OpenShift Container Platform 4.19

扩展

使用 Operator Lifecycle Manager (OLM) v1 在 OpenShift Container Platform 中使用扩展。

Last Updated: 2025-10-03

OpenShift Container Platform 4.19 扩展

使用 Operator Lifecycle Manager (OLM) v1 在 OpenShift Container Platform 中使用扩展。

Legal Notice

Abstract

本文档提供有关在 OpenShift Container Platform 上安装、管理和配置扩展和 Operator 的信息。

Table of Contents

第 1 章 扩展概述

扩展可让集群管理员在其 OpenShift Container Platform 集群上为用户扩展功能。

自 OpenShift Container Platform 4 初始发行以来，Operator Lifecycle Manager (OLM) 已包含在 OpenShift Container Platform 4 中。OpenShift Container Platform 4.19 包括了一个面向下一代 OLM 操作的组件，作为正式发行 (GA)功能，在此阶段被称为 *OLM v1*。此更新的框架改变了很多属于以前版本的 OLM 的概念，并添加了新功能。

注意

对于 OpenShift Container Platform 4.19，适用于 OLM v1 的流程都是基于 CLI 的。另外，管理员也可以使用普通方法（如 **Import YAML** 和 **Search** 页面）在 web 控制台中创建和查看相关对象。但是，现有的 **OperatorHub** 和 **Installed Operators** 页还不会显示 OLM v1 组件。

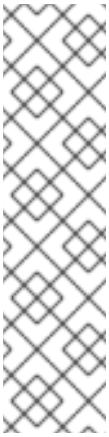
1.1. 亮点

管理员可以探索以下亮点：

支持 GitOps 工作流的全声明性模型

OLM v1 通过两个关键 API 简化扩展管理：

- 新的 **ClusterExtension** API 简化了已安装的扩展的管理，其中包括通过 **registry+v1** 捆绑包格式的 Operator，通过将面向用户的 API 整合到单个对象中。此 API 由新的 Operator Controller 组件作为 **clusterextension.olm.operatorframework.io** 提供。管理员和 SRE 可使用 API 自动进程并使用 GitOps 原则定义所需状态。

  注意

  OLM v1 的早期技术预览阶段引入了一个新的 **Operator** API；此 API 在 OpenShift Container Platform 4.16 中被重命名为 **ClusterExtension**，以解决以下改进：

  - 更准确地反映了扩展集群功能的简化功能
  - 准确地代表了一个更灵活的打包格式
  - **Cluster** 前缀明确表示 **ClusterExtension** 对象是集群范围的。这与 OLM (Classic) 不同，其中 Operator 可以是命名空间范围的或集群范围的

- **Catalog** API 由新 catalogd 组件提供，作为 OLM v1 的基础，为 on-cluster 客户端解包目录，以便用户可以发现可安装的内容，如 Kubernetes 扩展和 Operator。这可让您提高所有可用 Operator 捆绑包版本的可见性，包括它们的详情、频道和更新边缘。

如需更多信息，请参阅 Operator Controller 和 Catalogd。

改进了对扩展更新的控制

通过改进对目录内容的了解，管理员可以指定用于安装和更新的目标版本。这样，管理员可以更好地控制扩展更新的目标版本。如需更多信息，请参阅更新集群扩展。

灵活的扩展打包格式

管理员可以使用基于文件的目录来安装和管理扩展，如基于 OLM 的 Operator，与 OLM (Classic) 经验类似。

另外，捆绑包大小不再受 etcd 值大小限制。如需更多信息，请参阅安装扩展。

安全目录通信

OLM v1 使用 HTTPS 加密进行目录服务器响应。

代理环境和可信 CA 证书的基本支持

Operator Controller 和 catalogd 可以在代理环境中运行，并包括对可信 CA 证书的基本支持。

1.2. 用途

Operator Lifecycle Manager (OLM) 的任务是集中管理集群扩展的生命周期，并在 Kubernetes 集群上以声明性的方式管理。其目的是在整个基础集群生命周期中，对集群管理员和平台即服务(PaaS)管理员的安装、运行和更新功能扩展变得简单、安全且可重复生成。

由 OpenShift Container Platform 4 启动的 OLM 的初始版本，默认包含在内，专注于为特定类型的集群扩展（称为 Operator）提供特殊支持。Operator 被归类为一个或多个 Kubernetes 控制器，与一个或多个 API 扩展(**CustomResourceDefinition** （CRD）一起提供，以为集群提供额外的功能。

在生产环境集群中运行了多个版本后，OLM 的下一代产品旨在管理集群扩展（不仅限于 Operator）的生命周期。

第 2 章 架构

2.1. OLM V1 组件概述

Operator Lifecycle Manager (OLM) v1 由以下组件项目组成：

Operator 控制器

Operator Controller 是 OLM v1 的核心组件，它通过 API 扩展 Kubernetes，用户可以安装和管理 Operator 和扩展的生命周期。它消耗来自 catalogd 的信息。

Catalogd

Catalogd 是一个 Kubernetes 扩展，它解包基于文件的目录(FBC)内容，由容器镜像打包并提供，供集群客户端使用。作为 OLM v1 微服务架构的组件，用于由扩展作者打包的 Kubernetes 扩展的目录主机元数据，因此可帮助用户发现可安装的内容。

2.2. OPERATOR 控制器

Operator Controller 是 Operator Lifecycle Manager (OLM) v1 的中央组件，它消耗其他 OLM v1 组件 catalogd。它使用一个 API 扩展 Kubernetes，用户可以安装 Operator 和扩展。

2.2.1. ClusterExtension API

Operator Controller 提供了一个新的 **ClusterExtension** API 对象，它是一个代表安装扩展实例的单个资源，其中包括通过 **registry+v1** 捆绑包格式的 Operator。此 **clusterextension.olm.operatorframework.io** API 通过将面向用户的 API 整合到单个对象来简化安装的扩展管理。

重要

在 OLM v1 中，**ClusterExtension** 对象是集群范围的。这与 OLM (Classic) 不同，根据相关 **Subscription** 和 **OperatorGroup** 对象的配置，Operator 可以是命名空间范围的或集群范围的。

如需有关之前行为的更多信息，请参阅 *多租户和 Operator 共处*。

ClusterExtension 对象示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <extension_name>
spec:
  namespace: <namespace_name>
  serviceAccount:
    name: <service_account_name>
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      channels:
        - <channel>
      version: "<version>"
```

其他资源

- Operator Lifecycle Manager (OLM)→ Multitenancy 和 Operator colocation

2.2.1.1. 指定目标版本的自定义资源(CR)示例

在 Operator Lifecycle Manager (OLM) v1 中，集群管理员可以在自定义资源(CR)中声明性地设置 Operator 或扩展的目标版本。

您可以通过指定以下字段来定义目标版本：

- Channel
- 版本号
- 版本范围

如果您在 CR 中指定频道，OLM v1 会安装可在指定频道中解析的 Operator 或扩展的最新版本。当向指定的频道发布更新时，OLM v1 会自动更新至可以从频道解析的最新发行版本。

带有指定频道的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    name: <clusterextension_name>
  spec:
    namespace: <installed_namespace>
    serviceAccount:
      name: <service_account_installer_name>
    source:
      sourceType: Catalog
      catalog:
        packageName: <package_name>
        channels:
          - latest 1
```

1 可选：安装可从指定频道解析的最新版本。对频道的更新会自动安装。指定 **channels** 参数的值（数组）。

如果在 CR 中指定 Operator 或扩展的目标版本，OLM v1 将安装指定的版本。当在 CR 中指定目标版本时，OLM v1 在向目录发布更新时不会更改目标版本。

如果要更新集群中安装的 Operator 版本，您必须手动编辑 Operator 的 CR。指定 Operator 的目标版本将 Operator 的版本固定到指定的发行版本。

指定了目标版本的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    name: <clusterextension_name>
  spec:
    namespace: <installed_namespace>
    serviceAccount:
```

```
    name: <service_account_installer_name>
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      version: "1.11.1" 1
```

1 可选：指定目标版本。如果要更新安装的 Operator 或扩展版本，您必须手动将 CR 更新至所需的目标版本。

如果要为 Operator 或扩展定义可接受的版本范围，您可以使用比较字符串指定版本范围。当您指定版本范围时，OLM v1 会安装可由 Operator Controller 解析的 Operator 或扩展的最新版本。

指定了版本范围的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <clusterextension_name>
spec:
  namespace: <installed_namespace>
  serviceAccount:
    name: <service_account_installer_name>
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      version: ">1.11.1" 1
```

1 可选：指定所需版本范围是否大于 **1.11.1**。如需更多信息，请参阅"支持版本范围"。

创建或更新 CR 后，运行以下命令来应用配置文件：

命令语法

```
$ oc apply -f <extension_name>.yaml
```

2.2.2. 集群扩展的对象所有权

在 Operator Lifecycle Manager (OLM) v1 中，Kubernetes 对象一次只能由单个 **ClusterExtension** 对象所有。这样可确保 OpenShift Container Platform 集群中的对象一致管理，并防止尝试控制同一对象的多个集群扩展冲突。

2.2.2.1. 单个所有权

OLM v1 强制的核心所有权原则是每个对象只能有一个集群扩展作为其所有者。这可防止多个集群扩展重叠或冲突管理，确保每个对象仅与一个捆绑包关联。

单个所有权的影响

- 提供 **CustomResourceDefinition** (CRD) 对象的捆绑包只能安装一次。

  捆绑包提供 CRD 实例是 **ClusterExtension** 对象的一部分。这意味着您只能在集群中安装捆绑

捆绑包提供 CRD，它们是 **ClusterExtension** 对象的一部分。这意味着您只能在集群中安装捆绑包一次。尝试安装提供相同 CRD 的另一个捆绑包会导致失败，因为每个自定义资源只能有一个集群扩展作为其所有者。

- 集群扩展无法共享对象。
  OLM v1 的单所有者策略意味着集群扩展无法共享任何对象的所有权。如果一个集群扩展管理特定对象，如 **Deployment**、**CustomResourceDefinition** 或 **Service** 对象，则另一个集群扩展无法声明同一对象的所有权。尝试这样做会被 OLM v1 阻止。

2.2.2.2. 错误消息

当因为多个集群扩展试图管理同一对象而发生冲突时，Operator Controller 会返回一个错误消息，表示所有权冲突，如下所示：

错误信息示例

> CustomResourceDefinition 'logfilemetricexporters.logging.kubernetes.io' already exists in namespace 'kubernetes-logging' and cannot be managed by operator-controller

错误消息表示对象已经由另一个集群扩展管理，且无法重新分配或共享。

2.2.2.3. 注意事项

作为集群或扩展管理员，请查看以下注意事项：

捆绑包的唯一性

确保提供相同 CRD 的 Operator 捆绑包不会多次安装。这可以防止因为所有权冲突造成潜在的安装失败。

避免对象共享

如果您需要不同的集群扩展才能与类似的资源交互，请确保它们管理单独的对象。由于单所有者强制，集群扩展无法共同管理同一对象。

2.3. CATALOGD

Operator Lifecycle Manager (OLM) v1 使用 catalogd 组件及其资源来管理 Operator 和扩展目录。

2.3.1. 关于 OLM v1 中的目录

您可以使用 catalogd 组件查询 Kubernetes 扩展的目录，如 Operator 和控制器，从而发现可安装的内容。Catalogd 是一个 Kubernetes 扩展，为集群客户端解包目录内容，并是微服务的 Operator Lifecycle Manager (OLM) v1 套件的一部分。目前，catalogd 解包要打包并分发为容器镜像的目录内容。

其他资源

- 基于文件的目录
- 在集群中添加目录
- 红帽提供的目录

第 3 章 OPERATOR FRAMEWORK 常用术语表

以下与 Operator Framework 相关的术语，包括 Operator Lifecycle Manager (OLM) v1。

3.1. 捆绑包（BUNDLE）

在 Bundle Format 中，*捆绑包*是 Operator CSV、清单和元数据的集合。它们一起构成了可在集群中安装的 Operator 的唯一版本。

3.2. 捆绑包镜像

在 Bundle Format 中，*捆绑包镜像*是一个从 Operator 清单中构建的容器镜像，其中包含一个捆绑包。捆绑包镜像由 Open Container Initiative (OCI) spec 容器 registry 存储和发布，如 Quay.io 或 DockerHub。

3.3. 目录源

目录源 *catalog source* 代表 OLM 可查询的元数据存储，以发现和安装 Operator 及其依赖项。

3.4. CHANNEL

*频道*为 Operator 定义更新流，用于为订阅者推出更新。频道头指向该频道的最新版本。例如，**stable** 频道中会包含 Operator 的所有稳定版本，按由旧到新的顺序排列。

Operator 可以有几个频道，与特定频道绑定的订阅只会在该频道中查找更新。

3.5. 频道头

*频道头*是指特定频道中最新已知的更新。

3.6. 集群服务版本

集群服务版本（cluster service version，简称 CSV）是一个利用 Operator 元数据创建的 YAML 清单，可辅助 OLM 在集群中运行 Operator。它是 Operator 容器镜像附带的元数据，用于在用户界面填充徽标、描述和版本等信息。

此外，CSV 还是运行 Operator 所需的技术信息来源，类似于其需要的 RBAC 规则及其管理或依赖的自定义资源 (CR)。

3.7. 依赖项

Operator 可能会依赖于*集群*中存在的另一个 Operator。例如，Vault Operator 依赖于 etcd Operator 的数据持久性层。

OLM 通过确保在安装过程中在集群中安装 Operator 和 CRD 的所有指定版本来解决依赖关系。通过在目录中查找并安装满足所需 CRD API 且与软件包或捆绑包不相关的 Operator，解决这个依赖关系。

3.8. 扩展

扩展可让集群管理员在其 OpenShift Container Platform 集群上为用户扩展功能。扩展由 Operator Lifecycle Manager (OLM) v1 管理。

ClusterExtension API 简化了已安装的扩展的管理，其中包括通过 **registry+v1** 捆绑包格式的 Operator，通过将面向用户的 API 整合到单个对象中。管理员和 SRE 可使用 API 自动进程并使用 GitOps 原则定义所需状态。

3.9. 索引镜像

在 Bundle Format 中， *索引镜像*是一种数据库（数据库快照）镜像，其中包含关于 Operator 捆绑包（包括所有版本的 CSV 和 CRD）的信息。此索引可以托管集群中 Operator 的历史记录，并可使用 **opm** CLI 工具添加或删除 Operator 来加以维护。

3.10. 安装计划

*安装计划（install plan）*是一个列出了为自动安装或升级 CSV 而需创建的资源的计算列表。

3.11. 多租户

OpenShift Container Platform 中的 *租户* 是为一组部署的工作负载（通常由命名空间或项目表示）共享共同访问权限和特权的用户或组。您可以使用租户在不同的组或团队之间提供一定程度的隔离。

当集群由多个用户或组共享时，它被视为 *多租户* 集群。

3.12. OPERATOR

Operator 是一种打包、部署和管理 Kubernetes 应用程序的方法。Kubernetes 应用程序是一款 app，可在 Kubernetes 上部署，也可使用 Kubernetes API 和 **kubectl** 或 **oc** 工具进行管理。

Operator Lifecycle Manager (OLM) v1，**ClusterExtension** API 简化了已安装的扩展的管理，其中包括通过 **registry+v1** 捆绑包格式的 Operator，通过将面向用户的 API 整合到单个对象中。

3.13. OPERATOR 组

*Operator 组*将部署在同一命名空间中的所有 Operator 配置为 **OperatorGroup** 对象，以便在一系列命名空间或集群范围内监视其 CR。

3.14. 软件包

在 Bundle Format 中，*软件包*是一个目录，其中包含每个版本的 Operator 的发布历史记录。CSV 清单中描述了发布的 Operator 版本和 CRD。

3.15. 容器镜像仓库（REGISTRY）

Registry *是*一个存储了 Operator 捆绑包镜像的数据库，每个都包含所有频道的最新和历史版本。

3.16. SUBSCRIPTION

订阅（subscription） 通过跟踪软件包中的频道来保持 CSV 最新。

3.17. 更新图表

*更新图表*将 CSV 的版本关联到一起，与其他打包软件的更新图表类似。可以依次安装 Operator，也可以跳过某些版本。只有在添加新版本时，更新图表才会在频道头上扩大。

也称为*更新边缘（update edges）*或*更新路径（update paths）*。

第 4 章 目录

4.1. 基于文件的目录

OpenShift Container Platform 中的 Operator Lifecycle Manager (OLM) v1 支持 *基于文件的目录*来发现和源集群扩展，包括集群中的 Operator。

4.1.1. 亮点

*基于文件的目录*是 Operator Lifecycle Manager (OLM) 中目录格式的最新迭代。它是基于纯文本（JSON 或 YAML）和早期 SQLite 数据库格式的声明式配置演变，并且完全向后兼容。此格式的目标是启用 Operator 目录编辑、可组合性和可扩展性。

编辑

使用基于文件的目录，与目录内容交互的用户可以对格式进行直接更改，并验证其更改是否有效。由于这种格式是纯文本 JSON 或 YAML，因此目录维护人员可以通过手动或广泛支持的 JSON 或 YAML 工具（如 **jq** CLI）轻松操作目录元数据。

此可编辑功能启用以下功能和用户定义的扩展：

- 将现有捆绑包提升到新频道
- 更改软件包的默认频道
- 用于添加、更新和删除升级路径的自定义算法

Composability

基于文件的目录存储在任意目录层次结构中，从而启用目录组成。例如，考虑两个单独的基于文件的目录目录：**catalogA** 和 **catalogB**。目录维护人员可以通过生成新目录 **catalogC** 并将 **catalogA** 和 **catalogB** 复制到其中来创建新的组合目录。

这种可组合性支持分散的目录。格式允许 Operator 作者维护特定于 Operator 的目录，它允许维护人员轻松构建由单个 Operator 目录组成的目录。基于文件的目录可以通过组合多个其他目录、提取一个目录的子集或两者的组合来组成。

注意

不允许软件包中重复软件包和重复捆绑包。如果找到任何重复项，**opm validate** 命令将返回错误。

因为 Operator 作者最熟悉其 Operator、其依赖项及其升级兼容性，所以他们可以维护自己的 Operator 目录并直接控制其内容。对于基于文件的目录，Operator 作者负责在目录中构建和维护其软件包的任务。但是，复合目录维护者仅拥有在其目录中管理软件包并将目录发布到用户的任务。

可扩展性

基于文件的目录规格是目录的一个低级别表示。虽然目录维护器可以直接以低级形式维护，但目录维护人员可以在其自己的自定义工具上构建有趣的扩展，以供其自身的自定义工具用于实现任意数量的变异。

例如，工具可以将一个高级 API （如**(mode=semver)**） 转换为升级路径基于文件的低级别目录格式。或目录维护人员可能需要通过添加新属性到符合特定标准的捆绑包来自定义所有捆绑包元数据。

虽然这种可扩展性允许在低级别 API 上开发额外的官方工具，用于将来的 OpenShift Container Platform 版本，但目录维护人员也具有此功能。

4.1.2. 目录结构

基于文件的目录可从基于目录的文件系统进行存储和加载。**opm** CLI 通过遍历根目录并递归到子目录来加载目录。CLI 尝试加载它找到的每个文件，如果发生错误，则会失败。

可以使用 **.indexignore** 文件忽略非目录文件，这些文件对模式和优先级与 **.gitignore** 文件具有相同的规则。

示例 .indexignore 文件

```
# Ignore everything except non-object .json and .yaml files
**/*
!*.json
!*.yaml
**/objects/*.json
**/objects/*.yaml
```

目录维护人员具有选择所需的布局的灵活性，但建议将每个软件包基于文件的目录 Blob 存储在单独的子目录中。每个单独的文件可以是 JSON 或 YAML；目录中的每一文件并不需要使用相同的格式。

基本推荐结构

```
catalog
├── packageA
│   └── index.yaml
├── packageB
│   ├── .indexignore
│   ├── index.yaml
│   └── objects
│       └── packageB.v0.1.0.clusterserviceversion.yaml
└── packageC
    └── index.json
    └── deprecations.yaml
```

此推荐结构具有目录层次结构中的每个子目录都是自包含目录的属性，它使得目录组成、发现和导航简单文件系统操作。通过将目录复制到父目录的根目录，目录也可以包含在父目录中。

4.1.3. 模式

基于文件的目录使用基于 CUE 语言规范 的格式，该格式可使用任意模式进行扩展。以下 **_Meta** CUE 模式定义了所有基于文件的目录 Blob 必须遵循的格式：

_Meta 架构

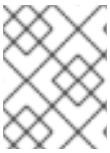
```
_Meta: {
  // schema is required and must be a non-empty string
  schema: string & !=""

  // package is optional, but if it's defined, it must be a non-empty string
  package?: string & !=""

  // properties is optional, but if it's defined, it must be a list of 0 or more properties
  properties?: [... #Property]
}
```

```
#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}
```



注意

此规格中列出的 CUE 模式不可被视为详尽模式。**opm validate** 命令具有额外的验证，很难或不可能在 CUE 中简洁地表达。

Operator Lifecycle Manager (OLM) 目录目前使用三种模式（**olm.package**、**olm.channel** 和 **olm.bundle**），它们对应于 OLM 的现有软件包和捆绑包概念。

目录中的每个 Operator 软件包都需要一个 **olm.package** blob、至少一个 **olm.channel** blob 以及一个或多个 **olm.bundle** blob。

注意

所有 **olm.*** 模式都为 OLM 定义的模式保留。自定义模式必须使用唯一前缀，如您拥有的域。

4.1.3.1. olm.package schema

olm.package 模式为 Operator 定义软件包级别的元数据。这包括其名称、描述、默认频道和图标。

例 4.1. olm.package schema

```
#Package: {
  schema: "olm.package"

  // Package name
  name: string & !=""

  // A description of the package
  description?: string

  // The package's default channel
  defaultChannel: string & !=""

  // An optional icon
  icon?: {
    base64data: string
    mediatype:  string
  }
}
```

4.1.3.2. olm.channel schema

olm.channel 模式在软件包中定义频道、属于频道成员的捆绑包条目，以及这些捆绑包的升级路径。

如果捆绑包条目代表多个 **olm.channel** blob 中的边缘，则每个频道只能显示一次。

它对条目的 **replaces** 值有效，以引用无法在此目录或其他目录中找到的另一捆绑包名称。但是，所有其他频道变量都必须为 true，比如频道没有多个磁头。

例 4.2. olm.channel schema

```
#Channel: {
  schema: "olm.channel"
  package: string & !=""
  name: string & !=""
  entries: [...#ChannelEntry]
}

#ChannelEntry: {
  // name is required. It is the name of an `olm.bundle` that
  // is present in the channel.
  name: string & !=""

  // replaces is optional. It is the name of bundle that is replaced
  // by this entry. It does not have to be present in the entry list.
  replaces?: string & !=""

  // skips is optional. It is a list of bundle names that are skipped by
  // this entry. The skipped bundles do not have to be present in the
  // entry list.
  skips?: [...string & !=""]

  // skipRange is optional. It is the semver range of bundle versions
  // that are skipped by this entry.
  skipRange?: string & !=""
}
```

警告

在使用 **skipRange** 字段时，跳过的 Operator 版本会从更新图中删除，因此不再可以被带有 **Subscription** 对象的 **spec.startingCSV** 属性的用户安装。

您可以使用 **skipRange** 和 **replaces** 字段以递增方式更新 Operator，同时保留以前安装的版本供用户使用。确保 **replaces** 字段指向相关的 Operator 版本前一个版本。

4.1.3.3. olm.bundle schema

例 4.3. olm.bundle schema

```
#Bundle: {
```

```
  schema: "olm.bundle"
  package: string & !=""
  name: string & !=""
  image: string & !=""
  properties: [...#Property]
  relatedImages?: [...#RelatedImage]
}

#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}

#RelatedImage: {
  // image is the image reference
  image: string & !=""

  // name is an optional descriptive name for an image that
  // helps identify its purpose in the context of the bundle
  name?: string & !=""
}
```

4.1.3.4. olm.deprecations schema

可选的 **olm.deprecations** 模式定义了目录中软件包、捆绑包和频道的弃用信息。Operator 作者可使用此模式向从目录运行这些 Operator 的用户提供与 Operator 相关的信息，如支持状态和推荐的升级路径。

当定义此模式时，OpenShift Container Platform Web 控制台会在 OperatorHub 的预安装页面上为 Operator 受影响元素显示警告徽标，包括任何自定义弃用信息。

olm.deprecations schema 条目包含一个或多个 **reference** 类型，这表示弃用范围。安装 Operator 后，可以在相关的 **Subscription** 对象上查看任何指定的信息作为状态条件。

表 4.1. 弃用的 reference 类型

类型	影响范围	状态条件
olm.package	代表整个软件包	**PackageDeprecated**
olm.channel	代表一个频道	**ChannelDeprecated**
olm.bundle	表示一个捆绑包版本	**BundleDeprecated**

每个 **reference** 类型都有自己的要求，如下例所示。

例 4.4. 带有每个 reference 类型的 olm.deprecations 模式示例

```
schema: olm.deprecations
```

```
package: my-operator 1
entries:
  - reference:
      schema: olm.package 2
    message: | 3
    The 'my-operator' package is end of life. Please use the
    'my-operator-new' package for support.
  - reference:
      schema: olm.channel
      name: alpha 4
    message: |
    The 'alpha' channel is no longer supported. Please switch to the
    'stable' channel.
  - reference:
      schema: olm.bundle
      name: my-operator.v1.68.0 5
    message: |
    my-operator.v1.68.0 is deprecated. Uninstall my-operator.v1.68.0 and
    install my-operator.v1.72.0 for support.
```

1. 每个弃用模式都必须有一个 **package** 值，且该软件包引用必须在目录间唯一。不能有关联的 **name** 字段。
2. **olm.package** 模式不得包含 **name** 字段，因为它由之前在 schema 中定义的 **package** 字段决定。
3. 所有 **message** 字段 （对于任何 **reference** 类型）都必须是一个非零长度，并以 opaque 文本 blob 表示。
4. **olm.channel** schema 的 **name** 字段是必需的。
5. **olm.bundle** schema 的 **name** 字段是必需的。

注意

弃用功能没有考虑重叠的弃用，例如，软件包与频道与捆绑包的比较。

Operator 作者可在与软件包的 **index.yaml** 文件相同的目录中将 **olm.deprecations** schema 条目保存为 **deprecations.yaml** 文件：

带有弃用的目录的目录结构示例

```
my-catalog
└── my-operator
    ├── index.yaml
    └── deprecations.yaml
```

其他资源

- 更新或过滤基于文件的目录镜像

4.1.4. Properties

属性是可附加到基于文件的目录方案的任意元数据片段。**type** 字段是一个有效指定 **value** 字段语义和语法含义的字符串。该值可以是任意 JSON 或 YAML。

OLM 定义几个属性类型，再次使用保留的 **olm.*** 前缀。

4.1.4.1. olm.package 属性

olm.package 属性定义软件包名称和版本。这是捆绑包上的必要属性，必须正好有一个这些属性。**packageName** 字段必须与捆绑包的 first-class **package** 字段匹配，并且 **version** 字段必须是有效的语义版本。

例 4.5. olm.package 属性

```
#PropertyPackage: {
  type: "olm.package"
  value: {
    packageName: string & !=""
    version: string & !=""
  }
}
```

4.1.4.2. olm.gvk 属性

olm.gvk 属性定义此捆绑包提供的 Kubernetes API 的 group/version/kind (GVK)。OLM 使用此属性解析捆绑包，作为列出与所需 API 相同的 GVK 的其他捆绑包的依赖项。GVK 必须遵循 Kubernetes GVK 验证。

例 4.6. olm.gvk 属性

```
#PropertyGVK: {
  type: "olm.gvk"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

4.1.4.3. olm.package.required

olm.package.required 属性定义此捆绑包需要的另一软件包的软件包名称和版本范围。对于捆绑包列表的每个所需软件包属性，OLM 确保集群中为列出的软件包和所需版本范围安装了一个 Operator。**versionRange** 字段必须是有效的语义版本（模拟）范围。

例 4.7. olm.package.required 属性

```
#PropertyPackageRequired: {
  type: "olm.package.required"
```

```
  value: {
    packageName: string & !=""
    versionRange: string & !=""
  }
}
```

4.1.4.4. olm.gvk.required

olm.gvk.required 属性定义此捆绑包需要的 Kubernetes API 的 group/version/kind (GVK)。对于捆绑包列表的每个必需的 GVK 属性，OLM 确保集群中安装了提供它的 Operator。GVK 必须遵循 Kubernetes GVK 验证。

例 4.8. olm.gvk.required 属性

```
#PropertyGVKRequired: {
  type: "olm.gvk.required"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

4.1.5. 目录示例

对于基于文件的目录，目录维护人员可以专注于 Operator 策展和兼容性。由于 Operator 作者已为其 Operator 创建了特定于 Operator 的目录，因此目录维护人员可以通过将每个 Operator 目录渲染到目录根目录的子目录来构建其目录。

构建基于文件的目录的方法有很多；以下步骤概述了一个简单的方法：

1. 为目录维护一个配置文件，其中包含目录中每个 Operator 的镜像引用：

   目录配置文件示例

   ```
   name: community-operators
   repo: quay.io/community-operators/catalog
   tag: latest
   references:
   - name: etcd-operator
     image: quay.io/etcd-operator/index@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03
   - name: prometheus-operator
     image: quay.io/prometheus-operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d84101eb317
   ```

2. 运行一个脚本，该脚本将解析配置文件并从其引用中创建新目录：

   脚本示例

```
name=$(yq eval '.name' catalog.yaml)
mkdir "$name"
yq eval '.name + "/" + .references[].name' catalog.yaml | xargs mkdir
for l in $(yq e '.name as $catalog | .references[] | .image + "|" + $catalog + "/" + .name +
"/index.yaml"' catalog.yaml); do
  image=$(echo $l | cut -d'|' -f1)
  file=$(echo $l | cut -d'|' -f2)
  opm render "$image" > "$file"
done
opm generate dockerfile "$name"
indexImage=$(yq eval '.repo + ":" + .tag' catalog.yaml)
docker build -t "$indexImage" -f "$name.Dockerfile" .
docker push "$indexImage"
```

4.1.6. 指南

在维护基于文件的目录时，请考虑以下准则。

4.1.6.1. 不可变捆绑包

Operator Lifecycle Manager (OLM) 的常规建议是捆绑包镜像及其元数据应视为不可变。

如果一个错误的捆绑包被推送到目录，您必须假设至少有一个用户已升级到该捆绑包。基于这种假设，您必须从损坏的捆绑包中发布另一个带有升级路径的捆绑包，以确保安装了有问题的捆绑包的用户收到升级。如果目录中更新了该捆绑包的内容，OLM 将不会重新安装已安装的捆绑包。

然而，在某些情况下首选更改目录元数据：

- 频道升级：如果您已发布了捆绑包，且之后决定将其添加到另一个频道，您可以在另一个 **olm.channel** blob 中添加捆绑包条目。
- 新的升级路径：如果您发布一个新的 **1.2.z** 捆绑包版本，如 **1.2.4**，但 **1.3.0** 已发布，您可以更新 **1.3.0** 的目录元数据以跳过 **1.2.4**。

4.1.6.2. 源控制

目录元数据应存储在源控制中，并被视为事实来源。目录镜像的更新应包括以下步骤：

1. 使用新的提交来更新源控制的目录目录。
2. 构建并推送目录镜像。使用一致的标记分类，如 **:latest** 或 **:<target_cluster_version>**，以便用户可以在目录可用时接收到更新。

4.1.7. CLI 用法

有关使用 **opm** CLI 创建基于文件的目录的说明，请参阅管理自定义目录。

有关管理基于文件的目录的 **opm** CLI 命令的参考文档，请参阅 CLI 工具。

4.1.8. 自动化

建议 Operator 作者和目录维护人员使用 CI/CD 工作流自动化其目录维护。目录维护人员可通过构建 GitOps 自动化以完成以下任务来进一步改进：

- 检查是否允许拉取请求 (PR) 作者进行请求的更改，例如更新其软件包的镜像引用。
- 检查目录更新是否通过 **opm validate** 命令。
- 检查是否有更新的捆绑包或目录镜像引用，目录镜像在集群中成功运行，来自该软件包的 Operator 可以成功安装。
- 自动合并通过之前检查的 PR。
- 自动重新构建和重新发布目录镜像。

4.2. 红帽提供的目录

红帽提供了一些默认包含在 OpenShift Container Platform 中的 Operator 目录。

4.2.1. 关于红帽提供的 Operator 目录

在 **openshift-marketplace** 命名空间中默认安装红帽提供的目录源，从而使目录在所有命名空间中都可用。

以下 Operator 目录由红帽发布：

目录	索引镜像	描述
redhat-operators	**registry.redhat.io/redhat/redhat-operator-index:v4.19**	已由红帽打包并提供的红帽产品。受红帽支持。
certified-operators	**registry.redhat.io/redhat/certified-operator-index:v4.19**	来自主要独立软件供应商 (ISV) 的产品。红帽与 ISV 合作打包并提供。受 ISV 支持。
community-operators	**registry.redhat.io/redhat/community-operator-index:v4.19**	由 redhat-openshift-ecosystem/community-operators-prod/operators GitHub 仓库中相关代表维护的软件。无官方支持。

在集群升级过程中，默认红帽提供的目录源的索引镜像标签由 Cluster Version Operator (CVO) 自动更新，以便 Operator Lifecycle Manager (OLM) 拉取目录的更新版本。例如，在从 OpenShift Container Platform 4.8 升级到 4.9 过程中，**redhat-operators** 目录的 **CatalogSource** 对象中的 **spec.image** 字段被更新：

```
registry.redhat.io/redhat/redhat-operator-index:v4.8
```

改为：

```
registry.redhat.io/redhat/redhat-operator-index:v4.9
```

4.3. 管理目录

集群管理员可向其集群添加 *目录*、或策展 Operator 和 Kubernetes 扩展集合。Operator 作者将其产品发布到这些目录中。当您向集群添加目录时，您可以访问发布到目录中的 Operator 和扩展的版本、补丁和无线更新。

您可以使用自定义资源 (CR) 从 CLI 中声明性管理目录和扩展。

*基于文件的目录*是 Operator Lifecycle Manager (OLM) 中目录格式的最新迭代。它是基于纯文本（JSON 或 YAML）和早期 SQLite 数据库格式的声明式配置演变，并且完全向后兼容。

重要

Kubernetes 定期弃用后续版本中删除的某些 API。因此，从使用删除 API 的 Kubernetes 版本的 OpenShift Container Platform 版本开始，Operator 无法使用删除 API 的 API。

4.3.1. 关于 OLM v1 中的目录

您可以使用 catalogd 组件查询 Kubernetes 扩展的目录，如 Operator 和控制器，从而发现可安装的内容。Catalogd 是一个 Kubernetes 扩展，为集群客户端解包目录内容，并是微服务的 Operator Lifecycle Manager (OLM) v1 套件的一部分。目前，catalogd 解包要打包并分发为容器镜像的目录内容。

其他资源

- 基于文件的目录

4.3.2. OLM v1 中红帽提供的 Operator 目录

Operator Lifecycle Manager (OLM) v1 默认在集群上包含以下红帽提供的 Operator 目录。如果要在集群中添加额外目录，请为目录创建一个自定义资源(CR)并将其应用到集群。以下自定义资源(CR)示例显示了集群中安装的默认目录。

Red Hat Operator 目录

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: openshift-redhat-operators
spec:
  priority: -100
  source:
    image:
      pollIntervalMinutes: <poll_interval_duration> 1
      ref: registry.redhat.io/redhat/redhat-operator-index:v4.19
    type: Image
```

1 指定轮询远程 registry 以获取较新的镜像摘要的时间间隔（以分钟为单位）。要禁用轮询，不要设置字段。

认证的 Operator 目录

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: openshift-certified-operators
```

```
spec:
priority: -200
  source:
    type: image
    image:
      pollIntervalMinutes: 10
      ref: registry.redhat.io/redhat/certified-operator-index:v4.19
    type: Image
```

Red Hat Marketplace 目录

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: openshift-redhat-marketplace
spec:
  priority: -300
  source:
    image:
      pollIntervalMinutes: 10
      ref: registry.redhat.io/redhat/redhat-marketplace-index:v4.19
    type: Image
```

社区 Operator 目录

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: openshift-community-operators
spec:
  priority: -400
  source:
    image:
      pollIntervalMinutes: 10
      ref: registry.redhat.io/redhat/community-operator-index:v4.19
    type: Image
```

以下命令在集群中添加目录：

命令语法

```
$ oc apply -f <catalog_name>.yaml 1
```

1 指定目录 CR，如 **my-catalog.yaml**。

4.3.3. 在集群中添加目录

要将目录添加到用于 Operator Lifecycle Manager (OLM) v1 使用量的集群，请创建一个 **ClusterCatalog** 自定义资源(CR)并将其应用到集群。

流程

1. 创建目录自定义资源(CR)，如下例所示：

my-redhat-operators.yaml 文件示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: my-redhat-operators (1)
spec:
  priority: 1000 (2)
  source:
    image:
      pollIntervalMinutes: 10 (3)
      ref: registry.redhat.io/redhat/community-operator-index:v4.19 (4)
    type: Image
```

(1) 当目录应用到集群时，该目录会使用 **metadata.name** 字段的值自动标记。有关标签和目录选择的更多信息，请参阅"目录内容解析"。

(2) 可选：指定与集群中的其他目录相关的目录优先级。如需更多信息，请参阅"按优先级选择目录"。

(3) 指定轮询远程 registry 以获取较新的镜像摘要的时间间隔（以分钟为单位）。要禁用轮询，不要设置字段。

(4) 在 **spec.source.image.ref** 字段中指定目录镜像。

2. 运行以下命令在集群中添加目录：

```
$ oc apply -f my-redhat-operators.yaml
```

输出示例

```
clustercatalog.olm.operatorframework.io/my-redhat-operators created
```

验证

- 运行以下命令以验证目录的状态：

  a. 运行以下命令检查目录是否可用：

  ```
  $ oc get clustercatalog
  ```

  输出示例

  ```
  NAME                     LASTUNPACKED   SERVING   AGE
  my-redhat-operators          55s          True      64s
  openshift-certified-operators   83m          True      84m
  openshift-community-operators   43m          True      84m
  openshift-redhat-marketplace    83m          True      84m
  openshift-redhat-operators      54m          True      84m
  ```

b. 运行以下命令，检查目录的状态：

```
$ oc describe clustercatalog my-redhat-operators
```

输出示例

```
Name:         my-redhat-operators
Namespace:
Labels:       olm.operatorframework.io/metadata.name=my-redhat-operators
Annotations:  <none>
API Version:  olm.operatorframework.io/v1
Kind:         ClusterCatalog
Metadata:
  Creation Timestamp:  2025-02-18T20:28:50Z
  Finalizers:
    olm.operatorframework.io/delete-server-cache
  Generation:        1
  Resource Version:  50248
  UID:               86adf94f-d2a8-4e70-895b-31139f2eeab7
Spec:
  Availability Mode:  Available
  Priority:           1000
  Source:
    Image:
      Poll Interval Minutes:  10
      Ref:                    registry.redhat.io/redhat/community-operator-index:v4.19
    Type:                     Image
Status: (1)
  Conditions:
    Last Transition Time:  2025-02-18T20:29:00Z
    Message:               Successfully unpacked and stored content from resolved source
    Observed Generation:   1
    Reason:                Succeeded (2)
    Status:                True
    Type:                  Progressing
    Last Transition Time:  2025-02-18T20:29:00Z
    Message:               Serving desired content from resolved source
    Observed Generation:   1
    Reason:                Available
    Status:                True
    Type:                  Serving
  Last Unpacked:           2025-02-18T20:28:59Z
  Resolved Source:
    Image:
      Ref:  registry.redhat.io/redhat/community-operator-
index@sha256:11627ea6fdd06b8092df815076e03cae9b7cede8b353c0b461328842d0289
6c5 (3)
    Type:   Image
  Urls:
    Base:  https://catalogd-service.openshift-catalogd.svc/catalogs/my-redhat-operators
Events:    <none>
```

(1) 描述目录的状态。

2 显示目录处于当前状态的原因。

3 显示目录的镜像引用。

4.3.4. 删除目录

您可以通过删除其自定义资源 (CR) 来删除目录。

先决条件

- 已安装目录。

流程

- 运行以下命令来删除目录：

```
$ oc delete clustercatalog <catalog_name>
```

输出示例

```
clustercatalog.olm.operatorframework.io "my-redhat-operators" deleted
```

验证

- 运行以下命令验证目录是否已删除：

```
$ oc get clustercatalog
```

4.3.5. 禁用默认目录

您可以禁用 OpenShift Container Platform 中包含的红帽提供的目录。

流程

- 运行以下命令来禁用默认目录：

```
$ oc patch clustercatalog openshift-certified-operators -p \
  '{"spec": {"availabilityMode": "Unavailable"}}' --type=merge
```

输出示例

```
clustercatalog.olm.operatorframework.io/openshift-certified-operators patched
```

验证

- 运行以下命令验证目录是否已禁用：

```
$ oc get clustercatalog openshift-certified-operators
```

输出示例

```
NAME                          LASTUNPACKED   SERVING   AGE
openshift-certified-operators                False     6h54m
```

4.4. 目录内容解析

当您指定要在自定义资源(CR)中安装的集群扩展时，Operator Lifecycle Manager (OLM) v1 使用目录选择来解析安装的内容。

您可以执行以下操作来控制目录内容的选择：

- 指定标签以选择目录。
- 使用匹配表达式在目录之间执行复杂的过滤。
- 设置目录优先级。

如果没有指定任何目录选择条件，Operator Lifecycle Manager (OLM) v1 从提供所需软件包的集群上任何可用目录选择一个扩展。

在解析过程中，默认情况下没有弃用的捆绑包优先于已弃用的捆绑包。

4.4.1. 按名称选择目录

当目录添加到集群中时，会使用目录自定义资源(CR)的 **metadata.name** 字段的值来创建标签。在扩展的 CR 中，您可以使用 **spec.source.catalog.selector.matchLabels** 字段指定目录名称。**matchLabels** 字段的值使用以下格式：

从 metadata.name 字段派生的标签示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
  labels:
    olm.operatorframework.io/metadata.name: <example_extension> 1
...
```

1 从 **metadata.name** 字段派生的标签，并在应用目录时自动添加。

以下示例从具有 **openshift-redhat-operators** 标签的目录解析 **<example_extension>-operator** 软件包：

扩展 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
spec:
  namespace: <example_namespace>
  serviceAccount:
    name: <example_extension>-installer
  source:
```

```
  sourceType: Catalog
  catalog:
    packageName: <example_extension>-operator
    selector:
      matchLabels:
        olm.operatorframework.io/metadata.name: openshift-redhat-operators
```

4.4.2. 根据标签或表达式选择目录

您可以使用集群目录的自定义资源(CR)中的标签将元数据添加到目录中。然后，您可以通过指定分配的标签或使用集群扩展 CR 中的表达式来过滤目录选择。

以下集群目录 CR 将值为 **true** 的 **example.com/support** 标签添加到 **catalog-a** 集群目录中：

带有标签的集群目录 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: catalog-a
  labels:
    example.com/support: "true"
spec:
  source:
    type: Image
    image:
      ref: quay.io/example/content-management-a:latest
```

以下集群扩展 CR 使用 **matchLabels** 选择器来选择带有 **example.com/support** 标签的目录，值为 **true**：

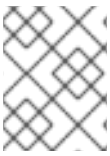
使用 matchLabels 选择器的集群扩展 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
spec:
  namespace: <example_namespace>
  serviceAccount:
    name: <example_extension>-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: <example_extension>-operator
      selector:
        matchLabels:
          example.com/support: "true"
```

您可以使用 **matchExpressions** 字段为标签执行更复杂的过滤。以下集群扩展 CR 选择带有 **example.com/support** 标签的目录，以及 **production** 或 **supported** 的值：

带有 matchExpression 选择器的集群扩展 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
spec:
  namespace: <example_namespace>
  serviceAccount:
    name: <example_extension>-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: <example_extension>-operator
      selector:
        matchExpressions:
          - key: example.com/support
            operator: In
            values:
              - "production"
              - "supported"
```



注意

如果同时使用 **matchLabels** 和 **matchExpressions** 字段，则所选目录必须满足所有指定标准。

4.4.3. 按标签或表达式进行目录排除

您可以在带有 **NotIn** 或 **DoesNotExist** 操作的元数据上使用 match 表达式来排除目录。

以下 CR 将 **example.com/testing** 标签添加到 **unwanted-catalog-1** 和 **unwanted-catalog-2** 集群目录中：

集群池 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: unwanted-catalog-1
  labels:
    example.com/testing: "true"
spec:
  source:
    type: Image
    image:
      ref: quay.io/example/content-management-a:latest
```

集群池 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: unwanted-catalog-2
  labels:
    example.com/testing: "true"
```

```
spec:
  source:
    type: Image
    image:
      ref: quay.io/example/content-management-b:latest
```

以下集群扩展 CR 从 **unwanted-catalog-1** 目录中排除选择：

排除一个特定目录的集群扩展 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
spec:
  namespace: <example_namespace>
  serviceAccount:
    name: <example_extension>-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: <example_extension>-operator
      selector:
        matchExpressions:
          - key: olm.operatorframework.io/metadata.name
            operator: NotIn
            values:
              - unwanted-catalog-1
```

以下集群扩展 CR 从没有 **example.com/testing** 标签的目录中进行选择。因此，**unwanted-catalog-1** 和 **unwanted-catalog-2** 都将重目录选择中排除。

使用特定标签排除目录的集群扩展 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <example_extension>
spec:
  namespace: <example_namespace>
  serviceAccount:
    name: <example_extension>-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: <example_extension>-operator
      selector:
        matchExpressions:
          - key: example.com/testing
            operator: DoesNotExist
```

4.4.4. 根据优先级选择目录

当多个目录提供相同的软件包时，您可以通过在每个目录的自定义资源(CR)中指定优先级来解决不确定性。如果未指定，则目录的默认优先级值为 **0**。优先级可以是任意正或负 32 位整数。

注意

- 在捆绑包解析过程中，将选择具有更高优先级值的目录，而不是优先级较低的目录。
- 未弃用的捆绑包优先于已弃用的捆绑包。
- 如果目录中存在多个具有相同优先级的捆绑包，并且目录选择是模糊的，则会打印错误。

具有较高优先级的集群目录 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: high-priority-catalog
spec:
  priority: 1000
  source:
    type: Image
    image:
      ref: quay.io/example/higher-priority-catalog:latest
```

具有较低优先级的集群目录 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterCatalog
metadata:
  name: lower-priority-catalog
spec:
  priority: 10
  source:
    type: Image
    image:
      ref: quay.io/example/lower-priority-catalog:latest
```

4.4.5. 目录选择错误故障排除

如果因为模糊或未选择目录而导致捆绑包解析失败，则会在集群扩展的 **status.conditions** 字段中输出错误消息。

执行以下操作排除目录选择错误：

- 使用标签或表达式重新定义您的选择标准。
- 调整目录优先级。
- 确保只有一个捆绑包与您的软件包名称和版本要求匹配。

4.5. 创建目录

目录维护人员可以使用基于文件的目录格式创建新目录，以用于 OpenShift Container Platform 上的 Operator Lifecycle Manager (OLM) v1。

4.5.1. 创建基于文件的目录镜像

您可以使用 **opm** CLI 创建一个目录镜像，它使用纯文本（*基于文件的目录*）格式（JSON 或 YAML），替换已弃用的 SQLite 数据库格式。

先决条件

- 已安装 **opm** CLI。
- 您有 **podman** 版本 1.9.3+。
- 已构建捆绑包镜像并推送到支持 Docker v2-2 的 registry。

流程

1. 初始化目录：

   a. 运行以下命令，为目录创建一个目录：

   ```
   $ mkdir <catalog_dir>
   ```

   b. 运行 **opm generate dockerfile** 命令生成可构建目录镜像的 Dockerfile：

   ```
   $ opm generate dockerfile <catalog_dir> \
       -i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4.19 (1)
   ```

   (1) 使用 **-i** 标志指定官方红帽基础镜像，否则 Dockerfile 使用默认的上游镜像。

   Dockerfile 必须与您在上一步中创建的目录目录位于相同的父目录中：

   目录结构示例

   ```
   . (1)
   ├──── <catalog_dir> (2)
   └──── <catalog_dir>.Dockerfile (3)
   ```

   (1) 父目录

   (2) Catalog 目录

   (3) **opm generate dockerfile** 命令生成的 Dockerfile

   c. 运行 **opm init** 命令，使用 Operator 的软件包定义填充目录：

   ```
   $ opm init <operator_name> \ (1)
       --default-channel=preview \ (2)
       --description=./README.md \ (3)
   ```

```
--icon=./operator-icon.svg \ 4
--output yaml \ 5
> <catalog_dir>/index.yaml 6
```

1. operator 或 package, name
2. 在未指定时该订阅默认到的频道
3. Operator 的 **README.md** 或者其它文档的路径
4. 到 Operator 图标的路径
5. 输出格式：JSON 或 YAML
6. 创建目录配置文件的路径

此命令在指定的目录配置文件中生成 **olm.package** 声明性配置 blob。

2. 运行 **opm render** 命令向目录添加捆绑包：

```
$ opm render <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
  --output=yaml \
  >> <catalog_dir>/index.yaml 2
```

1. 拉取捆绑包镜像的 spec
2. 目录配置文件的路径

注意

频道必须至少包含一个捆绑包。

3. 为捆绑包添加频道条目。例如，根据您的规格修改以下示例，并将其添加到 **<catalog_dir>/index.yaml** 文件中：

频道条目示例

```
---
schema: olm.channel
package: <operator_name>
name: preview
entries:
  - name: <operator_name>.v0.1.0 1
```

1. 确定在 **<operator_name>** 之后、版本 **v** 中包含句点 (**.**)。否则，条目无法传递 **opm validate** 命令。

4. 验证基于文件的目录：

   a. 针对目录目录运行 **opm validate** 命令：

```
$ opm validate <catalog_dir>
```

b. 检查错误代码是否为 **0**:

```
$ echo $?
```

输出示例

```
0
```

5. 运行 **podman build** 命令构建目录镜像：

```
$ podman build . \
   -f <catalog_dir>.Dockerfile \
   -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. 将目录镜像推送到 registry：

a. 如果需要，运行 **podman login** 命令与目标 registry 进行身份验证：

```
$ podman login <registry>
```

b. 运行 **podman push** 命令来推送目录镜像：

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

其他资源

- **opm** CLI reference

4.5.2. 更新或过滤基于文件的目录镜像

您可以使用 **opm** CLI 更新或过滤使用基于文件的目录格式的目录镜像。通过提取现有目录镜像的内容，您可以根据需要修改目录，例如：

- 添加软件包
- 删除软件包
- 更新现有软件包条目
- 详细说明每个软件包、频道和捆绑包的弃用信息

然后，您可以将镜像重新构建为目录的更新版本。

注意

或者，如果您已在镜像 registry 上已有目录镜像，您可以使用 oc-mirror CLI 插件在将其镜像到目标 registry 时自动从该目录镜像更新的源版本中修剪任何删除的镜像。

有关 oc-mirror 插件和此用例的更多信息，请参阅"更新您的镜像 registry 内容"部分，特别是"使用 oc-mirror 插件为断开连接的安装镜像镜像"部分。

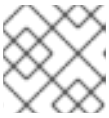
先决条件

- 在您的工作站上有以下内容：
  - **opm** CLI。
  - **podman** 版本 1.9.3+。
  - 基于文件的目录镜像。
  - 最近在与此目录相关的工作站上初始化的目录结构。
    如果您没有初始化的 catalog 目录，请创建目录并生成 Dockerfile。如需更多信息，请参阅"创建基于文件的目录镜像"中的"初始化目录"步骤。

流程

1. 以 YAML 格式将目录镜像的内容提取到 catalog 目录中的 **index.yaml** 文件中：

```
$ opm render <registry>/<namespace>/<catalog_image_name>:<tag> \
    -o yaml > <catalog_dir>/index.yaml
```



注意

或者，您可以使用 **-o json** 标志以 JSON 格式输出。

2. 将生成的 **index.yaml** 文件的内容修改为您的规格：

重要

在目录中发布捆绑包后，假设您安装了其中一个用户。确保之前发布目录中的所有捆绑包都具有到当前或更新频道头的更新路径，以避免安装该版本的用户。

- 要添加 Operator，请按照"创建基于文件的目录镜像"过程中创建软件包、捆绑包和频道条目的步骤进行操作。
- 要删除 Operator，请删除与软件包相关的 **olm.package**、**olm.channel** 和 **olm.bundle** blob 的集合。以下示例显示了一个需要删除的集合，才能从目录中删除 **example-operator** 软件包：

例 4.9. 删除条目示例

```
---
defaultChannel: release-2.7
icon:
  base64data: <base64_string>
  mediatype: image/svg+xml
name: example-operator
schema: olm.package
---
entries:
- name: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.0'
- name: example-operator.v2.7.1
  replaces: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.1'
```

```
- name: example-operator.v2.7.2
  replaces: example-operator.v2.7.1
  skipRange: '>=2.6.0 <2.7.2'
- name: example-operator.v2.7.3
  replaces: example-operator.v2.7.2
  skipRange: '>=2.6.0 <2.7.3'
- name: example-operator.v2.7.4
  replaces: example-operator.v2.7.3
  skipRange: '>=2.6.0 <2.7.4'
name: release-2.7
package: example-operator
schema: olm.channel
---
image: example.com/example-inc/example-operator-bundle@sha256:<digest>
name: example-operator.v2.7.0
package: example-operator
properties:
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyObject
    version: v1alpha1
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyOtherObject
    version: v1beta1
- type: olm.package
  value:
    packageName: example-operator
    version: 2.7.0
- type: olm.bundle.object
  value:
    data: <base64_string>
- type: olm.bundle.object
  value:
    data: <base64_string>
relatedImages:
- image: example.com/example-inc/example-related-image@sha256:<digest>
  name: example-related-image
schema: olm.bundle
---
```

- 要为 Operator 添加或更新弃用信息，请确保在与软件包的 **index.yaml** 文件相同的目录中有一个 **deprecations.yaml** 文件。有关 **deprecations.yaml** 文件格式的详情，请参考 "olm.deprecations schema"。

3. 保存您的更改。

4. 验证目录：

```
$ opm validate <catalog_dir>
```

5. 重建目录：

```
$ podman build . \
    -f <catalog_dir>.Dockerfile \
    -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. 将更新的目录镜像推送到 registry：

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

验证

1. 在 Web 控制台中，进入 **Administration → Cluster Settings → Configuration** 页面中的 OperatorHub 配置资源。
2. 添加目录源或更新现有目录源，以便将 pull spec 用于更新的目录镜像。
   如需更多信息，请参阅本节的"添加资源"中的"在集群中添加目录源"。
3. 在目录源处于 **READY** 状态后，进入 **Operators → OperatorHub** 页面，检查您所做的更改是否反映在 Operator 列表中。

其他资源

- Packaging format → Schemas → olm.deprecations schema
- 使用 oc-mirror 插件为断开连接的安装 mirror 镜像 → Keeping your mirror registry 内容已更新
- 在集群中添加目录源

4.6. OLM V1 中的断开连接的环境支持

为了支持需要实现高安全性而在没有与互联网连接的环境中的集群的管理员，特别是对于关键任务生产工作负载，Operator Lifecycle Manager (OLM) v1 包括集群扩展生命周期管理功能，从 OpenShift Container Platform 4.18 开始，可以在断开连接的环境中正常工作。

4.6.1. OLM v1 中的断开连接的支持和 oc-mirror 插件

从 OpenShift Container Platform 4.18 开始，Operator Lifecycle Manager (OLM) v1 支持断开连接的环境。在完全或部分断开连接的环境中为 OpenShift CLI (**oc**)使用 oc-mirror 插件将集群所需的镜像镜像到 mirror registry 后，OLM v1 可使用以下一组资源在这些环境中正常工作，具体取决于您使用的 oc-mirror 插件版本：

- **ImageContentSourcePolicy** 资源，由 oc-mirror 插件 v1 和 **ClusterCatalog** 资源自动生成，这些资源必须在使用 oc-mirror 插件 v1 后手动创建
- **ImageDigestMirrorSet**,**ImageTagMirrorSet**, 和 **ClusterCatalog** 资源，这些资源都由 oc-mirror 插件 v2 自动生成

注意

从 OpenShift Container Platform 4.18 开始，oc-mirror 插件 v2 是镜像的建议版本。

如需更多信息，请参阅您计划使用的 oc-mirror 插件版本的*断开连接的环境*指南。

其他资源

- 使用 oc-mirror 插件 v1 为断开连接的安装 mirror 镜像
- 使用 oc-mirror 插件 v2 为断开连接的安装 mirror 镜像

第 5 章 集群扩展

5.1. 管理集群扩展

将一个目录被添加到集群后，您可以访问发布到目录的扩展和 Operator 版本、补丁和无线更新。

您可以使用自定义资源(CR)从 CLI 声明性管理扩展。

注意

对于 OpenShift Container Platform 4.19，适用于 OLM v1 的流程都是基于 CLI 的。另外，管理员也可以使用普通方法（如 **Import YAML** 和 **Search** 页面）在 web 控制台中创建和查看相关对象。但是，现有的 **OperatorHub** 和 **Installed Operators** 页还不会显示 OLM v1 组件。

5.1.1. 支持的扩展

目前，Operator Lifecycle Manager (OLM) v1 支持安装满足以下所有条件的集群扩展：

- 扩展必须使用 OLM (Classic) 中引入的 **registry+v1** 捆绑包格式。
- 扩展必须通过 **AllNamespaces** 安装模式支持安装。
    - OpenShift Container Platform 4.19 中包含了对 **SingleNamespace** 和 **OwnNamespace** 安装模式的支持，作为技术预览功能。
- 扩展不能使用 Webhook。
- 扩展不能使用以下基于文件的目录属性声明依赖项：
    - **olm.gvk.required**
    - **olm.package.required**
    - **olm.constraint**

OLM v1 检查您要安装的扩展是否满足这些限制。如果要安装的扩展不符合这些限制，会在集群扩展条件中输出错误消息。

重要

Operator Lifecycle Manager (OLM) v1 不支持 OLM 中引入的 **OperatorConditions** API (Classic)。

如果扩展依赖于 **OperatorConditions** API 管理更新，扩展可能无法正确安装。大多数依赖此 API 的扩展都会在启动时失败，但在协调过程中可能会失败。

作为临时解决方案，您可以将扩展固定到特定的版本。当您想更新扩展时，请查阅扩展文档来查找何时安全将扩展固定到新版本。

其他资源

- Operator 条件

5.1.2. 从目录查找 Operator

在集群中添加目录后，您可以查询目录以查找要安装的 Operator 和扩展。

目前，在 Operator Lifecycle Manager (OLM) v1 中，您无法查询由 catalogd 管理的集群目录。在 OLM v1 中，您必须使用 **opm** 和 **jq** CLI 工具查询目录 registry。

先决条件

- 您已在集群中添加目录。
- 已安装 **jq** CLI 工具。
- 已安装 **opm** CLI 工具。

流程

1. 要返回支持 **AllNamespaces** 安装模式的扩展列表，且不使用 Webhook，请输入以下命令：

```
$ opm render <catalog_registry_url>:<tag> \
  | jq -cs '[.[] | select(.schema == "olm.bundle" \
  and (.properties[] | select(.type == "olm.csv.metadata").value.installModes[] \
  | select(.type == "AllNamespaces" and .supported == true)) \
  and .spec.webhookdefinitions == null) | .package] | unique[]'
```

其中：

catalog_registry_url

指定目录 registry 的 URL，如 **registry.redhat.io/redhat/redhat-operator-index**。

tag

指定目录的标签或版本，如 **v4.19** 或 **latest**。

例 5.1. 示例命令

```
$ opm render \
  registry.redhat.io/redhat/redhat-operator-index:v4.19 \
  | jq -cs '[.[] | select(.schema == "olm.bundle" \
  and (.properties[] | select(.type == "olm.csv.metadata").value.installModes[] \
  | select(.type == "AllNamespaces" and .supported == true)) \
  and .spec.webhookdefinitions == null) | .package] | unique[]'
```

例 5.2. 输出示例

```
"3scale-operator"
"amq-broker-rhel8"
"amq-online"
"amq-streams"
"amq-streams-console"
"ansible-automation-platform-operator"
"ansible-cloud-addons-operator"
"apicast-operator"
"authorino-operator"
"aws-load-balancer-operator"
```

```
"bamoe-kogito-operator"
"cephcsi-operator"
"cincinnati-operator"
"cluster-logging"
"cluster-observability-operator"
"compliance-operator"
"container-security-operator"
"cryostat-operator"
"datagrid"
"devspaces"
...
```

2. 运行以下命令，检查扩展的元数据内容：

```
$ opm render <catalog_registry_url>:<tag> \
  | jq -s '.[] | select( .schema == "olm.package") \
  | select( .name == "<package_name>")'
```

例 5.3. 示例命令

```
$ opm render \
  registry.redhat.io/redhat/redhat-operator-index:v4.19 \
  | jq -s '.[] | select( .schema == "olm.package") \
  | select( .name == "openshift-pipelines-operator-rh")'
```

例 5.4. 输出示例

```
{
  "schema": "olm.package",
  "name": "openshift-pipelines-operator-rh",
  "defaultChannel": "latest",
  "icon": {
    "base64data": "iVBORw0KGgoAAAANSUhE...",
    "mediatype": "image/png"
  }
}
```

5.1.2.1. 常见目录查询

您可以使用 **opm** 和 **jq** CLI 工具查询目录。下表显示了您可以在安装、更新和管理扩展生命周期时使用的通用目录查询。

命令语法

```
$ opm render <catalog_registry_url>:<tag> | <jq_request>
```

其中：

catalog_registry_url

指定目录 registry 的 URL，如 **registry.redhat.io/redhat/redhat-operator-index**。

tag

指定目录的标签或版本，如 **v4.19** 或 **latest**。

jq_request

指定您要在目录上运行的查询。

例 5.5. 示例命令

```
$ opm render \
  registry.redhat.io/redhat/redhat-operator-index:v4.19 \
  | jq -cs '[.[] | select(.schema == "olm.bundle" and (.properties[] \
  | select(.type == "olm.csv.metadata").value.installModes[] \
  | select(.type == "AllNamespaces" and .supported == true)) \
  and .spec.webhookdefinitions == null) \
  | .package] | unique[]'
```

表 5.1. 常见软件包查询

查询	Request（请求）
目录中的可用软件包	`$ opm render <catalog_registry_url>:<tag> \` `  | jq -s '.[] | select( .schema == "olm.package")'`

支持 **AllNamespaces** 安装模式且不使用 Webhook 的软件包	`$ opm render <catalog_registry_url>:<tag> \`
	`  | jq -cs '[.[] | select(.schema == "olm.bundle" and (.properties[] \`
	`  | select(.type == "olm.csv.metadata").value.installModes[] \`
	`  | select(.type == "AllNamespaces" and .supported == true)) \`
	`  and .spec.webhookdefinitions == null) \`
	`  | .package] | unique[]'`

软件包元数据	`$ opm render <catalog_registry_url>:<tag> \`
	`  | jq -s '.[] | select( .schema == "olm.package") \`
	`  | select( .name == "<package_name>")'`

软件包中的目录 Blob	`$ opm render <catalog_registry_url>:<tag> \` `  | jq -s '.[] | select( .package == "<package_name>")'`

表 5.2. 常见频道查询

查询	Request（请求）

软件包中的频道	$ opm render <catalog_registry_url>:<tag> \
	  | jq -s '.[] | select(.schema == "olm.channel") \
	  | select(.package == "<package_name>") | .name'

频道中的版本	$ opm render <catalog_registry_url>:<tag> \
	  | jq -s '.[] | select(.package == "<package_name>") \
	  | select(.schema == "olm.channel") \
	  | select(.name == "<channel_name>") .entries \
	  | .[] | .name'

• 频道中的最新版本
• 升级路径
	$ opm render <catalog_registry_url>:<tag> \
	  | jq -s '.[] | select(.schema == "olm.channel") \
	  | select (.name == "<channel_name>") \
	  | select(.package == "<package_name>")'

表 5.3. 常见捆绑包查询

查询	Request（请求）

软件包中的捆绑包	$ opm render <catalog_registry_url>:<tag> \
	  | jq -s '.[] | select(.schema == "olm.bundle") \
	  | select(.package == "<package_name>") | .name'

• 捆绑包依赖项
• 可用的 API
	$ opm render <catalog_registry_url>:<tag> \
	  | jq -s '.[] | select(.schema == "olm.bundle") \
	  | select (.name == "<bundle_name>") \
	  | select(.package == "<package_name>")'

5.1.3. 集群扩展权限

在 Operator Lifecycle Manager (OLM) Classic 中，具有集群管理员特权的单一服务帐户管理所有集群扩展。

OLM v1 被设计为默认比 OLM (Classic) 更安全。OLM v1 使用在扩展的自定义资源(CR)中指定的服务帐户管理集群扩展。集群管理员可以为每个集群扩展创建一个服务帐户。因此，管理员可以遵循最小特权的原则，只分配基于角色的访问控制(RBAC)来安装和管理该扩展。

您必须向集群角色或角色添加每个权限。然后，您必须使用集群角色绑定或角色绑定将集群角色或角色绑定到服务帐户。

您可以将 RBAC 限定为集群或命名空间。使用集群角色和集群角色绑定将权限范围到集群。使用角色和角色绑定将权限范围到命名空间。无论您将权限范围到集群还是命名空间取决于您要安装和管理的扩展设计。

重要

为简化以下步骤并改进可读性，以下示例清单使用范围到集群的权限。您可以进一步限制某些权限，方法是将它们限定到扩展名的命名空间，而不是集群。

如果已安装的扩展的新版本需要额外的权限，OLM v1 会停止更新过程，直到集群管理员授予这些权限。

5.1.3.1. 创建命名空间

在创建用于安装和管理集群扩展的服务帐户前，您必须创建一个命名空间。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

- 运行以下命令，为您要安装的扩展的服务帐户创建新命名空间：

```
$ oc adm new-project <new_namespace>
```

5.1.3.2. 为扩展创建服务帐户

您必须创建一个服务帐户来安装、管理和更新集群扩展。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 创建服务帐户，类似以下示例：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <extension>-installer
  namespace: <namespace>
```

例 5.6. extension-service-account.yaml 文件示例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-installer
  namespace: pipelines
```

2. 运行以下命令来应用服务帐户：

```
$ oc apply -f extension-service-account.yaml
```

5.1.3.3. 下载扩展的捆绑包清单

使用 **opm** CLI 工具下载您要安装的扩展的捆绑包清单。使用您选择的 CLI 工具或文本编辑器查看清单，并找到安装和管理扩展所需的权限。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 您已决定要安装的扩展。
- 已安装 **opm** CLI 工具。

流程

1. 运行以下命令，检查您要安装的扩展的可用版本和镜像：

```
$ opm render <registry_url>:<tag_or_version> | \
  jq -cs '.[] | select( .schema == "olm.bundle" ) | \
  select( .package == "<extension_name>") | \
  {"name":.name, "image":.image}'
```

例 5.7. 示例命令

```
$ opm render registry.redhat.io/redhat/redhat-operator-index:v4.19 | \
  jq -cs '.[] | select( .schema == "olm.bundle" ) | \
  select( .package == "openshift-pipelines-operator-rh") | \
  {"name":.name, "image":.image}'
```

例 5.8. 输出示例

```
{"name":"openshift-pipelines-operator-rh.v1.14.3","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:3f64b29f6903981470d0917b2557f49d84067bccdba0544bfe874ec4412f45b0"}
{"name":"openshift-pipelines-operator-rh.v1.14.4","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:dd3d18367da2be42539e5dde8e484dac3df33ba3ce1d5bcf896838954f3864ec"}
{"name":"openshift-pipelines-operator-rh.v1.14.5","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:f7b19ce26be742c4aaa458d37bc5ad373b5b29b20aaa7d308349687d3cbd8838"}
{"name":"openshift-pipelines-operator-rh.v1.15.0","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:22be152950501a933fe6e1df0e663c8056ca910a89dab3ea801c3bb2dc2bf1e6"}
{"name":"openshift-pipelines-operator-rh.v1.15.1","image":"registry.redhat.io/openshift-
```

```
pipelines/pipelines-operator-bundle@sha256:64afb32e3640bb5968904b3d1a317e9dfb307970f6fda0243e2018417207fd75"}
{"name":"openshift-pipelines-operator-rh.v1.15.2","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:8a593c1144709c9aeffbeb68d0b4b08368f528e7bb6f595884b2474bcfbcafcd"}
{"name":"openshift-pipelines-operator-rh.v1.16.0","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:a46b7990c0ad07dae78f43334c9bd5e6cba7b50ca60d3f880099b71e77bed214"}
{"name":"openshift-pipelines-operator-rh.v1.16.1","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:29f27245e93b3f605647993884751c490c4a44070d3857a878d2aee87d43f85b"}
{"name":"openshift-pipelines-operator-rh.v1.16.2","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:2037004666526c90329f4791f14cb6cc06e8775cb84ba107a24cc4c2cf944649"}
{"name":"openshift-pipelines-operator-rh.v1.17.0","image":"registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:d75065e999826d38408049aa1fde674cd1e45e384bfdc96523f6bad58a0e0dbc"}
```

2. 运行以下命令，创建一个目录来提取您要安装的捆绑包镜像：

```
$ mkdir <new_dir>
```

3. 运行以下命令来更改目录：

```
$ cd <new_dir>
```

4. 查找您要安装并运行以下命令的镜像引用：

```
$ oc image extract <full_path_to_registry_image>@sha256:<sha>
```

示例命令

```
$ oc image extract registry.redhat.io/openshift-pipelines/pipelines-operator-bundle@sha256:f7b19ce26be742c4aaa458d37bc5ad373b5b29b20aaa7d308349687d3cbd8838
```

5. 运行以下命令来更改 **manifests** 目录：

```
$ cd manifests
```

6. 输入以下命令来查看 manifests 目录的内容。输出中列出了安装、管理和操作扩展所需的资源清单。

```
$ tree
```

例 5.9. 输出示例

```
.
├── manifests
│   ├── config-logging_v1_configmap.yaml
│   ├── openshift-pipelines-operator-
monitor_monitoring.coreos.com_v1_servicemonitor.yaml
│   ├── openshift-pipelines-operator-prometheus-k8s-read-
binding_rbac.authorization.k8s.io_v1_rolebinding.yaml
│   ├── openshift-pipelines-operator-read_rbac.authorization.k8s.io_v1_role.yaml
│   ├── openshift-pipelines-operator-rh.clusterserviceversion.yaml
│   ├── operator.tekton.dev_manualapprovalgates.yaml
│   ├── operator.tekton.dev_openshiftpipelinesascodes.yaml
│   ├── operator.tekton.dev_tektonaddons.yaml
│   ├── operator.tekton.dev_tektonchains.yaml
│   ├── operator.tekton.dev_tektonconfigs.yaml
│   ├── operator.tekton.dev_tektonhubs.yaml
│   ├── operator.tekton.dev_tektoninstallersets.yaml
│   ├── operator.tekton.dev_tektonpipelines.yaml
│   ├── operator.tekton.dev_tektonresults.yaml
│   ├── operator.tekton.dev_tektontriggers.yaml
│   ├── tekton-config-defaults_v1_configmap.yaml
│   ├── tekton-config-observability_v1_configmap.yaml
│   ├── tekton-config-read-
rolebinding_rbac.authorization.k8s.io_v1_clusterrolebinding.yaml
│   ├── tekton-config-read-role_rbac.authorization.k8s.io_v1_clusterrole.yaml
│   ├── tekton-operator-controller-config-leader-election_v1_configmap.yaml
│   ├── tekton-operator-info_rbac.authorization.k8s.io_v1_rolebinding.yaml
│   ├── tekton-operator-info_rbac.authorization.k8s.io_v1_role.yaml
│   ├── tekton-operator-info_v1_configmap.yaml
│   ├── tekton-operator_v1_service.yaml
│   ├── tekton-operator-webhook-certs_v1_secret.yaml
│   ├── tekton-operator-webhook-config-leader-election_v1_configmap.yaml
│   ├── tekton-operator-webhook_v1_service.yaml
│   ├── tekton-result-read-
rolebinding_rbac.authorization.k8s.io_v1_clusterrolebinding.yaml
│   └── tekton-result-read-role_rbac.authorization.k8s.io_v1_clusterrole.yaml
├── metadata
│   ├── annotations.yaml
│   └── properties.yaml
└── root
    └── buildinfo
        ├── content_manifests
        │   └── openshift-pipelines-operator-bundle-container-v1.16.2-3.json
        └── Dockerfile-openshift-pipelines-pipelines-operator-bundle-container-v1.16.2-3
```

后续步骤

- 使用您首选的 CLI 工具或文本编辑器，查看 **manifests** 目录中集群服务版本(CSV)文件的 **install.spec.clusterpermissions** 小节的内容。以下示例引用了 Red Hat OpenShift Pipelines Operator 的 **openshift-pipelines-operator-rh.clusterserviceversion.yaml** 文件。
- 在以下流程中，保留此文件作为参考，同时为集群角色文件分配权限。

5.1.3.4. 安装和管理集群扩展所需的权限

您必须检查集群扩展的捆绑包镜像中包含的清单，以分配所需的权限。服务帐户需要足够的基于角色的访问控制(RBAC)来创建和管理以下资源。

重要

遵循最小特权和范围权限到特定资源名称的原则，且具有运行所需的最小 RBAC。

准入插件

因为 OpenShift Container Platform 集群使用 **OwnerReferencesPermissionEnforcement** 准入插件，集群扩展必须具有更新 **blockOwnerDeletion** 和 **ownerReferences** finalizers 的权限。

扩展的控制器的集群角色和集群角色绑定

您必须定义 RBAC，以便安装服务帐户可以为扩展控制器创建和管理集群角色和集群角色绑定。

集群服务版本(CSV)

您必须为集群扩展的 CSV 中定义的资源定义 RBAC。

集群范围的捆绑包资源

您必须定义 RBAC，以创建和管理捆绑包中包含的任何集群范围的资源。如果集群范围的资源与另一个资源类型匹配，如 **ClusterRole**，您可以在 **resources** 或 **resourceNames** 字段中将资源添加到预先存在的规则中。

自定义资源定义(CRD)

您必须定义 RBAC，以便安装服务帐户可以为扩展创建和管理 CRD。另外，您必须为 RBAC 的控制器授予服务帐户来管理其 CRD。

部署

您必须为安装服务帐户定义 RBAC，以创建和管理扩展控制器所需的部署，如服务和配置映射。

扩展权限

您必须包含 CSV 中定义的权限和集群权限的 RBAC。安装服务帐户需要能够向扩展控制器授予这些权限，因为扩展控制器需要这些权限才能正常运行。

命名空间范围的捆绑包资源

您必须为任何命名空间范围的捆绑包资源定义 RBAC。安装服务帐户需要相应的权限来创建和管理资源，如配置映射或服务。

角色和角色绑定

您必须为 CSV 中定义的任何角色或角色绑定定义 RBAC。安装服务帐户需要相应的权限来创建和管理这些角色和角色绑定。

服务帐户

您必须定义 RBAC，以便安装服务帐户可以为扩展控制器创建和管理服务帐户。

5.1.3.5. 为扩展创建集群角色

您必须查看集群服务版本(CSV)的 **install.spec.clusterpermissions** 小节，并仔细查看扩展的清单，以定义您要安装的扩展所需的基于角色的访问控制(RBAC)。您必须通过将所需的 RBAC 从 CSV 复制到新清单来创建集群角色。

提示

如果要测试在 OLM v1 中安装和更新扩展的过程，您可以使用以下集群角色授予集群管理员权限。此清单仅用于测试目的。它不应该在生产环境中使用。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <extension>-installer-clusterrole
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

以下流程使用 Red Hat OpenShift Pipelines Operator 的 **openshift-pipelines-operator-rh.clusterserviceversion.yaml** 文件作为示例。示例包括安装和管理 OpenShift Pipelines Operator 所需的 RBAC 摘录。有关完整的清单，请参阅"Red Hat OpenShift Pipelines Operator 的集群角色示例"。

重要

为简化以下步骤并改进可读性，以下示例清单使用范围到集群的权限。您可以进一步限制某些权限，方法是将它们限定到扩展名的命名空间，而不是集群。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 您已在要安装的扩展的镜像引用中下载了清单。

流程

1. 创建新集群角色清单，如下例所示：

   示例 <extension>-cluster-role.yaml 文件

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: <extension>-installer-clusterrole
   ```

2. 编辑集群角色清单使其包含在扩展上更新终结器的权限，如下例所示：

   示例 <extension>-cluster-role.yaml

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: pipelines-installer-clusterrole
   rules:
   - apiGroups:
     - olm.operatorframework.io
     resources:
     - clusterextensions/finalizers
   ```

```
verbs:
- update
# Scoped to the name of the ClusterExtension
resourceNames:
- <metadata_name> 1
```

1 指定来自扩展的自定义资源(CR)的 **metadata.name** 字段的值。

3. 在扩展的 CSV 文件中的 **rules.resources** 字段中搜索 **clusterrole** 和 **clusterrolebindings** 值。
   - 将 API 组、资源、操作动词和资源名称复制到您的清单中，如下例所示：

     集群角色清单示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
# ...
# ClusterRoles and ClusterRoleBindings for the controllers of the extension
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterroles
  verbs:
  - create 1
  - list
  - watch
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterroles
  verbs:
  - get
  - update
  - patch
  - delete
  resourceNames: 2
  - "*"
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  verbs:
  - get
  - update
```

```
  - patch
  - delete
  resourceNames:
  - "*"
# ...
```

1. 您不能将 **create**, **list**, 和 **watch** 权限限制到特定资源 (**resourceNames** 字段)。您需要将这些权限限定为它们的资源 (**resources** 字段)。

2. 有些资源名称使用以下格式生成：**<package_name>.<hash>**。安装扩展后，为扩展的控制器查找集群角色和集群角色绑定的资源名称。将此示例中的通配符字符替换为生成的名称，并遵循最小特权的原则。

4. 在扩展的 CSV 文件中的 **rules.resources** 字段中搜索 **customresourcedefinitions** 值。

   - 将 API 组、资源、操作动词和资源名称复制到您的清单中，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
# ...
# Custom resource definitions of the extension
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - get
  - update
  - patch
  - delete
  resourceNames:
  - manualapprovalgates.operator.tekton.dev
  - openshiftpipelinesascodes.operator.tekton.dev
  - tektonaddons.operator.tekton.dev
  - tektonchains.operator.tekton.dev
  - tektonconfigs.operator.tekton.dev
  - tektonhubs.operator.tekton.dev
  - tektoninstallersets.operator.tekton.dev
  - tektonpipelines.operator.tekton.dev
  - tektonresults.operator.tekton.dev
  - tektontriggers.operator.tekton.dev
# ...
```

5. 在 **rules.resources** spec 中搜索带有 **permissions** 和 **clusterPermissions** 值的小节。

- 将 API 组、资源、操作动词和资源名称复制到您的清单中，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
# ...
# Excerpt from install.spec.clusterPermissions
- apiGroups:
  - ''
  resources:
  - nodes
  - pods
  - services
  - endpoints
  - persistentvolumeclaims
  - events
  - configmaps
  - secrets
  - pods/log
  - limitranges
  verbs:
  - create
  - list
  - watch
  - delete
  - deletecollection
  - patch
  - get
  - update
- apiGroups:
  - extensions
  - apps
  resources:
  - ingresses
  - ingresses/status
  verbs:
  - create
  - list
  - watch
  - delete
  - patch
  - get
  - update
 # ...
```

6. 在 **install.spec.deployments** 小节中搜索资源的 CSV 文件。

- 将 API 组、资源、操作动词和资源名称复制到您的清单中，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
```

```
# ...
# Excerpt from install.spec.deployments
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - get
  - update
  - patch
  - delete
  # scoped to the extension controller deployment name
  resourceNames:
  - openshift-pipelines-operator
  - tekton-operator-webhook
# ...
```

7. 在扩展的 CSV 文件中的 **rules.resources** 字段中搜索 **services** 和 **configmaps** 值。

- 将 API 组、资源、操作动词和资源名称复制到您的清单中，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
# ...
# Services
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
  - update
  - patch
  - delete
  # scoped to the service name
  resourceNames:
  - openshift-pipelines-operator-monitor
```

```
  - tekton-operator
  - tekton-operator-webhook
 # configmaps
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
  - update
  - patch
  - delete
  # scoped to the configmap name
  resourceNames:
  - config-logging
  - tekton-config-defaults
  - tekton-config-observability
  - tekton-operator-controller-config-leader-election
  - tekton-operator-info
  - tekton-operator-webhook-config-leader-election
- apiGroups:
  - operator.tekton.dev
  resources:
  - tekton-config-read-role
  - tekton-result-read-role
  verbs:
  - get
  - watch
  - list
```

8. 运行以下命令，将集群角色清单添加到集群中：

```
$ oc apply -f <extension>-installer-clusterrole.yaml
```

示例命令

```
$ oc apply -f pipelines-installer-clusterrole.yaml
```

5.1.3.6. Red Hat OpenShift Pipelines Operator 的集群角色示例

如需 OpenShift Pipelines Operator 的完整集群角色清单，请参阅以下示例。

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: pipelines-installer-clusterrole
rules:
- apiGroups:
  - olm.operatorframework.io
  resources:
  - clusterextensions/finalizers
  verbs:
  - update
  # Scoped to the name of the ClusterExtension
  resourceNames:
  - pipes # the value from <metadata.name> from the extension's custom resource (CR)
# ClusterRoles and ClusterRoleBindings for the controllers of the extension
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterroles
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterroles
  verbs:
  - get
  - update
  - patch
  - delete
  resourceNames:
  - "*"
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  verbs:
  - get
  - update
  - patch
  - delete
  resourceNames:
  - "*"
# Extension's custom resource definitions
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
```

```
  - create
  - list
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - get
  - update
  - patch
  - delete
  resourceNames:
  - manualapprovalgates.operator.tekton.dev
  - openshiftpipelinesascodes.operator.tekton.dev
  - tektonaddons.operator.tekton.dev
  - tektonchains.operator.tekton.dev
  - tektonconfigs.operator.tekton.dev
  - tektonhubs.operator.tekton.dev
  - tektoninstallersets.operator.tekton.dev
  - tektonpipelines.operator.tekton.dev
  - tektonresults.operator.tekton.dev
  - tektontriggers.operator.tekton.dev
- apiGroups:
  - ''
  resources:
  - nodes
  - pods
  - services
  - endpoints
  - persistentvolumeclaims
  - events
  - configmaps
  - secrets
  - pods/log
  - limitranges
  verbs:
  - create
  - list
  - watch
  - delete
  - deletecollection
  - patch
  - get
  - update
- apiGroups:
  - extensions
  - apps
  resources:
  - ingresses
  - ingresses/status
  verbs:
  - create
  - list
  - watch
  - delete
```

```
  - patch
  - get
  - update
- apiGroups:
  - ''
  resources:
  - namespaces
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - apps
  resources:
  - deployments
  - daemonsets
  - replicasets
  - statefulsets
  - deployments/finalizers
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - monitoring.coreos.com
  resources:
  - servicemonitors
  verbs:
  - get
  - create
  - delete
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterroles
  - roles
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
  - bind
  - escalate
```

```
- apiGroups:
  - ''
  resources:
  - serviceaccounts
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
  - impersonate
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  - rolebindings
  verbs:
  - get
  - update
  - delete
  - patch
  - create
  - list
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  - customresourcedefinitions/status
  verbs:
  - get
  - create
  - update
  - delete
  - list
  - patch
  - watch
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - build.knative.dev
  resources:
  - builds
```

```
  - buildtemplates
  - clusterbuildtemplates
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - extensions
  resources:
  - deployments
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - extensions
  resources:
  - deployments/finalizers
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - operator.tekton.dev
  resources:
  - '*'
  - tektonaddons
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - tekton.dev
  - triggers.tekton.dev
  - operator.tekton.dev
  - pipelinesascode.tekton.dev
  resources:
  - '*'
  verbs:
```

```
  - add
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - dashboard.tekton.dev
  resources:
  - '*'
  - tektonaddons
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  verbs:
  - use
  - get
  - list
  - create
  - update
  - delete
- apiGroups:
  - events.k8s.io
  resources:
  - events
  verbs:
  - create
- apiGroups:
  - route.openshift.io
  resources:
  - routes
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - coordination.k8s.io
  resources:
```

```
  - leases
  verbs:
  - get
  - list
  - create
  - update
  - delete
  - patch
  - watch
- apiGroups:
  - console.openshift.io
  resources:
  - consoleyamlsamples
  - consoleclidownloads
  - consolequickstarts
  - consolelinks
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - delete
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  verbs:
  - delete
  - deletecollection
  - create
  - patch
  - get
  - list
  - update
  - watch
- apiGroups:
  - monitoring.coreos.com
  resources:
  - servicemonitors
  verbs:
  - delete
```

```
    - deletecollection
    - create
    - patch
    - get
    - list
    - update
    - watch
  - apiGroups:
    - batch
    resources:
    - jobs
    - cronjobs
    verbs:
    - delete
    - deletecollection
    - create
    - patch
    - get
    - list
    - update
    - watch
  - apiGroups:
    - ''
    resources:
    - namespaces/finalizers
    verbs:
    - update
  - apiGroups:
    - resolution.tekton.dev
    resources:
    - resolutionrequests
    - resolutionrequests/status
    verbs:
    - get
    - list
    - watch
    - create
    - delete
    - update
    - patch
  - apiGroups:
    - console.openshift.io
    resources:
    - consoleplugins
    verbs:
    - get
    - list
    - watch
    - create
    - delete
    - update
    - patch
  # Deployments specified in install.spec.deployments
  - apiGroups:
    - apps
    resources:
```

```
  - deployments
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - get
  - update
  - patch
  - delete
  # scoped to the extension controller deployment name
  resourceNames:
  - openshift-pipelines-operator
  - tekton-operator-webhook
# Service accounts in the CSV
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - create
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - get
  - update
  - patch
  - delete
  # scoped to the extension controller's deployment service account
  resourceNames:
  - openshift-pipelines-operator
# Services
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
  - update
  - patch
```

```
  - delete
  # scoped to the service name
  resourceNames:
  - openshift-pipelines-operator-monitor
  - tekton-operator
  - tekton-operator-webhook
# configmaps
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
  - update
  - patch
  - delete
  # scoped to the configmap name
  resourceNames:
  - config-logging
  - tekton-config-defaults
  - tekton-config-observability
  - tekton-operator-controller-config-leader-election
  - tekton-operator-info
  - tekton-operator-webhook-config-leader-election
- apiGroups:
  - operator.tekton.dev
  resources:
  - tekton-config-read-role
  - tekton-result-read-role
  verbs:
  - get
  - watch
  - list
---
```

5.1.3.7. 为扩展创建集群角色绑定

创建服务帐户和集群角色后，您必须将集群角色绑定到带有集群角色绑定清单的服务帐户。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 您已为要安装的扩展创建并应用以下资源：
  - Namespace
  - 服务帐户

- 集群角色

流程

1. 创建集群角色绑定将集群角色绑定绑定到服务帐户，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: <extension>-installer-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: <extension>-installer-clusterrole
subjects:
- kind: ServiceAccount
  name: <extension>-installer
  namespace: <namespace>
```

例 5.10. pipelines-cluster-role-binding.yaml 文件示例

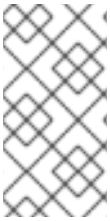
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pipelines-installer-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-installer-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-installer
  namespace: pipelines
```

2. 运行以下命令来应用集群角色绑定：

```
$ oc apply -f pipelines-cluster-role-binding.yaml
```

5.1.4. 在所有命名空间中安装集群扩展

您可以通过创建自定义资源 (CR) 并将其应用到集群来从目录安装扩展。Operator Lifecycle Manager (OLM) v1 支持安装集群扩展，包括 **registry+v1** 捆绑包格式的 OLM (Classic) Operator，它们仅限于集群。如需更多信息，请参阅*支持的扩展*。



注意

对于 OpenShift Container Platform 4.19，适用于 OLM v1 的流程都是基于 CLI 的。另外，管理员也可以使用普通方法（如 **Import YAML** 和 **Search** 页面）在 web 控制台中创建和查看相关对象。但是，现有的 **OperatorHub** 和 **Installed Operators** 页还不会显示 OLM v1 组件。

先决条件

- 您已创建了服务帐户，并分配了足够的基于角色的访问控制 (RBAC) 来安装、更新和管理您要安装的扩展。如需更多信息，请参阅"集群扩展权限"。

流程

1. 创建一个类似以下示例的 CR：

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <clusterextension_name>
spec:
  namespace: <installed_namespace> 1
  serviceAccount:
    name: <service_account_installer_name> 2
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      channels:
        - <channel_name> 3
      version: <version_or_version_range> 4
      upgradeConstraintPolicy: CatalogProvided 5
```

1. 指定您要安装捆绑包的命名空间，如 **pipelines** 或 **my-extension**。扩展仍然是集群范围的，可能包含在不同命名空间中安装的资源。
2. 指定您为安装、更新和管理扩展创建的服务帐户的名称。
3. 可选：将频道名称指定为数组，如 **pipelines-1.14** 或 **latest**。
4. 可选：指定您要安装的软件包的版本或版本范围，如 **1.14.0**、**1.14.x** 或 **>=1.16**。如需更多信息，请参阅"示例自定义资源(CR)指定目标版本"和"支持版本范围"。
5. 可选：指定升级约束策略。如果未指定，则默认设置为 **CatalogProvided**。只有在新版本满足软件包作者设置的升级限制时，**CatalogProvided** 设置才会更新。要强制更新或回滚，请将字段设置为 **SelfCertified**。如需更多信息，请参阅"设置更新或回滚"。

pipelines-operator.yaml CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: openshift-pipelines-operator-rh
      version: "1.14.x"
```

1. 运行以下命令，将 CR 应用到集群：

```
$ oc apply -f pipeline-operator.yaml
```

输出示例

```
clusterextension.olm.operatorframework.io/pipelines-operator created
```

验证

1. 运行以下命令，以 YAML 格式查看 Operator 或扩展 CR：

```
$ oc get clusterextension pipelines-operator -o yaml
```

例 5.11. 输出示例

```
apiVersion: v1
items:
- apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"olm.operatorframework.io/v1","kind":"ClusterExtension","metadata":
{"annotations":{},"name":"pipes"},"spec":{"namespace":"pipelines","serviceAccount":
{"name":"pipelines-installer"},"source":{"catalog":{"packageName":"openshift-pipelines-
operator-rh","version":"1.14.x"},"sourceType":"Catalog"}}}
    creationTimestamp: "2025-02-18T21:48:13Z"
    finalizers:
    - olm.operatorframework.io/cleanup-unpack-cache
    - olm.operatorframework.io/cleanup-contentmanager-cache
    generation: 1
    name: pipelines-operator
    resourceVersion: "72725"
    uid: e18b13fb-a96d-436f-be75-a9a0f2b07993
  spec:
    namespace: pipelines
    serviceAccount:
      name: pipelines-installer
    source:
      catalog:
        packageName: openshift-pipelines-operator-rh
        upgradeConstraintPolicy: CatalogProvided
        version: 1.14.x
      sourceType: Catalog
  status:
    conditions:
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: Deprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
```

```
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: PackageDeprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: ChannelDeprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: BundleDeprecated
    - lastTransitionTime: "2025-02-18T21:48:16Z"
      message: Installed bundle registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:f7b19ce26be742c4aaa458d37bc5ad373b5b29b20aaa7d308349687d3cbd
8838
        successfully
      observedGeneration: 1
      reason: Succeeded
      status: "True"
      type: Installed
    - lastTransitionTime: "2025-02-18T21:48:16Z"
      message: desired state reached
      observedGeneration: 1
      reason: Succeeded
      status: "True"
      type: Progressing
    install:
      bundle:
        name: openshift-pipelines-operator-rh.v1.14.5
        version: 1.14.5
kind: List
metadata:
  resourceVersion: ""
```

其中：

spec.channel

显示扩展 CR 中定义的频道。

spec.version

显示扩展 CR 中定义的版本或版本范围。

status.conditions

显示扩展状态和健康的信息。

type: Deprecated

显示以下一个或多个是否已弃用：

type: PackageDeprecated

显示解析的软件包是否已弃用。

type: ChannelDeprecated

显示解析的频道是否已弃用。

type: BundleDeprecated

显示解析捆绑包是否已弃用。

status 字段中的 **False** 值表示 **reason: Deprecated** 条件已弃用。**status** 字段中的 **True** 值表示 **reason: Deprecated** 条件已被弃用。

installedBundle.name

显示安装的捆绑包的名称。

installedBundle.version

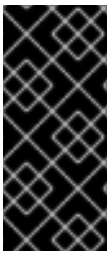
显示安装的捆绑包的版本。

5.1.5. 在特定命名空间中部署集群扩展（技术预览）

安装模式是 Operator Lifecycle Manager (OLM) Classic 的多租户功能。OLM v1 不支持多租户，并使用 **AllNamespaces** 安装模式将集群扩展部署到集群。

但是，一些现有集群扩展不支持 **AllNamespaces** 安装模式。您可以使用 **OwnNamespace** 或 **SingleNamespace** 安装模式作为 **registry+v1** Operator 捆绑包的技术预览功能在特定命名空间中部署扩展。

不支持 **MultiNamespace** 安装模式。因此，您无法在集群中多次安装同一 Operator。



重要

支持在特定命名空间中部署集群扩展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。

如需更多信息，请参阅"支持扩展"。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群
- 在集群中启用 **TechPreviewNoUpgrade** 功能集
- 支持 **OwnNamespace** 或 **SingleNamespace** 安装模式的 Operator

流程

1. 创建自定义资源(CR)，类似以下示例：

   示例 <cluster-extension-cr>.yaml 文件

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
```

```
metadata:
  name: <clusterextension_name>
  annotations:
    olm.operatorframework.io/watch-namespace: <namespace>
spec:
  namespace: <installed_namespace>
  serviceAccount:
    name: <service_account_installer_name>
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      channels:
        - <channel_name>
      version: <version_or_version_range>
      upgradeConstraintPolicy: CatalogProvided
```

其中：

namespace

指定您要部署集群扩展的命名空间。

- 如果 **namespace** 参数为空，或者注解不存在，则使用 **AllNamespaces** 安装模式部署扩展。
- 如果 **namespace** 参数的值与 **spec.namespace** 字段中的 **installed_namespace** 参数相同，则使用 **OwnNamespace** 安装模式部署扩展。
- 如果 **namespace** 参数指定与 **installed_namespace** 参数不同的命名空间，则使用 **SingleNamespace** 安装模式部署扩展。

2. 运行以下命令，将 CR 应用到集群：

```
$ oc apply -f <cluster_extension_cr>.yaml
```

其他资源

- 支持的扩展
- 项目和命名空间
- 创建一个服务帐户
- 指定目标版本的自定义资源(CR)示例
- 支持版本范围

5.1.6. 集群扩展的 preflight 权限检查（技术预览）

当您尝试安装扩展时，Operator Controller 会执行安装过程的空运行。此空运行验证指定的服务帐户是否可以执行安装扩展所需的所有操作。这包括在捆绑包中创建所有 Kubernetes 对象，以及捆绑包定义的角色和绑定的基于角色的访问控制(RBAC)规则。

重要

对集群扩展的 preflight 权限检查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。

如果服务帐户缺少任何所需的 RBAC 规则，preflight 检查会在实际安装进行前失败。如果 preflight 检查失败，Operator Controller 会在扩展的状态条件和 Operator Controller 日志中报告错误。

要继续安装，请更新角色和绑定，为服务帐户授予缺少的权限并应用更改。如果没有错误，Operator Controller 会协调更新的权限并完成安装。

5.1.6.1. preflight 权限检查的报告示例

以下报告表示服务帐户需要以下缺少的权限：

- 用于对整个集群核心 API 组中的 **services** 资源执行 **list** 和 **watch** 操作的 RBAC 规则
- 用于对 **pipelines** 命名空间的 **apps** API 组中的 **deployments** 资源执行 **create** 操作的 RBAC 规则。

您可以在集群扩展的状态条件中，从 preflight 权限检查报告。**oc describe clusterextension** 命令打印有关集群扩展的信息，包括状态条件。

示例命令

```
$ oc describe clusterextension <extension_name>
```

报告示例

```
apiVersion: v1
items:
- apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
...
Conditions:
  Type:    Progressing
  Status:  False
  Reason:  Retrying
  Message: pre-authorization failed: service account requires the following permissions to manage
cluster extension:
        Namespace:"" APIGroups:[] Resources:[services] Verbs:[list,watch]
        Namespace:"pipelines" APIGroups:["apps"] Resources:[deployments] Verbs:[create]
```

Namespace

在命名空间级别指定所需的 RBAC 规则范围，如 **pipelines** 命名空间。空命名空间值 **""** 表示您必须将权限范围到集群。

APIGroups

指定所需权限应用到的 API 组的名称。API 组中的空值（**[]**）表示权限应用到核心 API 组。例如，服务、secret 和配置映射都是核心资源。

如果资源属于命名 API 组，报告列出了方括号之间的名称。例如，**APIGroups:[apps]** 值表示扩展需要 RBAC 规则对 **apps** API 组中的资源执行操作。

Resources

指定需要权限的资源类型。例如，服务、secret 和自定义资源定义是常见的资源类型。

Verbs

指定服务帐户执行操作（或 *verbs*）所需的权限。如果报告列出了几个操作动词，则所有列出的操作动词都需要 RBAC 规则。

5.1.6.2. 常见权限错误

缺少操作动词

服务帐户没有执行所需操作的权限。要解决这个问题，请更新或创建角色和绑定来授予所需的权限。角色和角色绑定定义命名空间的资源权限。集群角色和集群角色绑定定义集群的资源权限。

权限升级

服务帐户没有足够的权限来创建扩展所需的角色或集群角色。当发生这种情况时，preflight 检查会报告缺少动词，以防止特权升级。要解决这个问题，请为服务帐户授予足够的权限，使其可以创建角色。

缺少角色引用

扩展引用 Operator Controller 无法找到的角色或集群角色。当发生这种情况时，preflight 检查会列出缺少的角色，并报告 **授权评估错误**。要解决这个问题，请创建或更新角色和集群角色，以确保所有角色引用都存在。

5.1.7. 更新集群扩展

您可以通过手动编辑自定义资源 (CR) 并应用更改来更新集群扩展或 Operator。

先决条件

- 已安装 Operator 或扩展。
- 已安装 **jq** CLI 工具。
- 已安装 **opm** CLI 工具。

流程

1. 通过完成以下步骤，检查目录文件本地副本中的频道和版本信息：
   a. 运行以下命令，从所选软件包中获取频道列表：

```
$ opm render <catalog_registry_url>:<tag> \
  | jq -s '.[] | select( .schema == "olm.channel" ) \
  | select( .package == "openshift-pipelines-operator-rh") | .name'
```

例 5.12. 示例命令

```
$ opm render registry.redhat.io/redhat/redhat-operator-index:v4.19 \
  | jq -s '.[] | select( .schema == "olm.channel" ) \
  | select( .package == "openshift-pipelines-operator-rh") | .name'
```

例 5.13. 输出示例

```
"latest"
"pipelines-1.14"
"pipelines-1.15"
"pipelines-1.16"
"pipelines-1.17"
```

b. 运行以下命令，获取频道中发布的版本列表：

```
$ opm render <catalog_registry_url>:<tag> \
  | jq -s '.[] | select( .package == "<package_name>" ) \
  | select( .schema == "olm.channel" ) \
  | select( .name == "<channel_name>" ) | .entries \
  | .[] | .name'
```

例 5.14. 示例命令

```
$ opm render registry.redhat.io/redhat/redhat-operator-index:v4.19 \
  | jq -s '.[] | select( .package == "openshift-pipelines-operator-rh" ) \
  | select( .schema == "olm.channel" ) | select( .name == "latest" ) \
  | .entries | .[] | .name'
```

例 5.15. 输出示例

```
"openshift-pipelines-operator-rh.v1.15.0"
"openshift-pipelines-operator-rh.v1.16.0"
"openshift-pipelines-operator-rh.v1.17.0"
"openshift-pipelines-operator-rh.v1.17.1"
```

2. 运行以下命令，查找在 Operator 或扩展 CR 中指定哪个版本或频道：

```
$ oc get clusterextension <operator_name> -o yaml
```

示例命令

```
$ oc get clusterextension pipelines-operator -o yaml
```

例 5.16. 输出示例

```
apiVersion: v1
items:
- apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"olm.operatorframework.io/v1","kind":"ClusterExtension","metadata":
```

{"annotations":{},"name":"pipes"},"spec":{"namespace":"pipelines","serviceAccount":{"name":"pipelines-installer"},"source":{"catalog":{"packageName":"openshift-pipelines-operator-rh","version":"1.14.x"},"sourceType":"Catalog"}}}

```
    creationTimestamp: "2025-02-18T21:48:13Z"
    finalizers:
    - olm.operatorframework.io/cleanup-unpack-cache
    - olm.operatorframework.io/cleanup-contentmanager-cache
    generation: 1
    name: pipelines-operator
    resourceVersion: "72725"
    uid: e18b13fb-a96d-436f-be75-a9a0f2b07993
  spec:
    namespace: pipelines
    serviceAccount:
      name: pipelines-installer
    source:
      catalog:
        packageName: openshift-pipelines-operator-rh
        upgradeConstraintPolicy: CatalogProvided
        version: 1.14.x
      sourceType: Catalog
  status:
    conditions:
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: Deprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: PackageDeprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: ChannelDeprecated
    - lastTransitionTime: "2025-02-18T21:48:13Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: BundleDeprecated
    - lastTransitionTime: "2025-02-18T21:48:16Z"
      message: Installed bundle registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:f7b19ce26be742c4aaa458d37bc5ad373b5b29b20aaa7d308349687d3cbd
8838
        successfully
      observedGeneration: 1
      reason: Succeeded
      status: "True"
      type: Installed
```

```
    - lastTransitionTime: "2025-02-18T21:48:16Z"
      message: desired state reached
      observedGeneration: 1
      reason: Succeeded
      status: "True"
      type: Progressing
    install:
      bundle:
        name: openshift-pipelines-operator-rh.v1.14.5
        version: 1.14.5
kind: List
metadata:
  resourceVersion: ""
```

3. 使用以下方法之一编辑 CR：

- 如果要将 Operator 或扩展固定到特定版本，如 **1.15.0**，请编辑类似以下示例的 CR：

  pipelines-operator.yaml CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: openshift-pipelines-operator-rh
      version: "1.15.0" (1)
```

  (1) 将版本从 **1.14.x** 更新至 **1.15.0**

- 如果要定义可接受的更新版本范围，请编辑类似以下示例的 CR：

  指定了版本范围的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: openshift-pipelines-operator-rh
      version: ">1.15, <1.17" (1)
```

-

1 指定所需的版本范围，大于 **1.15** 版本，但小于 **1.17**。如需更多信息，请参阅"支持版本范围"和"Version 比较字符串"。

- 如果要更新到可以从频道解析的最新版本，请编辑类似以下示例的 CR：

  带有指定频道的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: openshift-pipelines-operator-rh
      channels:
        - latest 1
```

1 安装可从指定频道解析的最新版本。对频道的更新会自动安装。输入值作为数组。

- 如果要指定频道和版本范围，请编辑类似以下示例的 CR：

  带有指定频道和版本范围的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    sourceType: Catalog
    catalog:
      packageName: openshift-pipelines-operator-rh
      channels:
        - latest
      version: "<1.16"
```

  如需更多信息，请参阅"指定目标版本的示例自定义资源(CR) "。

4. 运行以下命令，将更新应用到集群：

```
$ oc apply -f pipelines-operator.yaml
```

输出示例

-

```
clusterextension.olm.operatorframework.io/pipelines-operator configured
```

验证

- 运行以下命令验证频道和版本更新是否已应用：

```
$ oc get clusterextension pipelines-operator -o yaml
```

例 5.17. 输出示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"olm.operatorframework.io/v1","kind":"ClusterExtension","metadata":
{"annotations":{},"name":"pipes"},"spec":{"namespace":"pipelines","serviceAccount":
{"name":"pipelines-installer"},"source":{"catalog":{"packageName":"openshift-pipelines-
operator-rh","version":"\u003c1.16"},"sourceType":"Catalog"}}}
  creationTimestamp: "2025-02-18T21:48:13Z"
  finalizers:
  - olm.operatorframework.io/cleanup-unpack-cache
  - olm.operatorframework.io/cleanup-contentmanager-cache
  generation: 2
  name: pipes
  resourceVersion: "90693"
  uid: e18b13fb-a96d-436f-be75-a9a0f2b07993
spec:
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    catalog:
      packageName: openshift-pipelines-operator-rh
      upgradeConstraintPolicy: CatalogProvided
      version: <1.16
    sourceType: Catalog
status:
  conditions:
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: Deprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: PackageDeprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
```

```
    status: "False"
    type: ChannelDeprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: BundleDeprecated
  - lastTransitionTime: "2025-02-18T21:48:16Z"
    message: Installed bundle registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:8a593c1144709c9aeffbeb68d0b4b08368f528e7bb6f595884b2474bcfbcafcd
      successfully
    observedGeneration: 2
    reason: Succeeded
    status: "True"
    type: Installed
  - lastTransitionTime: "2025-02-18T21:48:16Z"
    message: desired state reached
    observedGeneration: 2
    reason: Succeeded
    status: "True"
    type: Progressing
  install:
    bundle:
      name: openshift-pipelines-operator-rh.v1.15.2
      version: 1.15.2
```

故障排除

- 如果您指定已弃用或不存在的目标版本或频道，您可以运行以下命令来检查扩展的状态：

```
$ oc get clusterextension <operator_name> -o yaml
```

例 5.18. 不存在版本的输出示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"olm.operatorframework.io/v1","kind":"ClusterExtension","metadata":
{"annotations":{},"name":"pipes"},"spec":{"namespace":"pipelines","serviceAccount":
{"name":"pipelines-installer"},"source":{"catalog":{"packageName":"openshift-pipelines-
operator-rh","version":"9.x"},"sourceType":"Catalog"}}}
  creationTimestamp: "2025-02-18T21:48:13Z"
  finalizers:
  - olm.operatorframework.io/cleanup-unpack-cache
  - olm.operatorframework.io/cleanup-contentmanager-cache
  generation: 3
  name: pipes
  resourceVersion: "93334"
  uid: e18b13fb-a96d-436f-be75-a9a0f2b07993
spec:
```

```
  namespace: pipelines
  serviceAccount:
    name: pipelines-installer
  source:
    catalog:
      packageName: openshift-pipelines-operator-rh
      upgradeConstraintPolicy: CatalogProvided
      version: 9.x
    sourceType: Catalog
status:
  conditions:
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: Deprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: PackageDeprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: ChannelDeprecated
  - lastTransitionTime: "2025-02-18T21:48:13Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: BundleDeprecated
  - lastTransitionTime: "2025-02-18T21:48:16Z"
    message: Installed bundle registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:8a593c1144709c9aeffbeb68d0b4b08368f528e7bb6f595884b2474bcfbcafc
d
      successfully
    observedGeneration: 3
    reason: Succeeded
    status: "True"
    type: Installed
  - lastTransitionTime: "2025-02-18T21:48:16Z"
    message: 'error upgrading from currently installed version "1.15.2": no bundles
      found for package "openshift-pipelines-operator-rh" matching version "9.x"'
    observedGeneration: 3
    reason: Retrying
    status: "True"
    type: Progressing
  install:
    bundle:
      name: openshift-pipelines-operator-rh.v1.15.2
      version: 1.15.2
```

其他资源

- 更新路径

5.1.8. 删除 Operator

您可以通过删除 **ClusterExtension** 自定义资源 (CR) 来删除 Operator 及其自定义资源定义 (CRD)。

先决条件

- 已安装目录。
- 已安装 Operator。

流程

- 运行以下命令来删除 Operator 及其 CRD：

```
$ oc delete clusterextension <operator_name>
```

输出示例

```
clusterextension.olm.operatorframework.io "<operator_name>" deleted
```

验证

- 运行以下命令验证您的 Operator 及其资源已被删除：
  - 运行以下命令验证 Operator 已被删除：

```
$ oc get clusterextensions
```

输出示例

```
No resources found
```

  - 运行以下命令验证 Operator 的系统命名空间是否已删除：

```
$ oc get ns <operator_name>-system
```

输出示例

```
Error from server (NotFound): namespaces "<operator_name>-system" not found
```

5.2. 用户访问扩展资源

在安装集群扩展并由 Operator Lifecycle Manager (OLM) v1 管理后，扩展通常可提供 **CustomResourceDefinition** 对象(CRD)，用于在集群中公开新的 API 资源。默认情况下，集群管理员通常对这些资源具有完全的管理访问权限，而非集群管理员用户或*常规用户*可能缺少足够的权限。

OLM v1 不会为常规用户自动配置或管理基于角色的访问控制(RBAC)，以与安装的扩展所提供的 API 进行交互。集群管理员需要自己定义所需的 RBAC 策略，以便为这样的用户创建、查看或编辑这些自定义资源(CR)。

注意

为用户访问扩展资源描述的 RBAC 权限与必须添加到服务帐户的权限不同，以启用集群扩展本身的基于 OLM v1- 的初始安装。有关安装扩展时的 RBAC 要求的更多信息，请参阅"管理扩展"中的"集群扩展权限"。

其他资源

- "Managing extensions" → "Cluster extension permissions"

5.2.1. 用户的常见默认集群角色

安装集群扩展可能包含默认集群角色，以确定常规用户到扩展提供的 API 资源的基于角色的访问控制(RBAC)。常见集群角色组类似以下策略：

view 集群角色

授予集群中指定 API 资源的所有自定义资源(CR)对象的只读访问权限。适用于需要了解资源的普通用户而无需任何权限修改它们。非常适合监控目的和有限访问查看。

edit 集群角色

允许用户修改集群中的所有 CR 对象。让用户能够创建、更新和删除资源，使其适合必须管理资源的团队成员，但不应控制 RBAC 或管理其他用户的权限。

admin 集群角色

通过集群中指定 API 资源的所有自定义资源对象，提供完整的权限，包括 **create**, **update**, 和 **delete** 操作动词。

其他资源

- 面向用户的角色 (Kubernetes 文档)

5.2.2. 查找集群扩展公开的 API 组和资源

要创建适当的 RBAC 策略来授予用户对集群扩展资源的访问权限，您必须知道哪些 API 组和资源由已安装的扩展公开。作为管理员，您可以使用 OpenShift CLI (**oc**) 检查集群中安装的自定义资源定义 (CRD)。

先决条件

- 在集群中安装了集群扩展。

流程

- 要在指定以特定集群扩展为目标的标签选择器来列出已安装的 CRD，以只查找该扩展拥有的 CRD，请运行以下命令：

```
$ oc get crds -l 'olm.operatorframework.io/owner-kind=ClusterExtension,olm.operatorframework.io/owner-name=<cluster_extension_name>'
```

- 另外，您还可以搜索所有安装的 CRD，并通过 CRD 名称单独检查它们：

a. 运行以下命令，列出当前在集群中安装的所有可用自定义资源定义(CRD)：

```
$ oc get crds
```

在输出中找到您要查找的 CRD。

b. 运行以下命令，进一步检查单个 CRD 以查找其 API 组：

```
$ oc get crd <crd_name> -o yaml
```

5.2.3. 使用自定义角色绑定授予用户对扩展资源的访问权限

作为集群管理员，您可以手动创建和配置基于角色的访问控制(RBAC)策略，以使用自定义角色绑定授予用户对扩展资源的访问权限。

先决条件

- 在集群中安装了集群扩展。
- 您有一个 API 组和资源名称的列表，如"查找由集群扩展公开的 API 组和资源中所述。

流程

1. 如果安装集群扩展不提供默认集群角色，请手动创建一个或多个角色：

   a. 考虑 "Common default cluster roles for users" 中描述的角色集合的用例。
   例如，创建以下 ClusterRole 对象定义中的一个或多个 **ClusterRole** 对象定义，将 **<cluster_extension_api_group>** 和 **<cluster_extension_custom_resource>** 替换为安装集群扩展提供的实际 API 组和资源名称：

   view-custom-resource.yaml 文件示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: view-custom-resource
rules:
- apiGroups:
  - <cluster_extension_api_group>
  resources:
  - <cluster_extension_custom_resources>
  verbs:
  - get
  - list
  - watch
```

   edit-custom-resource.yaml 文件示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: edit-custom-resource
rules:
- apiGroups:
```

```
  - <cluster_extension_api_group>
  resources:
  - <cluster_extension_custom_resources>
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

admin-custom-resource.yaml 文件示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: admin-custom-resource
rules:
- apiGroups:
  - <cluster_extension_api_group>
  resources:
  - <cluster_extension_custom_resources>
  verbs:
  - '*' (1)
```

(1) 在 **verbs** 中设置通配符 (*) 将允许对指定资源的所有操作。

b. 运行以下命令，为您创建的任何 YAML 文件创建集群角色：

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

2. 将集群角色关联到特定的用户或组，通过将集群角色绑定到单独的用户或组名称来授予其资源所需的权限：

a. 创建一个对象定义：对于*集群角色绑定*，授予对所有命名空间的访问权限；对于*角色绑定*，授予对一个特定命名空间的访问权限。：

- 以下示例集群角色绑定向在所有命名空间中授予自定义资源的只读 **view** 访问权限：

用户的 ClusterRoleBinding 对象示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: view-custom-resource-binding
subjects:
- kind: User
  name: <user_name>
roleRef:
  kind: ClusterRole
  name: view-custom-resource
  apiGroup: rbac.authorization.k8s.io
```

用户的 ClusterRoleBinding 对象示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: view-custom-resource-binding
subjects:
- kind: Group
  name: <group_name>
roleRef:
  kind: ClusterRole
  name: view-custom-resource
  apiGroup: rbac.authorization.k8s.io
```

- 以下角色绑定将 **edit** 权限限制到一个特定命名空间：

用户的 RoleBinding 对象示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: edit-custom-resource-edit-binding
  namespace: <namespace>
subjects:
- kind: User
  name: <username>
roleRef:
  kind: Role
  name: custom-resource-edit
  apiGroup: rbac.authorization.k8s.io
```

b. 将对象定义保存到 YAML 文件中。

c. 运行以下命令来创建对象：

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

5.2.4. 使用聚合集群角色授予用户对扩展资源的访问权限

作为集群管理员，您可以配置基于角色的访问控制(RBAC)策略，以使用聚合集群角色授予用户对扩展资源的访问权限。

要自动扩展现有的默认集群角色，您可以通过将以下一个或多个标签添加到 **ClusterRole** 对象来添加*聚合标签*：

ClusterRole 对象中的聚合标签

```
# ..
metadata:
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin: "true"
```

```
    rbac.authorization.k8s.io/aggregate-to-edit: "true"
    rbac.authorization.k8s.io/aggregate-to-view: "true"
# ..
```

这允许已经具有 **view**, **edit**, 或 **admin** 角色的用户与 **ClusterRole** 对象指定的自定义资源交互，而无需额外的角色或集群角色绑定到特定的用户或组。

先决条件

- 在集群中安装了集群扩展。
- 您有一个 API 组和资源名称的列表，如"查找由集群扩展公开的 API 组和资源中所述。

流程

1. 为集群角色创建对象定义，用于指定集群扩展提供的 API 组和资源，并添加聚合标签以扩展一个或多个现有的默认集群角色：

   带有聚合标签的 ClusterRole 对象示例

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: view-custom-resource-aggregated
     labels:
       rbac.authorization.k8s.io/aggregate-to-view: "true"
   rules:
     - apiGroups:
         - <cluster_extension_api_group>
       resources:
         - <cluster_extension_custom_resource>
       verbs:
         - get
         - list
         - watch
   ```

   您可以为 **edit** 和 **admin** 创建类似的 **ClusterRole** 对象，使其有适当的动词操作，如 **create**, **update**, 和 **delete**。通过使用聚合标签，自定义资源的权限将添加到默认角色中。

2. 将对象定义保存到 YAML 文件中。
3. 运行以下命令来创建对象：

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

其他资源

- 聚合的 ClusterRole (Kubernetes 文档)

5.3. 更新路径

在决定*更新路径*（也称为升级路径或升级限制）时，对于已安装的集群扩展，Operator Lifecycle Manager (OLM) v1 支持从 OpenShift Container Platform 4.16 开始的 OLM (Classic) 语义。这个支持遵循 OLM (Classic) 的行为（包括 **replaces**, **skips**, 和 **skipRange** 指令），但有以下一些区别。

通过支持 OLM (Classic) 语义，OLM v1 会准确反映目录中的更新图表。

与原始 OLM (Classic) 实现的不同

- 如果有多个可能的后续者，OLM v1 行为有所不同：
  - 在 OLM (Classic) 中，选择与频道头最接近的后续者。
  - 在 OLM v1 中，选择具有最高语义版本的后续版本(semver)。
- 考虑以下基于文件的目录 (FBC) 频道条目：

```
# ...
- name: example.v3.0.0
  skips: ["example.v2.0.0"]
- name: example.v2.0.0
  skipRange: >=1.0.0 <2.0.0
```

  如果安装了 **1.0.0**，OLM v1 行为如下：

  - OLM (Classic) 不会检测到 **v2.0.0** 的更新路径，因为 **v2.0.0** 被跳过，而不是在 **replaces** 的链中。
  - OLM v1 将检测更新路径，因为 OLM v1 没有 **replaces** 链的概念。OLM v1 找到所有带有 **replace**、**skip** 或 **skipRange** 值的条目，它们涵盖当前安装的版本。

其他资源

- OLM (Classic) 升级语义

5.3.1. 支持版本范围

在 Operator Lifecycle Manager (OLM) v1 中，您可以使用 Operator 或扩展的自定义资源(CR)中的比较字符串来指定版本范围。如果您在 CR 中指定版本范围，OLM v1 会安装或升级到可以在版本范围内解析的 Operator 的最新版本。

解析的版本工作流

- 解析的版本是满足 Operator 和环境限制的 Operator 的最新版本。
- 如果成功解决，则会自动安装指定范围内的 Operator 更新。
- 当更新在指定的范围之外，或者无法成功解决，则不会安装更新。

5.3.2. 版本比较字符串

您可以通过在 Operator 或扩展的自定义资源(CR)的 **spec.version** 字段中添加比较字符串来定义版本范围。比较字符串是一个空格或以逗号分隔的值列表，以及以双引号括起的一个或多个比较运算符(**"**)。您可以通过包括 **OR**、或双垂直栏(||)来添加另一个比较字符串，比较字符串之间的运算符。

表 5.4. 基本比较

比较运算符	定义
=	等于
!=	不等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于

您可以使用类似以下示例的范围比较在 Operator 或扩展 CR 中指定版本范围：

版本范围比较示例

```
apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    name: <clusterextension_name>
  spec:
    namespace: <installed_namespace>
    serviceAccount:
      name: <service_account_installer_name>
    source:
      sourceType: Catalog
      catalog:
        packageName: <package_name>
        version: ">=1.11, <1.13"
```

您可以在所有类型的比较字符串中使用通配符字符。OLM v1 接受 **x**、**X** 和星号 (*) 作为通配符字符。您可以在等号 (=) 比较运算符中使用通配符，代表在补丁或次版本级别上定义比较。

表 5.5. 比较字符串中的通配符字符示例

通配符比较	匹配字符串
1.11.x	>=1.11.0, <1.12.0
>=1.12.X	>=1.12.0
<=2.x	<3
*	>=0.0.0

您可以使用波形符 (~) 比较运算符进行补丁版本比较。补丁版本比较指定最高为下一个主版本的次版本。

表 5.6. 补丁版本比较示例

补丁版本比较	匹配字符串
~1.11.0	>=1.11.0, <1.12.0
~1	>=1, <2
~1.12	>=1.12, <1.13
~1.12.x	>=1.12.0, <1.13.0
~1.x	>=1, <2

您可以使用 caret (^) 比较运算符来比较主版本。如果您在第一个稳定版本发布前进行主发行版本比较，次版本定义了 API 的稳定性级别。在语义版本 (semver) 规格中，第一个稳定版本被发布为 **1.0.0** 版本。

表 5.7. 主发行版本比较示例

主发行版本比较	匹配字符串
^0	>=0.0.0, <1.0.0
^0.0	>=0.0.0, <0.1.0
^0.0.3	>=0.0.3, <0.0.4
^0.2	>=0.2.0, <0.3.0
^0.2.3	>=0.2.3, <0.3.0
^1.2.x	>= 1.2.0, < 2.0.0
^1.2.3	>= 1.2.3, < 2.0.0
^2.x	>= 2.0.0, < 3
^2.3	>= 2.3, < 3

5.3.3. 指定目标版本的自定义资源(CR)示例

在 Operator Lifecycle Manager (OLM) v1 中，集群管理员可以在自定义资源(CR)中声明性地设置 Operator 或扩展的目标版本。

您可以通过指定以下字段来定义目标版本：

- Channel

- 版本号
- 版本范围

如果您在 CR 中指定频道，OLM v1 会安装可在指定频道中解析的 Operator 或扩展的最新版本。当向指定的频道发布更新时，OLM v1 会自动更新至可以从频道解析的最新发行版本。

带有指定频道的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    name: <clusterextension_name>
  spec:
    namespace: <installed_namespace>
    serviceAccount:
      name: <service_account_installer_name>
    source:
      sourceType: Catalog
      catalog:
        packageName: <package_name>
        channels:
          - latest 1
```

1 可选：安装可从指定频道解析的最新版本。对频道的更新会自动安装。指定 **channels** 参数的值（数组）。

如果在 CR 中指定 Operator 或扩展的目标版本，OLM v1 将安装指定的版本。当在 CR 中指定目标版本时，OLM v1 在向目录发布更新时不会更改目标版本。

如果要更新集群中安装的 Operator 版本，您必须手动编辑 Operator 的 CR。指定 Operator 的目标版本将 Operator 的版本固定到指定的发行版本。

指定了目标版本的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
  kind: ClusterExtension
  metadata:
    name: <clusterextension_name>
  spec:
    namespace: <installed_namespace>
    serviceAccount:
      name: <service_account_installer_name>
    source:
      sourceType: Catalog
      catalog:
        packageName: <package_name>
        version: "1.11.1" 1
```

1 可选：指定目标版本。如果要更新安装的 Operator 或扩展版本，您必须手动将 CR 更新至所需的目标版本。

如果要为 Operator 或扩展定义可接受的版本范围，您可以使用比较字符串指定版本范围。当您指定版本范围时，OLM v1 会安装可由 Operator Controller 解析的 Operator 或扩展的最新版本。

指定了版本范围的 CR 示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: <clusterextension_name>
spec:
  namespace: <installed_namespace>
  serviceAccount:
    name: <service_account_installer_name>
  source:
    sourceType: Catalog
    catalog:
      packageName: <package_name>
      version: ">1.11.1" (1)
```

1 可选：指定所需版本范围是否大于 **1.11.1**。如需更多信息，请参阅"支持版本范围"。

创建或更新 CR 后，运行以下命令来应用配置文件：

命令语法

```
$ oc apply -f <extension_name>.yaml
```

5.3.4. 强制更新或回滚

OLM v1 不支持对下一个主版本的自动更新，或回滚到较早的版本。如果要执行主版本更新或回滚，您必须验证并强制更新。

警告

您必须验证强制进行手动更新或回滚的结果。验证强制更新或回滚失败可能会产生灾难性后果，如数据丢失。

先决条件

- 已安装目录。
- 已安装 Operator 或扩展。
- 您已创建了服务帐户，并分配了足够的基于角色的访问控制 (RBAC) 来安装、更新和管理您要安装的扩展。如需更多信息，请参阅 *创建服务帐户*。

流程

1. 编辑 Operator 或扩展的自定义资源 (CR)，如下例所示：

CR 示例

```
apiVersion: olm.operatorframework.io/v1
 kind: ClusterExtension
 metadata:
  name: <clusterextension_name>
 spec:
  namespace: <installed_namespace> 1
  serviceAccount:
   name: <service_account_installer_name> 2
  source:
   sourceType: Catalog
   catalog:
    packageName: <package_name>
    channels:
     - <channel_name> 3
    version: <version_or_version_range> 4
    upgradeConstraintPolicy: SelfCertified 5
```

1. 指定您要安装捆绑包的命名空间，如 **pipelines** 或 **my-extension**。扩展仍然是集群范围的，可能包含在不同命名空间中安装的资源。
2. 指定您为安装、更新和管理扩展创建的服务帐户的名称。
3. 可选：将频道名称指定为数组，如 **pipelines-1.14** 或 **latest**。
4. 可选：指定您要安装的软件包的版本或版本范围，如 **1.14.0**、**1.14.x** 或 **>=1.16**。如需更多信息，请参阅"示例自定义资源(CR)指定目标版本"和"支持版本范围"。
5. 可选：指定升级约束策略。要强制更新或回滚，请将字段设置为 **SelfCertified**。如果未指定，则默认设置为 **CatalogProvided**。只有在新版本满足软件包作者设置的升级限制时，**CatalogProvided** 设置才会更新。

2. 运行以下命令，对 Operator 或 extensions CR 应用更改：

```
$ oc apply -f <extension_name>.yaml
```

其他资源

- 支持版本范围

5.3.5. 与 OpenShift Container Platform 版本的兼容性

在集群管理员将其 OpenShift Container Platform 集群更新至其下一个次版本前，它们必须确保所有安装的 Operator 更新至与集群下一个次版本(4.y+1)兼容的捆绑包版本。

例如，Kubernetes 定期弃用后续版本中删除的某些 API。如果使用已弃用的 API，则在 OpenShift Container Platform 集群更新至删除 API 的 Kubernetes 版本后，它可能无法正常工作。

如果 Operator 作者知道不支持特定的捆绑包版本，且出于某种原因，在 OpenShift Container Platform 上比特定集群次版本稍后无法正常工作，他们可以配置其 Operator 兼容的最大 OpenShift Container Platform 版本。

在 Operator 项目的集群服务版本(CSV)中，作者可以设置 **olm.maxOpenShiftVersion** 注解，以防止管理员在将已安装的 Operator 更新至兼容版本前更新集群。

带有 olm.maxOpenShiftVersion 注解的 CSV 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    "olm.properties": '[{"type": "olm.maxOpenShiftVersion", "value": "<cluster_version>"}]' 1
```

1 指定 Operator 兼容的最新 OpenShift Container Platform (4.y)次要版本。例如，在集群中安装此捆绑包时，将 **value** 设为 **4.19** 可防止集群升级到 4.19 之后的次版本。

如果省略 **olm.maxOpenShiftVersion** 字段，则此 Operator 不会阻断集群更新。

注意

在决定集群的下一个次版本(4.y+1)时，OLM v1 只考虑主版本和次版本(x 和 y)进行比较。它会忽略任何 *z-stream* 版本(4.y.z)，也称为补丁版本或预发布版本。

例如，如果集群的当前版本是 **4.19.0**，则下一个次版本为 **4.20**。如果当前版本为 **4.19.0-rc1**，则下一个次版本仍为 **4.20**。

其他资源

- 弃用的 API 迁移指南 (Kubernetes 文档)

5.3.5.1. olm cluster Operator 阻止的集群更新

如果设置了已安装的 Operator **olm.maxOpenShiftVersion** 字段，并且集群管理员尝试将其集群更新至 Operator 没有提供有效更新路径的版本，则集群更新会失败，并且 **olm** cluster Operator 的 **Upgradeable** 设置为 **False**。

要解决这个问题，集群管理员必须将已安装的 Operator 更新至具有有效更新路径的版本（如果可用），或者必须卸载 Operator。然后，他们可以尝试集群更新。

其他资源

- 了解集群 Operator 条件类型
- 升级安装的 Operator
- 从集群中删除 Operator
- Cluster Operators 参考 → Operator Lifecycle Manager (OLM) v1 Operator

5.4. 自定义资源定义 (CRD) 升级安全

当您更新由集群扩展提供的自定义资源定义(CRD)时，Operator Lifecycle Manager (OLM) v1 运行 CRD 升级安全 preflight 检查，以确保与 CRD 的早期版本向后兼容。在允许更改在集群中进行前，CRD 更新必须通过验证检查。

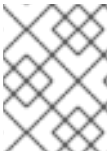
其他资源

- 更新集群扩展

5.4.1. 禁止 CRD 升级更改

CRD 升级安全 preflight 检查会发现对现有自定义资源定义 (CRD) 的以下更改，并阻止升级：

- 将新的必填字段添加到 CRD 的现有版本中
- 现有字段已从现有 CRD 版本中删除
- 在 CRD 的现有版本中更改现有字段类型
- 在之前没有默认值的字段中添加一个新的默认值
- 字段的默认值被改变
- 字段的现有默认值被删除
- 在之前没有 enum 限制的现有字段中添加了新的 enum 限制
- 现有字段中的现有 enum 值会被删除
- 现有字段的最小值会在现有版本中增加
- 在现有版本中现有字段的最大值会减少
- 在之前没有限制的字段中添加最小或最大字段限制



注意

对最小和最大值的更改规则适用于 **minimum**, **minLength**, **minProperties**, **minItems**, **maximum**, **maxLength**, **maxProperties**, 和 **maxItems** 约束。

CRD 升级安全 preflight 检查会报告对现有 CRD 的以下更改，并防止升级，尽管 Kubernetes API 服务器在技术上处理操作：

- 范围从 **Cluster** 改为 **Namespace** 或从 **Namespace** 改为 **Cluster**
- 删除现存已存储的 CRD 的现有版本

如果 CRD 升级安全 preflight 检查遇到禁止的升级更改之一，它会记录 CRD 升级过程中检测到的每个禁止更改的错误。

提示

如果对 CRD 的更改没有属于被禁止更改类别之一，但也无法正确地探测到允许的 CRD 升级安全 preflight 检查，则 CRD 升级安全 preflight 检查将阻止升级并记录"未知更改"的错误。

5.4.2. 允许 CRD 升级更改

对现有自定义资源定义 (CRD) 的以下更改可以安全地向后兼容，且不会导致 CRD 升级安全 preflight 检查停止升级：

- 在字段中允许枚举值列表中添加新的 enum 值
- 在现有版本中将现有必填字段更改为 optional
- 在现有版本中现有字段的最小值会减少
- 在现有版本中增加现有字段的最大值
- 在不对现有版本进行任何修改的情况下，会添加 CRD 的新版本

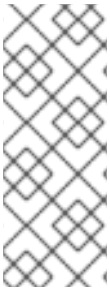
5.4.3. 禁用 CRD 升级安全 preflight 检查

您可以禁用自定义资源定义 (CRD) 升级安全 preflight 检查。在提供 CRD 的 **ClusterExtension** 对象中，将 **install.preflight.crdUpgradeSafety.enforcement** 字段设置为 **None**。

警告

禁用 CRD 升级安全 preflight 检查可能会破坏与 CRD 存储版本的向后兼容性，并在集群中造成其他意外后果。

您无法禁用单个字段验证器。如果您禁用 CRD 升级安全 preflight 检查，则会禁用所有字段验证器。



注意

如果您在 Operator Lifecycle Manager (OLM) v1 中禁用 CRD 升级安全 preflight 检查，Kubernetes API 服务器仍然会阻止以下操作：

- 将范围从 **Cluster** 改为 **Namespace** 或从 **Namespace** 改为 **Cluster**
- 删除 CRD 的现有存储版本

先决条件

- 已安装集群扩展。

流程

1. 编辑 CRD 的 **ClusterExtension** 对象：

```
$ oc edit clusterextension <clusterextension_name>
```

2. 将 **install.preflight.crdUpgradeSafety.enforcement** 字段设置为 **None** ：

ClusterExtension 对象示例

```
apiVersion: olm.operatorframework.io/v1
kind: ClusterExtension
metadata:
  name: clusterextension-sample
```

```
  spec:
    namespace: default
    serviceAccount:
      name: sa-example
    source:
      sourceType: "Catalog"
      catalog:
        packageName: argocd-operator
        version: 0.6.0
    install:
      preflight:
        crdUpgradeSafety:
          enforcement: None
```

5.4.4. 不安全的 CRD 更改示例

以下示例演示了 CRD 升级安全 preflight 检查示例自定义资源定义 (CRD) 中的具体更改。

对于以下示例，请考虑以下开始状态的 CRD 对象：

例 5.19. CRD 对象示例

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    controller-gen.kubebuilder.io/version: v0.13.0
  name: example.test.example.com
spec:
  group: test.example.com
  names:
    kind: Sample
    listKind: SampleList
    plural: samples
    singular: sample
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            type: string
          kind:
            type: string
          metadata:
            type: object
          spec:
            type: object
          status:
            type: object
          pollInterval:
            type: string
        type: object
    served: true
```

```
    storage: true
    subresources:
      status: {}
```

5.4.4.1. 范围更改

在以下自定义资源定义 (CRD) 示例中，**scope** 字段从 **Namespaced** 改为 **Cluster** ：

例 5.20. CRD 中的范围更改示例

```
spec:
  group: test.example.com
  names:
    kind: Sample
    listKind: SampleList
    plural: samples
    singular: sample
  scope: Cluster
  versions:
  - name: v1alpha1
```

例 5.21. 错误输出示例

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoScopeChange" validation failed: scope
changed from "Namespaced" to "Cluster"
```

5.4.4.2. 删除存储的版本

在以下自定义资源定义 (CRD) 示例中，现有存储版本 **v1alpha1** 已被删除：

例 5.22. 在 CRD 中删除存储版本示例

```
  versions:
  - name: v1alpha2
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            type: string
          kind:
            type: string
          metadata:
            type: object
          spec:
            type: object
          status:
            type: object
```

```
      pollInterval:
        type: string
    type: object
```

例 5.23. 错误输出示例

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoStoredVersionRemoved" validation failed:
stored version "v1alpha1" removed
```

5.4.4.3. 删除现有字段

在以下自定义资源定义 (CRD) 示例中，**pollInterval** 属性字段已从 **v1alpha1** 模式中删除：

例 5.24. 在 CRD 中删除现有字段示例

```
versions:
- name: v1alpha1
  schema:
    openAPIV3Schema:
      properties:
        apiVersion:
          type: string
        kind:
          type: string
        metadata:
          type: object
        spec:
          type: object
        status:
          type: object
      type: object
```

例 5.25. 错误输出示例

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoExistingFieldRemoved" validation failed:
crd/test.example.com version/v1alpha1 field/^.spec.pollInterval may not be removed
```

5.4.4.4. 添加必填字段

在以下自定义资源定义 (CRD) 示例中，**pollInterval** 属性已改为必填字段：

例 5.26. 在 CRD 中添加必填字段的示例

```
versions:
- name: v1alpha2
```

```
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            type: string
          kind:
            type: string
          metadata:
            type: object
          spec:
            type: object
          status:
            type: object
          pollInterval:
            type: string
        type: object
        required:
        - pollInterval
```

例 5.27. 错误输出示例

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "ChangeValidator" validation failed: version
"v1alpha1", field "^": new required fields added: [pollInterval]
```