Red Hat build of MicroShift 4.16

运行应用程序

在 MicroShift 中运行应用程序

Last Updated: 2024-07-02

Red Hat build of MicroShift 4.16 运行应用程序

在 MicroShift 中运行应用程序

法律通告

摘要

本文档提供了有关如何在 MicroShift 中运行应用程序的详细信息。

目录

第 1 章 使用 KUSTOMIZE 清单部署应用程序

您可以将 **kustomize** 配置管理工具与应用程序清单一起使用来部署应用程序。阅读以下流程，以了解 Kustomize 在 MicroShift 中的工作方式。

1.1. KUSTOMIZE 如何与清单一起工作来部署应用程序

kustomize 配置管理工具与 MicroShift 集成。您可以使用 Kustomize 和 OpenShift CLI (**oc**)将自定义应用到应用程序清单，并将这些应用程序部署到 MicroShift 集群。

- **kustomization.yaml** 文件是资源及自定义的规格。
- Kustomize 使用 **kustomization.yaml** 文件加载资源，如应用程序，然后应用您想要的应用程序清单的任何更改，并生成具有 overlaid 更改的清单副本。
- 使用带有覆盖的清单副本会使应用程序的原始配置文件保持不变，同时允许您有效地部署应用程序的迭代和自定义。
- 然后，您可以使用 **oc** 命令在 MicroShift 集群中部署应用程序。

1.1.1. MicroShift 如何使用清单

在每次开始时，MicroShift 会在以下清单目录中搜索 Kustomize 清单文件：

- **/etc/microshift/manifests**
- **/etc/microshift/manifests.d/***
- **/usr/lib/microshift/**
- **/usr/lib/microshift/manifests.d/***

MicroShift 会自动运行与 **kubectl apply -k** 命令等效的命令，以便在搜索目录中存在以下文件类型时将清单应用到集群：

- **kustomization.yaml**
- **kustomization.yml**
- **kustomization**

这从多个目录自动加载意味着您可以管理 MicroShift 工作负载，使不同工作负载可以独立运行。

表 1.1. MicroShift 清单目录

位置	作用
/etc/microshift/manifests	用于配置管理系统或开发的读写位置。
/etc/microshift/manifests.d/*	用于配置管理系统或开发的读写位置。
/usr/lib/microshift/manifests	在基于 OSTree 的系统上嵌入配置清单的只读位置。
/usr/lib/microshift/manifestsd./*	在基于 OSTree 的系统上嵌入配置清单的只读位置。

1.2. 覆盖清单路径列表

您可以使用新的单一路径或将新的 glob 模式用于多个文件来覆盖默认清单路径列表。使用以下步骤自定义清单路径。

流程

1. 通过插入您自己的值并运行以下命令之一来覆盖默认路径列表：

   a. 将配置文件中的 **manifests.kustomizePaths** 设置为 < **"/opt/alternate/path** ">。

   b. 在用于 glob 模式的配置文件中将 **kustomizePaths** 设置为 **,"/opt/alternative/path.d** mdadm"。

   ```
   manifests:
     kustomizePaths:
       - <location> 1
   ```

   1 使用 **"/opt/alternate/path"** 或 glob 模式将每个位置条目设置为准确路径，使用 **"/opt/alternative/path.d8:0:1::"**。

2. 要禁用加载清单，请将配置选项设置为空列表。

   ```
   manifests:
     kustomizePaths: []
   ```

注意

配置文件完全会覆盖默认值。如果设置了 **kustomizePaths** 值，则只使用配置文件中的值。将值设为空列表可禁用清单加载。

1.3. 使用清单示例

本例演示了使用 **/etc/microshift/manifests** 目录中的 **kustomize** 清单自动部署 BusyBox 容器。

流程

1. 运行以下命令来创建 BusyBox 清单文件：

   a. 定义目录位置：

   ```
   $ MANIFEST_DIR=/etc/microshift/manifests
   ```

   b. 创建目录：

   ```
   $ sudo mkdir -p ${MANIFEST_DIR}
   ```

   c. 将 YAML 文件放在目录中：

   ```
   sudo tee ${MANIFEST_DIR}/busybox.yaml &>/dev/null <<EOF
   apiVersion: v1
   kind: Namespace
   ```

```
metadata:
  name: busybox
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: busybox-deployment
spec:
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: BUSYBOX_IMAGE
        command: [ "/bin/sh", "-c", "while true ; do date; sleep 3600; done;" ]
EOF
```

2. 接下来，运行以下命令来创建 **kustomize** 清单文件：

   a. 将 YAML 文件放在目录中：

```
sudo tee ${MANIFEST_DIR}/kustomization.yaml &>/dev/null <<EOF
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: busybox
resources:
  - busybox.yaml
images:
  - name: BUSYBOX_IMAGE
    newName: busybox:1.35
EOF
```

3. 运行以下命令重启 MicroShift 以应用清单：

```
$ sudo systemctl restart microshift
```

4. 运行以下命令应用清单并启动 **busybox** pod：

```
$ oc get pods -n busybox
```

第 2 章 在 RHEL FOR EDGE 镜像中嵌入 MICROSHIFT 应用程序的选项

您可以在 Red Hat Enterprise Linux for Edge (RHEL for Edge)镜像中嵌入基于微服务的工作负载和应用程序，以便在 MicroShift 集群中运行。嵌入式应用程序可以直接安装在边缘设备上，以便在 air-gapped、断开连接或离线环境中运行。

2.1. 将应用程序 RPM 添加到 RPM-OSTREE 镜像

如果您的应用程序包含 API、容器镜像和配置文件，如清单等部署，您可以构建应用程序 RPM。然后，您可以在 RHEL for Edge 系统镜像中添加 RPM。

以下是将应用程序或工作负载嵌入到完全自包含的操作系统镜像中的流程：

- 构建包含应用程序清单的您自己的 RPM。
- 将 RPM 添加到用于安装 MicroShift 的红帽构建的蓝图中。
- 将工作负载容器镜像添加到同一蓝图中。
- 创建可引导 ISO。

有关在 RHEL for Edge 镜像中准备和嵌入应用程序的逐步教程，请使用以下教程：

- 嵌入应用程序教程

2.2. 在镜像中添加应用程序清单以离线使用

如果您有一个简单的应用程序，其中包含几个用于部署的文件，如清单，您可以将这些清单直接添加到 RHEL for Edge 系统镜像中。

有关示例，请参阅以下 RHEL for Edge 文档中的"创建自定义文件蓝图自定义"部分：

- 创建一个自定义文件蓝图自定义

2.3. 嵌入应用程序供离线使用

如果您的应用程序包含多个文件，您可以嵌入应用程序以离线使用。请参见以下步骤：

- 嵌入应用程序供离线使用

2.4. 其他资源

- 将红帽构建的 MicroShift 嵌入到 RPM-OSTree 镜像中
- 准备、安装和管理 RHEL for Edge 镜像
- 准备镜像构建
- 满足 Red Hat Device Edge
- 使用镜像构建器命令行制作 RHEL for Edge 镜像
- 镜像构建器系统要求

第 3 章 嵌入应用程序供离线使用

您可以在 Red Hat Enterprise Linux for Edge (RHEL for Edge)镜像中嵌入基于微服务的工作负载和应用程序。嵌入意味着您可以在 air-gapped、断开连接或离线环境中运行 MicroShift 集群的红帽构建。

3.1. 嵌入工作负载容器镜像供离线使用

要在没有网络连接的边缘将容器镜像嵌入到设备中，您必须创建新容器，挂载 ISO，然后将内容复制到文件系统中。

先决条件

- 有对主机的 root 访问权限。
- 应用程序 RPM 已添加到蓝图中。

流程

1. 呈现清单，提取所有容器镜像引用，并通过运行以下命令来将应用程序镜像转换为蓝图容器源：

```
$ oc kustomize ~/manifests | grep "image:" | grep -oE '[^ ]+$' | while read line; do echo -e "
[[containers]]\nsource = \"${line}\"\n"; done >><my_blueprint>.toml
```

2. 运行以下命令，将更新的蓝图推送到镜像构建器：

```
$ sudo composer-cli blueprints push <my_blueprint>.toml
```

3. 如果您的工作负载容器位于私有存储库中，则必须为镜像构建器提供所需的 pull secret：

   a. 在 **/etc/osbuild-worker/osbuild-worker.toml** 文件中的 **osbuilder worker** 配置的 **[containers]** 部分中设置 **auth_file_path**，以指向 pull secret。

   b. 如果需要，为 pull secret 创建目录和文件，例如：

   目录和文件示例

```
[containers]
auth_file_path = "/<path>/pull-secret.json" 1
```

   1 使用之前设置的自定义位置来复制和检索镜像。

4. 运行以下命令来构建容器镜像：

```
$ sudo composer-cli compose start-ostree <my_blueprint> edge-commit
```

5. 继续您首选的 **rpm-ostree** 镜像流，如等待构建完成，导出镜像并将其集成到 **rpm-ostree** 存储库或创建可引导 ISO。

3.2. 其他资源

- 在 RHEL for Edge 镜像中嵌入红帽构建的 MicroShift 应用程序的选项

- 创建 RHEL for Edge 镜像
- 将蓝图添加到镜像构建器并构建 ISO
- 下载 ISO 并为使用做准备
- 升级 RHEL for Edge 系统

第 4 章 嵌入红帽构建的 MICROSHIFT 应用程序教程

以下教程提供了如何在 RHEL for Edge 镜像中嵌入应用程序以便在各种环境中在 MicroShift 集群中使用的详细示例。

4.1. 嵌入应用程序 RPM 指南

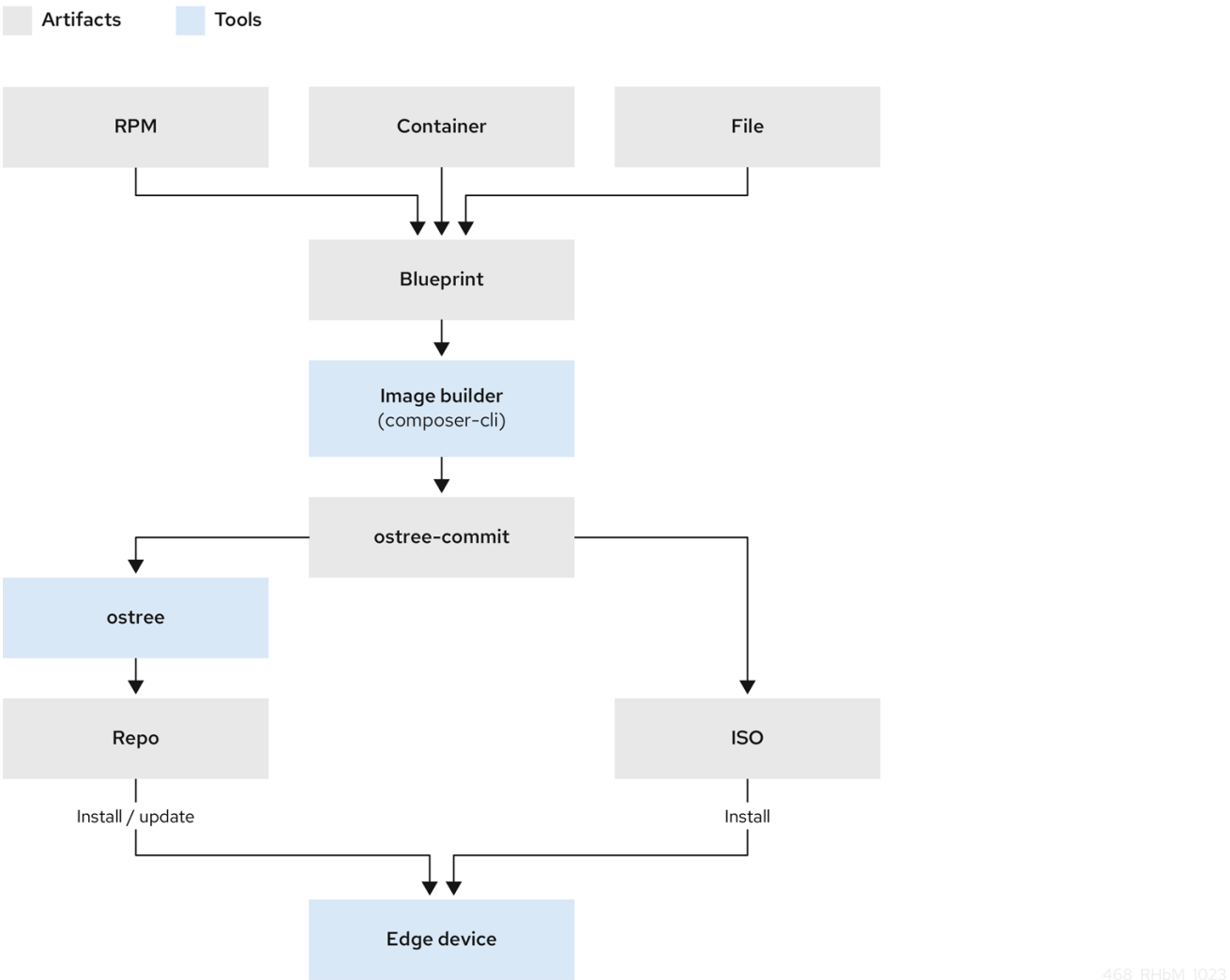
以下教程回顾 MicroShift 安装步骤，并添加用于嵌入应用程序的工作流的描述。如果您已经熟悉 **rpm-ostree** 系统，如 Red Hat Enterprise Linux for Edge (RHEL for Edge)和 MicroShift，您可以直接进入相关的操作。

4.1.1. 安装工作流查看

嵌入的应用程序需要类似的工作流将 MicroShift 嵌入到 RHEL for Edge 镜像中。

- 下图显示了系统工件（如 RPM、容器和文件）如何添加到蓝图中，并由镜像 composer 创建 ostree 提交。
- 然后，ostree 提交可以遵循 ISO 路径或边缘设备的存储库路径。
- ISO 路径可用于断开连接的环境，而存储库路径通常用于网络通常连接。

嵌入 MicroShift 工作流



查看这些步骤可帮助您了解嵌入应用程序所需的步骤：

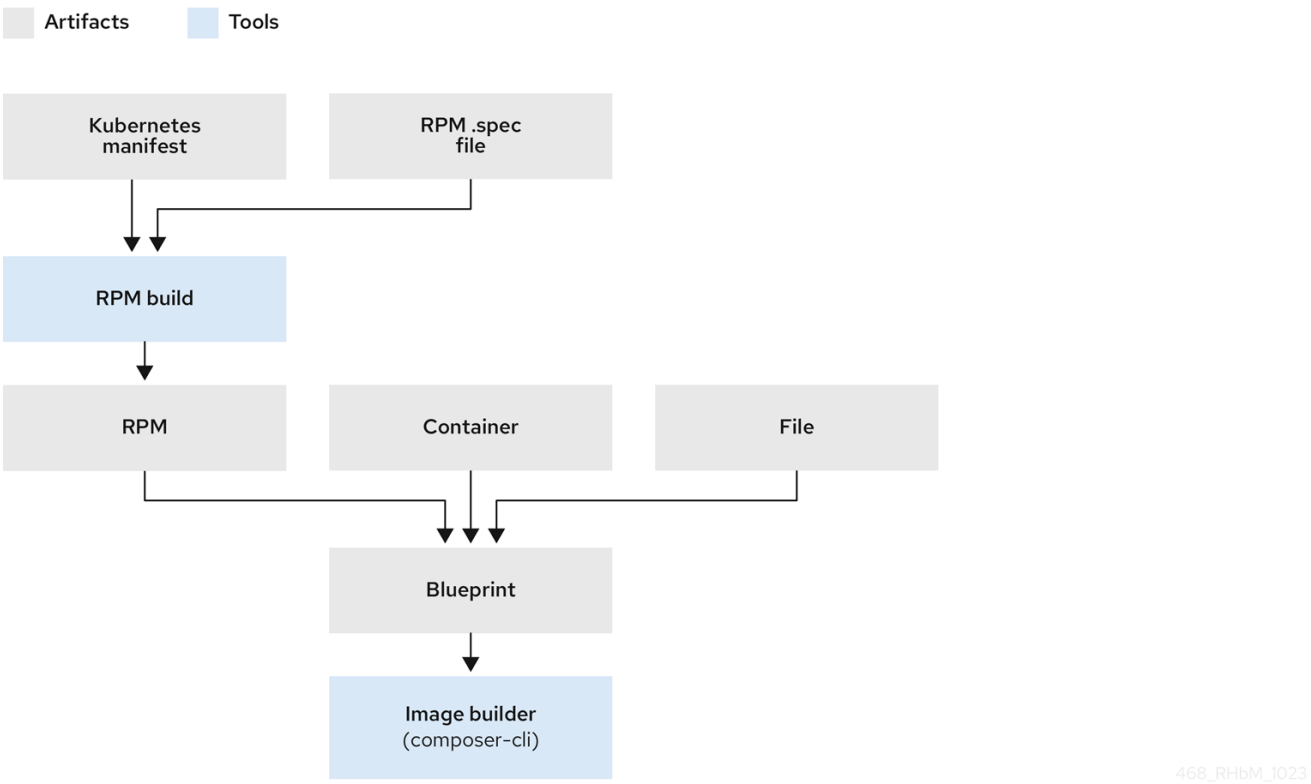
1. 要在 RHEL for Edge 上嵌入 MicroShift，您需要将 MicroShift 存储库添加到 Image Builder 中。
2. 您创建了声明您所需的所有 RPM、容器镜像、文件和自定义的蓝图，包括添加 MicroShift。
3. 您已将蓝图添加到镜像构建器，并使用 Image Builder CLI 工具(**composer-cli**)运行构建。此步骤创建 **rpm-ostree** 提交，用于创建容器镜像。此镜像包含 RHEL for Edge。
4. 将安装程序蓝图添加到镜像构建器中，以创建 **rpm-ostree** 镜像(ISO)来从其引导。此构建包含 RHEL for Edge 和 MicroShift。
5. 下载了使用 MicroShift 嵌入的 ISO，准备好使用、置备它，然后将其安装到边缘设备中。

4.1.2. 嵌入应用程序 RPM 工作流

设置满足 Image Builder 要求的构建主机后，您可以将应用程序以清单目录的形式添加到镜像。这些步骤后，将应用程序或工作负载嵌入到新 ISO 中最简单的方法是创建自己的包含清单的 RPM。您的应用程序 RPM 包含描述部署的所有配置文件。

以下"嵌入的应用程序工作流"镜像演示了如何将 Kubernetes 应用程序清单和 RPM 规格文件合并到单个应用程序 RPM 构建中。此构建成为工作流中包含的 RPM 工件，用于将 MicroShift 嵌入到 ostree 提交中。

嵌入应用程序工作流



以下流程使用 **rpmbuild** 工具创建规格文件和本地存储库。规范文件定义了如何构建软件包，将应用程序清单移到 MicroShift 的 RPM 软件包中的正确位置，以便获取它们。然后，该 RPM 软件包被嵌入到 ISO 中。

4.1.3. 准备进行应用程序 RPM

要构建自己的 RPM，请选择您选择的工具，如 **rpmbuild** 工具，并在主目录中初始化 RPM 构建树。以下是示例步骤。只要您的 RPM 可以被镜像构建器访问，就可以使用您喜欢构建应用程序 RPM 的方法。

先决条件

- 您已设置了满足 Image Builder 系统要求的 Red Hat Enterprise Linux for Edge (RHEL for Edge) 9.4 构建主机。
- 有对主机的 root 访问权限。

流程

1. 运行以下命令，安装 **rpmbuild** 工具并为其创建 yum 存储库：

```
$ sudo dnf install rpmdevtools rpmlint yum-utils createrepo
```

2. 运行以下命令，创建构建 RPM 软件包所需的文件树：

```
$ rpmdev-setuptree
```

验证

- 运行以下命令列出确认创建的目录：

```
$ ls ~/rpmbuild/
```

输出示例

```
BUILD RPMS SOURCES SPECS SRPMS
```

4.1.4. 为应用程序清单构建 RPM 软件包

要构建自己的 RPM，您必须创建一个 spec 文件，将应用程序清单添加到 RPM 软件包中。以下是示例步骤。只要应用程序 RPM 和其他镜像构建元素可供镜像构建器访问，您可以使用您喜欢的方法。

先决条件

- 您已设置了满足 Image Builder 系统要求的 Red Hat Enterprise Linux for Edge (RHEL for Edge) 9.4 构建主机。
- 有对主机的 root 访问权限。
- 构建 RPM 软件包所需的文件树已创建。

流程

1. 在 **~/rpmbuild/SPECS** 目录中，使用以下模板创建一个文件，如 & **lt;application_workload_manifests.spec** > ：

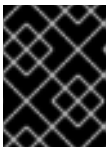
spec 文件示例

```
Name: <application_workload_manifests>
Version: 0.0.1
```

```
Release: 1%{?dist}
Summary: Adds workload manifests to microshift
BuildArch: noarch
License: GPL
Source0: %{name}-%{version}.tar.gz
#Requires: microshift
%description
Adds workload manifests to microshift
%prep
%autosetup
%install (1)
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%{_prefix}/lib/microshift/manifests
cp -pr ~/manifests $RPM_BUILD_ROOT/%{_prefix}/lib/microshift/
%clean
rm -rf $RPM_BUILD_ROOT

%files
%{_prefix}/lib/microshift/manifests/**
%changelog
* <DDD MM DD YYYY username@domain - V major.minor.patch>
- <your_change_log_comment>
```

1. **%install** 部分在 RPM 软件包 **/usr/lib/microshift/manifests/** 中创建目标目录，并从源主目录 **~/manifests** 复制清单。



重要

所有必需的 YAML 文件都必须位于源主目录 **~/manifests** 中，如果您使用的是 kustomize，则包括 **kustomize.yaml** 文件。

2. 运行以下命令，在 **~/rpmbuild/RPMS** 目录中构建 RPM 软件包：

```
$ rpmbuild -bb ~/rpmbuild/SPECS/<application_workload_manifests.spec>
```

4.1.5. 在蓝图中添加应用程序 RPM

要将应用程序 RPM 添加到蓝图中，您必须创建一个本地仓库，供 Image Builder 用于创建 ISO。使用这个流程，您的工作负载所需的容器镜像可以通过网络拉取。

先决条件

- 有对主机的 root 访问权限。
- 工作负载或应用程序 RPM 存在于 **~/rpmbuild/RPMS** 目录中。

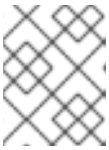
流程

1. 运行以下命令来创建本地 RPM 存储库：

```
$ createrepo ~/rpmbuild/RPMS/
```

2. 运行以下命令，为 Image Builder 授予对 RPM 存储库的访问权限：

```
$ sudo chmod a+rx ~
```



注意

您必须确保 Image Builder 具有访问镜像构建所需的所有文件所需的权限，或者构建无法继续。

3. 使用以下模板创建蓝图文件 **repo-local-rpmbuild.toml** ：

```
id = "local-rpm-build"
name = "RPMs build locally"
type = "yum-baseurl"
url = "file://<path>/rpmbuild/RPMS" (1)
check_gpg = false
check_ssl = false
system = false
```

(1) 指定创建您选择的位置的路径部分。在后续命令中使用此路径来设置存储库并复制 RPM。

4. 运行以下命令，将存储库添加为镜像构建器的源：

```
$ sudo composer-cli sources add repo-local-rpmbuild.toml
```

5. 通过添加以下几行，将 RPM 添加到蓝图中：

```
…
[[packages]]
name = "<application_workload_manifests>" (1)
version = "*"
…
```

(1) 在此处添加工作负载的名称。

6. 运行以下命令，将更新的蓝图推送到镜像构建器：

```
$ sudo composer-cli blueprints push repo-local-rpmbuild.toml
```

7. 此时，您可以运行 Image Builder 来创建 ISO，或嵌入容器镜像供离线使用。

   a. 要创建 ISO，请运行以下命令启动镜像构建器：

```
$ sudo composer-cli compose start-ostree repo-local-rpmbuild edge-commit
```

在这种情况下，容器镜像在启动时由边缘设备通过网络拉取。

其他资源

- 使用 Image Builder CLI 编写 RHEL for Edge 镜像

- 基于网络的部署工作流

4.2. 其他资源

- 嵌入应用程序供离线使用
- 将红帽构建的 MicroShift 嵌入到 RPM-OSTree 镜像中
- 准备、安装和管理 RHEL for Edge 镜像
- 准备镜像构建
- 使用红帽构建的 MicroShift 满足 Red Hat Device Edge
- 如何创建 Linux RPM 软件包
- 使用镜像构建器命令行制作 RHEL for Edge 镜像
- 镜像构建器系统要求

第 5 章 GREENBOOT 工作负载健康检查脚本

Greenboot 健康检查脚本在边缘设备（这些设备的直接服务可用性可能会非常有限或不存在）上很有用。您可以创建健康检查脚本来评估工作负载和应用程序的健康状态。这些额外的健康检查脚本是软件问题检查和自动系统回滚的有用组件。

MicroShift 健康检查脚本包含在 **microshift-greenboot** RPM 中。您还可以根据您正在运行的工作负载创建自己的健康检查脚本。例如，您可以编写一个来验证服务是否已启动。

5.1. 工作负载健康检查脚本的工作方式

本教程中描述的工作负载或应用程序健康检查脚本使用 **/usr/share/microshift/functions/greenboot.sh** 文件中可用的 MicroShift 健康检查功能。这可让您重复使用已经为 MicroShift 核心服务实施的流程。

脚本首先运行检查工作负载的基本功能是否如预期运行。要成功运行脚本：

- 从 root 用户帐户执行脚本。
- 启用 MicroShift 服务。

健康检查执行以下操作：

- 获取 **wait_for** 函数的当前引导周期的等待超时。
- 调用 **namespace_images_downloaded** 功能，以等待 pod 镜像可用。
- 调用 **namespace_pods_ready** 函数，以等待 pod 就绪。
- 调用 **namespace_pods_not_restarting** 功能来验证 pod 是否没有重启。

注意

重启 pod 可以表示崩溃循环。

5.2. 包括 GREENBOOT 健康检查

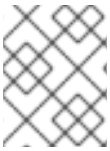
健康检查脚本在 **/usr/lib/greenboot/check** 中提供，这是 RPM-OSTree 系统的只读目录。以下健康检查包含在 **greenboot-default-health-checks** 框架中。

- 检查存储库 URL 仍然是 DNS 解析：
  此脚本位于 **/usr/lib/greenboot/check/required.d/01_repository_dns_check.sh** 下，并确保对存储库 URL 的 DNS 查询仍然可用。
- 检查更新平台是否仍然可访问：
  此脚本位于 **/usr/lib/greenboot/check/wanted.d/01_update_platform_check.sh**，并从 **/etc/ostree/remotes.d** 中定义的更新平台连接并获取 2XX 或 3XX HTTP 代码。
- 检查硬件 watchdog 是否触发当前的引导：
  此脚本位于 **/usr/lib/greenboot/check/required.d/02_watchdog.sh** 下，并检查当前引导是否已是 watchdog-triggered。
  - 如果在宽限期内发生 watchdog-triggered 重启，则当前引导将标记为红色。Greenboot 不会触发对上一个部署的回滚。

- 如果 watchdog-triggered 重启在宽限期后发生，则当前引导不会标记为红色。Greenboot 不会触发对上一个部署的回滚。
- 默认启用 24 小时宽限期。可以通过以下方法禁用宽限（grace）期：修改 **/etc/greenboot/greenboot.conf to false** 中的 **GREENBOOT_WATCHDOG_CHECK_ENABLED**；或通过 **/etc/greenboot/greenboot.conf** 中的 **GREENBOOT_WATCHDOG_GRACE_PERIOD=number_of_hours** 变量值进行配置。

5.3. 如何为应用程序创建健康检查脚本

您可以使用本文档中的示例在您选择的文本编辑器中创建工作负载或应用程序健康检查脚本。将脚本保存到 **/etc/greenboot/check/required.d** 目录中。当 **/etc/greenboot/check/required.d** 目录中的脚本退出并显示错误时，Greenboot 会在尝试修复系统时触发重启。



注意

如果 **/etc/greenboot/check/required.d** 目录退出，则 /etc/greenboot/check/required.d 目录中的所有脚本都会触发重启。

如果您的健康检查逻辑需要任何 post-check 步骤，您也可以创建额外的脚本并将其保存在相关的 greenboot 目录中。例如：

- 您还可以在 **/etc/greenboot/green.d** 中声明成功后要运行的 shell 脚本。
- 您可以在 **/etc/greenboot/red.d** 声明引导失败后要运行的 shell 脚本。例如，如果您在重启前有修复系统的步骤，您可以为您的用例创建脚本并将其放在 **/etc/greenboot/red.d** 目录中。

5.3.1. 关于工作负载健康检查脚本示例

以下示例使用 MicroShift 健康检查脚本作为模板。您可以将这个示例与提供的库一起使用，作为为应用程序创建基本健康检查脚本的指南。

5.3.1.1. 创建健康检查脚本的基本先决条件

- 必须安装工作负载。
- 您必须有 root 访问权限。

5.3.1.2. 功能要求示例

您可以从以下示例健康检查脚本开始。根据您的用例进行修改。在工作负载健康检查脚本中，您必须完成以下最小步骤：

- 设置环境变量。
- 定义用户工作负载命名空间。
- 列出预期的 pod 数量。

重要

为您的应用程序选择一个名称前缀，以确保它在 **40_microshift_running_check.sh** 脚本后运行，该脚本为核心服务实施红帽构建的 MicroShift 健康检查流程。

工作负载健康检查脚本示例

```
# #!/bin/bash
set -e

SCRIPT_NAME=$(basename $0)
PODS_NS_LIST=(<user_workload_namespace1> <user_workload_namespace2>)
PODS_CT_LIST=(<user_workload_namespace1_pod_count>
<user_workload_namespace2_pod_count>)
# Update these two lines with at least one namespace and the pod counts that are specific to your
workloads. Use the kubernetes <namespace> where your workload is deployed.

# Set Greenboot to read and execute the workload health check functions library.
source /usr/share/microshift/functions/greenboot.sh

# Set the exit handler to log the exit status.
trap 'script_exit' EXIT

# Set the script exit handler to log a `FAILURE` or `FINISHED` message depending on the exit status
of the last command.
# args: None
# return: None
function script_exit() {
    [ "$?" -ne 0 ] && status=FAILURE || status=FINISHED
    echo $status
}

# Set the system to automatically stop the script if the user running it is not 'root'.
if [ $(id -u) -ne 0 ] ; then
    echo "The '${SCRIPT_NAME}' script must be run with the 'root' user privileges"
    exit 1
fi

echo "STARTED"

# Set the script to stop without reporting an error if the MicroShift service is not running.
if [ $(systemctl is-enabled microshift.service 2>/dev/null) != "enabled" ] ; then
    echo "MicroShift service is not enabled. Exiting..."
    exit 0
fi

# Set the wait timeout for the current check based on the boot counter.
WAIT_TIMEOUT_SECS=$(get_wait_timeout)

# Set the script to wait for the pod images to be downloaded.
for i in ${!PODS_NS_LIST[@]}; do
    CHECK_PODS_NS=${PODS_NS_LIST[$i]}

    echo "Waiting ${WAIT_TIMEOUT_SECS}s for pod image(s) from the ${CHECK_PODS_NS}
namespace to be downloaded"
    wait_for ${WAIT_TIMEOUT_SECS} namespace_images_downloaded
done

# Set the script to wait for pods to enter ready state.
for i in ${!PODS_NS_LIST[@]}; do
    CHECK_PODS_NS=${PODS_NS_LIST[$i]}
```

```
    CHECK_PODS_CT=${PODS_CT_LIST[$i]}

    echo "Waiting ${WAIT_TIMEOUT_SECS}s for ${CHECK_PODS_CT} pod(s) from the
${CHECK_PODS_NS} namespace to be in 'Ready' state"
    wait_for ${WAIT_TIMEOUT_SECS} namespace_pods_ready
done

# Verify that pods are not restarting by running, which could indicate a crash loop.
for i in ${!PODS_NS_LIST[@]}; do
    CHECK_PODS_NS=${PODS_NS_LIST[$i]}

    echo "Checking pod restart count in the ${CHECK_PODS_NS} namespace"
    namespace_pods_not_restarting ${CHECK_PODS_NS}
done
```

5.4. 测试工作负载健康检查脚本

先决条件

- 有 root 访问权限。
- 已安装工作负载。
- 您已为工作负载创建了健康检查脚本。
- 启用 MicroShift 服务的红帽构建。

流程

1. 要测试 Greenboot 是否在运行健康检查脚本文件，请运行以下命令重启主机：

```
$ sudo reboot
```

2. 运行以下命令，检查 Greenboot 健康检查的输出：

```
$ sudo journalctl -o cat -u greenboot-healthcheck.service
```

注意

MicroShift 核心服务健康检查在工作负载健康检查前运行。

输出示例

```
GRUB boot variables:
boot_success=0
boot_indeterminate=0
Greenboot variables:
GREENBOOT_WATCHDOG_CHECK_ENABLED=true
...
...
FINISHED
```

```
Script '40_microshift_running_check.sh' SUCCESS
Running Wanted Health Check Scripts...
Finished greenboot Health Checks Runner.
```

5.5. 其他资源

- Greenboot 健康检查
- 自动应用清单

第 6 章 使用 GITOPS 控制器自动化应用程序管理

使用 Argo CD for MicroShift 的 GitOps 是一个轻量级、可选的附加控制器，它派生自 Red Hat OpenShift GitOps Operator。MicroShift 的 GitOps 使用 Argo CD 的命令行界面(CLI)与作为声明性 GitOps 引擎的 GitOps 控制器交互。您可以在集群和开发生命周期中一致配置和部署基于 Kubernetes 的基础架构和应用程序。

6.1. GITOPS 代理可以做什么

通过将 GitOps 与 Argo CD 代理与 MicroShift 搭配使用，您可以使用以下原则：

- 实施应用程序生命周期管理。
  - 使用在 Git 存储库中开发和维护软件的核心原则创建和管理集群和应用程序配置文件。
  - 您可以更新单个存储库，GitOps 会自动将新应用程序或更新部署到现有存储库。
  - 例如，如果您有 1,000 个边缘设备，每个设备都使用 MicroShift 和本地 GitOps 代理，您可以在所有 1,000 设备上轻松添加或更新应用程序，且只在中央 Git 存储库中有一个变化。
- Git 存储库包含指定环境中所需的基础架构声明描述，并包含自动化流程，使您的环境与上述状态匹配。
- 您还可以使用 Git 存储库作为更改的审计跟踪，以便您可以根据 Git 流创建进程，如检查和批准来合并实施配置更改的拉取请求。

6.2. 在 MICROSHIFT 上创建 GITOPS 应用程序

您可以创建自定义 YAML 配置来部署和管理 MicroShift 服务中的应用程序。要安装运行 GitOps 应用程序所需的软件包，请参阅"从 RPM 软件包安装 GitOps Argo CD 清单"中的文档。

先决条件

- 已安装 **microshift-gitops** 软件包。
- Argo CD pod 在 **openshift-gitops** 命名空间中运行。

流程

1. 创建 YAML 文件并为应用程序添加自定义配置：

   spring-petclinic 应用程序的 YAML 示例

```
kind: AppProject
apiVersion: argoproj.io/v1alpha1
metadata:
  name: default
  namespace: openshift-gitops
spec:
  clusterResourceWhitelist:
  - group: '*'
    kind: '*'
  destinations:
  - namespace: '*'
```

```
    server: '*'
  sourceRepos:
  - '*'
---
kind: Application
apiVersion: argoproj.io/v1alpha1
metadata:
  name: spring-petclinic
  namespace: openshift-gitops
spec:
  destination:
    namespace: spring-petclinic
    server: https://kubernetes.default.svc
  project: default
  source:
    directory:
      recurse: true
    path: app
    repoURL: https://github.com/siamaksade/openshift-gitops-getting-started
  syncPolicy:
    automated: {}
    syncOptions:
    - CreateNamespace=true
    - ServerSideApply=true
```

2. 要部署 YAML 文件中定义的应用程序，请运行以下命令：

```
$ oc apply -f <my-app>.yaml 1
```

1 将 *<my-app>* 替换为应用程序 YAML 的名称。

验证

- 要验证应用程序是否已部署并同步，请运行以下命令：

```
$ oc get applications -A
```

应用程序可能需要几分钟时间才能显示 **Healthy** 状态。

输出示例

```
NAMESPACE          NAME              SYNC STATUS   HEALTH STATUS
openshift-gitops   spring-petclinic  Synced        Healthy
```

其他资源

- 从 RPM 软件包安装 GitOps Argo CD 清单

6.3. 在 MICROSHIFT 中使用 GITOPS 代理的限制

MicroShift 的 Argo CD 的 GitOps 与 Red Hat OpenShift GitOps Operator 有以下区别：

- **gitops-operator** 组件不用于 MicroShift。

- 为了维护 MicroShift 的小资源使用，Argo CD web 控制台不可用。您可以使用 Argo CD CLI。
- 因为 MicroShift 是单节点，所以不支持多集群。MicroShift 的每个实例都与本地 GitOps 代理配对。
- **oc adm must-gather** 命令在 MicroShift 中不可用。

6.4. GITOPS 故障排除

如果 GitOps 控制器出现问题，您可以使用 OpenShift CLI (**oc**)工具。

6.4.1. 使用 oc adm inspect 调试 GitOps

您可以使用 OpenShift CLI (**oc**)调试 GitOps。

先决条件

- 已安装 **oc** 命令行工具。

流程

1. 在 GitOps 命名空间中运行 **oc adm inspect** 命令：

```
$ oc adm inspect ns/openshift-gitops
```

输出示例

```
Gathering data for ns/openshift-gitops...
W0501 20:34:35.978508 57625 util.go:118] the server doesn't have a resource type
egressfirewalls, skipping the inspection
W0501 20:34:35.980881 57625 util.go:118] the server doesn't have a resource type
egressqoses, skipping the inspection
W0501 20:34:36.040664 57625 util.go:118] the server doesn't have a resource type
servicemonitors, skipping the inspection
Wrote inspect data to inspect.local.2673575938140296280.
```

后续步骤

- 如果 **oc adm inspect** 没有提供您需要的信息，您可以运行 sos report。

6.5. 其他资源

- 从 RPM 软件包安装 GitOps Argo CD 清单
- 使用 sos 报告
- Red Hat OpenShift GitOps
- 为技术支持生成 sos 报告 (Red Hat Enterprise Linux)

第 7 章 POD 安全身份验证和授权

7.1. 了解并管理 POD 安全准入

Pod 安全准入是 Kubernetes pod 安全标准的实现。使用 pod 安全准入来限制 pod 的行为。

7.2. 安全性上下文约束与 POD 安全标准同步

MicroShift 包括 Kubernetes pod 安全准入。

除了全局 pod 安全准入控制配置外，还存在一个控制器，它根据给定命名空间中的服务帐户的安全上下文约束(SCC)权限将 pod 安全准入控制 **warn** 和 **audit** 标签应用到命名空间。

重要

定义为集群有效负载一部分的命名空间会永久禁用 pod 安全准入同步。您可以根据需要，在其他命名空间中启用 pod 安全准入同步。如果 Operator 安装在用户创建的 **openshift-*** 命名空间中，则在命名空间中创建集群服务版本(CSV)后，默认会开启同步。

控制器检查 **ServiceAccount** 对象权限，以便在每个命名空间中使用安全性上下文约束。安全性上下文约束 (SCC) 根据其字段值映射到 Pod 安全配置集，控制器使用这些翻译配置集。Pod 安全准入 **warn** 和 **audit** 标签被设置为命名空间中找到的最特权 pod 安全配置集，以防止在创建 pod 时出现警告和审计日志记录。

命名空间标签基于对命名空间本地服务帐户权限的考虑。

直接应用 pod 可能会使用运行 Pod 的用户的 SCC 特权。但是，在自动标记过程中不会考虑用户权限。

7.2.1. 查看命名空间中的安全性上下文约束

您可以查看给定命名空间中的安全性上下文约束(SCC)权限。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

- 要查看命名空间中的安全性上下文约束，请运行以下命令：

```
oc get --show-labels namespace <namespace>
```

7.3. 控制 POD 安全准入同步

您可以为大多数命名空间启用自动 pod 安全准入同步。

当 **security.openshift.io/scc.podSecurityLabelSync** 字段为空或设置为 **false** 时，系统默认值不会被强制使用。要进行同步，您必须将标签设置为 **true**。

重要

定义为集群有效负载一部分的命名空间会永久禁用 pod 安全准入同步。这些命名空间包括：

- **default**
- **kube-node-lease**
- **kube-system**
- **kube-public**
- **openshift**
- 所有带有 **openshift-** 前缀的系统创建命名空间，除了 **openshift-operators** 默认，所有具有 **openshift-** 前缀的命名空间都不会被同步。您可以为任何用户创建的 **openshift-*** 命名空间启用同步。除了 **openshift-operators** 之外，您无法为任何系统创建的 **openshift-*** 命名空间启用同步。

如果 Operator 安装在用户创建的 **openshift-*** 命名空间中，则在命名空间中创建集群服务版本(CSV)后，默认会开启同步。同步标签继承命名空间中服务帐户的权限。

流程

- 要在命名空间中启用 pod 安全准入标签同步，请将 **security.openshift.io/scc.podSecurityLabelSync** 标签的值设置为 **true**。
  运行以下命令:

```
$ oc label namespace <namespace> security.openshift.io/scc.podSecurityLabelSync=true
```

注意

您可以使用 --overwrite 标志在命名空间中反向 pod 安全标签同步的影响。

第 8 章 OPERATOR

8.1. 在 MICROSHIFT 中使用 OPERATOR

您可以将 Operator 与 MicroShift 搭配使用，以创建用于监控集群中运行的服务的应用程序。Operator 可以管理应用程序及其资源，如部署数据库或消息总线。作为在集群中运行的自定义软件，可以使用 Operator 来实现和自动化常见操作。

Operator 提供了更本地化的配置体验，并与 Kubernetes API 和 CLI 工具（如 **kubectl** 和 **oc** ）集成。Operator 是专为您的应用程序而设计的。Operator 允许您配置组件而不是修改全局配置文件。

MicroShift 应用程序通常预期部署在静态环境中。但是，如果在您的用例中很有用，Operator 就会可用。要确定 Operator 与 MicroShift 的 Operator 兼容性，请查看 Operator 文档。

8.1.1. 如何将 Operator 与 MicroShift 集群搭配使用

将 Operator 用于 MicroShift 集群的方法有两种：

8.1.1.1. Operator 的清单

可以使用清单直接安装和管理 Operator。您可以在 MicroShift 中使用 **kustomize** 配置管理工具来部署应用程序。使用相同的步骤使用清单安装 Operator。

- 详情请参阅 使用 Kustomize 清单 来部署应用程序和使用 清单示例。

8.1.1.2. Operator 的 Operator Lifecycle Manager

您还可以使用 Operator Lifecycle Manager (OLM)将附加组件 Operator 安装到 MicroShift 集群。OLM 可用于管理广泛可用的自定义 Operator 和 Operator。将 OLM 与 MicroShift 搭配使用需要构建目录。

- 详情请参阅在 MicroShift 中使用 Operator Lifecycle Manager。

8.2. 在 MICROSHIFT 中使用 OPERATOR LIFECYCLE MANAGER

MicroShift 中使用 Operator Lifecycle Manager (OLM)软件包管理器来安装和运行可选 的附加组件 Operator。

8.2.1. 在 MicroShift 中使用 OLM 的注意事项

- MicroShift 不使用 OpenShift Container Platform 中所应用的集群 Operator。
- 您必须为应用程序一起使用的附加组件 Operator 创建自己的目录。默认不提供目录。
  - 每个目录都必须集群中添加了一个可访问的 **CatalogSource**，以便 OLM catalog Operator 可以使用目录进行内容。
- 您必须使用 CLI 对 MicroShift 进行 OLM 活动。控制台和 OperatorHub GUI 不可用。
  - 使用带有网络连接集群的 Operator Package Manager **opm** CLI，或用于为使用内部 registry 的自定义 Operator 构建目录。
  - 要为断开连接的或离线集群镜像目录和 Operator，请安装 oc-mirror OpenShift CLI 插件。

重要

在使用 Operator 之前，请验证红帽构建的 MicroShift 上支持 Operator 的供应商。

8.2.2. 确定 OLM 安装类型

您可以安装 OLM 软件包管理器以用于 MicroShift 4.15 或更新版本。根据您的用例，可以为 MicroShift 集群安装 OLM 的不同方法。

- 当您可以在 Red Hat Enterprise Linux (RHEL)上安装 MicroShift RPM 时，您可以同时安装 **microshift-olm** RPM。
- 您可以在现有的 MicroShift 4.16 上安装 **microshift-olm**。在安装 OLM 后，重启 MicroShift 服务，以使更改生效。请参阅从 RPM 软件包安装 Operator Lifecycle Manager (OLM) 。
- 您可以在 Red Hat Enterprise Linux for Edge (RHEL for Edge)镜像中嵌入 OLM。请参阅 将 Operator Lifecycle Manager (OLM)服务添加到蓝图中。

8.2.3. MicroShift 中的命名空间使用

microshift-olm RPM 会创建三个默认命名空间：一个用于运行 OLM，两个用于目录和 Operator 安装。您可以根据需要为用例创建额外的命名空间。

8.2.3.1. 默认命名空间

下表列出了默认命名空间，以及各个命名空间如何工作的简要描述。

表 8.1. OLM 为 MicroShift 创建的默认命名空间

默认命名空间	详情
openshift-operator-lifecycle-manager	OLM 软件包管理器在此命名空间中运行。
openshift-marketplace	全局命名空间。默认为空。要使目录源对所有命名空间的用户全局可用，请在 catalog-source YAML 中设置 **openshift-marketplace** 命名空间。
openshift-operators	Operator 在 MicroShift 中运行的默认命名空间。引用 **openshift-operators** 命名空间中的目录的 Operator 必须具有 **AllNamespaces** 监视范围。

8.2.3.2. 自定义命名空间

如果要在单个命名空间中同时使用目录和 Operator，则必须创建自定义命名空间。创建命名空间后，您必须在该命名空间中创建目录。在自定义命名空间中运行的所有 Operator 必须具有相同的单命名空间监视范围。

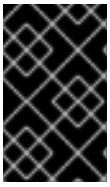
8.2.4. 关于构建 Operator 目录

要将 Operator Lifecycle Manager (OLM)与 MicroShift 搭配使用，您必须构建自定义 Operator 目录，然后使用 OLM 管理。MicroShift 不包含在 OpenShift Container Platform 中包含的标准目录。

8.2.4.1. 基于文件的目录

您可以为自定义 Operator 创建目录或过滤广泛可用 Operator 的目录。您可以组合这两种方法创建特定用例所需的目录。要使用您自己的 Operator 和 OLM 运行 MicroShift，请使用基于文件的目录结构创建一个目录。

- 详情请参阅 管理自定义目录和示例 目录。
- 另请参阅 **opm** CLI 参考。



重要

- 当 在集群中添加目录源 时，在 **catalogSource.yaml** 文件中将 **securityContextConfig** 值设置为 **restricted**。确保您的目录可以使用 **受限** 权限运行。

其他资源

- **opm** CLI 参考
- 关于 Operator 目录
- 要使用 **opm** CLI 创建基于文件的目录，请参阅管理自定义目录

8.2.5. 如何使用 OLM 部署 Operator

创建并部署自定义目录后，您必须创建一个 Subscription 自定义资源(CR)，以访问目录并安装您选择的 Operator。Operator 运行的位置取决于创建 Subscription CR 的命名空间。

重要

OLM 中的 Operator 具有监视范围。例如，一些 Operator 仅支持监视自己的命名空间，而其他 Operator 支持监视集群中的每个命名空间。在给定命名空间中安装的所有 Operator 必须具有相同的监视范围。

8.2.5.1. 连接和 OLM Operator 部署

可以在目录运行的任何位置部署 Operator。

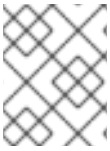
- 对于连接到互联网的集群，不需要镜像镜像。镜像可以通过网络拉取。
- 对于 MicroShift 只能访问内部网络的受限网络，镜像必须镜像到内部 registry。
- 对于 MicroShift 集群完全离线的用例，所有镜像都必须嵌入到 **osbuild** 蓝图中。

其他资源

- Operator 组成员

8.2.5.2. 使用全局命名空间将基于 OLM 的 Operator 添加到网络集群中

要将不同的 Operator 部署到不同的命名空间中，请使用此流程。对于具有网络连接的 MicroShift 集群，Operator Lifecycle Manager (OLM)可以访问托管在远程 registry 上的源。以下流程列出了使用配置文件安装使用全局命名空间的 Operator 的基本步骤。



注意

要使用安装在不同命名空间中的 Operator 或多个命名空间中，请确保目录源和引用 Operator 在 **openshift-marketplace** 命名空间中运行的 Subscription CR。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 已安装 Operator Lifecycle Manager (OLM)。
- 您已在全局命名空间中创建了一个自定义目录。

流程

1. 使用以下命令确认 OLM 正在运行：

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=olm-operator
```

输出示例

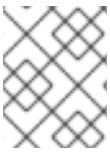
```
NAME                        READY   STATUS    RESTARTS   AGE
olm-operator-85b5c6786-n6kbc   1/1     Running   0          2m24s
```

2. 使用以下命令确认 OLM catalog Operator 正在运行：

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=catalog-operator
```

输出示例

```
NAME                             READY   STATUS    RESTARTS   AGE
catalog-operator-5fc7f857b6-tj8cf   1/1     Running   0          2m33s
```



注意

以下步骤假设您使用全局命名空间 **openshift-marketplace**。目录必须与 Operator 在同一命名空间中运行。Operator 必须支持 **AllNamespaces** 模式。

1. 使用以下示例 YAML 创建 **CatalogSource** 对象：

目录源 YAML 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: operatorhubio-catalog
  namespace: openshift-marketplace (1)
spec:
  sourceType: grpc
  image: quay.io/operatorhubio/catalog:latest
  displayName: Community Operators (2)
  publisher: OperatorHub.io
```

```
  grpcPodConfig:
    securityContextConfig: restricted (3)
  updateStrategy:
    registryPoll:
      interval: 60m
```

(1) 全局命名空间。将 **metadata.namespace** 设置为 **openshift-marketplace** 可让目录在所有命名空间中运行。任何命名空间中的订阅可以引用 **openshift-marketplace** 命名空间中创建的目录。

(2) 对于 MicroShift，默认情况下不通过 OLM 安装社区 Operator。此处仅列出的示例。

(3) 对于 MicroShift，**securityContextConfig** 的值必须设置为 **restricted**。

2. 运行以下命令来应用 **CatalogSource** 配置：

```
$ oc apply -f <my-catalog-source.yaml> (1)
```

(1) 将 **<my-catalog-source.yaml** > 替换为您的目录源配置文件名称。在本例中，使用 **catalogsource.yaml**。

输出示例

```
catalogsource.operators.coreos.com/operatorhubio-catalog created
```

3. 要验证是否已应用目录源，请使用以下命令检查 **READY** 状态：

```
$ oc describe catalogsources.operators.coreos.com -n openshift-marketplace operatorhubio-catalog
```

输出示例

```
Name:         operatorhubio-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
Metadata:
  Creation Timestamp:  2024-01-31T09:55:31Z
  Generation:          1
  Resource Version:    1212
  UID:                 4edc1a96-83cd-4de9-ac8c-c269ca895f3e
Spec:
  Display Name:  Community Operators
  Grpc Pod Config:
    Security Context Config:  restricted
  Image:                      quay.io/operatorhubio/catalog:latest
  Publisher:                  OperatorHub.io
  Source Type:                grpc
  Update Strategy:
    Registry Poll:
```

```
    Interval:  60m
Status:
  Connection State:
    Address:              operatorhubio-catalog.openshift-marketplace.svc:50051
    Last Connect:         2024-01-31T09:55:57Z
    Last Observed State:  READY (1)
  Registry Service:
    Created At:         2024-01-31T09:55:31Z
    Port:               50051
    Protocol:           grpc
    Service Name:       operatorhubio-catalog
    Service Namespace:  openshift-marketplace
Events:                 <none>
```

(1) 状态报告为 **READY**。

4. 使用以下命令确认目录源正在运行：

```
$ oc get pods -n openshift-marketplace -l olm.catalogSource=operatorhubio-catalog
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
operatorhubio-catalog-x24nh   1/1     Running   0          59s
```

5. 使用以下示例 YAML 创建 Subscription CR 配置文件：

Subscription 自定义资源 YAML 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-cert-manager
  namespace: openshift-operators
spec:
  channel: stable
  name: cert-manager
  source: operatorhubio-catalog
  sourceNamespace: openshift-marketplace (1)
```

(1) 全局命名空间。如果目录也在 **openshift-marketplace** 命名空间中运行，将 **sourceNamespace** 值设置为 **openshift-marketplace** 可让 Operator 在多个命名空间中运行。

6. 运行以下命令来应用 Subscription CR 配置：

```
$ oc apply -f <my-subscription-cr.yaml> (1)
```

(1) 将 **<my-subscription-cr.yaml&** gt; 替换为您的订阅 CR 文件名。在本例中，使用 **sub.yaml**。

输出示例

```
subscription.operators.coreos.com/my-cert-manager created
```

7. 您可以为您要使用的特定 Operand 创建配置文件，并现在应用它。

验证

1. 使用以下命令验证您的 Operator 是否正在运行：

```
$ oc get pods -n openshift-operators 1
```

1 使用 Subscription CR 中的命名空间。

注意

在 Operator 启动时，允许一两分钟。

输出示例

```
NAME                                  READY   STATUS    RESTARTS   AGE
cert-manager-7df8994ddb-4vrkr         1/1     Running   0          19s
cert-manager-cainjector-5746db8fd7-69442   1/1     Running   0          18s
cert-manager-webhook-f858bf58b-748nt      1/1     Running   0          18s
```

8.2.5.3. 将基于 OLM 的 Operator 添加到特定命名空间中的网络集群中

如果要为 Operator 指定命名空间，如 **olm-microshift**，请使用此流程。在本例中，目录有范围，可在全局 **openshift-marketplace** 命名空间中使用。Operator 使用来自全局命名空间的内容，但仅在 **olm-microshift** 命名空间中运行。对于具有网络连接的 MicroShift 集群，Operator Lifecycle Manager (OLM) 可以访问托管在远程 registry 上的源。

重要

在特定命名空间中安装的所有 Operator 必须具有相同的监视范围。在这种情况下，监视范围为 **OwnNamespace**。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 已安装 Operator Lifecycle Manager (OLM)。
- 您已创建了在全局命名空间中运行的自定义目录。

流程

1. 使用以下命令确认 OLM 正在运行：

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=olm-operator
```

输出示例

```
NAME                        READY   STATUS    RESTARTS   AGE
olm-operator-85b5c6786-n6kbc   1/1     Running   0          16m
```

2. 使用以下命令确认 OLM catalog Operator 正在运行：

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=catalog-operator
```

输出示例

```
NAME                            READY   STATUS    RESTARTS   AGE
catalog-operator-5fc7f857b6-tj8cf   1/1     Running   0          16m
```

3. 使用以下 YAML 示例创建命名空间：

命名空间 YAML 示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: olm-microshift
```

4. 使用以下命令应用命名空间配置：

```
$ oc apply -f _<ns.yaml>_ 1
```

1 将 *<ns.yaml>* 替换为命名空间配置文件的名称。在本例中，使用了 **olm-microshift**。

输出示例

```
namespace/olm-microshift created
```

5. 使用以下示例 YAML 创建 Operator 组 YAML：

Operator 组 YAML 示例

```
kind: OperatorGroup
apiVersion: operators.coreos.com/v1
metadata:
  name: og
  namespace: olm-microshift
spec: 1
  targetNamespaces:
  - olm-microshift
```

1 对于使用全局命名空间的 Operator，请省略 **spec.targetNamespaces** 字段和值。

6. 运行以下命令来应用 Operator 组配置：

```
$ oc apply -f _<og.yaml>_ 1
```

1 将 *<og.yaml>* 替换为 operator 组配置文件的名称。

输出示例

```
operatorgroup.operators.coreos.com/og created
```

7. 使用以下示例 YAML 创建 **CatalogSource** 对象：

目录源 YAML 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: operatorhubio-catalog
  namespace: openshift-marketplace 1
spec:
  sourceType: grpc
  image: quay.io/operatorhubio/catalog:latest
  displayName: Community Operators 2
  publisher: OperatorHub.io
  grpcPodConfig:
    securityContextConfig: restricted 3
  updateStrategy:
    registryPoll:
      interval: 60m
```

1 全局命名空间。将 **metadata.namespace** 设置为 **openshift-marketplace** 可让目录在所有命名空间中运行。任何命名空间中的订阅 CR 可以引用 **openshift-marketplace** 命名空间中创建的目录。

2 对于 MicroShift，默认情况下不通过 OLM 安装社区 Operator。此处仅列出的示例。

3 对于 MicroShift，**securityContextConfig** 的值必须设置为 **restricted**。

8. 运行以下命令来应用 **CatalogSource** 配置：

```
$ oc apply -f _<my-catalog-source.yaml>_ 1
```

1 将 *<my-catalog-source.yaml>* 替换为您的目录源配置文件名称。

9. 要验证是否已应用目录源，请使用以下命令检查 **READY** 状态：

```
$ oc describe catalogsources.operators.coreos.com -n openshift-marketplace operatorhubio-catalog
```

输出示例

```
Name:         operatorhubio-catalog
```

```
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
Metadata:
  Creation Timestamp:  2024-01-31T10:09:46Z
  Generation:          1
  Resource Version:    2811
  UID:                 60ce4a36-86d3-4921-b9fc-84d67c28df48
Spec:
  Display Name:  Community Operators
  Grpc Pod Config:
    Security Context Config:  restricted
  Image:                      quay.io/operatorhubio/catalog:latest
  Publisher:                  OperatorHub.io
  Source Type:                grpc
  Update Strategy:
    Registry Poll:
      Interval:  60m
Status:
  Connection State:
    Address:              operatorhubio-catalog.openshift-marketplace.svc:50051
    Last Connect:         2024-01-31T10:10:04Z
    Last Observed State:  READY (1)
  Registry Service:
    Created At:         2024-01-31T10:09:46Z
    Port:               50051
    Protocol:           grpc
    Service Name:       operatorhubio-catalog
    Service Namespace:  openshift-marketplace
Events:                 <none>
```

1 状态报告为 **READY**。

10. 使用以下命令确认目录源正在运行：

```
$ oc get pods -n openshift-marketplace -l olm.catalogSource=operatorhubio-catalog
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
operatorhubio-catalog-j7sc8   1/1     Running   0          43s
```

11. 使用以下示例 YAML 创建 Subscription CR 配置文件：

Subscription 自定义资源 YAML 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-gitlab-operator-kubernetes
  namespace: olm-microshift (1)
```

```
spec:
  channel: stable
  name: gitlab-operator-kubernetes
  source: operatorhubio-catalog
  sourceNamespace: openshift-marketplace 2
```

1 特定命名空间。Operator 引用内容的全局命名空间，但在 **olm-microshift** 命名空间中运行。

2 全局命名空间。任何命名空间中的订阅 CR 可以引用 **openshift-marketplace** 命名空间中创建的目录。

12. 运行以下命令来应用 Subscription CR 配置：

```
$ oc apply -f _<my-subscription-cr.yaml>_
```

输出示例

```
subscription.operators.coreos.com/my-gitlab-operator-kubernetes
```

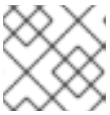
13. 您可以为您要使用的特定 Operand 创建配置文件，并现在应用它。

验证

1. 使用以下命令验证您的 Operator 是否正在运行：

```
$ oc get pods -n olm-microshift 1
```

1 使用 Subscription CR 中的命名空间。



注意

在 Operator 启动时，允许一两分钟。

输出示例

```
NAME                                    READY   STATUS    RESTARTS   AGE
gitlab-controller-manager-69bb6df7d6-g7ntx   2/2     Running   0          3m24s
```

其他资源

- 更新安装的 Operator
- 使用 CLI 从集群中删除 Operator

8.3. 使用 OC-MIRROR 插件创建自定义目录

您可以使用 oc-mirror OpenShift CLI (oc)插件创建带有广泛可用 Operator 的自定义目录，并对其进行镜像。

8.3.1. 使用红帽提供的 Operator 目录和镜像 registry

您可以使用 oc-mirror OpenShift CLI (oc)插件过滤和修剪目录以获取特定的 Operator 并进行镜像。您还可以在断开连接的设置中使用 Operator，或嵌入 Red Hat Enterprise Linux for Edge (RHEL for Edge)镜像中。要了解有关如何为镜像配置您的系统的更多详细信息，请使用以下"Additional resources"部分中的链接。如果您已准备好从红帽提供的 Operator 目录、镜像或将其嵌入到 RHEL for Edge 镜像中，请从以下部分开始，"使用 oc-mirror 插件创建目录内容"。

其他资源

- 在受限网络中使用 Operator Lifecycle Manager
- 配置主机以进行镜像 registry 访问
- 为完全断开连接的主机配置网络设置
- 获取镜像 registry 容器镜像列表
- 嵌入在 RHEL for Edge 镜像中供离线使用

8.3.2. 关于用于创建镜像 registry 的 oc-mirror 插件

您可以使用 MicroShift 的 oc-mirror OpenShift CLI (oc)插件来过滤和修剪 Operator 目录。然后，您可以将过滤的目录内容镜像到镜像 registry，或使用 RHEL for Edge 的断开连接的或离线部署中的容器镜像。

注意

MicroShift 使用 oc-mirror 插件的通用版本(1)。不要在 oc-mirror 插件的技术预览版本(2)中使用以下步骤。

您可以在本地将所需 Operator 所需的容器镜像镜像，或镜像到支持 Docker v2-2 （如 Red Hat Quay）的容器镜像。从连接到互联网到断开连接的镜像 registry 的 Red Hat 托管 registry 镜像内容的过程与您选择的 registry 无关。镜像目录内容后，将每个集群配置为从您的镜像 registry 中检索此内容。

8.3.2.1. 填充镜像 registry 时的连接注意事项

在填充 registry 时，您可以使用以下连接场景之一：

连接的镜像

如果您的主机可以同时访问互联网和您的镜像 registry，但不能访问您的集群节点，您可以直接从该机器中镜像内容。

断开连接的镜像

如果您没有可同时访问互联网和您的镜像 registry 的主机，您必须将镜像镜像(mirror)到文件系统中，然后将该主机或可移动介质置于断开连接的环境中。

重要

容器 registry 必须可以被您置备的集群中的每个机器访问。如果 registry 无法访问，则安装、更新和其他操作（如重新定位工作负载）可能会失败。

为了避免由无法访问的 registry 造成的问题，请使用以下标准实践：

- 以高可用性方式运行镜像 registry。
- 确保镜像 registry 至少与集群的生产环境可用性匹配。

其他资源

- 安装 oc mirror 插件

8.3.2.2. 使用 oc-mirror 插件检查目录内容

使用以下示例流程选择目录并列出可用 OpenShift Container Platform 内容的 Operator，以添加到 oc-mirror 插件镜像设置配置文件中。

注意

如果使用自己的目录和 Operator，您可以将镜像直接推送到内部 registry。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 已安装 Operator Lifecycle Manager (OLM)。
- 已安装 oc-mirror OpenShift CLI (oc)插件。

流程

1. 运行以下命令，获取可用红帽提供的 Operator 目录列表来过滤：

```
$ oc mirror list operators --version 4.16 --catalogs
```

2. 运行以下命令，获取 Red Hat Operator 目录中的 Operator 列表：

```
$ oc mirror list operators <--catalog=<catalog_source>> 1
```

1 指定目录源，如 **registry.redhat.io/redhat/redhat-operator-index:v4.16** 或 **quay.io/operatorhubio/catalog:latest**。

3. 选择一个 Operator。在本例中，选择了 **amq-broker-rhel8**。
4. 可选：要检查您要过滤的 Operator 的频道和版本，请输入以下命令：
   a. 运行以下命令来获取频道列表：

```
$ oc mirror list operators --catalog=registry.redhat.io/redhat/redhat-operator-index:v4.16 --package=amq-broker-rhel8
```

   b. 运行以下命令，获取频道中的版本列表：

```
$ oc mirror list operators --catalog=registry.redhat.io/redhat/redhat-operator-index:v4.16 --package=amq-broker-rhel8 --channel=7.11.x
```

后续步骤

- 使用此流程中收集的信息创建和编辑镜像设置配置文件。
- 将转换的镜像设置配置文件镜像到镜像 registry 或磁盘。

8.3.2.3. 创建镜像设置配置文件

您必须创建一个镜像设置配置文件，以使用 oc-mirror 插件镜像目录内容。镜像设置配置文件定义了哪些 Operator 与 oc-mirror 插件的其他配置设置一起镜像。生成默认镜像集文件后，您必须编辑内容，以便剩余的条目与 MicroShift 和您计划使用的 Operator 兼容。

您必须在镜像设置配置文件中指定存储后端。此存储后端可以是本地目录或支持 Docker v2-2 的 registry。oc-mirror 插件在创建镜像的过程中将元数据存储在这个存储后端中。

重要

不要删除或修改 oc-mirror 插件生成的元数据。每次针对同一镜像 registry 运行 oc-mirror 插件时，都必须使用相同的存储后端。

先决条件

- 您已创建了容器镜像 registry 凭证文件。请参阅配置允许镜像镜像的凭证。

流程

1. 使用 **oc mirror init** 命令为镜像设置配置创建模板，并将其保存到名为 **imageset-config.yaml** 的文件中：

```
$ oc mirror init <--registry <storage_backend> > imageset-config.yaml 1
```

1 指定存储后端的位置，如 **example.com/mirror/oc-mirror-metadata**。

默认镜像设置配置文件示例

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
storageConfig:
  registry:
    imageURL: registry.example.com/oc-mirror
    skipTLS: false
mirror:
  platform: 1
    channels:
    - name: stable-4.16
      type: ocp
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16
    packages:
    - name: serverless-operator
      channels:
      - name: stable
```

```
  additionalImages: 2
  - name: registry.redhat.io/ubi8/ubi:latest
  helm: {} 3
```

1. MicroShift 不支持 **platform** 字段和相关字段，必须被删除。
2. 指定要在镜像集中包含的任何其他镜像。如果不需要指定其他镜像，请删除此字段。
3. MicroShift 不支持 Helm，必须被删除。

2. 编辑镜像设置配置文件的值，以满足您要镜像的 MicroShift 和 Operator 的要求，如下例所示：

编辑 MicroShift 镜像设置配置文件示例

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
storageConfig: 1
  registry:
    imageURL: <storage_backend> 2
    skipTLS: false
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16 3
    packages:
    - name: amq-broker-rhel8 4
      channels:
      - name: 7.11.x 5
```

1. 设置保存镜像设置元数据的后端位置。此位置可以是 registry 或本地目录。必须指定 **storageConfig** 值。
2. 设置存储后端的注册表 URL，如 **<example.com/mirror/oc-mirror-metadata**。
3. 将 Operator 目录设置为从中检索镜像。
4. 指定要包含在镜像集中的 Operator 软件包。删除此字段以检索目录中的所有软件包。
5. 仅指定要包含在镜像集中的 Operator 软件包的某些频道。即使您没有使用该频道中的捆绑包，还必须始终包含 Operator 软件包的默认频道。您可以运行以下命令来找到默认频道：**oc mirror list operators --catalog=<catalog_name> --package=<package_name>**。

3. 保存更新的文件。

后续步骤

- 使用 oc-mirror 插件将镜像直接设置为目标镜像 registry。
- 配置 CRI-O。
- 将目录源应用到集群。

8.3.2.3.1. 镜像设置配置参数

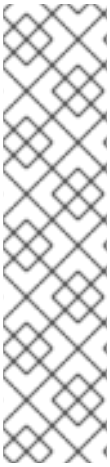
oc-mirror 插件需要一个镜像设置配置文件，该文件定义哪些镜像要镜像(mirror)。下表列出了 **ImageSetConfiguration** 资源的可用参数。

表 8.2. **ImageSetConfiguration 参数**

参数	描述	值
apiVersion	**ImageSetConfiguration** 内容的 API 版本。	字符串.例如：**mirror.openshift.io/v1alpha2**。
mirror	镜像集的配置。	对象
mirror.additionalImages	镜像集的额外镜像配置。	对象数组。例如： additionalImages: - name: registry.redhat.io/ubi8/ubi:latest
mirror.additionalImages.name	要 mirror 的镜像的标签或摘要。	字符串.例如：**registry.redhat.io/ubi8/ubi:latest**
mirror.blockedImages	阻止 mirror 的镜像的完整标签、摘要或模式。	字符串数组。例如：**docker.io/library/alpine**
mirror.operators	镜像集的 Operator 配置。	对象数组。例如： operators: - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16 packages: - name: elasticsearch-operator minVersion: '2.4.0'
mirror.operators.catalog	包括在镜像集中的 Operator 目录。	字符串.例如：**registry.redhat.io/redhat/redhat-operator-index:v4.16**。

参数	描述	值
mirror.operators.full	为 **true** 时，下载完整的目录、Operator 软件包或 Operator 频道。	布尔值.默认值为 **false**。
mirror.operators.packages	Operator 软件包配置。	对象数组。例如： operators: - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16 packages: - name: elasticsearch-operator minVersion: '5.2.3-31'
mirror.operators.packages.name	镜像集中要包含的 Operator 软件包名称	字符串.例如：**elasticsearch-operator**。
mirror.operators.packages.channels	Operator 软件包频道配置。	对象
mirror.operators.packages.channels.name	Operator 频道名称（软件包中唯一）要包括在镜像集中。	字符串.例如：**fast** 或 **stable-v4.16**。
mirror.operators.packages.channels.maxVersion	Operator 镜像的最高版本，在其中存在所有频道。详情请查看以下备注。	字符串.例如：**5.2.3-31**
mirror.operators.packages.channels.minBundle	要包含的最小捆绑包的名称，以及频道头更新图中的所有捆绑包。仅在命名捆绑包没有语义版本元数据时设置此字段。	字符串.例如：**bundleName**
mirror.operators.packages.channels.minVersion	Operator 的最低版本，用于镜像存在的所有频道。详情请查看以下备注。	字符串.例如：**5.2.3-31**
mirror.operators.packages.maxVersion	Operator 最高版本，可跨所有存在的频道进行镜像。详情请查看以下备注。	字符串.例如：**5.2.3-31**。
mirror.operators.packages.minVersion	Operator 的最低版本，用于镜像存在的所有频道。详情请查看以下备注。	字符串.例如：**5.2.3-31**。
mirror.operators.skipDependencies	如果为 **true**，则不会包含捆绑包的依赖项。	布尔值.默认值为 **false**。

参数	描述	值
mirror.operators.targetCatalog	要镜像引用的目录的替代名称和可选命名空间层次结构。	字符串.例如：**my-namespace/my-operator-catalog**
mirror.operators.targetName	将引用的目录镜像为。 **targetName** 参数已弃用。改为使用 **targetCatalog** 参数。	字符串.例如： **my-operator-catalog**
mirror.operators.targetTag	附加到 **targetName** 或 **targetCatalog** 的替代标签。	字符串.例如： **v1**
storageConfig	镜像集的后端配置。	对象
storageConfig.local	镜像集的本地后端配置。	对象
storageConfig.local.path	包含镜像设置元数据的目录路径。	字符串.例如：**./path/to/dir/**。
storageConfig.registry	镜像集的 registry 后端配置。	对象
storageConfig.registry.imageURL	后端 registry URI。可以选择在 URI 中包含命名空间引用。	字符串.例如：**quay.io/myuser/imageset:metadata**。
storageConfig.registry.skipTLS	（可选）跳过引用的后端 registry 的 TLS 验证。	布尔值.默认值为 **false**。



注意

使用 **minVersion** 和 **maxVersion** 属性过滤特定 Operator 版本范围可能会导致多个频道头错误。错误信息将显示有**多个频道**头。这是因为在应用过滤器时，Operator 的更新图会被截断。

Operator Lifecycle Manager 要求每个 operator 频道都包含一个端点组成更新图表的版本，即 Operator 的最新版本。在应用图形的过滤器范围时，可以进入两个或多个独立图形或具有多个端点的图形。

要避免这个错误，请不要过滤 Operator 的最新版本。如果您仍然遇到错误，具体取决于 Operator，则必须增加 **maxVersion** 属性，或者 **minVersion** 属性必须减少。因为每个 Operator 图都可以不同，所以您可能需要调整这些值，直到错误解决为止。

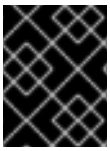
其他资源

- imageset 配置示例

8.3.2.4. 镜像(mirror)到镜像(mirror)的镜像

您可以使用 oc-mirror 插件将镜像直接设置为在镜像设置过程中可访问的目标镜像 registry。

您必须在镜像设置配置文件中指定存储后端。这个存储后端可以是本地目录或 Docker v2 registry。oc-mirror 插件在创建镜像的过程中将元数据存储在这个存储后端中。



重要

不要删除或修改 oc-mirror 插件生成的元数据。每次针对同一镜像 registry 运行 oc-mirror 插件时，都必须使用相同的存储后端。

先决条件

- 您可以访问互联网来获取所需的容器镜像。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 **oc-mirror** CLI 插件。
- 您已创建了镜像设置配置文件。

流程

- 运行 **oc mirror** 命令将指定镜像集配置中的镜像镜像到指定的 registry:

```
$ oc mirror --config=./<imageset-config.yaml> \ 1
  docker://registry.example:5000                2
```

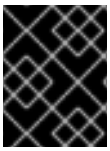
1. 指定您创建的镜像设置配置文件。例如，**imageset-config.yaml**。
2. 指定要镜像设置文件的 registry。registry 必须以 **docker://** 开头。如果为镜像 registry 指定顶层命名空间，则必须在后续执行时使用此命名空间。

输出示例

```
Rendering catalog image "registry.example.com/redhat/redhat-operator-index:v{ocp-version}" with file-based catalog
```

验证

1. 进入生成的 **oc-mirror-workspace/** 目录。
2. 导航到结果目录，例如，**results-1639608409/**。
3. 验证 **ImageContentSourcePolicy** 和 **CatalogSource** 资源是否存在 YAML 文件。



重要

ImageContentSourcePolicy YAML 文件用作在 MicroShift 中手动配置 CRI-O 的参考内容。您无法将资源直接应用到 MicroShift 集群。

后续步骤

- 转换 **ImageContentSourcePolicy** YAML 内容，用于手动配置 CRI-O。
- 如果需要，将镜像从 mirror 镜像到磁盘，以便断开连接或离线使用。
- 配置集群以使用 oc-mirror 生成的资源。

故障排除

- 无法检索源镜像。

其他资源

- 在部分断开连接的环境中镜像设置的镜像
- 镜像在完全断开连接的环境中设置的镜像

8.3.2.5. 为 Operator 使用 registry 镜像配置 CRI-O

您必须将通过 oc-mirror 插件创建的 **imageContentSourcePolicy.yaml** 文件转换为与 MicroShift 使用的 CRI-O 容器运行时配置兼容的格式。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 已安装 Operator Lifecycle Manager (OLM)。
- 已安装 oc-mirror OpenShift CLI (oc)插件。
- 已安装 **yq** 二进制文件。
- **ImageContentSourcePolicy** 和 **CatalogSource** YAML 文件包括在 **oc-mirror-workspace/results explained** 目录中。

流程

1. 运行以下命令确认 **imageContentSourcePolicy.yaml** 文件的内容：

```
$ cat oc-mirror-workspace/<results-directory>/imageContentSourcePolicy.yaml
```
(1)

(1) 指定 结果 目录名称，如 < **results-1707148826** >。

输出示例

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  labels:
    operators.openshift.org/catalog: "true"
  name: operator-0
spec:
  repositoryDigestMirrors:
```

```
  - mirrors:
    - registry.<example.com>/amq7
    source: registry.redhat.io/amq7
```

2. 运行以下命令，将 **imageContentSourcePolicy.yaml** 转换为 CRI-O 配置的格式：

```
yq '.spec.repositoryDigestMirrors[] as $item ireduce([]; . + [{"mirror": $item.mirrors[], "source":
($item | .source)}]) | .[] |
  "[[registry]]
     prefix = \"" + .source + "\"
     location = \"" + .mirror + "\"
     mirror-by-digest-only = true
     insecure = true
     "' ./icsp.yaml
```

输出示例

```
[[registry]]
     prefix = "registry.redhat.io/amq7"
     location = "registry.example.com/amq7"
     mirror-by-digest-only = true
     insecure = true
```

3. 将输出添加到 **/etc/containers/registries.conf.d/** 目录中的 CRI-O 配置文件中：

crio-config.yaml 镜像配置文件示例

```
[[registry]]
     prefix = "registry.redhat.io/amq7"
     location = "registry.example.com/amq7"
     mirror-by-digest-only = true
     insecure = true

[[registry]]
    prefix = ""
    location = "quay.io"
    mirror-by-digest-only = true
[[registry.mirror]]
    location = "<registry_host>:<port>" (1)
    insecure = false
```

(1) 指定镜像 registry 服务器的主机名和端口，如 **microshift-quay:8443**。

4. 使用以下命令重启 MicroShift 来应用 CRI-O 配置更改：

```
$ sudo systemctl restart crio
```

8.3.2.6. 安装使用 oc-mirror 插件创建的自定义目录

将镜像设置为镜像 registry 后，您必须将生成的 **CatalogSource** 自定义资源(CR)应用到集群。Operator Lifecycle Manager (OLM)使用 **CatalogSource** CR 来检索有关镜像 registry 中可用 Operator 的信息。然后，您必须创建并应用订阅 CR 来订阅自定义目录。

先决条件

- 您已将镜像设置为 registry 镜像。
- 您已将镜像引用信息添加到 CRI-O 容器运行时配置中。

流程

1. 运行以下命令，从 results 目录中应用目录源配置文件来创建目录源对象：

```
$ oc apply -f ./oc-mirror-workspace/results-1708508014/catalogSource-cs-redhat-operator-index.yaml
```

目录源配置文件示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: redhat-catalog
  namespace: openshift-marketplace (1)
spec:
  sourceType: grpc
  image: registry.example.com/redhat/redhat-operator-index:v4.16
  updateStrategy:
    registryPoll:
      interval: 60m
```

(1) 指定全局命名空间。将 **metadata.namespace** 设置为 **openshift-marketplace** 可让目录引用所有命名空间中的目录。任何命名空间中的订阅可以引用 **openshift-marketplace** 命名空间中创建的目录。

输出示例

```
catalogsource.operators.coreos.com/cs-redhat-operator-index created
```

2. 运行以下命令验证 **CatalogSource** 资源是否已成功安装：

```
$ oc get catalogsource --all-namespaces
```

3. 使用以下命令验证目录源是否正在运行：

```
$ oc get pods -n openshift-marketplace
```

输出示例

```
NAME                          READY   STATUS    RESTARTS   AGE
cs-redhat-operator-index-4227b   2/2     Running   0          2m5s
```

4. 创建一个 **Subscription** CR，类似以下示例：

Subscription CR 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-broker
  namespace: openshift-operators
spec:
  channel: 7.11.x
  name: amq-broker-rhel8
  source: cs-redhat-operator-index
  sourceNamespace: openshift-marketplace
```

5. 运行以下命令来应用 Subscription CR 配置：

```
$ oc apply -f ./<my-subscription-cr.yaml> 1
```

1 指定订阅的名称，如 **my-subscription-cr.yaml**。

输出示例

```
subscription.operators.coreos.com/amq-broker created
```

8.4. 将基于 OLM 的 OPERATOR 添加到断开连接的集群中

您可以将基于 OLM 的 Operator 嵌入到 Red Hat Enterprise Linux for Edge (RHEL for Edge)镜像中。

8.4.1. 关于将基于 OLM 的 Operator 添加到断开连接的集群中

对于在断开连接的集群上安装的 Operator，Operator Lifecycle Manager (OLM)默认无法访问托管在远程 registry 上的源，因为这些远程源需要足够的互联网连接。因此，您必须将远程 registry 镜像为高度可用的容器 registry。

在断开连接的环境中，需要执行以下步骤来使用基于 OLM 的 Operator：

- 将 OLM 包含在您的镜像 registry 的容器镜像列表中。
- 通过直接更新 CRI-O 配置将系统配置为使用您的镜像 registry。MicroShift 不支持 **ImageContentSourcePolicy**。
- 在集群中添加 **CatalogSource** 对象，以便 OLM catalog Operator 可以使用镜像 registry 上的本地目录。
- 确保 MicroShift 已安装在断开连接的容量中运行。
- 确保网络设置配置为以断开连接模式运行。

在断开连接的集群中启用 OLM 后，您可以继续使用互联网连接的工作站来在发布新版 Operator 时保持本地目录源更新。

其他资源

- 创建 RHEL for Edge 镜像
- 嵌入在 RHEL for Edge 镜像中供离线使用

- 为完全断开连接的主机配置网络设置

8.4.1.1. 执行空运行

您可以使用 oc-mirror 来执行空运行，而无需实际镜像(mirror)。这可让您查看要镜像的镜像列表，以及从镜像 registry 修剪的所有镜像。使用空运行（dry run）还允许您在早期版本中捕获与镜像集配置相关的任何错误，或使用生成的镜像列表以及其他工具来执行镜像操作。

先决条件

- 您可以访问互联网来获取所需的容器镜像。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 **oc-mirror** CLI 插件。
- 您已创建了镜像设置配置文件。

流程

1. 使用 **--dry-run** 标志运行 **oc mirror** 命令来执行空运行：

```
$ oc mirror --config=./imageset-config.yaml \ 1
  docker://registry.example:5000           \ 2
  --dry-run                                  3
```

1. 传递创建的镜像设置配置文件。此流程假设它名为 **imageset-config.yaml**。
2. 指定镜像 registry。在使用 **--dry-run** 标志时，不会镜像这个 registry。
3. 使用 **--dry-run** 标志来生成空运行工件，而不是实际的镜像设置文件。

输出示例

```
Checking push permissions for registry.example:5000
Creating directory: oc-mirror-workspace/src/publish
Creating directory: oc-mirror-workspace/src/v2
Creating directory: oc-mirror-workspace/src/charts
Creating directory: oc-mirror-workspace/src/release-signatures
No metadata detected, creating new workspace
wrote mirroring manifests to oc-mirror-workspace/operators.1658342351/manifests-redhat-operator-index

...

info: Planning completed in 31.48s
info: Dry run complete
Writing image mapping to oc-mirror-workspace/mapping.txt
```

2. 进入生成的工作区目录：

```
$ cd oc-mirror-workspace/
```

3. 查看生成的 **mapping.txt** 文件。
   此文件包含将要镜像的所有镜像的列表。

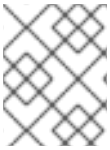
4. 查看生成的 **prune-plan.json** 文件。
   此文件包含在发布镜像集时从镜像 registry 中修剪的所有镜像的列表。

注意

只有在 oc-mirror 命令指向您的镜像 registry 且需要修剪的镜像时，才会生成 **prune-plan.json** 文件。

8.4.1.2. 在断开连接的环境中获取用于 RHEL for Edge 的目录和 Operator 容器镜像引用

使用 oc-mirror 插件执行空运行后，以查看您要镜像的镜像列表，您必须获取所有容器镜像引用，然后格式化添加到镜像构建器蓝图的输出。



注意

对于为专有 Operator 创建的目录，您可以在不按照以下流程的情况下格式化镜像构建器蓝图的镜像引用。

先决条件

- 有要使用的 Operator 的目录索引。
- 已安装 **jq** CLI 工具。
- 熟悉镜像构建器蓝图文件。
- 您有一个 Image Builder 蓝图 TOML 文件。

流程

1. 解析目录 **index.json** 文件，以获取 Image Builder 蓝图中包含的镜像引用。您可以使用 unfiltered 目录，也可以过滤掉无法镜像的镜像：

   a. 运行以下命令，解析未过滤的目录 **index.json** 文件以获取镜像引用：

   ```
   jq -r --slurp '.[] | select(.relatedImages != null) | "[[containers]]\nsource = \"" +
   .relatedImages[].image + "\"\n"'  ./oc-mirror-
   workspace/src/catalogs/registry.redhat.io/redhat/redhat-operator-
   index/v4.16/index/index.json
   ```

   b. 如果要过滤无法镜像的镜像，请运行以下命令过滤并解析目录 **index.json** 文件：

   ```
   $ jq -r --slurp '.[] | select(.relatedImages != null) | .relatedImages[] | select(.name |
   contains("ppc") or contains("s390x") | not) | "[[containers]]\\nsource = \\"" + .image +
   "\\"\\n"' ./oc-mirror-workspace/src/catalogs/registry.redhat.io/redhat/redhat-operator-
   index/v4.16/index/index.json
   ```

注意

此步骤使用 AMQ Broker Operator 作为示例。您可以在 **jq** 命令中添加其他条件来进一步过滤您的用例。

image-reference 输出示例

```
[[containers]]
source = "registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:0b2126cfb6054fdf428c1f43b69e36e93a09a49ce15350e9273c98cc08c6598b"

[[containers]]
source = "registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:0dde839c2dce7cb684094bf26523c8e16677de03149a0fff468b8c3f106e1f4f"
...
...

[[containers]]
source = "registry.redhat.io/amq7/amq-broker-rhel8@sha256:e8fa2a00e576ecb95561ffbdbf87b1c82d479c8791ab2c6ce741dd0d0b496d15"

[[containers]]
source = "registry.redhat.io/amq7/amq-broker-rhel8@sha256:ff6fefad518a6c997d4c5a6e475ba89640260167f0bc27715daf3cc30116fad1"
…
EOF
```

重要

对于镜像和断开连接的用例，请确保从目录 **index.json** 文件中过滤的所有源都是摘要。如果任何源使用标签而不是摘要，Operator 安装会失败。标签需要互联网连接。

2. 运行以下命令，查看 **imageset-config.yaml** 以获取 **CatalogSource** 自定义资源(CR)的目录镜像引用：

```
$ cat imageset-config.yaml
```

输出示例

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
storageConfig:
  registry:
    imageURL: registry.example.com/microshift-mirror
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16 1
    packages:
    - name: amq-broker-rhel8
      channels:
      - name: 7.11.x
```

1 使用 **mirror.catalog** catalog image reference for the follwing **jq** 命令中的值来获取镜像摘要。在本例中，< *registry.redhat.io/redhat/redhat-operator-index:v4.16* >。

3. 运行以下命令，获取目录索引镜像的 SHA：

```
$ skopeo inspect docker://<registry.redhat.io/redhat/redhat-operator-index:v4.16> | jq
`.Digest` 1
```

1 使用 **jq** 命令的 **mirror.catalog** 目录镜像引用中的值来获取镜像摘要。在本例中，< *registry.redhat.io/redhat/redhat-operator-index:v4.16* >。

输出示例

```
"sha256:7a76c0880a839035eb6e896d54ebd63668bb37b82040692141ba39ab4c539bc6"
```

4. 要准备在 Image Builder 蓝图文件中添加镜像引用，请使用以下示例格式化目录镜像引用：

```
[[containers]]
source = "registry.redhat.io/redhat/redhat-operator-
index@sha256:7a76c0880a839035eb6e896d54ebd63668bb37b82040692141ba39ab4c539bc
6"
```

5. 将前面所有步骤中的镜像引用添加到镜像构建器蓝图中。

生成的镜像构建器蓝图示例片断

```
name = "microshift_blueprint"
description = "MicroShift 4.16.1 on x86_64 platform"
version = "0.0.1"
modules = []
groups = []

[[packages]] 1
name = "microshift"
version = "4.16.1"
...
...

[customizations.services] 2
enabled = ["microshift"]

[customizations.firewall]
ports = ["22:tcp", "80:tcp", "443:tcp", "5353:udp", "6443:tcp", "30000-32767:tcp", "30000-
32767:udp"]
...
...

[[containers]] 3
source = "quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:f41e79c17e8b41f1b0a5a32c3e2dd7cd15b8274554d3f1ba12b2598a347475f4"

[[containers]]
```

```
source = "quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:dbc65f1fba7d92b36cf7514cd130fe83a9bd211005ddb23a8dc479e0eea645fd"
...
...

[[containers]] 4
source = "registry.redhat.io/redhat/redhat-operator-
index@sha256:7a76c0880a839035eb6e896d54ebd63668bb37b82040692141ba39ab4c539bc
6"
...
...

[[containers]]
source = "registry.redhat.io/amq7/amq-broker-init-
rhel8@sha256:0dde839c2dce7cb684094bf26523c8e16677de03149a0fff468b8c3f106e1f4f"
...
...

[[containers]]
source = "registry.redhat.io/amq7/amq-broker-
rhel8@sha256:e8fa2a00e576ecb95561ffbdbf87b1c82d479c8791ab2c6ce741dd0d0b496d15"


[[containers]]
source = "registry.redhat.io/amq7/amq-broker-
rhel8@sha256:ff6fefad518a6c997d4c5a6e475ba89640260167f0bc27715daf3cc30116fad1"
…
EOF
```

1. 使用与 **microshift-release-info** RPM 兼容的相同版本的所有非可选 MicroShift RPM 软件包的引用。
2. 在系统启动时自动启用 MicroShift 并应用默认网络设置的引用。
3. 对断开连接的部署所需的所有非可选 MicroShift 容器镜像的引用。
4. 目录索引的引用。

8.4.1.3. 在断开连接的 RHEL for Edge 镜像中应用目录和 Operator

为断开连接的环境创建了 RHEL for Edge 镜像并配置了 MicroShift 网络设置后，您可以配置命名空间和创建目录和 Operator 自定义资源(CR)以运行 Operator。

先决条件

- 您有一个 RHEL for Edge 镜像。
- 配置了网络以断开连接使用。
- 您完成了 oc-mirror 插件空运行过程。

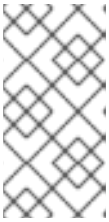
流程

1. 创建一个 **CatalogSource** 自定义资源(CR)，如下例所示：

my-catalog-source-cr.yaml 文件示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: cs-redhat-operator-index
  namespace: openshift-marketplace 1
spec:
  image: registry.example.com/redhat/redhat-operator-index:v4.16
  sourceType: grpc
  displayName:
  publisher:
  updateStrategy:
    registryPoll:
      interval: 60m
```

1 全局命名空间。将 **metadata.namespace** 设置为 **openshift-marketplace** 可让目录在所有命名空间中运行。任何命名空间中的订阅可以引用 **openshift-marketplace** 命名空间中创建的目录。



注意

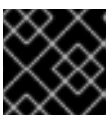
openshift-marketplace 的默认 pod 安全准入定义是 **baseline**，因此在该命名空间中创建的目录源自定义资源(CR)不需要设置 **spec.grpcPodConfig.securityContextConfig** 值。如果需要使用命名空间和 Operator，您可以设置 **传统** 或**受限** 值。

2. 将目录索引提交的 SHA 添加到目录源(CR)中，如下例所示：

命名空间 spec.image 配置示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: cs-redhat-operator-index
  namespace: openshift-marketplace
spec:
  image: registry.example.com/redhat/redhat-operator-
index@sha256:7a76c0880a839035eb6e896d54ebd63668bb37b82040692141ba39ab4c539bc
6 1
  sourceType: grpc
  displayName:
  publisher:
  updateStrategy:
    registryPoll:
      interval: 60m
```

1 镜像提交的 SHA。使用添加到镜像构建器蓝图中的相同 SHA。



重要

您必须使用 SHA 而不是目录 CR 中的标签，或者 pod 无法启动。

3. 运行以下命令，将 oc-mirror 插件空运行结果目录应用到集群：

```
$ oc apply -f ./oc-mirror-workspace/results-1708508014/catalogSource-cs-redhat-operator-index.yaml
```

输出示例

```
catalogsource.operators.coreos.com/cs-redhat-operator-index created
```

4. 运行以下命令验证 **CatalogSource** 资源是否已成功安装：

```
$ oc get catalogsource --all-namespaces
```

5. 使用以下命令验证目录源是否正在运行：

```
$ oc get pods -n openshift-marketplace
```

输出示例

```
NAME                                    READY   STATUS    RESTARTS   AGE
cs-redhat-operator-index-4227b   2/2     Running   0          2m5s
```

6. 创建一个 **Subscription** CR，类似以下示例：

my-subscription-cr.yaml 文件示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-broker
  namespace: openshift-operators
spec:
  channel: 7.11.x
  name: amq-broker-rhel8
  source: cs-redhat-operator-index
  sourceNamespace: openshift-marketplace
```

7. 运行以下命令来应用 **Subscription** CR：

```
$ oc apply -f ./<my-subscription-cr.yaml> 1
```

1 指定 **Subscription** CR 的名称，如 **my-subscription-cr.yaml**。

输出示例

```
subscription.operators.coreos.com/amq-broker created
```