **inform** 策略，并在目标受管集群中强制实施它们。这会将配置应用到受管集群。在集群生命周期的 GitOps ZTP 阶段之外，这允许您在不立即将这些更改部署到受影响的受管集群的情况下更改策略。您可以使用 TALM 控制时间和修复的集群集合。

自动创建 ClusterGroupUpgrade CR

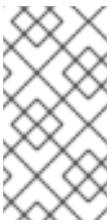
要自动执行新部署的集群的初始配置，TALM 会监控 hub 集群上所有 **ManagedCluster** CR 的状态。任何未应用 **ztp-done** 标签的 **ManagedCluster** CR，包括新创建的 **ManagedCluster** CR，会导致 TALM 自动创建一个具有以下特征的 **ClusterGroupUpgrade** CR：

- 在 **ztp-install** 命名空间中创建并启用 **ClusterGroupUpgrade** CR。
- **ClusterGroupUpgrade** CR 的名称与 **ManagedCluster** CR 的名称相同。
- 集群选择器仅包括与该 **ManagedCluster** CR 关联的集群。
- 受管策略集合包含 RHACM 在 **ClusterGroupUpgrade** 创建时绑定到集群的所有策略。
- 禁用预缓存。
- 超时设置为 4 小时（240 分钟）。

启用的 **ClusterGroupUpgrade** 的自动创建可确保初始零接触集群部署继续进行，而无需用户干预。另外，为任何没有 **ztp-done** 标签的 **ManagedCluster** 自动创建一个 **ClusterGroupUpgrade** CR，可以通过删除集群的 **ClusterGroupUpgrade** CR 来重启失败的 GitOps ZTP 安装。

Waves

从 **PolicyGenerator** 或 **PolicyGentemplate** CR 生成的每个策略都包含一个 **ztp-deploy-wave** 注解。此注解基于来自每个 CR 的同一注解，该注解包含在该策略中。wave 注解用于对自动生成的 **ClusterGroupUpgrade** CR 中的策略进行排序。wave 注解没有用于自动生成的 **ClusterGroupUpgrade** CR。



注意

同一策略中的所有 CR 都必须具有 **ztp-deploy-wave** 注解的设置。可以在 **PolicyGenerator** 或 **PolicyGentemplate** 中覆盖每个 CR 的此注解的默认值。源 CR 中的 wave 注解用来决定和设置策略 wave 注解。此注解已从每个构建 CR 中删除，该 CR 在运行时包含在生成的策略中。

TALM 按照 wave 注解指定的顺序应用配置策略。在移至下一个策略前，TALM 会等待每个策略兼容。确保每个 CR 的 wave 注解考虑要应用到集群的任何 CR 的先决条件。例如，必须在 Operator 配置之前或同时安装 Operator。同样，Operator 的 **CatalogSource** 必须在 Operator Subscription 之前或同时安装 wave。每个 CR 的默认 wave 值会考虑这些先决条件。

多个 CR 和策略可以共享相同的 wave 编号。拥有较少的策略可缩短部署速度并降低 CPU 用量。将多个 CR 分组到相对较少的waves 是最佳实践方案。

要检查每个源 CR 中的默认 wave 值，请针对从 **ztp-site-generate** 容器镜像中提取的 **out/source-crs** 目录运行以下命令：

```
$ grep -r "ztp-deploy-wave" out/source-crs
```

阶段标签

ClusterGroupUpgrade CR 会被自动创建，并包含在 GitOps ZTP 进程开始和结尾使用标签为 **ManagedCluster** CR 的说明。
当 GitOps ZTP 配置安装后启动时，**ManagedCluster** 会应用 **ztp-running** 标签。当所有策略都修复至集群并完全合规时，这些指令会导致 TALM 删除 **ztp-running** 标签并应用 **ztp-done** 标签。

对于使用 **informDuValidator** 策略的部署，当集群完全准备好部署应用程序时，会应用 **ztp-done** 标签。这包括 GitOps ZTP 应用的配置 CR 的所有协调并产生影响。**ztp-done** 标签会影响 TALM 创建自动 **ClusterGroupUpgrade** CR。不要在集群初始 GitOps ZTP 安装后操作此标签。

链接的 CR

自动创建的 **ClusterGroupUpgrade** CR 将所有者引用设置为派生于 **ManagedCluster** 的 ManagedCluster。此引用可确保删除 **ManagedCluster** CR 会导致 **ClusterGroupUpgrade** 的实例以及任何支持的资源被删除。

4.2. 使用 GITOPS ZTP 部署受管集群概述

Red Hat Advanced Cluster Management (RHACM) 使用 GitOps Zero Touch Provisioning (ZTP) 来部署单节点 OpenShift Container Platform 集群、三节点集群和标准集群。您可以在 Git 存储库中将站点配置数据作为 OpenShift Container Platform 自定义资源 (CR) 进行管理。GitOps ZTP 使用声明性 GitOps 方法进行开发一次，部署任意位置模型来部署受管集群。

集群部署包括：

- 在空白服务器上安装主机操作系统 (RHCOS)

- 部署 OpenShift Container Platform
- 创建集群策略和站点订阅
- 为服务器操作系统进行必要的网络配置
- 部署配置集 Operator 并执行任何所需的软件相关配置，如性能配置集、PTP 和 SR-IOV

受管站点安装过程概述
在 hub 集群中应用受管站点自定义资源 (CR) 后，会自动执行以下操作：

1. 在目标主机上生成并启动发现镜像 ISO 文件。
2. 当 ISO 文件成功在目标主机上引导时，它会将主机硬件信息报告给 RHACM。
3. 在所有主机被发现后，会安装 OpenShift Container Platform。
4. 当 OpenShift Container Platform 完成安装后，hub 在目标集群上安装 **klusterlet** 服务。
5. 请求的附加组件服务安装在目标集群中。

当在 hub 集群上创建受管集群的 **Agent** CR 时，发现镜像 ISO 过程已完成。

重要

目标裸机主机必须满足 vDU 应用程序工作负载的推荐单节点 OpenShift 集群配置 中列出的网络、固件和硬件要求。

4.3. 创建受管裸机主机 SECRET

将受管裸机主机所需的 **Secret** 自定义资源 (CR) 添加到 hub 集群。您需要 GitOps Zero Touch Provisioning (ZTP) 管道的 secret 来访问 Baseboard Management Controller (BMC) 和支持的安装程序服务的 secret，以便从 registry 中拉取集群安装镜像。

注意

secret 按名称从 **SiteConfig** CR 引用。命名空间必须与 **SiteConfig** 命名空间匹配。

流程

1. 创建一个 YAML secret 文件，其中包含主机 Baseboard Management Controller (BMC) 和安装 OpenShift 和所有附加组件集群 Operator 所需的凭证：

   a. 将以下 YAML 保存为文件 **example-sno-secret.yaml** ：

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
```

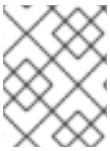
```
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno 3
data:
  .dockerconfigjson: <pull_secret> 4
type: kubernetes.io/dockerconfigjson
```

1. 必须与相关 **SiteConfig** CR 中配置的命名空间匹配
2. **password** 和 **username** 的 base64 编码值
3. 必须与相关 **SiteConfig** CR 中配置的命名空间匹配
4. Base64 编码的 pull secret

2. 将到 **example-sno-secret.yaml** 的相对路径添加用于安装集群的 **kustomization.yaml** 文件中。

4.4. 使用 GITOPS ZTP 为安装配置 DISCOVERY ISO 内核参数

GitOps Zero Touch Provisioning (ZTP) 工作流使用 Discovery ISO 作为托管裸机主机的 OpenShift Container Platform 安装过程的一部分。您可以编辑 **InfraEnv** 资源来为 Discovery ISO 指定内核参数。这对具有特定环境要求的集群安装非常有用。例如，为发现 ISO 配置 **rd.net.timeout.carrier** 内核参数以促进集群的静态网络，或者在在安装过程中下载根文件系统前接收 DHCP 地址。



注意

在 OpenShift Container Platform 4.16 中，您只能添加内核参数。您不能替换或删除内核参数。

先决条件

- 已安装 OpenShift CLI（oc）。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 创建 **InfraEnv** CR，并编辑 **spec.kernelArguments** 规格以配置内核参数。
   a. 将以下 YAML 保存到 **InfraEnv-example.yaml** 文件中：

注意

本例中的 **InfraEnv** CR 使用模板语法，如 **{{ .Cluster.ClusterName }}**，它根据 **SiteConfig** CR 中的值进行填充。**SiteConfig** CR 在部署过程中自动填充这些模板的值。不要手动编辑模板。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  annotations:
```

```
    argocd.argoproj.io/sync-wave: "1"
  name: "{{ .Cluster.ClusterName }}"
  namespace: "{{ .Cluster.ClusterName }}"
spec:
  clusterRef:
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
  kernelArguments:
    - operation: append (1)
      value: audit=0 (2)
    - operation: append
      value: trace=1
  sshAuthorizedKey: "{{ .Site.SshPublicKey }}"
  proxy: "{{ .Cluster.ProxySettings }}"
  pullSecretRef:
    name: "{{ .Site.PullSecretRef.Name }}"
  ignitionConfigOverride: "{{ .Cluster.IgnitionConfigOverride }}"
  nmStateConfigLabelSelector:
    matchLabels:
      nmstate-label: "{{ .Cluster.ClusterName }}"
  additionalNTPSources: "{{ .Cluster.AdditionalNTPSources }}"
```

1. 指定添加内核参数的 append 操作。
2. 指定您要配置的内核参数。这个示例配置了 audit 内核参数和 trace 内核参数。

2. 将 **InfraEnv-example.yaml** CR 提交到 Git 存储库中具有 **SiteConfig** CR 并推送您的更改的相同位置。以下示例显示了 Git 存储库结构示例：

```
~/example-ztp/install
        └── site-install
            ├── siteconfig-example.yaml
            ├── InfraEnv-example.yaml
            ...
```

3. 编辑 **SiteConfig** CR 中的 **spec.clusters.crTemplates** 规格来引用 Git 存储库中的 **InfraEnv-example.yaml** CR：

```
clusters:
  crTemplates:
    InfraEnv: "InfraEnv-example.yaml"
```

当您准备好通过提交和推送 **SiteConfig** CR 来部署集群时，构建管道会使用 Git 存储库中的自定义 **InfraEnv-example** CR 来配置基础架构环境，包括自定义内核参数。

验证

要验证是否应用了内核参数，在 Discovery 镜像验证 OpenShift Container Platform 是否准备好安装后，您可以在安装过程开始前通过 SSH 连接到目标主机。此时，您可以在 **/proc/cmdline** 文件中查看发现 ISO 的内核参数。

1. 使用目标主机开始 SSH 会话：

```
$ ssh -i /path/to/privatekey core@<host_name>
```

2. 使用以下命令查看系统的内核参数：

```
$ cat /proc/cmdline
```

4.5. 使用 SITECONFIG 和 GITOPS ZTP 部署受管集群

使用以下步骤创建 **SiteConfig** 自定义资源(CR) 和相关文件，并启动 GitOps Zero Touch Provisioning (ZTP) 集群部署。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已以具有 **cluster-admin** 权限的用户身份登录到 hub 集群。
- 配置了 hub 集群来生成所需的安装和策略 CR。
- 您创建了 Git 存储库，用于管理自定义站点配置数据。存储库必须可从 hub 集群访问，且必须将其配置为 ArgoCD 应用程序的源存储库。如需更多信息，请参阅"准备 GitOps ZTP 站点配置存储库"。

注意

在创建源存储库时，请确保使用从 **ztp-site-generate** 容器中提取的 **argocd/deployment/argocd-openshift-gitops-patch.json** patch-file 来修补 ArgoCD 应用程序。请参阅"使用 ArgoCD 配置 hub 集群"。

- 要准备好置备受管集群，每个裸机主机都需要以下内容：

网络连接

您的网络需要 DNS。受管集群主机应该可从 hub 集群访问。确保 hub 集群和受管集群主机之间存在第 3 层连接。

Baseboard Management Controller (BMC) 详情

GitOps ZTP 使用 BMC 用户名和密码详情来在集群安装过程中连接到 BMC。GitOps ZTP 插件根据站点 Git 仓库中的 **SiteConfig** CR 管理 hub 集群上的 **ManagedCluster** CR。您可以手动为每个主机创建单独的 **BMCSecret** CR。

流程

1. 在 hub 集群中创建所需的受管集群 secret。这些资源必须位于名称与集群名称匹配的命名空间中。例如，在 **out/argocd/example/siteconfig/example-sno.yaml** 中，集群名称和命名空间是 **example-sno**。

   a. 运行以下命令来导出集群命名空间：

   ```
   $ export CLUSTERNS=example-sno
   ```

   b. 创建命名空间：

   ```
   $ oc create namespace $CLUSTERNS
   ```

2. 为受管集群创建 pull secret 和 BMC **Secret** CR。pull secret 必须包含安装 OpenShift Container Platform 和其他需要安装的 Operator 所需的所有凭证。如需更多信息，请参阅"创建受管裸机主机 secret"。

注意

secret 根据名称从 **SiteConfig** 自定义资源 (CR) 引用。命名空间必须与 **SiteConfig** 命名空间匹配。

3. 在 Git 存储库本地克隆中为集群创建一个 **SiteConfig** CR：

   a. 从 **out/argocd/example/siteconfig/** 文件夹中选择适合您的 CR 示例。文件夹中包含单一节点、三节点和标准集群的示例文件：

      - **example-sno.yaml**
      - **example-3node.yaml**
      - **example-standard.yaml**

   b. 更改示例文件中的集群和主机详情，以匹配您想要的集群类型。例如：

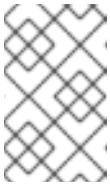
单节点 OpenShift SiteConfig CR 示例

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created
under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    # installConfigOverrides is a generic way of passing install-config
    # parameters through the siteConfig.  The 'capabilities' field configures
    # the composable openshift feature.  In this 'capabilities' setting, we
    # remove all but the marketplace component from the optional set of
    # components.
    # Notes:
    # - OperatorLifecycleManager is needed for 4.15 and later
    # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
    # - Ingress is needed for 4.16 and later
    installConfigOverrides: |
      {
        "capabilities": {
          "baselineCapabilitySet": "None",
          "additionalEnabledCapabilities": [
            "NodeTuning",
```

```
          "OperatorLifecycleManager"
          "Ingress"
        ]
      }
    }
  # It is strongly recommended to include crun manifests as part of the additional
install-time manifests for 4.13+.
  # The crun manifests can be obtained from source-crs/optional-extra-manifest/
and added to the git repo ie.sno-extra-manifest.
  # extraManifestPath: sno-extra-manifest
  clusterLabels:
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples
    du-profile: "latest"
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples in ../policygentemplates:
    # ../policygentemplates/common-ranGen.yaml will apply to all clusters with
'common: true'
    common: true
    # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all clusters with
'group-du-sno: ""'
    group-du-sno: ""
    # ../policygentemplates/example-sno-site.yaml will apply to all clusters with 'sites:
"example-sno"'
    # Normally this should match or contain the cluster name so it only applies to a
single cluster
    sites : "example-sno"
  clusterNetwork:
    - cidr: 1001:1::/48
      hostPrefix: 64
  machineNetwork:
    - cidr: 1111:2222:3333:4444::/64
  serviceNetwork:
    - 1001:2::/112
  additionalNTPSources:
    - 1111:2222:3333:4444::2
  # Initiates the cluster for workload partitioning. Setting specific reserved/isolated
CPUSets is done via PolicyTemplate
  # please see Workload Partitioning Feature for a complete guide.
  cpuPartitioningMode: AllNodes
  # Optionally; This can be used to override the KlusterletAddonConfig that is
created for this cluster:
  #crTemplates:
  #  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
  nodes:
    - hostName: "example-node1.example.com"
      role: "master"
      # Optionally; This can be used to configure desired BIOS setting on a host:
      #biosConfigRef:
      #  filePath: "example-hw.profile"
      bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embed
ded.1"
      bmcCredentialsName:
        name: "example-node1-bmh-secret"
      bootMACAddress: "AA:BB:CC:DD:EE:11"
```

```
      # Use UEFISecureBoot to enable secure boot
      bootMode: "UEFI"
      rootDeviceHints:
        deviceName: "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0"
      # disk partition at `/var/lib/containers` with ignitionConfigOverride. Some values
must be updated. See DiskPartitionContainer.md for more details
      ignitionConfigOverride: |
        {
         "ignition": {
          "version": "3.2.0"
         },
         "storage": {
          "disks": [
           {
            "device": "/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62",
            "partitions": [
             {
              "label": "var-lib-containers",
              "sizeMiB": 0,
              "startMiB": 250000
             }
          ],
          "wipeTable": false
         }
        ],
         "filesystems": [
          {
           "device": "/dev/disk/by-partlabel/var-lib-containers",
           "format": "xfs",
           "mountOptions": [
            "defaults",
            "prjquota"
           ],
           "path": "/var/lib/containers",
           "wipeFilesystem": true
          }
         ]
        },
        "systemd": {
         "units": [
          {
           "contents": "# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-
disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-
disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/di
sk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
           "enabled": true,
           "name": "var-lib-containers.mount"
          }
         ]
        }
       }
      nodeNetwork:
        interfaces:
```

```
    - name: eno1
      macAddress: "AA:BB:CC:DD:EE:11"
  config:
    interfaces:
      - name: eno1
        type: ethernet
        state: up
        ipv4:
          enabled: false
        ipv6:
          enabled: true
          address:
          # For SNO sites with static IP addresses, the node-specific,
          # API and Ingress IPs should all be the same and configured on
          # the interface
          - ip: 1111:2222:3333:4444::aaaa:1
            prefix-length: 64
    dns-resolver:
      config:
        search:
        - example.com
        server:
        - 1111:2222:3333:4444::2
    routes:
      config:
      - destination: ::/0
        next-hop-interface: eno1
        next-hop-address: 1111:2222:3333:4444::1
        table-id: 254
```



注意

有关 BMC 寻址的更多信息，请参阅"添加资源"部分。**installConfigOverrides** 和 **ignitionConfigOverride** 字段在示例中扩展，以简化可读性。

c. 您可以在 **out/argocd/extra-manifest** 中检查默认的 extra-manifest **MachineConfig** CR。它在安装时会自动应用到集群。

d. 可选：要在置备的集群中置备额外的安装清单，请在 Git 存储库中创建一个目录，如 **sno-extra-manifest/**，并将自定义清单 CR 添加到这个目录中。如果您的 **SiteConfig.yaml** 在 **extraManifestPath** 字段中引用这个目录，则这个引用目录中的所有 CR 都会被附加到默认的额外的清单集合中。

启用 CRUN OCI 容器运行时

为获得最佳的集群性能，请在单节点 OpenShift 中为 master 和 worker 节点启用 crun，使用额外的 worker 节点、三节点 OpenShift 和标准集群中启用单节点 OpenShift。

在 **ContainerRuntimeConfig** CR 中启用 crun，作为额外的第-0 天安装时间清单，以避免集群需要重启。

enable-crun-master.yaml 和 **enable-crun-worker.yaml** CR 文件位于您可以从 **ztp-site-generate** 容器中提取的 **out/source-crs/optional-extra-manifest/** 文件夹中。如需更多信息，请参阅"在 GitOps ZTP 管道中自定义额外安装清单"。

4. 在 **kustomization.yaml** 文件中将 **SiteConfig** CR 添加到 **generators** 部分中，类似于 **out/argocd/example/siteconfig/kustomization.yaml** 中显示的示例。

5. 在 Git 存储库中提交 **SiteConfig** CR 及关联的 **kustomization.yaml** 更改并推送更改。ArgoCD 管道检测到更改并开始受管集群部署。

验证

- 验证在部署节点后是否应用了自定义角色和标签：

```
$ oc describe node example-node.example.com
```

输出示例

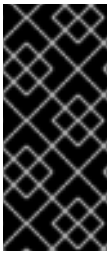
```
Name:   example-node.example.com
Roles:  control-plane,example-label,master,worker
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        custom-label/parameter1=true
        kubernetes.io/arch=amd64
        kubernetes.io/hostname=cnfdf03.telco5gran.eng.rdu2.redhat.com
        kubernetes.io/os=linux
        node-role.kubernetes.io/control-plane=
        node-role.kubernetes.io/example-label= (1)
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

(1) 自定义标签应用到节点。

其他资源

- 单节点 OpenShift SiteConfig CR 安装参考

4.5.1. 加速置备 GitOps ZTP

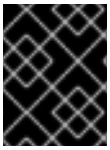


重要

GitOps ZTP 的加速置备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。

您可以通过为单节点 OpenShift 使用 GitOps ZTP 的加速置备来减少集群安装所需的时间。加速 ZTP 通过在早期阶段应用从策略派生的第 2 天清单来加快安装速度。



重要

只有在使用 Assisted Installer 安装单节点 OpenShift 时，才支持加速 GitOps ZTP 置备。否则，这个安装方法将失败。

4.5.1.1. 激活加速 ZTP

您可以使用 **spec.clusters.clusterLabels.accelerated-ztp** 标签激活加速 ZTP，如下例所示：

加速 ZTP SiteConfig CR 示例。

```
apiVersion: ran.openshift.io/v2
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  # ...
    clusterLabels:
      common: true
      group-du-sno: ""
      sites : "example-sno"
      accelerated-ztp: full
```

您可以使用 **accelerated-ztp: full** 来完全自动化加速过程。GitOps ZTP 使用对加速 GitOps ZTP **ConfigMap** 的引用来更新 **AgentClusterInstall** 资源，并包括 TALM 从策略提取的资源，以及加速 ZTP 作业清单。

如果您使用 **accelerated-ztp: 部分**，GitOps ZTP 不包括加速的作业清单，而是包括集群安装以下类型时创建的 policy-derived 对象：

- **PerformanceProfile.performance.openshift.io**

- Tuned.tuned.openshift.io
- Namespace
- CatalogSource.operators.coreos.com
- ContainerRuntimeConfig.machineconfiguration.openshift.io

在应用 Performance Profile、Tuned 和 ContainerRuntimeConfig 资源时，这个部分加速可以减少节点完成的重启数量。TALM 在 RHACM 完成导入集群后安装从策略派生的 Operator 订阅，遵循与标准 GitOps ZTP 相同的流程。

加速 ZTP 的好处会随着部署的规模而增加。使用 accelerated-ztp ： 对大量集群提供了更多好处。使用较少的集群，安装时间缩短会显著减少。全加速 ZTP 在需要手动删除的 spoke 上保留一个命名空间和完成的作业。

使用 accelerated-ztp: 部分 的好处之一是，如果库存实施出现问题，或者需要自定义功能，您可以覆盖 on-spoke 任务的功能。

4.5.1.2. 加速 ZTP 进程

加速 ZTP 使用额外的 ConfigMap 来创建从 spoke 集群上的策略派生的资源。标准 ConfigMap 包含 GitOps ZTP 工作流用来自定义集群安装的清单。

TALM 检测到设置了 accelerated-ztp 标签，然后创建第二个 ConfigMap。作为加速 ZTP 的一部分，SiteConfig 生成器使用命名约定 < spoke-cluster-name>-aztp 添加第二个 ConfigMap 的引用。

TALM 创建第二个 ConfigMap 后，它会找到绑定到受管集群的所有策略，并提取 GitOps ZTP 配置集信息。TALM 将 GitOps ZTP 配置集信息添加到 < spoke-cluster-name>-aztp ConfigMap 自定义资源(CR)中，并将 CR 应用到 hub 集群 API。

4.5.2. 使用 GitOps ZTP 和 SiteConfig 资源为单节点 OpenShift 集群配置 IPsec 加密

您可以使用 GitOps ZTP 和 Red Hat Advanced Cluster Management (RHACM)在受管单节点 OpenShift 集群中启用 IPsec 加密。您可以加密受管集群外部 pod 和 IPsec 端点之间的外部流量。OVN-

Kubernetes 集群网络上的节点之间的所有 pod 到 pod 网络流量都使用 IPsec 在传输模式中加密。

注意

在 OpenShift Container Platform 4.16 中，使用 GitOps ZTP 和 RHACM 部署 IPsec 加密，只针对单节点 OpenShift 集群验证。

GitOps ZTP IPsec 实现假设您部署到资源受限平台。因此，您只能使用单个 MachineConfig CR 安装该功能，您不需要在单节点 OpenShift 集群上安装 NMState Operator。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已配置了 RHACM 和 hub 集群，为受管集群生成所需的安装和策略自定义资源(CR)。
- 您已创建了管理自定义站点配置数据的 Git 存储库。该存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源仓库。
- 已安装 butane 工具，版本 0.20.0 或更高版本。
- 您有一个用于 IPsec 端点和 PEM 格式的 CA 证书的 PKCSautomationhub 证书。

流程

1. 提取 ztp-site-generate 容器源的最新版本，并将其与您管理自定义站点配置数据的存储库合并。
2. 使用在集群中配置 IPsec 所需的值来配置 optional-extra-manifest/ipsec/ipsec-endpoint-config.yaml。例如：

```
interfaces:
```

```
- name: hosta_conn
  type: ipsec
  libreswan:
    left: <cluster_node> 1
    leftid: '%fromcert'
    leftmodecfgclient: false
    leftcert: <left_cert> 2
    leftrsasigkey: '%cert'
    right: <external_host> 3
    rightid: '%fromcert'
    rightrsasigkey: '%cert'
    rightsubnet: <external_address> 4
    ikev2: insist 5
    type: tunnel
```

1 将 <cluster_node > 替换为 cluster-side IPsec 隧道的集群节点的 IP 地址或 DNS 主机名。

2 将 <left_cert& gt; 替换为 IPsec 证书 nickname。

3 使用 <external_host> 外部主机 IP 地址或 DNS 主机名替换。

4 在 IPsec 隧道的另一端，使用外部主机的 IP 地址或子网替换 <external_address>。

5 只使用 IKEv2 VPN 加密协议。不要使用 IKEv1，它已弃用。

3. 将您的 ca.pem 和 left_server.p12 证书添加到 optional-extra-manifest/ipsec 文件夹中。每个主机上的网络安全服务(NSS)数据库都需要证书文件。这些文件在后续步骤中作为 Butane 配置的一部分导入。

   a. left_server.p12：IPsec 端点的证书捆绑包

   b. ca.pem ：您使用签名证书的证书颁发机构

4. 在您维护自定义站点配置数据的 Git 存储库的 optional-extra-manifest/ipsec 文件夹下打开 shell 提示符。

5. 运行 optional-extra-manifest/ipsec/build.sh 脚本来生成所需的 Butane 和 MachineConfig CR 文件。

输出示例

```
out
└── argocd
    └── example
        └── optional-extra-manifest
            └── ipsec
                ├── 99-ipsec-master-endpoint-config.bu (1)
                ├── 99-ipsec-master-endpoint-config.yaml
                ├── 99-ipsec-worker-endpoint-config.bu
                ├── 99-ipsec-worker-endpoint-config.yaml
                ├── build.sh
                ├── ca.pem (2)
                ├── left_server.p12
                ├── enable-ipsec.yaml
                ├── ipsec-endpoint-config.yml
                └── README.md
```

(1) ipsec/build.sh 脚本生成 Butane 和端点配置 CR。

(2) 您提供与网络相关的 ca.pem 和 left_server.p12 证书文件。

6. 在您管理自定义站点配置数据的存储库中创建 custom-manifest/ 文件夹。将 enable-ipsec.yaml 和 99-ipsec fluentd YAML 文件添加到目录中。例如：

```
siteconfig
├── site1-sno-du.yaml
├── extra-manifest/
└── custom-manifest
```

```
        ├── enable-ipsec.yaml
        ├── 99-ipsec-worker-endpoint-config.yaml
        └── 99-ipsec-master-endpoint-config.yaml
```

7. 在 SiteConfig CR 中，将 custom-manifest/ 目录添加到 extraManifests.searchPaths 字段中。例如：

```
clusters:
- clusterName: "site1-sno-du"
  networkType: "OVNKubernetes"
  extraManifests:
    searchPaths:
      - extra-manifest/
      - custom-manifest/
```

8. 提交 SiteConfig CR 更改和更新的文件，并推送更改以置备受管集群并配置 IPsec 加密。

   Argo CD 管道检测到更改并开始受管集群部署。

   在集群置备过程中，GitOps ZTP 管道会将 /custom-manifest 目录中的 CR 附加到存储在 extra-manifest/ 中的默认额外清单集合中。

验证

要验证 IPsec 加密是否在受管单节点 OpenShift 集群中成功应用，请执行以下步骤：

1. 运行以下命令为受管集群启动 debug pod:

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

2. 检查 IPsec 策略是否在集群节点中应用：

```
sh-5.1# ip xfrm policy
```

输出示例

```
src 172.16.123.0/24 dst 10.1.232.10/32
  dir out priority 1757377 ptype main
  tmpl src 10.1.28.190 dst 10.1.232.10
```

```
    proto esp reqid 16393 mode tunnel
src 10.1.232.10/32 dst 172.16.123.0/24
  dir fwd priority 1757377 ptype main
  tmpl src 10.1.232.10 dst 10.1.28.190
    proto esp reqid 16393 mode tunnel
src 10.1.232.10/32 dst 172.16.123.0/24
  dir in priority 1757377 ptype main
  tmpl src 10.1.232.10 dst 10.1.28.190
    proto esp reqid 16393 mode tunnel
```

3. 检查 **IPsec 隧道是否已启动并连接：**

```
sh-5.1# ip xfrm state
```

输出示例

```
src 10.1.232.10 dst 10.1.28.190
  proto esp spi 0xa62a05aa reqid 16393 mode tunnel
  replay-window 0 flag af-unspec esn
  auth-trunc hmac(sha1) 0x8c59f680c8ea1e667b665d8424e2ab749cec12dc 96
  enc cbc(aes)
0x2818a489fe84929c8ab72907e9ce2f0eac6f16f2258bd22240f4087e0326badb
  anti-replay esn context:
   seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
   replay_window 128, bitmap-length 4
   00000000 00000000 00000000 00000000
src 10.1.28.190 dst 10.1.232.10
  proto esp spi 0x8e96e9f9 reqid 16393 mode tunnel
  replay-window 0 flag af-unspec esn
  auth-trunc hmac(sha1) 0xd960ddc0a6baaccb343396a51295e08cfd8aaddd 96
  enc cbc(aes)
0x0273c02e05b4216d5e652de3fc9b3528fea94648bc2b88fa01139fdf0beb27ab
  anti-replay esn context:
   seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
   replay_window 128, bitmap-length 4
   00000000 00000000 00000000 00000000
```

4. **在外部主机子网中 ping 已知 IP。例如，ping 您在 ipsec/ipsec-endpoint-config.yaml 中设置的 rightsubnet 范围内的 IP：**

```
sh-5.1# ping 172.16.110.8
```

输出示例

```
sh-5.1# ping 172.16.110.8
PING 172.16.110.8 (172.16.110.8) 56(84) bytes of data.
64 bytes from 172.16.110.8: icmp_seq=1 ttl=64 time=153 ms
64 bytes from 172.16.110.8: icmp_seq=2 ttl=64 time=155 ms
```

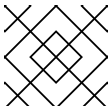
其他资源

- 配置 IPsec 加密
- 加密协议和 IPsec 模式
- 使用 RHACM 和 SiteConfig 资源安装受管集群

4.5.3. 单节点 OpenShift SiteConfig CR 安装参考

表 4.1. 单节点 OpenShift 集群的 SiteConfig CR 安装选项

SiteConfig CR 字段	描述
spec.cpuPartitioningMode	通过将 **cpuPartitioningMode** 的值设置为 **AllNodes** 来配置工作负载分区。要完成配置，请在 **PerformanceProfile** CR 中指定 **isolated** 和 **reserved** CPU。  **注意** 使用 **SiteConfig** CR 中的 **cpuPartitioningMode** 字段配置工作负载分区是 OpenShift Container Platform 4.13 中的技术预览功能。
metadata.name	将 **name** 设置为 **assisted-deployment-pull-secret**，并在与 **SiteConfig** CR 相同的命名空间中创建 **assisted-deployment-pull-secret** CR。
spec.clusterImageSetNameRef	为站点中的所有集群配置 hub 集群上可用的镜像集。要查看 hub 集群上支持的版本列表，请运行 **oc get clusterimagesets**。

SiteConfig CR 字段	描述
installConfigOverrides	将 **installConfigOverrides** 字段设置为在集群安装前启用或禁用可选组件。  **重要** 使用示例 **SiteConfig** CR 中指定的引用配置。在系统中添加其他组件可能需要额外的保留 CPU 容量。
spec.clusters.clusterImageSetNameRef	指定用于部署单个集群的集群镜像集。如果定义，它会在站点级别覆盖 **spec.clusterImageSetNameRef**。
spec.clusters.clusterLabels	配置集群标签，以对应于您定义的 **PolicyGenerator** 或 **PolicyGentemplate** CR 中的绑定规则。**PolicyGenerator** CR 使用 **policyDefaults.placement.labelSelector** 字段。**PolicyGenTemplate** CR 使用 **spec.bindingRules** 字段。 例如，**acmpolicygenerator/acm-common-ranGen.yaml** 应用到所有带有 **common: true** 设置的集群，**acmpolicygenerator/acm-group-du-sno-ranGen.yaml** 应用到带有 **group-du-sno: ""** 设置的所有集群。
spec.clusters.crTemplates.KlusterletAddonConfig	可选。将 **KlusterletAddonConfig** 设置为 **KlusterletAddonConfigOverride.yaml，以覆盖为集群创建的默认 'KlusterletAddonConfig**。
spec.clusters.nodes.hostName	对于单节点部署，请定义一个主机。对于三节点部署，请定义三个主机。对于标准部署，使用 **role: master** 定义三个主机，使用 **role: worker** 定义两个或更多主机。
spec.clusters.nodes.nodeLabels	为受管集群中的节点指定自定义角色。这些是其他角色，不供任何 OpenShift Container Platform 组件使用，仅限用户使用。添加自定义角色时，它可以与引用该角色的特定配置的自定义机器配置池相关联。在安装过程中添加自定义标签或角色可使部署过程更高效，并防止在安装完成后进行额外的重启。
spec.clusters.nodes.automatedCleaningMode	可选。取消注释并将值设置为 **metadata**，以只启用删除磁盘分区表，而无需完全擦除磁盘。默认值为 **Disabled**。
spec.clusters.nodes.bmcAddress	用于访问主机的 BMC 地址。适用于所有集群类型。GitOps ZTP 支持使用 Redfish 或 IPMI 协议进行 iPXE 和虚拟介质引导。要使用 iPXE 启动，您必须使用 RHACM 2.8 或更高版本。有关 BMC 寻址的更多信息，请参阅"添加资源"部分。
spec.clusters.nodes.bmcAddress	用于访问主机的 BMC 地址。适用于所有集群类型。GitOps ZTP 支持使用 Redfish 或 IPMI 协议进行 iPXE 和虚拟介质引导。要使用 iPXE 启动，您必须使用 RHACM 2.8 或更高版本。有关 BMC 寻址的更多信息，请参阅"添加资源"部分。  **注意** 在边缘 Telco 用例中，只有虚拟介质可用于 GitOps ZTP。

SiteConfig CR 字段	描述
spec.clusters.nodes.bmcCredentialsName	配置使用主机 BMC 凭证单独创建的 **bmh-secret** CR。在创建 **bmh-secret** CR 时，请使用与置备主机的 **SiteConfig** CR 相同的命名空间。
spec.clusters.nodes.bootMode	将主机的引导模式设置为 **UEFI**。默认值为 **UEFI**。使用 **UEFISecureBoot** 在主机上启用安全引导。
spec.clusters.nodes.rootDeviceHints	指定用于部署的设备。建议在重启后保持稳定标识符。例如，**wwn: <disk_wwn>** 或 **deviceName: /dev/disk/by-path/<device_path>**。**<by-path&** gt; 值是首选的。有关 stable 标识符的详细列表，请参阅"About root device hints"部分。
spec.clusters.nodes.ignitionConfigOverride	可选。使用此字段为持久性存储分配分区。将磁盘 ID 和大小调整为特定的硬件。
spec.clusters.nodes.nodeNetwork	配置节点的网络设置。
spec.clusters.nodes.nodeNetwork.config.interfaces.ipv6	为主机配置 IPv6 地址。对于带有静态 IP 地址的单节点 OpenShift 集群，特定于节点的 API 和 Ingress IP 应该相同。

其他资源

- **在 GitOps ZTP 管道中自定义额外的安装清单**
- **准备 GitOps ZTP 站点配置存储库**
- **使用 ArgoCD 配置 hub 集群**
- **使用验证器通知策略信号 GitOps ZTP 集群部署完成**
- **创建受管裸机主机 secret**
- **BMC 地址**

- 关于 root 设备提示

4.6. 监控受管集群安装进度

ArgoCD 管道使用 SiteConfig CR 生成集群配置 CR，并将其与 hub 集群同步。您可以在 ArgoCD 仪表板中监控此同步的进度。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

同步完成后，安装通常会按如下方式进行：

1. Assisted Service Operator 会在集群中安装 OpenShift Container Platform。您可以运行以下命令来从 RHACM 仪表板或命令行监控集群安装进度：
   a. 导出集群名称：

```
$ export CLUSTER=<clusterName>
```

   b. 查询受管集群的 AgentClusterInstall CR：

```
$ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o
jsonpath='{.status.conditions[?(@.type=="Completed")]}' | jq
```

   c. 获取集群的安装事件：

```
$ curl -sk $(oc get agentclusterinstall -n $CLUSTER $CLUSTER -o
jsonpath='{.status.debugInfo.eventsURL}') | jq '.[-2,-1]'
```

4.7. 通过验证安装 CR 对 GITOPS ZTP 进行故障排除

ArgoCD 管道使用 SiteConfig 和 PolicyGenerator 或 PolicyGentemplate 自定义资源(CR)生成集群配置 CR 和 Red Hat Advanced Cluster Management (RHACM)策略。使用以下步骤对此过程中可能出现的问题进行故障排除。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 您可以使用以下命令检查安装 CR 是否已创建：

```
$ oc get AgentClusterInstall -n <cluster_name>
```

如果没有返回对象，请使用以下步骤对从 SiteConfig 文件到安装 CR 的 ArgoCD 管道流进行故障排除。

2. 验证 ManagedCluster CR 是否使用 hub 集群上的 SiteConfig CR 生成：

```
$ oc get managedcluster
```

3. 如果缺少 ManagedCluster，请检查 clusters 应用程序是否将 Git 存储库中的文件与 hub 集群同步：

```
$ oc describe -n openshift-gitops application clusters
```

a. 检查 Status.Conditions 字段以查看受管集群的错误日志。例如，在 SiteConfig CR 中为 extraManifestPath: 设置无效的值会引发以下错误：

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/siteconfigs/ --enable-alpha-plugins` failed exit
status 1: 2021/11/26 17:21:40 Error could not create extra-manifest ranSite1.extra-
manifest3 stat extra-manifest3: no such file or directory 2021/11/26 17:21:40 Error:
could not build the entire SiteConfig defined by /tmp/kust-plugin-config-
```

```
913473579: stat extra-manifest3: no such file or directory Error: failure in plugin
configured via /tmp/kust-plugin-config-913473579; exit status 1: exit status 1
   Type:  ComparisonError
```

b. 检查 Status.Sync 字段。如果有日志错误，Status.Sync 字段可能会指示 Unknown 错误：

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  clusters-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:           sites-config
        Repo URL:       https://git.com/ran-sites/siteconfigs/.git
        Target Revision:  master
    Status:             Unknown
```

4.8. 在 SUPERMICRO 服务器中对 GITOPS ZTP 虚拟介质引导进行故障排除

在使用 https 协议提供镜像时，Supermicro X11 服务器不支持虚拟介质安装。因此，此环境的单节点 OpenShift 部署无法在目标节点上引导。要避免这个问题，请登录到 hub 集群并禁用 Provisioning 资源中的传输层安全 (TLS)。这样可确保镜像不通过 TLS 提供，即使镜像地址使用 https 方案。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 运行以下命令，在 Provisioning 资源中禁用 TLS：

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec":
{"disableVirtualMediaTLS": true}}'
```

2. 继续部署单节点 OpenShift 集群的步骤。

4.9. 从 GITOPS ZTP 管道中删除受管集群站点

您可以从 GitOps Zero Touch Provisioning (ZTP) 管道中删除受管站点以及关联的安装和配置策略 CR。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 通过从 kustomization.yaml 文件中删除关联的 SiteConfig 和 PolicyGentemplate 文件来删除站点和相关 CR。

   当您再次运行 GitOps ZTP 管道时，生成的 CR 会被删除。

2. 可选：如果要永久删除站点，您还应从 Git 仓库中删除 SiteConfig 和特定站点的 PolicyGenerator 或 PolicyGentemplate 文件。

3. 可选：如果要临时删除站点，例如在重新部署站点时，可以保留 Git 存储库中的 SiteConfig 和特定站点的 PolicyGenerator 或 PolicyGentemplate CR。

其他资源

- 有关删除集群的详情，请参考从管理中删除集群。

4.10. 从 GITOPS ZTP 管道中删除过时的内容

如果对 PolicyGenerator 或 PolicyGentemplate 配置的更改会导致过时的策略，例如，如果您重命名策略，请使用以下步骤删除过时的策略。

先决条件

- 已安装 OpenShift CLI(oc)。

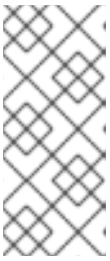
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 从 Git 存储库中删除受影响的 PolicyGenerator 或 PolicyGentemplate 文件，提交并推送到远程存储库。

2. 等待更改通过应用程序同步，并将受影响的策略从 hub 集群中删除。

3. 将更新的 PolicyGenerator 或 PolicyGentemplate 文件重新添加到 Git 存储库，然后提交并推送到远程存储库。



注意

从 Git 仓库中删除 GitOps Zero Touch Provisioning (ZTP) 策略，因此也会从 hub 集群中删除它们，不会影响受管集群的配置。由该策略管理的策略和 CR 保留在受管集群上。

4. 可选：作为替代方案，在更改生成过时策略的 PolicyGenerator 或 PolicyGentemplate CR 后，您可以手动从 hub 集群中删除这些策略。您可以使用 Governance 选项卡或运行以下命令来从 RHACM 控制台删除策略：

```
$ oc delete policy -n <namespace> <policy_name>
```

4.11. 弃用 GITOPS ZTP 管道

您可以删除 ArgoCD 管道和所有生成的 GitOps Zero Touch Provisioning (ZTP) 工件。

先决条件

- 已安装 OpenShift CLI(oc)。

- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 从 hub 集群上的 Red Hat Advanced Cluster Management (RHACM)分离所有集群。

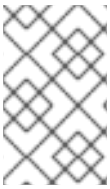
2. 使用以下命令，删除 deployment 目录中的 kustomization.yaml 文件：

```
$ oc delete -k out/argocd/deployment
```

3. 提交您的更改并推送到站点存储库。

第 5 章 使用 GITOPS ZTP 手动安装单节点 OPENSHIFT 集群

您可以使用 Red Hat Advanced Cluster Management (RHACM) 和支持的服务部署受管单节点 OpenShift 集群。



注意

如果要创建多个受管集群，请参阅使用 ZTP 部署边缘站点中描述的 SiteConfig 方法。

重要

目标裸机主机必须满足 vDU 应用程序工作负载的推荐集群配置中列出的网络、固件和硬件要求。

5.1. 手动生成 GITOPS ZTP 安装和配置 CR

使用 ztp-site-generate 容器的 generator 入口点，根据 SiteConfig 和 PolicyGenerator CR 为集群生成站点安装和配置自定义资源(CR)。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. 运行以下命令来创建输出文件夹：

```
$ mkdir -p ./out
```

2. 从 ztp-site-generate 容器镜像导出 argocd 目录：

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./out
```

./out 目录包含 out/argocd/example/ 文件夹中的参考 PolicyGenerator 和 SiteConfig CR。

输出示例

```
out
└── argocd
    └── example
        ├── acmpolicygenerator
        │   ├── {policy-prefix}common-ranGen.yaml
        │   ├── {policy-prefix}example-sno-site.yaml
        │   ├── {policy-prefix}group-du-sno-ranGen.yaml
        │   ├── {policy-prefix}group-du-sno-validator-ranGen.yaml
        │   ├── ...
        │   ├── kustomization.yaml
        │   └── ns.yaml
        └── siteconfig
            ├── example-sno.yaml
            ├── KlusterletAddonConfigOverride.yaml
            └── kustomization.yaml
```

3. 为站点安装 CR 创建输出文件夹：

```
$ mkdir -p ./site-install
```

4. 为您要安装的集群类型修改示例 SiteConfig CR。将 example-sno.yaml 复制到 site-1-sno.yaml，并修改 CR 以匹配您要安装的站点和裸机主机的详情，例如：

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
```

```
  networkType: "OVNKubernetes"
  # installConfigOverrides is a generic way of passing install-config
  # parameters through the siteConfig.  The 'capabilities' field configures
  # the composable openshift feature.  In this 'capabilities' setting, we
  # remove all but the marketplace component from the optional set of
  # components.
  # Notes:
  # - OperatorLifecycleManager is needed for 4.15 and later
  # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
  # - Ingress is needed for 4.16 and later
  installConfigOverrides: |
    {
      "capabilities": {
        "baselineCapabilitySet": "None",
        "additionalEnabledCapabilities": [
          "NodeTuning",
          "OperatorLifecycleManager"
          "Ingress"
        ]
      }
    }
  # It is strongly recommended to include crun manifests as part of the additional
install-time manifests for 4.13+.
  # The crun manifests can be obtained from source-crs/optional-extra-manifest/ and
added to the git repo ie.sno-extra-manifest.
  # extraManifestPath: sno-extra-manifest
  clusterLabels:
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples
    du-profile: "latest"
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples in ../policygentemplates:
    # ../policygentemplates/common-ranGen.yaml will apply to all clusters with
'common: true'
    common: true
    # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all clusters with
'group-du-sno: ""'
    group-du-sno: ""
    # ../policygentemplates/example-sno-site.yaml will apply to all clusters with 'sites:
"example-sno"'
    # Normally this should match or contain the cluster name so it only applies to a
single cluster
    sites : "example-sno"
  clusterNetwork:
    - cidr: 1001:1::/48
      hostPrefix: 64
  machineNetwork:
    - cidr: 1111:2222:3333:4444::/64
  serviceNetwork:
    - 1001:2::/112
  additionalNTPSources:
    - 1111:2222:3333:4444::2
  # Initiates the cluster for workload partitioning. Setting specific reserved/isolated
CPUSets is done via PolicyTemplate
  # please see Workload Partitioning Feature for a complete guide.
  cpuPartitioningMode: AllNodes
```

```
    # Optionally; This can be used to override the KlusterletAddonConfig that is created
for this cluster:
    #crTemplates:
    #  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
    nodes:
      - hostName: "example-node1.example.com"
        role: "master"
        # Optionally; This can be used to configure desired BIOS setting on a host:
        #biosConfigRef:
        #  filePath: "example-hw.profile"
        bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embed
ded.1"
        bmcCredentialsName:
          name: "example-node1-bmh-secret"
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        # Use UEFISecureBoot to enable secure boot
        bootMode: "UEFI"
        rootDeviceHints:
          deviceName: "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0"
        # disk partition at `/var/lib/containers` with ignitionConfigOverride. Some values
must be updated. See DiskPartitionContainer.md for more details
        ignitionConfigOverride: |
          {
           "ignition": {
            "version": "3.2.0"
           },
           "storage": {
            "disks": [
             {
              "device": "/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62",
              "partitions": [
               {
                "label": "var-lib-containers",
                "sizeMiB": 0,
                "startMiB": 250000
               }
             ],
             "wipeTable": false
            }
           ],
           "filesystems": [
            {
             "device": "/dev/disk/by-partlabel/var-lib-containers",
             "format": "xfs",
             "mountOptions": [
              "defaults",
              "prjquota"
             ],
             "path": "/var/lib/containers",
             "wipeFilesystem": true
            }
           ]
          },
          "systemd": {
           "units": [
```

```
          {
           "contents": "# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-
disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-
disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
           "enabled": true,
           "name": "var-lib-containers.mount"
          }
         ]
        }
       }
    nodeNetwork:
     interfaces:
      - name: eno1
        macAddress: "AA:BB:CC:DD:EE:11"
     config:
      interfaces:
       - name: eno1
         type: ethernet
         state: up
         ipv4:
           enabled: false
         ipv6:
           enabled: true
           address:
           # For SNO sites with static IP addresses, the node-specific,
           # API and Ingress IPs should all be the same and configured on
           # the interface
           - ip: 1111:2222:3333:4444::aaaa:1
             prefix-length: 64
      dns-resolver:
       config:
         search:
         - example.com
         server:
         - 1111:2222:3333:4444::2
      routes:
       config:
       - destination: ::/0
         next-hop-interface: eno1
         next-hop-address: 1111:2222:3333:4444::1
         table-id: 254
```

注意

从 ztp-site-generate 容器的 out/extra-manifest 目录中提取引用 CR 配置文件后，您可以使用 extraManifests.searchPaths 来包含包含这些文件的 git 目录的路径。这允许 GitOps ZTP 管道在集群安装过程中应用这些 CR 文件。如果您配置 searchPaths 目录，GitOps ZTP 管道不会在站点安装过程中从 ztp-site-generate 容器获取清单。

5. 运行以下命令，通过处理修改后的 SiteConfig CR site-1-sno.yaml 来生成第 0 天安装 CR：

```
$ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v
`pwd`/site-install:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16
generator install site-1-sno.yaml /output
```

输出示例

```
site-install
└── site-1-sno
    ├── site-1_agentclusterinstall_example-sno.yaml
    ├── site-1-sno_baremetalhost_example-node1.example.com.yaml
    ├── site-1-sno_clusterdeployment_example-sno.yaml
    ├── site-1-sno_configmap_example-sno.yaml
    ├── site-1-sno_infraenv_example-sno.yaml
    ├── site-1-sno_klusterletaddonconfig_example-sno.yaml
    ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
    ├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
    ├── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
    ├── site-1-sno_managedcluster_example-sno.yaml
    ├── site-1-sno_namespace_example-sno.yaml
    └── site-1-sno_nmstateconfig_example-node1.example.com.yaml
```

6. 可选：使用 -E 选项处理参考 SiteConfig CR，只为特定集群类型生成 day-0 MachineConfig 安装 CR。例如，运行以下命令：

   a. 为 MachineConfig CR 创建输出文件夹：

   ```
   $ mkdir -p ./site-machineconfig
   ```

   b. 生成 MachineConfig 安装 CR：

   ```
   $ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v
   `pwd`/site-machineconfig:/output:Z,U registry.redhat.io/openshift4/ztp-site-
   generate-rhel8:v4.16 generator install -E site-1-sno.yaml /output
   ```

   输出示例

```
site-machineconfig
└── site-1-sno
    ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
    ├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
    └── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
```

7. 使用上一步中的引用 PolicyGenerator CR 生成并导出第 2 天配置 CR。运行以下命令：

a. 为 day-2 CR 创建输出文件夹：

```
$ mkdir -p ./ref
```

b. 生成并导出第 2 天配置 CR：

```
$ podman run -it --rm -v
`pwd`/out/argocd/example/acmpolicygenerator:/resources:Z -v
`pwd`/ref:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16
generator config -N . /output
```

该命令在 ./ref 文件夹中为单节点 OpenShift、三节点集群和标准集群生成示例组和特定于站点的 PolicyGenerator CR。

输出示例

```
ref
└── customResource
    ├── common
    ├── example-multinode-site
    ├── example-sno
    ├── group-du-3node
    ├── group-du-3node-validator
    │   └── Multiple-validatorCRs
    ├── group-du-sno
    ├── group-du-sno-validator
    ├── group-du-standard
    └── group-du-standard-validator
        └── Multiple-validatorCRs
```

8. 使用生成的 CR 作为安装集群的 CR 的基础。您可以将安装 CR 应用到 hub 集群，如 "Installing a single managed cluster" 所述。配置 CR 可以在集群安装后应用到集群。

验证

- 验证在部署节点后是否应用了自定义角色和标签：

```
$ oc describe node example-node.example.com
```

输出示例

```
Name:   example-node.example.com
Roles:  control-plane,example-label,master,worker
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        custom-label/parameter1=true
        kubernetes.io/arch=amd64
        kubernetes.io/hostname=cnfdf03.telco5gran.eng.rdu2.redhat.com
        kubernetes.io/os=linux
        node-role.kubernetes.io/control-plane=
        node-role.kubernetes.io/example-label= 1
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

1 自定义标签应用到节点。

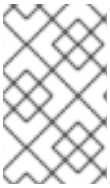
其他资源

- 工作负载分区
- BMC 地址

- 关于 root 设备提示
- 单节点 OpenShift SiteConfig CR 安装参考

5.2. 创建受管裸机主机 SECRET

将受管裸机主机所需的 Secret 自定义资源 (CR) 添加到 hub 集群。您需要 GitOps Zero Touch Provisioning (ZTP) 管道的 secret 来访问 Baseboard Management Controller (BMC) 和支持的安装程序服务的 secret，以便从 registry 中拉取集群安装镜像。



注意

secret 按名称从 SiteConfig CR 引用。命名空间必须与 SiteConfig 命名空间匹配。

流程

1. 创建一个 YAML secret 文件，其中包含主机 Baseboard Management Controller (BMC) 和安装 OpenShift 和所有附加组件集群 Operator 所需的凭证：

   a. 将以下 YAML 保存为文件 example-sno-secret.yaml ：

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno (3)
data:
  .dockerconfigjson: <pull_secret> (4)
type: kubernetes.io/dockerconfigjson
```

(1)

必须与相关 SiteConfig CR 中配置的命名空间匹配

2

password 和 username 的 base64 编码值

3

必须与相关 SiteConfig CR 中配置的命名空间匹配

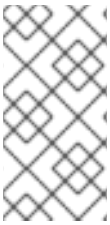
4

Base64 编码的 pull secret

2. 将到 example-sno-secret.yaml 的相对路径添加用于安装集群的 kustomization.yaml 文件中。

5.3. 使用 GITOPS ZTP 为手动安装配置 DISCOVERY ISO 内核参数

GitOps Zero Touch Provisioning (ZTP) 工作流使用 Discovery ISO 作为托管裸机主机的 OpenShift Container Platform 安装过程的一部分。您可以编辑 InfraEnv 资源来为 Discovery ISO 指定内核参数。这对具有特定环境要求的集群安装非常有用。例如，为发现 ISO 配置 rd.net.timeout.carrier 内核参数以促进集群的静态网络，或者在在安装过程中下载根文件系统前接收 DHCP 地址。



注意

在 OpenShift Container Platform 4.16 中，您只能添加内核参数。您不能替换或删除内核参数。

先决条件

- 已安装 OpenShift CLI（oc）。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已手动生成安装和配置自定义资源(CR)。

流程

1. 编辑 InfraEnv CR 中的 spec.kernelArguments 规格以配置内核参数：

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  kernelArguments:
    - operation: append 1
      value: audit=0 2
    - operation: append
      value: trace=1
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  pullSecretRef:
    name: pull-secret
```

1 指定添加内核参数的 append 操作。

2 指定您要配置的内核参数。这个示例配置了 audit 内核参数和 trace 内核参数。

注意

SiteConfig CR 生成 InfraEnv 资源，作为 day-0 安装 CR 的一部分。

验证

要验证是否应用了内核参数，在 Discovery 镜像验证 OpenShift Container Platform 是否准备好安装后，您可以在安装过程开始前通过 SSH 连接到目标主机。此时，您可以在 /proc/cmdline 文件中查看发现 ISO 的内核参数。

1. 使用目标主机开始 SSH 会话：

```
$ ssh -i /path/to/privatekey core@<host_name>
```

2. 使用以下命令查看系统的内核参数：

```
$ cat /proc/cmdline
```

5.4. 安装单个受管集群

您可以使用辅助服务和 Red Hat Advanced Cluster Management (RHACM) 手动部署单个受管集群。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了基板管理控制器(BMC) Secret 和镜像 pull-secret Secret 自定义资源 (CR)。详情请参阅"创建受管裸机主机 secret"。
- 您的目标裸机主机满足受管集群的网络和硬件要求。

流程

1. 为要部署的每个特定集群版本创建一个 ClusterImageSet，如 clusterImageSet-4.16.yaml。ClusterImageSet 具有以下格式：

```
apiVersion: hive.openshift.io/v1
kind: ClusterImageSet
metadata:
  name: openshift-4.16.0 (1)
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.16.0-x86_64 (2)
```

(1) 要部署的描述性版本。

(2) 指定要部署并决定操作系统镜像的 releaseImage 版本。发现 ISO 基于由 releaseImage 设置的镜像版本，如果准确版本不可用，则为最新版本。

2. 应用 clusterImageSet CR：

```
$ oc apply -f clusterImageSet-4.16.yaml
```

3. 在 cluster-namespace.yaml 文件中创建 Namespace CR：

```
apiVersion: v1
kind: Namespace
metadata:
  name: <cluster_name> 1
  labels:
    name: <cluster_name> 2
```

1 2 要置备的受管集群的名称。

4. 运行以下命令来应用 Namespace CR：

```
$ oc apply -f cluster-namespace.yaml
```

5. 应用从 ztp-site-generate 容器中提取的生成的 day-0 CR，并自定义以满足您的要求：

```
$ oc apply -R ./site-install/site-sno-1
```

其他资源

- 受管集群网络的连接先决条件
- 在单节点 OpenShift 集群上部署 LVM 存储
- 使用 PolicyGenerator CR 配置 LVM 存储

5.5. 监控受管集群安装状态

通过检查集群状态，确保集群置备成功。

先决条件

- 所有自定义资源都已配置并置备，在受管集群的 hub 上创建 Agent 自定义资源。

流程

1. 检查受管集群的状态：

```
$ oc get managedcluster
```

True 表示受管集群已就绪。

2. 检查代理状态：

```
$ oc get agent -n <cluster_name>
```

3. 使用 describe 命令，提供代理条件的深入描述。支持的状态包括 BackendError、InputError、ValidationsFailing、InFailed 和 AgentIsConnected。这些状态与 Agent 和 AgentClusterInstall 自定义资源相关。

```
$ oc describe agent -n <cluster_name>
```

4. 检查集群置备状态：

```
$ oc get agentclusterinstall -n <cluster_name>
```

5. 使用 describe 命令提供集群置备状态的深入描述：

```
$ oc describe agentclusterinstall -n <cluster_name>
```

6. 检查受管集群的附加服务的状态：

```
$ oc get managedclusteraddon -n <cluster_name>
```

7. 检索受管集群的 kubeconfig 文件的身份验证信息：

```
$ oc get secret -n <cluster_name> <cluster_name>-admin-kubeconfig -o jsonpath=
{.data.kubeconfig} | base64 -d > <directory>/<cluster_name>-kubeconfig
```

5.6. 受管集群故障排除

使用这个流程诊断受管集群中可能出现的任何安装问题。

流程

1. 检查受管集群的状态：

```
$ oc get managedcluster
```

输出示例

```
NAME          HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE
AGE
SNO-cluster   true                                  True     True     2d19h
```

如果 AVAILABLE 列中的状态为 True，受管集群由 hub 管理。

如果 AVAILABLE 列中的状态为 Unknown，则受管集群不会由 hub 管理。使用以下步骤继续检查 以了解更多信息。

2. 检查 AgentClusterInstall 安装状态：

```
$ oc get clusterdeployment -n <cluster_name>
```

输出示例

```
NAME      PLATFORM          REGION   CLUSTERTYPE   INSTALLED   INFRAID
VERSION   POWERSTATE AGE
Sno0026   agent-baremetal                 false                 Initialized
2d14h
```

如果 INSTALLED 列中的状态为 false，则安装会失败。

3. 如果安装失败，请输入以下命令查看 AgentClusterInstall 资源的状态：

```
$ oc describe agentclusterinstall -n <cluster_name> <cluster_name>
```

4. 解决错误并重置集群：

   a. 删除集群的受管集群资源：

   ```
   $ oc delete managedcluster <cluster_name>
   ```

   b. 删除集群的命名空间：

   ```
   $ oc delete namespace <cluster_name>
   ```

   这会删除为此集群创建的所有命名空间范围自定义资源。您必须等待 ManagedCluster CR 删除完成，然后才能继续。

   c. 为受管集群重新创建自定义资源。

5.7. RHACM 生成的集群安装 CR 参考

Red Hat Advanced Cluster Management (RHACM)支持在每个站点的 SiteConfig CR 上部署 OpenShift Container Platform，以及带有特定安装自定义资源 (CR) 的 OpenShift Container Platform。

注意

每个受管集群都有自己的命名空间，除 ManagedCluster 和 ClusterImageSet 以外的所有安装 CR 都位于该命名空间中。ManagedCluster 和 ClusterImageSet 是集群范围的，而不是命名空间范围的。命名空间和 CR 名称与集群名称匹配。

下表列出了在使用您配置的 SiteConfig CR 安装集群时 RHACM 辅助服务自动应用的安装 CR。

表 5.1. 由 RHACM 生成的集群安装 CR

CR	描述	使用方法
BareMetalHost	包含目标裸机主机 Baseboard Management Controller(BMC)的连接信息。	提供对 BMC 的访问，以使用 Redfish 协议在目标服务器上加载和启动发现镜像。
InfraEnv	包含在目标裸机主机上安装 OpenShift Container Platform 的信息。	与 **ClusterDeployment** 一起使用，为受管集群生成发现 ISO。
AgentClusterInstall	指定管理集群配置的详情，如网络和 control plane 节点的数量。安装完成后，显示集群 **kubeconfig** 和凭证。	指定受管集群配置信息，并在安装集群期间提供状态。
ClusterDeployment	引用要使用的 **AgentClusterInstall** CR。	与 **InfraEnv** 一起使用，为受管集群生成发现 ISO。
NMStateConfig	提供网络配置信息，如 **MAC** 地址到 **IP** 映射、DNS 服务器、默认路由和其他网络设置。	为受管集群的 Kube API 服务器设置静态 IP 地址。
Agent	包含有关目标裸机主机的硬件信息。	当目标机器的发现镜像引导时，在 hub 上自动创建。
ManagedCluster	当集群由 hub 管理时，必须导入并已知的集群。此 Kubernetes 对象提供该接口。	hub 使用这个资源来管理和显示受管集群的状态。
KlusterletAddonConfig	包含要部署到 **ManagedCluster** 资源的 hub 提供的服务列表。	告知 hub 部署到 **ManagedCluster** 资源的附加组件服务。
Namespace	hub 上已存在的 **ManagedCluster** 资源的逻辑空间。每个站点都是唯一的。	将资源传播到 **ManagedCluster**。

CR	描述	使用方法
Secret	创建两个 CR：**BMC Secret** 和 **Image Pull Secret**。	• **BMC Secret** 使用其用户名和密码向目标裸机主机进行身份验证。 • **Image Pull Secret** 包含目标裸机主机中安装的 OpenShift Container Platform 镜像的身份验证信息。
ClusterImageSet	包含 OpenShift Container Platform 镜像信息，如存储库和镜像名称。	传递给资源以提供 OpenShift Container Platform 镜像。

第 6 章 推荐的 VDU 应用程序工作负载的单节点 OPENSHIFT 集群配置

使用以下引用信息，了解在集群中部署虚拟分布式单元 (vDU) 应用程序所需的单节点 OpenShift 配置。配置包括用于高性能工作负载的集群优化、启用工作负载分区以及最大程度减少安装后所需的重启数量。

其他资源

- 要手动部署单个集群，请参阅使用 GitOps ZTP 手动安装单节点 OpenShift 集群。
- 要使用 GitOps Zero Touch Provisioning (ZTP) 部署集群集合，请参阅使用 ZTP 部署边缘站点。

6.1. 在 OPENSHIFT CONTAINER PLATFORM 上运行低延迟应用程序

OpenShift Container Platform 通过使用几个技术和专用硬件设备，为在商业现成 (COTS) 硬件上运行的应用程序启用低延迟处理：

RHCOS 的实时内核

确保以高度的进程确定性处理工作负载。

CPU 隔离

避免 CPU 调度延迟并确保 CPU 容量一致可用。

NUMA 感知拓扑管理

将内存和巨页与 CPU 和 PCI 设备对齐，以将容器内存和巨页固定到非统一内存访问(NUMA)节点。所有服务质量 (QoS) 类的 Pod 资源保留在同一个 NUMA 节点上。这可降低延迟并提高节点的性能。

巨页内存管理

使用巨页大小可减少访问页表所需的系统资源量，从而提高系统性能。

使用 PTP 进行精确计时同步

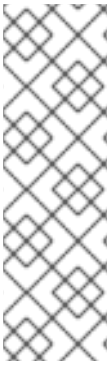
允许以子微秒的准确性在网络中的节点之间进行同步。

6.2. VDU 应用程序工作负载的推荐集群主机要求

运行 vDU 应用程序工作负载需要一个具有足够资源的裸机主机来运行 OpenShift Container Platform 服务和生产工作负载。

表 6.1. 最低资源要求

profile	vCPU	memory	Storage
最小值	4 到 8 个 vCPU 内核	32GB RAM	120GB



注意

当未启用并发多线程 (SMT) 或超线程时，一个 vCPU 相当于一个物理内核。启用后，使用以下公式来计算对应的比率：

- （每个内核的线程数 x 内核数）x 插槽数 = vCPU

重要

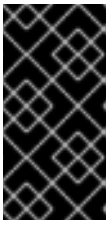
使用虚拟介质引导时，服务器必须具有基板管理控制器(BMC)。

6.3. 为低延迟和高性能配置主机固件

裸机主机需要在置备主机前配置固件。固件配置取决于您的特定硬件和安装的具体要求。

流程

1. 将 UEFI/BIOS Boot Mode 设置为 UEFI。
2. 在主机引导顺序中，设置 Hard drive first。
3. 为您的硬件应用特定的固件配置。下表描述了 Intel Xeon Skylake 服务器和更新的硬件生成的代表固件配置，具体取决于 Intel FlexRAN 4G 和 5G 基带 PHY 参考设计。



重要

确切的固件配置取决于您的特定硬件和网络要求。以下示例配置仅用于说明目的。

表 6.2. 固件配置示例

固件设置	配置
CPU Power 和性能策略	性能
非核心频率扩展	Disabled
性能限制	Disabled
增强的 Intel SpeedStep ® Tech	Enabled
Intel 配置的 TDP	Enabled
可配置 TDP 级别	2 级
Intel® Turbo Boost Technology	Enabled
节能 Turbo	Disabled
硬件 P-State	Disabled
软件包 C-State	C0/C1 状态
C1E	Disabled
处理器 C6	Disabled

注意

在主机的固件中启用全局 SR-IOV 和 VT-d 设置。这些设置与裸机环境相关。

6.4. 受管集群网络的连接先决条件

在安装并置备带有 GitOps Zero Touch Provisioning (ZTP) 管道的受管集群前，受管集群主机必须满足以下网络先决条件：

-

hub 集群中的 GitOps ZTP 容器和目标裸机主机的 Baseboard Management Controller (BMC) 之间必须有双向连接。

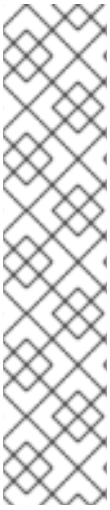
- 受管集群必须能够解析和访问 hub 主机名和 *.apps 主机名的 API 主机名。以下是 hub 和 *.apps 主机名的 API 主机名示例：
  - api.hub-cluster.internal.domain.com
  - console-openshift-console.apps.hub-cluster.internal.domain.com
- hub 集群必须能够解析并访问受管集群的 API 和 *.app 主机名。以下是受管集群的 API 主机名和 *.apps 主机名示例：
  - api.sno-managed-cluster-1.internal.domain.com
  - console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com

6.5. 使用 GITOPS ZTP 在单节点 OPENSHIFT 中的工作负载分区

工作负载分区配置 OpenShift Container Platform 服务、集群管理工作负载和基础架构 pod，以便在保留数量的主机 CPU 上运行。

要使用 GitOps Zero Touch Provisioning (ZTP)配置工作负载分区，您可以在用于安装集群的 SiteConfig 自定义资源(CR)中配置 cpuPartitioningMode 字段，并应用在主机上配置 isolated 和 reserved CPU 的 PerformanceProfile CR。

配置 SiteConfig CR 在集群安装过程中启用工作负载分区，并应用 PerformanceProfile CR 将 CPU 的特定分配配置为保留和隔离的集合。这两个步骤在集群置备过程中的不同点发生。



注意

使用 SiteConfig CR 中的 cpuPartitioningMode 字段配置工作负载分区是 OpenShift Container Platform 4.13 中的技术预览功能。

另外，您可以使用 SiteConfig 自定义资源(CR)的 cpuset 字段指定集群管理 CPU 资源，以及组 PolicyGenerator 或 PolicyGentemplate CR 的 reserved 字段。GitOps ZTP 管道使用这些值来填充工作负载分区 MachineConfig CR (cpuset) 和配置单节点 OpenShift 集群的 PerformanceProfile CR (reserved)中的所需字段。这个方法是 OpenShift Container Platform 4.14 中的正式发行(GA)。

工作负载分区配置将 OpenShift Container Platform 基础架构 pod 固定到 reserved CPU 集。systemd、CRI-O 和 kubelet 等平台服务在 reserved CPU 集中运行。isolated CPU 集只分配给容器工作负载。隔离 CPU 可确保工作负载保证对指定 CPU 的访问，而不会与同一节点上运行的其他应用程序竞争。所有不是隔离的 CPU 都应保留。

重要

确保 reserved 和 isolated CPU 集不会相互重叠。

其他资源

- 有关推荐的单节点 OpenShift 工作负载分区配置，请参阅 Workload partitioning。

6.6. 推荐的集群安装清单

ZTP 管道在集群安装过程中应用以下自定义资源 (CR)。这些配置 CR 确保集群满足运行 vDU 应用程序所需的功能和性能要求。

注意

当将 GitOps ZTP 插件和 SiteConfig CR 用于集群部署时，默认包含以下 MachineConfig CR。

使用 SiteConfig extraManifests 过滤器更改默认包括的 CR。如需更多信息，请参阅使用 SiteConfig CR 的高级受管集群配置。

6.6.1. 工作负载分区

运行 DU 工作负载的单节点 OpenShift 集群需要工作负载分区。这限制了运行平台服务的内核数，从而最大程度提高应用程序有效负载的 CPU 内核。

注意

工作负载分区只能在集群安装过程中启用。您不能在安装后禁用工作负载分区。但是，您可以通过 PerformanceProfile CR 更改分配给隔离和保留集的 CPU 集合。更改 CPU 设置会导致节点重新引导。

从 OPENSHIFT CONTAINER PLATFORM 4.12 升级到 4.13+

当使用 cpuPartitioningMode 启用工作负载分区时，从用来置备集群的 /extra-manifest 文件夹中删除工作负载分区 MachineConfig CR。

工作负载分区的建议 SiteConfig CR 配置

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "<site_name>"
  namespace: "<site_name>"
spec:
  baseDomain: "example.com"
  cpuPartitioningMode: AllNodes 1
```

1

将 cpuPartitioningMode 字段设置为 AllNodes，为集群中的所有节点配置工作负载分区。

验证

检查应用程序和集群系统 CPU 固定是否正确。运行以下命令：

1. 为受管集群打开远程 shell 提示符：

```
$ oc debug node/example-sno-1
```

2. 检查 OpenShift 基础架构应用程序 CPU 固定是否正确：

```
sh-4.4# pgrep ovn | while read i; do taskset -cp $i; done
```

输出示例

```
pid 8481's current affinity list: 0-1,52-53
pid 8726's current affinity list: 0-1,52-53
pid 9088's current affinity list: 0-1,52-53
pid 9945's current affinity list: 0-1,52-53
pid 10387's current affinity list: 0-1,52-53
pid 12123's current affinity list: 0-1,52-53
pid 13313's current affinity list: 0-1,52-53
```

3. 检查系统应用程序 CPU 固定是否正确：

```
sh-4.4# pgrep systemd | while read i; do taskset -cp $i; done
```

输出示例

```
pid 1's current affinity list: 0-1,52-53
pid 938's current affinity list: 0-1,52-53
pid 962's current affinity list: 0-1,52-53
pid 1197's current affinity list: 0-1,52-53
```

6.6.2. 减少平台管理占用空间

要减少平台的整体管理空间，需要一个 MachineConfig 自定义资源 (CR)，它将所有特定于 Kubernetes 的挂载点放在独立于主机操作系统的新命名空间中。以下 base64 编码的示例 MachineConfig CR 演示了此配置。

推荐的容器挂载命名空间配置 (01-container-mount-ns-and-kubelet-conf-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKCmRlYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkge
wogIGVjaG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDAplFVOSVQgW2VudmZpbGUgW3Zhcm5hb
WVdXQogIGVjaG8KICBlY2hvIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeG
VjU3RhcnQgc3Rhbnphe IGZyb20gdGhlIGdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuI
G l0lHRvlHN0ZG91dAogIGVjaG8KICBlY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1d
CBpdCBpbiB0aGVyZSBpbnN0ZWFkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1
lZCAndmFybmFtZSciCiAgZWNobyAiRGVmYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUI
GlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxCn0KClVOSVQ9JDEKRU5WRklMRT0kMgpWQVJ
OQU1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUID09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCI
gXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cmFjdGluZyBFeGVjU3RhcnQgZnJvbSA
kVU5JVCIKRklMRT0kHN5c3RlbWN0bCBjYXQgJFVOSVQgfCBoZWFkIC1uIDEpCkZJTEU9JHt
GSUxFI1wjIH0KaWYgW1sgISAtZiAkRklMRSBdXTsgdGhlbgogIGRlYnVnICJGYWlsZWQgdG8gZ
mluZCByb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSIKICBleGl0CmZpCmRlYnVnI
CJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0kHNlZCAtbiAtZS
AnL15FeGVjU3RhcnQ9LipcXCQvLC9bXlxcXSQvIHsgcy9eRXhlY1N0YXJ0PS8vOyBwIH0nIC1l
CcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHAgfSBpCgp
pZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBU
lR9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogI
GVjaG8gJEVYRUNTVEFSVApmaQo=
          mode: 493
          path: /usr/local/bin/extractExecStart
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50L
W5hbWVzcGFjZS9tbnQgliRAlgo=
          mode: 493
          path: /usr/local/bin/nsenterCmns
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Manages a mount namespace that both kubelet and crio can use to share
their container-specific mounts

            [Service]
            Type=oneshot
            RemainAfterExit=yes
            RuntimeDirectory=container-mount-namespace
            Environment=RUNTIME_DIRECTORY=%t/container-mount-namespace
```

```
          Environment=BIND_POINT=%t/container-mount-namespace/mnt
          ExecStartPre=bash -c "findmnt ${RUNTIME_DIRECTORY} || mount --make-unbindable --bind ${RUNTIME_DIRECTORY} ${RUNTIME_DIRECTORY}"
          ExecStartPre=touch ${BIND_POINT}
          ExecStart=unshare --mount=${BIND_POINT} --propagation slave mount --make-rshared /
          ExecStop=umount -R ${RUNTIME_DIRECTORY}
        name: container-mount-namespace.service
      - dropins:
        - contents: |
            [Unit]
            Wants=container-mount-namespace.service
            After=container-mount-namespace.service

            [Service]
            ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
            EnvironmentFile=-/%t/%N-execstart.env
            ExecStart=
            ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
              ${ORIG_EXECSTART}"
          name: 90-container-mount-namespace.conf
        name: crio.service
      - dropins:
        - contents: |
            [Unit]
            Wants=container-mount-namespace.service
            After=container-mount-namespace.service

            [Service]
            ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
            EnvironmentFile=-/%t/%N-execstart.env
            ExecStart=
            ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
              ${ORIG_EXECSTART} --housekeeping-interval=30s"
          name: 90-container-mount-namespace.conf
        - contents: |
            [Service]
            Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
            Environment="OPENSHIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
          name: 30-kubelet-interval-tuning.conf
        name: kubelet.service
```

6.6.3. SCTP

流控制传输协议 (SCTP) 是在 RAN 应用程序中使用的密钥协议。此 MachineConfig 对象向节点添加 SCTP 内核模块以启用此协议。

推荐的 control plane 节点 SCTP 配置 (03-sctp-machine-config-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module-master
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf
```

推荐的 worker 节点 SCTP 配置 (03-sctp-machine-config-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: load-sctp-module-worker
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
```

```
        filesystem: root
        mode: 420
        path: /etc/modules-load.d/sctp-load.conf
```

6.6.4. 设置 rcu_normal

以下 MachineConfig CR 将系统配置为在系统完成启动后将 rcu_normal 设置为 1。这提高了 vDU 应用程序的内核延迟。

在节点启动后禁用 rcu_expedited 的推荐配置 (08-set-rcu-normal-master.yaml)


```yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 08-set-rcu-normal-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKIwojIERpc2FibGUgcmN1X2V4cGVkaXRlZCBhZnRlciBub2RlIGhhc
yBmaW5pc2hlZCBib290aW5nCiMKIyBUaGUgZGVmYXVsdHMgYmVsb3cgY2FuIGJlIG92ZXJya
WRkZW4gdmlhIGVudmlyb25tZW50IHZhcmlhYmxlcwojCgojIERlZmF1bHQgd2FpdCB0aW1lIGlzI
DYwMHMgPSAxMG06Ck1BWElNVU1fV0FJVF9USU1FPSR7TUFYSU1VTV9XQUlUX1RJTUU6LT
YwMH0KCiMgRGVmYXVsdCBzdGVhZHktc3RhdGUgdGhyZXNob2xkID0gMiUKIyBBbGxvd2VkI
HZhbHVlczoKIyAgNCAgLSBhYnNvbHV0ZSBwb2QgY291bnQgKCsvLSkKIyAgNCUgLSBwZXJj
ZW50IGNoYW5nZSAoKy8tKQojICAtMSAtIGRpc2FibGUgdGhlIHN0ZWFkeS1zdGF0ZSBjaGVjaw
pTVEVBRFlfU1RBVEVfVEhSRVNIT0xEPSR7U1RFQURZX1NUQVRFX1RIUkVTSE9MRDotMiV9C
gojIERlZmF1bHQgc3RlYWR5LXN0YXRlIHdpbmRvdyA9IDYwcwojIElmIHRoZSBydW5uaW5nIH
BvZCBjb3VudCBzdGF5cyB3aXRoaW4gdGhlIGdpdmVuIHRocmVzaG9sZCBmb3IgdGhpcyB0a
W1lCiMgcGVyaW9kLCByZXR1cm4gQ1BVIHV0aWxpemF0aW9uIHRvIG5vcm1hbCBiZWZvcmU
gdGhlIG1heGltdW0gd2FpdCB0aW1lIGhhcwojIGV4cGlyZXMKU1RFQURZX1NUQVRFX1dJTkRP
Vz0ke1NURUFEWV9TVEFURV9XSU5ET1c6LTYwfQoKIyBEZWZhdWx0IHN0ZWFkeS1zdGF0ZS
BhbGxvd3MgYW55IHBvZCBjb3VudCB0byBiZSAic3RlYWR5IHN0YXRlIgojIEluY3JlYXNpbmcgd
GhpcyB3aWxsIHNraXAgYW55IHN0ZWFkeS1zdGF0ZSBjaGVja3MgdW50aWwgdGhlIGNvdW50I
HJpc2VzIGFib3ZlCiMgdGhpcyBudW1iZXIgdG8gYXZvaWQgZmFsc2UgcG9zaXRpdmVzIGlmIHR
oZXJlIGFyZSBzb21lIHBlcmlvZHMgd2hlcmUgdGhlCiMgY291bnQgZG9lc24ndCBpbmNyZWFzZS
BidXQgd2Uga25vdyB3ZSBjYW4ndCBiZSBhdCBzdGVhZHktc3RhdGUgeWV0LgpTVEVBRFlfU1
RBVEVfTUlOSU1VTT0ke1NURUFEWV9TVEFURV9NSU5JTVVNOi0wfQoKIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIwoKd2l0aGluKCkgewogIGxvY2FsIGxhc3Q
9JDEgY3VycmVudD0kMiB0aHJlc2hvbGQ9JDMKICBsb2NhbCBkZWx0YT0wIHBjaGFuZ2UKICB
kZWx0YT0kKCggY3VycmVudCAtIGxhc3QgKSkKICBpZiBbWyAkY3VycmVudCAtZXEgJGxhc3
```

```
QgXV07IHRoZW4KICAgIHBjaGFuZ2U9MAogIGVsaWYgW1sgJGxhc3QgLWVxIDAgXV07IHRoZ
W4KICAgIHBjaGFuZ2U9MTAwMDAwMAogIGVsc2UKICAgIHBjaGFuZ2U9JCgoICggIiRkZWx0Y
SIgKiAxMDApIC8gbGFzdCApKQogIGZpCiAgZWNobyAtbiAibGFzdDokbGFzdCBjdXJyZW50Oi
RjdXJyZW50IGRlbHRhOiRkZWx0YSBwY2hhbmdlOiR7cGNoYW5nZX0lOiAiCiAgbG9jYWwgYW
Jzb2x1dGUgbGltaXQKICBjYXNlICR0aHJlc2hvbGQgaW4KICAgICoKQogICAgICBhYnNvbHV0Z
T0ke3BjaGFuZ2Ujly19ICMgYWJzb2x1dGUgdmFsdWUKICAgICAgbGltaXQ9JHt0aHJlc2hvbGQl
JSV9CiAgICAgIDs7CiAgICAqKQogICAgICBhYnNvbHV0ZT0ke2RlbHRhlyMtfSAjIGFic29sdXRlI
HZhbHVlCiAgICAgIGxpbWl0PSR0aHJlc2hvbGQKICAgICAgOzsKICBlc2FjCiAgaWYgW1sgJGFi
c29sdXRlIC1sZSAkbGltaXQgXV07IHRoZW4KICAgICAgVjaG8gIndpdGhpbiAoKy8tKSR0aHJlc2hv
bGQiCiAgICBmV0dXJuIDEKICBmaQp9CgpzdGVhZHlzdGF0ZSgpIHsKICBsb2NhbCBsYXN0PSQx
IGN1cnJlbnQ9JDIKICBpZiBbbWyAkbGFzdCAtbHQgJFNURUFEWV9TVEFURV9NSU5JTVVN
IF1dOyB0aGVuCiAgICBlY2hvICJsYXN0OiRsYXN0IGN1cnJlbnQ6JGN1cnJlbnQgV2FpdGluZyB
0byByZWFjaCAkU1RFQURZX1NUQVRFX01JTklNVU0gYmVmb3JlIGNoZWNraW5nIGFjciBzdG
VhZHktc3RhdGUiCiAgICBzZXR1cm4gMQogIGZpCiAgd2l0aGluICIkbGFzdCIgIiRjdXJyZW50IiAi
JFNURUFEWV9TVEFURV9USFJFU0hPTEQiCn0KCndhaXRGb3JSZWFkeSgpIHsKICBsb2dnZXI
gIlJlY292ZXJ5OiBXYWl0aW5nICR7TUFYSU1VTV9XQUlUX1RJTUV9cyBmb3IgdGhlIGluaXRpYW
xpemF0aW9uIHRvIGNvbXBsZXRlIgogIGd2l0aGluICIkbGFzdCIgIiRjdXJyZW50IiAi
JFNURUFEWV9TVEFURV9USFJFU0hPTEQiCn0KCndhaXRGb3JSZWFkeSgpIHsKICBsb2dnZXI
glIJlY292ZXJ5OiBXYWl0aW5nICR7TUFYSU1VTV9XQUlUX1RJTUV9cyBmb3IgdGhlIGluaXRpYW
xpemF0aW9uIHRvIGNvbXBsZXRlIgogIGd2l0aGluICIkbGFzdCIgIiRjdXJyZW50IiAi
IERldGVjdCBzdGVhZHktc3RhdGUgcG9kIGNvdW50CiAgICBjY291bnQ9JChjcmljdGwgcHMgMj4
vZGV2L251bGwgfCB3YyAtbCkKICAgIGlmIFtbICRjY291bnQgLWd0IDAgXV0gJiYgc3RlYWR5c3
RhdGUgIiRsYXN0Q2NvdW50IiAiJGNjb3VudCI7IHRoZW4KICAgICAgKChzdGVhZHlTdGF0ZVRp
bWUgKz0gcykpKQogICAgICBsYXN0PSRjdXJyZW50CiAgICBlbHNlCiAgICAgIGlIN0ZWFkeS1zdGF0ZSBmb3lgJHtzdGVhZHlTdGF0ZVRpbW
V9cy8ke1NURUFEWV9TVEFURV9XSU5ET1d9cylKICAgICAgaWYgW1sgJHN0ZWFkeVN0YXRlV
GltZSAtZ2UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICAgICAgICBsb2dnZXIgIlJ
lY292ZXJ5OiBTdGVhZHktc3RhdGUgKCsvLSAkU1RFQURZX1NUQVRFX1RlUkVTE9MRCkgZ
m9yICR7U1RFQURZX1NUQVRFX1dJTkRPV31zOiBEb25lIgogICAgICAgIGlGZpCiAgICBlbHNlCiAgICAgIGlGZpCiBlbHNlCiAgICAgIGlGZpCiAgICBlbHNlCiAgICAgIGlGZpCiAgICBlY2hvICJSZXNldHRpbmcgc3RlYWR5LXN0YXRlIHRpbWVvylgogICAgICAgIGN0Z
WFkeVN0YXRlVGltZT0wCiAgICAgIGZpCiAgICBmaQogICAgbGFzdENjb3VudD0kY2NvdW50CiA
gZG9uZQogIGxvZ2dlciAiUmVjb3Zlcnk6IFJlY292ZXJ5IENvbXBsZXRlIgp9CgpkZWZhdWx0X
dFJjdU5vcm1hbCgpIHsKICBlY2hvICJTZXR0aW5nIHJjdV9ub3JtYWwgdG8gMSIKICBlY2hvIDEg
PiAvc3lzL2tlcm5lbC9yY3Vfbm9ybWFsCn0KCmltaW4oKSB7CiAgd2FpdEZvclJlYWR5CiAgZW
NobyAiV2FpdGluZyBmb3Igc3RlYWR5IHN0YXRlIHRvb2s6ICQoYXdrICd7cHJpbnQgaW50KCQx
LzM2MDApImgiLCBpbnQoKCQxJTM2MDApLzYwKSJtIiwgaW50KCQxJTYwKSJzIn0nIC9wcm9j
L3VwdGltZSkiCiAgc2V0UmN1Tm9ybWFsCn0KCmlmIFtbICIke0JBU0hfU09VUkNFWzBdfSIgPS
AiJHswfSIgXV07IHRoZW4KICBtYWluICIke0B9IgogIGV4aXQgJD8KZmkK
        mode: 493
        path: /usr/local/bin/set-rcu-normal.sh
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Disable rcu_expedited after node has finished booting by setting
rcu_normal to 1

            [Service]
            Type=simple
            ExecStart=/usr/local/bin/set-rcu-normal.sh

            # Maximum wait time is 600s = 10m:
            Environment=MAXIMUM_WAIT_TIME=600

            # Steady-state threshold = 2%
            # Allowed values:
```

```
        #  4  - absolute pod count (+/-)
        #  4% - percent change (+/-)
        #  -1 - disable the steady-state check
        # Note: '%' must be escaped as '%%' in systemd unit files
        Environment=STEADY_STATE_THRESHOLD=2%%

        # Steady-state window = 120s
        # If the running pod count stays within the given threshold for this time
        # period, return CPU utilization to normal before the maximum wait time has
        # expires
        Environment=STEADY_STATE_WINDOW=120

        # Steady-state minimum = 40
        # Increasing this will skip any steady-state checks until the count rises above
        # this number to avoid false positives if there are some periods where the
        # count doesn't increase but we know we can't be at steady-state yet.
        Environment=STEADY_STATE_MINIMUM=40

        [Install]
        WantedBy=multi-user.target
      enabled: true
      name: set-rcu-normal.service
```

6.6.5. 使用 kdump 自动内核崩溃转储

kdump 是一个 Linux 内核功能，可在内核崩溃时创建内核崩溃转储。kdump 使用以下 MachineConfig CR 启用。

推荐的 MachineConfig CR 从 control plane kdump 日志中删除 ice 驱动程序 (05-kdump-config-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 05-kdump-config-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
```

```
          [Unit]
          Description=Remove ice module when doing kdump
          Before=kdump.service
          [Service]
          Type=oneshot
          RemainAfterExit=true
          ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
          [Install]
          WantedBy=multi-user.target
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2U
gbW9kdWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWlul
HNlcnZlcnMuCiMgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0
xMzgyMzYgYW5kIGNhbiBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgo
Kc2V0IC14CgpTRUQ9li91c3lvYmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZG
UgZm9yIHRlc3RpbmcgcHVycG9zZXMKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9r
ZHVtcH0iCIJFTU9WRV9JQ0VfU1RSPSJtb2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZi
BmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mICR7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCi
MgZXhpdCBpZiBmaWxlIGFscmVhZHkgdXBkYXRlZAoke0dSRVB9IC1GcSAke1JFTU9WRV9JQ
0VfU1RSfSAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAoKIyBUaGlzIGlzIGEgdGVtcG9yYXJ5IH
cl9jcHVzPTEgLi4uIGhlc3RfZGlzYWJsZSlKIyBVc2Ugc2VkIHRvIG1hdGNoIGV2ZXJ5dGhpbmcg
YmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFJFTU9WRV9JQ0VfU1RSIHRvIGl0Ci
R7U0VEfSAtaSAncy9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPSJbXiJdKi8mICcke1JFTU9
WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
          mode: 448
          path: /usr/local/bin/kdump-remove-ice-module.sh
```

推荐的 control plane 节点 kdump 配置 (06-kdump-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 06-kdump-enable-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

推荐的 MachineConfig CR 从 worker 节点 kdump 日志中删除 ice 驱动程序 (05-kdump-config-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-kdump-config-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2U
gbW9kdWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWlul
HNlcnZlcnMuCiMgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0
xMzgyMzYgYW5kIGNhbiBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgo
Kc2V0IC14CgpTRUQ9li91c3lvYmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZG
UgZm9yIHRlc3RpbmcgcHVycG9zZXMKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9r
ZHVtcH0iCIJFTU9WRV9JQ0VfU1RSPSJtb2R1bGVfYmxhY2tsaXN0PWlje3NlZCIKCiMgZXhpdCBpZi
BmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mICR7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCi
MgZXhpdCBpZiBmaWxlIGFscmVhZHkgdXBkYXRlZAoke0dSRVB9IC1GcSAke1JFTU9WRV9JQ
0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKIyBUYXJnZXQgc2VkIHRvIG1hdGNoIGV2ZXJ5dGhpbmcg
YmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFFJFTU9WRV9JQ0VfU1RSIHRvIGl0Ci
R7U0VEfSAtaSAncy9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPSJbXiJdKi8mICcke1JFTU9
WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
          mode: 448
          path: /usr/local/bin/kdump-remove-ice-module.sh
```

推荐的 kdump worker 节点配置 (06-kdump-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 06-kdump-enable-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

6.6.6. 禁用自动 CRI-O 缓存擦除

在不受控制的主机关闭或集群重启后，CRI-O 会自动删除整个 CRI-O 缓存，从而导致在节点重启时从 registry 中拉取所有镜像。这可能导致不可接受的恢复时间或者恢复失败。要防止这会在使用 GitOps ZTP 安装的单节点 OpenShift 集群中发生，请在集群安装过程中禁用 CRI-O 删除缓存功能。

推荐的 MachineConfig CR 在 control plane 节点上禁用 CRI-O 缓存擦除 (99-crio-disable-wipe-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-crio-disable-wipe-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
```

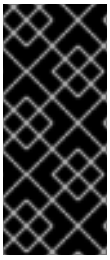
```
    files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAiIgo=
        mode: 420
        path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

推荐的 MachineConfig CR 在 worker 节点上禁用 CRI-O 缓存擦除 (99-crio-disable-wipe-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-crio-disable-wipe-worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAiIgo=
          mode: 420
          path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

6.6.7. 将 crun 配置为默认容器运行时

以下 ContainerRuntimeConfig 自定义资源 (CR) 将 crun 配置为 control plane 和 worker 节点的默认 OCI 容器运行时。crun 容器运行时快速且轻量级，内存占用较低。



重要

为获得最佳性能，请在单节点 OpenShift、三节点 OpenShift 和标准集群中为 control plane 和 worker 节点启用 crun。要避免在应用 CR 时重启集群，请将更改作为 GitOps ZTP 额外日期 0 安装清单应用。

control plane 节点推荐的 ContainerRuntimeConfig CR (enable-crun-master.yaml)

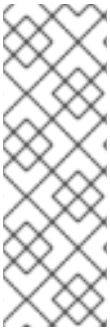
```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: enable-crun-master
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/master: ""
  containerRuntimeConfig:
    defaultRuntime: crun
```

worker 节点的推荐 ContainerRuntimeConfig CR (enable-crun-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: enable-crun-worker
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  containerRuntimeConfig:
    defaultRuntime: crun
```

6.7. 推荐的安装后集群配置

当集群安装完成后，ZTP 管道会应用运行 DU 工作负载所需的以下自定义资源 (CR)。



注意

在 GitOps ZTP v4.10 及更早版本中，您可以使用 MachineConfig CR 配置 UEFI 安全引导。GitOps ZTP v4.11 及更新的版本中不再需要。在 v4.11 中，您可以通过更新用于安装集群的 SiteConfig CR 中的 spec.clusters.nodes.bootMode 字段来为单节点 OpenShift 集群配置 UEFI 安全引导。如需更多信息，请参阅使用 SiteConfig 和 GitOps ZTP 部署受管集群。

6.7.1. Operator

运行 DU 工作负载的单节点 OpenShift 集群需要安装以下 Operator：

- Local Storage Operator
- Logging Operator
- PTP Operator
- Cluster Network Operator

您还需要配置自定义 CatalogSource CR，禁用默认的 OperatorHub 配置，并配置可从您安装的集群访问的 ImageContentSourcePolicy 镜像 registry。

推荐的 Storage Operator 命名空间和 Operator 组配置 (StorageNS.yaml,StorageOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-local-storage
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-local-storage
  namespace: openshift-local-storage
  annotations: {}
spec:
  targetNamespaces:
    - openshift-local-storage
```

推荐的 Cluster Logging Operator 命名空间和 Operator 组配置 (ClusterLogNS.yaml,ClusterLogOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations: {}
spec:
  targetNamespaces:
    - openshift-logging
```

推荐的 PTP Operator 命名空间和 Operator 组配置
(PtpSubscriptionNS.yaml,PtpSubscriptionOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
  annotations: {}
spec:
  targetNamespaces:
    - openshift-ptp
```

推荐的 SR-IOV Operator 命名空间和 Operator 组配置 (SriovSubscriptionNS.yaml,SriovSubscriptionOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  targetNamespaces:
    - openshift-sriov-network-operator
```

推荐的 CatalogSource 配置 (DefaultCatsrc.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: default-cat-source
  namespace: openshift-marketplace
  annotations:
    target.workload.openshift.io/management: '{"effect": "PreferredDuringScheduling"}'
spec:
  displayName: default-cat-source
  image: $imageUrl
  publisher: Red Hat
  sourceType: grpc
  updateStrategy:
    registryPoll:
      interval: 1h
status:
  connectionState:
    lastObservedState: READY
```

推荐的 ImageContentSourcePolicy 配置 (DisconnectedICSP.yaml)

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: disconnected-internal-icsp
  annotations: {}
spec:
  repositoryDigestMirrors:
    - $mirrors
```

推荐的 OperatorHub 配置 (OperatorHub.yaml)

```
apiVersion: config.openshift.io/v1
kind: OperatorHub
metadata:
  name: cluster
  annotations: {}
spec:
  disableAllDefaultSources: true
```

6.7.2. Operator 订阅

运行 DU 工作负载的单节点 OpenShift 集群需要以下 Subscription CR。订阅提供下载以下 Operator 的位置：

- Local Storage Operator
- Logging Operator
- PTP Operator

- Cluster Network Operator
- SRIOV-FEC Operator

对于每个 Operator 订阅，指定要从中获取 Operator 的频道。推荐的频道是 stable。

您可以指定 Manual 或 Automatic 更新。在 Automatic 模式中，Operator 会在 registry 中可用时自动更新到频道中最新版本。在 Manual 模式中，只有在被明确批准时才会安装新的 Operator 版本。

提示

对订阅使用 Manual 模式。这可让您控制 Operator 更新在调度的维护窗口中适合的时间。

推荐的 Local Storage Operator 订阅 (StorageSubscription.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
  annotations: {}
spec:
  channel: "stable"
  name: local-storage-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推荐的 SR-IOV Operator 订阅 (SriovSubscription.yaml)

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

```
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  channel: "stable"
  name: sriov-network-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推荐的 PTP Operator 订阅 (PtpSubscription.yaml)

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
  annotations: {}
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推荐的 Cluster Logging Operator 订阅 (ClusterLogSubscription.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations: {}
spec:
  channel: "stable"
  name: cluster-logging
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
```

```
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

6.7.3. 集群日志记录和日志转发

运行 DU 工作负载的单节点 OpenShift 集群需要日志记录和日志转发以进行调试。需要以下 ClusterLogging 和 ClusterLogForwarder 自定义资源 (CR)。

推荐的集群日志记录配置 (ClusterLogging.yaml)

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
  annotations: {}
spec:
  managementState: "Managed"
  collection:
    logs:
      type: "vector"
```

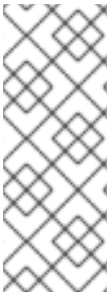
推荐的日志转发配置 (ClusterLogForwarder.yaml)

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
  annotations: {}
spec:
  outputs: $outputs
  pipelines: $pipelines
```

将 spec.outputs.url 字段设置为日志转发到的 Kafka 服务器的 URL。

6.7.4. 性能配置集

运行 DU 工作负载的单节点 OpenShift 集群需要 Node Tuning Operator 性能配置集才能使用实时主机功能和服务。



注意

在早期版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

以下示例 PerformanceProfile CR 演示了所需的单节点 OpenShift 集群配置。

推荐的性能配置集配置 (PerformanceProfile.yaml)

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
  # matches this name: include=openshift-node-performance-
${PerformanceProfile.metadata.name}
  # Also in file 'validatorCRs/informDuValidator.yaml':
  # name: 50-performance-${PerformanceProfile.metadata.name}
  name: openshift-node-performance-profile
  annotations:
    ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "vfio_pci.enable_sriov=1"
    - "vfio_pci.disable_idle_d3=1"
    - "module_blacklist=irdma"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
```

```
      node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  # To use the standard (non-realtime) kernel, set enabled to false
  realTimeKernel:
    enabled: true
  workloadHints:
    # WorkloadHints defines the set of upper level flags for different type of workloads.
    # See https://github.com/openshift/cluster-node-tuning-
operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
    # for detailed descriptions of each item.
    # The configuration below is set for a low latency, performance mode.
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false
```

表 6.3. 单节点 OpenShift 集群的 PerformanceProfile CR 选项

PerformanceProfile CR 字段	描述
metadata.name	确保**名称**与相关 GitOps ZTP 自定义资源(CR)中设置的以下字段匹配： • **TunedPerformancePatch.yaml** 中的 **include=openshift-node-performance-${PerformanceProfile.metadata.name}** • **validatorCRs/informDuValidator.yaml** 中的 **name: 50-performance-${PerformanceProfile.metadata.name}**
spec.additionalKernelArgs	**"efi=runtime"** 为集群主机配置 UEFI 安全引导。
spec.cpu.isolated	设置隔离的 CPU。确保所有 Hyper-Threading 对都匹配。  **重要** 保留和隔离的 CPU 池不得重叠，并且必须一起跨越所有可用的内核。未考虑导致系统中未定义的 CPU 内核。

PerformanceProfile CR 字段	描述
spec.cpu.reserved	设置保留的 CPU。启用工作负载分区时，系统进程、内核线程和系统容器线程仅限于这些 CPU。所有不是隔离的 CPU 都应保留。
spec.hugepages.pages	• 设置巨页数量(**数量**) • 设置巨页大小(**大小**)。 • 将 **node** 设置为 NUMA 节点，它是 **hugepages** 分配的位置 (**node**)
spec.realTimeKernel	将 **enabled** 设置为 **true** 以使用实时内核。
spec.workloadHints	使用 **workloadHints** 为不同类型的工作负载定义顶级标记集合。示例配置为低延迟和高性能配置集群。

6.7.5. 配置集群时间同步

为 **control plane 或 worker 节点运行一次性系统时间同步作业。**

推荐的 control plane 节点一次同步 (99-sync-time-once-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-sync-time-once-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Sync time once
            After=network-online.target
            Wants=network-online.target
            [Service]
            Type=oneshot
            TimeoutStartSec=300
            ExecCondition=/bin/bash -c 'systemctl is-enabled chronyd.service --quiet && exit 1 ||
```

```
exit 0'
          ExecStart=/usr/sbin/chronyd -n -f /etc/chrony.conf -q
          RemainAfterExit=yes
          [Install]
          WantedBy=multi-user.target
        enabled: true
        name: sync-time-once.service
```

推荐的 worker 节点一次同步时间 (99-sync-time-once-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-sync-time-once-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Sync time once
            After=network-online.target
            [Service]
            Type=oneshot
            TimeoutStartSec=300
            ExecCondition=/bin/bash -c 'systemctl is-enabled chronyd.service --quiet && exit 1 ||
exit 0'
            ExecStart=/usr/sbin/chronyd -n -f /etc/chrony.conf -q
            RemainAfterExit=yes
            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: sync-time-once.service
```

6.7.6. PTP

单节点 OpenShift 集群使用 Precision Time Protocol (PTP) 进行网络时间同步。以下示例 PtpConfig CR 演示了普通时钟、边界时钟和 grandmaster 时钟所需的 PTP 配置。您应用的确切配置将

取决于节点硬件和特定用例。

推荐的 PTP 普通时钟配置 (PtpConfigSlave.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: slave
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "slave"
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 1
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 255
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
        fault_reset_interval -4
```

```
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
```

```
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: "slave"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

推荐的边界时钟配置 (PtpConfigBoundary.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "boundary"
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        # The interface name is hardware-specific
        [$iface_slave]
        masterOnly 0
        [$iface_master_1]
        masterOnly 1
        [$iface_master_2]
        masterOnly 1
        [$iface_master_3]
        masterOnly 1
        [global]
```

```
#
# Default Data Set
#
twoStepFlag 1
slaveOnly 0
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 248
clockAccuracy 0xFE
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
```

```
      pi_proportional_const 0.0
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: "boundary"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

推荐的 PTP Westport Channel e810 grandmaster 时钟配置 (PtpConfigGmWpc.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4lOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffSet: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 100
          pins: $e810_pins
          #  "$iface_master":
          #    "U.FL2": "0 2"
          #    "U.FL1": "0 1"
          #    "SMA2": "0 2"
          #    "SMA1": "0 1"
          ublxCmds:
            - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
                - "-P"
                - "29.20"
                - "-z"
                - "CFG-HW-ANT_CFG_VOLTCTRL,1"
              reportOutput: false
            - args: #ubxtool -P 29.20 -e GPS
                - "-P"
                - "29.20"
                - "-e"
                - "GPS"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d Galileo
                - "-P"
                - "29.20"
                - "-d"
                - "Galileo"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d GLONASS
                - "-P"
                - "29.20"
                - "-d"
                - "GLONASS"
              reportOutput: false
```

```
      - args: #ubxtool -P 29.20 -d BeiDou
          - "-P"
          - "29.20"
          - "-d"
          - "BeiDou"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d SBAS
          - "-P"
          - "29.20"
          - "-d"
          - "SBAS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
          - "-P"
          - "29.20"
          - "-t"
          - "-w"
          - "5"
          - "-v"
          - "1"
          - "-e"
          - "SURVEYIN,600,50000"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p MON-HW
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
  ts2phcOpts: " "
  ts2phcConf: |
    [nmea]
    ts2phc.master 1
    [global]
    use_syslog  0
    verbose 1
    logging_level 7
    ts2phc.pulsewidth 100000000
    #cat /dev/GNSS to find available serial port
    #example value of gnss_serialport is /dev/ttyGNSS_1700_0
    ts2phc.nmea_serialport $gnss_serialport
    leapfile  /usr/share/zoneinfo/leap-seconds.list
    [$iface_master]
    ts2phc.extts_polarity rising
    ts2phc.extts_correction 0
  ptp4lConf: |
    [$iface_master]
    masterOnly 1
    [$iface_master_1]
    masterOnly 1
    [$iface_master_2]
    masterOnly 1
    [$iface_master_3]
    masterOnly 1
    [global]
    #
```

```
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
```

```
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      clock_servo pi
      sanity_freq_limit  200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
  recommend:
    - profile: "grandmaster"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

以下可选 **PtpOperatorConfig CR** 为节点配置 **PTP** 事件报告。

推荐的 PTP 事件配置 (PtpOperatorConfigForEvent.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
  annotations: {}
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/$mcp: ""
  ptpEventConfig:
    enableEventPublisher: true
    transportHost: "http://ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
```

6.7.7. 扩展的 Tuned 配置集

运行 DU 工作负载的单节点 OpenShift 集群需要额外的高性能工作负载所需的性能调优配置。以下 Tuned CR 示例扩展了 Tuned 配置集：

推荐的扩展 Tuned 配置集配置 (Tuned PerformancePatch.yaml)

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: performance-patch
      # Please note:
      # - The 'include' line must match the associated PerformanceProfile name, following below
pattern
      #   include=openshift-node-performance-${PerformanceProfile.metadata.name}
      # - When using the standard (non-realtime) kernel, remove the kernel.timer_migration
override from
      #   the [sysctl] section and remove the entire section if it is empty.
      data: |
        [main]
        summary=Configuration changes profile inherited from performance created tuned
        include=openshift-node-performance-openshift-node-performance-profile
        [scheduler]
```

```
      group.ice-ptp=0:f:10:*:ice-ptp.*
      group.ice-gnss=0:f:10:*:ice-gnss.*
      group.ice-dplls=0:f:10:*:ice-dplls.*
      [service]
      service.stalld=start,enable
      service.chronyd=stop,disable
  recommend:
   - machineConfigLabels:
      machineconfiguration.openshift.io/role: "$mcp"
     priority: 19
     profile: performance-patch
```

表 6.4. 单节点 OpenShift 集群的 Tuned CR 选项

TuneD CR 字段	描述
spec.profile.data	• 您在 **spec.profile.data** 中设置的 **include** 行必须与关联的 **PerformanceProfile** CR 名称匹配。例如 **include=openshift-node-performance-${PerformanceProfile.metadata.name}**。

6.7.8. SR-IOV

单根 I/O 虚拟化(SR-IOV)通常用于启用前端和中间网络。以下 YAML 示例为单节点 OpenShift 集群配置 SR-IOV。

注意

SriovNetwork CR 的配置会根据您的特定网络和基础架构要求而有所不同。

推荐的 SriovOperatorConfig CR 配置 (SriovOperatorConfig.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
```

```
configDaemonNodeSelector:
  "node-role.kubernetes.io/$mcp": ""
# Injector and OperatorWebhook pods can be disabled (set to "false") below
# to reduce the number of management pods. It is recommended to start with the
# webhook and injector pods enabled, and only disable them after verifying the
# correctness of user manifests.
#   If the injector is disabled, containers using sr-iov resources must explicitly assign
#   them in the  "requests"/"limits" section of the container spec, for example:
#    containers:
#    - name: my-sriov-workload-container
#      resources:
#        limits:
#          openshift.io/<resource_name>:  "1"
#        requests:
#          openshift.io/<resource_name>:  "1"
enableInjector: false
enableOperatorWebhook: false
# Disable drain is needed for single-node OpenShift.
disableDrain: true
logLevel: 0
```

表 6.5. 用于单节点 OpenShift 集群的 SriovOperatorConfig CR 选项

SriovOperatorConfig CR 字段	描述

spec.enableInjector	禁用 **Injector** pod 以减少管理 pod 的数量。从启用 **Injector** pod 开始，仅在验证用户清单后禁用它们。如果 Injector 被禁用，使用 SR-IOV 资源的容器必须在容器 spec 的 **requests** 和 **limits** 部分中明确分配它们。
	
	例如：
	
	`containers:`
	`- name: my-sriov-workload-container`
	`  resources:`
	`    limits:`
	`      openshift.io/<resource_name>:  "1"`
	`    requests:`
	`      openshift.io/<resource_name>:  "1"`

spec.enableOperatorWebhook	禁用 **OperatorWebhook** pod 以减少管理 pod 的数量。从启用 **OperatorWebhook** pod 开始，仅在验证用户清单后禁用它们。

推荐的 SriovNetwork 配置 (SriovNetwork.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: ""
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  #  resourceName: ""
  networkNamespace: openshift-sriov-network-operator
#  vlan: ""
#  spoofChk: ""
#  ipam: ""
#  linkState: ""
#  maxTxRate: ""
#  minTxRate: ""
#  vlanQoS: ""
#  trust: ""
#  capabilities: ""
```

表 6.6. 用于单节点 OpenShift 集群的 SriovNetwork CR 选项

SriovNetwork CR 字段	描述
spec.vlan	为 midhaul 网络配置 VLAN 的 **vlan**。

推荐的 SriovNetworkNodePolicy CR 配置 (SriovNetworkNodePolicy.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: $name
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  # The attributes for Mellanox/Intel based NICs as below.
  #     deviceType: netdevice/vfio-pci
  #     isRdma: true/false
  deviceType: $deviceType
  isRdma: $isRdma
  nicSelector:
    # The exact physical function name must match the hardware used
    pfNames: [$pfNames]
  nodeSelector:
    node-role.kubernetes.io/$mcp: ""
  numVfs: $numVfs
  priority: $priority
  resourceName: $resourceName
```

表 6.7. 用于单节点 OpenShift 集群的 SriovNetworkPolicy CR 选项

SriovNetworkNodePolicy CR 字段	描述
spec.deviceType	将 **deviceType** 配置为 **vfio-pci** 或 **netdevice**。对于 Mellanox NIC，设置 **deviceType: netdevice**, 和 **isRdma: true**。对于基于 Intel 的 NIC，设置 **deviceType: vfio-pci** 和 **isRdma: false**。
spec.nicSelector.pfNames	指定连接到前端网络的接口。
spec.numVfs	指定前端网络的 VF 数量。
spec.nicSelector.pfNames	物理功能的确切名称必须与硬件匹配。

推荐的 SR-IOV 内核配置 (07-sriov-related-kernel-args-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 07-sriov-related-kernel-args-master
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
    - intel_iommu=on
    - iommu=pt
```

6.7.9. Console Operator

使用集群功能来防止安装 Console Operator。当节点被集中管理时，不需要它。删除 Operator 为应用程序工作负载提供额外的空间和容量。

要在安装过程中禁用 Console Operator，请在 SiteConfig 自定义资源(CR)的 spec.clusters.0.installConfigOverrides 字段中设置以下内容：

```
installConfigOverrides:  "{\"capabilities\":{\"baselineCapabilitySet\": \"None\" }}"
```

6.7.10. Alertmanager

运行 DU 工作负载的单节点 OpenShift 集群需要减少 OpenShift Container Platform 监控组件所消耗的 CPU 资源。以下 ConfigMap 自定义资源(CR)禁用 Alertmanager。

推荐的集群监控配置 (ReduceMonitoringFootprint.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
  annotations: {}
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
    telemeterClient:
      enabled: false
    prometheusK8s:
      retention: 24h
```

6.7.11. Operator Lifecycle Manager

运行分布式单元工作负载的单节点 OpenShift 集群需要对 CPU 资源进行一致的访问。Operator Lifecycle Manager (OLM) 会定期从 Operator 收集性能数据，从而增加 CPU 利用率。以下 ConfigMap 自定义资源 (CR) 禁用 OLM 的 Operator 性能数据收集。

推荐的集群 OLM 配置 (ReduceOLMFootprint.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: collect-profiles-config
  namespace: openshift-operator-lifecycle-manager
data:
  pprof-config.yaml: |
    disabled: True
```

6.7.12. LVM 存储

您可以使用逻辑卷管理器(LVM)存储在单节点 OpenShift 集群上动态置备本地存储。

注意

推荐的单节点 OpenShift 存储解决方案是 Local Storage Operator。另外，您可以使用 LVM Storage，但需要额外的 CPU 资源。

以下 YAML 示例将节点的存储配置为可供 OpenShift Container Platform 应用程序使用。

推荐的 LVMCluster 配置 (StorageLVMCluster.yaml)

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: odf-lvmcluster
  namespace: openshift-storage
spec:
  storage:
    deviceClasses:
    - name: vg1
      deviceSelector:
        paths:
        - /usr/disk/by-path/pci-0000:11:00.0-nvme-1
      thinPoolConfig:
        name: thin-pool-1
        overprovisionRatio: 10
        sizePercent: 90
```

表 6.8. 单节点 OpenShift 集群的 LVMCluster CR 选项

LVMCluster CR 字段	描述
deviceSelector.paths	配置用于 LVM 存储的磁盘。如果没有指定磁盘，LVM 存储将使用指定精简池中所有未使用的磁盘。

6.7.13. 网络诊断

运行 DU 工作负载的单节点 OpenShift 集群需要较少的 pod 网络连接检查，以减少这些 pod 创建的额外负载。以下自定义资源 (CR) 禁用这些检查。

推荐的网络诊断配置 (DisableSnoNetworkDiag.yaml)

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
  annotations: {}
spec:
  disableNetworkDiagnostics: true
```

其他资源

- 使用 ZTP 部署边缘站点

第 7 章 为 VDU 应用程序工作负载验证单节点 OPENSHIFT 集群调整

在部署虚拟分布式单元 (vDU) 应用程序前，您需要调整并配置集群主机固件和各种其他集群配置设置。使用以下信息来验证集群配置以支持 vDU 工作负载。

其他资源

- 使用 GitOps ZTP 在单节点 OpenShift 中的工作负载分区
- 在单节点 OpenShift 中部署 vDU 的参考配置

7.1. VDU 集群主机的建议固件配置

使用下表为在 OpenShift Container Platform 4.16 上运行的 vDU 应用程序配置集群主机固件的基础。

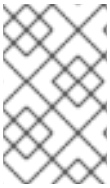
注意

下表是 vDU 集群主机固件配置的一般建议。具体固件设置将取决于您的要求和特定的硬件平台。固件的自动设置不会被零接触置备管道处理。

表 7.1. 推荐的集群主机固件设置

固件设置	配置	描述
HyperTransport (HT)	Enabled	HyperTransport (HT) 总线是由 AMD 开发的总线技术。HT 提供主机内存中组件与其他系统外围之间的高速链接。
UEFI	Enabled	为 vDU 主机启用从 UEFI 引导。
CPU Power 和性能策略	性能	设置 CPU 电源和性能策略，以优化系统以提高能源效率。
非核心频率扩展	Disabled	禁用 Uncore Frequency 扩展，以防止单独设置 CPU 的非内核部分和频率。
Uncore Frequency	最大值	将 CPU 的非内核部分（如缓存和内存控制器）设置为操作最多可能的频率。
性能限制	Disabled	禁用性能 P-limit 以防止处理器的 Uncore 频率协调。

固件设置	配置	描述
增强的 Intel® SpeedStep Tech	Enabled	启用增强的 Intel SpeedStep，以便系统动态调整处理器消耗和降低主机中功耗和 heat 生产的核心频率。
Intel® Turbo Boost Technology	Enabled	为基于 Intel 的 CPU 启用 Turbo Boost Technology，允许处理器内核比底层操作频率更快运行（如果它们低于 power、current 和 temperature 规格限制）。
Intel 配置的 TDP	Enabled	为 CPU 启用 Thermal Design Power (TDP)
可配置 TDP 级别	2 级	TDP 级别设置特定性能评级所需的 CPU 功耗。TDP 级别 2 以功耗为代价以实现最稳定的性能水平。
节能 Turbo	Disabled	禁用 Energy Efficient Turbo，以防止处理器使用基于能源效率的策略。
硬件 P-State	Enabled 或 Disabled	启用 OS 控制的 P-States 以允许节能配置。禁用 **P-states** （性能状态）以优化操作系统和 CPU 以提高功耗。
软件包 C-State	C0/C1 状态	使用 C0 或 C1 状态将处理器设置为完全活动状态 (C0) 或停止在软件中运行的 CPU 内部时钟 (C1)。
C1E	Disabled	CPU Enhanced Halt (C1E) 是 Intel 芯片中的节能功能。禁用 C1E 可防止操作系统在不活跃时向 CPU 发送 halt 命令。
处理器 C6	Disabled	C6 节能程序是 CPU 功能，可自动禁用空闲 CPU 内核和缓存。禁用 C6 可提高系统性能。
子 NUMA 集群	Disabled	子 NUMA 集群将处理器内核、缓存和内存划分为多个 NUMA 域。禁用这个选项可以提高对延迟敏感工作负载的性能。



注意

在主机的固件中启用全局 SR-IOV 和 VT-d 设置。这些设置与裸机环境相关。

注意

启用 C-states 和 OS 控制的 P-States 来允许每个 pod 电源管理。

7.2. 推荐的集群配置来运行 VDU 应用程序

运行虚拟化分布式单元 (vDU) 应用程序的集群需要高度调整和优化的配置。以下信息描述了在

OpenShift Container Platform 4.16 集群中支持 vDU 工作负载的各种元素。

7.2.1. 为单节点 OpenShift 集群推荐的集群 MachineConfig CR

检查您从 ztp-site-generate 容器中提取的 MachineConfig 自定义资源 (CR) 是否已在集群中应用。CR 可以在提取的 out/source-crs/extra-manifest/ 文件夹中找到。

ztp-site-generate 容器中的以下 MachineConfig CR 配置集群主机：

表 7.2. 推荐的 GitOps ZTP MachineConfig CR

MachineConfig CR	描述
01-container-mount-ns-and-kubelet-conf-master.yaml **01-container-mount-ns-and-kubelet-conf-worker.yaml**	配置容器挂载命名空间和 kubelet 配置。
03-sctp-machine-config-master.yaml **03-sctp-machine-config-worker.yaml**	加载 SCTP 内核模块。这些 **MachineConfig** CR 是可选的，如果您不需要这个内核模块，则可以省略。
05-kdump-config-master.yaml **05-kdump-config-worker.yaml** **06-kdump-master.yaml** **06-kdump-worker.yaml**	为集群配置 kdump 崩溃报告。
07-sriov-related-kernel-args-master.yaml	在集群中配置 SR-IOV 内核参数。
08-set-rcu-normal-master.yaml **08-set-rcu-normal-worker.yaml**	在集群重启后禁用 **rcu_expedited** 模式。
99-crio-disable-wipe-master.yaml **99-crio-disable-wipe-worker.yaml**	在集群重启后禁用自动 CRI-O 缓存擦除。
99-sync-time-once-master.yaml **99-sync-time-once-worker.yaml**	通过 Chrony 服务配置一次性检查并调整系统时钟。
enable-crun-master.yaml **enable-crun-worker.yaml**	启用 **crun** OCI 容器运行时。

MachineConfig CR	描述
extra-manifest/enable-cgroups-v1.yaml **source-crs/extra-manifest/enable-cgroups-v1.yaml**	在集群安装过程中和生成 RHACM 集群策略时启用 cgroup v1。

注意

在 OpenShift Container Platform 4.14 及更高版本中，您可以使用 SiteConfig CR 中的 cpuPartitioningMode 字段配置工作负载分区。

其他资源

- **使用 GitOps ZTP 在单节点 OpenShift 中的工作负载分区**
- **从 ztp-site-generate 容器中提取源 CR**

7.2.2. 推荐的集群 Operator

运行虚拟化分布式单元 (vDU) 应用程序的集群需要以下 Operator，它是基准参考配置的一部分：

- **Node Tuning Operator (NTO).与 Performance Addon Operator 一起提供的 NTO 软件包功能，现在是 NTO 的一部分。**
- **PTP Operator**
- **Cluster Network Operator**
- **Red Hat OpenShift Logging Operator**
- **Local Storage Operator**

7.2.3. 推荐的集群内核配置

始终使用集群中最新支持的实时内核版本。确保在集群中应用以下配置：

1. 确保在集群性能配置集中设置以下 additionalKernelArgs ：

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
# ...
spec:
  additionalKernelArgs:
  - "rcupdate.rcu_normal_after_boot=0"
  - "efi=runtime"
  - "vfio_pci.enable_sriov=1"
  - "vfio_pci.disable_idle_d3=1"
  - "module_blacklist=irdma"

  # ...
```

2. 可选：在 hardwareTuning 字段中设置 CPU 频率：

您可以使用硬件调优来为保留和隔离的内核 CPU 调优 CPU 频率。对于像应用程序一样的 FlexRAN，硬件厂商建议您在默认 CPU 上运行 CPU 频率。强烈建议您在设置任何频率之前，请参阅硬件厂商的准则，以获得处理器生成的最大频率设置。本例显示了为 Sapphire 快速硬件保留和隔离的 CPU 频率：

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
    cpu:
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
        - size: 1G
          count: 32
    realTimeKernel:
        enabled: true
    hardwareTuning:
        isolatedCpuFreq: 2500000
        reservedCpuFreq: 2800000
```

3. 确保 Tuned CR 中的 performance-patch 配置集配置与相关 PerformanceProfile CR 中设置的隔离 CPU 的正确 CPU 隔离集，例如：

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: performance-patch
      # The 'include' line must match the associated PerformanceProfile name, for
example:
      # include=openshift-node-performance-${PerformanceProfile.metadata.name}
      # When using the standard (non-realtime) kernel, remove the
kernel.timer_migration override from the [sysctl] section
      data: |
        [main]
        summary=Configuration changes profile inherited from performance created
tuned
        include=openshift-node-performance-openshift-node-performance-profile
        [scheduler]
        group.ice-ptp=0:f:10:*:ice-ptp.*
        group.ice-gnss=0:f:10:*:ice-gnss.*
        group.ice-dplls=0:f:10:*:ice-dplls.*
        [service]
        service.stalld=start,enable
        service.chronyd=stop,disable
# ...
```

7.2.4. 检查实时内核版本

在 OpenShift Container Platform 集群中，始终使用最新版本的 realtime 内核。如果您不确定集群中正在使用的内核版本，您可以将当前的 realtime 内核版本与发行版本进行比较。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您以具有 cluster-admin 权限的用户身份登录。
- 已安装 podman。

流程

1. 运行以下命令来获取集群版本：

```
$ OCP_VERSION=$(oc get clusterversion version -o
jsonpath='{.status.desired.version}{"\n"}')
```

2. 获取发行镜像 SHA 号：

```
$ DTK_IMAGE=$(oc adm release info --image-for=driver-toolkit quay.io/openshift-
release-dev/ocp-release:$OCP_VERSION-x86_64)
```

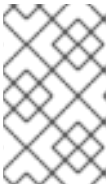
3. 运行发行镜像容器，并提取与集群当前发行版本一起打包的内核版本：

```
$ podman run --rm $DTK_IMAGE rpm -qa | grep 'kernel-rt-core-' | sed 's#kernel-rt-
core-##'
```

输出示例

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

这是版本附带的默认 realtime 内核版本。



注意

realtime 内核由内核版本中的字符串 .rt 表示。

验证

检查为集群当前发行版本列出的内核版本是否与集群中运行的实际实时内核匹配。运行以下命令检查运行的 realtime 内核版本：

1. 打开到集群节点的远程 shell 连接：

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

2. 检查 realtime 内核版本：

```
sh-4.4# uname -r
```

输出示例

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

7.3. 检查是否应用推荐的集群配置

您可以检查集群是否正在运行正确的配置。以下流程描述了如何检查在 OpenShift Container Platform 4.16 集群中部署 DU 应用程序的各种配置。

先决条件

- 您已部署了集群，并根据 vDU 工作负载对其进行调整。
- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。

流程

1. 检查默认 OperatorHub 源是否已禁用。运行以下命令：

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

输出示例

```
spec:
  disableAllDefaultSources: true
```

2. 运行以下命令，检查所有所需的 CatalogSource 资源是否标注了工作负载分区 (PreferredDuringScheduling) ：

```
$ oc get catalogsource -A -o jsonpath='{range .items[*]}{.metadata.name}{" -- "}
{.metadata.annotations.target\.workload\.openshift\.io/management}{"\n"}{end}'
```

输出示例

```
certified-operators -- {"effect": "PreferredDuringScheduling"}
community-operators -- {"effect": "PreferredDuringScheduling"}
ran-operators 1
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

1 未注解的 CatalogSource 资源也会返回。在本例中，ran-operators CatalogSource 资源没有被注解，它没有 PreferredDuringScheduling 注解。

注意

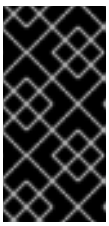
在正确配置的 vDU 集群中，只会列出注解的一个目录源。

3. 检查是否为工作负载分区注解了所有适用的 OpenShift Container Platform Operator 命名空间。这包括 OpenShift Container Platform 核心安装的所有 Operator，以及参考 DU 调整配置中包含的附加 Operator 集合。运行以下命令：

```
$ oc get namespaces -A -o jsonpath='{range .items[*]}{.metadata.name}{" -- "}
{.metadata.annotations.workload\.openshift\.io/allowed}{"\n"}{end}'
```

输出示例

```
default --
openshift-apiserver -- management
openshift-apiserver-operator -- management
openshift-authentication -- management
openshift-authentication-operator -- management
```



重要

对于工作负载分区，不得为其他 Operator 进行注解。在上一命令的输出中，应当列出额外的 Operator，而无需 -- 分隔符右侧的任何值。

4. 检查 ClusterLogging 配置是否正确。运行以下命令：

   a. 验证是否配置了适当的输入和输出日志：

```
$ oc get -n openshift-logging ClusterLogForwarder instance -o yaml
```

输出示例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  creationTimestamp: "2022-07-19T21:51:41Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "1030342"
  uid: 8c1a842d-80c5-447a-9150-40350bdf40f0
spec:
  inputs:
  - infrastructure: {}
    name: infra-logs
  outputs:
  - name: kafka-open
    type: kafka
    url: tcp://10.46.55.190:9092/test
  pipelines:
  - inputRefs:
    - audit
    name: audit-logs
    outputRefs:
    - kafka-open
```

```
  - inputRefs:
    - infrastructure
    name: infrastructure-logs
    outputRefs:
    - kafka-open
...
```

b. 检查策展调度是否适合您的应用程序：

```
$ oc get -n openshift-logging clusterloggings.logging.openshift.io instance -o
yaml
```

输出示例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  creationTimestamp: "2022-07-07T18:22:56Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "235796"
  uid: ef67b9b8-0e65-4a10-88ff-ec06922ea796
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    curator:
      schedule: 30 3 * * *
    type: curator
  managementState: Managed
...
```

5. 运行以下命令，检查 Web 控制台是否已禁用 (managementState: Removed)：

```
$ oc get consoles.operator.openshift.io cluster -o jsonpath="{ .spec.managementState
}"
```

输出示例

```
Removed
```

6. 运行以下命令，检查集群节点中禁用了 chronyd ：

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

检查节点上的 chronyd 状态：

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

```
sh-4.4# systemctl status chronyd
```

输出示例

```
● chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset:
enabled)
   Active: inactive (dead)
     Docs: man:chronyd(8)
           man:chrony.conf(5)
```

7. 使用连接到 linuxptp-daemon 容器和 PTP Management Client (pmc) 工具，检查 PTP 接口是否已成功同步到主时钟：

a. 运行以下命令，使用 linuxptp-daemon pod 的名称设置 $PTP_POD_NAME 变量：

```
$ PTP_POD_NAME=$(oc get pods -n openshift-ptp -l app=linuxptp-daemon -o
name)
```

.

b.

运行以下命令来检查 PTP 设备的同步状态：

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u
-f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

输出示例

```
sending: GET PORT_DATA_SET
  3cecef.fffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-1
    portState               SLAVE
    logMinDelayReqInterval  -4
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
  3cecef.fffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-2
    portState               LISTENING
    logMinDelayReqInterval  0
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
```

c.

运行以下 pmc 命令来检查 PTP 时钟状态：

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u
-f /var/run/ptp4l.0.config -b 0 'GET TIME_STATUS_NP'
```

输出示例

```
sending: GET TIME_STATUS_NP
  3cecef.fffe.7a7020-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset           10 (1)
```

```
ingress_time              1657275432697400530
cumulativeScaledRateOffset +0.000000000
scaledLastGmPhaseChange    0
gmTimeBaseIndicator        0
lastGmPhaseChange          0x0000'0000000000000000.0000
gmPresent                  true 2
gmIdentity                 3c2c30.ffff.670e00
```

1 master_offset 应该介于 -100 到 100 ns 之间。

2 这表示 PTP 时钟被同步到 master，本地时钟不是 grandmaster 时钟。

d. 检查在 linuxptp-daemon-container 日志中有与 /var/run/ptp4l.0.config 中的值对应的 master offset ：

```
$ oc logs $PTP_POD_NAME -n openshift-ptp -c linuxptp-daemon-container
```

输出示例

```
phc2sys[56020.341]: [ptp4l.1.config] CLOCK_REALTIME phc offset  -1731092 s2
freq -1546242 delay    497
ptp4l[56020.390]: [ptp4l.1.config] master offset         -2 s2 freq   -5863 path delay
541
ptp4l[56020.390]: [ptp4l.0.config] master offset         -8 s2 freq  -10699 path delay
533
```

8. 运行以下命令检查 SR-IOV 配置是否正确：

a. 检查 SriovOperatorConfig 资源中的 disableDrain 值是否已设置为 true ：

```
$ oc get sriovoperatorconfig -n openshift-sriov-network-operator default -o
jsonpath="{.spec.disableDrain}{'\n'}"
```

-

输出示例

```
true
```

b. 运行以下命令，检查 SriovNetworkNodeState 同步状态是否为 Succeeded ：

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o
jsonpath="{.items[*].status.syncStatus}{'\n'}"
```

输出示例

```
Succeeded
```

c. 验证为 SR-IOV 配置的每个接口下的虚拟功能（Vfs）预期数量和配置是否存在，并在 .status.interfaces 字段中是正确的。例如：

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: sriovnetwork.openshift.io/v1
  kind: SriovNetworkNodeState
...
  status:
    interfaces:
    ...
    - Vfs:
      - deviceID: 154c
        driver: vfio-pci
        pciAddress: 0000:3b:0a.0
        vendor: "8086"
```

```
    vfID: 0
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.1
    vendor: "8086"
    vfID: 1
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.2
    vendor: "8086"
    vfID: 2
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.3
    vendor: "8086"
    vfID: 3
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.4
    vendor: "8086"
    vfID: 4
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.5
    vendor: "8086"
    vfID: 5
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.6
    vendor: "8086"
    vfID: 6
  - deviceID: 154c
    driver: vfio-pci
    pciAddress: 0000:3b:0a.7
    vendor: "8086"
    vfID: 7
```

9. 检查集群性能配置集是否正确。cpu 和 hugepages 部分将根据您的硬件配置而有所不同。运行以下命令：

```
$ oc get PerformanceProfile openshift-node-performance-profile -o yaml
```

输出示例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
```

```
  creationTimestamp: "2022-07-19T21:51:31Z"
  finalizers:
  - foreground-deletion
  generation: 1
  name: openshift-node-performance-profile
  resourceVersion: "33558"
  uid: 217958c0-9122-4c62-9d4d-fdc27c31118c
spec:
  additionalKernelArgs:
  - idle=poll
  - rcupdate.rcu_normal_after_boot=0
  - efi=runtime
  cpu:
    isolated: 2-51,54-103
    reserved: 0-1,52-53
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 32
      size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
status:
  conditions:
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "True"
    type: Available
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "True"
    type: Upgradeable
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "False"
    type: Progressing
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "False"
    type: Degraded
  runtimeClass: performance-openshift-node-performance-profile
  tuned: openshift-cluster-node-tuning-operator/openshift-node-performance-
openshift-node-performance-profile
```

注意

CPU 设置取决于服务器上可用的内核数，应当与工作负载分区设置保持一致。巨页配置取决于服务器和应用程序。

10. 运行以下命令，检查 PerformanceProfile 是否已成功应用到集群：

```
$ oc get performanceprofile openshift-node-performance-profile -o jsonpath="{range
.status.conditions[*]}{ @.type }{' -- '}{@.status}{'\n'}{end}"
```

输出示例

```
Available -- True
Upgradeable -- True
Progressing -- False
Degraded -- False
```

11. 运行以下命令检查 Tuned 性能补丁设置：

```
$ oc get tuneds.tuned.openshift.io -n openshift-cluster-node-tuning-operator
performance-patch -o yaml
```

输出示例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  creationTimestamp: "2022-07-18T10:33:52Z"
  generation: 1
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  resourceVersion: "34024"
  uid: f9799811-f744-4179-bf00-32d4436c08fd
spec:
  profile:
  - data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
```

```
    [bootloader]
    cmdline_crash=nohz_full=2-23,26-47 1
    [sysctl]
    kernel.timer_migration=1
    [scheduler]
    group.ice-ptp=0:f:10:*:ice-ptp.*
    [service]
    service.stalld=start,enable
    service.chronyd=stop,disable
  name: performance-patch
recommend:
- machineConfigLabels:
    machineconfiguration.openshift.io/role: master
  priority: 19
  profile: performance-patch
```

1 cmdline=nohz_full= 中的 cpu 列表将根据您的硬件配置而有所不同。

12. 运行以下命令，检查是否禁用了集群网络诊断：

```
$ oc get networks.operator.openshift.io cluster -o
jsonpath='{.spec.disableNetworkDiagnostics}'
```

输出示例

```
true
```

13. 检查 Kubelet housekeeping 间隔是否调整为较慢的速度。这是在 containerMountNS 机器配置中设置的。运行以下命令：

```
$ oc describe machineconfig container-mount-namespace-and-kubelet-conf-master |
grep OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
```

输出示例

```
Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
```

14. 运行以下命令，检查 Grafana 和 alertManagerMain 是否已禁用，Prometheus 保留周期是否已设置为 24h：

```
$ oc get configmap cluster-monitoring-config -n openshift-monitoring -o jsonpath="{
.data.config\.yaml }"
```

输出示例

```
grafana:
  enabled: false
alertmanagerMain:
  enabled: false
prometheusK8s:
   retention: 24h
```

a. 使用以下命令验证集群中没有找到 Grafana 和 alertManagerMain 路由：

```
$ oc get route -n openshift-monitoring alertmanager-main
```

```
$ oc get route -n openshift-monitoring grafana
```

这两个查询都应返回 Error from server(NotFound) 消息。

15. 运行以下命令，检查是否已为每个 PerformanceProfile、Tuned 性能补丁、工作负载分区和内核命令行参数分配至少 4 个保留 CPU：

```
$ oc get performanceprofile -o jsonpath="{ .items[0].spec.cpu.reserved }"
```

输出示例

0-3

注意

根据您的工作负载要求，您可能需要分配额外的保留 CPU。

第 8 章 带有 SITECONFIG 资源的高级受管集群配置

您可以使用 SiteConfig 自定义资源 (CR) 在安装时在受管集群中部署自定义功能和配置。

8.1. 在 GITOPS ZTP 管道中自定义额外的安装清单

您可以定义一组额外的清单，以包含在 GitOps Zero Touch Provisioning (ZTP) 管道的安装阶段。这些清单链接到 siteConfig 自定义资源(CR)，并在安装过程中应用到集群。在安装时包括 MachineConfig CR 可提高安装过程的效率。

先决条件

- 创建一个 Git 存储库，在其中管理自定义站点配置数据。该存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源仓库。

流程

1. 创建 GitOps ZTP 管道用于自定义集群安装的一组额外清单 CR。

2. 在自定义 /siteconfig 目录中，为您的额外清单创建一个子目录 /custom-manifest。以下示例演示了一个带有 /custom-manifest 文件夹的 /siteconfig 示例：

```
siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
├── extra-manifest/
└── custom-manifest
    └── 01-example-machine-config.yaml
```

注意

整个使用的子目录名称 /custom-manifest 和 /extra-manifest 只是示例名称。不需要使用这些名称，并且对如何命名这些子目录没有限制。在本例中，/extra-manifest 是指从 ztp-site-generate 容器存储 /extra-manifest 的内容的 Git 子目录。

3. 将自定义额外清单 CR 添加到 siteconfig/custom-manifest 目录中。

4. 在 SiteConfig CR 中，在 extraManifests.searchPaths 字段中输入目录名称，例如：

```
clusters:
- clusterName: "example-sno"
  networkType: "OVNKubernetes"
  extraManifests:
    searchPaths:
      - extra-manifest/ 1
      - custom-manifest/ 2
```

1 从 ztp-site-generate 容器复制的清单的文件夹。

2 自定义清单的文件夹。

5. 保存 SiteConfig、/extra-manifest 和 /custom-manifest CR，并将它们推送到站点配置存储库。

在集群置备过程中，GitOps ZTP 管道会将 /custom-manifest 目录中的 CR 附加到存储在 extra-manifest/ 中的默认额外清单集合中。

注意

从版本 4.14 extraManifestPath 开始，会受弃用警告。

虽然 extraManifestPath 仍然被支持，但我们建议您使用 extraManifests.searchPaths。如果您在 SiteConfig 文件中定义 extraManifests.searchPaths，GitOps ZTP 管道不会在站点安装过程中从 ztp-site-generate 容器获取清单。

如果您在 Siteconfig CR 中定义 extraManifestPath 和 extraManifests.searchPaths，则为 extraManifests.searchPaths 定义的设置具有优先权。

强烈建议您从 ztp-site-generate 容器中提取 /extra-manifest 的内容，并将它推送到 GIT 存储库。

8.2. 使用 SITECONFIG 过滤器过滤自定义资源

通过使用过滤器，您可以轻松地自定义 SiteConfig 自定义资源 (CR)，使其包含或排除其他 CR，以便在 GitOps Zero Touch Provisioning (ZTP) 管道的安装阶段使用。

您可以为 SiteConfig CR 指定一个 inclusionDefault 值（include 或 exclude），以及您要包含或排除的特定 extraManifest RAN CR 列表。将 inclusionDefault 设置为 include 可使 GitOps ZTP 管道在安装过程中应用 /source-crs/extra-manifest 中的所有文件。将 includeDefault 设置为 exclude 的作用相反。

您可以从 /source-crs/extra-manifest 文件夹中排除默认会被包括的 CR。以下示例配置了自定义单节点 OpenShift SiteConfig CR，以在安装时排除 /source-crs/extra-manifest/03-sctp-machine-config-worker.yaml CR。

另外还介绍了一些额外的可选过滤场景。

先决条件

- 配置了 hub 集群来生成所需的安装和策略 CR。
- 您创建了 Git 存储库，用于管理自定义站点配置数据。该存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源仓库。

流程

1. 要防止 GitOps ZTP 管道应用 03-sctp-machine-config-worker.yaml CR 文件，请在 SiteConfig CR 中应用以下 YAML：

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "site1-sno-du"
  namespace: "site1-sno-du"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.16"
  sshPublicKey: "<ssh_public_key>"
  clusters:
- clusterName: "site1-sno-du"
  extraManifests:
```

```
    filter:
      exclude:
        - 03-sctp-machine-config-worker.yaml
```

GitOps ZTP 管道在安装过程中跳过 03-sctp-machine-config-worker.yaml CR。应用 /source-crs/extra-manifest 中的所有其他 CR。

2. 保存 SiteConfig CR，并将更改推送到站点配置存储库。

GitOps ZTP 管道监控并调整根据 SiteConfig 过滤器指令所应用的 CR。

3. 可选：要防止 GitOps ZTP 管道在集群中应用所有 /source-crs/extra-manifest CR，请在 SiteConfig CR 中应用以下 YAML：

```
- clusterName: "site1-sno-du"
  extraManifests:
    filter:
      inclusionDefault: exclude
```

4. 可选：要排除所有 /source-crs/extra-manifest RAN CR，并在安装过程中包括自定义 CR 文件，编辑自定义 SiteConfig CR 来设置自定义清单文件夹和 include 文件，例如：

```
clusters:
- clusterName: "site1-sno-du"
  extraManifestPath: "<custom_manifest_folder>" 1
  extraManifests:
    filter:
      inclusionDefault: exclude 2
      include:
        - custom-sctp-machine-config-worker.yaml
```

1 将 <custom_manifest_folder> 替换为包含自定义安装 CR 的文件夹名称，如 user-custom-manifest/。

2 将 inclusionDefault 设置为 exclude 以防止 GitOps ZTP 管道在安装过程中应用 /source-crs/extra-manifest 中的文件。

以下示例演示了自定义文件夹结构：

```
siteconfig
  ├── site1-sno-du.yaml
  └── user-custom-manifest
      └── custom-sctp-machine-config-worker.yaml
```

8.3. 使用 SITECONFIG CR 删除节点

通过使用 SiteConfig 自定义资源(CR)，您可以删除并重新创建节点。这个方法比手动删除节点更高效。

先决条件

- 您已将 hub 集群配置为生成所需的安装和策略 CR。
- 您已创建了 Git 存储库，您可以在其中管理自定义站点配置数据。存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源存储库。

流程

1. 更新 SiteConfig CR，使其包含 bmac.agent-install.openshift.io/remove-agent-and-node-on-delete=true 注解：

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "cnfdf20"
  namespace: "cnfdf20"
spec:
  Clusters:
    nodes:
    - hostname: node6
      role: "worker"
      crAnnotations:
        add:
          BareMetalHost:
            bmac.agent-install.openshift.io/remove-agent-and-node-on-delete: true
# ...
```

2. 通过更新 SiteConfig CR 使其包含 crSuppression. BareMetalHost 注解来抑制 BareMetalHost CR 的生成：

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "cnfdf20"
  namespace: "cnfdf20"
spec:
  clusters:
  - nodes:
    - hostName: node6
      role: "worker"
      crSuppression:
      - BareMetalHost
# ...
```

3. 将更改推送到 Git 存储库并等待取消置备启动。BareMetalHost CR 的状态应更改为 deprovisioning。等待 BareMetalHost 完成取消置备，并完全删除。

验证

1. 运行以下命令，验证 worker 节点的 BareMetalHost 和 Agent CR 已从 hub 集群中删除：

```
$ oc get bmh -n <cluster-ns>
```

```
$ oc get agent -n <cluster-ns>
```

2. 运行以下命令，验证节点记录是否已从 spoke 集群中删除：

```
$ oc get nodes
```

注意

如果使用 secret，删除 secret 太早可能会导致问题，因为 ArgoCD 需要 secret 在删除后完成重新同步。只有在当前 ArgoCD 同步完成后，仅在节点清理后删除 secret。

后续步骤

要重新置备节点，请删除之前添加到 SiteConfig 中的更改，将更改推送到 Git 存储库，并等待同步完成。这会重新生成 worker 节点的 BareMetalHost CR，并触发重新安装节点。

第 9 章 使用 POLICYGENERATOR 资源管理集群策略

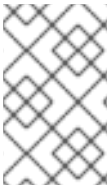
9.1. 使用 POLICYGENERATOR 资源配置受管集群策略

应用的 Policy 自定义资源(CR)配置您置备的受管集群。您可以自定义 Red Hat Advanced Cluster Management (RHACM)如何使用 PolicyGenerator CR 生成应用的 Policy CR。

重要

在 GitOps ZTP 中使用 PolicyGenerator 资源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。



注意

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

9.1.1. 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

在 GitOps ZTP 中可以使用 PolicyGenerator 自定义资源(CR)和 PolicyGenTemplate CR 为受管集群生成 RHACM 策略。

当涉及到使用 GitOps ZTP 修补 OpenShift Container Platform 资源时，使用 PolicyGenerator CR 而不是 PolicyGenTemplate CR 有优点。使用 RHACM PolicyGenerator API 提供了一种通用方法来修补资源，这些资源无法使用 PolicyGenTemplate 资源。

PolicyGenerator API 是 Open Cluster Management 标准的一部分，而 PolicyGenTemplate API 不是。下表描述了 PolicyGenerator 和 PolicyGenTemplate 资源补丁和放置策略的比较。

重要

使用 PolicyGenTemplate CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 PolicyGenerator CR 提供了等效和改进的功能。

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

表 9.1. RHACM PolicyGenerator 和 PolicyGenTemplate 补丁的比较

PolicyGenerator patching	PolicyGenTemplate patching
使用 Kustomize 战略性合并来合并资源。如需更多信息，请参阅使用 Kustomize 管理 Kubernetes 对象。	通过将变量替换为补丁所定义的值来工作。与 Kustomize 合并策略相比，这不太灵活。
支持 **ManagedClusterSet** 和 **Binding** 资源。	不支持 **ManagedClusterSet** 和 **Binding** 资源。
依赖于打补丁，不需要嵌入的变量替换。	覆盖补丁中定义的变量值。
不支持合并补丁中的合并列表。支持替换合并补丁中的列表。	合并和替换列表会有限，您只能在列表中合并一个对象。
目前不支持资源 补丁的 OpenAPI 规格。这意味着补丁中需要额外的指令来合并不遵循模式的内容，如 **PtpConfig** 资源。	通过将字段和值替换为补丁所定义的值来工作。
需要额外的指令，例如，补丁中的 **$patch: replace** 来合并不遵循模式的内容。	将源 CR 中定义的字段和值替换为补丁中定义的值，如 **$name**。
可以修补引用源 CR 中定义的 **Name** 和 **Namespace** 字段，但只有当 CR 文件只有一个对象时。	可以修补引用源 CR 中定义的 **Name** 和 **Namespace** 字段。

9.1.2. 关于 PolicyGenerator CRD

PolicyGenerator 自定义资源定义(CRD)告知 PolicyGen 策略生成器在集群配置中包含哪些自定义资源(CR)，如何将 CR 组合到生成的策略中，以及这些 CR 中的项目需要使用 overlay 内容更新。

以下示例显示了从 ztp-site-generate 引用容器中提取的 PolicyGenerator CR (acm-common-du-

ranGen.yaml)。acm-common-du-ranGen.yaml 文件定义了两个 Red Hat Advanced Cluster Management (RHACM)策略。策略管理配置 CR 集合，每个 CR 中的 policyName 值对应一个。acm-common-du-ranGen.yaml 创建一个单个放置绑定和一个放置规则，根据 policyDefaults.placement.labelSelector 部分中列出的标签将策略绑定到集群。

示例 PolicyGenerator CR - acm-common-ranGen.yaml

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: common-latest
placementBindingDefaults:
  name: common-latest-placement-binding (1)
policyDefaults:
  namespace: ztp-common
  placement:
    labelSelector:
      matchExpressions:
        - key: common
          operator: In
          values:
            - "true"
        - key: du-profile
          operator: In
          values:
            - latest
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: common-latest-config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/ReduceMonitoringFootprint.yaml
      - path: source-crs/DefaultCatsrc.yaml (2)
        patches:
          - metadata:
              name: redhat-operators-disconnected
            spec:
              displayName: disconnected-redhat-operators
              image: registry.example.com:5000/disconnected-redhat-operators/disconnected-redhat-operator-index:v4.9
      - path: source-crs/DisconnectedICSP.yaml
        patches:
```

```
        - spec:
            repositoryDigestMirrors:
              - mirrors:
                  - registry.example.com:5000
                source: registry.redhat.io
    - name: common-latest-subscriptions-policy
      policyAnnotations:
        ran.openshift.io/ztp-deploy-wave: "2"
      manifests: 3
        - path: source-crs/SriovSubscriptionNS.yaml
        - path: source-crs/SriovSubscriptionOperGroup.yaml
        - path: source-crs/SriovSubscription.yaml
        - path: source-crs/SriovOperatorStatus.yaml
        - path: source-crs/PtpSubscriptionNS.yaml
        - path: source-crs/PtpSubscriptionOperGroup.yaml
        - path: source-crs/PtpSubscription.yaml
        - path: source-crs/PtpOperatorStatus.yaml
        - path: source-crs/ClusterLogNS.yaml
        - path: source-crs/ClusterLogOperGroup.yaml
        - path: source-crs/ClusterLogSubscription.yaml
        - path: source-crs/ClusterLogOperatorStatus.yaml
        - path: source-crs/StorageNS.yaml
        - path: source-crs/StorageOperGroup.yaml
        - path: source-crs/StorageSubscription.yaml
        - path: source-crs/StorageOperatorStatus.yaml
```

1

将策略应用到具有此标签的所有集群。

2

DefaultCatsrc.yaml 文件包含断开连接的 registry 和相关 registry 配置详情的目录源。

3

policies.manifests 下列出的文件为已安装的集群创建 Operator 策略。

PolicyGenerator CR 可以使用任意数量的包含 CR 来构建。在 hub 集群中应用以下示例 CR 来生成包含单个 CR 的策略：

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno
placementBindingDefaults:
```

```
  name: group-du-sno-placement-binding
policyDefaults:
  namespace: ztp-group
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: group-du-sno-config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: '10'
    manifests:
      - path: source-crs/PtpConfigSlave-MCP-master.yaml
        patches:
          - metadata: null
            name: du-ptp-slave
            namespace: openshift-ptp
            annotations:
              ran.openshift.io/ztp-deploy-wave: '10'
            spec:
              profile:
                - name: slave
                  interface: $interface
                  ptp4lOpts: '-2 -s'
                  phc2sysOpts: '-a -r -n 24'
                  ptpSchedulingPolicy: SCHED_FIFO
                  ptpSchedulingPriority: 10
                  ptpSettings:
                    logReduce: 'true'
                  ptp4lConf: |
                    [global]
                    #
                    # Default Data Set
                    #
                    twoStepFlag 1
                    slaveOnly 1
                    priority1 128
                    priority2 128
                    domainNumber 24
                    #utc_offset 37
                    clockClass 255
                    clockAccuracy 0xFE
                    offsetScaledLogVariance 0xFFFF
                    free_running 0
                    freq_est_interval 1
```

```
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
```

```
            #
            # Transport options
            #
            transportSpecific 0x0
            ptp_dst_mac 01:1B:19:00:00:00
            p2p_dst_mac 01:80:C2:00:00:0E
            udp_ttl 1
            udp6_scope 0x0E
            uds_address /var/run/ptp4l
            #
            # Default interface options
            #
            clock_type OC
            network_transport L2
            delay_mechanism E2E
            time_stamping hardware
            tsproc_mode filter
            delay_filter moving_median
            delay_filter_length 10
            egressLatency 0
            ingressLatency 0
            boundary_clock_jbod 0
            #
            # Clock description
            #
            productDescription ;;
            revisionData ;;
            manufacturerIdentity 00:00:00
            userDescription ;
            timeSource 0xA0
        recommend:
          - profile: slave
            priority: 4
            match:
              - nodeLabel: node-role.kubernetes.io/master
```

使用源文件 PtpConfigSlave.yaml 作为示例，文件会定义一个 PtpConfig CR。为 PtpConfigSlave 示例生成的策略名为 group-du-sno-config-policy。生成的 group-du-sno-config-policy 中定义的 PtpConfig CR 被命名为 du-ptp-slave。PtpConfigSlave.yaml 中定义的 spec 放置在 du-ptp-slave 下，以及与源文件中定义的其他 spec 项目一起放置。

以下示例显示了 group-du-sno-config-policy CR：

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
```

```
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-operator-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
        - metadata:
            name: redhat-operators
          spec:
            displayName: Red Hat Operators Catalog
            image: registry.example.com:5000/olm/redhat-operators:v4.14
            updateStrategy:
              registryPoll:
                interval: 1h
          status:
            connectionState:
              lastObservedState: READY
```

9.1.3. 自定义 PolicyGenerator CR 时的建议

在自定义站点配置 PolicyGenerator 自定义资源(CR)时请考虑以下最佳实践：

- 根据需要使用一些策略。使用较少的策略需要较少的资源。每个附加策略会为 hub 集群和部署的受管集群创建 CPU 负载增加。CR 根据 PolicyGenerator CR 中的 policyName 字段合并到策略中。同一 PolicyGenerator 中的 CR，在单个策略下管理相同的 policyName 值。

- 在断开连接的环境中，通过将 registry 配置为包含所有 Operator 的单个索引，为所有 Operator 使用单个目录源。受管集群中的每个额外 CatalogSource CR 会增加 CPU 用量。

- MachineConfig CR 应包含在 siteConfig CR 中作为 extraManifests，以便在安装过程中应用它们。这可减少在集群就绪部署应用程序前所花费的总时间。

- PolicyGenerator CR 应该覆盖 channel 字段来明确识别所需的版本。这样可确保源 CR 在升级过程中的更改不会更新生成的订阅。

其他资源

- 有关使用 RHACM 扩展集群的建议，请参阅性能和可扩展性。

注意

在 hub 集群中管理大量 spoke 集群时，请最小化策略数量来减少资源消耗。

将多个配置 CR 分组到单个或有限的策略中，一种方法是减少 hub 集群上的总体策略数量。在使用 common/group/site 层次结构来管理站点配置时，务必要将特定于站点的配置组合成单一策略。

9.1.4. RAN 部署的 PolicyGenerator CR

使用 PolicyGenerator 自定义资源(CR)使用 GitOps Zero Touch Provisioning (ZTP)管道自定义应用到集群的配置。PolicyGenerator CR 允许您生成一个或多个策略来管理集群中的配置 CR 集合。PolicyGenerator CR 标识一组受管 CR，将它们捆绑到策略中，构建与这些 CR 相关的策略，并使用标签绑定规则将策略与集群相关联。

从 GitOps ZTP 容器获取的参考配置旨在提供一组关键功能和节点调优设置，以确保集群可以支持字符串的性能和资源利用率限制，典型的 RAN 分布式单元(DU)应用程序。来自基准配置的更改或禁止可能会影响功能可用性、性能和资源利用率。使用引用 PolicyGenerator CR 作为基础来创建根据您的特定站点要求量身定制的配置文件的层次结构。

为 RAN DU 集群配置定义的基准 策略生成器 CR 可以从 GitOps ZTP ztp-site-generate 容器中提取。如需了解更多详细信息，请参阅"准备 GitOps ZTP 站点配置存储库"。

PolicyGenerator CR 可以在 ./out/argocd/example/acmpolicygenerator/ 文件夹中找到。参考架构具有共同、组和特定站点的配置 CR。每个 PolicyGenerator CR 都引用可在 ./out/source-crs 文件夹中找到的其他 CR。

与 RAN 集群配置相关的 PolicyGenerator CR 如下所述。为组 PolicyGenerator CR 提供变体，以考虑单节点、三节点紧凑和标准集群配置中的差别。同样，为单节点集群和多节点（compact 或 standard）集群提供了特定于站点的配置变体。使用与部署相关的组和特定于站点的配置变体。

表 9.2. RAN 部署的 PolicyGenerator CR

PolicyGenerator CR	描述
acm-example-multinode-site.yaml	包含一组应用于多节点集群的 CR。这些 CR 配置 SR-IOV 功能，用于 RAN 安装。
acm-example-sno-site.yaml	包含一组应用于单节点 OpenShift 集群的 CR。这些 CR 配置 SR-IOV 功能，用于 RAN 安装。
acm-common-mno-ranGen.yaml	包含一组应用于多节点集群的通用 RAN 策略配置。
acm-common-ranGen.yaml	包含一组应用于所有集群的通用 RAN CR。这些 CR 订阅一组 operator，提供典型的 RAN 和基准集群调整功能。
acm-group-du-3node-ranGen.yaml	仅包含三节点集群的 RAN 策略。
acm-group-du-sno-ranGen.yaml	仅包含单节点集群的 RAN 策略。
acm-group-du-standard-ranGen.yaml	包含标准三个 control-plane 集群的 RAN 策略。
acm-group-du-3node-validator-ranGen.yaml	用于生成三节点集群所需的各种策略的 **PolicyGenerator** CR。
acm-group-du-standard-validator-ranGen.yaml	用于生成标准集群所需的各种策略的 **PolicyGenerator** CR。
acm-group-du-sno-validator-ranGen.yaml	用于生成单节点 OpenShift 集群所需的各种策略的 **PolicyGenerator** CR。

其他资源

- 准备 GitOps ZTP 站点配置存储库

9.1.5. 使用 PolicyGenerator CR 自定义受管集群

使用以下步骤自定义应用于使用 GitOps Zero Touch Provisioning (ZTP) 管道置备的受管集群的策略。

先决条件

- 已安装 OpenShift CLI(oc)。

- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 配置了 hub 集群来生成所需的安装和策略 CR。
- 您创建了 Git 存储库，用于管理自定义站点配置数据。该存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源仓库。

流程

1. 为特定于站点的配置 CR 创建 PolicyGenerator CR。
   a. 从 out/argocd/example/acmpolicygenerator/ 文件夹中选择适当的 CR 示例，例如 acm-example-sno-site.yaml 或 acm-example-multinode-site.yaml。
   b. 更改示例文件中的 policyDefaults.placement.labelSelector 字段，以匹配 SiteConfig CR 中包含的特定于站点的标签匹配。在示例 SiteConfig 文件中，特定于站点的标签是 sites: example-sno。

   注意

   确保 PolicyGenerator policyDefaults.placement.labelSelector 字段中定义的标签与相关受管集群 SiteConfig CR 中定义的标签对应。

   c. 更改示例文件中的内容，使其与所需配置匹配。
2. 可选：为应用到整个集群的任何通用配置 CR 创建一个 PolicyGenerator CR。
   a. 从 out/argocd/example/acmpolicygenerator/ 文件夹中选择适当的 CR 示例，例如 acm-common-ranGen.yaml。
   b. 更改示例文件中的内容，使其与所需配置匹配。

3. 可选：为应用到团队中特定集群组的任何组配置 CR 创建一个 PolicyGenerator CR。

确保 overlaid spec 文件的内容与您的所需最终状态匹配。作为参考，out/source-crs 目录包含可用于包含和由 PolicyGenerator 模板提供的 source-crs 的完整列表。

注意

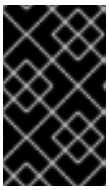
根据集群的特定要求，每个集群类型可能需要一个组策略，特别是考虑示例组策略各自有一个 PerformancePolicy.yaml 文件，如果这些集群是由相同的硬件配置，则只能在一组集群中共享。

a. 从 out/argocd/example/acmpolicygenerator/ 文件夹中选择适当的 CR 示例，例如 acm-group-du-sno-ranGen.yaml。

b. 更改示例文件中的内容，使其与所需配置匹配。

4. 可选。当 GitOps ZTP 安装和配置完成后，创建一个验证器通知策略 PolicyGenerator CR。如需更多信息，请参阅"创建验证器通知策略"。

5. 在 YAML 文件中定义所有策略命名空间，类似于示例 out/argocd/example/acmpolicygenerator//ns.yaml 文件。



重要

不要在带有 PolicyGenerator CR 的同一文件中包括 Namespace CR。

6. 将 PolicyGenerator CR 和 Namespace CR 添加到 generators 部分中的 kustomization.yaml 文件中，类似于 out/argocd/example/acmpolicygenerator/kustomization.yaml 所示的示例。

7. 在 Git 存储库中提交 PolicyGenerator CR、Namespace CR 和关联的 kustomization.yaml 文件并推送更改。

ArgoCD 管道检测到更改并开始受管集群部署。您可以同时将更改推送到 SiteConfig CR 和 PolicyGenerator CR。

其他资源

- 使用验证器通知策略信号 GitOps ZTP 集群部署完成

9.1.6. 监控受管集群策略部署进度

ArgoCD 管道使用 Git 中的 PolicyGenerator CR 来生成 RHACM 策略，然后将其同步到 hub 集群。您可以在辅助服务在受管集群中安装 OpenShift Container Platform 后监控受管集群策略同步的进度。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. Topology Aware Lifecycle Manager(TALM)应用绑定到集群的配置策略。

   集群安装完成后，集群变为 Ready，ClusterGroupUpgrade CR 对应于此集群，且由 run.openshift.io/ztp-deploy-wave annotations 定义的已排序策略列表由 TALM 自动创建。集群的策略按 ClusterGroupUpgrade CR 中列出的顺序应用。

   您可以使用以下命令监控配置策略协调的高级进度：

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[-1:]}' | jq
```

输出示例

```
{
 "lastTransitionTime": "2022-11-09T07:28:09Z",
 "message": "Remediating non-compliant policies",
 "reason": "InProgress",
```

```
  "status": "True",
  "type": "Progressing"
 }
```

2. 您可以使用 RHACM 仪表板或命令行监控详细的集群策略合规状态。

a. 要使用 oc 检查策略合规性，请运行以下命令：

```
$ oc get policies -n $CLUSTER
```

输出示例

```
NAME                                        REMEDIATION ACTION   COMPLIANCE
STATE   AGE
ztp-common.common-config-policy                  inform            Compliant
3h42m
ztp-common.common-subscriptions-policy           inform
NonCompliant       3h42m
ztp-group.group-du-sno-config-policy             inform            NonCompliant
3h42m
ztp-group.group-du-sno-validator-du-policy       inform            NonCompliant
3h42m
ztp-install.example1-common-config-policy-pjz9s  enforce           Compliant
167m
ztp-install.example1-common-subscriptions-policy-zzd9k   enforce
NonCompliant       164m
ztp-site.example1-config-policy                  inform            NonCompliant
3h42m
ztp-site.example1-perf-policy                    inform            NonCompliant
3h42m
```

b. 要从 RHACM Web 控制台检查策略状态，请执行以下操作：

i. 点 Governance → Find policies。

ii. 点集群策略检查其状态。

当所有集群策略都合规时，集群的 GitOps ZTP 安装和配置已完成。ztp-done 标签添加到集群中。

在引用配置中，合规的最终策略是 *-du-validator-policy 策略中定义的。此策略当在一个集群中合规时，确保所有集群配置、Operator 安装和 Operator 配置已完成。

9.1.7. 验证配置策略 CR 的生成

策略 自定义资源(CR)在与创建它们的 PolicyGenerator 相同的命名空间中生成。相同的故障排除流程适用于从 PolicyGenerator 生成的所有策略 CR，无论它们是 ztp-common、ztp-group 还是 ztp-site，如以下命令所示：

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

应该会显示预期的策略嵌套 CR 集合。

如果策略失败的同步，请使用以下故障排除步骤。

流程

1. 要显示策略的详细信息，请运行以下命令：

```
$ oc describe -n openshift-gitops application policies
```

2. 检查 Status: Conditions: 来显示错误日志。例如，将无效的 sourceFile 条目设置为 fileName: 生成以下错误：

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or
```

```
directory Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit
status 1: exit status 1
  Type:  ComparisonError
```

3. 检查 Status: Sync:。如果 Status: Conditions: 中存在日志错误，则 Status: Sync: 显示 Unknown 或 Error:

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  policies-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:           policies
        Repo URL:       https://git.com/ran-sites/policies/.git
        Target Revision:  master
    Status:             Error
```

4. 当 Red Hat Advanced Cluster Management(RHACM)识别策略应用到 ManagedCluster 对象时，策略 CR 对象应用到集群命名空间。检查策略是否已复制到集群命名空间中：

```
$ oc get policy -n $CLUSTER
```

输出示例：

```
NAME                                    REMEDIATION ACTION   COMPLIANCE STATE   AGE
ztp-common.common-config-policy             inform             Compliant         13d
ztp-common.common-subscriptions-policy      inform             Compliant         13d
ztp-group.group-du-sno-config-policy        inform             Compliant         13d
ztp-group.group-du-sno-validator-du-policy   inform             Compliant         13d
ztp-site.example-sno-config-policy          inform             Compliant         13d
```

RHACM 将所有适用的策略复制到集群命名空间中。复制的策略名称的格式有： <PolicyGenerator.Namespace>.<PolicyGenerator.Name>-<policyName>。

5. 检查放置规则中是否有没有复制到集群命名空间中的策略。这些策略 的放置 中的 matchSelector 应该与 ManagedCluster 对象上的标签匹配：

```
$ oc get Placement -n $NS
```

6. 使用以下命令，记录适合缺失策略、常见、组或站点的 放置 名称：

```
$ oc get Placement -n $NS <placement_rule_name> -o yaml
```

- status-decisions 应该包括集群名称。
- spec 中 matchSelector 的键值对必须与受管集群上的标签匹配。

7. 使用以下命令检查 ManagedCluster 对象上的标签：

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. 使用以下命令查看合规哪些策略：

```
$ oc get policy -n $CLUSTER
```

如果 Namespace、OperatorGroup 和 Subscription 策略兼容，但 Operator 配置策略不兼容，则 Operator 可能不会在受管集群中安装。这会导致 Operator 配置策略无法应用，因为 CRD 还没有应用到 spoke。

9.1.8. 重启策略协调

当发生意外合规问题时，您可以重启策略协调，例如 ClusterGroupUpgrade 自定义资源 (CR) 超时时。

流程

1. 在受管集群变为 Ready 后，Topology Aware Lifecycle Manager 在命名空间 ztp-install 中生成 ClusterGroupUpgrade CR：

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. 如果出现意外问题，且策略无法在配置超时（默认为 4 小时）内变为合规，ClusterGroupUpgrade CR 的状态会显示 UpgradeTimedOut ：

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[?(@.type=="Ready")]}'
```

3. UpgradeTimedOut 状态的 ClusterGroupUpgrade CR 每小时自动重启其策略协调。如果更改了策略，可以通过删除现有 ClusterGroupUpgrade CR 来启动立即重试。这会触发自动创建新的 ClusterGroupUpgrade CR，以开始立即协调策略：

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

请注意，当 ClusterGroupUpgrade CR 完成，其状态为 UpgradeCompleted，并且受管集群应用了 ztp-done 标签，您可以使用 PolicyGenerator 创建额外的配置更改。删除现有的 ClusterGroupUpgrade CR 将无法生成新的 CR。

此时，GitOps ZTP 完成了与集群的交互，任何进一步的交互都应被视为更新，并为补救策略创建新的 ClusterGroupUpgrade CR。

其他资源

- 有关使用 Topology Aware Lifecycle Manager (TALM)来构建自己的 ClusterGroupUpgrade CR 的详情，请参考关于 ClusterGroupUpgrade CR。

9.1.9. 使用策略更改应用的受管集群 CR

您可以通过策略从受管集群中部署的自定义资源(CR)中删除内容。

默认情况下，从 Policy Generator CR 创建的所有 Policy CR 都将 complianceType 字段设置为 musthave。没有删除内容的 musthave 策略仍然合规，因为受管集群上的 CR 具有所有指定的内容。使用这个配置，当从 CR 中删除内容时，TALM 从策略中删除内容，但不会从受管集群的 CR 中删除内容。

当 complianceType 字段为 mustonlyhave 时，策略可确保集群中的 CR 与策略中指定的内容完全匹配。

先决条件

- 已安装 OpenShift CLI(oc)。

- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已从运行 RHACM 的 hub 集群部署了受管集群。
- 您已在 hub 集群中安装了 Topology Aware Lifecycle Manager。

流程

1. 从受影响的 CR 中删除您不再需要的内容。在本例中，disableDrain: false 行已从 SriovOperatorConfig CR 中删除。

CR 示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    "node-role.kubernetes.io/$mcp": ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
```

2. 在 acm-group-du-sno-ranGen.yaml 文件中将受影响策略的 complianceType 更改为 mustonlyhave。

YAML 示例

```
# ...
policyDefaults:
  complianceType: "mustonlyhave"
# ...
policies:
  - name: config-policy
```

```
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: ""
    manifests:
      - path: source-crs/SriovOperatorConfig.yaml
```

3. 创建 ClusterGroupUpdates CR，并指定必须接收 CR 更改的集群：

ClusterGroupUpdates CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-remove
  namespace: default
spec:
  managedPolicies:
    - ztp-group.group-du-sno-config-policy
  enable: false
  clusters:
  - spoke1
  - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:
```

4. 运行以下命令来创建 ClusterGroupUpgrade CR：

```
$ oc create -f cgu-remove.yaml
```

5. 当您准备好应用更改时，例如在适当的维护窗口中，运行以下命令将 spec.enable 字段的值改为 true ：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

验证

1. 运行以下命令，检查策略的状态：

```
$ oc get <kind> <changed_cr_name>
```

输出示例

```
NAMESPACE   NAME                                        REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-ztp-group.group-du-sno-config-policy          enforce
17m
default     ztp-group.group-du-sno-config-policy              inform
NonCompliant       15h
```

当策略的 COMPLIANCE STATE 为 Compliant 时，这意味着已更新 CR，并删除不需要的内容。

2. 在受管集群中运行以下命令来检查策略是否已从目标集群中移除：

```
$ oc get <kind> <changed_cr_name>
```

如果没有结果，则会从受管集群中删除 CR。

9.1.10. 假定为 GitOps ZTP 安装

GitOps Zero Touch Provisioning (ZTP) 简化了检查集群的 GitOps ZTP 安装状态的过程。GitOps ZTP 状态分为三个阶段：集群安装、集群配置和 GitOps ZTP。

集群安装阶段

集群安装阶段由 ManagedCluster CR 中的 ManagedClusterJoined 和 ManagedClusterAvailable 条件显示。如果 ManagedCluster CR 没有这些条件，或者条件设置为 False，集群仍然处于安装阶段。有关安装的更多信息，请参阅 AgentClusterInstall 和 ClusterDeployment CR。如需更多信息，请参阅"Troubleshooting GitOps ZTP"。

集群配置阶段

集群配置阶段由 ztp-running 标签显示，在集群中应用 ManagedCluster CR。

完成 GitOps ZTP

集群安装和配置在 GitOps ZTP 完成。这可以通过删除 ztp-running 标签并在 ManagedCluster CR 中添加 ztp-done 标签来显示。ztp-done 标签显示应用了配置，基准 DU 配置已完成集群调整。

对 GitOps ZTP 完成的状态的更改是在 Red Hat Advanced Cluster Management (RHACM)验证器通知策略合规状态的条件。这个策略捕获了已完成的安装的现有条件，并确认只有在受管集群的 GitOps ZTP 置备完成后才会变为合规状态。

验证器通知策略可确保完全应用集群的配置，Operator 已完成初始化。策略验证以下内容：

- 目标 MachineConfigPool 包含预期的条目，并已完成更新。所有节点都可用，且没有降级。
- 至少有一个 SriovNetworkNodeState 带有 syncStatus: Succeeded 则代表 SR-IOV Operator 已完成初始化。
- PTP Operator 守护进程集已存在。

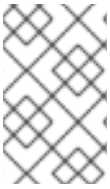
9.2. 带有 POLICYGENERATOR 资源的高级受管集群配置

您可以使用 PolicyGenerator CR 在受管集群中部署自定义功能。

重要

在 GitOps ZTP 中使用 PolicyGenerator 资源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。



注意

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

9.2.1. 为集群部署额外的更改

如果您需要在基本 GitOps Zero Touch Provisioning (ZTP) 管道配置之外更改集群配置，则有三个选项：

在 GitOps ZTP 管道完成后应用额外的配置

当 GitOps ZTP 管道部署完成后，部署的集群就可以用于应用程序工作负载。此时，您可以安装其他 Operator 并应用具体具体要求的配置。确保额外的配置不会影响平台或分配的 CPU 预算的性能。

在 GitOps ZTP 库中添加内容

使用 GitOps ZTP 管道部署的基本源自定义资源 (CR) 可以根据需要使用自定义内容增强。

为集群安装创建额外的清单

在安装过程中应用额外的清单，并使安装过程更高效。

重要

提供额外的源 CR 或修改现有源 CR 可能会影响 OpenShift Container Platform 的性能或 CPU 配置集。

其他资源

- 在 GitOps ZTP 管道中自定义额外的安装清单

9.2.2. 使用 PolicyGenerator CR 覆盖源 CR 内容

PolicyGenerator 自定义资源(CR)允许您覆盖 ztp-site-generate 容器中通过 GitOps 插件提供的基本源 CR 之上的额外配置详情。您可以将 PolicyGenerator CR 视为基础 CR 的逻辑合并或补丁。使用 PolicyGenerator CR 更新基本 CR 的单个字段，或覆盖基本 CR 的整个内容。您可以更新不在基本 CR 中的值和插入字段。

以下示例流程描述了如何根据 acm-group-du-sno-ranGen.yaml 文件中的 PolicyGenerator CR 为引用配置更新生成的 PerformanceProfile CR 中的字段。使用以下步骤根据您的要求修改

PolicyGenerator 的其他部分。

先决条件

- 创建一个 Git 存储库，在其中管理自定义站点配置数据。存储库必须可从 hub 集群访问，并定义为 Argo CD 的源存储库。

流程

1. 查看基准源 CR 以查找现有内容。您可以通过从 GitOps Zero Touch Provisioning (ZTP)容器中提取，来查看引用 PolicyGenerator CR 中列出的源 CR。

   a. 创建 /out 文件夹：

   ```
   $ mkdir -p ./out
   ```

   b. 提取源 CR：

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16.1 extract /home/ztp --tar | tar x -C ./out
   ```

2. 查看 ./out/source-crs/PerformanceProfile.yaml 中的基线 PerformanceProfile CR：

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
```

```
    net:
      userLevelNetworking: true
    nodeSelector:
      node-role.kubernetes.io/$mcp: ''
    numa:
      topologyPolicy: "restricted"
    realTimeKernel:
      enabled: true
```

注意

如果 PolicyGenerator CR 中未提供，包含 $... 的任何字段都会从生成的 CR 中删除。

3. 更新 acm-group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenerator 条目。以下示例 PolicyGenerator CR 小节提供了适当的 CPU 规格，设置 hugepages 配置，并添加一个新的字段，将 globallyDisableIrqLoadBalancing 设置为 false。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        # These must be tailored for the specific hardware platform
        cpu:
          isolated: "2-19,22-39"
          reserved: "0-1,20-21"
        hugepages:
          defaultHugepagesSize: 1G
          pages:
          - size: 1G
            count: 10
        globallyDisableIrqLoadBalancing: false
```

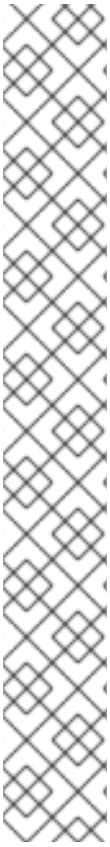
4. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

输出示例

GitOps ZTP 应用程序生成包含生成的 PerformanceProfile CR 的 RHACM 策略。该 CR 的内容通过将 PolicyGenerator 中的 PerformanceProfile 条目的 metadata 和 spec 内容合并到源 CR 中。生成的 CR 包含以下内容：

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: openshift-node-performance-profile
spec:
```

```
additionalKernelArgs:
  - idle=poll
  - rcupdate.rcu_normal_after_boot=0
cpu:
  isolated: 2-19,22-39
  reserved: 0-1,20-21
globallyDisableIrqLoadBalancing: false
hugepages:
  defaultHugepagesSize: 1G
  pages:
    - count: 10
      size: 1G
machineConfigPoolSelector:
  pools.operator.machineconfiguration.openshift.io/master: ""
net:
  userLevelNetworking: true
nodeSelector:
  node-role.kubernetes.io/master: ""
numa:
  topologyPolicy: restricted
realTimeKernel:
  enabled: true
```



注意

在从 ztp-site-generate 容器中提取的 /source-crs 文件夹中，$ 语法用于模板替换。相反，如果 policyGen 工具看到字符串的 $ 前缀，并且您没有在相关 PolicyGenerator CR 中为该字段指定值，则会完全从输出 CR 省略该字段。

一个例外是 /source-crs YAML 文件中的 $mcp 变量，该文件被替换为来自 PolicyGenerator CR 的 mcp 的指定的值。例如，在 example/policygentemplates/acm-group-du-standard-ranGen.yaml 中，mcp 的值为 worker ：

```
spec:
  bindingRules:
    group-du-standard: ""
  mcp: "worker"
```

policyGen 工具将输出 CR 中的 $mcp 实例替换为 worker。

9.2.3. 在 GitOps ZTP 管道中添加自定义内容

执行以下步骤在 GitOps ZTP 管道中添加新内容。

流程

1. 在目录中创建一个名为 source-crs 的子目录，其中包含 PolicyGenerator 自定义资源(CR)的 kustomization.yaml 文件。

2. 将用户提供的 CR 添加到 source-crs 子目录中，如下例所示：

```
example
└── acmpolicygenerator
    ├── dev.yaml
    ├── kustomization.yaml
    ├── mec-edge-sno1.yaml
    ├── sno.yaml
    └── source-crs 1
        ├── PaoCatalogSource.yaml
        ├── PaoSubscription.yaml
        ├── custom-crs
        │   ├── apiserver-config.yaml
        │   └── disable-nic-lldp.yaml
        └── elasticsearch
            ├── ElasticsearchNS.yaml
            └── ElasticsearchOperatorGroup.yaml
```

1 source-crs 子目录必须与 kustomization.yaml 文件位于同一个目录中。

3. 更新所需的 PolicyGenerator CR，使其包含对 source-crs/custom-crs 和 source-crs/elasticsearch 目录中添加的内容的引用。例如：

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
   name: group-dev
placementBindingDefaults:
   name: group-dev-placement-binding
policyDefaults:
   namespace: ztp-clusters
   placement:
      labelSelector:
         matchExpressions:
            - key: dev
              operator: In
              values:
                - "true"
   remediationAction: inform
   severity: low
   namespaceSelector:
      exclude:
         - kube-*
```

```
      include:
        - '*'
    evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
  - name: group-dev-group-dev-cluster-log-ns
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogNS.yaml
  - name: group-dev-group-dev-cluster-log-operator-group
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogOperGroup.yaml
  - name: group-dev-group-dev-cluster-log-sub
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogSubscription.yaml
  - name: group-dev-group-dev-lso-ns
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/StorageNS.yaml
  - name: group-dev-group-dev-lso-operator-group
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/StorageOperGroup.yaml
  - name: group-dev-group-dev-lso-sub
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/StorageSubscription.yaml
  - name: group-dev-group-dev-pao-cat-source
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/PaoSubscriptionCatalogSource.yaml
        patches:
          - spec:
              image: <container_image_url>
  - name: group-dev-group-dev-pao-ns
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/PaoSubscriptionNS.yaml
  - name: group-dev-group-dev-pao-sub
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/PaoSubscription.yaml
  - name: group-dev-group-dev-elasticsearch-ns
    policyAnnotations:
```

```
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: elasticsearch/ElasticsearchNS.yaml 1
  - name: group-dev-group-dev-elasticsearch-operator-group
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: elasticsearch/ElasticsearchOperatorGroup.yaml
  - name: group-dev-group-dev-apiserver-config
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: custom-crs/apiserver-config.yaml 2
  - name: group-dev-group-dev-disable-nic-lldp
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: custom-crs/disable-nic-lldp.yaml
```

1 2 设置 policies.manifests.path，使其包含 /source-crs 父目录中文件的相对路径。

4. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP Argo CD 策略应用程序监控的 Git 存储库。

5. 更新 ClusterGroupUpgrade CR，使其包含更改后的 PolicyGenerator，并将它保存为 cgu-test.yaml。以下示例显示了生成的 cgu-test.yaml 文件。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: custom-source-cr
  namespace: ztp-clusters
spec:
  managedPolicies:
    - group-dev-config-policy
  enable: true
  clusters:
  - cluster1
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

6. 运行以下命令来应用更新的 ClusterGroupUpgrade CR：

```
$ oc apply -f cgu-test.yaml
```

验证

- 运行以下命令检查更新是否成功：

```
$ oc get cgu -A
```

输出示例

```
NAMESPACE     NAME             AGE   STATE       DETAILS
ztp-clusters  custom-source-cr   6s    InProgress   Remediating non-compliant policies
ztp-install   cluster1          19h   Completed    All clusters are compliant with all the
managed policies
```

9.2.4. 为 PolicyGenerator CR 配置策略合规性评估超时

使用在 hub 集群上安装的 Red Hat Advanced Cluster Management (RHACM) 来监控和报告您的受管集群是否合规。RHACM 使用策略模板来应用预定义的策略控制器和策略。策略控制器是 Kubernetes 自定义资源定义（CRD）实例。

您可以使用 PolicyGenerator 自定义资源(CR)覆盖默认策略评估间隔。您可以配置持续时间设置，以定义 ConfigurationPolicy CR 在 RHACM 重新评估集群策略前处于策略合规或不合规的时长。

GitOps Zero Touch Provisioning (ZTP) 策略生成器使用预定义的策略评估间隔生成 ConfigurationPolicy CR 策略。noncompliant 状态的默认值为 10 秒。compliant 状态的默认值为 10 分钟。要禁用评估间隔，将值设为 never。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

流程

1. 要为 PolicyGenerator CR 中的所有策略配置评估间隔，请为 evaluationInterval 字段设置适当的 合规 和 不合规的 值。例如：

```
policyDefaults:
  evaluationInterval:
    compliant: 30m
    noncompliant: 45s
```

注意

您还可以将 合规 和 不合规 字段设置为 永远不会 在达到特定合规状态后停止评估策略。

2. 要在 PolicyGenerator CR 中为单个策略对象配置评估间隔，请添加 evaluationInterval 字段并设置适当的值。例如：

```
policies:
  - name: "sriov-sub-policy"
    manifests:
      - path: "SriovSubscription.yaml"
        evaluationInterval:
          compliant: never
          noncompliant: 10s
```

3. 在 Git 存储库中提交 PolicyGenerator CR 文件并推送您的更改。

验证

检查管理的 spoke 集群策略是否以预期间隔监控。

1. 在受管集群中以具有 cluster-admin 权限的用户身份登录。

2. 获取在 open-cluster-management-agent-addon 命名空间中运行的 pod。运行以下命令：

```
$ oc get pods -n open-cluster-management-agent-addon
```

输出示例

```
NAME                                READY   STATUS    RESTARTS       AGE
config-policy-controller-858b894c68-v4xdb   1/1     Running   22 (5d8h ago)   10d
```

3. 检查应用的策略是以 config-policy-controller pod 的日志中预期间隔评估：

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-
858b894c68-v4xdb
```

输出示例

```
2022-05-10T15:10:25.280Z    info    configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation
due to the policy not reaching the evaluation interval  {"policy": "compute-1-config-
policy-config"}
2022-05-10T15:10:25.280Z    info    configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation
due to the policy not reaching the evaluation interval  {"policy": "compute-1-common-
compute-1-catalog-policy-config"}
```

9.2.5. 使用验证器通知策略信号 GitOps ZTP 集群部署完成

创建一个验证器通知策略，在 GitOps Zero Touch Provisioning (ZTP) 安装和配置完成部署集群时信号。此策略可用于部署单节点 OpenShift 集群、三节点集群和标准集群。

流程

1. 创建包含源文件 validatorCR/informDuValidator.yaml 的独立 PolicyGenerator 自定义资源(CR)。每个集群类型只需要一个独立 PolicyGenerator CR。例如，此 CR 为单节点 OpenShift 集群应用验证器通知策略：

单节点集群验证器通知策略 CR (acm-group-du-sno-validator-ranGen.yaml)示例

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
   name: group-du-sno-validator-latest
placementBindingDefaults:
   name: group-du-sno-validator-latest-placement-binding
policyDefaults:
   namespace: ztp-group
   placement:
      labelSelector:
         matchExpressions:
            - key: du-profile
              operator: In
              values:
               - latest
            - key: group-du-sno
              operator: Exists
            - key: ztp-done
              operator: DoesNotExist
   remediationAction: inform
   severity: low
   namespaceSelector:
      exclude:
         - kube-*
      include:
         - '*'
   evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
   - name: group-du-sno-validator-latest-du-policy
     policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "10000"
     evaluationInterval:
      compliant: 5s
     manifests:
      - path: source-crs/validatorCRs/informDuValidator-MCP-master.yaml
```

2. 在 Git 存储库中提交 PolicyGenerator CR 文件并推送更改。

其他资源

- 升级 GitOps ZTP

9.2.6. 使用 PolicyGenerator CR 配置电源状态

对于低延迟和高性能部署，需要禁用或限制 C-states 和 P-states。使用这个配置，CPU 以恒定的频率运行，通常是最大 turbo 频率。这样可确保 CPU 始终以最大速度运行，这会导致高性能和低延迟。这会导致工作负载的最佳延迟。但是，这也会导致最高的功耗，这可能并不适用于所有工作负载。

工作负载可以归类为关键或非关键状态，需要为高性能和低延迟禁用 C-state 和 P-state 设置，而非关键工作负载在某些延迟和性能方面使用 C-state 和 P-state 设置。您可以使用 GitOps Zero Touch Provisioning (ZTP) 配置以下三个电源状态：

- 高性能模式以最高的功耗提供大量低延迟。
- 性能模式在相对高功耗时提供低延迟。
- 节能通过增加延迟来降低功耗。

默认配置用于低延迟性能模式。

PolicyGenerator 自定义资源(CR)允许您将额外的配置详情覆盖 ztp-site-generate 容器中由 GitOps 插件提供的基本源 CR。

根据 acm-group-du-sno-ranGen.yaml 中的 PolicyGenerator CR，为引用配置更新生成的 PerformanceProfile CR 中的 workloadHints 字段来配置电源状态。

以下常见先决条件适用于配置所有三个电源状态。

先决条件

- 您已创建了管理自定义站点配置数据的 Git 存储库。存储库必须可从 hub 集群访问，并定义为 Argo CD 的源存储库。
- 您已遵循"准备 GitOps ZTP 站点配置存储库"中所述的步骤。

其他资源

- 使用工作负载提示配置节点功耗和实时处理

9.2.6.1. 使用 PolicyGenerator CR 配置性能模式

按照以下示例，根据 acm-group-du-sno-ranGen.yaml 中的 PolicyGenerator CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置性能模式。

性能模式在相对高功耗时提供低延迟。

先决条件

- 您已按照"配置主机固件以实现低延迟和高性能"中的指导配置了与性能相关的 BIOS。

流程

1. 在 out/argocd/example/acmpolicygenerator// 中更新 acm-group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenerator 条目，以设置性能模式。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        workloadHints:
          realTime: true
          highPowerConsumption: false
          perPodPowerManagement: false
```

2. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP Argo CD 应用程序监控的 Git 存储库。

9.2.6.2. 使用 PolicyGenerator CR 配置高性能模式

按照以下示例，根据 acm-group-du-sno-ranGen.yaml 中的 PolicyGenerator CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置高性能模式。

高性能模式以最高的功耗提供大量低延迟。

先决条件

-

- 您已按照"配置主机固件以实现低延迟和高性能"中的指导配置了与性能相关的 BIOS。

流程

1. 在 out/argocd/example/acmpolicygenerator/ 中更新 acm-group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenerator 条目，以设置高性能模式。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
   - spec:
       workloadHints:
           realTime: true
           highPowerConsumption: true
           perPodPowerManagement: false
```

2. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP Argo CD 应用程序监控的 Git 存储库。

9.2.6.3. 使用 PolicyGenerator CR 配置节能模式

按照以下示例，根据 acm-group-du-sno-ranGen.yaml 中的 PolicyGenerator CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置节能模式。

节能模式会在增加延迟的情况下平衡功耗。

先决条件

- 您在 BIOS 中启用了 C-states 和 OS 控制的 P-states。

流程

1. 在 out/argocd/example/acmpolicygenerator/ 中更新 acm-group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenerator 条目，以配置节能模式。建议您通过额外的内核参数对象为节能模式配置 CPU 调控器。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
   - spec:
       # ...
       workloadHints:
         realTime: true
```

```
      highPowerConsumption: false
      perPodPowerManagement: true
    # ...
    additionalKernelArgs:
      - # ...
      - "cpufreq.default_governor=schedutil" 1
```

1

建议使用 schedutil governor，但您也可以使用其他 governor，包括 ondemand 和 powersave。

2. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP Argo CD 应用程序监控的 Git 存储库。

验证

1. 使用以下命令，从标识的节点列表中选择部署的集群中的 worker 节点：

```
$ oc get nodes
```

2. 使用以下命令登录到节点：

```
$ oc debug node/<node-name>
```

将 <node-name> 替换为您要验证电源状态的节点的名称。

3. 将 /host 设置为 debug shell 中的根目录。debug pod 在 pod 中的 /host 中挂载主机的 root 文件系统。通过将根目录改为 /host，您可以运行主机可执行路径中包含的二进制文件，如下例所示：

```
# chroot /host
```

4. 运行以下命令验证应用的电源状态：

```
# cat /proc/cmdline
```

预期输出

- 对于节能模式，intel_pstate=passive。

其他资源

- 为运行 colocated 高和低优先级工作负载的节点配置节能
- 为低延迟和高性能配置主机固件
- 准备 GitOps ZTP 站点配置存储库

9.2.6.4. 最大化节能

建议限制最大 CPU 频率，以实现最大节能。在非关键工作负载 CPU 中启用 C-states，而不会限制最大 CPU 频率，从而提高了关键 CPU 的频率。

通过更新 sysfs 插件字段来最大化节能，为参考配置的 Tuned PerformancePatch CR 中的 max_perf_pct 设置适当的值。这个示例基于 acm-group-du-sno-ranGen.yaml 描述了限制最大 CPU 频率的步骤。

先决条件

- 您已配置了节能模式，如 "Using PolicyGenerator CR to configure power savings mode" 所述。

流程

1. 更新 out/argocd/example/acmpolicygenerator/ 中的 acm-group-du-sno-ranGen.yaml 参考文件中的 Tuned PerformancePatch 的 PolicyGenerator 条目。要最大化节能，请添加 max_perf_pct，如下例所示：

```
- path: source-crs/TunedPerformancePatch.yaml
  patches:
    - spec:
      profile:
        - name: performance-patch
          data: |
            # ...
            [sysfs]
            /sys/devices/system/cpu/intel_pstate/max_perf_pct=<x> 1
```

1

max_perf_pct 控制 cpufreq 驱动程序的最大频率，以最大百分比的形式设置支持的 CPU 频率。这个值适用于所有 CPU。您可以检查 /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq 中的最大支持频率。作为起点，您可以使用以 All Cores Turbo 频率封装所有 CPU 的百分比。All Cores Turbo 频率是所有内核在内核完全占用时运行的频率。

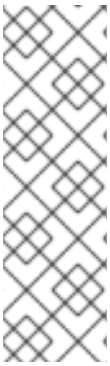
注意

要最大化节能，请设置一个较低的值。为 max_perf_pct 设置较低值会限制最大 CPU 频率，从而减少功耗，但可能会影响性能。试验不同的值并监控系统性能和功耗，以查找您的用例的最佳设置。

2. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP Argo CD 应用程序监控的 Git 存储库。

9.2.7. 使用 PolicyGenerator CR 配置 LVM 存储

您可以使用 GitOps Zero Touch Provisioning (ZTP)为部署的受管集群配置逻辑卷管理器(LVM)存储。



注意

当使用 PTP 事件或带有 HTTP 传输的裸机硬件事件时，您可以使用 LVM Storage 来保留事件订阅。

将 Local Storage Operator 用于在分布式单元中使用本地卷的持久性存储。

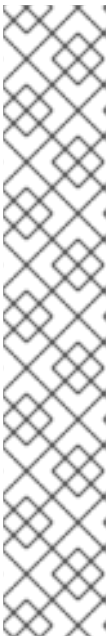
先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 创建一个 Git 存储库，在其中管理自定义站点配置数据。

流程

1. 要为新的受管集群配置 LVM 存储，请在 acm-common-ranGen.yaml 文件中的 policies.manifests 中添加以下 YAML：

```
- name: subscription-policies
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/StorageLVMOSubscriptionNS.yaml
    - path: source-crs/StorageLVMOSubscriptionOperGroup.yaml
    - path: source-crs/StorageLVMOSubscription.yaml
      spec:
        name: lvms-operator
        channel: stable-4.16
```



注意

Storage LVMO 订阅已弃用。在以后的 OpenShift Container Platform 版本中，存储 LVMO 订阅将不可用。反之，您必须使用 Storage LVMS 订阅。

在 OpenShift Container Platform 4.16 中，您可以使用 Storage LVMS 订阅而不是 LVMO 订阅。LVMS 订阅不需要在 acm-common-ranGen.yaml 文件中手动覆盖。将以下 YAML 添加到 acm-common-ranGen.yaml 文件中的 policies.manifests 以使用 Storage LVMS 订阅：

```
- path: source-crs/StorageLVMSubscriptionNS.yaml
- path: source-crs/StorageLVMSubscriptionOperGroup.yaml
- path: source-crs/StorageLVMSubscription.yaml
```

2. 将 LVMCluster CR 添加到特定组或单个站点配置文件中的 policies.manifests。例如，在 acm-group-du-sno-ranGen.yaml 文件中，添加以下内容：

```
- fileName: StorageLVMCluster.yaml
  policyName: "lvms-config"
    metadata:
      name: "lvms-storage-cluster-config"
        spec:
          storage:
            deviceClasses:
            - name: vg1
              thinPoolConfig:
                name: thin-pool-1
                sizePercent: 90
                overprovisionRatio: 10
```

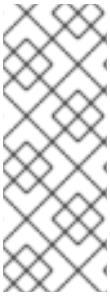
这个示例配置创建一个带有所有可用设备的卷组 (vg1)，但安装了 OpenShift Container Platform 的磁盘除外。也创建了一个精简池逻辑卷。

3. 将任何其他必要的更改和文件与自定义站点存储库合并。

4. 提交 Git 中的 PolicyGenerator 更改，然后将更改推送到您的站点配置存储库，以使用 GitOps ZTP 将 LVM 存储部署到新站点。

9.2.8. 使用 PolicyGenerator CR 配置 PTP 事件

您可以使用 GitOps ZTP 管道来配置使用 HTTP 或 AMQP 传输的 PTP 事件。



注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

9.2.8.1. 配置使用 HTTP 传输的 PTP 事件

您可以配置使用 GitOps Zero Touch Provisioning (ZTP)管道部署的受管集群中使用 HTTP 传输的 PTP 事件。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 根据您的要求，将以下 PolicyGenerator 应用到 acm-group-du-3node-ranGen.yaml,acm-group-du-sno-ranGen.yaml, 或 acm-group-du-standard-ranGen.yaml 文

件：

a. 在 policies.manifests 中，添加 PtpOperatorConfig CR 文件来配置传输主机：

```
- path: source-crs/PtpOperatorConfigForEvent.yaml
  patches:
  - metadata:
      name: default
      namespace: openshift-ptp
      annotations:
        ran.openshift.io/ztp-deploy-wave: "10"
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/$mcp: ""
      ptpEventConfig:
        enableEventPublisher: true
        transportHost: "http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043"
```

注意

在 OpenShift Container Platform 4.13 或更高版本中，在使用带有 PTP 事件的 HTTP 传输时，您不需要在 PtpOperatorConfig 资源中设置 transportHost 字段。

b. 为 PTP 时钟类型和接口配置 linuxptp 和 phc2sys。例如，将以下 YAML 添加到 policies.manifests 中：

```
- path: source-crs/PtpConfigSlave.yaml (1)
  patches:
  - metadata:
      name: "du-ptp-slave"
    spec:
      recommend:
      - match:
        - nodeLabel: node-role.kubernetes.io/master
        priority: 4
        profile: slave
      profile:
      - name: "slave"
        # This interface must match the hardware in this group
        interface: "ens5f0" (2)
        ptp4lOpts: "-2 -s --summary_interval -4" (3)
        phc2sysOpts: "-a -r -n 24" (4)
        ptpSchedulingPolicy: SCHED_FIFO
        ptpSchedulingPriority: 10
        ptpSettings:
```

```
    logReduce: "true"
  ptp4lConf: |
    [global]
    #
    # Default Data Set
    #
    twoStepFlag 1
    slaveOnly 1
    priority1 128
    priority2 128
    domainNumber 24
    #utc_offset 37
    clockClass 255
    clockAccuracy 0xFE
    offsetScaledLogVariance 0xFFFF
    free_running 0
    freq_est_interval 1
    dscp_event 0
    dscp_general 0
    dataset_comparison G.8275.x
    G.8275.defaultDS.localPriority 128
    #
    # Port Data Set
    #
    logAnnounceInterval -3
    logSyncInterval -4
    logMinDelayReqInterval -4
    logMinPdelayReqInterval -4
    announceReceiptTimeout 3
    syncReceiptTimeout 0
    delayAsymmetry 0
    fault_reset_interval -4
    neighborPropDelayThresh 20000000
    masterOnly 0
    G.8275.portDS.localPriority 128
    #
    # Run time options
    #
    assume_two_step 0
    logging_level 6
    path_trace_enabled 0
    follow_up_info 0
    hybrid_e2e 0
    inhibit_multicast_service 0
    net_sync_monitor 0
    tc_spanning_tree 0
    tx_timestamp_timeout 50
    unicast_listen 0
    unicast_master_table 0
    unicast_req_duration 3600
    use_syslog 1
    verbose 0
    summary_interval 0
    kernel_leap 1
    check_fup_sync 0
    clock_class_threshold 7
```

```
    #
    # Servo Options
    #
    pi_proportional_const 0.0
    pi_integral_const 0.0
    pi_proportional_scale 0.0
    pi_proportional_exponent -0.3
    pi_proportional_norm_max 0.7
    pi_integral_scale 0.0
    pi_integral_exponent 0.4
    pi_integral_norm_max 0.3
    step_threshold 2.0
    first_step_threshold 0.00002
    max_frequency 900000000
    clock_servo pi
    sanity_freq_limit 200000000
    ntpshm_segment 0
    #
    # Transport options
    #
    transportSpecific 0x0
    ptp_dst_mac 01:1B:19:00:00:00
    p2p_dst_mac 01:80:C2:00:00:0E
    udp_ttl 1
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type OC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0xA0
  ptpClockThreshold: 5
    holdOverTimeout: 30 # seconds
    maxOffsetThreshold: 100  # nano seconds
    minOffsetThreshold: -100
```

1

可以是 PtpConfigMaster.yaml、PtpConfigSlave.yaml 或 PtpConfigSlaveCvl.yaml 之一，具体取决于您的要求。PtpConfigSlaveCvl.yaml 为

Intel E810 Columbiaville NIC 配置 linuxptp 服务。对于基于 acm-group-du-sno-ranGen.yaml 或 acm-group-du-3node-ranGen.yaml 的配置，请使用 PtpConfigSlave.yaml。

2 特定于设备的接口名称。

3 您必须将 --summary_interval -4 值附加到 .spec.sourceFiles.spec.profile 中的 ptp4lOpts 中，以启用 PTP fast 事件。

4 所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

5 可选。如果 ptpClockThreshold 小节不存在，则默认值用于 ptpClockThreshold 字段。小节显示默认的 ptpClockThreshold 值。ptpClockThreshold 值配置 PTP master 时钟在触发 PTP 事件前的时长。holdOverTimeout 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 FREERUN 前的时间值（以秒为单位）。maxOffsetThreshold 和 minOffsetThreshold 设置以纳秒为单位，它们与 CLOCK_REALTIME (phc2sys) 或 master 偏移 (ptp4l) 的值进行比较。当 ptp4l 或 phc2sys 偏移值超出这个范围时，PTP 时钟状态被设置为 FREERUN。当偏移值在这个范围内时，PTP 时钟状态被设置为 LOCKED。

2. 将任何其他必要的更改和文件与自定义站点存储库合并。

3. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将 PTP 快速事件部署到新站点。

其他资源

- 使用 PolicyGenerator CR 覆盖源 CR 内容

9.2.8.2. 配置使用 AMQP 传输的 PTP 事件

您可以在使用 GitOps Zero Touch Provisioning (ZTP) 管道部署的受管集群中配置使用 AMQP 传输的 PTP 事件。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 将以下 YAML 添加到 acm-common-ranGen.yaml 文件中的 policies.manifests 中，以配置 AMQP Operator：

```
#AMQ Interconnect Operator for fast events
- path: source-crs/AmqSubscriptionNS.yaml
- path: source-crs/AmqSubscriptionOperGroup.yaml
- path: source-crs/AmqSubscription.yaml
```

2. 根据您的要求，将以下 PolicyGenerator 应用到 acm-group-du-3node-ranGen.yaml,acm-group-du-sno-ranGen.yaml, 或 acm-group-du-standard-ranGen.yaml 文件：

   a. 在 policies.manifests 中，添加 PtpOperatorConfig CR 文件，该文件将 AMQ 传输主机配置为 config-policy ：

```
- path: source-crs/PtpOperatorConfigForEvent.yaml
  patches:
  - metadata:
      name: default
      namespace: openshift-ptp
      annotations:
        ran.openshift.io/ztp-deploy-wave: "10"
    spec:
      daemonNodeSelector:
```

```
      node-role.kubernetes.io/$mcp: ""
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: "amqp://amq-router.amq-router.svc.cluster.local"
```

b. 为 PTP 时钟类型和接口配置 linuxptp 和 phc2sys。例如，将以下 YAML 添加到 policies.manifests 中：

```
- path: source-crs/PtpConfigSlave.yaml (1)
  patches:
  - metadata:
      name: "du-ptp-slave"
    spec:
      recommend:
      - match:
        - nodeLabel: node-role.kubernetes.io/master
        priority: 4
        profile: slave
      profile:
      - name: "slave"
        # This interface must match the hardware in this group
        interface: "ens5f0" (2)
        ptp4lOpts: "-2 -s --summary_interval -4" (3)
        phc2sysOpts: "-a -r -n 24" (4)
        ptpSchedulingPolicy: SCHED_FIFO
        ptpSchedulingPriority: 10
        ptpSettings:
          logReduce: "true"
        ptp4lConf: |
          [global]
          #
          # Default Data Set
          #
          twoStepFlag 1
          slaveOnly 1
          priority1 128
          priority2 128
          domainNumber 24
          #utc_offset 37
          clockClass 255
          clockAccuracy 0xFE
          offsetScaledLogVariance 0xFFFF
          free_running 0
          freq_est_interval 1
          dscp_event 0
          dscp_general 0
          dataset_comparison G.8275.x
          G.8275.defaultDS.localPriority 128
          #
          # Port Data Set
          #
          logAnnounceInterval -3
          logSyncInterval -4
```

```
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
```

```
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
    ptpClockThreshold: 5
      holdOverTimeout: 30 # seconds
      maxOffsetThreshold: 100  # nano seconds
      minOffsetThreshold: -100
```

1

可以是 PtpConfigMaster.yaml、PtpConfigSlave.yaml 或 PtpConfigSlaveCvl.yaml 之一，具体取决于您的要求。PtpConfigSlaveCvl.yaml 为 Intel E810 Columbiaville NIC 配置 linuxptp 服务。对于基于 acm-group-du-sno-ranGen.yaml 或 acm-group-du-3node-ranGen.yaml 的配置，请使用 PtpConfigSlave.yaml。

2

特定于设备的接口名称。

3

您必须将 --summary_interval -4 值附加到 .spec.sourceFiles.spec.profile 中的 ptp4lOpts 中，以启用 PTP fast 事件。

4

所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

5

可选。如果 ptpClockThreshold 小节不存在，则默认值用于 ptpClockThreshold 字段。小节显示默认的 ptpClockThreshold

值。ptpClockThreshold 值配置 PTP master 时钟在触发 PTP 事件前的时长。holdOverTimeout 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 FREERUN 前的时间值（以秒为单位）。maxOffsetThreshold 和 minOffsetThreshold 设置以纳秒为单位，它们与 CLOCK_REALTIME (phc2sys) 或 master 偏移 (ptp4l) 的值进行比较。当 ptp4l 或 phc2sys 偏移值超出这个范围时，PTP 时钟状态被设置为 FREERUN。当偏移值在这个范围内时，PTP 时钟状态被设置为 LOCKED。

3. 将以下 PolicyGenerator 更改应用到特定的站点 YAML 文件，如 acm-example-sno-site.yaml ：

   a. 在 policies.manifests 中，添加 Interconnect CR 文件，该文件将 AMQ 路由器配置为 config-policy ：

   ```
   - path: source-crs/AmqInstance.yaml
   ```

4. 将任何其他必要的更改和文件与自定义站点存储库合并。

5. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将 PTP 快速事件部署到新站点。

其他资源

- 安装 AMQ 消息传递总线
- OpenShift 镜像 registry 概述

9.2.9. 使用 PolicyGenerator CR 配置裸机事件

您可以使用 GitOps ZTP 管道来配置使用 HTTP 或 AMQP 传输的裸机事件。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

9.2.9.1. 配置使用 HTTP 传输的裸机事件

您可以配置使用 GitOps Zero Touch Provisioning (ZTP)管道部署的受管集群中使用 HTTP 传输的裸机事件。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 通过在 acm-common-ranGen.yaml 文件中的 policies.manifests 中添加以下 YAML 来配置 Bare Metal Event Relay Operator：

```
# Bare Metal Event Relay Operator
- path: source-crs/BareMetalEventRelaySubscriptionNS.yaml
- path: source-crs/BareMetalEventRelaySubscriptionOperGroup.yaml
- path: source-crs/BareMetalEventRelaySubscription.yaml
```

2. 将 HardwareEvent CR 添加到特定组配置文件中的 policies.manifests，例如在 acm-group-du-sno-ranGen.yaml 文件中：

```
- path: source-crs/HardwareEvent.yaml (1)
  patches:
    - spec:
        logLevel: debug
        nodeSelector: {}
        transportHost: http://hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043
```

(1) 每个基板管理控制器 (BMC) 只需要一个 HardwareEvent CR。

注意

在 OpenShift Container Platform 4.13 或更高版本中，当将 HTTP 传输用于裸机事件时，您不需要在 HardwareEvent 自定义资源 (CR) 中设置 transportHost 字段。

3. 将任何其他必要的更改和文件与自定义站点存储库合并。

4. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将裸机事件部署到新站点。

5. 运行以下命令来创建 Redfish Secret：

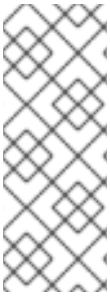
```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

其他资源

- 使用 CLI 安装裸机事件中继
- 创建裸机事件和 Secret CR

9.2.9.2. 配置使用 AMQP 传输的裸机事件

您可以在使用 GitOps Zero Touch Provisioning (ZTP) 管道部署的受管集群中配置使用 AMQP 传输的裸机事件。



注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 要配置 AMQ Interconnect Operator 和 Bare Metal Event Relay Operator，请在 acm-common-ranGen.yaml 文件中的 policies.manifests 中添加以下 YAML：

```
# AMQ Interconnect Operator for fast events
- path: source-crs/AmqSubscriptionNS.yaml
- path: source-crs/AmqSubscriptionOperGroup.yaml
- path: source-crs/AmqSubscription.yaml
# Bare Metal Event Relay Operator
- path: source-crs/BareMetalEventRelaySubscriptionNS.yaml
- path: source-crs/BareMetalEventRelaySubscriptionOperGroup.yaml
- path: source-crs/BareMetalEventRelaySubscription.yaml
```

2. 将 Interconnect CR 添加到站点配置文件中的 policies.manifests，例如 acm-example-sno-site.yaml 文件：

```
- path: source-crs/AmqInstance.yaml
```

3. 将 HardwareEvent CR 添加到特定组配置文件中的 policies.manifests，例如在 acm-group-du-sno-ranGen.yaml 文件中：

```
- path: HardwareEvent.yaml
  patches:
    nodeSelector: {}
    transportHost: "amqp://<amq_interconnect_name>.
<amq_interconnect_namespace>.svc.cluster.local" 1
    logLevel: "info"
```

1. transportHost URL 由现有的 AMQ Interconnect CR 名称和命名空间组成。例如，在 transportHost: "amq-router.amq-router.svc.cluster.local" 中，AMQ Interconnect name 和 namespace 都被设置为 amq-router。

注意

每个基板管理控制器 (BMC) 仅需要一个 HardwareEvent 资源。

4. 提交 Git 中的 PolicyGenerator 更改，然后将更改推送到您的站点配置存储库，以使用 GitOps ZTP 将裸机事件监控部署到新站点。

5. 运行以下命令来创建 Redfish Secret：

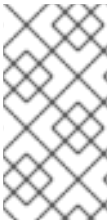
```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

9.2.10. 配置 Image Registry Operator 以进行镜像的本地缓存

OpenShift Container Platform 使用本地 registry 管理镜像缓存。在边缘计算用例中，集群通常会受到带宽限制，与集中式镜像 registry 通信时，这可能会导致长时间镜像下载时间。

在初始部署期间，长时间下载时间不可避免。随着时间的推移，CRI-O 会在出现意外关闭时擦除 /var/lib/containers/storage 目录的风险。要解决镜像下载时间长的问题，您可以使用 GitOps Zero Touch Provisioning (ZTP) 在远程受管集群上创建本地镜像 registry。当集群部署在网络边缘时，这非常有用。

在使用 GitOps ZTP 设置本地镜像 registry 前，您需要在用于安装远程受管集群的 SiteConfig CR 中配置磁盘分区。安装后，您可以使用 PolicyGenerator CR 配置本地镜像 registry。然后，GitOps ZTP 管道创建持久性卷 (PV) 和持久性卷声明(PVC) CR，并修补 imageregistry 配置。



注意

本地镜像 registry 只能用于用户应用程序镜像，不能用于 OpenShift Container Platform 或 Operator Lifecycle Manager operator 镜像。

其他资源

- OpenShift Container Platform registry 概述

9.2.10.1. 使用 SiteConfig 配置磁盘分区

使用 SiteConfig CR 和 GitOps Zero Touch Provisioning (ZTP) 为受管集群配置磁盘分区。SiteConfig CR 中的磁盘分区详情必须与底层磁盘匹配。

重要

您必须在安装时完成这个步骤。

先决条件

- 安装 Butane。

流程

1. 创建 storage.bu 文件。

```
variant: fcos
version: 1.3.0
storage:
  disks:
  - device: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0 (1)
    wipe_table: false
    partitions:
    - label: var-lib-containers
      start_mib: <start_of_partition> (2)
      size_mib: <partition_size> (3)
  filesystems:
    - path: /var/lib/containers
      device: /dev/disk/by-partlabel/var-lib-containers
      format: xfs
      wipe_filesystem: true
      with_mount_unit: true
      mount_options:
        - defaults
        - prjquota
```

(1) 指定根磁盘。

(2) 以 MiB 为单位指定分区的起始位置。如果值太小，安装会失败。

(3)

指定分区的大小。如果值太小，部署会失败。

2. 运行以下命令，将 storage.bu 转换为 Ignition 文件：

```
$ butane storage.bu
```

输出示例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# # Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-partlabel/var-lib-containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. 使用 JSON Pretty Print 等工具将输出转换为 JSON 格式。

4. 将输出复制到 SiteConfig CR 中的 .spec.clusters.nodes.ignitionConfigOverride 字段中。

Example

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: |
          {
            "ignition": {
              "version": "3.2.0"
            },
```

```
          "storage": {
            "disks": [
              {
                "device": "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0",
                "partitions": [
                  {
                    "label": "var-lib-containers",
                    "sizeMiB": 0,
                    "startMiB": 250000
                  }
                ],
                "wipeTable": false
              }
            ],
            "filesystems": [
              {
                "device": "/dev/disk/by-partlabel/var-lib-containers",
                "format": "xfs",
                "mountOptions": [
                  "defaults",
                  "prjquota"
                ],
                "path": "/var/lib/containers",
                "wipeFilesystem": true
              }
            ]
          },
          "systemd": {
            "units": [
              {
                "contents": "# # Generated by Butane\n[Unit]\nRequires=systemd-
fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-
fsck@dev-disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
                "enabled": true,
                "name": "var-lib-containers.mount"
              }
            ]
          }
        }
[...]
```

注意

如果 **.spec.clusters.nodes.ignitionConfigOverride** 字段不存在，请创建它。

验证

1. 在安装过程中，运行以下命令来在 hub 集群上验证 BareMetalHost 对象显示注解：

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-install.openshift.io/ignition-config-overrides"]
```

输出示例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":[{\"device\":\"/dev/disk/by-partlabel/var-lib-containers\",\"format\":\"xfs\",\"mountOptions\":[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"systemd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-fsck@dev-disk-by\\\\x2dpartlabel-var\\\\x2dlib\\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-by\\\\x2dpartlabel-var\\\\x2dlib\\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/dev/disk/by-partlabel/var-lib-containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. 安装后，检查单节点 OpenShift 磁盘状态。

a. 运行以下命令，在单节点 OpenShift 节点上进入 debug 会话。此步骤被实例化为一个名为 <node_name>-debug 的 debug pod:

```
$ oc debug node/my-sno-node
```

b. 运行以下命令，将 /host 设置为 debug shell 中的根目录。debug pod 在 pod 中的 /host 中挂载主机的 root 文件系统。将根目录改为 /host，您可以运行主机可执行路径中包含的二进制文件：

```
# chroot /host
```

c. 运行以下命令，列出所有可用块设备的信息：

```
# lsblk
```

输出示例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                      /sysroot/ostree/deploy/rhcos/var
│                      /usr
│                      /etc
│                      /
│                      /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

d. 运行以下命令，显示文件系统磁盘空间使用情况的信息：

```
# df -h
```

输出示例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

9.2.10.2. 使用 PolicyGenerator CR 配置镜像 registry

使用 PolicyGenerator (PGT) CR 应用配置镜像 registry 所需的 CR，并修补 imageregistry 配置。

先决条件

- 您已在受管集群中配置了磁盘分区。
- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了 Git 存储库，在其中管理自定义站点配置数据以用于 GitOps Zero Touch Provisioning (ZTP)。

流程

1. 在适当的 PolicyGenerator CR 中配置存储类、持久性卷声明、持久性卷和镜像 registry 配置。例如，要配置单个站点，请将以下 YAML 添加到文件 acm-example-sno-site.yaml 中：

```
sourceFiles:
  # storage class
  - fileName: StorageClass.yaml
    policyName: "sc-for-image-registry"
    metadata:
      name: image-registry-sc
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100" (1)
  # persistent volume claim
  - fileName: StoragePVC.yaml
    policyName: "pvc-for-image-registry"
    metadata:
      name: image-registry-pvc
      namespace: openshift-image-registry
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
      storageClassName: image-registry-sc
      volumeMode: Filesystem
  # persistent volume
  - fileName: ImageRegistryPV.yaml (2)
    policyName: "pv-for-image-registry"
    metadata:
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
```

```
- fileName: ImageRegistryConfig.yaml
  policyName: "config-for-image-registry"
  complianceType: musthave
  metadata:
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
  spec:
    storage:
      pvc:
        claim: "image-registry-pvc"
```

1

根据您要在站点、通用或组级别配置镜像 registry，为 ztp-deploy-wave 设置适当的值。ZTP-deploy-wave: "100" 适用于开发或测试，因为它允许您将引用的源文件分组到一起。

2

在 ImageRegistryPV.yaml 中，确保将 spec.local.path 字段设置为 /var/imageregistry，以匹配 SiteConfig CR 中为 mount_point 字段设置的值。

重要

不要为 - fileName: ImageRegistryConfig.yaml 配置设置 complianceType: mustonlyhave。这可能导致 registry pod 部署失败。

2. 提交 Git 中的 PolicyGenerator 更改，然后推送到由 GitOps ZTP ArgoCD 应用程序监控的 Git 存储库。

验证

使用以下步骤排除受管集群中本地镜像 registry 的错误：

- 在登录到受管集群时，验证是否成功登录到 registry。运行以下命令：

a. 导出受管集群名称：

```
$ cluster=<managed_cluster_name>
```

'

b. 获取受管集群 kubeconfig 详情：

```
$ oc get secret -n $cluster $cluster-admin-password -o
jsonpath='{.data.password}' | base64 -d > kubeadmin-password-$cluster
```

c. 下载并导出集群 kubeconfig ：

```
$ oc get secret -n $cluster $cluster-admin-kubeconfig -o
jsonpath='{.data.kubeconfig}' | base64 -d > kubeconfig-$cluster && export
KUBECONFIG=./kubeconfig-$cluster
```

d. 验证从受管集群访问镜像 registry。请参阅"访问 registry"。

- 检查 imageregistry.operator.openshift.io 组实例的 Config CRD 是否没有报告错误。登录到受管集群时运行以下命令：

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

输出示例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2021-10-08T19:02:39Z"
  generation: 5
  name: cluster
  resourceVersion: "688678648"
  uid: 0406521b-39c0-4cda-ba75-873697da75a4
spec:
  additionalTrustedCA:
    name: acm-ice
```

- 检查受管集群上的 PersistentVolumeClaim 是否填充了数据。登录到受管集群时运行以下命令：

```
$ oc get pv image-registry-sc
```

- 检查 registry* pod 是否正在运行，并位于 openshift-image-registry 命名空间下。

```
$ oc get pods -n openshift-image-registry | grep registry*
```

输出示例

```
cluster-image-registry-operator-68f5c9c589-42cfg   1/1     Running     0          8d
image-registry-5f8987879-6nx6h                     1/1     Running     0          8d
```

- 检查受管集群中的磁盘分区是否正确：

a. 为受管集群打开默认 shell:

```
$ oc debug node/sno-1.example.com
```

b. 运行 lsblk 以检查主机磁盘分区：

```
sh-4.4# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
  |-sda1   8:1    0     1M  0 part
  |-sda2   8:2    0   127M  0 part
  |-sda3   8:3    0   384M  0 part /boot
  |-sda4   8:4    0 336.3G  0 part /sysroot
  `-sda5   8:5    0 100.1G  0 part /var/imageregistry (1)
sdb      8:16   0 446.6G  0 disk
sr0     11:0    1   104M  0 rom
```

(1) /var/imageregistry 表示磁盘已被正确分区。

其他资源

- 访问registry

9.3. 使用 POLICYGENERATOR 资源和 TALM 在断开连接的环境中更新受管集群

您可以使用 Topology Aware Lifecycle Manager (TALM)来管理使用 GitOps Zero Touch Provisioning (ZTP)和 Topology Aware Lifecycle Manager (TALM)部署的受管集群的软件生命周期。TALM 使用 Red Hat Advanced Cluster Management (RHACM) PolicyGenerator 策略来管理和控制应用到目标集群的更改。

重要

在 GitOps ZTP 中使用 PolicyGenerator 资源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。

其他资源

- 有关 Topology Aware Lifecycle Manager 的更多信息，请参阅关于 Topology Aware Lifecycle Manager。

9.3.1. 设置断开连接的环境

TALM 可以同时执行平台和 Operator 更新。

您必须在镜像 registry 中镜像您要升级到的平台镜像和 Operator 镜像，然后才能使用 TALM 更新断开连接的集群。完成以下步骤以镜像镜像：

- 对于平台更新，您必须执行以下步骤：

1. 镜像所需的 OpenShift Container Platform 镜像存储库。根据"镜像 OpenShift Container Platform 镜像存储库"流程在附加资源中链接，确保所需的平台镜像已被镜像。在 imageContentSources.yaml 文件中保存 imageContentSources 部分的内容：

输出示例

```
imageContentSources:
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-release
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. 保存已镜像的所需平台镜像的镜像签名。您必须将镜像签名添加到用于平台更新的 PolicyGenerator CR 中。要获取镜像签名，请执行以下步骤：

   a. 运行以下命令指定所需的 OpenShift Container Platform 标签：

   ```
   $ OCP_RELEASE_NUMBER=<release_version>
   ```

   b. 运行以下命令指定集群的构架：

   ```
   $ ARCHITECTURE=<cluster_architecture> 1
   ```

   1 指定集群的构架，如 x86_64, aarch64, s390x, 获 ppc64le。

   c. 运行以下命令，从 Quay 获取发行版本镜像摘要

   ```
   $ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-
   release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From:
   .*@//p')"
   ```

   d. 运行以下命令来设置摘要算法：

   ```
   $ DIGEST_ALGO="${DIGEST%%:*}"
   ```

e. 运行以下命令来设置摘要签名：

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. 运行以下命令，从 mirror.openshift.com 网站获取镜像签名：

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

g. 运行以下命令，将镜像签名保存到 checksum-<OCP_RELEASE_NUMBER>.yaml 文件中：

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 准备更新图表。您可以通过两个选项来准备更新图形：

a. 使用 OpenShift Update Service。

有关如何在 hub 集群上设置图形的更多信息，请参阅为 OpenShift Update Service 部署 Operator 并构建图形数据 init 容器。

b. 生成上游图形的本地副本。在可访问受管集群的断开连接的环境中的 http 或 https 服务器上托管更新图表。要下载更新图表，请使用以下命令：

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.16 -o ~/upgrade-graph_stable-4.16
```

- 对于 Operator 更新，您必须执行以下任务：
  - 镜像 Operator 目录。确保所需的 Operator 镜像按照"Mirroring Operator 目录以用于断开连接的集群"部分中的步骤进行镜像。

其他资源

-

有关如何更新 GitOps Zero Touch Provisioning (ZTP) 的更多信息，请参阅升级 GitOps ZTP。

- 有关如何镜像 OpenShift Container Platform 镜像存储库的更多信息，请参阅 镜像 OpenShift Container Platform 镜像存储库。
- 有关如何为断开连接的集群镜像 Operator 目录的更多信息，请参阅 镜像 Operator 目录以用于断开连接的集群。
- 有关如何准备断开连接的环境并镜像所需的镜像存储库的更多信息，请参阅准备断开连接的环境。
- 有关更新频道和发行版本的更多信息，请参阅了解更新频道和发行版本。

9.3.2. 使用 PolicyGenerator CR 执行平台更新

您可以使用 TALM 执行平台更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 镜像所需的镜像存储库。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 为平台更新创建一个 PolicyGenerator CR：

   a. 将以下 PolicyGenerator CR 保存到 du-upgrade.yaml 文件中：

平台更新的 PolicyGenerator 示例

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-platform-upgrade
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "100"
    manifests:
      - path: source-crs/ClusterVersion.yaml (1)
        patches:
          - metadata:
              name: version
            spec:
              channel: stable-4.16
              desiredUpdate:
                version: 4.16.4
              upstream: http://upgrade.example.com/images/upgrade-graph_stable-
4.16
            status:
              history:
                - state: Completed
                  version: 4.16.4
  - name: du-upgrade-platform-upgrade-prep
```

```
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "1"
  manifests:
    - path: source-crs/ImageSignature.yaml 2
    - path: source-crs/DisconnectedICSP.yaml
      patches:
        - metadata:
            name: disconnected-internal-icsp-for-ocp
          spec:
            repositoryDigestMirrors: 3
              - mirrors:
                  - quay-intern.example.com/ocp4/openshift-release-dev
                source: quay.io/openshift-release-dev/ocp-release
              - mirrors:
                  - quay-intern.example.com/ocp4/openshift-release-dev
                source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1

显示触发更新的 ClusterVersion CR。对于预缓存， channel, upstream, 和 desiredVersion 项都是必需的。

2

ImageSignature.yaml 包含所需的发行镜像的镜像签名。镜像签名用于在应用平台更新前验证镜像。

3

显示包含所需 OpenShift Container Platform 镜像的镜像存储库。获取在"设置 environment"部分中的步骤时所保存的 imageContentSources.yaml 文件中的镜像。

PolicyGenerator CR 生成两个策略：

- du-upgrade-platform-upgrade-prep 策略为平台更新做准备。它为所需的发行版本镜像签名创建 ConfigMap CR，创建镜像的发行镜像存储库的镜像内容源，并使用所需的更新频道更新集群版本，以及在断开连接的环境中由 spoke 集群访问的更新图。
- du-upgrade-platform-upgrade 策略用于执行平台升级。

b. 将 du-upgrade.yaml 文件内容添加到 kustomization.yaml 文件中，位于

PolicyGenerator CR 的 GitOps ZTP Git 存储库中，并将更改推送到 Git 存储库。

ArgoCD 从 Git 存储库拉取更改并在 hub 集群上生成策略。

c. 运行以下命令检查创建的策略：

```
$ oc get policies -A | grep platform-upgrade
```

2. 为平台更新创建 ClusterGroupUpdate CR，将 spec.enable 项设置为 false。

a. 将平台更新 ClusterGroupUpdate CR 的内容，带有 du-upgrade-platform-upgrade-prep 和 du-upgrade-platform-upgrade 策略，以及目标集群保存到 cgu-platform-upgrade.yml 文件，如以下示例所述：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

b. 运行以下命令，将 ClusterGroupUpdate CR 应用到 hub 集群：

```
$ oc apply -f cgu-platform-upgrade.yml
```

3. 可选：缓存平台更新的镜像。

a. 运行以下命令，在 ClusterGroupUpdate CR 中启用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 监控更新过程，并等待预缓存完成。在 hub 集群中运行以下命令来检查预缓存的状态：

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

4. 启动平台更新：

a. 运行以下命令启用 cgu-platform-upgrade 策略并禁用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

其他资源

- 有关在断开连接的环境中镜像镜像的更多信息，请参阅准备断开连接的环境。

9.3.3. 使用 PolicyGenerator CR 执行 Operator 更新

您可以使用 TALM 执行 Operator 更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 镜像捆绑包镜像、捆绑包镜像以及捆绑包镜像中引用的所有 Operator 镜像。

- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 为 Operator 更新更新 PolicyGenerator CR。

   a. 使用 du-upgrade.yaml 文件中的以下额外内容更新 du-upgrade PolicyGenerator CR：

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-operator-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
          - metadata:
              name: redhat-operators
            spec:
              displayName: Red Hat Operators Catalog
              image: registry.example.com:5000/olm/redhat-operators:v4.16 (1)
              updateStrategy: (2)
                registryPoll:
```

```
      interval: 1h
  status:
    connectionState:
      lastObservedState: READY 3
```

1

包含所需的 Operator 镜像。如果索引镜像始终推送到相同的镜像名称和标签，则不需要此更改。

2

设置 Operator Lifecycle Manager (OLM)使用 registryPoll.interval 字段轮询新 Operator 版本的索引镜像的频率。如果为 y-stream 和 z-stream Operator 更新而总是推送新的索引镜像标签，则不需要此更改。registryPoll.interval 字段可以设置为较短的间隔，以加快更新，但较短的间隔会增大计算负载。要影响这个问题，您可以在更新完成后将 registryPoll.interval 恢复到默认值。

3

显示目录连接的观察状态。READY 值确保 CatalogSource 策略已就绪，表示索引 Pod 已拉取并在运行。这样，TALM 根据最新的策略合规性状态升级 Operator。

b.

在这个版本中，生成一个策略 du-upgrade-operator-catsrc-policy，以使用包含所需 Operator 镜像的新索引镜像更新 redhat-operators 目录源。

注意

如果要使用 Operator 预缓存，并且有来自 redhat-operators 以外的其他目录源的 Operator，您必须执行以下任务：

- 使用新的索引镜像或 registry 轮询间隔更新准备单独的目录源策略。
- 为来自不同目录源的所需 Operator 准备单独的订阅策略。

例如，所需的 SRIOV-FEC Operator 在 certified-operators 目录源中提供。要更新目录源和 Operator 订阅，请添加以下内容来生成两个策略： du-upgrade-fec-catsrc-policy 和 du-upgrade-subscriptions-fec-policy ：

```
apiVersion: policy.open-cluster-management.io/v1
```

```
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-fec-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
          - metadata:
              name: certified-operators
            spec:
              displayName: Intel SRIOV-FEC Operator
              image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
              updateStrategy:
                registryPoll:
                  interval: 10m
  - name: du-upgrade-subscriptions-fec-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/AcceleratorsSubscription.yaml
        patches:
          - spec:
              channel: stable
              source: certified-operators
```

c.

如果存在，在通用 PolicyGenerator CR 中删除指定的订阅频道。GitOps ZTP 镜像的默认订阅频道用于更新。

注意

通过 GitOps ZTP 4.16 应用的 Operator 的默认频道是 stable，但 performance-addon-operator 除外。从 OpenShift Container Platform 4.11 开始，performance-addon-operator 功能被移到 node-tuning-operator 中。对于 4.10 发行版本，PAO 的默认频道是 v4.10。您还可以在 common PolicyGenerator CR 中指定默认频道。

d. 将 PolicyGenerator CR 更新推送到 GitOps ZTP Git 存储库。

ArgoCD 从 Git 存储库拉取更改并在 hub 集群上生成策略。

e. 运行以下命令检查创建的策略：

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. 在启动 Operator 更新前，应用所需的目录源更新。

a. 使用目录源策略将名为 operator-upgrade-prep 的 ClusterGroupUpgrade CR 的内容保存到 cgu-operator-upgrade-prep.yml 文件中：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

b. 运行以下命令，将策略应用到 hub 集群：

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

c. 监控更新过程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. 为 Operator 更新创建 ClusterGroupUpgrade CR，并将 spec.enable 字段设置为 false。

a. 使用 du-upgrade-operator-catsrc-policy 策略和从通用 PolicyGenerator 创建的订阅策略，将 Operator 更新 ClusterGroupUpgrade CR 的内容保存到 cgu-operator-upgrade.yml 文件中，如下例所示：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy (1)
  - common-subscriptions-policy (2)
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

(1) 镜像预缓存功能需要该策略，以便从目录源检索 Operator 镜像。

(2) 策略包含 Operator 订阅。如果您遵循了参考 PolicyGenTemplates 的结构和内容，则所有 Operator 订阅都分组到 common-subscriptions-policy 策略中。

注意

一个 ClusterGroupUpgrade CR 只能从 ClusterGroupUpgrade CR 中包含的一个目录源中预缓存订阅策略中定义的 Operator 镜像。如果所需的 Operator 来自不同目录源，如 SRIOV-FEC Operator 示例，则必须使用 du-upgrade-fec-catsrc-policy 和 du-upgrade-subscriptions-fec-policy 镜像（pre-FEC Operator 镜像）创建另一个 ClusterGroupUpgrade CR。

b. 运行以下命令，将 ClusterGroupUpgrade CR 应用到 hub 集群：

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. 可选：缓存 Operator 更新的镜像。

   a. 在启动镜像预缓存前，运行以下命令验证订阅策略在此时是否是 NonCompliant ：

   ```
   $ oc get policy common-subscriptions-policy -n <policy_namespace>
   ```

   输出示例

   ```
   NAME                         REMEDIATION ACTION   COMPLIANCE STATE     AGE
   common-subscriptions-policy   inform              NonCompliant         27d
   ```

   b. 运行以下命令，在 ClusterGroupUpgrade CR 中启用预缓存：

   ```
   $ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
   operator-upgrade \
   --patch '{"spec":{"preCaching": true}}' --type=merge
   ```

   c. 监控进程并等待预缓存完成。在受管集群中运行以下命令来检查预缓存的状态：

   ```
   $ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
   ```

   d. 运行以下命令，检查预缓存是否在启动更新前完成：

   ```
   $ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
   ```

   输出示例

   ```
   [
     {
       "lastTransitionTime": "2022-03-08T20:49:08.000Z",
       "message": "The ClusterGroupUpgrade CR is not enabled",
       "reason": "UpgradeNotStarted",
   ```

```
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-03-08T20:55:30.000Z",
      "message": "Precaching is completed",
      "reason": "PrecachingCompleted",
      "status": "True",
      "type": "PrecachingDone"
    }
]
```

5. 启动 Operator 更新。

a. 运行以下命令，启用 cgu-operator-upgrade ClusterGroupUpgrade CR，并禁用预缓存来启动 Operator 更新：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

其他资源

- 有关更新 GitOps ZTP 的更多信息，请参阅升级 GitOps ZTP。

9.3.4. 使用 PolicyGenerator CR 对丢失的 Operator 更新进行故障排除

在某些情况下，Topology Aware Lifecycle Manager (TALM)可能会因为过时的策略合规状态而丢失 Operator 更新。

在目录源更新后，Operator Lifecycle Manager (OLM)需要时间来更新订阅状态。当 TALM 决定是否需要补救时，订阅策略的状态可能会继续显示为合规。因此，订阅策略中指定的 Operator 不会升级。

要避免这种情况，请将另一个目录源配置添加到 PolicyGenerator 中，并为需要更新的任何 Operator

在订阅中指定此配置。

流程

1. 在 PolicyGenerator 资源中添加目录源配置：

```
manifests:
- path: source-crs/DefaultCatsrc.yaml
  patches:
    - metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v{product-version}
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY
- path: source-crs/DefaultCatsrc.yaml
  patches:
    - metadata:
        name: redhat-operators-v2 (1)
      spec:
        displayName: Red Hat Operators Catalog v2 (2)
        image: registry.example.com:5000/olredhat-operators:<version> (3)
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY
```

(1) 更新新配置的名称。

(2) 更新新配置的显示名称。

(3) 更新索引镜像 URL。此 policies.manifests.patches.spec.image 字段覆盖 DefaultCatsrc.yaml 文件中的任何配置。

2. 更新 Subscription 资源，以指向需要更新的 Operator 的新配置：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: operator-subscription
  namespace: operator-namspace
# ...
spec:
  source: redhat-operators-v2 1
# ...
```

1. 输入您在 PolicyGenerator 资源中定义的额外目录源配置的名称。

9.3.5. 一起执行平台和 Operator 更新

您可以同时执行平台和 Operator 更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 按照 "forming a platform update" 和 "Performing an Operator update" 部分所述的步骤为更新创建 PolicyGenerator CR。

2. 为平台和 Operator 更新应用准备工作。

   a. 使用平台更新准备工作、目录源更新和目标集群的 ClusterGroupUpgrade CR 将内容保存到 cgu-platform-operator-upgrade-prep.yml 文件中，例如：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

   b. 运行以下命令，将 cgu-platform-operator-upgrade-prep.yml 文件应用到 hub 集群：

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

   c. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

3. 为平台创建 ClusterGroupUpdate CR，并将 spec.enable 字段设置为 false 的 Operator 更新。

   a. 将平台的内容和带有策略和目标集群的 Operator 更新 ClusterGroupUpdate CR 保存为 cgu-platform-operator-upgrade.yml 文件，如下例所示：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade 1
  - du-upgrade-operator-catsrc-policy 2
```

```
- common-subscriptions-policy 3
preCaching: true
clusterSelector:
- group-du-sno
remediationStrategy:
  maxConcurrency: 1
enable: false
```

1 这是平台更新策略。

2 这是包含要更新 Operator 的目录源信息的策略。预缓存功能需要它来确定要下载至受管集群的 Operator 镜像。

3 这是更新 Operator 的策略。

b. 运行以下命令，将 cgu-platform-operator-upgrade.yml 文件应用到 hub 集群：

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. 可选：为平台和 Operator 更新缓存镜像。

a. 运行以下命令，在 ClusterGroupUpgrade CR 中启用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 监控更新过程，并等待预缓存完成。在受管集群中运行以下命令来检查预缓存的状态：

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

c. 运行以下命令，检查预缓存是否在启动更新前完成：

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5. 启动平台和 Operator 更新。

a. 运行以下命令，启用 cgu-du-upgrade ClusterGroupUpgrade CR 来启动平台和 Operator 更新：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

注意

可通过将设置配置为 spec.enable: true，从开始创建平台和 Operator 更新 CR。在这种情况下，更新会在预缓存完成后立即启动，且不需要手动启用 CR。

预缓存和更新都创建额外的资源，如策略、放置规则、放置规则、受管集群操作和受管集群视图，以帮助完成这个过程。将 afterCompletion.deleteObjects 字段设置为 true 在更新完成后删除所有这些资源。

9.3.6. 使用 PolicyGenerator CR 从部署的集群中删除 Performance Addon Operator 订阅

在早期版本的 OpenShift Container Platform 中，Performance Addon Operator 为应用程序提供了自动、低延迟的性能调整。在 OpenShift Container Platform 4.11 或更高版本中，这些功能是 Node Tuning Operator 的一部分。

不要在运行 OpenShift Container Platform 4.11 或更高版本的集群中安装 Performance Addon Operator。如果您升级到 OpenShift Container Platform 4.11 或更高版本，Node Tuning Operator 会自动删除 Performance Addon Operator。

注意

您需要删除创建 Performance Addon Operator 订阅的任何策略，以防止重新安装 Operator。

参考 DU 配置集在 PolicyGenerator CR acm-common-ranGen.yaml 中包含 Performance Addon Operator。要从部署的受管集群中删除订阅，您必须更新 acm-common-ranGen.yaml。

注意

如果在 OpenShift Container Platform 4.11 或更高版本上安装 Performance Addon Operator 4.10.3-5 或更高版本，Performance Addon Operator 会检测到集群版本并自动休眠，以避免与 Node Tuning Operator 正常工作。但是，为了确保获得最佳性能，请从 OpenShift Container Platform 4.11 集群中删除 Performance Addon Operator。

先决条件

- 创建一个 Git 存储库，在其中管理自定义站点配置数据。存储库必须可从 hub 集群访问，并定义为 Argo CD 的源存储库。
- 更新至 OpenShift Container Platform 4.11 或更高版本。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在 acm-common-ranGen.yaml 文件中，将 Performance Addon Operator 命名空间、Operator 组和订阅的 complianceType 更改为 mustnothave。

```
- name: group-du-sno-pg-subscriptions-policy
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/PaoSubscriptionNS.yaml
    - path: source-crs/PaoSubscriptionOperGroup.yaml
    - path: source-crs/PaoSubscription.yaml
```

2. 将更改与自定义站点存储库合并，并等待 ArgoCD 应用程序对 hub 集群同步更改。common-subscriptions-policy 策略的状态更改为 Non-Compliant。
3. 使用 Topology Aware Lifecycle Manager 将更改应用到您的目标集群。有关滚动配置更改的更多信息，请参阅“附加资源”部分。
4.

监控进程。当目标集群的 common-subscriptions-policy 策略的状态为 Compliant 时，Performance Addon Operator 已从集群中移除。运行以下命令，获取 common-subscriptions-policy 的状态：

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

5. 从 acm-common-ranGen.yaml 文件中的 policies.manifests 中删除 Performance Addon Operator 命名空间、Operator 组和订阅 CR。

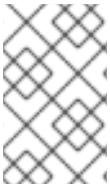
6. 将更改与自定义站点存储库合并，并等待 ArgoCD 应用程序对 hub 集群同步更改。策略保持合规。

9.3.7. 在单节点 OpenShift 集群中使用 TALM 预缓存用户指定的镜像

在升级应用程序前，您可以在单节点 OpenShift 集群上预缓存应用程序相关的工作负载镜像。

您可以使用以下自定义资源(CR)指定预缓存作业的配置选项：

- PreCachingConfig CR
- ClusterGroupUpgrade CR



注意

PreCachingConfig CR 中的所有字段都是可选的。

PreCachingConfig CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: exampleconfig-ns
spec:
  overrides: 1
    platformImage: quay.io/openshift-release-dev/ocp-
```

```
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
   operatorsIndexes:
    - registry.example.com:5000/custom-redhat-operators:1.0.0
   operatorsPackagesAndChannels:
    - local-storage-operator: stable
    - ptp-operator: stable
    - sriov-network-operator: stable
  spaceRequired: 30 Gi 2
  excludePrecachePatterns: 3
   - aws
   - vsphere
  additionalImages: 4
   -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808f
c925ada29c559a47e2e1ef
   -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808f
c925ada29c559a47adfaef
   -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafde
dga0991234afdadfsa09
```

1

默认情况下，TALM 会自动填充 platformImage、operatorIndexes 和受管集群策略中的 operatorsPackagesAndChannels 字段。您可以指定值来覆盖这些字段的默认 TALM-derived 值。

2

指定集群上的最低磁盘空间。如果未指定，TALM 为 OpenShift Container Platform 镜像定义一个默认值。磁盘空间字段必须包含整数值和存储单元。例如：40 GiB、200 MB、1 TiB。

3

根据镜像名称匹配，指定要从预缓存中排除的镜像。

4

指定要预缓存的额外镜像列表。

带有 PreCachingConfig CR 引用的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
```

```
kind: ClusterGroupUpgrade
metadata:
  name: cgu
spec:
  preCaching: true (1)
  preCachingConfigRef:
    name: exampleconfig (2)
    namespace: exampleconfig-ns (3)
```

(1) preCaching 字段设置为 true 可启用预缓存作业。

(2) preCachingConfigRef.name 字段指定您要使用的 PreCachingConfig CR。

(3) preCachingConfigRef.namespace 指定您要使用的 PreCachingConfig CR 的命名空间。

9.3.7.1. 为预缓存创建自定义资源

您必须在 ClusterGroupUpgrade CR 之前或同时创建 PreCachingConfig CR。

1. 使用您要预缓存的额外镜像列表创建 PreCachingConfig CR。

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: default (1)
spec:
[...]
  spaceRequired: 30Gi (2)
  additionalImages:
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
```

```
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340
adfafdedga0991234afdadfsa09
```

1 namespace 必须可以被 hub 集群访问。

2 建议设置最小磁盘空间所需字段，以确保预缓存镜像有足够的存储空间。

2. 创建一个 ClusterGroupUpgrade CR，并将 preCaching 字段设置为 true 并指定上一步中创建的 PreCachingConfig CR：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
  namespace: default
spec:
  clusters:
  - sno1
  - sno2
  preCaching: true
  preCachingConfigRef:
  - name: exampleconfig
    namespace: default
  managedPolicies:
    - du-upgrade-platform-upgrade
    - du-upgrade-operator-catsrc-policy
    - common-subscriptions-policy
  remediationStrategy:
    timeout: 240
```

警告

在集群上安装镜像后，您无法更改或删除它们。

3. 当您要启动预缓存镜像时，请运行以下命令应用 ClusterGroupUpgrade CR：

```
$ oc apply -f cgu.yaml
```

TALM 验证 ClusterGroupUpgrade CR。

此时，您可以继续 TALM 预缓存工作流。

注意

所有站点都同时预缓存。

验证

1. 运行以下命令，检查应用 ClusterUpgradeGroup CR 的 hub 集群上的预缓存状态：

```
$ oc get cgu <cgu_name> -n <cgu_namespace> -oyaml
```

输出示例

```
 precaching:
   spec:
     platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2
e1ef
     operatorsIndexes:
       - registry.example.com:5000/custom-redhat-operators:1.0.0
     operatorsPackagesAndChannels:
       - local-storage-operator: stable
       - ptp-operator: stable
       - sriov-network-operator: stable
     excludePrecachePatterns:
       - aws
       - vsphere
     additionalImages:
       -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47e2e1ef
       -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47adfaef
       -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340
adfafdedga0991234afdadfsa09
     spaceRequired: "30"
```

```
  status:
    sno1: Starting
    sno2: Starting
```

通过检查受管策略是否存在预缓存配置来验证预缓存配置。ClusterGroupUpgrade 和 PreCachingConfig CR 的有效配置会导致以下状态：

有效 CR 的输出示例

```
- lastTransitionTime: "2023-01-01T00:00:01Z"
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: "True"
  type: ClusterSelected
- lastTransitionTime: "2023-01-01T00:00:02Z"
  message: Completed validation
  reason: ValidationCompleted
  status: "True"
  type: Validated
- lastTransitionTime: "2023-01-01T00:00:03Z"
  message: Precaching spec is valid and consistent
  reason: PrecacheSpecIsWellFormed
  status: "True"
  type: PrecacheSpecValid
- lastTransitionTime: "2023-01-01T00:00:04Z"
  message: Precaching in progress for 1 clusters
  reason: InProgress
  status: "False"
  type: PrecachingSucceeded
```

无效的 PreCachingConfig CR 示例

```
Type:    "PrecacheSpecValid"
Status:  False,
Reason:  "PrecacheSpecIncomplete"
Message: "Precaching spec is incomplete: failed to get PreCachingConfig resource
due to PreCachingConfig.ran.openshift.io "<pre-caching_cr_name>" not found"
```

2. 您可以在受管集群中运行以下命令来查找预缓存作业：

```
$ oc get jobs -n openshift-talo-pre-cache
```

预缓存作业正在进行的示例

```
NAME        COMPLETIONS      DURATION      AGE
pre-cache   0/1              1s            1s
```

3. 您可以运行以下命令来检查为预缓存作业创建的 pod 状态：

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

预缓存作业正在进行的示例

```
Type        Reason              Age    From              Message
Normal      SuccesfulCreate     19s    job-controller    Created pod: pre-cache-abcd1
```

4. 您可以运行以下命令来获取作业状态的实时更新：

```
$ oc logs -f pre-cache-abcd1 -n openshift-talo-pre-cache
```

5. 要验证预缓存作业是否已成功完成，请运行以下命令：

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

完成的预缓存作业示例

```
Type      Reason           Age   From            Message
Normal    SuccesfulCreate  5m19s job-controller  Created pod: pre-cache-abcd1
Normal    Completed        19s   job-controller  Job completed
```

6. 要验证镜像是否在单节点 OpenShift 上成功预缓存，请执行以下操作：

   a. 以 debug 模式进入节点：

   ```
   $ oc debug node/cnfdf00.example.lab
   ```

   b. 将 root 更改为 host ：

   ```
   $ chroot /host/
   ```

   c. 搜索所需的镜像：

   ```
   $ sudo podman images | grep <operator_name>
   ```

其他资源

- 有关 TALM 预缓存工作流的更多信息，请参阅使用容器镜像预缓存功能。

9.3.8. 关于为 GitOps ZTP 自动创建的 ClusterGroupUpgrade CR

TALM 有一个名为 ManagedClusterForCGU 的控制器，它监控 hub 集群上的 ManagedCluster CR 的 Ready 状态，并为 GitOps Zero Touch Provisioning (ZTP) 创建 ClusterGroupUpgrade CR。

对于没有应用 ztp-done 标签的 Ready 状态中的任何受管集群，ManagedClusterForCGU 控制器会在 ztp-install 命名空间中创建一个带有在 GitOps ZTP 进程中创建的关联 RHACM 策略的 ClusterGroupUpgrade CR。然后，TALM 会修复自动创建 ClusterGroupUpgrade CR 中列出的一组配置策略，将配置 CR 推送到受管集群。

如果集群变为 Ready 时，没有用于受管集群的策略，则会创建一个没有策略的 ClusterGroupUpgrade CR。完成 ClusterGroupUpgrade 受管集群后，受管集群被标记为 ztp-done。如果要对该受管集群应用策略，请手动创建一个 ClusterGroupUpgrade 作为第 2 天操作。

GitOps ZTP 自动创建的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" (1)
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" (2)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

(1)

当 TALM 完成集群配置时，应用到受管集群。

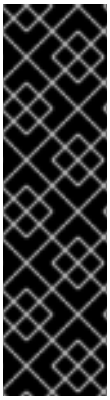
2

当 TALM 开始部署配置策略时，应用到受管集群。

第 10 章 使用 POLICYGENTEMPLATE 资源管理集群策略

10.1. 使用 POLICYGENTEMPLATE 资源配置受管集群策略

应用的 Policy 自定义资源(CR)配置您置备的受管集群。您可以自定义 Red Hat Advanced Cluster Management (RHACM)如何使用 PolicyGenTemplate CR 生成应用的 Policy CR。



重要

使用 PolicyGenTemplate CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 PolicyGenerator CR 提供了等效和改进的功能。

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

10.1.1. 关于 PolicyGenTemplate CRD

PolicyGenTemplate 自定义资源定义(CRD) 告知 PolicyGen 策略生成器在集群配置中包含哪些自定义资源 (CR)，如何将 CR 组合到生成的策略中，以及这些 CR 中的项目需要使用 overlay 内容更新。

以下示例显示了从 ztp-site-generate 引用容器中提取的 PolicyGenTemplate CR (common-du-ranGen.yaml)。common-du-ranGen.yaml 文件定义了两个 Red Hat Advanced Cluster Management (RHACM) 策略。策略管理配置 CR 集合，每个 CR 中的 policyName 值对应一个。common-du-ranGen.yaml 创建一个单个放置绑定和一个放置规则，根据 spec.bindingRules 部分中列出的标签将策略绑定到集群。

示例 PolicyGenTemplate CR - common-ranGen.yaml

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
```

```
  name: "common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true" (1)
    du-profile: "latest"
  sourceFiles: (2)
    - fileName: SriovSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogNS.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageNS.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: DefaultCatsrc.yaml (3)
      policyName: "config-policy" (4)
      metadata:
        name: redhat-operators-disconnected
      spec:
        displayName: disconnected-redhat-operators
        image: registry.example.com:5000/disconnected-redhat-operators/disconnected-redhat-
operator-index:v4.9
    - fileName: DisconnectedICSP.yaml
      policyName: "config-policy"
      spec:
        repositoryDigestMirrors:
        - mirrors:
          - registry.example.com:5000
          source: registry.redhat.io
```

1

common: "true" 将策略应用到具有此标签的所有集群。

2

sourceFiles 下列出的文件为已安装的集群创建 Operator 策略。

3

DefaultCatsrc.yaml 配置断开连接的 registry 的目录源。

4

policyName: "config-policy" 配置 Operator 订阅。OperatorHub CR 禁用默认值，此 CR 将 redhat-operators 替换为指向断开连接的 registry 的 CatalogSource CR。

PolicyGenTemplate CR 可以使用任意数量的包含 CR 来构建。在 hub 集群中应用以下示例 CR 来生成包含单个 CR 的策略：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno"
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
    - fileName: PtpConfigSlave.yaml
      policyName: "config-policy"
      metadata:
        name: "du-ptp-slave"
      spec:
        profile:
        - name: "slave"
          interface: "ens5f0"
          ptp4lOpts: "-2 -s --summary_interval -4"
          phc2sysOpts: "-a -r -n 24"
```

使用源文件 PtpConfigSlave.yaml 作为示例，文件会定义一个 PtpConfig CR。为 PtpConfigSlave 示例生成的策略名为 group-du-sno-config-policy。生成的 group-du-sno-config-policy 中定义的

PtpConfig CR 被命名为 du-ptp-slave。PtpConfigSlave.yaml 中定义的 spec 放置在 du-ptp-slave 下，以及与源文件中定义的其他 spec 项目一起放置。

以下示例显示了 group-du-sno-config-policy CR：

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: group-du-ptp-config-policy
  namespace: groups-sub
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
spec:
    remediationAction: inform
    disabled: false
    policy-templates:
      - objectDefinition:
          apiVersion: policy.open-cluster-management.io/v1
          kind: ConfigurationPolicy
          metadata:
            name: group-du-ptp-config-policy-config
          spec:
            remediationAction: inform
            severity: low
            namespaceselector:
              exclude:
                - kube-*
              include:
                - '*'
            object-templates:
              - complianceType: musthave
                objectDefinition:
                  apiVersion: ptp.openshift.io/v1
                  kind: PtpConfig
                  metadata:
                    name: du-ptp-slave
                    namespace: openshift-ptp
                  spec:
                    recommend:
                      - match:
                      - nodeLabel: node-role.kubernetes.io/worker-du
                        priority: 4
                        profile: slave
                    profile:
                      - interface: ens5f0
                        name: slave
                        phc2sysOpts: -a -r -n 24
                        ptp4lConf: |
                          [global]
                          #
                          # Default Data Set
```

```
                #
                twoStepFlag 1
                slaveOnly 0
                priority1 128
                priority2 128
                domainNumber 24
```

10.1.2. 在自定义 PolicyGenTemplate CR 时建议

在自定义站点配置 PolicyGenTemplate 自定义资源 (CR) 时，请考虑以下最佳实践：

- 根据需要使用一些策略。使用较少的策略需要较少的资源。每个附加策略会为 hub 集群和部署的受管集群创建 CPU 负载增加。CR 根据 PolicyGenTemplate CR 中的 policyName 字段合并到策略中。同一 PolicyGenTemplate 中的 CR，在单个策略下管理相同的 policyName 值。

- 在断开连接的环境中，通过将 registry 配置为包含所有 Operator 的单个索引，为所有 Operator 使用单个目录源。受管集群中的每个额外 CatalogSource CR 会增加 CPU 用量。

- MachineConfig CR 应包含在 siteConfig CR 中作为 extraManifests，以便在安装过程中应用它们。这可减少在集群就绪部署应用程序前所花费的总时间。

- PolicyGenTemplate CR 应该覆盖 channel 字段来显式识别所需的版本。这样可确保源 CR 在升级过程中的更改不会更新生成的订阅。

其他资源

- 有关使用 RHACM 扩展集群的建议，请参阅性能和可扩展性。

注意

在 hub 集群中管理大量 spoke 集群时，请最小化策略数量来减少资源消耗。

将多个配置 CR 分组到单个或有限的策略中，一种方法是减少 hub 集群上的总体策略数量。在使用 common, group, 和 site 层次结构来管理站点配置时，务必要将特定于站点的配置组合成单一策略。

10.1.3. RAN 部署的 PolicyGenTemplate CR

使用 PolicyGenTemplate 自定义资源(CR)使用 GitOps Zero Touch Provisioning (ZTP)管道自定义应用到集群的配置。PolicyGenTemplate CR 允许您生成一个或多个策略来管理集群中的配置 CR 集合。PolicyGenTemplate CR 标识一组受管 CR，将它们捆绑到策略中，构建与这些 CR 相关的策略，并使用标签绑定规则将策略与集群相关联。

从 GitOps ZTP 容器获取的参考配置旨在提供一组关键功能和节点调优设置，以确保集群可以支持字符串的性能和资源利用率限制，典型的 RAN 分布式单元(DU)应用程序。来自基准配置的更改或禁止可能会影响功能可用性、性能和资源利用率。使用 PolicyGenTemplate CR 作为参考来创建根据您的特定站点要求量身定制的配置文件的层次结构。

为 RAN DU 集群配置定义的基准 PolicyGenTemplate CR 可以从 GitOps ZTP ztp-site-generate 容器中提取。如需了解更多详细信息，请参阅"准备 GitOps ZTP 站点配置存储库"。

PolicyGenTemplate CR 可以在 ./out/argocd/example/policygentemplates 文件夹中找到。参考架构具有共同、组和特定站点的配置 CR。每个 PolicyGenTemplate CR 都引用可在 ./out/source-crs 文件夹中找到的其他 CR。

与 RAN 集群配置相关的 PolicyGenTemplate CR 如下所述。为组 PolicyGenTemplate CR 提供了变量，以考虑单节点、三节点紧凑和标准集群配置的不同。同样，为单节点集群和多节点（compact 或 standard）集群提供了特定于站点的配置变体。使用与部署相关的组和特定于站点的配置变体。

表 10.1. RAN 部署的 PolicyGenTemplate CR

PolicyGenTemplate CR	描述
example-multinode-site.yaml	包含一组应用于多节点集群的 CR。这些 CR 配置 SR-IOV 功能，用于 RAN 安装。
example-sno-site.yaml	包含一组应用于单节点 OpenShift 集群的 CR。这些 CR 配置 SR-IOV 功能，用于 RAN 安装。
common-mno-ranGen.yaml	包含一组应用于多节点集群的通用 RAN 策略配置。
common-ranGen.yaml	包含一组应用于所有集群的通用 RAN CR。这些 CR 订阅一组 operator，提供典型的 RAN 和基准集群调整功能。
group-du-3node-ranGen.yaml	仅包含三节点集群的 RAN 策略。
group-du-sno-ranGen.yaml	仅包含单节点集群的 RAN 策略。
group-du-standard-ranGen.yaml	包含标准三个 control-plane 集群的 RAN 策略。

PolicyGenTemplate CR	描述
group-du-3node-validator-ranGen.yaml	**PolicyGenTemplate** CR 用于生成三节点集群所需的各种策略。
group-du-standard-validator-ranGen.yaml	**PolicyGenTemplate** CR 用于生成标准集群所需的各种策略。
group-du-sno-validator-ranGen.yaml	**PolicyGenTemplate** CR 用于生成单节点 OpenShift 集群所需的各种策略。

其他资源

- 准备 GitOps ZTP 站点配置存储库

10.1.4. 使用 PolicyGenTemplate CR 自定义受管集群

使用以下步骤自定义应用于使用 GitOps Zero Touch Provisioning (ZTP) 管道置备的受管集群的策略。

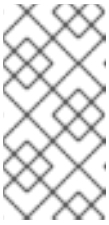
先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 配置了 hub 集群来生成所需的安装和策略 CR。
- 您创建了 Git 存储库，用于管理自定义站点配置数据。该存储库必须可从 hub 集群访问，并定义为 Argo CD 应用程序的源仓库。

流程

1. 为特定于站点的配置 CR 创建 PolicyGenTemplate CR。
   a. 从 out/argocd/example/policygentemplates 文件夹中选择适当的 CR 示例，例如 example-sno-site.yaml 或 example-multinode-site.yaml。

b. 更改示例文件中的 spec.bindingRules 字段，以匹配 SiteConfig CR 中包含的特定于站点的标签匹配。在示例 SiteConfig 文件中，特定于站点的标签是 sites: example-sno。



注意

确保 PolicyGenTemplate spec.bindingRules 字段中定义的标签与相关受管集群 SiteConfig CR 中定义的标签对应。

c. 更改示例文件中的内容，使其与所需配置匹配。

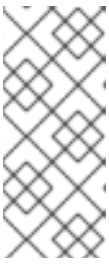
2. 可选：为应用到集群的任何通用配置 CR 创建一个 PolicyGenTemplate CR。

a. 从 out/argocd/example/policygentemplates 文件夹中选择适合您的 CR 示例，例如 common-ranGen.yaml。

b. 更改示例文件中的内容，使其与所需配置匹配。

3. 可选：为应用到团队中特定集群组的任何组配置 CR 创建一个 PolicyGenTemplate CR。

确保 overlaid spec 文件的内容与您的所需最终状态匹配。作为参考，out/source-crs 目录包含可用于包含和由 PolicyGenTemplate 模板提供的 source-crs 的完整列表。



注意

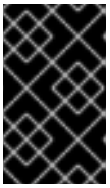
根据集群的特定要求，每个集群类型可能需要一个组策略，特别是考虑示例组策略各自有一个 PerformancePolicy.yaml 文件，如果这些集群是由相同的硬件配置，则只能在一组集群中共享。

a. 从 out/argocd/example/policygentemplates 文件夹中选择适当的 CR 示例，例如 group-du-sno-ranGen.yaml。

b. 更改示例文件中的内容，使其与所需配置匹配。

4. 可选。当 GitOps ZTP 安装和配置完成后，创建验证器通知策略 PolicyGenTemplate CR。如需更多信息，请参阅"创建验证器通知策略"。

5. 在 YAML 文件中定义所有策略命名空间，类似于示例 out/argocd/example/policygentemplates/ns.yaml 文件。



重要

不要在带有 PolicyGenTemplate CR 的同一文件中包括 Namespace CR。

6. 将 PolicyGenTemplate CR 和 Namespace CR 添加到 generators 部分中的 kustomization.yaml 文件中，类似于 out/argocd/example/policygentemplateskustomization.yaml 所示的示例。

7. 在 Git 存储库中提交 PolicyGenTemplate CR、Namespace CR 和关联的 kustomization.yaml 文件并推送更改。

ArgoCD 管道检测到更改并开始受管集群部署。您可以同时将更改推送到 SiteConfig CR 和 PolicyGenTemplate CR。

其他资源

- 使用验证器通知策略信号 GitOps ZTP 集群部署完成

10.1.5. 监控受管集群策略部署进度

ArgoCD 管道使用 Git 中的 PolicyGenTemplate CR 生成 RHACM 策略，然后将其同步到 hub 集群。您可以在辅助服务在受管集群中安装 OpenShift Container Platform 后监控受管集群策略同步的进度。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。

流程

1. Topology Aware Lifecycle Manager(TALM)应用绑定到集群的配置策略。

集群安装完成后，集群变为 Ready，ClusterGroupUpgrade CR 对应于此集群，且由 run.openshift.io/ztp-deploy-wave annotations 定义的已排序策略列表由 TALM 自动创建。集群的策略按 ClusterGroupUpgrade CR 中列出的顺序应用。

您可以使用以下命令监控配置策略协调的高级进度：

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[-1:]}' | jq
```

输出示例

```
{
 "lastTransitionTime": "2022-11-09T07:28:09Z",
 "message": "Remediating non-compliant policies",
 "reason": "InProgress",
 "status": "True",
 "type": "Progressing"
}
```

2. 您可以使用 RHACM 仪表板或命令行监控详细的集群策略合规状态。

   a. 要使用 oc 检查策略合规性，请运行以下命令：

```
$ oc get policies -n $CLUSTER
```

输出示例

```
NAME                                REMEDIATION ACTION   COMPLIANCE
STATE   AGE
ztp-common.common-config-policy                  inform            Compliant
```

```
3h42m
ztp-common.common-subscriptions-policy                inform
NonCompliant       3h42m
ztp-group.group-du-sno-config-policy                  inform            NonCompliant
3h42m
ztp-group.group-du-sno-validator-du-policy            inform            NonCompliant
3h42m
ztp-install.example1-common-config-policy-pjz9s       enforce           Compliant
167m
ztp-install.example1-common-subscriptions-policy-zzd9k   enforce
NonCompliant       164m
ztp-site.example1-config-policy                       inform            NonCompliant
3h42m
ztp-site.example1-perf-policy                         inform            NonCompliant
3h42m
```

b. 要从 RHACM Web 控制台检查策略状态，请执行以下操作：

i. 点 Governance → Find policies。

ii. 点集群策略检查其状态。

当所有集群策略都合规时，集群的 GitOps ZTP 安装和配置已完成。ztp-done 标签添加到集群中。

在引用配置中，合规的最终策略是 *-du-validator-policy 策略中定义的。此策略当在一个集群中合规时，确保所有集群配置、Operator 安装和 Operator 配置已完成。

10.1.6. 验证配置策略 CR 的生成

策略 自定义资源(CR)在与创建它们的 PolicyGenTemplate 相同的命名空间中生成。同样的故障排除流程适用于从 PolicyGenTemplate 生成的所有策略 CR，无论它们是 ztp-common、ztp-group，还是基于 ztp-site，请使用以下命令：

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

应该会显示预期的策略嵌套 CR 集合。

如果策略失败的同步，请使用以下故障排除步骤。

流程

1. 要显示策略的详细信息，请运行以下命令：

```
$ oc describe -n openshift-gitops application policies
```

2. 检查 Status: Conditions: 来显示错误日志。例如，将无效的 sourceFile 条目设置为 fileName: 生成以下错误：

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or
directory Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit
status 1: exit status 1
    Type:  ComparisonError
```

3. 检查 Status: Sync:。如果 Status: Conditions: 中存在日志错误，则 Status: Sync: 显示 Unknown 或 Error:

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  policies-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:           policies
        Repo URL:       https://git.com/ran-sites/policies/.git
        Target Revision:  master
    Status:             Error
```

4. 当 Red Hat Advanced Cluster Management(RHACM)识别策略应用到 ManagedCluster 对象时，策略 CR 对象应用到集群命名空间。检查策略是否已复制到集群命名空间中：

```
$ oc get policy -n $CLUSTER
```

输出示例：

```
NAME                                     REMEDIATION ACTION   COMPLIANCE STATE   AGE
ztp-common.common-config-policy          inform               Compliant          13d
ztp-common.common-subscriptions-policy   inform               Compliant          13d
ztp-group.group-du-sno-config-policy     inform               Compliant          13d
ztp-group.group-du-sno-validator-du-policy   inform           Compliant          13d
ztp-site.example-sno-config-policy       inform               Compliant          13d
```

RHACM 将所有适用的策略复制到集群命名空间中。复制的策略名称的格式有：<PolicyGenTemplate.Namespace>.<PolicyGenTemplate.Name>-<policyName>。

5. 检查放置规则中是否有没有复制到集群命名空间中的策略。这些策略的 PlacementRule 中的 matchSelector 应与 ManagedCluster 对象上的标签匹配：

```
$ oc get PlacementRule -n $NS
```

6. 使用以下命令，注意适合缺少策略、通用、组或站点的 PlacementRule 名称：

```
$ oc get PlacementRule -n $NS <placement_rule_name> -o yaml
```

- status-decisions 应该包括集群名称。
- spec 中 matchSelector 的键值对必须与受管集群上的标签匹配。

7. 使用以下命令检查 ManagedCluster 对象上的标签：

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. 使用以下命令查看合规哪些策略：

```
$ oc get policy -n $CLUSTER
```

如果 Namespace、OperatorGroup 和 Subscription 策略兼容，但 Operator 配置策略不兼容，则 Operator 可能不会在受管集群中安装。这会导致 Operator 配置策略无法应用，因为 CRD 还没有应用到 spoke。

10.1.7. 重启策略协调

当发生意外合规问题时，您可以重启策略协调，例如 ClusterGroupUpgrade 自定义资源 (CR) 超时时。

流程

1. 在受管集群变为 Ready 后，Topology Aware Lifecycle Manager 在命名空间 ztp-install 中生成 ClusterGroupUpgrade CR：

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. 如果出现意外问题，且策略无法在配置超时（默认为 4 小时）内变为合规，ClusterGroupUpgrade CR 的状态会显示 UpgradeTimedOut ：

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[?(@.type=="Ready")]}'
```

3. UpgradeTimedOut 状态的 ClusterGroupUpgrade CR 每小时自动重启其策略协调。如果更改了策略，可以通过删除现有 ClusterGroupUpgrade CR 来启动立即重试。这会触发自动创建新的 ClusterGroupUpgrade CR，以开始立即协调策略：

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

请注意，当 ClusterGroupUpgrade CR 完成，其状态为 UpgradeCompleted，并且受管集群应用了 ztp-done 标签，您可以使用 PolicyGenTemplate 创建额外的配置更改。删除现有的 ClusterGroupUpgrade CR 将无法生成新的 CR。

此时，GitOps ZTP 完成了与集群的交互，任何进一步的交互都应被视为更新，并为补救策略创建新的 ClusterGroupUpgrade CR。

其他资源

-

有关使用 Topology Aware Lifecycle Manager (TALM)来构建自己的 ClusterGroupUpgrade CR 的详情，请参考关于 ClusterGroupUpgrade CR。

10.1.8. 使用策略更改应用的受管集群 CR

您可以通过策略从受管集群中部署的自定义资源(CR)中删除内容。

默认情况下，从 PolicyGenTemplate CR 创建的所有 Policy CR 将 complianceType 字段设置为 musthave。没有删除内容的 musthave 策略仍然合规，因为受管集群上的 CR 具有所有指定的内容。使用这个配置，当从 CR 中删除内容时，TALM 从策略中删除内容，但不会从受管集群的 CR 中删除内容。

当 complianceType 字段为 mustonlyhave 时，策略可确保集群中的 CR 与策略中指定的内容完全匹配。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已从运行 RHACM 的 hub 集群部署了受管集群。
- 您已在 hub 集群中安装了 Topology Aware Lifecycle Manager。

流程

1. 从受影响的 CR 中删除您不再需要的内容。在本例中，disableDrain: false 行已从 SriovOperatorConfig CR 中删除。

   CR 示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
```

```
spec:
  configDaemonNodeSelector:
    "node-role.kubernetes.io/$mcp": ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
```

2. 在 group-du-sno-ranGen.yaml 文件中，将受影响的策略的 complianceType 更改为 mustonlyhave。

YAML 示例

```
- fileName: SriovOperatorConfig.yaml
  policyName: "config-policy"
  complianceType: mustonlyhave
```

3. 创建 ClusterGroupUpdates CR，并指定必须接收 CR 更改的集群：

ClusterGroupUpdates CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-remove
  namespace: default
spec:
  managedPolicies:
    - ztp-group.group-du-sno-config-policy
  enable: false
  clusters:
  - spoke1
  - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:
```

4. 运行以下命令来创建 ClusterGroupUpgrade CR：

```
$ oc create -f cgu-remove.yaml
```

5. 当您准备好应用更改时，例如在适当的维护窗口中，运行以下命令将 spec.enable 字段的值改为 true ：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

验证

1. 运行以下命令，检查策略的状态：

```
$ oc get <kind> <changed_cr_name>
```

输出示例

```
NAMESPACE   NAME                                        REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-ztp-group.group-du-sno-config-policy          enforce
17m
default     ztp-group.group-du-sno-config-policy              inform
NonCompliant       15h
```

当策略的 COMPLIANCE STATE 为 Compliant 时，这意味着已更新 CR，并删除不需要的内容。

2. 在受管集群中运行以下命令来检查策略是否已从目标集群中移除：

```
$ oc get <kind> <changed_cr_name>
```

如果没有结果，则会从受管集群中删除 CR。

10.1.9. 假定为 GitOps ZTP 安装

GitOps Zero Touch Provisioning (ZTP) 简化了检查集群的 GitOps ZTP 安装状态的过程。GitOps ZTP 状态分为三个阶段：集群安装、集群配置和 GitOps ZTP。

集群安装阶段

集群安装阶段由 ManagedCluster CR 中的 ManagedClusterJoined 和 ManagedClusterAvailable 条件显示。如果 ManagedCluster CR 没有这些条件，或者条件设置为 False，集群仍然处于安装阶段。有关安装的更多信息，请参阅 AgentClusterInstall 和 ClusterDeployment CR。如需更多信息，请参阅"Troubleshooting GitOps ZTP"。

集群配置阶段

集群配置阶段由 ztp-running 标签显示，在集群中应用 ManagedCluster CR。

完成 GitOps ZTP

集群安装和配置在 GitOps ZTP 完成。这可以通过删除 ztp-running 标签并在 ManagedCluster CR 中添加 ztp-done 标签来显示。ztp-done 标签显示应用了配置，基准 DU 配置已完成集群调整。

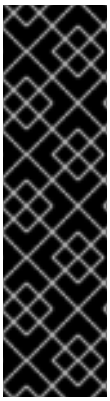
对 GitOps ZTP 完成的状态的更改是在 Red Hat Advanced Cluster Management (RHACM)验证器通知策略合规状态的条件。这个策略捕获了已完成的安装的现有条件，并确认只有在受管集群的 GitOps ZTP 置备完成后才会变为合规状态。

验证器通知策略可确保完全应用集群的配置，Operator 已完成初始化。策略验证以下内容：

- 目标 MachineConfigPool 包含预期的条目，并已完成更新。所有节点都可用，且没有降级。
- 至少有一个 SriovNetworkNodeState 带有 syncStatus: Succeeded 则代表 SR-IOV Operator 已完成初始化。
- PTP Operator 守护进程集已存在。

10.2. 使用 POLICYGENTEMPLATE 资源进行高级受管集群配置

您可以使用 PolicyGenTemplate CR 在受管集群中部署自定义功能。



重要

使用 PolicyGenTemplate CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 PolicyGenerator CR 提供了等效和改进的功能。

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

10.2.1. 为集群部署额外的更改

如果您需要在基本 GitOps Zero Touch Provisioning (ZTP) 管道配置之外更改集群配置，则有三个选项：

在 GitOps ZTP 管道完成后应用额外的配置

当 GitOps ZTP 管道部署完成后，部署的集群就可以用于应用程序工作负载。此时，您可以安装其他 Operator 并应用具体具体要求的配置。确保额外的配置不会影响平台或分配的 CPU 预算的性能。

在 GitOps ZTP 库中添加内容

使用 GitOps ZTP 管道部署的基本源自定义资源 (CR) 可以根据需要使用自定义内容增强。

为集群安装创建额外的清单

在安装过程中应用额外的清单，并使安装过程更高效。

重要

提供额外的源 CR 或修改现有源 CR 可能会影响 OpenShift Container Platform 的性能或 CPU 配置集。

10.2.2. 使用 PolicyGenTemplate CR 覆盖源 CR 内容

PolicyGenTemplate 自定义资源 (CR) 允许您覆盖与 ztp-site-generate 容器中提供的 GitOps 插件提供的基本源 CR 之上的额外配置详情。您可以将 PolicyGenTemplate CR 视为基础 CR 的逻辑合并或补丁。使用 PolicyGenTemplate CR 更新基本 CR 的单个字段，或覆盖基本 CR 的整个内容。您可以更新不在基本 CR 中的值和插入字段。

以下示例步骤描述了如何根据 group-du-sno-ranGen.yaml 文件中的 PolicyGenTemplate CR 为参考配置更新生成的 PerformanceProfile CR 中的字段。根据要求，使用流程修改 PolicyGenTemplate 的其他部分。

先决条件

- 创建一个 Git 存储库，在其中管理自定义站点配置数据。存储库必须可从 hub 集群访问，并定义为 Argo CD 的源存储库。

流程

1. 查看基准源 CR 以查找现有内容。您可以通过从 GitOps Zero Touch Provisioning (ZTP) 容器中提取引用 PolicyGenTemplate CR 中列出的源 CR。

   a. 创建 /out 文件夹：

   ```
   $ mkdir -p ./out
   ```

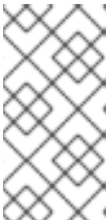
   b. 提取源 CR：

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16.1 extract /home/ztp --tar | tar x -C ./out
   ```

2. 查看 ./out/source-crs/PerformanceProfile.yaml 中的基线 PerformanceProfile CR：

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   ```

```
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```



注意

如果 PolicyGenTemplate CR 中未提供，则包含 $... 的任何字段都会从生成的 CR 中删除。

3. 在 group-du-sno-ranGen.yaml 参考文件中为 PerformanceProfile 更新 PolicyGenTemplate 条目。以下示例 PolicyGenTemplate CR 小节提供了适当的 CPU 规格，设置 hugepages 配置，并添加一个新的字段，将 globallyDisableIrqLoadBalancing 设置为 false。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    name: openshift-node-performance-profile
  spec:
    cpu:
      # These must be tailored for the specific hardware platform
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
```

```
      - size: 1G
        count: 10
    globallyDisableIrqLoadBalancing: false
```

4. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

输出示例

GitOps ZTP 应用程序生成包含生成的 PerformanceProfile CR 的 RHACM 策略。该 CR 的内容通过将 PolicyGenTemplate 中的 PerformanceProfile 条目的 metadata 和 spec 内容合并到源 CR 中。生成的 CR 包含以下内容：

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
  cpu:
    isolated: 2-19,22-39
    reserved: 0-1,20-21
  globallyDisableIrqLoadBalancing: false
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 10
        size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
```

注意

在从 ztp-site-generate 容器中提取的 /source-crs 文件夹中，$ 语法用于模板替换。相反，如果 policyGen 工具看到字符串的 $ 前缀，并且您不会在相关 PolicyGenTemplate CR 中为该字段指定值，则会完全从输出 CR 省略该字段。

一个例外是 /source-crs YAML 文件中的 $mcp 变量，该文件被替换为来自 PolicyGenTemplate CR 的 mcp 的指定的值。例如，在 example/policygentemplates/group-du-standard-ranGen.yaml 中，mcp 的值为 worker ：

```
spec:
  bindingRules:
    group-du-standard: ""
  mcp: "worker"
```

policyGen 工具将输出 CR 中的 $mcp 实例替换为 worker。

10.2.3. 在 GitOps ZTP 管道中添加自定义内容

执行以下步骤在 GitOps ZTP 管道中添加新内容。

流程

1. 在目录中创建一个名为 source-crs 的子目录，其中包含 PolicyGenTemplate 自定义资源 (CR)的 kustomization.yaml 文件。

2. 将用户提供的 CR 添加到 source-crs 子目录中，如下例所示：

```
example
└── policygentemplates
    ├── dev.yaml
    ├── kustomization.yaml
    ├── mec-edge-sno1.yaml
    ├── sno.yaml
    └── source-crs 1
        ├── PaoCatalogSource.yaml
        ├── PaoSubscription.yaml
        ├── custom-crs
        |   ├── apiserver-config.yaml
        |   └── disable-nic-lldp.yaml
```

```
    └── elasticsearch
        ├── ElasticsearchNS.yaml
        └── ElasticsearchOperatorGroup.yaml
```

1

source-crs 子目录必须与 kustomization.yaml 文件位于同一个目录中。

3. 更新所需的 PolicyGenTemplate CR，使其包含对 source-crs/custom-crs 和 source-crs/elasticsearch 目录中添加的内容的引用。例如：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-dev"
  namespace: "ztp-clusters"
spec:
  bindingRules:
    dev: "true"
  mcp: "master"
  sourceFiles:
    # These policies/CRs come from the internal container Image
    #Cluster Logging
    - fileName: ClusterLogNS.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-ns"
    - fileName: ClusterLogOperGroup.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-operator-group"
    - fileName: ClusterLogSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-sub"
    #Local Storage Operator
    - fileName: StorageNS.yaml
      remediationAction: inform
      policyName: "group-dev-lso-ns"
    - fileName: StorageOperGroup.yaml
      remediationAction: inform
      policyName: "group-dev-lso-operator-group"
    - fileName: StorageSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-lso-sub"
    #These are custom local polices that come from the source-crs directory in the git
repo
    # Performance Addon Operator
    - fileName: PaoSubscriptionNS.yaml
      remediationAction: inform
      policyName: "group-dev-pao-ns"
    - fileName: PaoSubscriptionCatalogSource.yaml
      remediationAction: inform
      policyName: "group-dev-pao-cat-source"
      spec:
```

```
      image: <container_image_url>
    - fileName: PaoSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-pao-sub"
    #Elasticsearch Operator
    - fileName: elasticsearch/ElasticsearchNS.yaml 1
      remediationAction: inform
      policyName: "group-dev-elasticsearch-ns"
    - fileName: elasticsearch/ElasticsearchOperatorGroup.yaml
      remediationAction: inform
      policyName: "group-dev-elasticsearch-operator-group"
    #Custom Resources
    - fileName: custom-crs/apiserver-config.yaml 2
      remediationAction: inform
      policyName: "group-dev-apiserver-config"
    - fileName: custom-crs/disable-nic-lldp.yaml
      remediationAction: inform
      policyName: "group-dev-disable-nic-lldp"
```

1 2

将 fileName 设置为包含 /source-crs 父目录中文件的相对路径。

4. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP Argo CD 策略应用程序监控的 Git 存储库。

5. 更新 ClusterGroupUpgrade CR，使其包含更改的 PolicyGenTemplate，并将它保存为 cgu-test.yaml。以下示例显示了生成的 cgu-test.yaml 文件。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: custom-source-cr
  namespace: ztp-clusters
spec:
  managedPolicies:
    - group-dev-config-policy
  enable: true
  clusters:
  - cluster1
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

6. 运行以下命令来应用更新的 ClusterGroupUpgrade CR：

```
$ oc apply -f cgu-test.yaml
```

验证

- 运行以下命令检查更新是否成功：

```
$ oc get cgu -A
```

输出示例

```
NAMESPACE     NAME             AGE   STATE       DETAILS
ztp-clusters  custom-source-cr   6s    InProgress   Remediating non-compliant policies
ztp-install   cluster1          19h   Completed    All clusters are compliant with all the
managed policies
```

10.2.4. 为 PolicyGenTemplate CR 配置策略合规性评估超时

使用在 hub 集群上安装的 Red Hat Advanced Cluster Management (RHACM) 来监控和报告您的受管集群是否合规。RHACM 使用策略模板来应用预定义的策略控制器和策略。策略控制器是 Kubernetes 自定义资源定义（CRD）实例。

您可以使用 PolicyGenTemplate 自定义资源 (CR) 覆盖默认策略评估间隔。您可以配置持续时间设置，以定义 ConfigurationPolicy CR 在 RHACM 重新评估集群策略前处于策略合规或不合规的时长。

GitOps Zero Touch Provisioning (ZTP) 策略生成器使用预定义的策略评估间隔生成 ConfigurationPolicy CR 策略。noncompliant 状态的默认值为 10 秒。compliant 状态的默认值为 10 分钟。要禁用评估间隔，将值设为 never。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

流程

1. 要为 PolicyGenTemplate CR 中的所有策略配置评估间隔，请为 evaluationInterval 字段设置适当的 合规 和 不合规的 值。例如：

```
spec:
  evaluationInterval:
    compliant: 30m
    noncompliant: 20s
```

注意

您还可以将 合规 和 不合规 字段设置为 永远不会 在达到特定合规状态后停止评估策略。

2. 要在 PolicyGenTemplate CR 中为单个策略对象配置评估间隔，请添加 evaluationInterval 字段并设置适当的值。例如：

```
spec:
  sourceFiles:
    - fileName: SriovSubscription.yaml
      policyName: "sriov-sub-policy"
      evaluationInterval:
        compliant: never
        noncompliant: 10s
```

3. 在 Git 存储库中提交 PolicyGenTemplate CR 文件并推送您的更改。

验证

检查管理的 spoke 集群策略是否以预期间隔监控。

1. 在受管集群中以具有 cluster-admin 权限的用户身份登录。

2. 获取在 open-cluster-management-agent-addon 命名空间中运行的 pod。运行以下命令：

```
$ oc get pods -n open-cluster-management-agent-addon
```

输出示例

```
NAME                                READY   STATUS    RESTARTS       AGE
config-policy-controller-858b894c68-v4xdb   1/1     Running   22 (5d8h ago)   10d
```

3. 检查应用的策略是以 config-policy-controller pod 的日志中预期间隔评估：

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-
858b894c68-v4xdb
```

输出示例

```
2022-05-10T15:10:25.280Z    info    configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation
due to the policy not reaching the evaluation interval  {"policy": "compute-1-config-
policy-config"}
2022-05-10T15:10:25.280Z    info    configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation
due to the policy not reaching the evaluation interval  {"policy": "compute-1-common-
compute-1-catalog-policy-config"}
```

10.2.5. 使用验证器通知策略信号 GitOps ZTP 集群部署完成

创建一个验证器通知策略，在 GitOps Zero Touch Provisioning (ZTP) 安装和配置完成部署集群时信号。此策略可用于部署单节点 OpenShift 集群、三节点集群和标准集群。

流程

1. 创建包含源文件 validatorCR/informDuValidator.yaml 的独立 PolicyGenTemplate 自定义资源 (CR)。每个集群类型只需要一个独立 PolicyGenTemplate CR。例如，此 CR 为单节点 OpenShift 集群应用验证器通知策略：

Example single-node cluster validator inform policy CR (group-du-sno-validator-ranGen.yaml)

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-sno: "" 3
  bindingExcludedRules:
    ztp-done: "" 4
  mcp: "master" 5
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform 6
      policyName: "du-policy" 7
```

1 {policy-gen-crs} 对象的名称。此名称也用作在请求的 namespace 中创建的 placementBinding、placementRule 和 policy 的一部分。

2 这个值应该与组 policy-gen-crs 中使用的 命名空间匹配。

3 bindingRules 中定义的 group-du-* 标签必须存在于 SiteConfig 文件中。

4 bindingExcludedRules 中定义的标签必须是'ztp-done:'。ztp-done 标签用于与 Topology Aware Lifecycle Manager 协调。

5 mcp 定义在源文件 validatorCR/informDuValidator.yaml 中使用的 MachineConfigPool 对象。它应该是单一节点的 master，以及用于标准集群部署的三节点集群部署和 worker。

6 可选。默认值是 inform。

7

2. 在 Git 存储库中提交 PolicyGenTemplate CR 文件并推送更改。

其他资源

- 升级 GitOps ZTP

10.2.6. 使用 PolicyGenTemplate CR 配置电源状态

对于低延迟和高性能部署，需要禁用或限制 C-states 和 P-states。使用这个配置，CPU 以恒定的频率运行，通常是最大 turbo 频率。这样可确保 CPU 始终以最大速度运行，这会导致高性能和低延迟。这会导致工作负载的最佳延迟。但是，这也会导致最高的功耗，这可能并不适用于所有工作负载。

工作负载可以归类为关键或非关键状态，需要为高性能和低延迟禁用 C-state 和 P-state 设置，而非关键工作负载在某些延迟和性能方面使用 C-state 和 P-state 设置。您可以使用 GitOps Zero Touch Provisioning (ZTP) 配置以下三个电源状态：

- 高性能模式以最高的功耗提供大量低延迟。
- 性能模式在相对高功耗时提供低延迟。
- 节能通过增加延迟来降低功耗。

默认配置用于低延迟性能模式。

PolicyGenTemplate 自定义资源 (CR) 允许您覆盖与 ztp-site-generate 容器中提供的 GitOps 插件提供的基本源 CR 之上的额外配置详情。

根据 group-du-sno-ranGen.yaml 中的 PolicyGenTemplate CR，通过更新生成的 PerformanceProfile CR 中的 workloadHints 字段来配置电源状态。

以下常见先决条件适用于配置所有三个电源状态。

先决条件

- 您已创建了管理自定义站点配置数据的 Git 存储库。存储库必须可从 hub 集群访问，并定义为 Argo CD 的源存储库。
- 您已遵循"准备 GitOps ZTP 站点配置存储库"中所述的步骤。

其他资源

- 使用工作负载提示配置节点功耗和实时处理

10.2.6.1. 使用 PolicyGenTemplate CR 配置性能模式

按照以下示例，根据 group-du-sno-ranGen.yaml 中的 PolicyGenTemplate CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置性能模式。

性能模式在相对高功耗时提供低延迟。

先决条件

- 您已按照"配置主机固件以实现低延迟和高性能"中的指导配置了与性能相关的 BIOS。

流程

1. 在 out/argocd/example/policygentemplates// 中更新 group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenTemplate 条目，以设置性能模式。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
    # ...
    workloadHints:
      realTime: true
      highPowerConsumption: false
      perPodPowerManagement: false
```

2. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

10.2.6.2. 使用 PolicyGenTemplate CR 配置高性能模式

按照以下示例，根据 group-du-sno-ranGen.yaml 中的 PolicyGenTemplate CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置高性能模式。

高性能模式以最高的功耗提供大量低延迟。

先决条件

- 您已按照"配置主机固件以实现低延迟和高性能"中的指导配置了与性能相关的 BIOS。

流程

1. 在 out/argocd/example/policygentemplates/ 中更新 group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenTemplate 条目，以设置高性能模式。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
  # ...
    workloadHints:
      realTime: true
      highPowerConsumption: true
      perPodPowerManagement: false
```

2. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

10.2.6.3. 使用 PolicyGenTemplate CR 配置节能模式

按照以下示例，根据 group-du-sno-ranGen.yaml 中的 PolicyGenTemplate CR 更新生成的 PerformanceProfile CR 中的 workloadHints 字段来设置节能模式。

节能模式会在增加延迟的情况下平衡功耗。

先决条件

- 您在 BIOS 中启用了 C-states 和 OS 控制的 P-states。

流程

1. 在 out/argocd/example/policygentemplates/ 中更新 group-du-sno-ranGen.yaml 参考文件中的 PerformanceProfile 的 PolicyGenTemplate 条目，以配置节能模式。建议您通过额外的内核参数对象为节能模式配置 CPU 调控器。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
    # ...
    workloadHints:
      realTime: true
      highPowerConsumption: false
      perPodPowerManagement: true
    # ...
    additionalKernelArgs:
      - # ...
      - "cpufreq.default_governor=schedutil" (1)
```

(1) 建议使用 schedutil governor，但可以使用的其他 governor 包括 ondemand 和 powersave。

2. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

验证

1. 使用以下命令，从标识的节点列表中选择部署的集群中的 worker 节点：

```
$ oc get nodes
```

2. 使用以下命令登录到节点：

```
$ oc debug node/<node-name>
```

将 <node-name> 替换为您要验证电源状态的节点的名称。

3. 将 /host 设置为 debug shell 中的根目录。debug pod 在 pod 中的 /host 中挂载主机的 root 文件系统。通过将根目录改为 /host，您可以运行主机可执行路径中包含的二进制文件，如下例所示：

```
# chroot /host
```

4. 运行以下命令验证应用的电源状态：

```
# cat /proc/cmdline
```

预期输出

- 对于节能模式，intel_pstate=passive。

其他资源

- 为运行 colocated 高和低优先级工作负载的节点配置节能
- 为低延迟和高性能配置主机固件
- 准备 GitOps ZTP 站点配置存储库

10.2.6.4. 最大化节能

建议限制最大 CPU 频率，以实现最大节能。在非关键工作负载 CPU 中启用 C-states，而不会限制最大 CPU 频率，从而提高了关键 CPU 的频率。

通过更新 sysfs 插件字段来最大化节能，为参考配置的 Tuned PerformancePatch CR 中的 max_perf_pct 设置适当的值。这个示例基于 group-du-sno-ranGen.yaml 描述了限制最大 CPU 频率的步骤。

先决条件

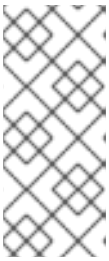
- 您已配置了节能模式，如"使用 PolicyGenTemplate CR 来配置节能模式"中所述。

流程

1. 在 out/argocd/example/policygentemplates/ 中，更新 group-du-sno-ranGen.yaml 参考文件中的 Tuned PerformancePatch 的 PolicyGenTemplate 条目。要最大化节能，请添加 max_perf_pct，如下例所示：

```
- fileName: TunedPerformancePatch.yaml
  policyName: "config-policy"
  spec:
    profile:
      - name: performance-patch
        data: |
          # ...
          [sysfs]
          /sys/devices/system/cpu/intel_pstate/max_perf_pct=<x> 1
```

1 max_perf_pct 控制 cpufreq 驱动程序的最大频率，以最大百分比的形式设置支持的 CPU 频率。这个值适用于所有 CPU。您可以检查 /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq 中的最大支持频率。作为起点，您可以使用以 All Cores Turbo 频率封装所有 CPU 的百分比。All Cores Turbo 频率是所有内核在运行的频率，当内核完全占用时。



注意

要最大化节能，请设置一个较低的值。为 max_perf_pct 设置较低值会限制最大 CPU 频率，从而减少功耗，但可能会影响性能。试验不同的值并监控系统性能和功耗，以查找您的用例的最佳设置。

2. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP argo CD 应用程序监控的 Git 存储库。

10.2.7. 使用 PolicyGenTemplate CR 配置 LVM 存储

您可以使用 GitOps Zero Touch Provisioning (ZTP)为部署的受管集群配置逻辑卷管理器(LVM)存储。

注意

当使用 PTP 事件或带有 HTTP 传输的裸机硬件事件时，您可以使用 LVM Storage 来保留事件订阅。

将 Local Storage Operator 用于在分布式单元中使用本地卷的持久性存储。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 创建一个 Git 存储库，在其中管理自定义站点配置数据。

流程

1. 要为新受管集群配置 LVM Storage，请在 common-ranGen.yaml 文件中的 spec.sourceFiles 中添加以下 YAML：

```
- fileName: StorageLVMOSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscription.yaml
  spec:
    name: lvms-operator
    channel: stable-4.16
  policyName: subscription-policies
```

注意

Storage LVMO 订阅已弃用。在以后的 OpenShift Container Platform 版本中，存储 LVMO 订阅将不可用。反之，您必须使用 Storage LVMS 订阅。

在 OpenShift Container Platform 4.16 中，您可以使用 Storage LVMS 订阅而不是 LVMO 订阅。LVMS 订阅不需要在 common-ranGen.yaml 文件中手动覆盖。将以下 YAML 添加到 common-ranGen.yaml 文件中的 spec.sourceFiles 中，以使用 Storage LVMS 订阅：

```
- fileName: StorageLVMSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscription.yaml
  policyName: subscription-policies
```

2. **将 LVMCluster CR 添加到特定组或单个站点配置文件中的 spec.sourceFiles 中。例如，在 group-du-sno-ranGen.yaml 文件中添加以下内容：**

```
- fileName: StorageLVMCluster.yaml
  policyName: "lvms-config"
  spec:
    storage:
      deviceClasses:
      - name: vg1
        thinPoolConfig:
          name: thin-pool-1
          sizePercent: 90
          overprovisionRatio: 10
```

这个示例配置创建一个带有所有可用设备的卷组 (vg1)，但安装了 OpenShift Container Platform 的磁盘除外。也创建了一个精简池逻辑卷。

3. **将任何其他必要的更改和文件与自定义站点存储库合并。**

4. **提交 Git 中的 PolicyGenTemplate 更改，然后将更改推送到站点配置存储库，以使用 GitOps ZTP 将 LVM 存储部署到新站点。**

10.2.8. 使用 PolicyGenTemplate CR 配置 PTP 事件

您可以使用 GitOps ZTP 管道来配置使用 HTTP 或 AMQP 传输的 PTP 事件。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

10.2.8.1. 配置使用 HTTP 传输的 PTP 事件

您可以配置使用 GitOps Zero Touch Provisioning (ZTP)管道部署的受管集群中使用 HTTP 传输的 PTP 事件。

先决条件

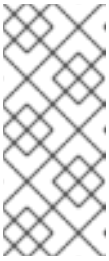
- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 根据您的具体要求，将以下 PolicyGenTemplate 应用到 group-du-3node-ranGen.yaml、group-du-sno-ranGen.yaml 或 group-du-standard-ranGen.yaml 文件：

   a. 在 spec.sourceFiles 中，添加 PtpOperatorConfig CR 文件来配置传输主机：

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043
```



注意

在 OpenShift Container Platform 4.13 或更高版本中，在使用带有 PTP 事件的 HTTP 传输时，您不需要在 PtpOperatorConfig 资源中设置 transportHost 字段。

b. 为 PTP 时钟类型和接口配置 linuxptp 和 phc2sys。例如，将以下 YAML 添加到 spec.sourceFiles 中：

```
- fileName: PtpConfigSlave.yaml (1)
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
    - name: "slave"
      interface: "ens5f1" (2)
      ptp4lOpts: "-2 -s --summary_interval -4" (3)
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" (4)
    ptpClockThreshold: (5)
      holdOverTimeout: 30 # seconds
      maxOffsetThreshold: 100  # nano seconds
      minOffsetThreshold: -100
```

(1) 可以是 PtpConfigMaster.yaml、PtpConfigSlave.yaml 或 PtpConfigSlaveCvl.yaml 之一，具体取决于您的要求。PtpConfigSlaveCvl.yaml 为 Intel E810 Columbiaville NIC 配置 linuxptp 服务。对于基于 group-du-sno-ranGen.yaml 或 group-du-3node-ranGen.yaml 的配置，请使用 PtpConfigSlave.yaml。

(2) 特定于设备的接口名称。

(3) 您必须将 --summary_interval -4 值附加到 .spec.sourceFiles.spec.profile 中的 ptp4lOpts 中，以启用 PTP fast 事件。

(4) 所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

(5)

2. 将任何其他必要的更改和文件与自定义站点存储库合并。

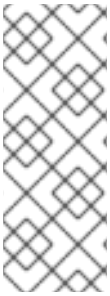
3. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将 PTP 快速事件部署到新站点。

其他资源

- 使用 PolicyGenTemplate CR 覆盖源 CR 内容

10.2.8.2. 配置使用 AMQP 传输的 PTP 事件

您可以在使用 GitOps Zero Touch Provisioning (ZTP) 管道部署的受管集群中配置使用 AMQP 传输的 PTP 事件。



注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 将以下 YAML 添加到 common-ranGen.yaml 文件中的 spec.sourceFiles 中，以配置 AMQP Operator：

```
#AMQ interconnect operator for fast events
```

```
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
```

2. 根据您的具体要求，将以下 PolicyGenTemplate 应用到 group-du-3node-ranGen.yaml、group-du-sno-ranGen.yaml 或 group-du-standard-ranGen.yaml 文件：

   a. 在 spec.sourceFiles 中，添加 PtpOperatorConfig CR 文件，该文件将 AMQ 传输主机配置为 config-policy ：

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: "amqp://amq-router.amq-router.svc.cluster.local"
```

   b. 为 PTP 时钟类型和接口配置 linuxptp 和 phc2sys。例如，将以下 YAML 添加到 spec.sourceFiles 中：

```
- fileName: PtpConfigSlave.yaml (1)
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
    - name: "slave"
      interface: "ens5f1" (2)
      ptp4lOpts: "-2 -s --summary_interval -4" (3)
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" (4)
    ptpClockThreshold: (5)
      holdOverTimeout: 30 # seconds
      maxOffsetThreshold: 100  # nano seconds
      minOffsetThreshold: -100
```

(1) 可以是 PtpConfigMaster.yaml、PtpConfigSlave.yaml 或 PtpConfigSlaveCvl.yaml 之一，具体取决于您的要求。PtpConfigSlaveCvl.yaml 为 Intel E810 Columbiaville NIC 配置 linuxptp 服务。对于基于 group-du-sno-ranGen.yaml 或 group-du-3node-ranGen.yaml 的配置，请使用 PtpConfigSlave.yaml。

(2)

3 您必须将 --summary_interval -4 值附加到 .spec.sourceFiles.spec.profile 中的 ptp4lOpts 中，以启用 PTP fast 事件。

4 所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

5 可选。如果 ptpClockThreshold 小节不存在，则默认值用于 ptpClockThreshold 字段。小节显示默认的 ptpClockThreshold 值。ptpClockThreshold 值配置 PTP master 时钟在触发 PTP 事件前的时长。holdOverTimeout 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 FREERUN 前的时间值（以秒为单位）。maxOffsetThreshold 和 minOffsetThreshold 设置以纳秒为单位，它们与 CLOCK_REALTIME (phc2sys) 或 master 偏移 (ptp4l) 的值进行比较。当 ptp4l 或 phc2sys 偏移值超出这个范围时，PTP 时钟状态被设置为 FREERUN。当偏移值在这个范围内时，PTP 时钟状态被设置为 LOCKED。

3. 将以下 PolicyGenTemplate 更改应用到您的特定站点 YAML 文件，如 example-sno-site.yaml ：

   a. 在 spec.sourceFiles 中，添加 Interconnect CR 文件，该文件将 AMQ 路由器配置为 config-policy ：

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

4. 将任何其他必要的更改和文件与自定义站点存储库合并。

5. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将 PTP 快速事件部署到新站点。

其他资源

- 安装 AMQ 消息传递总线

- OpenShift 镜像 registry 概述

10.2.9. 使用 PolicyGenTemplate CR 配置裸机事件

您可以使用 GitOps ZTP 管道来配置使用 HTTP 或 AMQP 传输的裸机事件。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

10.2.9.1. 配置使用 HTTP 传输的裸机事件

您可以配置使用 GitOps Zero Touch Provisioning (ZTP)管道部署的受管集群中使用 HTTP 传输的裸机事件。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 通过在 common-ranGen.yaml 文件中的 spec.sourceFiles 中添加以下 YAML 来配置 Bare Metal Event Relay Operator：

```
# Bare Metal Event Relay Operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. 将 HardwareEvent CR 添加到特定组配置文件中的 spec.sourceFiles，例如在 group-du-sno-ranGen.yaml 文件中：

```
- fileName: HardwareEvent.yaml (1)
  policyName: "config-policy"
  spec:
    nodeSelector: {}
    transportHost: "http://hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043"
    logLevel: "info"
```

(1) 每个基板管理控制器 (BMC) 只需要一个 HardwareEvent CR。

注意

在 OpenShift Container Platform 4.13 或更高版本中，当将 HTTP 传输用于裸机事件时，您不需要在 HardwareEvent 自定义资源 (CR) 中设置 transportHost 字段。

3. 将任何其他必要的更改和文件与自定义站点存储库合并。

4. 将更改推送到站点配置存储库，以使用 GitOps ZTP 将裸机事件部署到新站点。

5. 运行以下命令来创建 Redfish Secret：

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password> \
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

其他资源

- 使用 CLI 安装裸机事件中继
- 创建裸机事件和 Secret CR

10.2.9.2. 配置使用 AMQP 传输的裸机事件

您可以在使用 GitOps Zero Touch Provisioning (ZTP) 管道部署的受管集群中配置使用 AMQP 传输的裸机事件。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已创建了管理自定义站点配置数据的 Git 存储库。

流程

1. 要配置 AMQ Interconnect Operator 和 Bare Metal Event Relay Operator，请将以下 YAML 添加到 common-ranGen.yaml 文件中的 spec.sourceFiles 中：

```
# AMQ Interconnect Operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
# Bare Metal Event Relay Operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. 将 Interconnect CR 添加到站点配置文件中的 spec.sourceFiles 中，例如 example-sno-site.yaml 文件：

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

3. 将 HardwareEvent CR 添加到特定组配置文件中的 spec.sourceFiles，例如在 group-du-sno-ranGen.yaml 文件中：

```
- path: HardwareEvent.yaml
  patches:
    nodeSelector: {}
    transportHost: "amqp://<amq_interconnect_name>.
<amq_interconnect_namespace>.svc.cluster.local" 1
    logLevel: "info"
```

1 transportHost URL 由现有的 AMQ Interconnect CR 名称和命名空间组成。例如，在 transportHost: "amq-router.amq-router.svc.cluster.local" 中，AMQ Interconnect name 和 namespace 都被设置为 amq-router。

注意

每个基板管理控制器 (BMC) 仅需要一个 HardwareEvent 资源。

4. 在 Git 中提交 PolicyGenTemplate 更改，然后将更改推送到您的站点配置存储库，以使用 GitOps ZTP 将裸机事件监控部署到新站点。

5. 运行以下命令来创建 Redfish Secret：

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

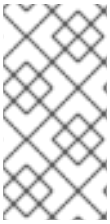
10.2.10. 配置 Image Registry Operator 以进行镜像的本地缓存

OpenShift Container Platform 使用本地 registry 管理镜像缓存。在边缘计算用例中，集群通常会受到带宽限制，与集中式镜像 registry 通信时，这可能会导致长时间镜像下载时间。

在初始部署期间，长时间下载时间不可避免。随着时间的推移，CRI-O 会在出现意外关闭时擦除

/var/lib/containers/storage 目录的风险。要解决镜像下载时间长的问题，您可以使用 GitOps Zero Touch Provisioning (ZTP) 在远程受管集群上创建本地镜像 registry。当集群部署在网络边缘时，这非常有用。

在使用 GitOps ZTP 设置本地镜像 registry 前，您需要在用于安装远程受管集群的 SiteConfig CR 中配置磁盘分区。安装后，您可以使用 PolicyGenTemplate CR 配置本地镜像 registry。然后，GitOps ZTP 管道创建持久性卷 (PV) 和持久性卷声明(PVC) CR，并修补 imageregistry 配置。



注意

本地镜像 registry 只能用于用户应用程序镜像，不能用于 OpenShift Container Platform 或 Operator Lifecycle Manager operator 镜像。

其他资源

- OpenShift Container Platform registry 概述

10.2.10.1. 使用 SiteConfig 配置磁盘分区

使用 SiteConfig CR 和 GitOps Zero Touch Provisioning (ZTP) 为受管集群配置磁盘分区。SiteConfig CR 中的磁盘分区详情必须与底层磁盘匹配。

重要

您必须在安装时完成这个步骤。

先决条件

- 安装 Butane。

流程

1. 创建 storage.bu 文件。

```
variant: fcos
version: 1.3.0
storage:
  disks:
  - device: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0 (1)
```

```
    wipe_table: false
    partitions:
    - label: var-lib-containers
      start_mib: <start_of_partition> 2
      size_mib: <partition_size> 3
  filesystems:
    - path: /var/lib/containers
      device: /dev/disk/by-partlabel/var-lib-containers
      format: xfs
      wipe_filesystem: true
      with_mount_unit: true
      mount_options:
        - defaults
        - prjquota
```

1 指定根磁盘。

2 以 MiB 为单位指定分区的起始位置。如果值太小，安装会失败。

3 指定分区的大小。如果值太小，部署会失败。

2. 运行以下命令，将 storage.bu 转换为 Ignition 文件：

```
$ butane storage.bu
```

输出示例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# # Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-partlabel/var-lib-containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. 使用 JSON Pretty Print 等工具将输出转换为 JSON 格式。

4. 将输出复制到 SiteConfig CR 中的 .spec.clusters.nodes.ignitionConfigOverride 字段中。

Example

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: |
          {
            "ignition": {
              "version": "3.2.0"
            },
            "storage": {
              "disks": [
                {
                  "device": "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0",
                  "partitions": [
                    {
                      "label": "var-lib-containers",
                      "sizeMiB": 0,
                      "startMiB": 250000
                    }
                  ],
                  "wipeTable": false
                }
              ],
              "filesystems": [
                {
                  "device": "/dev/disk/by-partlabel/var-lib-containers",
                  "format": "xfs",
                  "mountOptions": [
                    "defaults",
                    "prjquota"
                  ],
                  "path": "/var/lib/containers",
                  "wipeFilesystem": true
                }
              ]
            },
            "systemd": {
              "units": [
```

```
            {
              "contents": "# # Generated by Butane\n[Unit]\nRequires=systemd-
fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-
fsck@dev-disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
              "enabled": true,
              "name": "var-lib-containers.mount"
            }
          ]
        }
      }
[...]
```

注意

如果 .spec.clusters.nodes.ignitionConfigOverride 字段不存在，请创建它。

验证

1. 在安装过程中，运行以下命令来在 hub 集群上验证 BareMetalHost 对象显示注解：

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-
install.openshift.io/ignition-config-overrides"]
```

输出示例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-
id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-
containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":
[{\"device\":\"/dev/disk/by-partlabel/var-lib-
containers\",\"format\":\"xfs\",\"mountOptions\":
[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"sys
temd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-
fsck@dev-disk-by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-
by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/
dev/disk/by-partlabel/var-lib-
containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-
fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. **安装后，检查单节点 OpenShift 磁盘状态。**

a. **运行以下命令，在单节点 OpenShift 节点上进入 debug 会话。此步骤被实例化为一个名为 <node_name>-debug 的 debug pod:**

```
$ oc debug node/my-sno-node
```

b. **运行以下命令，将 /host 设置为 debug shell 中的根目录。debug pod 在 pod 中的 /host 中挂载主机的 root 文件系统。将根目录改为 /host，您可以运行主机可执行路径中包含的二进制文件：**

```
# chroot /host
```

c. **运行以下命令，列出所有可用块设备的信息：**

```
# lsblk
```

输出示例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                      /sysroot/ostree/deploy/rhcos/var
│                      /usr
│                      /etc
│                      /
│                      /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

d. **运行以下命令，显示文件系统磁盘空间使用情况的信息：**

```
# df -h
```

输出示例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

10.2.10.2. 使用 PolicyGenTemplate CR 配置镜像 registry

使用 PolicyGenTemplate (PGT) CR 应用配置镜像 registry 所需的 CR 并对 imageregistry 配置进行补丁。

先决条件

- 您已在受管集群中配置了磁盘分区。
- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了 Git 存储库，在其中管理自定义站点配置数据以用于 GitOps Zero Touch Provisioning (ZTP)。

流程

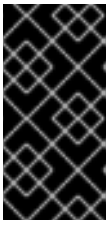
1. 在适当的 PolicyGenTemplate CR 中配置存储类、持久性卷声明、持久性卷和镜像 registry 配置。例如，要配置单个站点，请将以下 YAML 添加到文件 example-sno-site.yaml 中：

```
sourceFiles:
```

```
# storage class
- fileName: StorageClass.yaml
  policyName: "sc-for-image-registry"
  metadata:
    name: image-registry-sc
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100" (1)
# persistent volume claim
- fileName: StoragePVC.yaml
  policyName: "pvc-for-image-registry"
  metadata:
    name: image-registry-pvc
    namespace: openshift-image-registry
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
  spec:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 100Gi
    storageClassName: image-registry-sc
    volumeMode: Filesystem
# persistent volume
- fileName: ImageRegistryPV.yaml (2)
  policyName: "pv-for-image-registry"
  metadata:
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
- fileName: ImageRegistryConfig.yaml
  policyName: "config-for-image-registry"
  complianceType: musthave
  metadata:
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
  spec:
    storage:
      pvc:
        claim: "image-registry-pvc"
```

(1) 根据您要在站点、通用或组级别配置镜像 registry，为 ztp-deploy-wave 设置适当的值。ZTP-deploy-wave: "100" 适用于开发或测试，因为它允许您将引用的源文件分组到一起。

(2) 在 ImageRegistryPV.yaml 中，确保将 spec.local.path 字段设置为 /var/imageregistry，以匹配 SiteConfig CR 中为 mount_point 字段设置的值。



重要

不要为 - fileName: ImageRegistryConfig.yaml 配置设置 complianceType: mustonlyhave。这可能导致 registry pod 部署失败。

2. 提交 Git 中的 PolicyGenTemplate 更改，然后推送到由 GitOps ZTP Argo CD 应用程序监控的 Git 存储库。

验证

使用以下步骤排除受管集群中本地镜像 registry 的错误：

- 在登录到受管集群时，验证是否成功登录到 registry。运行以下命令：

  a. 导出受管集群名称：

  ```
  $ cluster=<managed_cluster_name>
  ```

  b. 获取受管集群 kubeconfig 详情：

  ```
  $ oc get secret -n $cluster $cluster-admin-password -o
  jsonpath='{.data.password}' | base64 -d > kubeadmin-password-$cluster
  ```

  c. 下载并导出集群 kubeconfig ：

  ```
  $ oc get secret -n $cluster $cluster-admin-kubeconfig -o
  jsonpath='{.data.kubeconfig}' | base64 -d > kubeconfig-$cluster && export
  KUBECONFIG=./kubeconfig-$cluster
  ```

  d. 验证从受管集群访问镜像 registry。请参阅"访问 registry"。

- 检查 imageregistry.operator.openshift.io 组实例的 Config CRD 是否没有报告错误。登录到受管集群时运行以下命令：

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

  输出示例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2021-10-08T19:02:39Z"
  generation: 5
  name: cluster
  resourceVersion: "688678648"
  uid: 0406521b-39c0-4cda-ba75-873697da75a4
spec:
  additionalTrustedCA:
    name: acm-ice
```

- 检查受管集群上的 PersistentVolumeClaim 是否填充了数据。登录到受管集群时运行以下命令：

```
$ oc get pv image-registry-sc
```

- 检查 registry* pod 是否正在运行，并位于 openshift-image-registry 命名空间下。

```
$ oc get pods -n openshift-image-registry | grep registry*
```

输出示例

```
cluster-image-registry-operator-68f5c9c589-42cfg   1/1     Running     0          8d
image-registry-5f8987879-6nx6h                     1/1     Running     0          8d
```

- 检查受管集群中的磁盘分区是否正确：

  a. 为受管集群打开默认 shell：

```
$ oc debug node/sno-1.example.com
```

b. 运行 lsblk 以检查主机磁盘分区：

```
sh-4.4# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
  |-sda1   8:1    0     1M  0 part
  |-sda2   8:2    0   127M  0 part
  |-sda3   8:3    0   384M  0 part /boot
  |-sda4   8:4    0 336.3G  0 part /sysroot
  `-sda5   8:5    0 100.1G  0 part /var/imageregistry 1
sdb      8:16   0 446.6G  0 disk
sr0     11:0    1   104M  0 rom
```

1 /var/imageregistry 表示磁盘已被正确分区。

其他资源

- 访问registry

10.3. 使用 POLICYGENTEMPLATE 资源和 TALM 在断开连接的环境中更新受管集群

您可以使用 Topology Aware Lifecycle Manager (TALM)来管理您使用 GitOps Zero Touch Provisioning (ZTP)和 Topology Aware Lifecycle Manager (TALM)部署的受管集群的软件生命周期。TALM 使用 Red Hat Advanced Cluster Management (RHACM) PolicyGenTemplate 策略来管理和控制应用到目标集群的更改。

重要

使用 PolicyGenTemplate CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 PolicyGenerator CR 提供了等效和改进的功能。

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁
- 关于 Topology Aware Lifecycle Manager

10.3.1. 设置断开连接的环境

TALM 可以同时执行平台和 Operator 更新。

您必须在镜像 registry 中镜像您要升级到的平台镜像和 Operator 镜像，然后才能使用 TALM 更新断开连接的集群。完成以下步骤以镜像镜像：

- 对于平台更新，您必须执行以下步骤：

1. 镜像所需的 OpenShift Container Platform 镜像存储库。根据"镜像 OpenShift Container Platform 镜像存储库"流程在附加资源中链接，确保所需的平台镜像已被镜像。在 imageContentSources.yaml 文件中保存 imageContentSources 部分的内容：

输出示例

```
imageContentSources:
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-release
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. 保存已镜像的所需平台镜像的镜像签名。对于平台更新，您必须将镜像签名添加到 PolicyGenTemplate CR 中。要获取镜像签名，请执行以下步骤：

a. 运行以下命令指定所需的 OpenShift Container Platform 标签：

```
$ OCP_RELEASE_NUMBER=<release_version>
```

b. 运行以下命令指定集群的构架：

```
$ ARCHITECTURE=<cluster_architecture> 1
```

1 指定集群的构架，如 x86_64, aarch64, s390x, 获 ppc64le。

c. 运行以下命令，从 Quay 获取发行版本镜像摘要

```
$ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From: .*@//p')"
```

d. 运行以下命令来设置摘要算法：

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

e. 运行以下命令来设置摘要签名：

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. 运行以下命令，从 mirror.openshift.com 网站获取镜像签名：

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

g. 运行以下命令，将镜像签名保存到 checksum-<OCP_RELEASE_NUMBER>.yaml 文件中：

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 准备更新图表。您可以通过两个选项来准备更新图形：

   a. 使用 OpenShift Update Service。

      有关如何在 hub 集群上设置图形的更多信息，请参阅为 OpenShift Update Service 部署 Operator 并构建图形数据 init 容器。

   b. 生成上游图形的本地副本。在可访问受管集群的断开连接的环境中的 http 或 https 服务器上托管更新图表。要下载更新图表，请使用以下命令：

      ```
      $ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.16 -o ~/upgrade-graph_stable-4.16
      ```

- 对于 Operator 更新，您必须执行以下任务：

  - 镜像 Operator 目录。确保所需的 Operator 镜像按照"Mirroring Operator 目录以用于断开连接的集群"部分中的步骤进行镜像。

其他资源

- 升级 GitOps ZTP
- 镜像 OpenShift Container Platform 镜像存储库
- 镜像用于断开连接的集群的 Operator 目录
- 准备断开连接的环境
- 了解更新频道和发行版本

10.3.2. 使用 PolicyGenTemplate CR 执行平台更新

您可以使用 TALM 执行平台更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 镜像所需的镜像存储库。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 为平台更新创建 PolicyGenTemplate CR：
   a. 将以下 PolicyGenTemplate CR 保存到 du-upgrade.yaml 文件中：

平台更新的 PolicyGenTemplate 示例

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: ImageSignature.yaml 1
```

```
  policyName: "platform-upgrade-prep"
  binaryData:
    ${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64} 2
- fileName: DisconnectedICSP.yaml
  policyName: "platform-upgrade-prep"
  metadata:
    name: disconnected-internal-icsp-for-ocp
  spec:
    repositoryDigestMirrors: 3
      - mirrors:
        - quay-intern.example.com/ocp4/openshift-release-dev
        source: quay.io/openshift-release-dev/ocp-release
      - mirrors:
        - quay-intern.example.com/ocp4/openshift-release-dev
        source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
- fileName: ClusterVersion.yaml 4
  policyName: "platform-upgrade"
  metadata:
    name: version
  spec:
    channel: "stable-4.16"
    upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.16
    desiredUpdate:
      version: 4.16.4
  status:
    history:
      - version: 4.16.4
        state: "Completed"
```

1

ConfigMap CR 包含要更新到的所需发行镜像的签名。

2

显示所需 OpenShift Container Platform 发行版本的镜像签名。按照"设置环境"部分中的步骤，从您保存的 checksum-${OCP_RELEASE_NUMBER}.yaml 文件中获取签名。

3

显示包含所需 OpenShift Container Platform 镜像的镜像存储库。获取在"设置 environment"部分中的步骤时所保存的 imageContentSources.yaml 文件中的镜像。

4

显示触发更新的 ClusterVersion CR。对于预缓存， channel, upstream, 和 desiredVersion 项都是必需的。

PolicyGenTemplate CR 会生成两个策略：

- du-upgrade-platform-upgrade-prep 策略为平台更新做准备。它为所需的发行版本镜像签名创建 ConfigMap CR，创建镜像的发行镜像存储库的镜像内容源，并使用所需的更新频道更新集群版本，以及在断开连接的环境中由 spoke 集群访问的更新图。
- du-upgrade-platform-upgrade 策略用于执行平台升级。

b. 将 du-upgrade.yaml 文件内容添加到 PolicyGenTemplate CR 的 GitOps ZTP Git 存储库中的 kustomization.yaml 文件中，并将更改推送到 Git 存储库。

ArgoCD 从 Git 存储库拉取更改并在 hub 集群上生成策略。

c. 运行以下命令检查创建的策略：

```
$ oc get policies -A | grep platform-upgrade
```

2. 为平台更新创建 ClusterGroupUpdate CR，将 spec.enable 项设置为 false。

a. 将平台更新 ClusterGroupUpdate CR 的内容，带有 du-upgrade-platform-upgrade-prep 和 du-upgrade-platform-upgrade 策略，以及目标集群保存到 cgu-platform-upgrade.yml 文件，如以下示例所述：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

b. 运行以下命令，将 ClusterGroupUpdate CR 应用到 hub 集群：

```
$ oc apply -f cgu-platform-upgrade.yml
```

3. 可选：缓存平台更新的镜像。

a. 运行以下命令，在 ClusterGroupUpdate CR 中启用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 监控更新过程，并等待预缓存完成。在 hub 集群中运行以下命令来检查预缓存的状态：

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

4. 启动平台更新：

a. 运行以下命令启用 cgu-platform-upgrade 策略并禁用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

其他资源

- 准备断开连接的环境

10.3.3. 使用 PolicyGenTemplate CR 执行 Operator 更新

您可以使用 TALM 执行 Operator 更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 镜像捆绑包镜像、捆绑包镜像以及捆绑包镜像中引用的所有 Operator 镜像。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 为 Operator 更新更新 PolicyGenTemplate CR。

   a. 使用 du-upgrade.yaml 文件中的以下额外内容更新 du-upgrade PolicyGenTemplate CR：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v4.16 (1)
```

```
  updateStrategy: 2
    registryPoll:
      interval: 1h
status:
  connectionState:
    lastObservedState: READY 3
```

1

索引镜像 URL 包含所需的 Operator 镜像。如果索引镜像始终推送到相同的镜像名称和标签，则不需要此更改。

2

使用 registryPoll.interval 字段设置 Operator Lifecycle Manager(OLM)轮询新 Operator 版本的索引镜像。如果为 y-stream 和 z-stream Operator 更新而总是推送新的索引镜像标签，则不需要此更改。registryPoll.interval 字段可以设置为较短的间隔，以加快更新，但较短的间隔会增大计算负载。要影响此行为，您可以在更新完成后将 registryPoll.interval 恢复到默认值。

3

目录连接的最后观察到状态。READY 值确保 CatalogSource 策略已就绪，表示索引 Pod 已拉取并在运行。这样，TALM 根据最新的策略合规性状态升级 Operator。

b.

在这个版本中，生成一个策略 du-upgrade-operator-catsrc-policy，以使用包含所需 Operator 镜像的新索引镜像更新 redhat-operators 目录源。

注意

如果要使用 Operator 预缓存，并且有来自 redhat-operators 以外的其他目录源的 Operator，您必须执行以下任务：

- 使用新的索引镜像或 registry 轮询间隔更新准备单独的目录源策略。
- 为来自不同目录源的所需 Operator 准备单独的订阅策略。

例如，所需的 SRIOV-FEC Operator 在 certified-operators 目录源中提供。要更新目录源和 Operator 订阅，请添加以下内容来生成两个策略： du-upgrade-fec-catsrc-policy 和 du-upgrade-subscriptions-fec-policy ：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    # ...
  - fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "fec-catsrc-policy"
    metadata:
      name: certified-operators
    spec:
      displayName: Intel SRIOV-FEC Operator
      image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
      updateStrategy:
        registryPoll:
          interval: 10m
  - fileName: AcceleratorsSubscription.yaml
    policyName: "subscriptions-fec-policy"
    spec:
      channel: "stable"
      source: certified-operators
```

c.

如果存在，在常规 PolicyGenTemplate CR 中删除指定的订阅频道。GitOps ZTP 镜像的默认订阅频道用于更新。

注意

通过 GitOps ZTP 4.16 应用的 Operator 的默认频道是 stable，但 performance-addon-operator 除外。从 OpenShift Container Platform 4.11 开始，performance-addon-operator 功能被移到 node-tuning-operator 中。对于 4.10 发行版本，PAO 的默认频道是 v4.10。您还可以在常规 PolicyGenTemplate CR 中指定默认频道。

d.

将 PolicyGenTemplate CR 更新推送到 GitOps ZTP Git 存储库。

ArgoCD 从 Git 存储库拉取更改并在 hub 集群上生成策略。

e.

运行以下命令检查创建的策略：

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. 在启动 Operator 更新前，应用所需的目录源更新。

   a. 使用目录源策略将名为 operator-upgrade-prep 的 ClusterGroupUpgrade CR 的内容保存到 cgu-operator-upgrade-prep.yml 文件中：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

   b. 运行以下命令，将策略应用到 hub 集群：

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

   c. 监控更新过程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. 为 Operator 更新创建 ClusterGroupUpgrade CR，并将 spec.enable 字段设置为 false。

   a. 使用 du-upgrade-operator-catsrc-policy 策略和从常规 PolicyGenTemplate 创建的订阅策略，将 Operator 更新 ClusterGroupUpgrade CR 的内容保存到 cgu-operator-upgrade.yml 文件，如下例所示：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
```

```
- du-upgrade-operator-catsrc-policy 1
- common-subscriptions-policy 2
preCaching: false
clusters:
- spoke1
remediationStrategy:
  maxConcurrency: 1
enable: false
```

1 镜像预缓存功能需要该策略，以便从目录源检索 Operator 镜像。

2 策略包含 Operator 订阅。如果您遵循了参考 PolicyGenTemplates 的结构和内容，则所有 Operator 订阅都分组到 common-subscriptions-policy 策略中。

注意

一个 ClusterGroupUpgrade CR 只能从 ClusterGroupUpgrade CR 中包含的一个目录源中预缓存订阅策略中定义的 Operator 镜像。如果所需的 Operator 来自不同目录源，如 SRIOV-FEC Operator 示例，则必须使用 du-upgrade-fec-catsrc-policy 和 du-upgrade-subscriptions-fec-policy 镜像（pre-FEC Operator 镜像）创建另一个 ClusterGroupUpgrade CR。

b. 运行以下命令，将 ClusterGroupUpgrade CR 应用到 hub 集群：

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. 可选：缓存 Operator 更新的镜像。

a. 在启动镜像预缓存前，运行以下命令验证订阅策略在此时是否是 NonCompliant ：

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

输出示例

```
NAME                          REMEDIATION ACTION   COMPLIANCE STATE     AGE
common-subscriptions-policy   inform               NonCompliant         27d
```

b. 运行以下命令，在 ClusterGroupUpgrade CR 中启用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

c. 监控进程并等待预缓存完成。在受管集群中运行以下命令来检查预缓存的状态：

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

d. 运行以下命令，检查预缓存是否在启动更新前完成：

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
```

输出示例

```
[
  {
   "lastTransitionTime": "2022-03-08T20:49:08.000Z",
   "message": "The ClusterGroupUpgrade CR is not enabled",
   "reason": "UpgradeNotStarted",
   "status": "False",
   "type": "Ready"
  },
  {
   "lastTransitionTime": "2022-03-08T20:55:30.000Z",
   "message": "Precaching is completed",
   "reason": "PrecachingCompleted",
   "status": "True",
   "type": "PrecachingDone"
  }
]
```

5. 启动 Operator 更新。

a.

运行以下命令，启用 cgu-operator-upgrade ClusterGroupUpgrade CR，并禁用预缓存来启动 Operator 更新：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

其他资源

- 升级 GitOps ZTP

10.3.4. 使用 PolicyGenTemplate CR 对丢失的 Operator 更新进行故障排除

在某些情况下，Topology Aware Lifecycle Manager (TALM)可能会因为过时的策略合规状态而丢失 Operator 更新。

在目录源更新后，Operator Lifecycle Manager (OLM)需要时间来更新订阅状态。当 TALM 决定是否需要补救时，订阅策略的状态可能会继续显示为合规。因此，订阅策略中指定的 Operator 不会升级。

要避免这种情况，请在 PolicyGenTemplate 中添加另一个目录源配置，并为需要更新的任何 Operator 在订阅中指定此配置。

流程

1. 在 PolicyGenTemplate 资源中添加目录源配置：

```
- fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "operator-catsrc-policy"
    metadata:
      name: redhat-operators
    spec:
      displayName: Red Hat Operators Catalog
      image: registry.example.com:5000/olm/redhat-operators:v{product-version}
      updateStrategy:
        registryPoll:
          interval: 1h
```

```
    status:
      connectionState:
        lastObservedState: READY
- fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "operator-catsrc-policy"
    metadata:
      name: redhat-operators-v2 (1)
    spec:
      displayName: Red Hat Operators Catalog v2 (2)
      image: registry.example.com:5000/olredhat-operators:<version> (3)
      updateStrategy:
        registryPoll:
          interval: 1h
    status:
      connectionState:
        lastObservedState: READY
```

(1) 更新新配置的名称。

(2) 更新新配置的显示名称。

(3) 更新索引镜像 URL。此 fileName.spec.image 字段覆盖 DefaultCatsrc.yaml 文件中的任何配置。

2. 更新 Subscription 资源，以指向需要更新的 Operator 的新配置：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: operator-subscription
  namespace: operator-namspace
# ...
spec:
  source: redhat-operators-v2 (1)
# ...
```

(1) 输入您在 PolicyGenTemplate 资源中定义的额外目录源配置的名称。

10.3.5. 一起执行平台和 Operator 更新

您可以同时执行平台和 Operator 更新。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 将 GitOps Zero Touch Provisioning (ZTP) 更新至最新版本。
- 使用 GitOps ZTP 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 按照 "forming a platform update" 和 "Performing an Operator update" 部分所述的步骤为更新创建 PolicyGenTemplate CR。
2. 为平台和 Operator 更新应用准备工作。
   a. 使用平台更新准备工作、目录源更新和目标集群的 ClusterGroupUpgrade CR 将内容保存到 cgu-platform-operator-upgrade-prep.yml 文件中，例如：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
```

```
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

b. 运行以下命令，将 cgu-platform-operator-upgrade-prep.yml 文件应用到 hub 集群：

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

c. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

3. 为平台创建 ClusterGroupUpdate CR，并将 spec.enable 字段设置为 false 的 Operator 更新。

a. 将平台的内容和带有策略和目标集群的 Operator 更新 ClusterGroupUpdate CR 保存为 cgu-platform-operator-upgrade.yml 文件，如下例所示：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade 1
  - du-upgrade-operator-catsrc-policy 2
  - common-subscriptions-policy 3
  preCaching: true
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1 这是平台更新策略。

2 这是包含要更新 Operator 的目录源信息的策略。预缓存功能需要它来确定要下载至受管集群的 Operator 镜像。

3

b. 运行以下命令，将 cgu-platform-operator-upgrade.yml 文件应用到 hub 集群：

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. 可选：为平台和 Operator 更新缓存镜像。

a. 运行以下命令，在 ClusterGroupUpgrade CR 中启用预缓存：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-
upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 监控更新过程，并等待预缓存完成。在受管集群中运行以下命令来检查预缓存的状态：

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

c. 运行以下命令，检查预缓存是否在启动更新前完成：

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5. 启动平台和 Operator 更新。

a. 运行以下命令，启用 cgu-du-upgrade ClusterGroupUpgrade CR 来启动平台和 Operator 更新：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-
upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. 监控进程。在完成后，运行以下命令来确保策略兼容：

```
$ oc get policies --all-namespaces
```

注意

可通过将设置配置为 **spec.enable: true**，从开始创建平台和 **Operator** 更新 **CR**。在这种情况下，更新会在预缓存完成后立即启动，且不需要手动启用 **CR**。

预缓存和更新都创建额外的资源，如策略、放置规则、放置规则、受管集群操作和受管集群视图，以帮助完成这个过程。将 **afterCompletion.deleteObjects** 字段设置为 **true** 在更新完成后删除所有这些资源。

10.3.6. 使用 PolicyGenTemplate CR 从部署的集群中删除 Performance Addon Operator 订阅

在早期版本的 **OpenShift Container Platform** 中，**Performance Addon Operator** 为应用程序提供了自动、低延迟的性能调整。在 **OpenShift Container Platform 4.11** 或更高版本中，这些功能是 **Node Tuning Operator** 的一部分。

不要在运行 **OpenShift Container Platform 4.11** 或更高版本的集群中安装 **Performance Addon Operator**。如果您升级到 **OpenShift Container Platform 4.11** 或更高版本，**Node Tuning Operator** 会自动删除 **Performance Addon Operator**。

注意

您需要删除创建 **Performance Addon Operator** 订阅的任何策略，以防止重新安装 **Operator**。

参考 **DU** 配置集在 **PolicyGenTemplate CR common-ranGen.yaml** 中包含 **Performance Addon Operator**。要从部署的受管集群中删除订阅，您必须更新 **common-ranGen.yaml**。

注意

如果在 **OpenShift Container Platform 4.11** 或更高版本上安装 **Performance Addon Operator 4.10.3-5** 或更高版本，**Performance Addon Operator** 会检测到集群版本并自动休眠，以避免与 **Node Tuning Operator** 正常工作。但是，为了确保获得最佳性能，请从 **OpenShift Container Platform 4.11** 集群中删除 **Performance Addon Operator**。

先决条件

- 创建一个 **Git** 存储库，在其中管理自定义站点配置数据。存储库必须可从 **hub** 集群访问，并

定义为 Argo CD 的源存储库。

- 更新至 OpenShift Container Platform 4.11 或更高版本。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在 common-ranGen.yaml 文件中，为 Performance Addon Operator 命名空间、Operator 组和订阅的 complianceType 更改为 mustnothave。

```
- fileName: PaoSubscriptionNS.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscription.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
```

2. 将更改与自定义站点存储库合并，并等待 ArgoCD 应用程序对 hub 集群同步更改。common-subscriptions-policy 策略的状态更改为 Non-Compliant。

3. 使用 Topology Aware Lifecycle Manager 将更改应用到您的目标集群。有关滚动配置更改的更多信息，请参阅“附加资源”部分。

4. 监控进程。当目标集群的 common-subscriptions-policy 策略的状态为 Compliant 时，Performance Addon Operator 已从集群中移除。运行以下命令，获取 common-subscriptions-policy 的状态：

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

5. 从 common-ranGen.yaml 文件中的 spec.sourceFiles 中删除 Performance Addon Operator 命名空间、Operator 组和订阅 CR。

6. 将更改与自定义站点存储库合并，并等待 ArgoCD 应用程序对 hub 集群同步更改。策略保持合规。

10.3.7. 在单节点 OpenShift 集群中使用 TALM 预缓存用户指定的镜像

在升级应用程序前，您可以在单节点 OpenShift 集群上预缓存应用程序相关的工作负载镜像。

您可以使用以下自定义资源(CR)指定预缓存作业的配置选项：

- PreCachingConfig CR
- ClusterGroupUpgrade CR

注意

PreCachingConfig CR 中的所有字段都是可选的。

PreCachingConfig CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: exampleconfig-ns
spec:
  overrides: 1
    platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    operatorsIndexes:
      - registry.example.com:5000/custom-redhat-operators:1.0.0
    operatorsPackagesAndChannels:
      - local-storage-operator: stable
      - ptp-operator: stable
      - sriov-network-operator: stable
  spaceRequired: 30 Gi 2
  excludePrecachePatterns: 3
    - aws
    - vsphere
  additionalImages: 4
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808f
c925ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808f
c925ada29c559a47adfaef
```

```
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafde
dga0991234afdadfsa09
```

1

默认情况下，TALM 会自动填充 platformImage、operatorIndexes 和受管集群策略中的 operatorsPackagesAndChannels 字段。您可以指定值来覆盖这些字段的默认 TALM-derived 值。

2

指定集群上的最低磁盘空间。如果未指定，TALM 为 OpenShift Container Platform 镜像定义一个默认值。磁盘空间字段必须包含整数值和存储单元。例如：40 GiB、200 MB、1 TiB。

3

根据镜像名称匹配，指定要从预缓存中排除的镜像。

4

指定要预缓存的额外镜像列表。

带有 PreCachingConfig CR 引用的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
spec:
  preCaching: true 1
  preCachingConfigRef:
    name: exampleconfig 2
    namespace: exampleconfig-ns 3
```

1

preCaching 字段设置为 true 可启用预缓存作业。

2

3

preCachingConfigRef.namespace 指定您要使用的 PreCachingConfig CR 的命名空间。

10.3.7.1. 为预缓存创建自定义资源

您必须在 ClusterGroupUpgrade CR 之前或同时创建 PreCachingConfig CR。

1. 使用您要预缓存的额外镜像列表创建 PreCachingConfig CR。

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: default (1)
spec:
[...]
  spaceRequired: 30Gi (2)
  additionalImages:
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
```

1

namespace 必须可以被 hub 集群访问。

2

建议设置最小磁盘空间所需字段，以确保预缓存镜像有足够的存储空间。

2. 创建一个 ClusterGroupUpgrade CR，并将 preCaching 字段设置为 true 并指定上一步中创建的 PreCachingConfig CR：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
```

```
metadata:
  name: cgu
  namespace: default
spec:
  clusters:
  - sno1
  - sno2
  preCaching: true
  preCachingConfigRef:
  - name: exampleconfig
    namespace: default
  managedPolicies:
    - du-upgrade-platform-upgrade
    - du-upgrade-operator-catsrc-policy
    - common-subscriptions-policy
  remediationStrategy:
    timeout: 240
```

警告

在集群上安装镜像后，您无法更改或删除它们。

3. 当您要启动预缓存镜像时，请运行以下命令应用 ClusterGroupUpgrade CR：

```
$ oc apply -f cgu.yaml
```

TALM 验证 ClusterGroupUpgrade CR。

此时，您可以继续 TALM 预缓存工作流。

注意

所有站点都同时预缓存。

验证

1. 运行以下命令，检查应用 ClusterUpgradeGroup CR 的 hub 集群上的预缓存状态：

```
$ oc get cgu <cgu_name> -n <cgu_namespace> -oyaml
```

输出示例

```
  precaching:
    spec:
      platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2
e1ef
      operatorsIndexes:
        - registry.example.com:5000/custom-redhat-operators:1.0.0
      operatorsPackagesAndChannels:
        - local-storage-operator: stable
        - ptp-operator: stable
        - sriov-network-operator: stable
      excludePrecachePatterns:
        - aws
        - vsphere
      additionalImages:
        -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47e2e1ef
        -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47adfaef
        -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340
adfafdedga0991234afdadfsa09
      spaceRequired: "30"
    status:
      sno1: Starting
      sno2: Starting
```

通过检查受管策略是否存在预缓存配置来验证预缓存配置。ClusterGroupUpgrade 和 PreCachingConfig CR 的有效配置会导致以下状态：

有效 CR 的输出示例

```
- lastTransitionTime: "2023-01-01T00:00:01Z"
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: "True"
  type: ClusterSelected
- lastTransitionTime: "2023-01-01T00:00:02Z"
```

```
  message: Completed validation
  reason: ValidationCompleted
  status: "True"
  type: Validated
- lastTransitionTime: "2023-01-01T00:00:03Z"
  message: Precaching spec is valid and consistent
  reason: PrecacheSpecIsWellFormed
  status: "True"
  type: PrecacheSpecValid
- lastTransitionTime: "2023-01-01T00:00:04Z"
  message: Precaching in progress for 1 clusters
  reason: InProgress
  status: "False"
  type: PrecachingSucceeded
```

无效的 PreCachingConfig CR 示例

```
Type:    "PrecacheSpecValid"
Status:  False,
Reason:  "PrecacheSpecIncomplete"
Message: "Precaching spec is incomplete: failed to get PreCachingConfig resource
due to PreCachingConfig.ran.openshift.io "<pre-caching_cr_name>" not found"
```

2. 您可以在受管集群中运行以下命令来查找预缓存作业：

```
$ oc get jobs -n openshift-talo-pre-cache
```

预缓存作业正在进行的示例

```
NAME        COMPLETIONS      DURATION      AGE
pre-cache   0/1              1s            1s
```

3. 您可以运行以下命令来检查为预缓存作业创建的 pod 状态：

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

预缓存作业正在进行的示例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate   19s    job-controller  Created pod: pre-cache-abcd1
```

4. 您可以运行以下命令来获取作业状态的实时更新：

```
$ oc logs -f pre-cache-abcd1 -n openshift-talo-pre-cache
```

5. 要验证预缓存作业是否已成功完成，请运行以下命令：

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

完成的预缓存作业示例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate   5m19s  job-controller  Created pod: pre-cache-abcd1
Normal     Completed         19s    job-controller  Job completed
```

6. 要验证镜像是否在单节点 OpenShift 上成功预缓存，请执行以下操作：

   a. 以 debug 模式进入节点：

   ```
   $ oc debug node/cnfdf00.example.lab
   ```

   b. 将 root 更改为 host ：

   ```
   $ chroot /host/
   ```

c. 搜索所需的镜像：

```
$ sudo podman images | grep <operator_name>
```

其他资源

- 使用容器镜像预缓存功能

10.3.8. 关于为 GitOps ZTP 自动创建的 ClusterGroupUpgrade CR

TALM 有一个名为 ManagedClusterForCGU 的控制器，它监控 hub 集群上的 ManagedCluster CR 的 Ready 状态，并为 GitOps Zero Touch Provisioning (ZTP) 创建 ClusterGroupUpgrade CR。

对于没有应用 ztp-done 标签的 Ready 状态中的任何受管集群，ManagedClusterForCGU 控制器会在 ztp-install 命名空间中创建一个带有在 GitOps ZTP 进程中创建的关联 RHACM 策略的 ClusterGroupUpgrade CR。然后，TALM 会修复自动创建 ClusterGroupUpgrade CR 中列出的一组配置策略，将配置 CR 推送到受管集群。

如果集群变为 Ready 时，没有用于受管集群的策略，则会创建一个没有策略的 ClusterGroupUpgrade CR。完成 ClusterGroupUpgrade 受管集群后，受管集群被标记为 ztp-done。如果要对该受管集群应用策略，请手动创建一个 ClusterGroupUpgrade 作为第 2 天操作。

GitOps ZTP 自动创建的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
```

```
    addClusterLabels:
      ztp-done: "" 1
    deleteClusterLabels:
      ztp-running: ""
    deleteObjects: true
  beforeEnable:
    addClusterLabels:
      ztp-running: "" 2
clusters:
- spoke1
enable: true
managedPolicies:
- common-spoke1-config-policy
- common-spoke1-subscriptions-policy
- group-spoke1-config-policy
- spoke1-config-policy
- group-spoke1-validator-du-policy
preCaching: false
remediationStrategy:
  maxConcurrency: 1
  timeout: 240
```

1 当 TALM 完成集群配置时，应用到受管集群。

2 当 TALM 开始部署配置策略时，应用到受管集群。

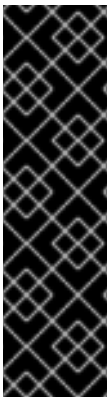
第 11 章 在 POLICYGENERATOR 或 POLICYGENTEMPLATE CR 中使用 HUB 模板

Topology Aware Lifecycle Manager 支持在 GitOps Zero Touch Provisioning (ZTP) 的配置策略中支持部分 Red Hat Advanced Cluster Management (RHACM) hub 集群模板功能。

hub-side 集群模板允许您定义可动态自定义到目标集群的配置策略。这可减少为具有辅助配置但具有不同值的很多集群创建单独的策略的需求。

重要

策略模板仅限于与定义策略的命名空间相同的命名空间。这意味着，您必须在创建策略的同一命名空间中创建 hub 模板中引用的对象。



重要

使用 PolicyGenTemplate CR 管理和监控对受管集群的策略将在即将发布的 OpenShift Container Platform 发行版本中弃用。使用 Red Hat Advanced Cluster Management (RHACM)和 PolicyGenerator CR 提供了等效和改进的功能。

有关 PolicyGenerator 资源的更多信息，请参阅 RHACM 策略生成器 文档。

其他资源

- 使用 PolicyGenerator 资源配置受管集群策略
- 比较 RHACM 策略生成器和 PolicyGenTemplate 资源补丁

11.1. 在配置策略中使用 RHACM HUB 集群模板

Topology Aware Lifecycle Manager 支持在 GitOps Zero Touch Provisioning (ZTP) 的配置策略中支持部分 Red Hat Advanced Cluster Management (RHACM) hub 集群模板功能。

以下支持的 hub 模板功能可用于 TALM 的 GitOps ZTP：

- fromConfigmap 返回命名的 ConfigMap 资源中提供的 data 键的值。

注意

ConfigMap CR 有一个 1 MiB 大小限制。ConfigMap CR 的有效大小被 last-applied-configuration 注解进一步限制。要避免 last-applied-configuration 限制，请在模板 ConfigMap 中添加以下注解：

```
argocd.argoproj.io/sync-options: Replace=true
```

- base64enc 返回输入字符串的 base64 编码值
- base64dec 返回 base64 编码的输入字符串的解码值
- indent 返回输入字符串，并带有添加的缩进空格
- autoindent 返回输入字符串，并根据父模板中使用的空间添加空格
- toInt casts 并返回输入值的整数值
- toBool 将输入字符串转换为布尔值，并返回布尔值

各种 开源社区功能 也可用于 GitOps ZTP。

其他资源

- 在配置策略中支持 hub 集群模板的 RHACM 支持

11.2. HUB 模板示例

以下代码示例是有效的 hub 模板。每个模板都会从 default 命名空间的 ConfigMap CR 返回的其名称为 test-config 的值。

- 使用键 common-key 返回值：

```
{{hub fromConfigMap "default" "test-config" "common-key" hub}}
```

- 使用 .ManagedClusterName 字段的串联值和字符串 -name 返回一个字符串：

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName)
hub}}
```

- casts 并从 .ManagedClusterName 字段的串联值和字符串 -name 返回布尔值：

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName)
| toBool hub}}
```

- casts 并从 .ManagedClusterName 字段的串联值和字符串 -name 返回整数值：

```
{{hub (printf "%s-name" .ManagedClusterName) | fromConfigMap "default" "test-
config" | toInt hub}}
```

11.3. 在组 POLICYGENERATOR 或 POLICYGENTEMPLATE CR 中指定组和站点配置

您可以使用 hub 模板填充应用到受管集群的生成的策略中的组和站点值来管理带有 ConfigMap CR 的集群配置。在 site PolicyGenerator 或 PolicyGentemplate CR 中使用 hub 模板意味着您不需要为每个站点创建一个策略 CR。

您可以根据用例（如硬件类型或区域）将集群分组到不同的类别中。每个集群都应该有一个与集群所在的组或组对应的标签。如果您在不同的 ConfigMap CR 中管理每个组的配置值，则只需要一个组策略 CR，才能使用 hub 模板将更改应用到组中的所有集群。

以下示例演示了如何使用三个 ConfigMap CR 和一个 PolicyGenerator CR 将站点和组配置应用到按硬件类型和区域分组的集群。

注意

当您使用 fromConfigmap 功能时，printf 变量仅适用于模板资源 data 键字段。您不能将其与 name 和 namespace 字段一起使用。

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了管理自定义站点配置数据的 Git 存储库。存储库必须可从 hub 集群访问，并定义为 GitOps ZTP ArgoCD 应用程序的源存储库。

流程

1. 创建包含组和站点配置的三个 ConfigMap CR：

   a. 创建名为 group-hardware-types-configmap 的 ConfigMap CR，以存放特定于硬件的配置。例如：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: group-hardware-types-configmap
  namespace: ztp-group
  annotations:
    argocd.argoproj.io/sync-options: Replace=true (1)
data:
  # SriovNetworkNodePolicy.yaml
  hardware-type-1-sriov-node-policy-pfNames-1: "[\"ens5f0\"]"
  hardware-type-1-sriov-node-policy-pfNames-2: "[\"ens7f0\"]"
  # PerformanceProfile.yaml
  hardware-type-1-cpu-isolated: "2-31,34-63"
  hardware-type-1-cpu-reserved: "0-1,32-33"
  hardware-type-1-hugepages-default: "1G"
  hardware-type-1-hugepages-size: "1G"
  hardware-type-1-hugepages-count: "32"
```

   (1) 只有在 ConfigMap 大于 1 MiB 时，才需要 argocd.argoproj.io/sync-options 注解。

   b. 创建名为 group-zones-configmap 的 ConfigMap CR，以存放区域配置。例如：

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: group-zones-configmap
  namespace: ztp-group
data:
  # ClusterLogForwarder.yaml
  zone-1-cluster-log-fwd-outputs: "[{\"type\":\"kafka\", \"name\":\"kafka-open\",
\"url\":\"tcp://10.46.55.190:9092/test\"}]"
  zone-1-cluster-log-fwd-pipelines: "[{\"inputRefs\":[\"audit\", \"infrastructure\"],
\"labels\": {\"label1\": \"test1\", \"label2\": \"test2\", \"label3\": \"test3\", \"label4\":
\"test4\"}, \"name\": \"all-to-default\", \"outputRefs\": [\"kafka-open\"]}]"
```

c. 创建名为 site-data-configmap 的 ConfigMap CR，以存放特定于站点的配置。例如：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: site-data-configmap
  namespace: ztp-group
data:
  # SriovNetwork.yaml
  du-sno-1-zone-1-sriov-network-vlan-1: "140"
  du-sno-1-zone-1-sriov-network-vlan-2: "150"
```

注意

每个 ConfigMap CR 都必须与从组 PolicyGenerator CR 生成的策略位于同一个命名空间中。

2. 提交 Git 中的 ConfigMap CR，然后推送到由 Argo CD 应用程序监控的 Git 存储库。

3. 将硬件类型和区域标签应用到集群。以下命令适用于名为 du-sno-1-zone-1 的单个集群，选择的标签为 "hardware-type": "hardware-type-1" 和 "group-du-sno-zone": "zone-1":

```
$ oc patch managedclusters.cluster.open-cluster-management.io/du-sno-1-zone-1 --
type merge -p '{"metadata":{"labels":{"hardware-type": "hardware-type-1", "group-du-
sno-zone": "zone-1"}}}'
```

4. 根据您的要求，创建一个组 PolicyGenerator 或 PolicyGentemplate CR，它使用 hub 模板从 ConfigMap 对象获取所需的数据：

a. 创建组 PolicyGenerator CR。这个示例 PolicyGenerator CR 为与 policyDefaults.placement 字段中列出的标签匹配的集群配置日志、VLAN ID、NIC 和

Performance Profile：

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno-pgt
placementBindingDefaults:
  name: group-du-sno-pgt-placement-binding
policyDefaults:
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno-zone
          operator: In
          values:
            - zone-1
        - key: hardware-type
          operator: In
          values:
            - hardware-type-1
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: group-du-sno-pgt-group-du-sno-cfg-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "10"
    manifests:
      - path: source-crs/ClusterLogForwarder.yaml
        patches:
          - spec:
              outputs: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-outputs" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
              pipelines: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-pipelines" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
      - path: source-crs/PerformanceProfile-MCP-master.yaml
        patches:
          - metadata:
              name: openshift-node-performance-profile
            spec:
              additionalKernelArgs:
                - rcupdate.rcu_normal_after_boot=0
                - vfio_pci.enable_sriov=1
                - vfio_pci.disable_idle_d3=1
                - efi=runtime
```

```
            cpu:
              isolated: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-isolated" (index .ManagedClusterLabels "hardware-type")) hub}}'
              reserved: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-reserved" (index .ManagedClusterLabels "hardware-type")) hub}}'
            hugepages:
              defaultHugepagesSize: '{{hub fromConfigMap "" "group-hardware-
types-configmap" (printf "%s-hugepages-default" (index .ManagedClusterLabels
"hardware-type")) hub}}'
              pages:
                - count: '{{hub fromConfigMap "" "group-hardware-types-
configmap" (printf "%s-hugepages-count" (index .ManagedClusterLabels
"hardware-type")) | toInt hub}}'
                  size: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-hugepages-size" (index .ManagedClusterLabels "hardware-type"))
hub}}'
            realTimeKernel:
              enabled: true
  - name: group-du-sno-pgt-group-du-sno-sriov-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "100"
    manifests:
      - path: source-crs/SriovNetwork.yaml
        patches:
          - metadata:
              name: sriov-nw-du-fh
            spec:
              resourceName: du_fh
              vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-1" .ManagedClusterName) | toInt hub}}'
      - path: source-crs/SriovNetworkNodePolicy-MCP-master.yaml
        patches:
          - metadata:
              name: sriov-nnp-du-fh
            spec:
              deviceType: netdevice
              isRdma: false
              nicSelector:
                pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-1" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
              numVfs: 8
              priority: 10
              resourceName: du_fh
      - path: source-crs/SriovNetwork.yaml
        patches:
          - metadata:
              name: sriov-nw-du-mh
            spec:
              resourceName: du_mh
              vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-2" .ManagedClusterName) | toInt hub}}'
      - path: source-crs/SriovNetworkNodePolicy-MCP-master.yaml
        patches:
          - metadata:
              name: sriov-nw-du-fh
```

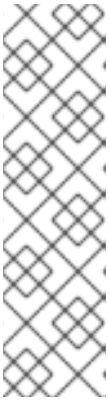
```
          spec:
            deviceType: netdevice
            isRdma: false
            nicSelector:
              pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-2" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
            numVfs: 8
            priority: 10
            resourceName: du_fh
```

b. 创建组 PolicyGenTemplate CR。此 PolicyGenTemplate CR 示例为与 spec.bindingRules 下列出的标签匹配的集群配置日志、VLAN ID、NIC 和 Performance Profile：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: group-du-sno-pgt
  namespace: ztp-group
spec:
  bindingRules:
    # These policies will correspond to all clusters with these labels
    group-du-sno-zone: "zone-1"
    hardware-type: "hardware-type-1"
  mcp: "master"
  sourceFiles:
    - fileName: ClusterLogForwarder.yaml # wave 10
      policyName: "group-du-sno-cfg-policy"
      spec:
        outputs: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-outputs" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
        pipelines: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-pipelines" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'

    - fileName: PerformanceProfile.yaml # wave 10
      policyName: "group-du-sno-cfg-policy"
      metadata:
        name: openshift-node-performance-profile
      spec:
        additionalKernelArgs:
        - rcupdate.rcu_normal_after_boot=0
        - vfio_pci.enable_sriov=1
        - vfio_pci.disable_idle_d3=1
        - efi=runtime
        cpu:
          isolated: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-cpu-isolated" (index .ManagedClusterLabels "hardware-type")) hub}}'
          reserved: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-reserved" (index .ManagedClusterLabels "hardware-type")) hub}}'
        hugepages:
          defaultHugepagesSize: '{{hub fromConfigMap "" "group-hardware-types-
```

```
configmap" (printf "%s-hugepages-default" (index .ManagedClusterLabels
"hardware-type")) hub}}'
        pages:
         - size: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-hugepages-size" (index .ManagedClusterLabels "hardware-type")) hub}}'
           count: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-hugepages-count" (index .ManagedClusterLabels "hardware-type")) | toInt
hub}}'
      realTimeKernel:
        enabled: true

  - fileName: SriovNetwork.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-fh
    spec:
      resourceName: du_fh
      vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-1" .ManagedClusterName) | toInt hub}}'

  - fileName: SriovNetworkNodePolicy.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nnp-du-fh
    spec:
      deviceType: netdevice
      isRdma: false
      nicSelector:
        pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-1" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
      numVfs: 8
      priority: 10
      resourceName: du_fh

  - fileName: SriovNetwork.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-mh
    spec:
      resourceName: du_mh
      vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-2" .ManagedClusterName) | toInt hub}}'

  - fileName: SriovNetworkNodePolicy.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-fh
    spec:
      deviceType: netdevice
      isRdma: false
      nicSelector:
        pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-2" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
```

```
      numVfs: 8
      priority: 10
      resourceName: du_fh
```



注意

要检索特定于站点的配置值，请使用 .ManagedClusterName 字段。这是一个模板上下文值设置为目标受管集群的名称。

要检索特定于组的配置，请使用 .ManagedClusterLabels 字段。这是一个模板上下文值设置为受管集群标签的值。

5. 提交 Git 中的 site PolicyGenerator 或 PolicyGentemplate CR，并推送到由 ArgoCD 应用程序监控的 Git 存储库。

注意

对引用的 ConfigMap CR 的后续更改不会自动同步到应用的策略。您需要手动同步新的 ConfigMap 更改来更新现有的 PolicyGenerator CR。请参阅 "Syncing new ConfigMap changes to existing PolicyGenerator 或 PolicyGenTemplate CR"。

您可以将相同的 PolicyGenerator 或 PolicyGentemplate CR 用于多个集群。如果有配置更改，则唯一需要进行修改的 ConfigMap 对象是保存每个集群配置和受管集群标签的 ConfigMap 对象。

11.4. 将新 CONFIGMAP 更改同步到现有的 POLICYGENERATOR 或 POLICYGENTEMPLATE CR

先决条件

- 已安装 OpenShift CLI(oc)。
- 已以具有 cluster-admin 权限的用户身份登录到 hub 集群。
- 您已创建了 PolicyGenerator 或 PolicyGentemplate CR，它使用 hub 集群模板从 ConfigMap CR 中拉取信息。

流程

1. 更新 ConfigMap CR 的内容，并应用 hub 集群中的更改。

2. 要将更新的 ConfigMap CR 的内容同步到部署的策略中，请执行以下操作之一：

   a. 选项 1：删除现有策略。ArgoCD 使用 PolicyGenerator 或 PolicyGentemplate CR 立即重新创建已删除的策略。例如，运行以下命令：

   ```
   $ oc delete policy <policy_name> -n <policy_namespace>
   ```

   b. 选项 2：在每次更新 ConfigMap 时，每次更新 ConfigMap 时，将特殊注解 policy.open-cluster-management.io/trigger-update 应用到策略。例如：

   ```
   $ oc annotate policy <policy_name> -n <policy_namespace> policy.open-cluster-management.io/trigger-update="1"
   ```

   注意

   您必须应用更新的策略才能使更改生效。如需更多信息，请参阅重新处理的特殊注解。

3. 可选：如果存在，删除包含策略的 ClusterGroupUpdate CR。例如：

   ```
   $ oc delete clustergroupupgrade <cgu_name> -n <cgu_namespace>
   ```

   a. 创建新的 ClusterGroupUpdate CR，其中包含要应用更新的 ConfigMap 更改的策略。例如，将以下 YAML 添加到文件 cgr-example.yaml 中：

   ```
   apiVersion: ran.openshift.io/v1alpha1
   kind: ClusterGroupUpgrade
   metadata:
     name: <cgr_name>
     namespace: <policy_namespace>
   spec:
     managedPolicies:
       - <managed_policy>
     enable: true
     clusters:
     - <managed_cluster_1>
     - <managed_cluster_2>
   ```

```
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

b. 应用更新的策略：

```
$ oc apply -f cgr-example.yaml
```

第 12 章 使用 TOPOLOGY AWARE LIFECYCLE MANAGER 更新受管集群

您可以使用 Topology Aware Lifecycle Manager (TALM) 来管理多个集群的软件生命周期。TALM 使用 Red Hat Advanced Cluster Management(RHACM)策略在目标集群中进行更改。

重要

在 GitOps ZTP 中使用 PolicyGenerator 资源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。

12.1. 关于 TOPOLOGY AWARE LIFECYCLE MANAGER 配置

Topology Aware Lifecycle Manager(TALM)管理一个或多个 OpenShift Container Platform 集群的部署 Red Hat Advanced Cluster Management(RHACM)策略。通过在大型集群网络中使用 TALM，可以使用有限制的批处理，在集群中逐步实施相关的策略。这有助于最大程度降低更新时可能造成的服务中断。使用 TALM，您可以控制以下操作：

- 更新的时间
- RHACM 管理的集群数量
- 将策略应用到的受管集群的子集
- 集群的更新顺序
- 在集群中修复的一组策略
- 在集群中修复的策略顺序

- Canary 集群的分配

对于单节点 OpenShift，Topology Aware Lifecycle Manager (TALM) 提供以下功能：

- 在升级前创建部署的备份
- 有限带宽的集群预缓存镜像

TALM 支持编排 OpenShift Container Platform y-stream 和 z-stream 更新，以及 y-streams 和 z-streams 上的 day-two 操作。

12.2. 关于用于 TOPOLOGY AWARE LIFECYCLE MANAGER 的受管策略

Topology Aware Lifecycle Manager(TALM)使用 RHACM 策略进行集群更新。

TALM 可以用来管理任何策略 CR 的推出，其中 remediationAction 字段被设置为 inform。支持的用例包括：

- 手动创建用户策略 CR
- 从 PolicyGenerator 或 PolicyGentemplate 自定义资源定义(CRD)自动生成策略

对于使用手动批准更新 Operator 订阅的策略，TALM 提供了额外的功能来批准更新的 Operator 的安装。

有关受管策略的更多信息，请参阅 RHACM 文档中的策略概述。

其他资源

- 关于 PolicyGenerator CRD

12.3. 使用 WEB 控制台安装 TOPOLOGY AWARE LIFECYCLE MANAGER

您可以使用 OpenShift Container Platform Web 控制台安装 Topology Aware Lifecycle Manager。

先决条件

- 安装最新版本的 RHACM Operator。
- TALM 4.16 需要 RHACM 2.9 或更高版本。
- 使用断开连接的 registry 设置 hub 集群。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 从可用的 Operator 列表中选择 Topology Aware Lifecycle Manager，然后点 Install。
3. 保持默认的选择 Installation mode ["All namespaces on the cluster (default)"] 和 Installed Namespace ("openshift-operators") 以确保 Operator 已正确安装。
4. 点 Install。

验证

确认安装成功：

1. 进入到 Operators → Installed Operators 页面。
2. 检查 Operator 是否在 All Namespaces 命名空间中，其状态为 Succeeded。

如果 Operator 没有成功安装：

1. 导航到 Operators → Installed Operators 页面，并检查 Status 列中是否有任何错误或故障。

2. 进入到 Workloads → Pods 页面，检查 cluster-group-upgrades-controller-manager pod 中报告问题的容器日志。

12.4. 使用 CLI 安装 TOPOLOGY AWARE LIFECYCLE MANAGER

您可以使用 OpenShift CLI(oc)安装 Topology Aware Lifecycle Manager(TALM)。

先决条件

- 安装 OpenShift CLI (oc) 。
- 安装最新版本的 RHACM Operator。
- TALM 4.16 需要 RHACM 2.9 或更高版本。
- 使用断开连接的 registry 设置 hub 集群。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个 Subscription CR：

   a. 定义 Subscription CR 并保存 YAML 文件，如 talm-subscription.yaml ：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-topology-aware-lifecycle-manager-subscription
```

```
  namespace: openshift-operators
spec:
  channel: "stable"
  name: topology-aware-lifecycle-manager
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. 运行以下命令来创建 Subscription CR：

```
$ oc create -f talm-subscription.yaml
```

验证

1. 检查 CSV 资源来验证安装是否成功：

```
$ oc get csv -n openshift-operators
```

输出示例

```
NAME                                         DISPLAY                       VERSION
REPLACES                      PHASE
topology-aware-lifecycle-manager.4.16.x   Topology Aware Lifecycle Manager   4.16.x
Succeeded
```

2. 验证 TALM 是否正在运行：

```
$ oc get deploy -n openshift-operators
```

输出示例

```
NAMESPACE                              NAME                                     READY   UP-TO-
DATE   AVAILABLE   AGE
openshift-operators                    cluster-group-upgrades-controller-manager
1/1    1            1           14s
```

12.5. 关于 CLUSTERGROUPUPGRADE CR

Topology Aware Lifecycle Manager(TALM)为一组集群从 ClusterGroupUpgrade CR 构建补救计划。您可以在 ClusterGroupUpgrade CR 中定义以下规格：

- 组中的集群
- 阻塞 ClusterGroupUpgrade CR
- 适用的受管策略列表
- 并发更新数
- 适用的 Canary 更新
- 更新前和更新之后执行的操作
- 更新数据

您可以使用 ClusterGroupUpgrade CR 中的 enable 字段控制更新的开始时间。例如，如果您调度的维护窗口为 4 小时，您可以准备 ClusterGroupUpgrade CR，并将 enable 字段设置为 false。

您可以通过配置 spec.remediationStrategy.timeout 设置来设置超时，如下所示：

```
spec
  remediationStrategy:
        maxConcurrency: 1
        timeout: 240
```

您可以使用 batchTimeoutAction 来确定更新是否有集群发生的情况。您可以指定 continue 跳过失败的集群，并继续升级其他集群，或 abort 以停止所有集群的策略补救。超时后，TALM 删除所有 enforce 策略，以确保对集群不进行进一步的更新。

要应用更改，您可以将 enabled 字段设置为 true。

如需更多信息，请参阅"将更新策略应用到受管集群"部分。

当 TALM 通过对指定集群进行补救时，ClusterGroupUpgrade CR 可以为多个条件报告 true 或 false 状态。

注意

当 TALM 完成集群更新后，集群不会在同一 ClusterGroupUpgrade CR 控制下再次更新。在以下情况下，必须创建新的 ClusterGroupUpgrade CR：

- 当您需要再次更新集群时
- 当集群在更新后变为与 inform 策略不符合时

12.5.1. 选择集群

TALM 构建补救计划并根据以下字段选择集群：

- clusterLabelSelector 字段指定您要更新的集群标签。这由来自 k8s.io/apimachinery/pkg/apis/meta/v1 的标准标签选择器的列表组成。列表中的每个选择器都使用标签值对或标签表达式。来自每个选择器的匹配会添加到集群的最终列表中，以及来自 clusterSelector 字段和 cluster 字段的匹配项。
- clusters 字段指定要更新的集群列表。
- canaries 字段指定集群进行 Canary 更新。
- maxConcurrency 字段指定批处理中要更新的集群数量。
- actions 字段指定 TALM 在启动更新过程时执行的 beforeEnable 操作，以及 TALM 在完成每个集群策略补救时执行的 afterCompletion 操作。

您可以使用 clusters、clusterLabelSelector 和 clusterSelector 字段来创建组合的集群列表。

补救计划从 canaries 字段中列出的集群开始。每个 canary 集群组成一个集群批处理。

Sample ClusterGroupUpgrade CR，带有 the enabled field 设置为 false

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion: 1
      addClusterLabels:
        upgrade-done: ""
      deleteClusterLabels:
        upgrade-running: ""
      deleteObjects: true
    beforeEnable: 2
      addClusterLabels:
        upgrade-running: ""
  backup: false
  clusters: 3
    - spoke1
  enable: false 4
  managedPolicies: 5
    - talm-policy
  preCaching: false
  remediationStrategy: 6
    canaries: 7
      - spoke1
    maxConcurrency: 2 8
    timeout: 240
  clusterLabelSelectors: 9
    - matchExpressions:
      - key: label1
      operator: In
      values:
        - value1a
```

```
      - value1b
  batchTimeoutAction: 10
status: 11
  computedMaxConcurrency: 2
  conditions:
   - lastTransitionTime: '2022-11-18T16:27:15Z'
     message: All selected clusters are valid
     reason: ClusterSelectionCompleted
     status: 'True'
     type: ClustersSelected 12
   - lastTransitionTime: '2022-11-18T16:27:15Z'
     message: Completed validation
     reason: ValidationCompleted
     status: 'True'
     type: Validated 13
   - lastTransitionTime: '2022-11-18T16:37:16Z'
     message: Not enabled
     reason: NotEnabled
     status: 'False'
     type: Progressing
  managedPoliciesForUpgrade:
   - name: talm-policy
     namespace: talm-namespace
  managedPoliciesNs:
    talm-policy: talm-namespace
  remediationPlan:
   - - spoke1
   - - spoke2
     - spoke3
  status:
```

1

指定 TALM 完成每个集群的策略补救时执行的操作。

2

指定 TALM 在开始更新过程时执行的操作。

3

定义要更新的集群列表。

4

enable 字段设置为 false。

5

6

定义集群更新的具体信息。

7

定义可用于 canary 更新的集群。

8

定义批处理中的最大并发更新数。补救批处理数量是 canary 集群的数量，加上除 Canary 集群外的集群数量除以 maxConcurrency 值。已兼容所有受管策略的集群不包括在补救计划中。

9

显示选择集群的参数。

10

控制批处理超时时会发生什么。可能的值有 abort 或 continue。如果未指定，则默认为 continue。

11

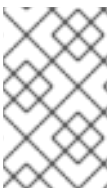
显示更新状态的信息。

12

ClustersSelected 条件显示所有选择的集群有效。

13

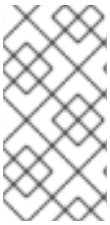
Validated 条件显示所有选择的集群都已验证。



注意

在更新 canary 集群的过程中任何错误都会停止更新过程。

当成功创建补救计划时，您可以将 enable 字段设置为 true，TALM 会开始使用指定的受管策略更新不合规的集群。



注意

只有 ClusterGroupUpgrade CR 的 enable 字段设置为 false 时，才能更改 spec 字段。

12.5.2. 验证

TALM 检查所有指定的受管策略是否可用并正确，并使用 Validated 条件来报告状态和原因：

- true

  验证已完成。

- false

  策略缺失或无效，或者指定了无效的平台镜像。

12.5.3. 预缓存

集群可能具有有限的带宽来访问容器镜像 registry，这可能会在更新完成前造成超时。在单节点 OpenShift 集群中，您可以使用预缓存来避免这种情况。当创建 ClusterGroupUpgrade CR 时，容器镜像预缓存会启动，并将 preCaching 字段设置为 true。TALM 将可用磁盘空间与预计 OpenShift Container Platform 镜像大小进行比较，以确保有足够的空间。如果集群没有足够的空间，TALM 会取消该集群的预缓存，且不会修复其上的策略。

TALM 使用 PrecacheSpecValid 条件来报告状态信息，如下所示：

- true

  预缓存规格有效且一致。

- false

  预缓存规格不完整。

TALM 使用 PrecachingSucceeded 条件来报告状态信息，如下所示：

- true

  TALM 已完成预缓存过程。如果任何集群的预缓存失败，则该集群的更新会失败，但会继续执行所有其他集群。如果任何集群预缓存失败，您会接收到一个通知信息。

- false

  预缓存仍在为一个或多个集群处理，或者所有集群都失败。

如需更多信息，请参阅"使用容器镜像预缓存功能"部分。

12.5.4. 创建备份

对于单节点 OpenShift，TALM 可以在更新前创建部署的备份。如果更新失败，您可以恢复之前的版本并将集群恢复到工作状态，而无需重新置备应用程序。要使用备份功能，您首先创建一个 ClusterGroupUpgrade CR，并将 backup 字段设置为 true。为确保备份内容为最新版本，在 ClusterGroupUpgrade CR 中的 enable 字段设置为 true 之前，不会进行备份。

TALM 使用 BackupSucceeded 条件来报告状态，如下所示：

- true

  备份对于所有集群都完成，或备份运行已完成但对一个或多个集群失败。如果任何集群的备份失败，则该集群的更新会失败，但会继续执行所有其他集群。

- false

  备份仍在为一个或多个集群处理，或者所有集群都失败。

如需更多信息，请参阅"在升级前创建集群资源备份"部分。

12.5.5. 更新集群

TALM 按照补救计划强制实施策略。在当前批处理的所有集群与所有受管策略兼容后，对后续批处理的策略强制启动。如果批处理超时，TALM 会进入下一个批处理。批处理的超时值是 spec.timeout 字段除以补救计划中的批处理数量。

TALM 使用 Progressing 条件来报告状态以及如下原因：

- true

  TALM 是补救不合规的策略。

- false

  更新没有进行。可能的原因包括：

  - 所有集群都符合所有受管策略。
  - 当策略补救用时过长时，更新会超时。
  - 阻塞系统中丢失或尚未完成的 CR。
  - ClusterGroupUpgrade CR 不会被启用。
  - 备份仍在进行中。

注意

受管策略会按照 ClusterGroupUpgrade CR 中的 managedPolicies 字段中列出的顺序进行应用。一个受管策略被应用于指定的集群。当集群符合当前策略时，会应用下一个受管策略。

处于 Progressing 状态的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: true
  remediationStrategy:
    canaries:
        - spoke1
    maxConcurrency: 2
    timeout: 240
  clusterLabelSelectors:
    - matchExpressions:
      - key: label1
        operator: In
        values:
          - value1a
          - value1b
  batchTimeoutAction:
status:
    clusters:
      - name: spoke1
        state: complete
    computedMaxConcurrency: 2
    conditions:
      - lastTransitionTime: '2022-11-18T16:27:15Z'
        message: All selected clusters are valid
        reason: ClusterSelectionCompleted
        status: 'True'
        type: ClustersSelected
      - lastTransitionTime: '2022-11-18T16:27:15Z'
        message: Completed validation
        reason: ValidationCompleted
        status: 'True'
        type: Validated
      - lastTransitionTime: '2022-11-18T16:37:16Z'
```

```
    message: Remediating non-compliant policies
    reason: InProgress
    status: 'True'
    type: Progressing 1
managedPoliciesForUpgrade:
  - name: talm-policy
    namespace: talm-namespace
managedPoliciesNs:
  talm-policy: talm-namespace
remediationPlan:
  - - spoke1
  - - spoke2
    - spoke3
status:
  currentBatch: 2
  currentBatchRemediationProgress:
    spoke2:
      state: Completed
    spoke3:
      policyIndex: 0
      state: InProgress
  currentBatchStartedAt: '2022-11-18T16:27:16Z'
  startedAt: '2022-11-18T16:27:15Z'
```

1

Progressing 字段显示 TALM 处于补救策略的过程。

12.5.6. 更新状态

TALM 使用 Succeeded 条件来报告状态和如下原因：

- true

  所有集群都符合指定的受管策略。

- false

  因为没有集群可用于补救，策略补救会失败，或者因为以下原因之一策略补救用时过长：

  - 在当前批处理包含 Canary 更新时，批处理中的集群不会遵循批处理超时中的所有受管

策略。

- 集群不符合 remediationStrategy 字段中指定的 timeout 值的受管策略。

处于 Succeeded 状态的 ClusterGroupUpgrade CR 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
  - spoke1
  - spoke4
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: 1
  clusters:
    - name: spoke1
      state: complete
    - name: spoke4
      state: complete
  conditions:
  - message: All selected clusters are valid
    reason: ClusterSelectionCompleted
    status: "True"
    type: ClustersSelected
  - message: Completed validation
    reason: ValidationCompleted
    status: "True"
    type: Validated
  - message: All clusters are compliant with all the managed policies
    reason: Completed
    status: "False"
    type: Progressing 2
  - message: All clusters are compliant with all the managed policies
    reason: Completed
    status: "True"
    type: Succeeded 3
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
```

```
    remediationPlan:
    - - spoke1
    - - spoke4
    status:
      completedAt: '2022-11-18T16:27:16Z'
      startedAt: '2022-11-18T16:27:15Z'
```

2 在 Progressing 字段中，更新完成时状态为 false ；集群与所有受管策略兼容。

3 Succeeded 字段显示验证成功完成。

1 status 字段包含集群列表及其状态。集群的状态可以是 complete 或 timedout。

timedout 状态的 Sample ClusterGroupUpgrade CR

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
    - spoke2
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: false
```

```
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
status:
  clusters:
    - name: spoke1
      state: complete
    - currentPolicy: (1)
        name: talm-policy
        status: NonCompliant
      name: spoke2
      state: timedout
  computedMaxConcurrency: 2
  conditions:
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: All selected clusters are valid
      reason: ClusterSelectionCompleted
      status: 'True'
      type: ClustersSelected
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: Completed validation
      reason: ValidationCompleted
      status: 'True'
      type: Validated
    - lastTransitionTime: '2022-11-18T16:37:16Z'
      message: Policy remediation took too long
      reason: TimedOut
      status: 'False'
      type: Progressing
    - lastTransitionTime: '2022-11-18T16:37:16Z'
      message: Policy remediation took too long
      reason: TimedOut
      status: 'False'
      type: Succeeded (2)
  managedPoliciesForUpgrade:
    - name: talm-policy
      namespace: talm-namespace
  managedPoliciesNs:
    talm-policy: talm-namespace
  remediationPlan:
    - - spoke1
      - spoke2
  status:
      startedAt: '2022-11-18T16:27:15Z'
      completedAt: '2022-11-18T20:27:15Z'
```

(1) 如果集群的状态是 timedout，currentPolicy 字段会显示策略名称和策略状态。

(2)

12.5.7. 阻塞 ClusterGroupUpgrade CR

您可以创建多个 ClusterGroupUpgrade CR，并控制应用程序的顺序。

例如，如果您创建了 ClusterGroupUpgrade CR C，它会阻塞 ClusterGroupUpgrade CR A 的启动，那么 ClusterGroupUpgrade CR A 将无法启动，直到 ClusterGroupUpgrade CR C 变为 UpgradeComplete 状态。

一个 ClusterGroupUpgrade CR 可以有多个阻塞 CR。在这种情况下，所有块 CR 都必须在升级当前 CR 升级前完成。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 将 ClusterGroupUpgrade CR 的内容保存到 cgu-a.yaml、cgu-b.yaml 和 cgu-c.yaml 文件中。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-c
    namespace: default
  clusters:
```

```
  - spoke1
  - spoke2
  - spoke3
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
```

1

定义阻塞 CR。cgu-a 更新无法启动，直到 cgu-c 完成后。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
```

```
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

1

cgu-b 更新无法启动，直到 cgu-a 完成后。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec: 1
  clusters:
  - spoke6
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
```

```
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status: {}
```

1. cgu-c 更新没有任何阻塞 CR。当 enable 字段设为 true 时，TALM 会启动 cgu-c 更新。

2. 通过为每个相关 CR 运行以下命令创建 ClusterGroupUpgrade CR：

```
$ oc apply -f <name>.yaml
```

3. 通过为每个相关 CR 运行以下命令启动更新过程：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/<name> \
--type merge -p '{"spec":{"enable":true}}'
```

以下示例显示 enable 字段设为 true 的 ClusterGroupUpgrade CR：

带有阻塞 CR 的 cgu-a 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs:
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-c]' (1)
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
  status: {}
```

1

显示阻塞 CR 的列表。

带有阻塞 CR 的 cgu-b 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs:
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-a]' 1
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
```

```
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

1

显示阻塞 CR 的列表。

带有阻塞 CR 的 cgu-c 示例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec:
  clusters:
  - spoke6
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant 1
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
```

```
- cgu-c-policy4-common-sriov-sub-policy
placementRules:
- cgu-c-policy1-common-cluster-version-policy
- cgu-c-policy4-common-sriov-sub-policy
remediationPlan:
- - spoke6
status:
  currentBatch: 1
  remediationPlanForBatch:
    spoke6: 0
```

1

cgu-c 更新没有任何阻塞 CR。

12.6. 更新受管集群上的策略

Topology Aware Lifecycle Manager (TALM)修复了在 ClusterGroupUpgrade 自定义资源(CR)中指定的集群的 inform 策略。TALM 通过 PlacementBinding CR 中的 bindingOverrides.remediationAction 和 subFilter 规格控制 Policy CR 中的 remediationAction 规格来修复 inform 策略。每个策略都有自己的对应的 RHACM 放置规则和 RHACM 放置绑定。

例如，TALM 将每个集群从当前批处理添加到与适用受管策略相对应的放置规则。如果集群已与策略兼容，TALM 会在兼容集群上跳过应用该策略。TALM 然后进入到将下一个策略应用到还没有合规的集群的步骤。TALM 在批处理中完成更新后，所有集群都会从与策略关联的放置规则中删除。然后，下一个批处理的更新会启动。

如果 spoke 集群没有向 RHACM 报告任何合规状态，则 hub 集群上的受管策略可能会缺少 TALM 需要的状态信息。TALM 通过以下方法处理这些情况：

- 如果缺少策略的 status.compliant 字段，TALM 忽略策略并添加日志条目。然后，TALM 继续查看策略的 status.status 字段。
- 如果缺少策略的 status.status，TALM 会生成错误。
- 如果策略的 status.status 字段中缺少集群的合规状态，TALM 会将该集群视为与该策略不兼容。

ClusterGroupUpgrade CR 的 batchTimeoutAction 决定升级失败时是否有什么情况。您可以指定 continue 跳过失败的集群，并继续升级其他集群，或者指定 abort 以停止所有集群的策略补救。超时后，TALM 会删除它创建的所有资源，以确保对集群不进行进一步的更新。

升级策略示例

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: ocp-4.4.16.4
  namespace: platform-upgrade
spec:
  disabled: false
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: upgrade
      spec:
        namespaceselector:
          exclude:
          - kube-*
          include:
          - '*'
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: config.openshift.io/v1
            kind: ClusterVersion
            metadata:
              name: version
            spec:
              channel: stable-4.16
              desiredUpdate:
                version: 4.4.16.4
              upstream: https://api.openshift.com/api/upgrades_info/v1/graph
            status:
              history:
                - state: Completed
                  version: 4.4.16.4
        remediationAction: inform
        severity: low
  remediationAction: inform
```

有关 RHACM 策略的更多信息，请参阅策略概述。

其他资源

- 关于 PolicyGenerator CRD

12.6.1. 使用 TALM 为安装的受管集群配置 Operator 订阅

Topology Aware Lifecycle Manager (TALM) 只能在 Operator 的 Subscription 自定义资源(CR) 包含 status.state.AtLatestKnown 字段时批准 Operator 的安装计划。

流程

1. 将 status.state.AtLatestKnown 字段添加到 Operator 的 Subscription CR 中：

Subscription CR 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: "stable"
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown 1
```

1. status.state: AtLatestKnown 字段用于 Operator 目录中可用的最新 Operator 版本。

注意

当 registry 中有新版本的 Operator 时，相关的策略将变为不合规。

2. 使用 ClusterGroupUpgrade CR 将更改的 Subscription 策略应用到受管集群。

12.6.2. 将更新策略应用到受管集群

您可以通过应用策略来更新受管集群。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- TALM 4.16 需要 RHACM 2.9 或更高版本。
- 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 在 hub 集群中创建 RHACM 策略。

流程

1. 将 ClusterGroupUpgrade CR 的内容保存到 cgu-1.yaml 文件中。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
  managedPolicies: 1
    - policy1-common-cluster-version-policy
    - policy2-common-nto-sub-policy
    - policy3-common-ptp-sub-policy
    - policy4-common-sriov-sub-policy
```

```
enable: false
clusters: 2
- spoke1
- spoke2
- spoke5
- spoke6
remediationStrategy:
  maxConcurrency: 2 3
  timeout: 240 4
batchTimeoutAction: 5
```

1 要应用的策略的名称。

2 要更新的集群列表。

3 maxConcurrency 字段表示同时更新的集群数量。

4 更新超时（以分钟为单位）。

5 控制批处理超时时会发生什么。可能的值有 abort 或 continue。如果未指定，则默认为 continue。

2. 运行以下命令来创建 ClusterGroupUpgrade CR：

```
$ oc create -f cgu-1.yaml
```

a. 运行以下命令，检查 hub 集群中是否已创建 ClusterGroupUpgrade CR：

```
$ oc get cgu --all-namespaces
```

输出示例

```
NAMESPACE   NAME  AGE  STATE      DETAILS
default     cgu-1 8m55 NotEnabled Not Enabled
```

b. 运行以下命令检查更新的状态：

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

输出示例

```
{
  "computedMaxConcurrency": 2,
  "conditions": [
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "Not enabled", 1
      "reason": "NotEnabled",
      "status": "False",
      "type": "Progressing"
    }
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-
tuning-operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-
operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "
[{\"kind\":\"Subscription\",\"name\":\"sriov-network-operator-
subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-nto-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
```

```
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {}
}
```

1 ClusterGroupUpgrade CR 中的 spec.enable 字段设置为 false。

3. 运行以下命令，将 spec.enable 字段的值更改为 true ：

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-1 \
--patch '{"spec":{"enable":true}}' --type=merge
```

验证

1. 运行以下命令检查更新的状态：

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

输出示例

```
{
  "computedMaxConcurrency": 2,
  "conditions": [ 1
    {
      "lastTransitionTime": "2022-02-25T15:33:07Z",
      "message": "All selected clusters are valid",
      "reason": "ClusterSelectionCompleted",
      "status": "True",
      "type": "ClustersSelected"
    },
    {
      "lastTransitionTime": "2022-02-25T15:33:07Z",
      "message": "Completed validation",
      "reason": "ValidationCompleted",
      "status": "True",
      "type": "Validated"
    },
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "Remediating non-compliant policies",
      "reason": "InProgress",
      "status": "True",
      "type": "Progressing"
    }
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-
tuning-operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-
operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-
network-operator-subscription\",\"namespace\":\"openshift-sriov-network-
operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-nto-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
```

```
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {
    "currentBatch": 1,
    "currentBatchRemediationProgress": {
      "spoke1": {
        "policyIndex": 1,
        "state": "InProgress"
      },
      "spoke2": {
        "policyIndex": 1,
        "state": "InProgress"
      }
    },
    "currentBatchStartedAt": "2022-02-25T15:54:16Z",
    "startedAt": "2022-02-25T15:54:16Z"
  }
}
```


1

反映当前批处理的更新进度。再次运行该命令以接收有关进度的更新信息。

2. 运行以下命令，检查策略的状态：

```
oc get policies -A
```

输出示例

```
NAMESPACE   NAME                                    REMEDIATION ACTION   COMPLIANCE
STATE    AGE
spoke1    default.policy1-common-cluster-version-policy enforce          Compliant
18m
spoke1    default.policy2-common-nto-sub-policy       enforce            NonCompliant
18m
spoke2    default.policy1-common-cluster-version-policy enforce          Compliant
18m
spoke2    default.policy2-common-nto-sub-policy       enforce            NonCompliant
18m
spoke5    default.policy3-common-ptp-sub-policy       inform             NonCompliant
18m
spoke5    default.policy4-common-sriov-sub-policy     inform             NonCompliant
18m
spoke6    default.policy3-common-ptp-sub-policy       inform             NonCompliant
18m
spoke6    default.policy4-common-sriov-sub-policy     inform             NonCompliant
18m
default   policy1-common-ptp-sub-policy               inform             Compliant
18m
default   policy2-common-sriov-sub-policy             inform             NonCompliant
18m
default   policy3-common-ptp-sub-policy               inform             NonCompliant
18m
default   policy4-common-sriov-sub-policy             inform             NonCompliant
18m
```

- spec.remediationAction 值会更改为从当前批处理应用到集群的子策略 强制 值。
- spec.remedationAction 值会针对剩余的集群中的子策略保持 inform。

- 批处理完成后，spec.remediationAction 值会重新更改为 inform 强制执行的子策略。

3. 如果策略包含 Operator 订阅，您可以在单节点集群中直接检查安装进度。

   a. 运行以下命令，导出用于检查安装的单节点集群的 KUBECONFIG 文件：

   ```
   $ export KUBECONFIG=<cluster_kubeconfig_absolute_path>
   ```

   b. 运行以下命令，检查单节点集群中存在的所有订阅，并在您要通过 ClusterGroupUpgrade CR 安装的策略中查找您要通过 ClusterGroupUpgrade CR 安装的订阅：

   ```
   $ oc get subs -A | grep -i <subscription_name>
   ```

   cluster-logging 策略的输出示例

   ```
   NAMESPACE                    NAME                 PACKAGE
   SOURCE            CHANNEL
   openshift-logging            cluster-logging      cluster-logging
   redhat-operators  stable
   ```

4. 如果其中一个受管策略包含 ClusterVersion CR，则根据 spoke 集群运行以下命令来检查当前批处理中的平台更新状态：

   ```
   $ oc get clusterversion
   ```

   输出示例

   ```
   NAME     VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
   version  4.4.16.5  True        True          43s     Working towards 4.4.16.7: 71 of 735 done
   (9% complete)
   ```

5. 运行以下命令检查 Operator 订阅：

```
$ oc get subs -n <operator-namespace> <operator-subscription> -ojsonpath="
{.status}"
```

6. 运行以下命令，检查与所需订阅关联的单节点集群中是否存在安装计划：

```
$ oc get installplan -n <subscription_namespace>
```

cluster-logging Operator 的输出示例

```
NAMESPACE                    NAME            CSV                        APPROVAL
APPROVED
openshift-logging            install-6khtw   cluster-logging.5.3.3-4    Manual
true (1)
```

(1) 安装计划在 TALM 批准安装计划后将其 Approval 字段设置为 Manual，其 Approved 字段会从 false 改为 true。

注意

当 TALM 修复包含订阅的策略时，它会自动批准附加到该订阅的任何安装计划。如果需要多个安装计划将 Operator 升级到最新的已知版本，TALM 可能会批准多个安装计划，通过一个或多个中间版本进行升级以进入最终版本。

7. 运行以下命令，检查正在安装 ClusterGroupUpgrade 的策略的 Operator 的集群服务版本是否已进入 Succeeded 阶段：

```
$ oc get csv -n <operator_namespace>
```

OpenShift Logging Operator 的输出示例

```
NAME                 DISPLAY                VERSION   REPLACES   PHASE
cluster-logging.5.4.2   Red Hat OpenShift Logging   5.4.2             Succeeded
```

12.7. 在升级前创建集群资源备份

对于单节点 OpenShift，Topology Aware Lifecycle Manager (TALM) 可以在升级前创建部署备份。如果升级失败，您可以恢复之前的版本并将集群恢复到工作状态，而无需重新置备应用程序。

要使用备份功能，您首先创建一个 ClusterGroupUpgrade CR，并将 backup 字段设置为 true。为确保备份内容为最新版本，在 ClusterGroupUpgrade CR 中的 enable 字段设置为 true 之前，不会进行备份。

TALM 使用 BackupSucceeded 条件来报告状态，如下所示：

- true

  备份对于所有集群都完成，或备份运行已完成但对一个或多个集群失败。如果任何集群的备份失败，则不会为该集群进行更新。

- false

  备份仍在为一个或多个集群处理，或者所有集群都失败。在 spoke 集群中运行的备份过程可以具有以下状态：

  - PreparingToStart

    第一个协调通过正在进行。TALM 删除所有 spoke 备份命名空间和 hub 查看在升级尝试中创建的资源。

  - Starting

    正在创建备份先决条件和备份作业。

- **Active**

  备份正在进行。

- **Succeeded**

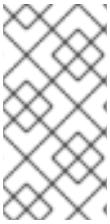
  备份成功。

- **BackupTimeout**

  工件备份部分完成。

- **UnrecoverableError**

  备份以非零退出代码结尾。



注意

如果集群备份失败，且进入 BackupTimeout 或 UnrecoverableError 状态，集群更新不会对集群进行。对其他集群的更新不会受到影响，并继续。

12.7.1. 使用备份创建 ClusterGroupUpgrade CR

您可以在单节点 OpenShift 集群上升级前创建部署备份。如果升级失败，您可以使用 Topology Aware Lifecycle Manager (TALM) 生成的 upgrade-recovery.sh 脚本将系统返回到其 preupgrade 状态。备份由以下项目组成：

集群备份

etcd 和静态 pod 清单的快照。

内容备份

文件夹备份，例如 /etc、/usr/local、/var/lib/kubelet。

已更改的文件备份

由 machine-config 管理的任何文件都已更改。

Deployment

固定 ostree 部署。

镜像（可选）

使用的任何容器镜像。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 Red Hat Advanced Cluster Management (RHACM)。

注意

强烈建议您创建一个恢复分区。以下是一个恢复分区的 SiteConfig 自定义资源 (CR) 示例，大小为 50 GB：

```
nodes:
   - hostName: "node-1.example.com"
   role: "master"
   rootDeviceHints:
      hctl: "0:2:0:0"
      deviceName: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
...
   #Disk /dev/disk/by-id/scsi-3600508b400105e210000900000490000:
   #893.3 GiB, 959119884288 bytes, 1873281024 sectors
   diskPartition:
      - device: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
      partitions:
      - mount_point: /var/recovery
         size: 51200
         start: 800000
```

流程

1. 在 clustergroupupgrades-group-du.yaml 文件中保存 ClusterGroupUpgrade CR 的内容，并 backup 和 enable 字段设置为 true：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true
  backup: true
  clusters:
  - cnfdb1
  - cnfdb2
  enable: true
  managedPolicies:
  - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

2. 要启动更新，请运行以下命令来应用 ClusterGroupUpgrade CR：

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

验证

- 运行以下命令，检查 hub 集群中的升级状态：

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

输出示例

```
{
   "backup": {
      "clusters": [
         "cnfdb2",
         "cnfdb1"
   ],
   "status": {
      "cnfdb1": "Succeeded",
      "cnfdb2": "Failed" 1
   }
},
```

```
"computedMaxConcurrency": 1,
"conditions": [
  {
    "lastTransitionTime": "2022-04-05T10:37:19Z",
    "message": "Backup failed for 1 cluster", 2
    "reason": "PartiallyDone", 3
    "status": "True", 4
    "type": "Succeeded"
  }
],
"precaching": {
  "spec": {}
},
"status": {}
```

1 对一个集群进行备份失败。

2 消息确认一个集群的备份失败。

3 备份部分成功。

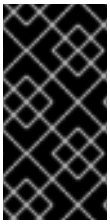
4 备份过程已完成。

12.7.2. 在升级后恢复集群

如果集群的升级失败，您可以手动登录到集群，并使用备份使集群返回到其升级前的状态。有两个阶段：

回滚（Rollback）

如果尝试升级包括对平台操作系统部署的更改，则必须在运行恢复脚本前回滚到以前的版本。



重要

回滚仅适用于从 TALM 和单节点 OpenShift 升级。这个过程不适用于从任何其他升级类型进行回滚。

恢复

恢复会关闭容器，并使用备份分区中的文件来重新启动容器并恢复集群。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 置备一个或多个受管集群。
- 安装 Red Hat Advanced Cluster Management (RHACM)。
- 以具有 cluster-admin 特权的用户身份登录。
- 运行为备份而配置的升级。

流程

1. 运行以下命令来删除之前创建的 ClusterGroupUpgrade 自定义资源 (CR)：

```
$ oc delete cgu/du-upgrade-4918 -n ztp-group-du-sno
```

2. 登录到要恢复的集群。
3. 运行以下命令，检查平台操作系统部署的状态：

```
$ ostree admin status
```

输出示例

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9.0
    Version: 49.84.202202230006-0
    Pinned: yes (1)
    origin refspec:
c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9
```

(1) 当前部署已被固定。不需要平台操作系统部署回滚。

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa.0
    Version: 410.84.202204050541-0
    origin refspec:
f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa
rhcos ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca.0
(rollback) (1)
    Version: 410.84.202203290245-0
    Pinned: yes (2)
    origin refspec:
ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca
```

(1) 此平台操作系统部署标记为回滚。

(2) 以前的部署已被固定，可以回滚。

4. 要触发平台操作系统部署的回滚，请运行以下命令：

```
$ rpm-ostree rollback -r
```

5. 恢复的第一阶段会关闭容器，并将文件从备份分区恢复到目标目录。要开始恢复，请运行以下命令：

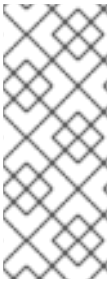
```
$ /var/recovery/upgrade-recovery.sh
```

6. 提示时，运行以下命令重启集群：

```
$ systemctl reboot
```

7. 重新引导后，运行以下命令重启恢复：

```
$ /var/recovery/upgrade-recovery.sh  --resume
```



注意

如果恢复工具失败，您可以使用 --restart 选项重试：

```
$ /var/recovery/upgrade-recovery.sh --restart
```

验证

- 运行以下命令检查恢复的状态：

```
$ oc get clusterversion,nodes,clusteroperator
```

输出示例

```
NAME                                         VERSION   AVAILABLE   PROGRESSING   SINCE
STATUS
clusterversion.config.openshift.io/version   4.4.16.23   True        False         86d
Cluster version is 4.4.16.23 1


NAME                        STATUS   ROLES          AGE   VERSION
node/lab-test-spoke1-node-0   Ready    master,worker   86d   v1.22.3+b93fd35 2

NAME                                                       VERSION   AVAILABLE
PROGRESSING   DEGRADED   SINCE   MESSAGE
clusteroperator.config.openshift.io/authentication                   4.4.16.23   True
False         False      2d7h    3
clusteroperator.config.openshift.io/baremetal                        4.4.16.23   True
False         False      86d


.............
```

1

集群版本可用，并具有正确的版本。

2

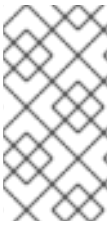
节点状态为 Ready。

3

ClusterOperator 对象的可用性为 True。

12.8. 使用容器镜像预缓存功能

单节点 OpenShift 集群可能有限带宽来访问容器镜像 registry，这可能会在更新完成前造成超时。



注意

TALM 不会设置更新的时间。您可以在通过手动应用程序或外部自动化进行更新时应用 ClusterGroupUpgrade CR。

当 preCaching 字段在 ClusterGroupUpgrade CR 中被设置为 true 时，容器镜像预缓存会启动。

TALM 使用 PrecacheSpecValid 条件来报告状态信息，如下所示：

- true

  预缓存规格有效且一致。

- false

  预缓存规格不完整。

TALM 使用 PrecachingSucceeded 条件来报告状态信息，如下所示：

- true

  TALM 已完成预缓存过程。如果任何集群的预缓存失败，则该集群的更新会失败，但会继续执行所有其他集群。如果任何集群预缓存失败，您会接收到一个通知信息。

- false

  预缓存仍在为一个或多个集群处理，或者所有集群都失败。

在成功预缓存后，您可以启动补救策略。当 enable 字段设置为 true 时，补救操作会启动。如果集群中存在预缓存失败，则对该集群的升级会失败。升级过程将继续用于成功预缓存的所有其他集群。

预缓存过程可以处于以下状态：

- NotStarted

  这是所有集群在第一次协调时会自动分配给 ClusterGroupUpgrade CR 的初始状态。在这个状态中，TALM 会删除来自之前更新中所有 spoke 集群的预缓存命名空间和 hub 查看资源。然后，TALM 为 spoke 创建一个新的 ManagedClusterView 资源，以便在 PrecachePreparing 状态验证删除。

- PreparingToStart

  清理之前不完整更新中的所有剩余的资源，资源正在进行中。

- Starting

  预缓存任务前提条件并创建了作业。

- Active

该作业的状态为"Active"状态。

- Succeeded

  pre-cache（预缓存）作业成功。

- PrecacheTimeout

  工件预缓存是部分完成的。

- UnrecoverableError

  作业以非零退出代码结束。

12.8.1. 使用容器镜像预缓存过滤器

预缓存功能通常下载比集群进行更新所需要镜像更多的镜像。您可以控制将哪些预缓存镜像下载到集群中。这可缩短下载时间，并节省带宽和存储。

您可以使用以下命令查看要下载的所有镜像的列表：

```
$ oc adm release info <ocp-version>
```

以下 ConfigMap 示例演示了如何使用 excludePrecachePatterns 字段排除镜像。

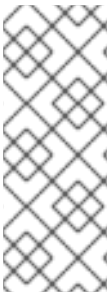
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-group-upgrade-overrides
data:
  excludePrecachePatterns: |
    azure (1)
    aws
    vsphere
    alibaba
```

(1)

TALM 排除所有带有名称的镜像，其中包括此处列出的任何模式。

12.8.2. 使用预缓存创建 ClusterGroupUpgrade CR

对于单节点 OpenShift，在更新启动前，预缓存功能允许在 spoke 集群上存在所需的容器镜像。



注意

对于预缓存，TALM 使用 ClusterGroupUpgrade CR 中的 spec.remediationStrategy.timeout 值。您必须设置一个 timeout 值，允许足够时间完成预缓存作业。当您在预缓存完成后启用 ClusterGroupUpgrade CR 时，您可以将 timeout 值改为适合更新的持续时间。

先决条件

- 安装 Topology Aware Lifecycle Manager(TALM)。
- 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在 clustergroupupgrades-group-du.yaml 文件中将 preCaching 字段设置为 true 来保存 ClusterGroupUpgrade CR 的内容：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true 1
  clusters:
  - cnfdb1
  - cnfdb2
  enable: false
  managedPolicies:
  - du-upgrade-platform-upgrade
```

```
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

1

preCaching 字段设为 true，它允许 TALM 在开始更新前拉取容器镜像。

2. 当您要启动预缓存时，请运行以下命令应用 ClusterGroupUpgrade CR：

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

验证

1. 运行以下命令，检查 hub 集群中是否存在 ClusterGroupUpgrade CR：

```
$ oc get cgu -A
```

输出示例

```
NAMESPACE          NAME             AGE   STATE       DETAILS
ztp-group-du-sno   du-upgrade-4918   10s   InProgress   Precaching is required and not
done 1
```

1

CR 被创建。

2. 运行以下命令，检查预缓存任务的状态：

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

输出示例

```
{
```

```
 "conditions": [
  {
   "lastTransitionTime": "2022-01-27T19:07:24Z",
   "message": "Precaching is required and not done",
   "reason": "InProgress",
   "status": "False",
   "type": "PrecachingSucceeded"
  },
  {
   "lastTransitionTime": "2022-01-27T19:07:34Z",
   "message": "Pre-caching spec is valid and consistent",
   "reason": "PrecacheSpecIsWellFormed",
   "status": "True",
   "type": "PrecacheSpecValid"
  }
 ],
 "precaching": {
  "clusters": [
   "cnfdb1" 1
   "cnfdb2"
  ],
  "spec": {
   "platformImage": "image.example.io"},
  "status": {
   "cnfdb1": "Active"
   "cnfdb2": "Succeeded"}
  }
}
```

1 显示已识别的集群列表。

3. 在 spoke 集群中运行以下命令来检查预缓存作业的状态：

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

输出示例

```
NAME                COMPLETIONS   DURATION   AGE
job.batch/pre-cache   0/1           3m10s      3m10s

NAME                  READY   STATUS    RESTARTS   AGE
pod/pre-cache--1-9bmlr   1/1     Running   0          3m10s
```

4. 运行以下命令，检查 ClusterGroupUpgrade CR 的状态：

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

输出示例

```
"conditions": [
   {
    "lastTransitionTime": "2022-01-27T19:30:41Z",
    "message": "The ClusterGroupUpgrade CR has all clusters compliant with all the
managed policies",
    "reason": "UpgradeCompleted",
    "status": "True",
    "type": "Ready"
   },
   {
    "lastTransitionTime": "2022-01-27T19:28:57Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
    "type": "PrecachingSucceeded" 1
   }
```

1 预缓存任务已完成。

12.9. 对 TOPOLOGY AWARE LIFECYCLE MANAGER 进行故障排除

Topology Aware Lifecycle Manager(TALM)是一个 OpenShift Container Platform Operator，用于修复 RHACM 策略。出现问题时，使用 oc adm must-gather 命令来收集详情和日志，并采取调试问题的步骤。

有关相关主题的更多信息，请参阅以下文档：

- Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix
- Red Hat Advanced Cluster Management 故障排除
- "故障排除 Operator 问题"部分

12.9.1. 常规故障排除

您可以通过查看以下问题来确定问题的原因：

- 您要应用的配置是否被支持？
  - RHACM 和 OpenShift Container Platform 版本是否兼容？
  - TALM 和 RHACM 版本是否兼容？
- 以下哪个组件导致了此问题？
  - 第 12.9.3 节 “受管策略”
  - 第 12.9.4 节 “Clusters”
  - 第 12.9.5 节 “补救策略”
  - 第 12.9.6 节 “Topology Aware Lifecycle Manager”

为确保 ClusterGroupUpgrade 配置可以正常工作，您可以执行以下操作：

1. 创建 ClusterGroupUpgrade CR，并将 spec.enable 字段设置为 false。

2. 等待状态更新，再完成故障排除问题。

3. 如果所有内容都如预期，在 ClusterGroupUpgrade CR 中将 spec.enable 字段设置为 true。

警告

在 ClusterUpgradeGroup CR 中将 spec.enable 字段设置为 true 后，更新过程会启动，您无法再编辑 CR 的 spec 字段。

12.9.2. 无法修改 ClusterUpgradeGroup CR

问题

在启用更新后，您无法编辑 ClusterUpgradeGroup CR。

解决方案

通过执行以下步骤来重启操作：

1. 运行以下命令删除旧 ClusterGroupUpgrade CR：

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. 检查并修复受管集群和策略的现有问题。

   a. 确保所有集群都是受管集群并可用。

   b. 确保所有策略都存在，并将 spec.remediationAction 字段设置为 inform。

3. 使用正确的配置创建一个新的 ClusterGroupUpgrade CR。

```
$ oc apply -f <ClusterGroupUpgradeCR_YAML>
```

12.9.3. 受管策略

检查系统中的受管策略

问题

您需要检查系统中是否有正确的受管策略。

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

输出示例

```
["group-du-sno-validator-du-validator-policy", "policy2-common-nto-sub-policy", "policy3-common-ptp-sub-policy"]
```

检查 remediationAction 模式

问题

您要检查在受管策略的 spec 中是否将 remediationAction 字段设置为 inform。

解决方案

运行以下命令：

```
$ oc get policies --all-namespaces
```

输出示例

```
NAMESPACE   NAME                                        REMEDIATION ACTION   COMPLIANCE STATE   AGE
default     policy1-common-cluster-version-policy       inform               NonCompliant
```

```
5d21h
default    policy2-common-nto-sub-policy                inform           Compliant
5d21h
default    policy3-common-ptp-sub-policy                inform           NonCompliant
5d21h
default    policy4-common-sriov-sub-policy               inform           NonCompliant
5d21h
```

检查策略合规状态

问题

您需要检查策略的合规性状态。

解决方案

运行以下命令：

```
$ oc get policies --all-namespaces
```

输出示例

```
NAMESPACE   NAME                                  REMEDIATION ACTION   COMPLIANCE
STATE   AGE
default    policy1-common-cluster-version-policy          inform           NonCompliant
5d21h
default    policy2-common-nto-sub-policy                inform           Compliant
5d21h
default    policy3-common-ptp-sub-policy                inform           NonCompliant
5d21h
default    policy4-common-sriov-sub-policy               inform           NonCompliant
5d21h
```

12.9.4. Clusters

检查是否有受管集群

问题

您需要检查 ClusterGroupUpgrade CR 中的集群是受管集群。

解决方案

运行以下命令：

```
$ oc get managedclusters
```

输出示例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.example.com:6443      True    Unknown     13d
spoke1          true           https://api.spoke1.example.com:6443   True    True        13d
spoke3          true           https://api.spoke3.example.com:6443   True    True        27h
```

1. 或者，检查 TALM manager 日志：

   a. 运行以下命令，获取 TALM Manager 的名称：

   ```
   $ oc get pod -n openshift-operators
   ```

   输出示例

   ```
   NAME                                         READY   STATUS    RESTARTS   AGE
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp   2/2     Running
   0          45m
   ```

   b. 运行以下命令检查 TALM manager 日志：

   ```
   $ oc logs -n openshift-operators \
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
   ```

输出示例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler
error {"reconciler group": "ran.openshift.io", "reconciler kind":
"ClusterGroupUpgrade", "name": "lab-upgrade", "namespace": "default",
"error": "Cluster spoke5555 is not a ManagedCluster"} (1)
sigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).processNextWorkItem
```

(1) 错误消息显示集群不是受管集群。

检查受管集群是否可用

问题

您需要检查 ClusterGroupUpgrade CR 中指定的受管集群是否可用。

解决方案

运行以下命令：

```
$ oc get managedclusters
```

输出示例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.testlab.com:6443      True    Unknown     13d
spoke1          true           https://api.spoke1.testlab.com:6443   True    True        13d (1)
spoke3          true           https://api.spoke3.testlab.com:6443   True    True        27h (2)
```

(1) (2)

受管集群的 AVAILABLE 字段的值是 True。

检查 clusterLabelSelector

问题

您需要检查 ClusterGroupUpgrade CR 中指定的 clusterLabelSelector 字段是否至少与其中一个受管集群匹配。

解决方案

运行以下命令：

```
$ oc get managedcluster --selector=upgrade=true 1
```

1

要更新的集群标签是 upgrade:true。

输出示例

```
NAME     HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED   AVAILABLE   AGE
spoke1   true           https://api.spoke1.testlab.com:6443   True     True        13d
spoke3   true           https://api.spoke3.testlab.com:6443   True     True        27h
```

检查是否有 canary 集群

问题

您要检查集群列表中是否存在 Canary 集群。

ClusterGroupUpgrade CR 示例

```
spec:
  remediationStrategy:
```

```
    canaries:
    - spoke3
    maxConcurrency: 2
    timeout: 240
  clusterLabelSelectors:
   - matchLabels:
      upgrade: true
```

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.clusters}'
```

输出示例

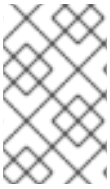
```
["spoke1", "spoke3"]
```

1. 运行以下命令，检查与 clusterLabelSelector 标签匹配的集群列表中是否存在 Canary 集群：

```
$ oc get managedcluster --selector=upgrade=true
```

输出示例

```
NAME     HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE   AGE
spoke1   true           https://api.spoke1.testlab.com:6443   True     True        13d
spoke3   true           https://api.spoke3.testlab.com:6443   True     True        27h
```



注意

集群可以存在于 spec.clusters 中，还可与 spec.clusterLabelSelector 标签匹配。

检查 spoke 集群上的预缓存状态

1. 在 spoke 集群中运行以下命令来检查预缓存的状态：

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

12.9.5. 补救策略

检查 ClusterGroupUpgrade CR 中是否存在 remediationStrategy

问题

您需要检查 ClusterGroupUpgrade CR 是否存在 remediationStrategy。

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

输出示例

```
{"maxConcurrency":2, "timeout":240}
```

检查 ClusterGroupUpgrade CR 中是否指定了 maxConcurrency

问题

您需要检查是否在 ClusterGroupUpgrade CR 中指定 maxConcurrency。

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy.maxConcurrency}'
```

-

输出示例

```
2
```

12.9.6. Topology Aware Lifecycle Manager

检查 ClusterGroupUpgrade CR 中的条件消息和状态

问题

您要检查 ClusterGroupUpgrade CR 中的 status.conditions 字段的值。

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

输出示例

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"Missing managed policies:
[policyList]", "reason":"NotAllManagedPoliciesExist", "status":"False", "type":"Validated"}
```

检查 status.remediationPlan 是否已计算

问题

您需要检查 status.remediationPlan 是否被计算。

解决方案

运行以下命令：

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.remediationPlan}'
```

输出示例

```
[["spoke2", "spoke3"]]
```

TALM manager 容器中的错误

问题

您要检查 TALM 的 manager 容器的日志。

解决方案

运行以下命令：

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

输出示例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error
{"reconciler group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade",
"name": "lab-upgrade", "namespace": "default", "error": "Cluster spoke5555 is not a
ManagedCluster"} (1)
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

(1) 显示错误。

在 ClusterGroupUpgrade CR 完成后，集群可能不符合一些策略

问题

TALM 用来决定是否需要补救的策略合规状态，还没有为所有集群完全更新。这可能是因为：

- 在策略创建或更新后，CGU 很快就会运行。
- 策略的补救会影响 ClusterGroupUpgrade CR 中后续策略的合规性。

解决方案

创建并应用具有相同规格的新 ClusterGroupUpdate CR。

在 GitOps ZTP 工作流中自动创建 ClusterGroupUpgrade CR 没有受管策略

问题

如果集群变为 Ready 时，没有用于受管集群的策略，则会自动创建一个没有策略的 ClusterGroupUpgrade CR。完成 ClusterGroupUpgrade CR 后，受管集群被标记为 ztp-done。如果在推送 SiteConfig 资源后，如果 PolicyGenerator 或 PolicyGenTemplate CR 没有推送到 Git 存储库，则当集群变为 Ready 时，这可能会导致目标集群没有可用的策略。

解决方案

验证您要应用的策略是否在 hub 集群中可用，然后使用所需策略创建一个 ClusterGroupUpgrade CR。

您可以手动创建 ClusterGroupUpgrade CR，或再次触发自动创建。要触发 ClusterGroupUpgrade CR 的自动创建，请从集群中删除 ztp-done 标签，并删除之前在 zip-install 命名空间中创建的空 ClusterGroupUpgrade CR。

预缓存失败

问题

预缓存可能会因为以下原因之一失败：

- 节点上没有足够的可用空间。
- 对于断开连接的环境，预缓存镜像没有正确镜像。
- 创建 pod 时存在问题。

解决方案

1. 要检查预缓存是否已因为空间不足而失败，请检查节点中预缓存 pod 的日志。

   a. 使用以下命令查找 pod 的名称：

   ```
   $ oc get pods -n openshift-talo-pre-cache
   ```

   b. 使用以下命令检查日志以查看错误是否与足够空间相关：

   ```
   $ oc logs -n openshift-talo-pre-cache <pod name>
   ```

2. 如果没有日志，使用以下命令检查 pod 状态：

   ```
   $ oc describe pod -n openshift-talo-pre-cache <pod name>
   ```

3. 如果 pod 不存在，请检查作业状态以查看它无法使用以下命令创建 pod 的原因：

   ```
   $ oc describe job -n openshift-talo-pre-cache pre-cache
   ```

其他资源

- OpenShift Container Platform 故障排除 Operator 问题
- 使用 Topology Aware Lifecycle Manager 更新受管策略
- 关于 PolicyGenerator CRD

第 13 章 使用 GITOPS ZTP 扩展单节点 OPENSHIFT 集群

您可以使用 GitOps Zero Touch Provisioning (ZTP) 扩展单节点 OpenShift 集群。将 worker 节点添加到单节点 OpenShift 集群时，原始单节点 OpenShift 集群会保留 control plane 节点角色。添加 worker 节点不需要现有单节点 OpenShift 集群的任何停机时间。

注意

虽然您可以添加到单节点 OpenShift 集群的 worker 节点数量没有指定的限制，但您必须为额外的 worker 节点重新评估 control plane 节点上的保留 CPU 分配。

如果需要 worker 节点上的工作负载分区，则必须在安装节点前在 hub 集群中部署并修复受管集群策略。这样，工作负载分区 MachineConfig 对象会被呈现，并在 GitOps ZTP 工作流将 MachineConfig ignition 文件应用到 worker 节点前与 worker 机器配置池相关联。

建议您首先修复策略，然后安装 worker 节点。如果在安装 worker 节点后创建工作负载分区清单，您必须手动排空该节点并删除由守护进程集管理的所有 pod。当管理守护进程集创建新 pod 时，新 pod 会处理工作负载分区过程。

重要

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

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

其他资源

- 有关为 vDU 应用程序部署调整的单节点 OpenShift 集群的更多信息，请参阅在单节点 OpenShift 中部署 vDU 的参考配置。
- 如需有关 worker 节点的更多信息，请参阅将 worker 节点添加到单节点 OpenShift 集群。
- 有关从扩展的单节点 OpenShift 集群中删除 worker 节点的详情，请参考使用命令行界面删除受管集群节点。

13.1. 使用 POLICYGENERATOR 或 POLICYGENTEMPLATE 资源将配置集应用到 WORKER 节点

您可以使用 DU 配置集配置额外的 worker 节点。

您可以使用 GitOps Zero Touch Provisioning (ZTP)通用、组和特定于站点的 PolicyGenerator 或 PolicyGenTemplate 资源，将 RAN 分布式单元(DU)配置集应用到 worker 节点集群。链接到 ArgoCD 策略 应用程序的 GitOps ZTP 管道包括以下 CR，您可以在提取 ztp-site-generate 容器时在相关的 out/argocd/example 文件夹中找到：

/acmpolicygenerator 资源

- acm-common-ranGen.yaml
- acm-group-du-sno-ranGen.yaml
- acm-example-sno-site.yaml
- ns.yaml
- kustomization.yaml

/policygentemplates resources

- common-ranGen.yaml
- group-du-sno-ranGen.yaml
- example-sno-site.yaml
- ns.yaml

- kustomization.yaml

在 worker 节点上配置 DU 配置集被视为升级。要启动升级流，您必须更新现有策略或创建额外的策略。然后，您必须创建一个 ClusterGroupUpgrade CR 来协调集群组中的策略。

13.2. 确保 PTP 和 SR-IOV 守护进程选择器兼容性

如果 DU 配置集使用 GitOps Zero Touch Provisioning (ZTP)插件版本 4.11 或更早版本部署，则 PTP 和 SR-IOV Operator 可能会被配置为仅将守护进程放在标记为 master 的节点上。此配置可防止 PTP 和 SR-IOV 守护进程在 worker 节点上运行。如果系统上正确配置了 PTP 和 SR-IOV 守护进程节点选择器，您必须更改守护进程，然后才能继续 worker DU 配置集配置。

流程

1. 在其中一个 spoke 集群中检查 PTP Operator 的守护进程节点选择器设置：

```
$ oc get ptpoperatorconfig/default -n openshift-ptp -ojsonpath='{.spec}' | jq
```

PTP Operator 的输出示例

```
{"daemonNodeSelector":{"node-role.kubernetes.io/master":""}} 1
```

1. 如果节点选择器设置为 master，则 spoke 会使用需要更改的 GitOps ZTP 插件的版本进行部署。

2. 在其中一个 spoke 集群中检查 SR-IOV Operator 的守护进程节点选择器设置：

```
$  oc get sriovoperatorconfig/default -n \
openshift-sriov-network-operator -ojsonpath='{.spec}' | jq
```

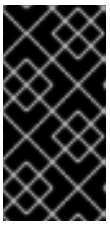
SR-IOV Operator 的输出示例

```
{"configDaemonNodeSelector":{"node-
role.kubernetes.io/worker":""},"disableDrain":false,"enableInjector":true,"enableOpera
torWebhook":true} (1)
```

(1) 如果节点选择器设置为 master，则 spoke 会使用需要更改的 GitOps ZTP 插件的版本进行部署。

3. 在组策略中，添加以下 complianceType 和 spec 条目：

```
spec:
  - fileName: PtpOperatorConfig.yaml
    policyName: "config-policy"
    complianceType: mustonlyhave
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/worker: ""
  - fileName: SriovOperatorConfig.yaml
    policyName: "config-policy"
    complianceType: mustonlyhave
    spec:
      configDaemonNodeSelector:
        node-role.kubernetes.io/worker: ""
```



重要

更改 daemonNodeSelector 字段会导致临时 PTP 同步丢失和 SR-IOV 连接丢失。

4. 提交 Git 中的更改，然后推送到由 GitOps ZTP ArgoCD 应用程序监控的 Git 存储库。

13.3. PTP 和 SR-IOV 节点选择器兼容性

PTP 配置资源和 SR-IOV 网络节点策略使用 node-role.kubernetes.io/master: "" 作为节点选择器。如果额外的 worker 节点与 control plane 节点具有相同的 NIC 配置，则用于配置 control plane 节点的策

略可以被 worker 节点重复使用。但是，节点选择器必须更改为选择两种节点类型，例如使用 "node-role.kubernetes.io/worker" 标签。

13.4. 使用 POLICYGENERATOR CR 将 WORKER 节点策略应用到 WORKER 节点

您可以使用 PolicyGenerator CR 为 worker 节点创建策略。

流程

1. 创建以下 PolicyGenerator CR：

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: example-sno-workers
placementBindingDefaults:
  name: example-sno-workers-placement-binding
policyDefaults:
  namespace: example-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: sites
          operator: In
          values:
            - example-sno 1
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: example-sno-workers-config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "10"
    manifests:
      - path: source-crs/MachineConfigGeneric.yaml 2
        patches:
          - metadata:
              labels:
                machineconfiguration.openshift.io/role: worker 3
              name: enable-workload-partitioning
            spec:
              config:
                storage:
```

```
                files:
                  - contents:
                      source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhd
GlvbI9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFu
YWdlbWVudCIKYW5ub3RhdGlvbI9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm
9wZW5zaGlmdC5pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiI
D0gIjAtMyIgfQo=
                    mode: 420
                    overwrite: true
                    path: /etc/crio/crio.conf.d/01-workload-partitioning
                    user:
                      name: root
                  - contents:
                      source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50ljogewogICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg
==
                    mode: 420
                    overwrite: true
                    path: /etc/kubernetes/openshift-workload-pinning
                    user:
                      name: root
      - path: source-crs/PerformanceProfile-MCP-worker.yaml
        patches:
          - metadata:
              name: openshift-worker-node-performance-profile
            spec:
              cpu: 4
                isolated: 4-47
                reserved: 0-3
              hugepages:
                defaultHugepagesSize: 1G
                pages:
                  - count: 32
                    size: 1G
              realTimeKernel:
                enabled: true
      - path: source-crs/TunedPerformancePatch-MCP-worker.yaml
        patches:
          - metadata:
              name: performance-patch-worker
            spec:
              profile:
                - data: |
                  [main]
                  summary=Configuration changes profile inherited from performance
created tuned
                  include=openshift-node-performance-openshift-worker-node-
performance-profile
                  [bootloader]
                  cmdline_crash=nohz_full=4-47 5
                  [sysctl]
                  kernel.timer_migration=1
                  [scheduler]
                  group.ice-ptp=0:f:10:*:ice-ptp.*
                  [service]
```

```
        service.stalld=start,enable
        service.chronyd=stop,disable
        name: performance-patch-worker
    recommend:
      - profile: performance-patch-worker
```

1 该策略应用于带有此标签的所有集群。

2 此通用 MachineConfig CR 用于在 worker 节点上配置工作负载分区。

3 MCP 字段必须设置为 worker。

4 必须为每个特定的硬件平台配置 cpu.isolated 和 cpu.reserved 字段。

5 cmdline_crash CPU 集必须与 PerformanceProfile 部分中设置的 cpu.isolated 匹配。

通用 MachineConfig CR 用于在 worker 节点上配置工作负载分区。您可以生成 crio 和 kubelet 配置文件的内容。

2. 将创建的策略模板添加到由 ArgoCD policies 应用程序监控的 Git 存储库中。

3. 在 kustomization.yaml 文件中添加策略。

4. 提交 Git 中的更改，然后推送到由 GitOps ZTP ArgoCD 应用程序监控的 Git 存储库。

5. 要将新策略修复到 spoke 集群，请创建一个 TALM 自定义资源：

```
$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
```

```
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
  - example-sno
  enable: true
  managedPolicies:
  - group-du-sno-config-policy
  - example-sno-workers-config-policy
  - example-sno-config-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
EOF
```

13.5. 使用 POLICYGENTEMPLATE CR 将 WORKER 节点策略应用到 WORKER 节点

您可以使用 PolicyGenTemplate CR 为 worker 节点创建策略。

流程

1. 创建以下 PolicyGenTemplate CR：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-sno-workers"
  namespace: "example-sno"
spec:
  bindingRules:
    sites: "example-sno" 1
  mcp: "worker" 2
  sourceFiles:
    - fileName: MachineConfigGeneric.yaml 3
      policyName: "config-policy"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: worker
        name: enable-workload-partitioning
      spec:
        config:
          storage:
            files:
            - contents:
                source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhd
GlvbI9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFu
YWdlbWVudCIKYW5ub3RhdGlvbI9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm
```

```
9wZW5zaGlmdC5pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMyIgfQo=
            mode: 420
            overwrite: true
            path: /etc/crio/crio.conf.d/01-workload-partitioning
            user:
              name: root
          - contents:
              source: data:text/plain;charset=utf-8;base64,ewogICJtYW5hZ2VtZW50Ijogewog ICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg==
            mode: 420
            overwrite: true
            path: /etc/kubernetes/openshift-workload-pinning
            user:
              name: root
    - fileName: PerformanceProfile.yaml
      policyName: "config-policy"
      metadata:
        name: openshift-worker-node-performance-profile
      spec:
        cpu: 4
          isolated: "4-47"
          reserved: "0-3"
        hugepages:
          defaultHugepagesSize: 1G
          pages:
            - size: 1G
              count: 32
        realTimeKernel:
          enabled: true
    - fileName: TunedPerformancePatch.yaml
      policyName: "config-policy"
      metadata:
        name: performance-patch-worker
      spec:
        profile:
          - name: performance-patch-worker
            data: |
              [main]
              summary=Configuration changes profile inherited from performance created tuned
              include=openshift-node-performance-openshift-worker-node-performance-profile
              [bootloader]
              cmdline_crash=nohz_full=4-47 5
              [sysctl]
              kernel.timer_migration=1
              [scheduler]
              group.ice-ptp=0:f:10:*:ice-ptp.*
              [service]
              service.stalld=start,enable
              service.chronyd=stop,disable
        recommend:
        - profile: performance-patch-worker
```

1 该策略应用于带有此标签的所有集群。

2 MCP 字段必须设置为 worker。

3 此通用 MachineConfig CR 用于在 worker 节点上配置工作负载分区。

4 必须为每个特定的硬件平台配置 cpu.isolated 和 cpu.reserved 字段。

5 cmdline_crash CPU 集必须与 PerformanceProfile 部分中设置的 cpu.isolated 匹配。

通用 MachineConfig CR 用于在 worker 节点上配置工作负载分区。您可以生成 crio 和 kubelet 配置文件的内容。

2. 将创建的策略模板添加到由 ArgoCD policies 应用程序监控的 Git 存储库中。

3. 在 kustomization.yaml 文件中添加策略。

4. 提交 Git 中的更改，然后推送到由 GitOps ZTP ArgoCD 应用程序监控的 Git 存储库。

5. 要将新策略修复到 spoke 集群，请创建一个 TALM 自定义资源：

```
$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
```

```
  - example-sno
  enable: true
  managedPolicies:
  - group-du-sno-config-policy
  - example-sno-workers-config-policy
  - example-sno-config-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
EOF
```

13.6. 使用 GITOPS ZTP 将 WORKER 节点添加到单节点 OPENSHIFT 集群

您可以将一个或多个 worker 节点添加到现有的单节点 OpenShift 集群，以增加集群中的可用 CPU 资源。

先决条件

- 在 OpenShift Container Platform 4.11 或更高版本的裸机 hub 集群中安装和配置 RHACM 2.6 或更高版本
- 在 hub 集群中安装 Topology Aware Lifecycle Manager
- 在 hub 集群中安装 Red Hat OpenShift GitOps
- 使用 GitOps ZTP ztp-site-generate 容器镜像版本 4.12 或更高版本
- 使用 GitOps ZTP 部署受管单节点 OpenShift 集群
- 配置中央基础架构管理，如 RHACM 文档所述
- 配置 DNS 服务集群来解析内部 API 端点 api-int.<cluster_name>.<base_domain>

流程

1. 如果您使用 example-sno.yaml SiteConfig 清单部署集群，请将新的 worker 节点添加到 spec.clusters['example-sno'].nodes 列表中：

```
nodes:
- hostName: "example-node2.example.com"
  role: "worker"
  bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embed
ded.1"
  bmcCredentialsName:
    name: "example-node2-bmh-secret"
  bootMACAddress: "AA:BB:CC:DD:EE:11"
  bootMode: "UEFI"
  nodeNetwork:
    interfaces:
      - name: eno1
        macAddress: "AA:BB:CC:DD:EE:11"
    config:
      interfaces:
        - name: eno1
          type: ethernet
          state: up
          macAddress: "AA:BB:CC:DD:EE:11"
          ipv4:
            enabled: false
          ipv6:
            enabled: true
            address:
            - ip: 1111:2222:3333:4444::1
              prefix-length: 64
      dns-resolver:
        config:
          search:
          - example.com
          server:
          - 1111:2222:3333:4444::2
      routes:
        config:
        - destination: ::/0
          next-hop-interface: eno1
          next-hop-address: 1111:2222:3333:4444::1
          table-id: 254
```

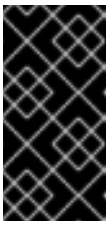
2. 为新主机创建一个 BMC 身份验证 secret，如 SiteConfig 文件的 spec.nodes 部分中的 bmcCredentialsName 字段引用：

```
apiVersion: v1
data:
  password: "password"
  username: "username"
kind: Secret
metadata:
  name: "example-node2-bmh-secret"
  namespace: example-sno
type: Opaque
```

3. 提交 Git 中的更改，然后推送到由 GitOps ZTP ArgoCD 应用程序监控的 Git 存储库。

当 ArgoCD cluster 应用程序同步时，由 GitOps ZTP 插件生成的 hub 集群中会出现两个新清单：

- BareMetalHost
- NMStateConfig



重要

不应为 worker 节点配置 cpuset 字段。worker 节点的工作负载分区会在节点安装完成后通过管理策略添加。

验证

您可以通过几种方法监控安装过程。

- 运行以下命令，检查预置备镜像是否已创建：

```
$ oc get ppimg -n example-sno
```

输出示例

```
NAMESPACE      NAME           READY   REASON
example-sno    example-sno    True    ImageCreated
example-sno    example-node2  True    ImageCreated
```

- 检查裸机主机的状态：

```
$ oc get bmh -n example-sno
```

输出示例

```
NAME          STATE        CONSUMER   ONLINE   ERROR   AGE
example-sno   provisioned             true             69m
example-node2 provisioning            true             4m50s (1)
```

1

provisioning 状态表示从安装介质引导的节点正在进行中。

- 持续监控安装过程：

a. 运行以下命令监控代理安装过程：

```
$ oc get agent -n example-sno --watch
```

输出示例

```
NAME                                   CLUSTER   APPROVED   ROLE     STAGE
671bc05d-5358-8940-ec12-d9ad22804faa   example-sno   true     master   Done
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Starting
installation
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Installing
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Writing
image to disk
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Waiting
for control plane
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Rebooting
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true     worker   Done
```

b. 当 worker 节点安装完成后，worker 节点证书会被自动批准。此时，worker 会出现在 ManagedClusterInfo 状态中。运行以下命令查看状态：

-

```
$ oc get managedclusterinfo/example-sno -n example-sno -o \
jsonpath='{range .status.nodeList[*]}{.name}{"\t"}{.conditions}{"\t"}{.labels}{"\n"}
{end}'
```

输出示例

```
example-sno [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/master":"","node-role.kubernetes.io/worker":""}
example-node2 [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/worker":""}
```

第 14 章 用于单节点 OPENSHIFT 部署的预缓存镜像

在有限带宽的环境中，您可以使用 GitOps Zero Touch Provisioning (ZTP) 解决方案来部署大量集群，您需要避免下载引导和安装 OpenShift Container Platform 所需的所有镜像。远程单节点 OpenShift 站点上的有限带宽可能会导致长时间部署时间。factory-precaching-cli 工具允许您在将服务器发送到 ZTP 置备的远程站点前预暂存服务器。

factory-precaching-cli 工具执行以下操作：

- 下载最小 ISO 所需的 RHCOS rootfs 镜像。
- 从标记为 data 的安装磁盘中创建分区。
- 将磁盘格式化为 xfs。
- 在磁盘末尾创建 GUID 分区表 (GPT) 数据分区，其中分区的大小可以被工具进行配置。
- 复制安装 OpenShift Container Platform 所需的容器镜像。
- 复制 ZTP 安装 OpenShift Container Platform 所需的容器镜像。
- 可选：将 Day-2 Operator 复制到分区。

重要

factory-precaching-cli 工具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。

14.1. 获取 FACTORY-PRECACHING-CLI 工具

factory-precaching-cli 工具 Go 二进制文件在 {rds} 工具容器镜像 中公开提供。容器镜像中的 factory-precaching-cli 工具 Go 二进制文件在使用 podman 运行 RHCOS live 镜像的服务器上执行。如果您在断开连接的环境中工作或具有私有 registry，则需要将镜像复制到服务器。

流程

- 运行以下命令拉取 factory-precaching-cli 工具镜像：

```
# podman pull quay.io/openshift-kni/telco-ran-tools:latest
```

验证

- 要检查该工具是否可用，请查询 factory-precaching-cli 工具 Go 二进制文件的当前版本：

```
# podman run quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli -v
```

输出示例

```
factory-precaching-cli version 20221018.120852+main.feecf17
```

14.2. 从实时操作系统镜像引导

您可以使用带有 的 factory-precaching-cli 工具来引导只有一个磁盘可用的服务器，外部磁盘驱动器无法附加到服务器。

警告

当磁盘即将使用 RHCOS 镜像写入时，RHCOS 要求不使用磁盘。

根据服务器硬件，您可以使用以下方法之一将 RHCOS live ISO 挂载到空白服务器上：

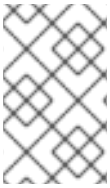
- 在 Dell 服务器上使用 Dell RACADM 工具。
- 在 HP 服务器上使用 HPONCFG 工具。
- 使用 Redfish BMC API。

注意

建议自动执行挂载过程。要自动化这个过程，您需要拉取所需的镜像并在本地 HTTP 服务器上托管它们。

先决条件

- 您打开了主机电源。
- 有到主机的网络连接。



流程

本例流程使用 Redfish BMC API 来挂载 RHCOS live ISO。

1. 挂载 RHCOS live ISO：

   a. 检查虚拟介质状态：

```
$ curl --globoff -H "Content-Type: application/json" -H \
"Accept: application/json" -k -X GET --user ${username_password} \
https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1 | python -m
json.tool
```

   b. 将 ISO 文件挂载为虚拟介质：

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\\n" -ku
${username_password} -H "Content-Type: application/json" -H "Accept:
```

```
application/json" -d '{"Image": "http://[$HTTPd_IP]/RHCOS-live.iso"}' -X POST
https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1/Actions/VirtualMe
dia.InsertMedia
```

c. 将引导顺序设置为从虚拟介质引导一次：

```
$ curl --globoff  -L -w "%{http_code} %{url_effective}\\n"  -ku
${username_password}  -H "Content-Type: application/json" -H "Accept:
application/json" -d '{"Boot":{ "BootSourceOverrideEnabled": "Once",
"BootSourceOverrideTarget": "Cd", "BootSourceOverrideMode": "UEFI"}}' -X
PATCH https://$BMC_ADDRESS/redfish/v1/Systems/Self
```

2. 重新引导并确保服务器从虚拟介质启动。

其他资源

- 有关 butane 工具的更多信息，请参阅关于 Butane。
- 有关创建自定义 live RHCOS ISO 的更多信息，请参阅为远程服务器访问创建自定义 live RHCOS ISO。
- 有关使用 Dell RACADM 工具的更多信息，请参阅集成 Dell Remote Access Controller 9 RACADM CLI 指南。
- 有关使用 HPONCFG 工具的更多信息，请参阅使用 HPONCFG。
- 有关使用 Redfish BMC API 的更多信息，请参阅使用 Redfish API 从 HTTP 托管 ISO 镜像引导。

14.3. 对磁盘进行分区

要运行完整的预缓存过程，您必须从 live ISO 启动，并使用 factory-precaching-cli 工具从容器镜像引导到分区并预缓存所有需要的工件。

需要 live ISO 或 RHCOS live ISO，因为当操作系统(RHCOS) 在置备过程中写入该设备时，磁盘不得被使用。单磁盘服务器也可使用此流程启用。

先决条件

- 您有一个没有分区的磁盘。
- 您可以访问 quay.io/openshift-kni/telco-ran-tools:latest 镜像。
- 有足够的存储来安装 OpenShift Container Platform 并预缓存所需的镜像。

流程

1. 验证磁盘是否已清除：

```
# lsblk
```

输出示例

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0     7:0    0  93.8G  0 loop /run/ephemeral
loop1     7:1    0 897.3M  1 loop /sysroot
sr0      11:0    1   999M  0 rom  /run/media/iso
nvme0n1 259:1    0   1.5T  0 disk
```

2. 从设备中删除任何文件系统、RAID 或分区表签名：

```
# wipefs -a /dev/nvme0n1
```

输出示例

```
/dev/nvme0n1: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 54
/dev/nvme0n1: 8 bytes were erased at offset 0x1749a955e00 (gpt): 45 46 49 20 50 41 52
54
/dev/nvme0n1: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
```

重要

如果磁盘不是空的，该工具会失败，因为它使用设备的分区号 1 来预缓存工件。

14.3.1. 创建分区

设备就绪后，您可以创建一个单个分区和 GPT 分区表。分区自动标记为 data，并在设备末尾创建。否则，分区将被 coreos-installer 覆盖。

重要

coreos-installer 要求在设备末尾创建分区，并将其标记为 data。在将 RHCOS 镜像写入磁盘时，这两个要求都需要保存分区。

先决条件

- 由于格式化主机设备，容器必须以特权运行。
- 您必须挂载 /dev 文件夹，以便可以在容器内执行该进程。

流程

在以下示例中，分区的大小为 250 GiB，因为允许为第 2 天 Operator 预缓存 DU 配置集。

1. 以特权运行容器，并对磁盘进行分区：

```
# podman run -v /dev:/dev --privileged \
--rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli partition \ 1
-d /dev/nvme0n1 \ 2
-s 250 3
```

1. 指定 factory-precaching-cli 工具的分区功能。

2 定义磁盘上的根目录。

3 以 GB 为单位定义磁盘大小。

2. 检查存储信息：

```
# lsblk
```

输出示例

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:3    0   250G  0 part
```

验证

您必须验证是否满足要求：

- 该设备有 GPT 分区表
- 该分区使用设备的最新扇区。
- 分区被正确标记为 data。

查询磁盘状态以验证磁盘是否按预期分区：

```
# gdisk -l /dev/nvme0n1
```

输出示例

```
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/nvme0n1: 3125627568 sectors, 1.5 TiB
Model: Dell Express Flash PM1725b 1.6TB SFF
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): CB5A9D44-9B3C-4174-A5C1-C64957910B61
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 3125627534
Partitions will be aligned on 2048-sector boundaries
Total free space is 2601338846 sectors (1.2 TiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1      2601338880      3125627534   250.0 GiB   8300  data
```

14.3.2. 挂载分区

验证磁盘是否已正确分区后，您可以将设备挂载到 /mnt 中。

重要

建议将设备挂载到 /mnt，因为在 GitOps ZTP 准备过程中使用了该挂载点。

1. 验证分区是否格式化为 xfs ：

```
# lsblk -f /dev/nvme0n1
```

输出示例

```
NAME        FSTYPE LABEL UUID                                 MOUNTPOINT
nvme0n1
└─nvme0n1p1 xfs          1bee8ea4-d6cf-4339-b690-a76594794071
```

2. 挂载分区：

```
# mount /dev/nvme0n1p1 /mnt/
```

验证

- 检查分区是否已挂载：

```
# lsblk
```

输出示例

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:2    0   250G  0 part /var/mnt (1)
```

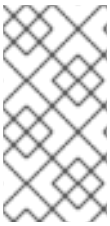
(1) 挂载点是 /var/mnt，因为 RHCOS 中的 /mnt 文件夹是到 /var/mnt 的链接。

14.4. 下载镜像

factory-precaching-cli 工具允许您将以下镜像下载到分区服务器中：

- OpenShift Container Platform 镜像

- 包含在 5G RAN 站点的分布式单元 (DU) 配置集中的 Operator 镜像
- 来自断开连接的 registry 的 Operator 镜像



注意

可用的 Operator 镜像列表在不同的 OpenShift Container Platform 版本中可能会有所不同。

14.4.1. 使用并行 worker 下载

factory-precaching-cli 工具使用并行 worker 同时下载多个镜像。您可以使用 --parallel 或 -p 选项配置 worker 数量。默认数量设置为服务器可用 CPU 的 80%。

注意

您的登录 shell 可能仅限于 CPU 的子集，这降低了容器可用的 CPU。要删除此限制，您可以在命令前面使用 taskset 0xffffffff，例如：

```
# taskset 0xffffffff podman run --rm quay.io/openshift-kni/telco-ran-tools:latest
factory-precaching-cli download --help
```

14.4.2. 准备下载 OpenShift Container Platform 镜像

要下载 OpenShift Container Platform 容器镜像，您需要知道多集群引擎版本。当使用 --du-profile 标志时，您还需要指定在要置备单节点 OpenShift 的 hub 集群中运行的 Red Hat Advanced Cluster Management (RHACM) 版本。

先决条件

- 已安装 RHACM 和多集群引擎 Operator。
- 您对存储设备进行分区。
- 您有足够的空间用于分区设备上的镜像。

- 您已将裸机服务器连接到互联网。
- 具有有效的 pull secret。

流程

1. 在 hub 集群中运行以下命令来检查 RHACM 版本和多集群引擎版本：

```
$ oc get csv -A | grep -i advanced-cluster-management
```

输出示例

```
open-cluster-management                 advanced-cluster-management.v2.6.3
Advanced Cluster Management for Kubernetes   2.6.3              advanced-cluster-
management.v2.6.3              Succeeded
```

```
$ oc get csv -A | grep -i multicluster-engine
```

输出示例

```
multicluster-engine                     cluster-group-upgrades-operator.v0.0.3
cluster-group-upgrades-operator          0.0.3
Pending
multicluster-engine                     multicluster-engine.v2.1.4             multicluster
engine for Kubernetes          2.1.4              multicluster-engine.v2.0.3
Succeeded
multicluster-engine                     openshift-gitops-operator.v1.5.7       Red Hat
OpenShift GitOps               1.5.7              openshift-gitops-operator.v1.5.6-
0.1664915551.p   Succeeded
multicluster-engine                     openshift-pipelines-operator-rh.v1.6.4      Red
Hat OpenShift Pipelines        1.6.4              openshift-pipelines-operator-rh.v1.6.3
Succeeded
```

2. 要访问容器 registry，请在服务器中复制有效的 pull secret 以供安装：

a. 创建 .docker 文件夹：

```
$ mkdir /root/.docker
```

b. 将 config.json 文件中有效的 pull 复制到之前创建的 .docker/ 文件夹：

```
$ cp config.json /root/.docker/config.json 1
```

1 /root/.docker/config.json 是默认路径，podman 检查 registry 的登录凭证。

注意

如果使用其他 registry 来拉取所需的工件，则需要复制正确的 pull secret。如果本地 registry 使用 TLS，则需要也包含来自 registry 的证书。

14.4.3. 下载 OpenShift Container Platform 镜像

factory-precaching-cli 工具允许您预缓存置备特定 OpenShift Container Platform 发行版本所需的所有容器镜像。

流程

- 运行以下命令预缓存发行版本：

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools -- \
  factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
```

1

指定 factory-precaching-cli 工具的下载功能。

2

定义 OpenShift Container Platform 发行版本。

3

定义 RHACM 版本。

4

定义多集群引擎版本。

5

定义要在磁盘上下载镜像的文件夹。

6

可选。定义存储额外镜像的存储库。这些镜像在磁盘上下载并预缓存。

输出示例

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
Processing artifact [1/176]: ocp-v4.0-art-
dev@sha256_6ac2b96bf4899c01a87366fd0feae9f57b1b61878e3b5823da0c3f34f707fbf5
Processing artifact [2/176]: ocp-v4.0-art-
dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657
c
Processing artifact [3/176]: ocp-v4.0-art-
dev@sha256_a480390e91b1c07e10091c3da2257180654f6b2a735a4ad4c3b69dbdb77bb
c06
Processing artifact [4/176]: ocp-v4.0-art-
dev@sha256_ecc5d8dbd77e326dba6594ff8c2d091eefbc4d90c963a9a85b0b2f0e6155f99
5
Processing artifact [5/176]: ocp-v4.0-art-
dev@sha256_274b6d561558a2f54db08ea96df9892315bb773fc203b1dbcea418d20f4c7a
d1
Processing artifact [6/176]: ocp-v4.0-art-
dev@sha256_e142bf5020f5ca0d1bdda0026bf97f89b72d21a97c9cc2dc71bf85050e822bb
f
...
Processing artifact [175/176]: ocp-v4.0-art-
dev@sha256_16cd7eda26f0fb0fc965a589e1e96ff8577e560fcd14f06b5fda1643036ed6c8
```

```
Processing artifact [176/176]: ocp-v4.0-art-
dev@sha256_cf4d862b4a4170d4f611b39d06c31c97658e309724f9788e155999ae51e7188
f
...
Summary:

Release:                    4.16.0
Hub Version:                  2.6.3
ACM Version:                   2.6.3
MCE Version:                   2.1.4
Include DU Profile:             No
Workers:                    83
```

验证

- 检查所有镜像是否在服务器的目标文件夹中压缩：

```
$ ls -l /mnt (1)
```

(1) 建议您预缓存 /mnt 文件夹中的镜像。

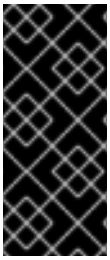
输出示例

```
-rw-r--r--. 1 root root  136352323 Oct 31 15:19 ocp-v4.0-art-
dev@sha256_edec37e7cd8b1611d0031d45e7958361c65e2005f145b471a8108f1b54316c
07.tgz
-rw-r--r--. 1 root root  156092894 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_ee51b062b9c3c9f4fe77bd5b3cc9a3b12355d040119a1434425a824f137c61a
9.tgz
-rw-r--r--. 1 root root  172297800 Oct 31 15:29 ocp-v4.0-art-
dev@sha256_ef23d9057c367a36e4a5c4877d23ee097a731e1186ed28a26c8d21501cd827
18.tgz
-rw-r--r--. 1 root root  171539614 Oct 31 15:23 ocp-v4.0-art-
dev@sha256_f0497bb63ef6834a619d4208be9da459510df697596b891c0c633da144dbb0
25.tgz
-rw-r--r--. 1 root root  160399150 Oct 31 15:20 ocp-v4.0-art-
dev@sha256_f0c339da117cde44c9aae8d0bd054bceb6f19fdb191928f6912a703182330ac
2.tgz
-rw-r--r--. 1 root root  175962005 Oct 31 15:17 ocp-v4.0-art-
dev@sha256_f19dd2e80fb41ef31d62bb8c08b339c50d193fdb10fc39cc15b353cbbfeb9b2
4.tgz
-rw-r--r--. 1 root root  174942008 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_f1dbb81fa1aa724e96dd2b296b855ff52a565fbef003d08030d63590ae6454d
```

```
f.tgz
-rw-r--r--. 1 root root  246693315 Oct 31 15:31 ocp-v4.0-art-
dev@sha256_f44dcf2c94e4fd843cbbf9b11128df2ba856cd813786e42e3da1fdfb0f6ddd01
.tgz
-rw-r--r--. 1 root root  170148293 Oct 31 15:00 ocp-v4.0-art-
dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657
c.tgz
-rw-r--r--. 1 root root  168899617 Oct 31 15:16 ocp-v4.0-art-
dev@sha256_f5099b0989120a8d08a963601214b5c5cb23417a707a8624b7eb52ab788a7f
75.tgz
-rw-r--r--. 1 root root  176592362 Oct 31 15:05 ocp-v4.0-art-
dev@sha256_f68c0e6f5e17b0b0f7ab2d4c39559ea89f900751e64b97cb42311a478338d9c
3.tgz
-rw-r--r--. 1 root root  157937478 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_f7ba33a6a9db9cfc4b0ab0f368569e19b9fa08f4c01a0d5f6a243d61ab781bd
8.tgz
-rw-r--r--. 1 root root  145535253 Oct 31 15:26 ocp-v4.0-art-
dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de
5.tgz
-rw-r--r--. 1 root root  158048761 Oct 31 15:40 ocp-v4.0-art-
dev@sha256_f914228ddbb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb4
7.tgz
-rw-r--r--. 1 root root  167914526 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_fa3ca9401c7a9efda0502240aeb8d3ae2d239d38890454f17fe5158b6230501
0.tgz
-rw-r--r--. 1 root root  164432422 Oct 31 15:24 ocp-v4.0-art-
dev@sha256_fc4783b446c70df30b3120685254b40ce13ba6a2b0bf8fb1645f116cf6a392f1
.tgz
-rw-r--r--. 1 root root  306643814 Oct 31 15:11
troubleshoot@sha256_b86b8aea29a818a9c22944fd18243fa0347c7a2bf1ad8864113ff2b
b2d8e0726.tgz
```

14.4.4. 下载 Operator 镜像

您还可以预缓存第 2 个 Operator，用于 5G Radio 访问网络 (RAN) 分布式单元 (DU) 集群配置。Day-2 Operator 依赖于已安装的 OpenShift Container Platform 版本。



重要

您需要使用 --acm-version 和 --mce-version 标志包含 RHACM hub 和 multicluster engine Operator 版本，以便 factory-precaching-cli 工具可以预缓存 RHACM 和 multicluster engine Operator 的适当容器镜像。

流程

- 预缓存 Operator 镜像：

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s 7
```

1 指定 factory-precaching-cli 工具的下载功能。

2 定义 OpenShift Container Platform 发行版本。

3 定义 RHACM 版本。

4 定义多集群引擎版本。

5 定义要在磁盘上下载镜像的文件夹。

6 可选。定义存储额外镜像的存储库。这些镜像在磁盘上下载并预缓存。

7 指定 DU 配置中包含的 Operator 预缓存。

输出示例

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
```

```
Processing artifact [1/379]: ocp-v4.0-art-
dev@sha256_7753a8d9dd5974be8c90649aadd7c914a3d8a1f1e016774c7ac7c9422e9f99
58
Processing artifact [2/379]: ose-kube-rbac-
proxy@sha256_c27a7c01e5968aff16b6bb6670423f992d1a1de1a16e7e260d12908d33224
31c
Processing artifact [3/379]: ocp-v4.0-art-
dev@sha256_370e47a14c798ca3f8707a38b28cfc28114f492bb35fe1112e55d1eb51022c9
9
...
Processing artifact [378/379]: ose-local-storage-
operator@sha256_0c81c2b79f79307305e51ce9d3837657cf9ba5866194e464b4d1b299f85
034d0
Processing artifact [379/379]: multicluster-operators-channel-
rhel8@sha256_c10f6bbb84fe36e05816e873a72188018856ad6aac6cc16271a1b3966f73ce
b3
...
Summary:

Release:                 4.16.0
Hub Version:               2.6.3
ACM Version:               2.6.3
MCE Version:               2.1.4
Include DU Profile:          Yes
Workers:                  83
```

14.4.5. 在断开连接的环境中预缓存自定义镜像

在生成 ImageSetConfiguration 自定义资源 (CR) 后，-- generate-imageset 参数会停止 factory-precaching-cli 工具。这可让您在下载任何镜像前自定义 ImageSetConfiguration CR。自定义 CR 后，您可以使用 --skip-imageset 参数下载您在 ImageSetConfiguration CR 中指定的镜像。

您可以使用以下方法自定义 ImageSetConfiguration CR：

- 添加 Operator 和其他镜像
- 删除 Operator 和其他镜像
- 将 Operator 和目录源改为本地或断开连接的 registry

流程

1. 预缓存镜像：

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s \ 7
  --generate-imageset 8
```

1 指定 factory-precaching-cli 工具的下载功能。

2 定义 OpenShift Container Platform 发行版本。

3 定义 RHACM 版本。

4 定义多集群引擎版本。

5 定义要在磁盘上下载镜像的文件夹。

6 可选。定义存储额外镜像的存储库。这些镜像在磁盘上下载并预缓存。

7 指定 DU 配置中包含的 Operator 预缓存。

8 --generate-imageset 参数只生成 ImageSetConfiguration CR，您可以对 CR 进行自定义。

输出示例

```
Generated /mnt/imageset.yaml
```

ImageSetConfiguration CR 示例

```
apiVersion: mirror.openshift.io/v1alpha2
kind: ImageSetConfiguration
mirror:
  platform:
    channels:
    - name: stable-4.16
      minVersion: 4.16.0 (1)
      maxVersion: 4.16.0
  additionalImages:
    - name: quay.io/custom/repository
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16
      packages:
        - name: advanced-cluster-management (2)
          channels:
            - name: 'release-2.6'
              minVersion: 2.6.3
              maxVersion: 2.6.3
        - name: multicluster-engine (3)
          channels:
            - name: 'stable-2.1'
              minVersion: 2.1.4
              maxVersion: 2.1.4
        - name: local-storage-operator (4)
          channels:
            - name: 'stable'
        - name: ptp-operator (5)
          channels:
            - name: 'stable'
        - name: sriov-network-operator (6)
          channels:
            - name: 'stable'
        - name: cluster-logging (7)
          channels:
            - name: 'stable'
        - name: lvms-operator (8)
          channels:
            - name: 'stable-4.16'
        - name: amq7-interconnect-operator (9)
```

```
      channels:
        - name: '1.10.x'
      - name: bare-metal-event-relay 10
        channels:
        - name: 'stable'
    - catalog: registry.redhat.io/redhat/certified-operator-index:v4.16
      packages:
      - name: sriov-fec 11
        channels:
        - name: 'stable'
```

1

平台版本与传递给工具的版本匹配。

2 3

RHACM 和多集群引擎 Operator 的版本与传递给工具的版本匹配。

4 5 6 7 8 9 10 11

CR 包含所有指定的 DU Operator。

2. 在 CR 中自定义目录资源：

```
apiVersion: mirror.openshift.io/v1alpha2
kind: ImageSetConfiguration
mirror:
  platform:
[...]
  operators:
    - catalog: eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/certified-operator-
index:v4.16
      packages:
        - name: sriov-fec
          channels:
            - name: 'stable'
```

当使用本地或断开连接的 registry 下载镜像时，您必须首先为您要从中拉取内容的 registry 添加证书。

3. 要避免任何错误，请将 registry 证书复制到您的服务器中：

```
# cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
```

4. 然后，更新证书信任存储：

```
# update-ca-trust
```

5. 将主机 /etc/pki 文件夹挂载到 factory-cli 镜像中：

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged
--rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s \ 7
  --skip-imageset 8
```

1. 指定 factory-precaching-cli 工具的下载功能。

2. 定义 OpenShift Container Platform 发行版本。

3. 定义 RHACM 版本。

4. 定义多集群引擎版本。

5. 定义要在磁盘上下载镜像的文件夹。

6. 可选。定义存储额外镜像的存储库。这些镜像在磁盘上下载并预缓存。

7.

8 通过 --skip-imageset 参数，您可以下载您在自定义 ImageSetConfiguration CR 中指定的镜像。

6. 在不生成新的 imageSetConfiguration CR 的情况下下载镜像：

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download -r 4.16.0
\
--acm-version 2.6.3 --mce-version 2.1.4 -f /mnt \
--img quay.io/custom/repository \
--du-profile -s \
--skip-imageset
```

其他资源

- 要访问在线红帽 registry，请参阅 OpenShift 安装自定义工具。
- 有关使用多集群引擎的更多信息，请参阅关于 multicluster engine operator 的集群生命周期。

14.5. GITOPS ZTP 中的预缓存镜像

SiteConfig 清单定义如何安装和配置 OpenShift 集群。在 GitOps Zero Touch Provisioning (ZTP) 置备工作流中，factory-precaching-cli 工具需要 SiteConfig 清单中的以下附加字段：

- clusters.ignitionConfigOverride
- nodes.installerArgs
- nodes.ignitionConfigOverride

带有其他字段的 SiteConfig 示例

-

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-5g-lab"
  namespace: "example-5g-lab"
spec:
  baseDomain: "example.domain.redhat.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "img4.9.10-x86-64-appsub" 1
  sshPublicKey: "ssh-rsa ..."
  clusters:
  - clusterName: "sno-worker-0"
    clusterImageSetNameRef: "eko4-img4.11.5-x86-64-appsub" 2
    clusterLabels:
      group-du-sno: ""
      common-411: true
      sites : "example-5g-lab"
      vendor: "OpenShift"
    clusterNetwork:
      - cidr: 10.128.0.0/14
        hostPrefix: 23
    machineNetwork:
      - cidr: 10.19.32.192/26
    serviceNetwork:
      - 172.30.0.0/16
    networkType: "OVNKubernetes"
    additionalNTPSources:
      - clock.corp.redhat.com
    ignitionConfigOverride:
      '{
        "ignition": {
          "version": "3.1.0"
        },
        "systemd": {
          "units": [
            {
              "name": "var-mnt.mount",
              "enabled": true,
              "contents": "[Unit]\nDescription=Mount partition with artifacts\nBefore=precache-
images.service\nBindsTo=precache-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precach
e-images.service"
            },
            {
              "name": "precache-images.service",
              "enabled": true,
              "contents": "[Unit]\nDescription=Extracts the precached images in discovery
stage\nAfter=var-
mnt.mount\nBefore=agent.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory
=/var/mnt\nExecStart=bash /usr/local/bin/extract-
ai.sh\n#TimeoutStopSec=30\n\n[Install]\nWantedBy=multi-user.target
default.target\nWantedBy=agent.service"
            }
          ]
```

```
      },
      "storage": {
        "files": [
          {
            "overwrite": true,
            "path": "/usr/local/bin/extract-ai.sh",
            "mode": 755,
            "user": {
              "name": "root"
            },
            "contents": {
              "source":
"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3
A-ai-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER
%0A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%
20total%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%2
0%27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%2
0%7C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%
20%24uri%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20exi
sting%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7
Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%
20%20%20%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20
%20%20%20%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri
%22%20%7C%20%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%
20%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20
%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20
copy%20dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29
%29%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEA
SE_LIST%7D%0A%0A%23%20workaround%20while%20https%3A%2F%2Fgithub.com%2Fope
nshift%2Fassisted-service%2Fpull%2F3546%0A%23cp%20%2Fvar%2Fmnt%2Fmodified-
rhcos-4.10.3-x86_64-metal.x86_64.raw.gz%20%2Fvar%2Ftmp%2F.%0A%0Aexit%200"
            }
          },
          {
            "overwrite": true,
            "path": "/usr/local/bin/agent-fix-bz1964591",
            "mode": 755,
            "user": {
              "name": "root"
            },
            "contents": {
              "source":
"data:,%23%21%2Fusr%2Fbin%2Fsh%0A%0A%23%20This%20script%20is%20a%20workarou
nd%20for%20bugzilla%201964591%20where%20symlinks%20inside%20%2Fvar%2Flib%2Fcon
```

```
tainers%2F%20get%0A%23%20corrupted%20under%20some%20circumstances.%0A%23%0A
%23%20In%20order%20to%20let%20agent.service%20start%20correctly%20we%20are%20che
cking%20here%20whether%20the%20requested%0A%23%20container%20image%20exists%2
0and%20in%20case%20%22podman%20images%22%20returns%20an%20error%20we%20try
%20removing%20the%20faulty%0A%23%20image.%0A%23%0A%23%20In%20such%20a%20s
cenario%20agent.service%20will%20detect%20the%20image%20is%20not%20present%20and
%20pull%20it%20again.%20In%20case%0A%23%20the%20image%20is%20present%20and%2
0can%20be%20detected%20correctly%2C%20no%20any%20action%20is%20required.%0A%0
AIMAGE%3D%24%28echo%20%241%20%7C%20sed%20%27s%2F%3A.%2A%2F%2F%27%29
%0Apodman%20image%20exists%20%24IMAGE%20%7C%7C%20echo%20%22already%20loa
ded%22%20%7C%7C%20echo%20%22need%20to%20be%20pulled%22%0A%23podman%20i
mages%20%7C%20grep%20%24IMAGE%20%7C%7C%20podman%20rmi%20--
force%20%241%20%7C%7C%20true"
          }
         }
        ]
       }
     }'
   nodes:
    - hostName: "snonode.sno-worker-0.example.domain.redhat.com"
      role: "master"
      bmcAddress: "idrac-
virtualmedia+https://10.19.28.53/redfish/v1/Systems/System.Embedded.1"
      bmcCredentialsName:
        name: "worker0-bmh-secret"
      bootMACAddress: "e4:43:4b:bd:90:46"
      bootMode: "UEFI"
      rootDeviceHints:
        deviceName: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0
      installerArgs: '["--save-partlabel", "data"]'
      ignitionConfigOverride: |
        {
         "ignition": {
          "version": "3.1.0"
         },
         "systemd": {
          "units": [
           {
            "name": "var-mnt.mount",
            "enabled": true,
            "contents": "[Unit]\nDescription=Mount partition with artifacts\nBefore=precache-
ocp-images.service\nBindsTo=precache-ocp-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precach
e-ocp-images.service"
           },
           {
            "name": "precache-ocp-images.service",
            "enabled": true,
            "contents": "[Unit]\nDescription=Extracts the precached OCP images into
containers storage\nAfter=var-mnt.mount\nBefore=machine-config-daemon-pull.service
nodeip-
configuration.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory=/var/mnt\nE
xecStart=bash /usr/local/bin/extract-
ocp.sh\nTimeoutStopSec=60\n\n[Install]\nWantedBy=multi-user.target"
           }
```

```
          ]
        },
        "storage": {
          "files": [
            {
              "overwrite": true,
              "path": "/usr/local/bin/extract-ocp.sh",
              "mode": 755,
              "user": {
                "name": "root"
              },
              "contents": {
                "source":
"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3
A-ocp-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER
%0A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%
20total%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%2
0%27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%2
0%7C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%
20%24uri%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20exi
sting%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7
Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%
20%20%20%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20
%20%20%20%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri
%22%20%7C%20%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%
20%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20
%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20
copy%20dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29
%29%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEA
SE_LIST%7D%0A%0Aexit%200"
              }
            }
          ]
        }
      }
    nodeNetwork:
      config:
        interfaces:
          - name: ens1f0
            type: ethernet
            state: up
            macAddress: "AA:BB:CC:11:22:33"
            ipv4:
              enabled: true
```

```
        dhcp: true
      ipv6:
        enabled: false
    interfaces:
     - name: "ens1f0"
       macAddress: "AA:BB:CC:11:22:33"
```

1

指定用于部署的集群镜像集，除非您在 spec.clusters.clusterImageSetNameRef 字段中指定不同的镜像集。

2

指定用于部署单个集群的集群镜像集。如果定义，它将覆盖站点级别的 spec.clusterImageSetNameRef。

14.5.1. 了解 cluster.ignitionConfigOverride 字段

clusters.ignitionConfigOverride 字段在 GitOps ZTP 发现阶段以 Ignition 格式添加配置。该配置包括挂载到虚拟介质中的 ISO 中的 systemd 服务。这样，脚本是发现 RHCOS live ISO 的一部分，可用于加载辅助安装程序(AI)镜像。

systemd 服务

systemd 服务为 var-mnt.mount 和 precache-images.services。precache-images.service 依赖于 var-mnt.mount 单元在 /var/mnt 中挂载的磁盘分区。该服务调用名为 extract-ai.sh 的脚本。

extract-ai.sh

extract-ai.sh 脚本提取并将所需的镜像从磁盘分区加载到本地容器存储。脚本成功完成后，您可以在本地使用镜像。

agent-fix-bz1964591

agent-fix-bz1964591 脚本是 AI 问题的一个临时解决方案。要防止 AI 删除镜像，这样可强制 agent.service 从 registry 中再次拉取镜像，agent-fix-bz1964591 脚本会检查请求的容器镜像是否存在。

14.5.2. 了解 nodes.installerArgs 字段

nodes.installerArgs 字段允许您配置 coreos-installer 实用程序如何将 RHCOS live ISO 写入磁盘。您需要指示保存标记为 data 的磁盘分区，因为 OpenShift Container Platform 安装过程中需要保存在

data 分区中的工件。

额外的参数直接传递给将 live RHCOS 写入磁盘的 coreos-installer 工具。下一次重启时，操作系统从磁盘启动。

您可以将几个选项传递给 coreos-installer 工具：

```
OPTIONS:
...
    -u, --image-url <URL>
            Manually specify the image URL

    -f, --image-file <path>
            Manually specify a local image file

    -i, --ignition-file <path>
            Embed an Ignition config from a file

    -I, --ignition-url <URL>
            Embed an Ignition config from a URL
...
        --save-partlabel <lx>...
            Save partitions with this label glob

        --save-partindex <id>...
            Save partitions with this number or range
...
        --insecure-ignition
            Allow Ignition URL without HTTPS or hash
```

14.5.3. 了解 nodes.ignitionConfigOverride 字段

与 clusters.ignitionConfigOverride 类似，nodes.ignitionConfigOverride 项允许对 coreos-installer 工件程序的额外配置（使用 Ignition 格式），但在 OpenShift Container Platform 的安装阶段。当 RHCOS 写入磁盘时，GitOps ZTP 发现 ISO 中包含的额外配置不再可用。在发现阶段，额外的配置存储在实时操作系统的内存中。

注意

在这个阶段，提取和加载的容器镜像数量会大于发现阶段。根据 OpenShift Container Platform 发行版本以及安装 day-2 Operator，安装时间可能会有所不同。

在安装阶段，使用 var-mnt.mount 和 precache-ocp.services systemd 服务。

precache-ocp.service

precache-ocp.service 依赖于 var-mnt.mount 单元在 /var/mnt 中挂载的磁盘分区。precache-ocp.service 服务调用一个名为 extract-ocp.sh 的脚本。

重要

要在 OpenShift Container Platform 安装前提取所有镜像，您必须先执行 precache-ocp.service，然后才能执行 machine-config-daemon-pull.service 和 nodeip-configuration.service 服务。

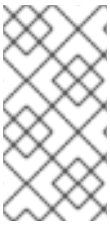
extract-ocp.sh

extract-ocp.sh 脚本提取并将所需的镜像从磁盘分区加载到本地容器存储。

当您将 SiteConfig 和可选的 PolicyGenerator 或 PolicyGenTemplate 自定义资源(CR)提交到监控 Argo CD 的 Git 仓库时，您可以通过将 CR 与 hub 集群同步来启动 GitOps ZTP 工作流。

14.6. "RENDERED CATALOG IS INVALID" 错误故障排除

当使用本地或断开连接的 registry 下载镜像时，您可能会看到 The rendered catalog is invalid 错误。这意味着您将缺少要从中拉取内容的新 registry 证书。



注意

factory-precaching-cli 工具镜像基于 UBI RHEL 镜像构建。RHCOS 上的证书路径和位置相同。

错误示例

```
Generating list of pre-cached artifacts...
error: unable to run command oc-mirror -c /mnt/imageset.yaml file:///tmp/fp-cli-3218002584/mirror --ignore-history --dry-run: Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/publish
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/v2
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/charts
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/release-signatures
backend is not configured in /mnt/imageset.yaml, using stateless mode
backend is not configured in /mnt/imageset.yaml, using stateless mode
No metadata detected, creating new workspace
```

```
level=info msg=trying next host error=failed to do request: Head
"https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-
index/manifests/v4.11": x509: certificate signed by unknown authority
host=eko4.cloud.lab.eng.bos.redhat.com:8443

The rendered catalog is invalid.

Run "oc-mirror list operators --catalog CATALOG-NAME --package PACKAGE-NAME" for
more information.

error: error rendering new refs: render reference
"eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/redhat-operator-index:v4.11": error resolving
name : failed to do request: Head
"https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-
index/manifests/v4.11": x509: certificate signed by unknown authority
```

流程

1. **将 registry 证书复制到您的服务器中：**

```
# cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
```

2. **更新证书信任存储：**

```
# update-ca-trust
```

3. **将主机 /etc/pki 文件夹挂载到 factory-cli 镜像中：**

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged
-it --rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli download -r 4.16.0 --acm-version 2.5.4 \
  --mce-version 2.0.4 -f /mnt \--img quay.io/custom/repository
  --du-profile -s --skip-imageset
```

第 15 章 单节点 OPENSHIFT 集群的基于镜像的升级

15.1. 了解单节点 OPENSHIFT 集群的基于镜像的升级

从 OpenShift Container Platform 4.14.13，生命周期代理为您提供了升级单节点 OpenShift 集群的平台版本的替代方法。基于镜像的升级比标准升级方法快，允许您直接从 OpenShift Container Platform <4.y> 升级到 <4.y+2>，并将 <4.y.z> 升级到 <4.y.z+n>。

此升级方法从目标单节点 OpenShift 集群上安装的专用 seed 集群中生成 OCI 镜像作为新的 ostree stateroot。seed 集群是一个单节点 OpenShift 集群，它部署了目标 OpenShift Container Platform 版本、第 2 天 Operator 和配置，适用于所有目标集群。

您可以使用 seed 镜像（从 seed 集群生成的镜像）升级任何单节点 OpenShift 集群上的平台版本，这些版本与 seed 集群相同的硬件、第 2 天 Operator 和集群配置组合。

重要

基于镜像的升级使用特定于集群在其上运行的硬件平台的自定义镜像。每个不同的硬件平台都需要单独的看到的镜像。

Lifecycle Agent 在参与集群中使用两个自定义资源(CR)来编配升级：

- 在 seed 集群中，SeedGenerator CR 允许 seedGenerator CR 生成。此 CR 指定要将 seed 镜像推送到的存储库。
- 在目标集群中，ImageBasedUpgrade CR 为升级目标集群和工作负载的备份配置指定 seed 镜像。

示例 SeedGenerator CR

```
apiVersion: lca.openshift.io/v1
kind: SeedGenerator
metadata:
  name: seedimage
spec:
  seedImage: <seed_image>
```

Example ImageBasedUpgrade CR

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle (1)
  seedImageRef: (2)
    version: <target_version>
    image: <seed_container_image>
    pullSecretRef:
      name: <seed_pull_secret>
  autoRollbackOnFailure: {}
#   initMonitorTimeoutSeconds: 1800 (3)
  extraManifests: (4)
  - name: example-extra-manifests
    namespace: openshift-lifecycle-agent
  oadpContent: (5)
  - name: oadp-cm-example
    namespace: openshift-adp
```

(1) 定义 ImageBasedUpgrade CR 所需的阶段。该值可以是 Idle、Prep、Upgrade 或 Rollback。

(2) 定义目标平台版本、要使用的 seed 镜像以及访问镜像所需的 secret。

(3) （可选）指定当升级在第一次重启后没有完成时回滚的时间帧（可选）。如果没有定义或设置为 0，则使用默认值 1800 秒(30 分钟)。

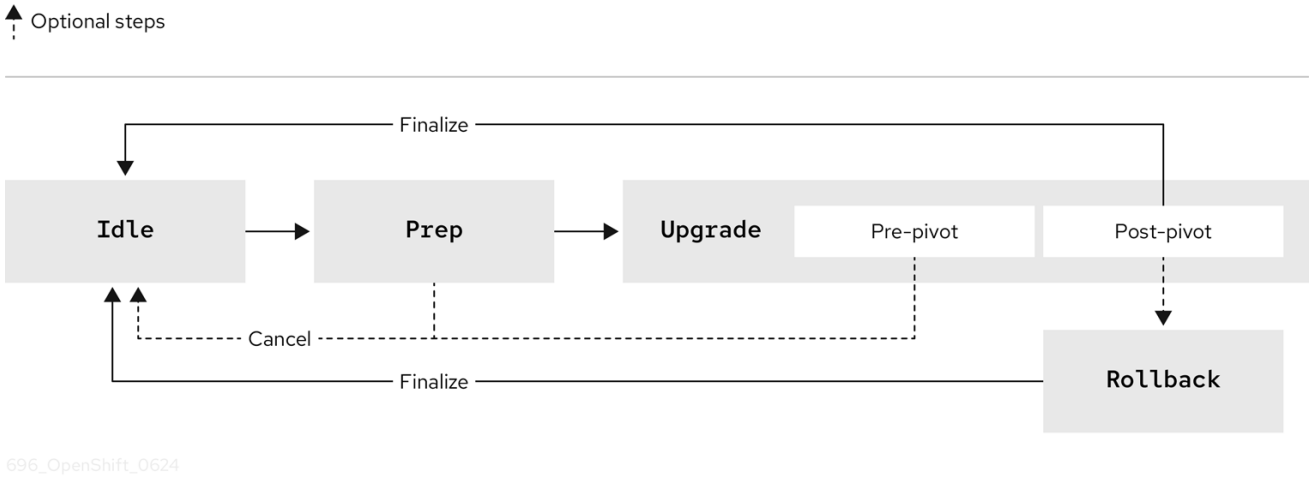
(4) （可选）指定要在升级后保留的自定义目录源的 ConfigMap 资源列表，以及要应用到不是 seed 镜像一部分的目标集群的额外清单。

(5)

15.1.1. 基于镜像的升级的阶段

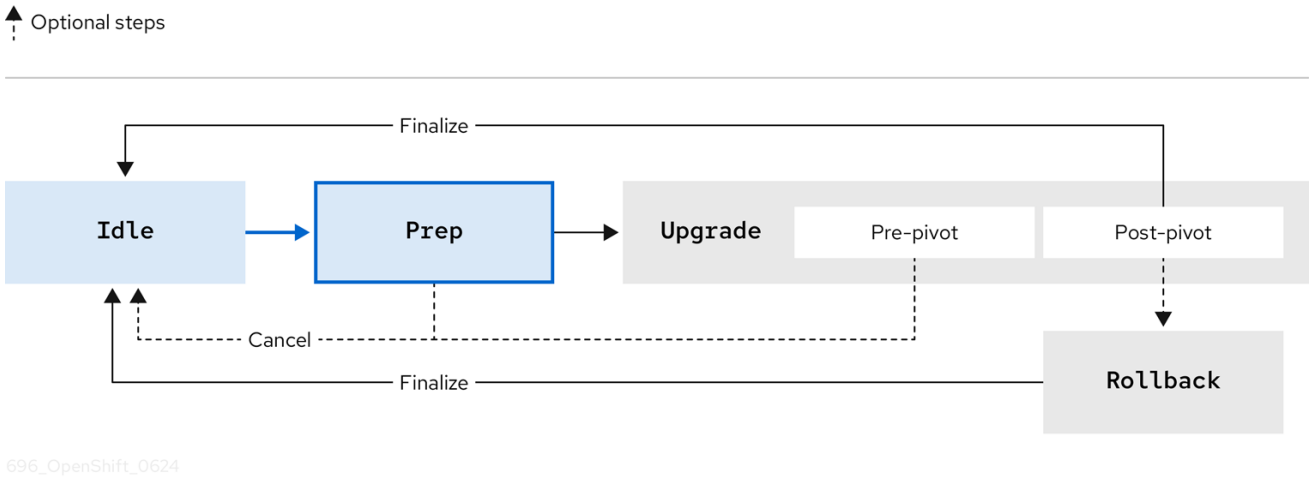
在 seed 集群中生成 seed 镜像后，您可以通过将 spec.stage 字段设置为 ImageBasedUpgrade CR 中的以下值之一来进入目标集群的阶段：

- idle
- PReP
- Upgrade（升级）
- 回滚 （可选）



15.1.1.1. idle stage

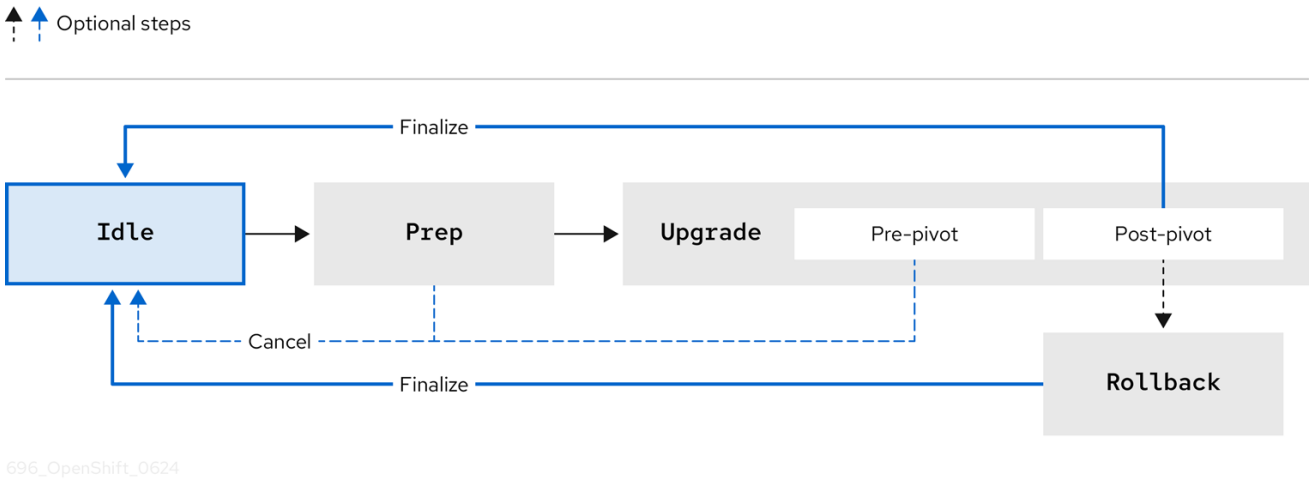
Lifecycle Agent 创建一个 ImageBasedUpgrade CR 设置为 stage: Idle （当 Operator 首次部署时）。这是默认的阶段。没有持续升级，集群已准备好移至 Prep 阶段。

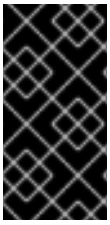


您还可以移至 Idle 阶段来执行以下步骤之一：

- 完成成功升级
- 完成回滚
- 取消持续升级，直到 Upgrade 阶段的 pre-pivot 阶段为止

移到 Idle 阶段可确保 Lifecycle Agent 清理资源，以便集群可以再次升级。





重要

如果在取消升级时使用 RHACM，则必须从目标受管集群中删除 import.open-cluster-management.io/disable-auto-import 注解，以重新启用集群的自动导入。

15.1.1.2. Prep 阶段

注意

您可以在调度的维护窗口前完成此阶段。

对于 Prep 阶段，您可以在 ImageBasedUpgrade CR 中指定以下升级详情：

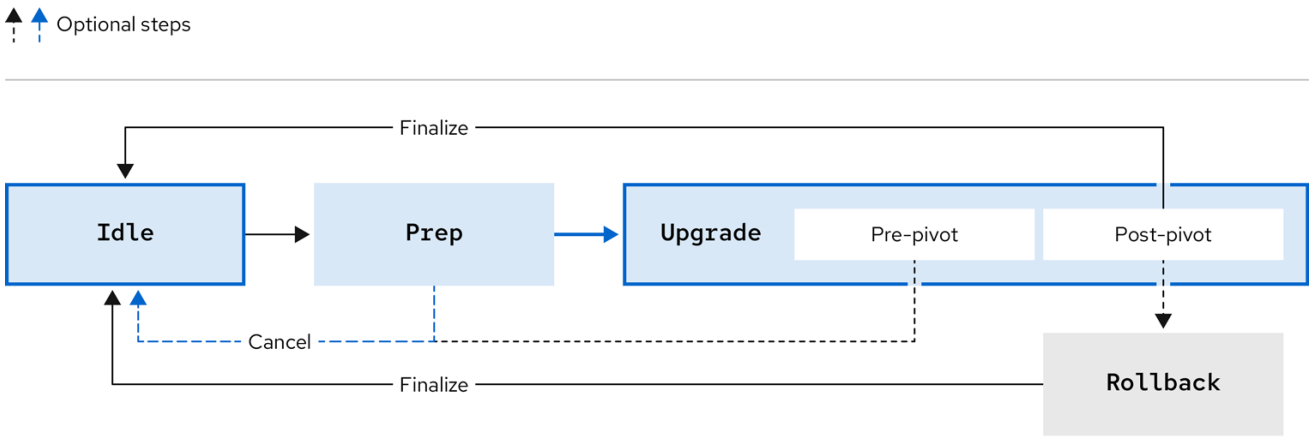
- 要使用的 seed 镜像
- 要备份的资源
- 升级后要应用和自定义目录源的额外清单（若有）

然后，根据您指定的内容，生命周期代理准备升级，而不影响当前运行的版本。在这个阶段，生命周期代理可确保目标集群已准备好进入 Upgrade 阶段，方法是检查它是否满足特定条件，并使用 seed 镜像中指定的其他容器镜像将 seed 镜像推送到目标集群。

您还可以使用 OADP Operator 的 Backup 和 Restore CR 准备备份资源。这些 CR 在 Upgrade 阶段用于重新配置集群，使用 RHACM 注册集群，并恢复应用程序工件。

除了 OADP Operator 外，生命周期代理使用 ostree 版本系统创建一个备份，它允许在升级和回滚后完成集群重新配置。

在 Prep 阶段完成后，您可以通过移到 Idle 阶段来取消升级过程，或者您可以通过移到 ImageBasedUpgrade CR 中的 Upgrade 阶段来启动升级。如果取消升级，Operator 会执行清理操作。



15.1.1.3. 升级阶段

Upgrade 阶段由两个阶段组成：

pre-pivot

在点新 stateroot 之前，生命周期代理会收集所需的特定于集群的工件，并将其存储在新的 stateroot 中。在 Prep 阶段指定的集群资源备份会在兼容对象存储解决方案中创建。Lifecycle Agent 导出在 ImageBasedUpgrade CR 中的 extraManifests 字段中指定的 CR，或绑定到目标集群的 ZTP 策略中描述的 CR。在 pre-pivot 阶段完成后，生命周期代理会将新的 stateroot 部署设置为默认的引导条目并重启节点。

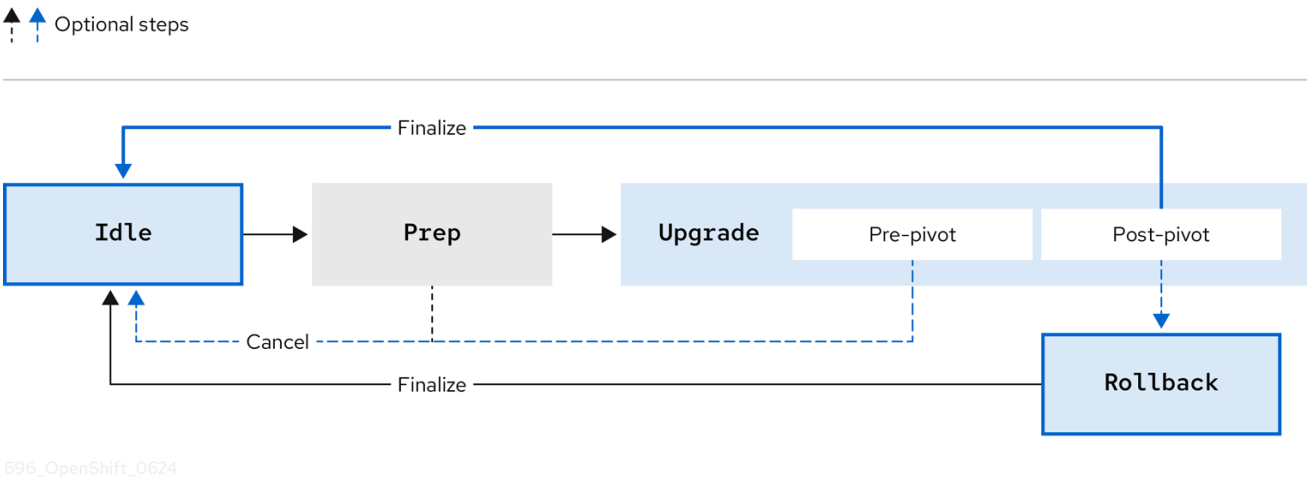
post-pivot

从新 stateroot 引导后，生命周期代理通过应用在 pre-pivot 阶段收集的特定于集群的工件来重新配置集群。Operator 应用所有保存的 CR，并恢复备份。Operator 还重新生成 seed 镜像的集群加密。这样可确保每个单节点 OpenShift 集群使用相同的 seed 镜像升级都有唯一的且有效的加密对象。

升级完成后，您可以对更改满意，您可以通过移至 Idle 阶段来完成升级。

重要

完成升级后，您无法回滚到原始版本。



如果要取消升级，可以直到 Upgrade 阶段的 pre-pivot 阶段为止。如果在升级后遇到问题，您可以移到手动回滚的 Rollback 阶段。

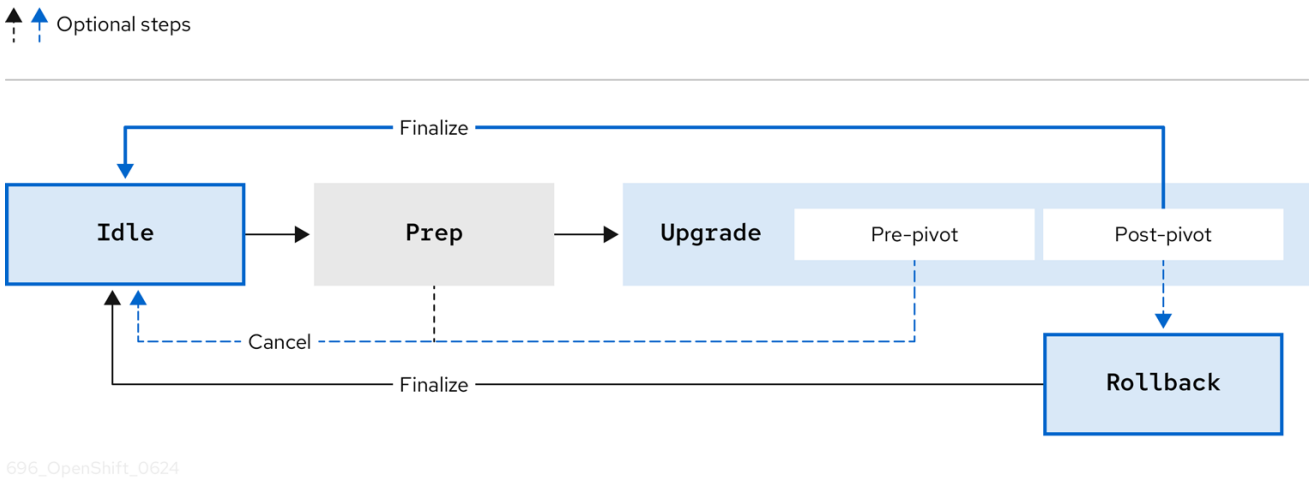
15.1.1.4. （可选） Rollback stage

Rollback 阶段可以手动启动，或者在失败时自动启动。在 Rollback 阶段，生命周期代理会将原始 ostree stateroot 部署设置为默认值。然后，节点会使用以前的 OpenShift Container Platform 和应用程序配置重启。

警告

如果在回滚后移至 Idle 阶段，则 Lifecycle Agent 会清理可用于对失败的升级进行故障排除的资源。

如果升级没有在指定时间限制内完成，则 Lifecycle Agent 将启动自动回滚。如需有关自动回滚的更多信息，请参阅相关的 *（可选）初始使用 Lifecycle Agent 的回滚* 部分。



其他资源

- 使用生命周期代理执行基于镜像的升级
- 使用 Lifecycle Agent 和 GitOps ZTP 执行基于镜像的升级
- （可选）使用 Lifecycle Agent 移到基于镜像的升级的 Rollback 阶段
- （可选）使用 Lifecycle Agent 和 GitOps ZTP 移到基于镜像升级的 Rollback 阶段

15.1.2. 基于镜像的升级指南

要成功进行基于镜像的升级，您的部署必须满足某些要求。

您可以在执行基于镜像的升级时有不同的部署方法：

GitOps ZTP

您可以使用 GitOps Zero Touch Provisioning (ZTP)来部署和配置集群。

Non-GitOps

您只使用 Red Hat Advanced Cluster Management (RHACM)来部署和配置集群。

您可以在断开连接的环境中执行基于镜像的升级。有关如何为断开连接的环境镜像镜像的更多信息，

请参阅"为断开连接的安装镜像镜像"。

其他资源

- 为断开连接的安装 mirror 镜像

15.1.2.1. 组件的最低软件版本

根据您的部署方法，基于镜像的升级需要以下最低软件版本。

表 15.1. 组件的最低软件版本

组件	软件版本	必填
生命周期代理	4.16	是
OADP Operator	1.3.1	是
受管集群版本	4.14.13	是
hub 集群版本	4.16	是，如果使用 RHACM
RHACM	2.10.2	是，如果使用 RHACM
GitOps ZTP 插件	4.16	只适用于 GitOps ZTP 部署方法
Red Hat OpenShift GitOps	1.12	只适用于 GitOps ZTP 部署方法
Topology Aware Lifecycle Manager (TALM)	4.16	只适用于 GitOps ZTP 部署方法
Local Storage Operator [1]	4.14	是
逻辑卷管理器(LVM)存储 [1]	4.14.2	是

1. 持久性存储必须由 LVM Storage 或 Local Storage Operator 提供，不能由这两个 Operator 提供。

15.1.2.2. hub 集群指南

如果使用 Red Hat Advanced Cluster Management (RHACM)，您的 hub 集群需要满足以下条件：

- 为了避免在 seed 镜像中包含任何 RHACM 资源，您需要在生成 seed 镜像前禁用所有可选 RHACM 附加组件。

- 在执行基于镜像的升级前，您的 hub 集群必须至少升级到目标版本，然后才能执行目标单节点 OpenShift 集群。

15.1.2.3. seed 镜像指南

seed 镜像以一组具有类似配置的单节点 OpenShift 集群为目标。这意味着 seed 集群需要与以下项目的目标集群具有相同的配置：

- 性能配置集

- 目标集群的 MachineConfig 资源

- IP 版本

  注意

  这个版本不支持双栈网络。

- 第 2 天 Operator 集，包括 Lifecycle Agent 和 OADP Operator

- 断开连接的 registry

- FIPS 配置

- 如果目标集群有多个 IP，且其中一个属于用于创建 seed 镜像的子网，如果目标集群的节点 IP 不属于该子网，则升级会失败。

以下配置只需要在参与的集群中部分匹配：

- 如果目标集群有代理配置，seed 集群必须具有代理配置，但配置不必相同。

- 所有参与的集群都需要在主磁盘上的一个专用分区。但是，分区的大小和起始不必相同。只有 MachineConfig CR 中的 spec.config.storage.disks.partitions.label: varlibcontainers 标签必须在 seed 和目标集群上匹配。有关如何创建磁盘分区的更多信息，请参阅"使用 GitOps ZTP 时在 ostree stateroots 之间配置共享目录目录，或"在使用 GitOps ZTP 时在 ostree stateroot 之间配置共享容器目录。

有关 seed 镜像中包含的内容的更多信息，请参阅 "Seed image configuration" 和 "Seed image configuration using the RAN DU profile"。

其他资源

- 在 ostree stateroot 之间配置共享容器目录

- 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录

- seed 镜像配置

15.1.2.4. OADP 备份和恢复指南

使用 OADP Operator，您可以使用 ConfigMap 对象中的 Backup 和 Restore CR 来备份和恢复目标集群上的应用程序。应用程序必须在当前和目标 OpenShift Container Platform 版本中工作，以便在升级后恢复它们。备份必须包含初始创建的资源。

备份中必须排除以下资源：

- pods

- 端点

- controllerrevision

- PodMetrics
- PackageManifest
- replicaSet
- LocalVolume，如果使用 Local Storage Operator (LSO)

单节点 OpenShift 有两个本地存储实现：

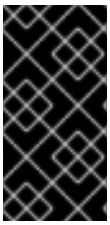
Local Storage Operator (LSO)

Lifecycle Agent 备份和恢复所需的工件，包括 LocalVolumes 及其关联的 StorageClasses。您必须在应用程序 Backup CR 中排除 persistentVolumes 资源。

LVM 存储

您必须为 LVM Storage 工件创建 Backup 和 Restore CR。您必须在应用程序 Backup CR 中包含 persistentVolumes 资源。

对于基于镜像的升级，给定目标集群只支持一个 Operator。



重要

对于这两个 Operator，您不能通过 ImageBasedUpgrade CR 将 Operator CR 应用为额外清单。

持久性卷内容会被保留并在 pivot 后使用。当您配置 DataProtectionApplication CR 时，您必须确保基于镜像的升级将 .spec.configuration.restic.enable 设置为 false。这禁用了容器存储接口集成。

15.1.2.4.1. lca.openshift.io/apply-wave guidelines

lca.openshift.io/apply-wave 注解决定 Backup 或 Restore CR 的应用顺序。注解的值必须是字符串号。如果您在 Backup 或 Restore CR 中定义 lca.openshift.io/apply-wave 注解，它们会根据注解值以递增顺序应用。如果没有定义注解，则会一起应用它们。

lca.openshift.io/apply-wave 注解必须在平台 Restore CR 中数字较低，如 RHACM 和 LVM Storage 工件，而不是应用程序工件。这样，在应用程序前会恢复平台工件。

如果应用程序包含集群范围的资源，则必须创建单独的 Backup 和 Restore CR，将备份范围限制为应用程序创建的特定集群范围资源。在剩余的应用程序 Restore CR 前，必须恢复集群范围的资源的 Restore CR。

15.1.2.4.2. lca.openshift.io/apply-label guidelines

您可以使用 lca.openshift.io/apply-label 注解来专门备份特定资源。根据您在注解中定义的资源，生命周期代理应用 lca.openshift.io/backup: <backup_name& gt; 标签，并在创建 Backup CR 时将 labelSelector.matchLabels.lca.openshift.io/backup: <backup_name > 标签选择器应用到指定的资源。

要使用 lca.openshift.io/apply-label 注解来备份特定资源，注解中列出的资源也必须包含在 spec 部分。如果在 Backup CR 中使用 lca.openshift.io/apply-label 注解，则只备份注解中列出的资源，即使 spec 部分中指定了其他资源类型。

CR 示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
  annotations:
    lca.openshift.io/apply-label:
rbac.authorization.k8s.io/v1/clusterroles/klusterlet,apps/v1/deployments/open-cluster-
management-agent/klusterlet (1)
  labels:
    velero.io/storage-location: default
spec:
  includedNamespaces:
   - open-cluster-management-agent
  includedClusterScopedResources:
   - clusterroles
  includedNamespaceScopedResources:
   - deployments
```

(1)

该值必须是以 group/version/resource/name 格式为集群范围的资源或 group/version/resource/namespace/name 格式的逗号分隔列表，且必须附加到相关的 Backup CR。

15.1.2.5. 额外清单指南

使用新的默认 stateroot 部署重启后，在恢复应用程序工件前，生命周期代理使用额外的清单来恢复您的目标集群。

不同的部署方法需要不同的方法来应用额外清单：

GitOps ZTP

您可以使用 lca.openshift.io/target-ocp-version: <target_ocp_version > 标签标记 Lifecycle Agent 必须提取并应用的额外清单。您可以使用 ImageBasedUpgrade CR 中的 lca.openshift.io/target-ocp-version 指定标记为 lca.openshift.io/target-ocp-version-manifest-count 注解的清单数量。如果指定，生命周期代理会验证从策略中提取的清单数量是否与 prep 和 upgrade 阶段注解中提供的数字匹配。

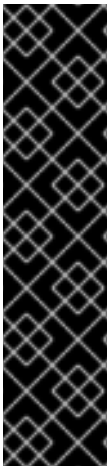
lca.openshift.io/target-ocp-version-manifest-count 注解示例

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  annotations:
    lca.openshift.io/target-ocp-version-manifest-count: "5"
  name: upgrade
```

non-Gitops

您可以使用 lca.openshift.io/apply-wave 注解标记额外的清单，以确定应用顺序。标记的额外清单嵌套在 ConfigMap 对象中，并在 Lifecycle Agent 在 pivot 后使用 的镜像Upgrade CR 中引用。

如果目标集群使用自定义目录源，您必须将它们作为指向正确发行版本的额外清单包含。



重要

您不能将以下项目作为额外清单应用：

- MachineConfig 对象
- OLM Operator 订阅

其他资源

- 使用生命周期代理执行基于镜像的升级
- 使用 Lifecycle Agent 和 GitOps ZTP 执行基于镜像的升级
- 为 ZTP 准备 hub 集群
- 使用 Lifecycle Agent 为基于镜像的升级创建 ConfigMap 对象
- 使用 GitOps ZTP 为基于镜像的升级创建 ConfigMap 对象
- 关于安装 OADP

15.2. 为单节点 OPENSHIFT 集群准备基于镜像的升级

15.2.1. 为基于镜像的升级配置共享目录

您的单节点 OpenShift 集群需要具有基于镜像的升级的共享 var/lib/containers 分区。您可在安装时执行此操作。

15.2.1.1. 在 ostree stateroot 之间配置共享容器目录

在安装过程中将 MachineConfig 应用到 seed 和目标集群，以创建独立分区，并在升级过程中要使用的两个 ostree stateroot 间共享 /var/lib/containers 目录。

重要

您必须在安装时完成这个步骤。

流程

- 应用 MachineConfig 创建独立分区：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-containers-partitioned
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      disks:
        - device: /dev/disk/by-id/wwn-<root_disk> 1
          partitions:
            - label: varlibcontainers
              startMiB: <start_of_partition> 2
              sizeMiB: <partition_size> 3
      filesystems:
        - device: /dev/disk/by-partlabel/varlibcontainers
          format: xfs
          mountOptions:
            - defaults
            - prjquota
          path: /var/lib/containers
          wipeFilesystem: true
    systemd:
      units:
        - contents: |-
            # Generated by Butane
            [Unit]
            Before=local-fs.target
            Requires=systemd-fsck@dev-disk-by\x2dpartlabel-varlibcontainers.service
            After=systemd-fsck@dev-disk-by\x2dpartlabel-varlibcontainers.service

            [Mount]
            Where=/var/lib/containers
            What=/dev/disk/by-partlabel/varlibcontainers
            Type=xfs
            Options=defaults,prjquota

            [Install]
```

```
    RequiredBy=local-fs.target
  enabled: true
  name: var-lib-containers.mount
```

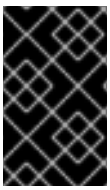
1 指定根磁盘。

2 以 MiB 为单位指定分区的起始位置。如果该值太小，则安装将失败。

3 为 500 GB 分区指定最小大小，以确保为 precached 镜像有足够的磁盘空间。如果值太小，则安装后部署将失败。

15.2.1.2. 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录

当您使用 GitOps Zero Touch Provisioning (ZTP)工作流时，您可以执行以下步骤在 seed 和目标集群上创建单独的磁盘分区，并共享 /var/lib/containers 目录。



重要

您必须在安装时完成这个步骤。

先决条件

- 安装 Butane。

流程

1. 创建 storage.bu 文件：

```
variant: fcos
version: 1.3.0
storage:
  disks:
  - device: /dev/disk/by-id/wwn-<root_disk> 1
    wipe_table: false
    partitions:
    - label: var-lib-containers
      start_mib: <start_of_partition> 2
```

```
    size_mib: <partition_size> 3
filesystems:
  - path: /var/lib/containers
    device: /dev/disk/by-partlabel/var-lib-containers
    format: xfs
    wipe_filesystem: true
    with_mount_unit: true
    mount_options:
      - defaults
      - prjquota
```

1 指定根磁盘。

2 以 MiB 为单位指定分区的起始位置。如果该值太小，则安装将失败。

3 为 500 GB 分区指定最小大小，以确保为 precached 镜像有足够的磁盘空间。如果值太小，则安装后部署将失败。

2. 将 storage.bu 转换为 Ignition 文件：

```
$ butane storage.bu
```

输出示例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-partlabel/var-lib-containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. 将输出复制到 SiteConfig CR 中的 .spec.clusters.nodes.ignitionConfigOverride 字段中：

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: '{"ignition":{"version":"3.2.0"},"storage":{"disks":
[{"device":"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62","partitions":
[{"label":"var-lib-
containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":
[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":
["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd"
:{"units":[{"contents":"# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-
disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-
disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}'
[...]
```

验证

1. 在安装过程中，在 hub 集群上验证 BareMetalHost 对象显示注解：

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-
install.openshift.io/ignition-config-overrides"]
```

输出示例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-
id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-
containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":
[{\"device\":\"/dev/disk/by-partlabel/var-lib-
containers\",\"format\":\"xfs\",\"mountOptions\":
[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"sys
temd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-
fsck@dev-disk-by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-
by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/
dev/disk/by-partlabel/var-lib-
containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-
fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. **安装后，检查单节点 OpenShift 磁盘状态：**

```
# lsblk
```

输出示例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                      /sysroot/ostree/deploy/rhcos/var
│                      /usr
│                      /etc
│                      /
│                      /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

```
# df -h
```

输出示例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

15.2.2. 为基于镜像的升级安装 Operator

通过安装 Lifecycle Agent 和 OADP Operator 为升级准备集群。

要使用非 GitOps 方法安装 OADP Operator，请参阅"安装 OADP Operator"。

其他资源

- 安装 OADP Operator
- 关于备份和恢复位置及其 secret
- 创建备份 CR
- 创建恢复 CR

15.2.2.1. 使用 CLI 安装生命周期代理

您可以使用 OpenShift CLI (oc)安装生命周期代理。

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. 为 Lifecycle Agent 创建一个 Namespace 对象 YAML 文件，如 lcao-namespace.yaml ：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-lifecycle-agent
  annotations:
    workload.openshift.io/allowed: management
```

a. 运行以下命令来创建 Namespace CR：

```
$ oc create -f lcao-namespace.yaml
```

2. 为 Lifecycle Agent 创建 OperatorGroup 对象 YAML 文件，如 lcao-operatorgroup.yaml：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-lifecycle-agent
  namespace: openshift-lifecycle-agent
spec:
  targetNamespaces:
  - openshift-lifecycle-agent
```

a. 运行以下命令来创建 OperatorGroup CR:

```
$ oc create -f lcao-operatorgroup.yaml
```

3. 创建一个 Subscription CR，如 lcao-subscription.yaml ：

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: openshift-lifecycle-agent-subscription
  namespace: openshift-lifecycle-agent
spec:
  channel: "stable"
  name: lifecycle-agent
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

a. 运行以下命令来创建 Subscription CR：

```
$ oc create -f lcao-subscription.yaml
```

验证

1. 要验证安装是否成功，请运行以下命令来检查 CSV 资源：

```
$ oc get csv -n openshift-lifecycle-agent
```

输出示例

```
NAME                         DISPLAY                VERSION           REPLACES
PHASE
lifecycle-agent.v4.16.0      Openshift Lifecycle Agent   4.16.0              Succeeded
```

2. 运行以下命令验证 Lifecycle Agent 是否正在运行：

```
$ oc get deploy -n openshift-lifecycle-agent
```

输出示例

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
lifecycle-agent-controller-manager   1/1     1            1           14s
```

15.2.2.2. 使用 Web 控制台安装 Lifecycle Agent

您可以使用 OpenShift Container Platform Web 控制台安装 Lifecycle Agent。

先决条件

- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 从可用的 Operator 列表中选择 Lifecycle Agent，然后单击 Install。
3. 在 Install Operator 页面中，在 A specific namespace on the cluster 下选择 openshift-

lifecycle-agent。

4. 点 Install。

验证

1. 确认安装成功：

a. 点 Operators → Installed Operators。

b. 确保 openshift-lifecycle-agent 项目中列出的 Lifecycle Agent 的 Status 为 InstallSucceeded。

注意

在安装过程中，Operator 可能会显示 Failed 状态。如果安装过程结束后有 InstallSucceeded 信息，您可以忽略这个 Failed 信息。

如果 Operator 没有成功安装：

1. 点 Operators → Installed Operators，检查 Operator Subscriptions 和 Install Plans 选项卡中的 Status 是否有任何错误。

2. 点 Workloads → Pods，在 openshift-lifecycle-agent 项目中检查 pod 的日志。

15.2.2.3. 使用 GitOps ZTP 安装生命周期代理

使用 GitOps Zero Touch Provisioning (ZTP)安装生命周期代理，以执行基于镜像的升级。

先决条件

- 在 source-crs 目录中创建一个名为 custom-crs 的目录。source-crs 目录必须与 kustomization.yaml 文件位于同一个位置。

流程

1. 在 openshift-lifecycle-agent 命名空间中创建以下 CR，并将它们推送到 source-crs/custom-crs 目录：

LcaSubscriptionNS.yaml 文件示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-lifecycle-agent
  annotations:
    workload.openshift.io/allowed: management
    ran.openshift.io/ztp-deploy-wave: "2"
  labels:
    kubernetes.io/metadata.name: openshift-lifecycle-agent
```

LcaSubscriptionOperGroup.yaml 文件示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: lifecycle-agent-operatorgroup
  namespace: openshift-lifecycle-agent
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  targetNamespaces:
    - openshift-lifecycle-agent
```

LcaSubscription.yaml 文件示例

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: lifecycle-agent
  namespace: openshift-lifecycle-agent
```

```
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: "stable"
  name: lifecycle-agent
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
  status:
    state: AtLatestKnown
```

目录结构示例

```
├── kustomization.yaml
├── sno
│   ├── example-cnf.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   └── ns.yaml
├── source-crs
│   ├── custom-crs
│   │   ├── LcaSubscriptionNS.yaml
│   │   ├── LcaSubscriptionOperGroup.yaml
│   │   ├── LcaSubscription.yaml
```

2. 将 CR 添加到通用 PolicyGenTemplate 中：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true"
    du-profile: "latest"
  sourceFiles:
    - fileName: custom-crs/LcaSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/LcaSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
```

```
    - fileName: custom-crs/LcaSubscription.yaml
      policyName: "subscriptions-policy"
[...]
```

15.2.2.4. 使用 GitOps ZTP 安装和配置 OADP Operator

在开始升级前，使用 GitOps ZTP 安装和配置 OADP Operator。

先决条件

- 在 source-crs 目录中创建一个名为 custom-crs 的目录。source-crs 目录必须与 kustomization.yaml 文件位于同一个位置。

流程

1. 在 openshift-adp 命名空间中创建以下 CR，并将其推送到 source-crs/custom-crs 目录：

OadpSubscriptionNS.yaml 文件示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  labels:
    kubernetes.io/metadata.name: openshift-adp
```

OadpSubscriptionOperGroup.yaml 文件示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: redhat-oadp-operator
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
```

```
spec:
  targetNamespaces:
  - openshift-adp
```

OadpSubscription.yaml 文件示例

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: redhat-oadp-operator
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: stable-1.3
  name: redhat-oadp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

OadpOperatorStatus.yaml 文件示例

```
apiVersion: operators.coreos.com/v1
kind: Operator
metadata:
  name: redhat-oadp-operator.openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
status:
  components:
    refs:
    - kind: Subscription
      namespace: openshift-adp
      conditions:
      - type: CatalogSourcesUnhealthy
        status: "False"
    - kind: InstallPlan
      namespace: openshift-adp
      conditions:
      - type: Installed
```

```
      status: "True"
    - kind: ClusterServiceVersion
      namespace: openshift-adp
      conditions:
      - type: Succeeded
        status: "True"
        reason: InstallSucceeded
```

目录结构示例

```
├── kustomization.yaml
├── sno
│   ├── example-cnf.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   └── ns.yaml
├── source-crs
│   ├── custom-crs
│   │   ├── OadpSubscriptionNS.yaml
│   │   ├── OadpSubscriptionOperGroup.yaml
│   │   ├── OadpSubscription.yaml
│   │   ├── OadpOperatorStatus.yaml
```

2. 将 CR 添加到通用 PolicyGenTemplate 中：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true"
    du-profile: "latest"
  sourceFiles:
    - fileName: custom-crs/OadpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/OadpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/OadpSubscription.yaml
      policyName: "subscriptions-policy"
```

```
    - fileName: custom-crs/OadpOperatorStatus.yaml
      policyName: "subscriptions-policy"
[...]
```

3. 创建 DataProtectionApplication CR 和 S3 secret：

   a. 在 source-crs/custom-crs 目录中创建以下 CR：

DataProtectionApplication.yaml 文件示例

```
apiVersion: oadp.openshift.io/v1
kind: DataProtectionApplication
metadata:
  name: dataprotectionapplication
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
spec:
  configuration:
    restic:
      enable: false (1)
    velero:
      defaultPlugins:
        - aws
        - openshift
      resourceTimeout: 10m
  backupLocations:
    - velero:
        config:
          profile: "default"
          region: minio
          s3Url: $url
          insecureSkipTLSVerify: "true"
          s3ForcePathStyle: "true"
        provider: aws
        default: true
        credential:
          key: cloud
          name: cloud-credentials
        objectStorage:
          bucket: $bucketName (2)
          prefix: $prefixName (3)
status:
  conditions:
  - reason: Complete
    status: "True"
    type: Reconciled
```

1

对于基于镜像的升级，spec.configuration.restic.enable 字段必须设置为 false，因为升级后持久性卷内容会被保留并重复使用。

2 3

bucket 定义在 S3 后端中创建的存储桶名称。前缀定义要在存储桶中自动创建的子目录的名称。bucket 和前缀的组合对于每个目标集群都必须是唯一的，以避免它们间的干扰。要确保每个目标集群的唯一存储目录，您可以使用 RHACM hub 模板功能，例如 prefix: {{hub .ManagedClusterName hub}}。

OadpSecret.yaml 文件示例

```
apiVersion: v1
kind: Secret
metadata:
  name: cloud-credentials
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
type: Opaque
```

OadpBackupStorageLocationStatus.yaml 文件示例

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
status:
  phase: Available
```

OadpBackupStorageLocationStatus.yaml CR 验证 OADP 创建的备份存储位置的可用性。

b. 使用覆盖将 CR 添加到站点 PolicyGenTemplate 中：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-cnf"
  namespace: "ztp-site"
spec:
  bindingRules:
    sites: "example-cnf"
    du-profile: "latest"
  mcp: "master"
  sourceFiles:
    ...
  - fileName: custom-crs/OadpSecret.yaml
    policyName: "config-policy"
    data:
      cloud: <your_credentials> 1
  - fileName: custom-crs/DataProtectionApplication.yaml
    policyName: "config-policy"
    spec:
      backupLocations:
        - velero:
            config:
              region: minio
              s3Url: <your_S3_URL> 2
              profile: "default"
              insecureSkipTLSVerify: "true"
              s3ForcePathStyle: "true"
            provider: aws
            default: true
            credential:
              key: cloud
              name: cloud-credentials
            objectStorage:
              bucket: <your_bucket_name> 3
              prefix: <cluster_name> 4
  - fileName: custom-crs/OadpBackupStorageLocationStatus.yaml
    policyName: "config-policy"
```

1 为您的 S3 存储后端指定您的凭证。

2 指定 S3 兼容存储桶的 URL。

3 4

bucket 定义在 S3 后端中创建的存储桶名称。前缀 定义存储桶中自动创建的子目录的名称。bucket 和 前缀 的组合对于每个目标集群都必须是唯一的，以避免它们间的干扰。要确保每个目标集群的唯一存储目录，您可以使用 RHACM hub 模板功能，例如 prefix: {{hub .ManagedClusterName hub}}。

15.2.3. 使用生命周期代理为基于镜像的升级生成 seed 镜像

使用 Lifecycle Agent 生成带有 SeedGenerator 自定义资源(CR)的 seed 镜像。

15.2.3.1. seed 镜像配置

seed 镜像以一组具有类似配置的单节点 OpenShift 集群为目标。这意味着 seed 镜像必须具有与目标集群的 seed 集群共享的所有组件和配置。因此，从 seed 集群生成的 seed 镜像不能包含任何特定于集群的配置。

下表列出了您必须且不得包含在 seed 镜像中的组件、资源和配置：

表 15.2. seed 镜像配置

集群配置	在 seed 镜像中包括
性能配置集	是
目标集群的 **MachineConfig** 资源	是
IP 版本 [1]	是
第 2 天 Operator 集，包括 Lifecycle Agent 和 OADP Operator	是
断开连接的 registry 配置	是
有效的代理配置 [2]	是
FIPS 配置	是
与目标集群大小匹配的容器存储的主磁盘上的专用分区	是

集群配置	在 seed 镜像中包括
本地卷 • LSO 的 **LocalVolume** 中使用的 **StorageClass** • LSO 的 **LocalVolume** • LVMS 的 **LVMCluster** CR	否
OADP **DataProtectionApplication** CR	否

1. **这个版本不支持双栈网络。**

2. **代理配置不必相同。**

15.2.3.1.1. 使用 RAN DU 配置集的 seed 镜像配置

下表列出了在使用 RAN DU 配置集时必须且不得包含在 seed 镜像中的组件、资源和配置：

表 15.3. 使用 RAN DU 配置集的 seed 镜像配置

资源	在 seed 镜像中包括
作为第 0 天安装的一部分应用的所有额外清单	是
所有第 2 天 Operator 订阅	是
ClusterLogging.yaml	是
DisableOLMPprof.yaml	是
TunedPerformancePatch.yaml	是
PerformanceProfile.yaml	是
SriovOperatorConfig.yaml	是
DisableSnoNetworkDiag.yaml	是
StorageClass.yaml	No，如果在 **StorageLV.yaml**中使用它
StorageLV.yaml	否

资源	在 seed 镜像中包括
StorageLVMCluster.yaml	否

表 15.4. 使用 RAN DU 配置集的 seed 镜像配置用于额外的清单

资源	应用作为额外清单
ClusterLogForwarder.yaml	是
ReduceMonitoringFootprint.yaml	是
SriovFecClusterConfig.yaml	是
PtpOperatorConfigForEvent.yaml	是
DefaultCatsrc.yaml	是
PtpConfig.yaml	如果目标集群的接口与 seed 集群是通用的，您可以在 seed 镜像中包含它们。否则，将它作为额外清单应用。
SriovNetwork.yamlSriovNetworkNodePolicy.yaml	如果 seed 和目标集群上的配置（包括命名空间）完全相同，您可以在 seed 镜像中包括它们。否则，应用它们作为额外的清单。

15.2.3.2. 使用生命周期代理生成 seed 镜像

使用 Lifecycle Agent 生成带有 SeedGenerator CR 的 seed 镜像。Operator 会检查所需的系统配置，在生成 seed 镜像前执行任何必要的系统清理，并启动镜像生成。seed 镜像生成包括以下任务：

- **停止集群 Operator**
- **准备 seed 镜像配置**
- **生成并推送 seed Generator CR 中指定的镜像存储库**
- **恢复集群 Operator**

- 使 seed 集群证书过期
- 为 seed 集群生成新证书
- 在 seed 集群中恢复和更新 SeedGenerator CR

先决条件

- 在 seed 集群上配置共享目录。
- 在 seed 集群上安装 OADP Operator 和 Lifecycle Agent。

流程

1. 从 hub 中分离集群，从 seed 集群中删除任何不能位于 seed 镜像中的特定于集群的资源：

   a. 如果使用 RHACM，请运行以下命令来手动分离 seed 集群：

   ```
   $ oc delete managedcluster sno-worker-example
   ```

      i. 等待 ManagedCluster CR 被删除。删除 CR 后，创建正确的 SeedGenerator CR。Lifecycle Agent 清理 RHACM 工件。

   b. 如果使用 GitOps ZTP，请通过从 kustomization.yaml 中删除 seed 集群的 SiteConfig CR 来分离集群：

      i. 从 kustomization.yaml 中删除您看到的集群的 SiteConfig CR。

   ```
   apiVersion: kustomize.config.k8s.io/v1beta1
   kind: Kustomization

   generators:
   #- example-seed-sno1.yaml
   - example-target-sno2.yaml
   - example-target-sno3.yaml
   ```

ii. 在 Git 存储库中提交 kustomization.yaml 更改并推送更改。

ArgoCD 管道检测到更改并删除受管集群。

2. 创建 Secret ：

a. 运行以下命令来创建身份验证文件：

```
$ MY_USER=myuserid
$ AUTHFILE=/tmp/my-auth.json
$ podman login --authfile ${AUTHFILE} -u ${MY_USER} quay.io/${MY_USER}
```

```
$ base64 -w 0 ${AUTHFILE} ; echo
```

b. 将输出复制到 openshift-lifecycle-agent 命名空间中名为 seedgen 的 Secret YAML 文件中的 seedAuth 字段中：

```
apiVersion: v1
kind: Secret
metadata:
  name: seedgen (1)
  namespace: openshift-lifecycle-agent
type: Opaque
data:
  seedAuth: <encoded_AUTHFILE> (2)
```

(1) Secret 资源必须具有 name: seedgen 和 namespace: openshift-lifecycle-agent 字段。

(2) 指定用于推送生成的 seed 镜像的 write-access 到 registry 的 base64 编码的 authfile。

c. 运行以下命令来应用 Secret ：

```
$ oc apply -f secretseedgenerator.yaml
```

3. 创建 SeedGenerator CR：

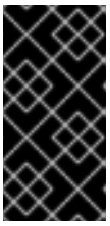
```
apiVersion: lca.openshift.io/v1
kind: SeedGenerator
metadata:
  name: seedimage (1)
spec:
  seedImage: <seed_container_image> (2)
```

(1) SeedGenerator CR 必须命名为 seedimage。

(2) 指定容器镜像 URL，如 quay.io/example/seed-container-image:<tag>。建议您使用 < seed_cluster_name>:<ocp_version> 格式。

4. 运行以下命令来生成 seed 镜像：

```
$ oc apply -f seedgenerator.yaml
```



重要

当 Lifecycle Agent 生成 seed 镜像时，集群会重启并丢失 API 功能。应用 SeedGenerator CR 会停止 kubelet 和 CRI-O 操作，然后启动镜像生成。

如果要生成更多看到的镜像，则必须使用您要从中生成 seed 镜像的版本置备一个新的 seed 集群。

验证

1. 集群恢复并可用后，您可以通过运行以下命令来检查 SeedGenerator CR 的状态：

```
$ oc get seedgenerator -o yaml
```

输出示例

```
status:
```

```
conditions:
- lastTransitionTime: "2024-02-13T21:24:26Z"
  message: Seed Generation completed
  observedGeneration: 1
  reason: Completed
  status: "False"
  type: SeedGenInProgress
- lastTransitionTime: "2024-02-13T21:24:26Z"
  message: Seed Generation completed
  observedGeneration: 1
  reason: Completed
  status: "True"
  type: SeedGenCompleted (1)
observedGeneration: 1
```

(1) seed 图像生成已完成。

其他资源

- 在 ostree stateroot 之间配置共享容器目录
- 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录

15.2.4. 使用 Lifecycle Agent 为基于镜像的升级创建 ConfigMap 对象

Lifecycle Agent 需要嵌套在 ConfigMap 对象中的所有 OADP 资源、额外清单和自定义目录源才能处理它们以进行基于镜像的升级。

15.2.4.1. 使用生命周期代理为基于镜像的升级创建 OADP ConfigMap 对象

创建用于在升级过程中备份和恢复资源的 OADP 资源。

先决条件

- 从兼容 seed 集群生成 seed 镜像。

- 创建 OADP 备份和恢复资源。

- 在目标集群中为在 stateroots 之间共享的容器镜像创建一个单独的分区。有关的更多信息，请参阅"为基于镜像升级配置共享容器目录"。

- 部署与 seed 镜像一起使用的版本兼容的生命周期代理版本。

- 在目标集群中安装 OADP Operator、DataProtectionApplication CR 及其 secret。

- 创建一个兼容 S3 的存储解决方案，以及一个可直接使用的存储桶，并配置了正确的凭证。如需更多信息，请参阅"关于安装 OADP"。

流程

1. 为安装 OADP Operator 的同一命名空间中的平台工件(openshift-adp)创建 OADP Backup 和 Restore CR。

   a. 如果目标集群由 RHACM 管理，请添加以下 YAML 文件来备份和恢复 RHACM 工件：

用于 RHACM 的 PlatformBackupRestore.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  annotations:
    lca.openshift.io/apply-label: "apps/v1/deployments/open-cluster-management-agent/klusterlet,v1/secrets/open-cluster-management-agent/bootstrap-hub-kubeconfig,rbac.authorization.k8s.io/v1/clusterroles/klusterlet,v1/serviceaccounts/open-cluster-management-agent/klusterlet,scheduling.k8s.io/v1/priorityclasses/klusterlet-critical,rbac.authorization.k8s.io/v1/clusterroles/open-cluster-management:klusterlet-admin-aggregate-clusterrole,rbac.authorization.k8s.io/v1/clusterrolebindings/klusterlet,operator.open-cluster-management.io/v1/klusterlets/klusterlet,apiextensions.k8s.io/v1/customresourcedefinitions/klusterlets.operator.open-cluster-management.io,v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials" (1)
  labels:
    velero.io/storage-location: default
```

```
  namespace: openshift-adp
spec:
  includedNamespaces:
  - open-cluster-management-agent
  includedClusterScopedResources:
  - klusterlets.operator.open-cluster-management.io
  - clusterroles.rbac.authorization.k8s.io
  - clusterrolebindings.rbac.authorization.k8s.io
  - priorityclasses.scheduling.k8s.io
  includedNamespaceScopedResources:
  - deployments
  - serviceaccounts
  - secrets
  excludedNamespaceScopedResources: []
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "1"
spec:
  backupName:
    acm-klusterlet
```

1

如果您的 multiclusterHub CR 没有定义 .spec.imagePullSecret，且 hub 集群中的 open-cluster-management-agent 命名空间中不存在 secret，请删除 v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials。

b. 如果您通过 LVM 存储在集群中创建了持久性卷，请为 LVM Storage 工件添加以下 YAML 文件：

用于 LVM 存储的 PlatformBackupRestoreLvms.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
```

```
  name: lvmcluster
  namespace: openshift-adp
spec:
  includedNamespaces:
    - openshift-storage
  includedNamespaceScopedResources:
    - lvmclusters
    - lvmvolumegroups
    - lvmvolumegroupnodestatuses
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: lvmcluster
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "2" (1)
spec:
  backupName:
    lvmcluster
```

(1) lca.openshift.io/apply-wave 值必须小于应用程序 Restore CR 中指定的值。

2. （可选）如果您需要升级后恢复应用程序，在 openshift-adp 命名空间中为应用程序创建 OADP 备份和恢复 CR。

   a. 在 openshift-adp 命名空间中为集群范围的应用程序工件创建 OADP CR。

LSO 和 LVM Storage 的集群范围应用程序工件的 OADP CR 示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    lca.openshift.io/apply-label:
"apiextensions.k8s.io/v1/customresourcedefinitions/test.example.com,security.openshift.io/v1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterroles/test-
role,rbac.authorization.k8s.io/v1/clusterrolebindings/system:openshift:scc:test"
```

```
  name: backup-app-cluster-resources (1)
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedClusterScopedResources:
  - customresourcedefinitions
  - securitycontextconstraints
  - clusterrolebindings
  - clusterroles
  excludedClusterScopedResources:
  - Namespace
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app-cluster-resources
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3" (2)
spec:
  backupName:
    backup-app-cluster-resources
```

(1) 将示例资源名称替换为您的实际资源。

(2) lca.openshift.io/apply-wave 值必须高于平台 Restore CR 中的值，并低于应用程序命名空间范围 Restore CR 中的值。

b. 为您的命名空间范围的应用程序工件创建 OADP CR。

使用 LSO 时 OADP CR 命名空间范围的应用程序工件示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
```

```
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  excludedClusterScopedResources:
  - persistentVolumes
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
```

1

为应用程序定义自定义资源。

使用 LVM 存储时 OADP CR 命名空间范围的应用程序工件示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
```

```
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  includedClusterScopedResources:
  - persistentVolumes 2
  - logicalvolumes.topolvm.io 3
  - volumesnapshotcontents 4
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
  restorePVs: true
  restoreStatus:
    includedResources:
    - logicalvolumes 5
```

1

为应用程序定义自定义资源。

2

必填字段。

3

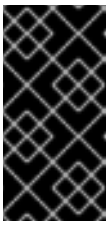
必填字段

4

如果使用 LVM 存储卷快照，则可选。

5

必填字段。



重要

同一版本的应用程序必须在 OpenShift Container Platform 的当前和目标发行版本中正常工作。

3. 运行以下命令，为 OADP CR 创建 ConfigMap 对象：

```
$ oc create configmap oadp-cm-example --from-file=example-oadp-resources.yaml=
<path_to_oadp_crs> -n openshift-adp
```

4. 运行以下命令来修补 ImageBasedUpgrade CR：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"oadpContent": [{"name": "oadp-cm-example", "namespace":
"openshift-adp"}]}}' \
  --type=merge -n openshift-lifecycle-agent
```

其他资源

- 在 ostree stateroot 之间配置共享容器目录
- 关于安装 OADP

15.2.4.2. 使用生命周期代理为基于镜像的升级创建额外清单的 ConfigMap 对象

您可以创建要应用到目标集群的额外清单。

流程

1. 创建包含额外清单的 YAML 文件，如 SR-IOV。

SR-IOV 资源示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: "example-sriov-node-policy"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  isRdma: false
  nicSelector:
    pfNames: [ens1f0]
  nodeSelector:
    node-role.kubernetes.io/master: ""
  mtu: 1500
  numVfs: 8
  priority: 99
  resourceName: example-sriov-node-policy
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: "example-sriov-network"
  namespace: openshift-sriov-network-operator
spec:
  ipam: |-
    {
    }
  linkState: auto
  networkNamespace: sriov-namespace
  resourceName: example-sriov-node-policy
  spoofChk: "on"
  trust: "off"
```

2. 运行以下命令来创建 ConfigMap 对象：

```
$ oc create configmap example-extra-manifests-cm --from-file=example-extra-
manifests.yaml=<path_to_extramanifest> -n openshift-lifecycle-agent
```

3. 运行以下命令来修补 ImageBasedUpgrade CR：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"extraManifests": [{"name": "example-extra-manifests-cm",
"namespace": "openshift-lifecycle-agent"}]}}' \
```

```
--type=merge -n openshift-lifecycle-agent
```

15.2.4.3. 使用 Lifecycle Agent 为基于镜像的升级创建自定义目录源的 ConfigMap 对象

您可以通过为目录源生成 ConfigMap 对象并将其添加到 ImageBasedUpgrade CR 的 spec.extraManifest 字段中，在升级后保留自定义目录源。有关目录源的更多信息，请参阅"目录源"。

流程

1. 创建包含 CatalogSource CR 的 YAML 文件：

```
apiVersion: operators.coreos.com/v1
kind: CatalogSource
metadata:
  name: example-catalogsources
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  displayName: disconnected-redhat-operators
  image: quay.io/example-org/example-catalog:v1
```

2. 运行以下命令来创建 ConfigMap 对象：

```
$ oc create configmap example-catalogsources-cm --from-file=example-
catalogsources.yaml=<path_to_catalogsource_cr> -n openshift-lifecycle-agent
```

3. 运行以下命令来修补 ImageBasedUpgrade CR：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"extraManifests": [{"name": "example-catalogsources-cm",
"namespace": "openshift-lifecycle-agent"}]}}' \
  --type=merge -n openshift-lifecycle-agent
```

其他资源

- 目录源
- 使用生命周期代理执行基于镜像的升级

15.2.5. 使用 GitOps ZTP 为基于镜像的升级创建 ConfigMap 对象

创建 OADP 资源、额外清单和自定义目录源，以准备基于镜像的升级。

15.2.5.1. 使用 GitOps ZTP 为基于镜像的升级创建 OADP 资源

准备 OADP 资源，以便在升级后恢复应用程序。

先决条件

- 使用 GitOps ZTP 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 从兼容 seed 集群生成 seed 镜像。
- 在目标集群中为在 stateroots 之间共享的容器镜像创建一个单独的分区。如需更多信息，请参阅"在使用 GitOps ZTP 时，在 ostree stateroots 之间配置共享容器目录"。
- 部署与 seed 镜像一起使用的版本兼容的生命周期代理版本。
- 在目标集群中安装 OADP Operator、DataProtectionApplication CR 及其 secret。
- 创建一个兼容 S3 的存储解决方案，以及一个可直接使用的存储桶，并配置了正确的凭证。如需更多信息，请参阅"使用 GitOps ZTP 安装和配置 OADP Operator"。

流程

1. 确保用于 ArgoCD 策略应用程序的 Git 存储库包含以下目录结构：

```
├── source-crs/
│   ├── ibu/
│   │   ├── ImageBasedUpgrade.yaml
│   │   ├── PlatformBackupRestore.yaml
│   │   ├── PlatformBackupRestoreLvms.yaml
├── ...
├── ibu-upgrade-ranGen.yaml
├── kustomization.yaml
```

-

重要

kustomization.yaml 文件必须位于与前面所示相同的目录结构中，以便引用 ibu-upgrade-ranGen.yaml 清单。

source-crs/ibu/PlatformBackupRestore.yaml 文件在 ZTP 容器镜像中提供。

PlatformBackupRestore.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  annotations:
    lca.openshift.io/apply-label: "apps/v1/deployments/open-cluster-management-
agent/klusterlet,v1/secrets/open-cluster-management-agent/bootstrap-hub-
kubeconfig,rbac.authorization.k8s.io/v1/clusterroles/klusterlet,v1/serviceaccounts/ope
n-cluster-management-agent/klusterlet,scheduling.k8s.io/v1/priorityclasses/klusterlet-
critical,rbac.authorization.k8s.io/v1/clusterroles/open-cluster-management:klusterlet-
admin-aggregate-
clusterrole,rbac.authorization.k8s.io/v1/clusterrolebindings/klusterlet,operator.open-
cluster-
management.io/v1/klusterlets/klusterlet,apiextensions.k8s.io/v1/customresourcedefinit
ions/klusterlets.operator.open-cluster-management.io,v1/secrets/open-cluster-
management-agent/open-cluster-management-image-pull-credentials" (1)
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedNamespaces:
  - open-cluster-management-agent
  includedClusterScopedResources:
  - klusterlets.operator.open-cluster-management.io
  - clusterroles.rbac.authorization.k8s.io
  - clusterrolebindings.rbac.authorization.k8s.io
  - priorityclasses.scheduling.k8s.io
  includedNamespaceScopedResources:
  - deployments
  - serviceaccounts
  - secrets
  excludedNamespaceScopedResources: []
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
```

```
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "1"
spec:
  backupName:
    acm-klusterlet
```

1

如果您的 multiclusterHub CR 没有定义 .spec.imagePullSecret，且 hub 集群中的 open-cluster-management-agent 命名空间中不存在 secret，请删除 v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials。

如果使用 LVM 存储来创建持久性卷，您可以使用 ZTP 容器镜像中提供的 source-crs/ibu/PlatformBackupRestoreLvms.yaml 来备份 LVM 存储资源。

PlatformBackupRestoreLvms.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: lvmcluster
  namespace: openshift-adp
spec:
  includedNamespaces:
    - openshift-storage
  includedNamespaceScopedResources:
    - lvmclusters
    - lvmvolumegroups
    - lvmvolumegroupnodestatuses
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: lvmcluster
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "2" 1
```

```
spec:
  backupName:
    lvmcluster
```

1

lca.openshift.io/apply-wave 值必须小于应用程序 Restore CR 中指定的值。

2. 可选：如果您需要在升级后恢复应用程序，在 openshift-adp 命名空间中为应用程序创建 OADP 备份和恢复 CR：

a. 在 openshift-adp 命名空间中为集群范围的应用程序工件创建 OADP CR：

LSO 和 LVM Storage 的集群范围应用程序工件的 OADP CR 示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    lca.openshift.io/apply-label:
"apiextensions.k8s.io/v1/customresourcedefinitions/test.example.com,security.openshift.io/v1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterroles/test-
role,rbac.authorization.k8s.io/v1/clusterrolebindings/system:openshift:scc:test"
1
  name: backup-app-cluster-resources
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedClusterScopedResources:
  - customresourcedefinitions
  - securitycontextconstraints
  - clusterrolebindings
  - clusterroles
  excludedClusterScopedResources:
  - Namespace
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app-cluster-resources
  namespace: openshift-adp
```

```
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3" 2
spec:
  backupName:
    backup-app-cluster-resources
```

1

将示例资源名称替换为您的实际资源。

2

lca.openshift.io/apply-wave 值必须高于平台 Restore CR 中的值，并低于应用程序命名空间范围 Restore CR 中的值。

b. 在 source-crs/custom-crs 目录中为您的命名空间范围的应用程序工件创建 OADP CR：

使用 LSO 时 OADP CR 命名空间范围的应用程序工件示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  excludedClusterScopedResources:
```

```
  - persistentVolumes
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
```

1

为应用程序定义自定义资源。

使用 LVM 存储时 OADP CR 命名空间范围的应用程序工件示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  includedClusterScopedResources:
  - persistentVolumes 2
  - logicalvolumes.topolvm.io 3
  - volumesnapshotcontents 4
---
```

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
  restorePVs: true
  restoreStatus:
    includedResources:
    - logicalvolumes 5
```

1

为应用程序定义自定义资源。

2

必填字段。

3

必填字段

4

如果使用 LVM 存储卷快照，则可选。

5

必填字段。

重要

同一版本的应用程序必须在 OpenShift Container Platform 的当前和目标发行版本中正常工作。

3.

通过名为 ibu-upgrade -ranGen.yaml 的新 PolicyGenTemplate 中的 oadp-cm -policy 来创建 oadp-cm ConfigMap 对象：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: example-group-ibu
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  evaluationInterval:
    compliant: 10s
    noncompliant: 10s
  sourceFiles:
  - fileName: ConfigMapGeneric.yaml
    complianceType: mustonlyhave
    policyName: "oadp-cm-policy"
    metadata:
      name: oadp-cm
      namespace: openshift-adp
```

4. 使用以下内容创建一个 kustomization.yaml ：

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators: (1)
- ibu-upgrade-ranGen.yaml

configMapGenerator: (2)
- files:
  - source-crs/ibu/PlatformBackupRestore.yaml
  #- source-crs/custom-crs/ApplicationClusterScopedBackupRestore.yaml
  #- source-crs/custom-crs/ApplicationApplicationBackupRestoreLso.yaml
  name: oadp-cm
  namespace: ztp-group
generatorOptions:
  disableNameSuffixHash: true


patches: (3)
- target:
    group: policy.open-cluster-management.io
    version: v1
    kind: Policy
    name: group-ibu-oadp-cm-policy
  patch: |-
    - op: replace
```

```
    path: /spec/policy-templates/0/objectDefinition/spec/object-
templates/0/objectDefinition/data
    value: '{{hub copyConfigMapData "ztp-group" "oadp-cm" hub}}'
```

1 生成 oadp-cm-policy。

2 使用 Backup 和 Restore CR 在 hub 集群中创建 oadp-cm ConfigMap 对象。

3 覆盖在 oadp-cm-policy 中添加了 ConfigMap 的 data 字段。hub 模板用于将 oadp-cm ConfigMap 传播到所有目标集群。

5. 将更改推送到您的 Git 存储库。

其他资源

- 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录
- 使用 GitOps ZTP 安装和配置 OADP Operator

15.2.5.2. 使用 GitOps ZTP 为基于镜像的升级标记额外的清单

标记您的额外清单，以便 Lifecycle Agent 可以提取使用 lca.openshift.io/target-ocp-version: <target_version> 标签标记 的资源。

先决条件

- 使用 GitOps ZTP 置备一个或多个受管集群。
- 以具有 cluster-admin 特权的用户身份登录。
- 从兼容 seed 集群生成 seed 镜像。

- 在目标集群中为在 stateroots 之间共享的容器镜像创建一个单独的分区。如需更多信息，请参阅"在使用 GitOps ZTP 时，在 ostree stateroots 之间配置共享容器目录"。

- 部署与 seed 镜像一起使用的版本兼容的生命周期代理版本。

流程

1. 使用现有 PolicyGenTemplate CR 中的 lca.openshift.io/target-ocp-version: <target_version > 标签标记所需的额外清单：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: example-sno
spec:
  bindingRules:
    sites: "example-sno"
    du-profile: "4.15"
  mcp: "master"
  sourceFiles:
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-fh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15" (1)
      spec:
        resourceName: du_fh
        vlan: 140
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-fh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15"
      spec:
        deviceType: netdevice
        isRdma: false
        nicSelector:
          pfNames: ["ens5f0"]
        numVfs: 8
        priority: 10
        resourceName: du_fh
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-mh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15"
      spec:
```

```
    resourceName: du_mh
    vlan: 150
  - fileName: SriovNetworkNodePolicy.yaml
    policyName: "config-policy"
    metadata:
      name: "sriov-nnp-du-mh"
      labels:
        lca.openshift.io/target-ocp-version: "4.15"
    spec:
      deviceType: vfio-pci
      isRdma: false
      nicSelector:
        pfNames: ["ens7f0"]
      numVfs: 8
      priority: 10
      resourceName: du_mh
  - fileName: DefaultCatsrc.yaml 2
    policyName: "config-policy"
    metadata:
      name: default-cat-source
      namespace: openshift-marketplace
      labels:
        lca.openshift.io/target-ocp-version: "4.15"
    spec:
      displayName: default-cat-source
      image: quay.io/example-org/example-catalog:v1
```

1 确保 lca.openshift.io/target-ocp-version 标签与 ImageBasedUpgrade CR 的 spec.seedImageRef.version 字段中指定的目标 OpenShift Container Platform 版本的 y-stream 或 z-stream 匹配。Lifecycle Agent 仅应用与指定版本匹配的 CR。

2 如果您不想使用自定义目录源，请删除此条目。

2. 将更改推送到您的 Git 存储库。

其他资源

- 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录
- 使用 GitOps ZTP 为单节点 OpenShift 集群执行基于镜像的升级

15.3. 为单节点 OPENSHIFT 集群执行基于镜像的升级

您可以使用 Lifecycle Agent 手动升级单节点 OpenShift 集群。

当您在集群中部署生命周期代理时，会自动创建 ImageBasedUpgrade CR。您可以更新此 CR，以指定 seed 镜像的镜像存储库，并通过不同的阶段移动。

15.3.1. 使用生命周期代理移动到基于镜像的升级的 Prep 阶段

当您在集群中部署生命周期代理时，会自动创建 ImageBasedUpgrade 自定义资源(CR)。

在创建了升级过程中需要的所有资源后，您可以进入 Prep 阶段。如需更多信息，请参阅"使用 Lifecycle Agent 为基于镜像升级创建 ConfigMap 对象"部分。

先决条件

- 您已创建了资源来备份和恢复集群。

流程

1. 检查您是否修补了 ImageBasedUpgrade CR:

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle
  seedImageRef:
    version: 4.15.2 (1)
    image: <seed_container_image> (2)
    pullSecretRef: <seed_pull_secret> (3)
  autoRollbackOnFailure: {}
#    initMonitorTimeoutSeconds: 1800 (4)
  extraManifests: (5)
  - name: example-extra-manifests-cm
    namespace: openshift-lifecycle-agent
  - name: example-catalogsources-cm
    namespace: openshift-lifecycle-agent
  oadpContent: (6)
  - name: oadp-cm-example
    namespace: openshift-adp
```

(1)

指定目标平台版本。该值必须与 seed 镜像的版本匹配。

2

指定目标集群可从中拉取 seed 镜像的存储库。

3

如果镜像位于私有 registry 中，则指定对带有凭证的 secret 的引用。

4

（可选）指定在第一次重启后在那个时间段内回滚的时间帧（以秒为单位）。如果没有定义或设置为 0， 则使用默认值 1800 秒(30 分钟)。

5

（可选）指定在升级后包含自定义目录源的 ConfigMap 资源列表，以及要应用到不是 seed 镜像一部分的目标集群的额外清单。

6

使用 OADP ConfigMap 信息添加 oadpContent 部分。

2. 要启动 Prep 阶段，请运行以下命令将 stage 字段的值改为 ImageBasedUpgrade CR 中的 Prep ：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Prep"}}' --type=merge -n openshift-lifecycle-agent
```

如果您为 OADP 资源和额外清单提供 ConfigMap 对象，则 Lifecycle Agent 会在 Prep 阶段验证指定的 ConfigMap 对象。您可能会遇到以下问题：

- 如果 Lifecycle Agent 检测到 extraManifests 参数的问题，验证警告或错误。
- 如果 Lifecycle Agent 检测到 oadpContent 参数的任何问题，验证错误。

验证警告不会阻止 Upgrade 阶段，但您必须确定是否可以安全地进行升级。这些警告（如缺少 CRD、命名空间或空运行失败）更新 ImageBasedUpgrade CR 中的 Prep stage 和

annotation 字段的 status.conditions，其中包含有关警告的详情。

验证警告示例

```
[...]
metadata:
annotations:
  extra-manifest.lca.openshift.io/validation-warning: '...'
[...]
```

但是，验证错误，如将 MachineConfig 或 Operator 清单添加到额外清单中，从而导致 Prep 阶段失败并阻止 Upgrade 阶段。

当验证通过时，集群会创建一个新的 ostree stateroot，它涉及拉取和解包 seed 镜像，并运行主机级命令。最后，目标集群中会包括所有必需的镜像。

验证

- 运行以下命令，检查 ImageBasedUpgrade CR 的状态：

```
$ oc get ibu -o yaml
```

输出示例

```
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 13
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 13
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep stage completed successfully
```

```
    observedGeneration: 13
    reason: Completed
    status: "True"
    type: PrepCompleted
  observedGeneration: 13
  validNextStages:
  - Idle
  - Upgrade
```

其他资源

- 使用 Lifecycle Agent 为基于镜像的升级创建 ConfigMap 对象

15.3.2. 使用生命周期代理移动到基于镜像的升级升级阶段

生成 seed 镜像并完成 Prep 阶段后，您可以升级目标集群。在升级过程中，OADP Operator 会创建 OADP 自定义资源(CR)中指定的工件备份，然后升级集群。

如果升级失败或停止，则会启动一个自动回滚。如果在升级后有问题，您可以启动手动回滚。有关手动回滚的更多信息，请参阅" （可选）使用生命周期代理启动回滚"。

先决条件

- 完成 Prep 阶段。

流程

1. 要移到 Upgrade 阶段，请运行以下命令将 ImageBased Upgrade CR 中的 stage 字段的值改为 Upgrade ：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Upgrade"}}' --type=merge
```

2. 运行以下命令，检查 ImageBasedUpgrade CR 的状态：

```
$ oc get ibu -o yaml
```

输出示例

```
status:
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 5
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 5
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed successfully
    observedGeneration: 5
    reason: Completed
    status: "True"
    type: PrepCompleted
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: |-
      Waiting for system to stabilize: one or more health checks failed
        - one or more ClusterOperators not yet ready: authentication
        - one or more MachineConfigPools not yet ready: master
        - one or more ClusterServiceVersions not yet ready: sriov-fec.v2.8.0
    observedGeneration: 1
    reason: InProgress
    status: "True"
    type: UpgradeInProgress
  observedGeneration: 1
  rollbackAvailabilityExpiration: "2024-05-19T14:01:52Z"
  validNextStages:
  - Rollback
```

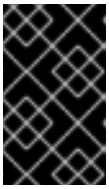
OADP Operator 创建 OADP 备份和恢复 CR 中指定的数据的备份，目标集群重启。

3. 运行以下命令监控 CR 的状态：

```
$ oc get ibu -o yaml
```

4. 如果您对升级满意，请运行以下命令在 ImageBasedUpgrade CR 中将 stage 字段的值修补为 Idle 来完成更改：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Idle"}}' --type=merge
```



重要

升级后，您无法回滚更改。

Lifecycle Agent 会删除升级过程中创建的所有资源。

验证

1. 运行以下命令，检查 ImageBasedUpgrade CR 的状态：

```
$ oc get ibu -o yaml
```

输出示例

```
status:
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 5
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 5
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed successfully
    observedGeneration: 5
    reason: Completed
    status: "True"
    type: PrepCompleted
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Upgrade completed
    observedGeneration: 1
    reason: Completed
    status: "False"
    type: UpgradeInProgress
```

```
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Upgrade completed
    observedGeneration: 1
    reason: Completed
    status: "True"
    type: UpgradeCompleted
  observedGeneration: 1
  rollbackAvailabilityExpiration: "2024-01-01T09:00:00Z"
  validNextStages:
  - Idle
  - Rollback
```

2. 运行以下命令检查集群恢复的状态：

```
$ oc get restores -n openshift-adp -o custom-
columns=NAME:.metadata.name,Status:.status.phase,Reason:.status.failureReason
```

输出示例

```
NAME           Status      Reason
acm-klusterlet   Completed   <none> 1
apache-app       Completed   <none>
localvolume      Completed   <none>
```

1 acm-klusterlet 只特定于 RHACM 环境。

其他资源

- （可选）使用 Lifecycle Agent 移到基于镜像的升级的 Rollback 阶段

15.3.3. （可选）使用 Lifecycle Agent 移到基于镜像的升级的 Rollback 阶段

如果升级在重新引导后在 initMonitorTimeoutSeconds 字段中指定的时间范围内没有完成，则会启动一个自动回滚。

Example ImageBasedUpgrade CR

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle
  seedImageRef:
    version: 4.15.2
    image: <seed_container_image>
  autoRollbackOnFailure: {}
#   initMonitorTimeoutSeconds: 1800 (1)
[...]
```

(1) （可选）指定在第一次重启后在那个时间段内回滚的时间帧（以秒为单位）。如果没有定义或设置为 0， 则使用默认值 1800 秒(30 分钟)。

如果您在升级后遇到无法解析的问题，您可以手动回滚更改。

先决条件

- 以具有 cluster-admin 权限的用户身份登录 hub 集群。

流程

1. 要移到回滚阶段，请运行以下命令对 ImageBasedUpgrade CR 中的 stage 字段的值进行补丁：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Rollback"}}' --type=merge
```

Lifecycle Agent 使用之前安装的 OpenShift Container Platform 版本重启集群，并恢复应用程序。

2. 如果您对更改满意，请运行以下命令在 ImageBasedUpgrade CR 中将 stage 字段的值修补

为 Idle 来完成回滚：

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Idle"}}' --type=merge -n openshift-lifecycle-agent
```

警告

如果在回滚后移至 Idle 阶段，则 Lifecycle Agent 会清理可用于对失败的升级进行故障排除的资源。

15.3.4. 使用生命周期代理对基于镜像的升级进行故障排除

在基于镜像的升级过程中可能会遇到问题。

15.3.4.1. 收集日志

您可以使用 oc adm must-gather CLI 来收集用于调试和故障排除的信息。

流程

- 运行以下命令收集有关 Operator 的数据：

```
$  oc adm must-gather \
--dest-dir=must-gather/tmp \
--image=$(oc -n openshift-lifecycle-agent get deployment.apps/lifecycle-agent-
controller-manager -o jsonpath='{.spec.template.spec.containers[?(@.name ==
"manager")].image}') \
--image=quay.io/konveyor/oadp-must-gather:latest \ 1
--image=quay.io/openshift/origin-must-gather:latest 2
```

1. （可选）如果您需要从 OADP Operator 收集更多信息，您可以添加这个选项。

2. （可选）如果您需要从 SR-IOV Operator 收集更多信息，您可以添加这个选项。

15.3.4.2. AbortFailed 或 FinalizeFailed 错误

问题

在完成阶段或停止 Prep 阶段时，生命周期代理会清理以下资源：

- Stateroot 不再需要
- 预缓存资源
- OADP CR
- ImageBasedUpgrade CR

如果 Lifecycle Agent 无法执行上述步骤，它将过渡到 AbortFailed 或 FinalizeFailed 状态。条件消息和日志显示哪些步骤失败。

错误信息示例

```
message: failed to delete all the backup CRs. Perform cleanup manually then add
'lca.openshift.io/manual-cleanup-done' annotation to ibu CR to transition back to Idle
    observedGeneration: 5
    reason: AbortFailed
    status: "False"
    type: Idle
```

解决方案

1. 检查日志，以确定发生失败的原因。
2. 要提示生命周期代理重试清理，请将 lca.openshift.io/manual-cleanup-done 注解添加到 ImageBasedUpgrade CR。

   观察此注解后，生命周期代理会重试清理，如果成功，ImageBasedUpgrade 阶段会过

渡到 Idle。

如果清理再次失败，您可以手动清理资源。

15.3.4.2.1. 手动清理 stateroot

问题

在 Prep 阶段停止，生命周期代理会清理新的 stateroot。在成功升级或回滚后最终调整时，生命周期代理会清理旧的 stateroot。如果此步骤失败，建议您检查日志以确定失败的原因。

解决方案

1. 运行以下命令，检查 stateroot 中是否有现有部署：

```
$ ostree admin status
```

2. 如果存在，请运行以下命令清理现有部署：

```
$ ostree admin undeploy <index_of_deployment>
```

3. 清理 stateroot 的所有部署后，运行以下命令来擦除 stateroot 目录：

警告

确保引导的部署不处于这个 stateroot。

```
$ stateroot="<stateroot_to_delete>"
```

```
$ unshare -m /bin/sh -c "mount -o remount,rw /sysroot && rm -rf
/sysroot/ostree/deploy/${stateroot}"
```

15.3.4.2.2. 手动清理 OADP 资源

问题

由于生命周期代理和 S3 后端之间的连接问题，自动清理 OADP 资源可能会失败。通过恢复连接并添加 lca.openshift.io/manual-cleanup-done 注解，生命周期代理可以成功清理备份资源。

解决方案

1. 运行以下命令检查后端连接：

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

输出示例

```
NAME                          PHASE       LAST VALIDATED   AGE   DEFAULT
dataprotectionapplication-1   Available   33s              8d    true
```

2. 删除所有备份资源，然后将 lca.openshift.io/manual-cleanup-done 注解添加到 ImageBasedUpgrade CR。

15.3.4.3. LVM 存储卷内容没有恢复

当使用 LVM 存储来提供动态持久性卷存储时，如果配置不正确，LVM 存储可能无法恢复持久性卷内容。

15.3.4.3.1. Backup CR 中缺少与 LVM Storage 相关的字段

问题

您的 Backup CR 可能缺少恢复持久性卷所需的字段。您可以通过运行以下命令来检查应用程序 pod 中的事件，以确定是否出现这个问题：

```
$ oc describe pod <your_app_name>
```

显示 Backup CR 中缺少 LVM Storage 相关字段的输出示例

```
Events:
  Type     Reason          Age              From               Message
  ----     ------          ----             ----               -------
```

```
  Warning  FailedScheduling  58s (x2 over 66s)  default-scheduler  0/1 nodes are available:
pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available:
1 Preemption is not helpful for scheduling..
  Normal   Scheduled         56s                default-scheduler  Successfully assigned default/db-
1234 to sno1.example.lab
  Warning  FailedMount       24s (x7 over 55s)  kubelet            MountVolume.SetUp failed for
volume "pvc-1234" : rpc error: code = Unknown desc = VolumeID is not found
```

解决方案

您必须在应用程序 Backup CR 中包含 logicalvolumes.topolvm.io。如果没有此资源，应用程序可以正确地恢复其持久性卷声明和持久性卷清单，但与这个持久性卷关联的 逻辑卷 在 pivot 后无法正确恢复。

备份 CR 示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: small-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  includedClusterScopedResources: (1)
  - persistentVolumes
  - volumesnapshotcontents
  - logicalvolumes.topolvm.io
```

(1)

要恢复应用程序的持久性卷，您必须配置本节，如下所示。

15.3.4.3.2. Restore CR 中缺少与 LVM Storage 相关的字段

问题

应用程序的预期资源会被恢复，但升级后持久性卷内容不会被保留。

1. 在 pivot 前运行以下命令来列出应用程序的持久性卷：

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

pivot 前的输出示例

```
NAME                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Retain           Bound    default/pvc-
db   lvms-vg1              4h45m

NAMESPACE   NAME                    STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       4h45m

NAMESPACE   NAME                          AGE
          logicalvolume.topolvm.io/pvc-1234   4h45m
```

2. 在 pivot 后运行以下命令来列出应用程序的持久性卷：

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

pivot 后的输出示例

```
NAME                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Delete           Bound    default/pvc-
db   lvms-vg1              19s

NAMESPACE   NAME                    STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       19s
```

```
NAMESPACE   NAME                                AGE
            logicalvolume.topolvm.io/pvc-1234   18s
```

解决方案

造成这个问题的原因是 logicalvolume 状态没有在 Restore CR 中保留。这个状态非常重要，因为 Velero 需要引用 pivoting 后必须保留的卷。您必须在应用程序 Restore CR 中包含以下字段：

Restore CR 示例

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: sample-vote-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3"
spec:
  backupName:
    sample-vote-app
  restorePVs: true (1)
  restoreStatus: (2)
    includedResources:
      - logicalvolumes
```

(1) 要保留应用程序的持久性卷，您必须将 restorePV 设置为 true。

(2) 要为应用程序保留持久性卷，您必须配置本节，如下所示。

15.3.4.4. 调试失败的备份和恢复 CR

问题

工件的备份或恢复失败。

解决方案

您可以调试 Backup 和 Restore CR，并使用 Velero CLI 工具检索日志。Velero CLI 工具比 OpenShift CLI 工具提供更详细的信息。

1. 运行以下命令描述包含错误的 Backup CR：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe backup -n openshift-adp backup-acm-klusterlet --details
```

2. 运行以下命令描述包含错误的 Restore CR：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe restore -n openshift-adp restore-acm-klusterlet --details
```

3. 运行以下命令，将备份的资源下载到本地目录：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero backup download -n openshift-adp backup-acm-klusterlet -o ~/backup-acm-klusterlet.tar.gz
```

15.4. 使用 GITOPS ZTP 为单节点 OPENSHIFT 集群执行基于镜像的升级

您可以通过 GitOps Zero Touch Provisioning (ZTP)使用基于镜像的升级来升级受管单节点 OpenShift 集群。

当您在集群中部署生命周期代理时，会自动创建 ImageBasedUpgrade CR。您可以更新此 CR，以指定 seed 镜像的镜像存储库，并通过不同的阶段移动。

15.4.1. 使用 Lifecycle Agent 和 GitOps ZTP 移到基于镜像的升级的 Prep 阶段

当您在集群中部署生命周期代理时，会自动创建 ImageBasedUpgrade CR。您可以更新此 CR，以指定 seed 镜像的镜像存储库，并通过不同的阶段移动。

先决条件

-

为基于镜像升级中使用的资源创建策略和 ConfigMap 对象。如需更多信息，请参阅"使用 GitOps ZTP 基于镜像升级创建 ConfigMap 对象

流程

1. 将 Prep、Upgrade 和 Idle 阶段的策略添加到名为 ibu-upgrade-ranGen.yaml 的现有组 PolicyGenTemplate 中：

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: example-group-ibu
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  evaluationInterval: 1
    compliant: 10s
    noncompliant: 10s
  sourceFiles:
    - fileName: ConfigMapGeneric.yaml
      complianceType: mustonlyhave
      policyName: "oadp-cm-policy"
      metadata:
        name: oadp-cm
        namespace: openshift-adp
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "prep-stage-policy"
      spec:
        stage: Prep
        seedImageRef: 2
          version: "4.15.0"
          image: "quay.io/user/lca-seed:4.15.0"
          pullSecretRef:
            name: "<seed_pull_secret>"
        oadpContent: 3
        - name: "oadp-cm"
          namespace: "openshift-adp"
      status:
        conditions:
          - reason: Completed
            status: "True"
            type: PrepCompleted
            message: "Prep stage completed successfully"
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "upgrade-stage-policy"
      spec:
        stage: Upgrade
      status:
        conditions:
          - reason: Completed
```

```
        status: "True"
        type: UpgradeCompleted
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "finalize-stage-policy"
      complianceType: mustonlyhave
      spec:
        stage: Idle
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "finalize-stage-policy"
      status:
        conditions:
          - reason: Idle
            status: "True"
            type: Idle
```

1. 合规和不合规的策略的策略评估间隔。将它们设置为 10s，以确保策略状态准确反映当前的升级状态。

2. 在 Prep 阶段为升级定义 seed 镜像、OpenShift Container Platform 版本和 pull secret。

3. 定义备份和恢复所需的 OADP ConfigMap 资源。

2. 运行以下命令，验证是否创建了基于镜像的升级所需的策略：

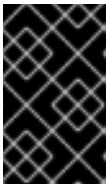
```
$ oc get policies -n spoke1 | grep -E "example-group-ibu"
```

输出示例

```
ztp-group.example-group-ibu-oadp-cm-policy       inform            NonCompliant
31h
ztp-group.example-group-ibu-prep-stage-policy        inform            NonCompliant
31h
ztp-group.example-group-ibu-upgrade-stage-policy      inform            NonCompliant
31h
ztp-group.example-group-ibu-finalize-stage-policy     inform            NonCompliant
31h
ztp-group.example-group-ibu-rollback-stage-policy     inform            NonCompliant
31h
```

3. 将 du-profile 集群标签更新至目标平台版本，或将 SiteConfig CR 中对应的 policy-binding 标签更新为目标平台。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
[...]
spec:
  [...]
    clusterLabels:
      du-profile: "4.15.0"
```



重要

将标签更新至目标平台版本，取消绑定现有策略集。

4. 提交更新的 SiteConfig CR 并将其推送到 Git 存储库。

5. 当您准备好进入 Prep 阶段时，使用 Prep 和 OADP ConfigMap 策略在目标 hub 集群上创建 ClusterGroupUpgrade CR：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-oadp-cm-policy
  - example-group-ibu-prep-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

6. 运行以下命令来应用 Prep 策略：

```
$ oc apply -f cgu-ibu-prep.yml
```

-

如果您为 OADP 资源和额外清单提供 ConfigMap 对象，则 Lifecycle Agent 会在 Prep 阶段验证指定的 ConfigMap 对象。您可能会遇到以下问题：

- 如果 Lifecycle Agent 检测到 extraManifests的问题，验证警告或错误
- 如果 Lifecycle Agent 检测到任何带有 oadpContent的问题，验证错误

验证警告不会阻止 Upgrade 阶段，但您必须确定是否可以安全地进行升级。这些警告（如缺少 CRD、命名空间或空运行失败）更新 ImageBasedUpgrade CR 中的 status.conditions 和 annotation 字段，其中包含有关警告的详情。

验证警告示例

```
[...]
metadata:
annotations:
  extra-manifest.lca.openshift.io/validation-warning: '...'
[...]
```

但是，验证错误，如将 MachineConfig 或 Operator 清单添加到额外清单中，从而导致 Prep 阶段失败并阻止 Upgrade 阶段。

当验证通过时，集群会创建一个新的 ostree stateroot，它涉及拉取和解包 seed 镜像，并运行主机级别命令。最后，目标集群中会包括所有必需的镜像。

7. 运行以下命令监控状态并等待 cgu-ibu-prep ClusterGroupUpgrade 报告 Completed ：

```
$ oc get cgu -n default
```

输出示例

```
NAME              AGE   STATE      DETAILS
cgu-ibu-prep      31h   Completed  All clusters are compliant with all the managed
policies
```

其他资源

- 为独立版本准备 GitOps ZTP 站点配置存储库
- 使用 GitOps ZTP 为基于镜像的升级创建 ConfigMap 对象
- 使用 GitOps ZTP 时在 ostree stateroots 之间配置共享容器目录
- 关于备份和恢复位置及其 secret
- 创建备份 CR
- 创建恢复 CR

15.4.2. 使用 Lifecycle Agent 和 GitOps ZTP 移到基于镜像的升级的 Upgrade 阶段

完成 Prep 阶段后，您可以升级目标集群。在升级过程中，OADP Operator 会为 OADP CR 中指定的工件创建备份，然后升级集群。

如果升级失败或停止，则会启动一个自动回滚。如果在升级后有问题，您可以启动手动回滚。有关手动回滚的更多信息，请参阅"（可选）使用 Lifecycle Agent 和 GitOps ZTP 启动回滚。

先决条件

- 完成 Prep 阶段。

流程

1.

当您准备好进入 Upgrade 阶段时，请在引用 Upgrade 策略的目标 hub 集群中创建 ClusterGroupUpgrade CR：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-upgrade
  namespace: default
spec:
  actions:
    beforeEnable:
      addClusterAnnotations:
        import.open-cluster-management.io/disable-auto-import: "true" 1
    afterCompletion:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import 2
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-upgrade-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

1

在开始升级前，将 disable-auto-import 注解应用到受管集群。此注解可确保在升级阶段禁用自动导入受管集群，直到集群就绪为止。

2

升级完成后删除 disable-auto-import 注解。

2. 运行以下命令来应用 Upgrade 策略：

```
$ oc apply -f cgu-ibu-upgrade.yml
```

3. 运行以下命令监控状态，并等待 cgu-ibu-upgrade ClusterGroupUpgrade 报告 Completed：

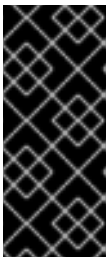
```
$ oc get cgu -n default
```

输出示例

```
NAME                    AGE   STATE      DETAILS
cgu-ibu-prep            31h   Completed  All clusters are compliant with all the managed
policies
cgu-ibu-upgrade         31h   Completed  All clusters are compliant with all the managed
policies
```

4. 当您对更改满意并准备好完成升级时，请在目标 hub 集群上创建一个 ClusterGroupUpgrade CR，该集群引用完成升级的策略：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-finalize
  namespace: default
spec:
  actions:
    beforeEnable:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-finalize-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```



重要

确保没有其他 ClusterGroupUpgrade CR 正在进行，因为这会导致 TALM 持续协调它们。在应用 cgu-ibu-finalize.yaml 前，删除所有 "In-Progress" ClusterGroupUpgrade CR。

5. 运行以下命令来应用策略：

```
$ oc apply -f cgu-ibu-finalize.yaml
```

6.

运行以下命令监控状态：并等待 cgu-ibu-finalize ClusterGroupUpgrade 报告 Completed：

```
$ oc get cgu -n default
```

输出示例

```
NAME                AGE   STATE       DETAILS
cgu-ibu-finalize      30h   Completed   All clusters are compliant with all the managed policies
cgu-ibu-prep          31h   Completed   All clusters are compliant with all the managed policies
cgu-ibu-upgrade       31h   Completed   All clusters are compliant with all the managed policies
```

其他资源

- （可选）使用 Lifecycle Agent 和 GitOps ZTP 移到基于镜像升级的 Rollback 阶段

15.4.3. （可选）使用 Lifecycle Agent 和 GitOps ZTP 移到 Rollback 阶段

如果在升级后遇到问题，您可以启动手动回滚。

流程

1. 将 du-profile 或对应的 policy-binding 标签恢复到 SiteConfig CR 中的原始平台版本：

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
[...]
spec:
  [...]
    clusterLabels:
      du-profile: "4.14.x"
```

2. 当您准备好启动回滚时，将 Rollback 策略添加到现有组 PolicyGenTemplate CR 中：

```
[...]
```

```
- fileName: ibu/ImageBasedUpgrade.yaml
  policyName: "rollback-stage-policy"
  spec:
    stage: Rollback
  status:
    conditions:
      - message: Rollback completed
        reason: Completed
        status: "True"
        type: RollbackCompleted
```

3. 在目标 hub 集群上创建一个 ClusterGroupUpgrade CR，以引用 Rollback 策略：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-rollback
  namespace: default
spec:
  actions:
    beforeEnable:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-rollback-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

4. 运行以下命令来应用 Rollback 策略：

```
$ oc apply -f cgu-ibu-rollback.yml
```

5. 当您对更改满意且准备好完成回滚时，请在目标 hub 集群上创建一个 ClusterGroupUpgrade CR，该集群引用完成回滚的策略：

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-finalize
  namespace: default
spec:
  actions:
    beforeEnable:
```

```
    removeClusterAnnotations:
    - import.open-cluster-management.io/disable-auto-import
clusters:
- spoke1
enable: true
managedPolicies:
- example-group-ibu-finalize-stage-policy
remediationStrategy:
  canaries:
    - spoke1
  maxConcurrency: 1
  timeout: 240
```

6. 运行以下命令来应用策略：

```
$ oc apply -f cgu-ibu-finalize.yml
```

15.4.4. 使用生命周期代理对基于镜像的升级进行故障排除

在基于镜像的升级过程中可能会遇到问题。

15.4.4.1. 收集日志

您可以使用 oc adm must-gather CLI 来收集用于调试和故障排除的信息。

流程

- 运行以下命令收集有关 Operator 的数据：

```
$  oc adm must-gather \
--dest-dir=must-gather/tmp \
--image=$(oc -n openshift-lifecycle-agent get deployment.apps/lifecycle-agent-
controller-manager -o jsonpath='{.spec.template.spec.containers[?(@.name ==
"manager")].image}') \
--image=quay.io/konveyor/oadp-must-gather:latest \ 1
--image=quay.io/openshift/origin-must-gather:latest 2
```

1. （可选）如果您需要从 OADP Operator 收集更多信息，您可以添加这个选项。

2. （可选）如果您需要从 SR-IOV Operator 收集更多信息，您可以添加这个选项。

15.4.4.2. AbortFailed 或 FinalizeFailed 错误

问题

在完成阶段或停止 Prep 阶段时，生命周期代理会清理以下资源：

- Stateroot 不再需要
- 预缓存资源
- OADP CR
- ImageBasedUpgrade CR

如果 Lifecycle Agent 无法执行上述步骤，它将过渡到 AbortFailed 或 FinalizeFailed 状态。条件消息和日志显示哪些步骤失败。

错误信息示例

```
message: failed to delete all the backup CRs. Perform cleanup manually then add
'lca.openshift.io/manual-cleanup-done' annotation to ibu CR to transition back to Idle
    observedGeneration: 5
    reason: AbortFailed
    status: "False"
    type: Idle
```

解决方案

1. 检查日志，以确定发生失败的原因。
2. 要提示生命周期代理重试清理，请将 lca.openshift.io/manual-cleanup-done 注解添加到 ImageBasedUpgrade CR。

   观察此注解后，生命周期代理会重试清理，如果成功，ImageBasedUpgrade 阶段会过

渡到 Idle。

如果清理再次失败，您可以手动清理资源。

15.4.4.2.1. 手动清理 stateroot

问题

在 Prep 阶段停止，生命周期代理会清理新的 stateroot。在成功升级或回滚后最终调整时，生命周期代理会清理旧的 stateroot。如果此步骤失败，建议您检查日志以确定失败的原因。

解决方案

1. 运行以下命令，检查 stateroot 中是否有现有部署：

```
$ ostree admin status
```

2. 如果存在，请运行以下命令清理现有部署：

```
$ ostree admin undeploy <index_of_deployment>
```

3. 清理 stateroot 的所有部署后，运行以下命令来擦除 stateroot 目录：

警告

确保引导的部署不处于这个 stateroot。

```
$ stateroot="<stateroot_to_delete>"
```

```
$ unshare -m /bin/sh -c "mount -o remount,rw /sysroot && rm -rf
/sysroot/ostree/deploy/${stateroot}"
```

15.4.4.2.2. 手动清理 OADP 资源

问题

由于生命周期代理和 S3 后端之间的连接问题，自动清理 OADP 资源可能会失败。通过恢复连接并添加 lca.openshift.io/manual-cleanup-done 注解，生命周期代理可以成功清理备份资源。

解决方案

1. 运行以下命令检查后端连接：

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

输出示例

```
NAME                          PHASE       LAST VALIDATED   AGE   DEFAULT
dataprotectionapplication-1   Available   33s              8d    true
```

2. 删除所有备份资源，然后将 lca.openshift.io/manual-cleanup-done 注解添加到 ImageBasedUpgrade CR。

15.4.4.3. LVM 存储卷内容没有恢复

当使用 LVM 存储来提供动态持久性卷存储时，如果配置不正确，LVM 存储可能无法恢复持久性卷内容。

15.4.4.3.1. Backup CR 中缺少与 LVM Storage 相关的字段

问题

您的 Backup CR 可能缺少恢复持久性卷所需的字段。您可以通过运行以下命令来检查应用程序 pod 中的事件，以确定是否出现这个问题：

```
$ oc describe pod <your_app_name>
```

显示 Backup CR 中缺少 LVM Storage 相关字段的输出示例

```
Events:
  Type     Reason          Age              From               Message
  ----     ------          ----             ----               -------
```

```
  Warning  FailedScheduling  58s (x2 over 66s)  default-scheduler  0/1 nodes are available:
pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available:
1 Preemption is not helpful for scheduling..
  Normal   Scheduled         56s                default-scheduler  Successfully assigned default/db-
1234 to sno1.example.lab
  Warning  FailedMount       24s (x7 over 55s)  kubelet            MountVolume.SetUp failed for
volume "pvc-1234" : rpc error: code = Unknown desc = VolumeID is not found
```

解决方案

您必须在应用程序 Backup CR 中包含 logicalvolumes.topolvm.io。如果没有此资源，应用程序可以正确地恢复其持久性卷声明和持久性卷清单，但与这个持久性卷关联的 逻辑卷 在 pivot 后无法正确恢复。

备份 CR 示例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: small-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  includedClusterScopedResources: (1)
  - persistentVolumes
  - volumesnapshotcontents
  - logicalvolumes.topolvm.io
```

(1)

要恢复应用程序的持久性卷，您必须配置本节，如下所示。

15.4.4.3.2. Restore CR 中缺少与 LVM Storage 相关的字段

问题

应用程序的预期资源会被恢复，但升级后持久性卷内容不会被保留。

1. 在 pivot 前运行以下命令来列出应用程序的持久性卷：

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

pivot 前的输出示例

```
NAME                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Retain           Bound    default/pvc-
db   lvms-vg1              4h45m

NAMESPACE   NAME                    STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       4h45m

NAMESPACE   NAME                          AGE
          logicalvolume.topolvm.io/pvc-1234   4h45m
```

2. 在 pivot 后运行以下命令来列出应用程序的持久性卷：

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

pivot 后的输出示例

```
NAME                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Delete           Bound    default/pvc-
db   lvms-vg1              19s

NAMESPACE   NAME                    STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       19s
```

```
NAMESPACE   NAME                              AGE
            logicalvolume.topolvm.io/pvc-1234   18s
```

解决方案

造成这个问题的原因是 logicalvolume 状态没有在 Restore CR 中保留。这个状态非常重要，因为 Velero 需要引用 pivoting 后必须保留的卷。您必须在应用程序 Restore CR 中包含以下字段：

Restore CR 示例

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: sample-vote-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3"
spec:
  backupName:
    sample-vote-app
  restorePVs: true (1)
  restoreStatus: (2)
    includedResources:
      - logicalvolumes
```

1. 要保留应用程序的持久性卷，您必须将 restorePV 设置为 true。
2. 要为应用程序保留持久性卷，您必须配置本节，如下所示。

15.4.4.4. 调试失败的备份和恢复 CR

问题

工件的备份或恢复失败。

解决方案

您可以调试 Backup 和 Restore CR，并使用 Velero CLI 工具检索日志。Velero CLI 工具比 OpenShift CLI 工具提供更详细的信息。

1. 运行以下命令描述包含错误的 Backup CR：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe backup -n openshift-adp backup-acm-klusterlet --details
```

2. 运行以下命令描述包含错误的 Restore CR：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe restore -n openshift-adp restore-acm-klusterlet --details
```

3. 运行以下命令，将备份的资源下载到本地目录：

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero backup download -n openshift-adp backup-acm-klusterlet -o ~/backup-acm-klusterlet.tar.gz
```