OpenShift Container Platform 4.19

Operator

在 OpenShift Container Platform 中使用 Operator

Last Updated: 2025-12-10

OpenShift Container Platform 4.19 Operator

在 OpenShift Container Platform 中使用 Operator

Legal Notice

Abstract

本文档提供有关在 OpenShift Container Platform 中使用 Operator 的信息。文中为集群管理员提供了 Operator 的安装和管理说明，为开发人员提供了如何通过所安装的 Operator 创建应用程序的信息。另外还提供了一些使用 Operator SDK 构建自用 Operator 的指南。

Table of Contents

第 1 章 OPERATOR 概述

Operator 是 OpenShift Container Platform 中最重要的组件。Operator 是 control plane 上打包、部署和管理服务的首选方法。它们还可以为用户运行的应用程序提供优势。

Operator 与 Kubernetes API 和 CLI 工具（如 **kubectl** 和 OpenShift CLI **oc** 命令）集成。它们提供了监控应用程序、执行健康检查、管理无线(OTA)更新的方法，并确保应用程序保持在指定的状态。

Operator 为 Kubernetes 原生应用程序专门设计，以实施并自动化常见的第 1 天操作，如安装和配置。Operator 也可以自动执行第 2 天操作，如自动缩放，创建备份。所有这些活动都由集群中运行的一个软件进行控制。

虽然这两个操作都遵循类似的 Operator 概念和目标，但 OpenShift Container Platform 中的 Operator 由两个不同的系统管理，具体取决于其用途：

Cluster Operators

由 Cluster Version Operator (CVO) 管理并默认安装来执行集群功能。

可选的附加组件 Operator

由 Operator Lifecycle Manager (OLM) 管理，并可供用户在其应用程序中运行。也称为 *基于 OLM 的 Operator*。

1.1. 对于开发人员

作为 Operator 作者，您可以为基于 OLM 的 Operator 执行以下开发任务：

- 安装 Operator 并订阅命名空间。
- 通过 Web 控制台 从已安装的 Operator 创建应用程序。

其他资源

- Operator 开发人员的机器删除生命周期 hook 示例

1.2. 对于管理员

作为集群管理员，您可以为基于 OLM 的 Operator 执行以下管理任务：

- 管理自定义目录。
- 允许非集群管理员安装 Operator。
- 从 OperatorHub 安装 Operator。
- 查看 Operator 状态。
- 管理 Operator 条件。
- 升级已安装的 Operator。
- 删除已安装的 Operator。
- 配置代理支持。
- 在断开连接的环境中使用 Operator Lifecycle Manager。

有关红帽提供的集群 Operator 的详情，请参考 Cluster Operator 参考。

1.3. 后续步骤

- 什么是 Operator?

第 2 章 了解 OPERATOR

2.1. 什么是 OPERATOR?

从概念上讲，*Operator* 会收集人类操作知识，并将其编码成更容易分享给消费者的软件。

Operator 是一组软件，可用于降低运行其他软件的操作复杂程度。它可以被看作是软件厂商的工程团队的扩展，可以在 Kubernetes 环境中（如 OpenShift Container Platform）监控软件的运行情况，并根据软件的当前状态实时做出决策。Advanced Operator 被设计为用来无缝地处理升级过程，并对出现的错误自动进行响应，而且不会采取"捷径"（如跳过软件备份过程来节省时间）。

从技术上讲，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。要想充分利用 Kubernetes，您需要一组统一的 API 进行扩展，以便服务和管理 Kubernetes 上运行的应用程序。可将 Operator 看成管理 Kubernetes 中这类应用程序的运行时。

2.1.1. 为什么要使用 Operator？

Operator 可以：

- 重复安装和升级。
- 持续对每个系统组件执行运行状况检查。
- 无线 (OTA) 更新 OpenShift 组件和 ISV 内容。
- 汇总现场工程师了解的情况并将其传输给所有用户，而非一两个用户。

为什么在 Kubernetes 上部署？

Kubernetes（扩展至 OpenShift Container Platform）包含构建复杂分布式系统（可在本地和云提供商之间工作）需要的所有原语，包括 secret 处理、负载均衡、服务发现、自动扩展。

为什么使用 Kubernetes API 和 kubectl 工具来管理您的应用程序？

这些 API 功能丰富，所有平台均有对应的客户端，并可插入到集群的访问控制/审核中。Operator 会使用 Kubernetes 的扩展机制"自定义资源定义 (CRD)"支持您的自定义对象，如 **MongoDB**，它类似于内置的原生 Kubernetes 对象。

Operator 与 Service Broker 的比较？

服务代理（service broker）是实现应用程序的编程发现和部署的一个步骤。但它并非一个长时间运行的进程，所以无法执行第 2 天操作，如升级、故障转移或扩展。它在安装时提供对可调参数的自定义和参数化，而 Operator 则可持续监控集群的当前状态。非集群服务仍非常适合于 Service Broker，但也存在合适于这些服务的 Operator。

2.1.2. Operator Framework

Operator Framework 是基于上述客户体验提供的一系列工具和功能。不仅仅是编写代码；测试、交付和更新 Operator 也同样重要。Operator Framework 组件包含用于解决这些问题的开源工具：

Operator Lifecycle Manager

Operator Lifecycle Manager (OLM) 能够控制集群中 Operator 的安装、升级和基于角色的访问控制 (RBAC)。它默认部署在 OpenShift Container Platform 4.19 中。

Operator Registry

Operator Registry 存储 ClusterServiceVersions (CSV) 和自定义资源定义 (CRD) 以便在集群中创建，并存储有关软件包和频道的 Operator 元数据。它运行在 Kubernetes 或 OpenShift 集群中，向 OLM 提供这些 Operator 目录数据。

OperatorHub

OperatorHub 是一个 web 控制台，供集群管理员用来发现并选择要在其集群上安装的 Operator。它在 OpenShift Container Platform 中默认部署。

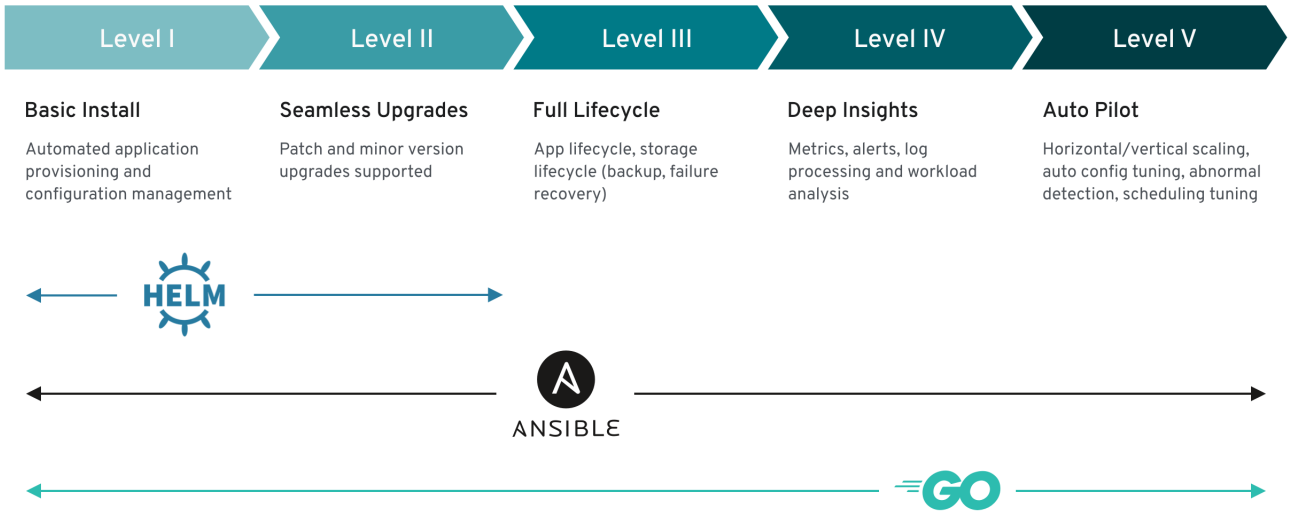
这些工具可组合使用，因此您可自由选择对您有用的工具。

2.1.3. Operator 成熟度模型

Operator 内部封装的管理逻辑的复杂程度各有不同。该逻辑通常还高度依赖于 Operator 所代表的服务类型。

对于大部分 Operator 可能包含的特定功能集来说，可以大致推断出 Operator 封装操作的成熟度等级。就此而言，以下 Operator 成熟度模型针对 Operator 的第二天通用操作定义了五个成熟度阶段：

图 2.1. Operator 成熟度模型



2.2. OPERATOR FRAMEWORK 打包格式

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager (OLM) 所支持的 Operator 打包格式。

2.2.1. 捆绑包格式

Operator 的 *Bundle Format* 是 Operator Framework 引入的新打包格式。为提高可伸缩性并为自行托管目录的上游用户提供更好地支持，Bundle Format 规格简化了 Operator 元数据的发布。

Operator 捆绑包代表 Operator 的单一版本。磁盘上的*捆绑包清单*是容器化的，并作为*捆绑包镜像*提供，该镜像是一个不可运行的容器镜像，其中存储了 Kubernetes 清单和 Operator 元数据。然后，使用现有容器工具（如 **podman** 和 **docker**）和容器 registry（如 Quay）来管理捆绑包镜像的存储和发布。

Operator 元数据可以包括：

- 标识 Operator 的信息，如名称和版本。

- 驱动 UI 的额外信息，例如其图标和一些示例自定义资源 (CR)。
- 所需的和所提供的 API。
- 相关镜像。

将清单加载到 Operator Registry 数据库中时，会验证以下要求：

- 该捆绑包必须在注解中至少定义一个频道。
- 每个捆绑包都只有一个集群服务版本（CSV）。
- 如果 CSV 拥有自定义资源定义（CRD），则该 CRD 必须存在于捆绑包中。

2.2.1.1. 清单

捆绑包清单指的是一组 Kubernetes 清单，用于定义 Operator 的部署和 RBAC 模型。

捆绑包包括每个目录的一个 CSV，一般情况下，用来定义 CRD 所拥有的 API 的 CRD 位于 **/manifest** 目录中。

捆绑包格式布局示例

```
etcd
 ├──── manifests
 │     ├──── etcdcluster.crd.yaml
 │     └──── etcdoperator.clusterserviceversion.yaml
 │     └──── secret.yaml
 │     └──── configmap.yaml
 └──── metadata
       └──── annotations.yaml
       └──── dependencies.yaml
```

2.2.1.1.1. 额外支持的对象

以下对象类型也可以包括在捆绑包的 **/manifests** 目录中：

支持的可选对象类型

- **ClusterRole**
- **ClusterRoleBinding**
- **ConfigMap**
- **ConsoleCLIDownload**
- **ConsoleLink**
- **ConsoleQuickStart**
- **ConsoleYamlSample**
- **PodDisruptionBudget**

- **PriorityClass**
- **PrometheusRule**
- **角色**
- **RoleBinding**
- **Secret**
- **服务**
- **ServiceAccount**
- **ServiceMonitor**
- **VerticalPodAutoscaler**

当捆绑包中包含这些可选对象时，Operator Lifecycle Manager（OLM）可以从捆绑包创建对象，并随 CSV 一起管理其生命周期：

可选对象的生命周期

- 删除 CSV 后，OLM 会删除可选对象。
- 当 CSV 被升级时：
  - 如果可选对象的名称相同，OLM 会更新它。
  - 如果可选对象的名称在版本间有所变化，OLM 会删除并重新创建它。

2.2.1.2. 注解

捆绑包还在其 **/metadata** 文件夹中包含 **annotations.yaml** 文件。此文件定义了更高级别的聚合数据，以帮助描述有关如何将捆绑包添加到捆绑包索引中的格式和软件包信息：

annotations.yaml 示例

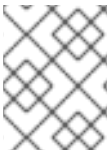
```
annotations:
  operators.operatorframework.io.bundle.mediatype.v1: "registry+v1" 1
  operators.operatorframework.io.bundle.manifests.v1: "manifests/" 2
  operators.operatorframework.io.bundle.metadata.v1: "metadata/" 3
  operators.operatorframework.io.bundle.package.v1: "test-operator" 4
  operators.operatorframework.io.bundle.channels.v1: "beta,stable" 5
  operators.operatorframework.io.bundle.channel.default.v1: "stable" 6
```

1. Operator 捆绑包的介质类型或格式。**registry+v1** 格式表示它包含 CSV 及其关联的 Kubernetes 对象。
2. 镜像中的该路径指向含有 Operator 清单的目录。该标签保留给以后使用，当前默认为 **manifests/**。**manifests.v1** 值表示捆绑包包含 Operator 清单。
3. 镜像中的该路径指向包含捆绑包元数据文件的目录。该标签保留给以后使用，当前默认为 **metadata/**。**metadata.v1** 值表示这个捆绑包包含 Operator 元数据。

4 捆绑包的软件包名称。

5 捆绑包添加到 Operator Registry 时订阅的频道列表。

6 从 registry 安装时，Operator 应该订阅到的默认频道。



注意

如果出现不匹配的情况，则以 **annotations.yaml** 文件为准，因为依赖这些注解的集群 Operator Registry 只能访问此文件。

2.2.1.3. 依赖项

Operator 的依赖项列在捆绑包的 **metadata/** 目录中的 **dependencies.yaml** 文件中。此文件是可选的，目前仅用于指明 Operator-version 依赖项。

依赖项列表中，每个项目包含一个 **type** 字段，用于指定这一依赖项的类型。支持以下 Operator 依赖项：

olm.package

这个类型表示特定 Operator 版本的依赖项。依赖项信息必须包含软件包名称以及软件包的版本，格式为 semver。例如，您可以指定具体版本，如 **0.5.2**，也可指定一系列版本，如 **>0.5.1**。

olm.gvk

使用这个类型，作者可以使用 group/version/kind(GVK)信息指定依赖项，类似于 CSV 中现有 CRD 和基于 API 的使用量。该路径使 Operator 作者可以合并所有依赖项、API 或显式版本，使它们处于同一位置。

olm.constraint

这个类型在任意 Operator 属性上声明通用限制。

在以下示例中，为 Prometheus Operator 和 etcd CRD 指定依赖项：

dependencies.yaml 文件示例

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
      group: etcd.database.coreos.com
      kind: EtcdCluster
      version: v1beta2
```

其他资源

- Operator Lifecycle Manager 依赖项解析

2.2.1.4. 关于 opm CLI

opm CLI 工具由 Operator Framework 提供,用于 Operator 捆绑格式。您可以通过此工具从与软件存储库类似的 Operator 捆绑包列表中创建和维护 Operator 目录。其结果是一个容器镜像，它可以存储在容器的 registry 中，然后安装到集群中。

目录包含一个指向 Operator 清单内容的指针数据库，可通过在运行容器镜像时提供的已包含 API 进行查询。在 OpenShift Container Platform 中，Operator Lifecycle Manager (OLM) 可以引用由 **CatalogSource** 对象定义的目录源中的镜像，它会定期轮询镜像，以对集群上安装的 Operator 进行更新。

- 有关安装 **opm** CLI 的步骤，请参阅 CLI 工具。

2.2.2. 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。

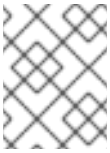
此可编辑功能启用以下功能和用户定义的扩展：

- 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。

重要

从 OpenShift Container Platform 4.11 开始，默认的红帽提供的 Operator 目录以基于文件的目录格式发布。通过以过时的 SQLite 数据库格式发布的 4.10，用于 OpenShift Container Platform 4.6 的默认红帽提供的 Operator 目录。

与 SQLite 数据库格式相关的 **opm** 子命令、标志和功能已被弃用，并将在以后的版本中删除。功能仍被支持，且必须用于使用已弃用的 SQLite 数据库格式的目录。

许多 **opm** 子命令和标志都用于 SQLite 数据库格式，如 **opm index prune**，它们无法使用基于文件的目录格式。有关使用基于文件的目录的更多信息，请参阅管理自定义目录以及使用 oc-mirror 插件为断开连接的安装 mirror 镜像 。

2.2.2.1. 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。

2.2.2.2. 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

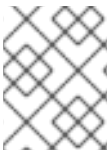
```
_Meta: {
  // schema is required and must be a non-empty string
  schema: string & !=""

  // 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。

2.2.2.2.1. 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 2.1. 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
  }
}
```

2.2.2.2.2. 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 2.2. 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。

2.2.2.2.3. olm.bundle schema

例 2.3. 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 & !=""
}
```

2.2.2.2.4. 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 2.1. 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 2.4. 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

2.2.2.3. 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。

2.2.2.3.1. 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 2.5. olm.package 属性

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

2.2.2.3.2. 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 2.6. olm.gvk 属性

```
#PropertyGVK: {
  type: "olm.gvk"
```

```
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

2.2.2.3.3. 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 2.7. olm.package.required 属性

```
#PropertyPackageRequired: {
  type: "olm.package.required"
  value: {
    packageName: string & !=""
    versionRange: string & !=""
  }
}
```

2.2.2.3.4. 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 2.8. olm.gvk.required 属性

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

2.2.2.4. 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@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f
6be03
- name: prometheus-operator
  image: quay.io/prometheus-
operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d84101
eb317
```

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"
```

2.2.2.5. 指南

在维护基于文件的目录时，请考虑以下准则。

2.2.2.5.1. 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**。

2.2.2.5.2. 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。

2.2.2.6. 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。

2.2.2.7. 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。

2.3. OPERATOR FRAMEWORK 常用术语表

本主题提供了与 Operator Framework 相关的常用术语表，包括 Operator Lifecycle Manager (OLM)。

2.3.1. 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。

2.3.2. 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。

2.3.3. 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。

2.3.4. 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。

2.3.5. 频道头

*频道头*是指特定频道中最新已知的更新。

2.3.6. 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)。

2.3.7. 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。

2.3.8. 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。

2.3.9. 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。

2.3.10. 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。

2.3.11. 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。

2.3.12. 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。

2.3.13. 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。

2.3.14. 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。

2.3.15. 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。

2.3.16. Subscription

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

2.3.17. 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）*。

2.4. OPERATOR LIFECYCLE MANAGER (OLM)

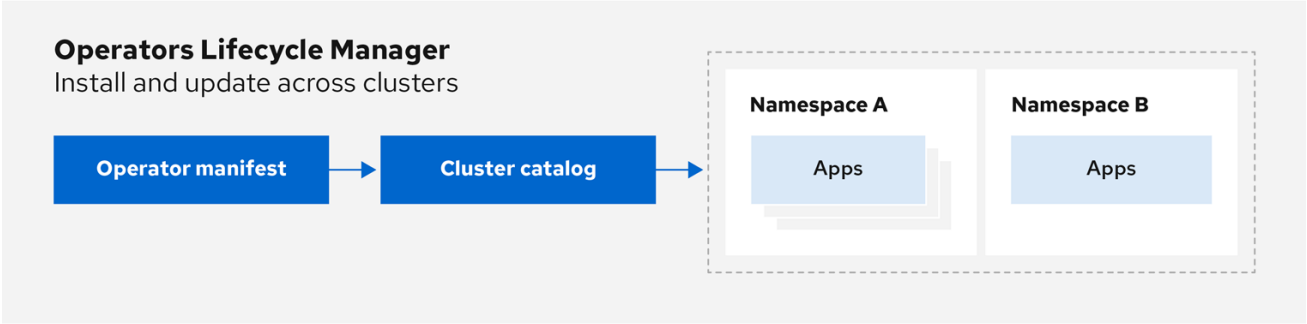
2.4.1. Operator Lifecycle Manager 概念和资源

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager (OLM) 背后的概念。

2.4.1.1. 什么是 Operator Lifecycle Manager (OLM) Classic？

Operator Lifecycle Manager (OLM) Classic 可帮助用户安装、更新和管理 Kubernetes 原生应用程序 (Operator)以及在 OpenShift Container Platform 集群中运行的关联服务的生命周期。它是 Operator Framework 的一部分，后者是一个开源工具包，用于以有效、自动化且可扩展的方式管理 Operator。

图 2.2. OLM (Classic) 工作流



OLM 默认在 OpenShift Container Platform 4.19 中运行，辅助集群管理员对集群上运行的 Operator 进行安装、升级和授予访问权。OpenShift Container Platform Web 控制台提供一些管理界面，供集群管理员安装 Operator，以及为特定项目授权以便使用集群上的可用 Operator 目录。

开发人员通过自助服务体验，无需成为相关问题的专家也可自由置备和配置数据库、监控和大数据服务的实例，因为 Operator 已将相关知识融入其中。

2.4.1.2. OLM 资源

以下自定义资源定义 (CRD) 由 Operator Lifecycle Manager (OLM) 定义和管理：

表 2.2. 由 OLM 和 Catalog Operator 管理的 CRD

资源	短名称	描述
ClusterServiceVersion (CSV)	**csv**	应用程序元数据。例如：名称、版本、图标、所需资源。
CatalogSource	**catsrc**	定义应用程序的 CSV、CRD 和软件包存储库。
Subscription	**sub**	通过跟踪软件包中的频道来保持 CSV 最新。
InstallPlan	**ip**	为自动安装或升级 CSV 而需创建的资源的计算列表。
OperatorGroup	**og**	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自定义资源 (CR)。
OperatorConditions	-	在 OLM 和它管理的 Operator 之间创建通信频道。操作员可以写入 **Status.Conditions** 数组，以向 OLM 通报复杂状态。

2.4.1.2.1. 集群服务版本

集群服务版本 (CSV) 代表 OpenShift Container Platform 集群中运行的 Operator 的特定版本。它是一个利用 Operator 元数据创建的 YAML 清单，可辅助 Operator Lifecycle Manager (OLM) 在集群中运行 Operator。

OLM 需要与 Operator 相关的元数据，以确保它可以在集群中安全运行，并在发布新版 Operator 时提供有关如何应用更新的信息。这和传统的操作系统的打包软件相似；可将 OLM 的打包步骤认为是制作 **rpm**、**deb** 或 **apk** 捆绑包的阶段。

CSV 中包含 Operator 容器镜像附带的元数据，用于在用户界面填充名称、版本、描述、标签、存储库链接和徽标等信息。

此外，CSV 还是运行 Operator 所需的技术信息来源，例如其管理或依赖的自定义资源 (CR)、RBAC 规则、集群要求和安装策略。此信息告诉 OLM 如何创建所需资源并将 Operator 设置为部署。

2.4.1.2.2. 目录源

catalog source 代表元数据存储，通常通过引用存储在容器 registry 中的 *index image*。Operator Lifecycle Manager (OLM) 查询目录源来发现和安装 Operator 及其依赖项。OpenShift Container Platform Web 控制台中的 OperatorHub 也会显示由目录源提供的 Operator。

提示

集群管理员可以使用 web 控制台中的 **Administration → Cluster Settings → Configuration → OperatorHub** 页面查看集群中已启用的目录源提供的 Operator 的完整列表。

CatalogSource 的 **spec** 指明了如何构造 pod，以及如何与服务于 Operator Registry gRPC API 的服务进行通信。

例 2.9. CatalogSource 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog 1
  namespace: openshift-marketplace 2
  annotations:
    olm.catalogImageTemplate: 3
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}.
{kube_patch_version}"
spec:
  displayName: Example Catalog 4
  image: quay.io/example-org/example-catalog:v1 5
  priority: -400 6
  publisher: Example Org
  sourceType: grpc 7
  grpcPodConfig:
    securityContextConfig: <security_mode> 8
    nodeSelector: 9
      custom_label: <label>
    priorityClassName: system-cluster-critical 10
    tolerations: 11
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
  updateStrategy:
    registryPoll: 12
      interval: 30m0s
status:
  connectionState:
    address: example-catalog.openshift-marketplace.svc:50051
    lastConnect: 2021-08-26T18:14:31Z
    lastObservedState: READY 13
  latestImageRegistryPoll: 2021-08-26T18:46:25Z 14
  registryService: 15
    createdAt: 2021-08-26T16:16:37Z
    port: 50051
```

```
protocol: grpc
serviceName: example-catalog
serviceNamespace: openshift-marketplace
```

1. **CatalogSource** 对象的名称。此值也用作在请求的命名空间中创建相关 pod 的名称的一部分。

2. 要创建目录的命名空间。要使目录在所有命名空间中都可用，请将此值设置为 **openshift-marketplace**。默认红帽提供的目录源也使用 **openshift-marketplace** 命名空间。否则，将值设置为特定命名空间，使 Operator 仅在该命名空间中可用。

3. 可选：为避免集群升级可能会使 Operator 安装处于不受支持的状态或没有持续更新路径，您可以启用自动更改 Operator 目录的索引镜像版本作为集群升级的一部分。

   将 **olm.catalogImageTemplate** 注解设置为索引镜像名称，并使用一个或多个 Kubernetes 集群版本变量，如为镜像标签构建模板时所示。该注解会在运行时覆盖 **spec.image** 字段。如需了解更多详细信息，请参阅"用于自定义目录源的镜像模板"。

4. 在 Web 控制台和 CLI 中显示目录的名称。

5. 目录的索引镜像。在使用 **olm.catalogImageTemplate** 注解时，也可以省略，该注解会在运行时设置 pull spec。

6. 目录源的权重。OLM 在依赖项解析过程中使用权重进行优先级排序。权重越高，表示目录优先于轻量级目录。

7. 源类型包括以下内容：

   - 带有**镜像**引用的 **grpc**：OLM 拉取镜像并运行 pod，为兼容的 API 服务。
   - 带有**地址**字段的 **grpc**：OLM 会尝试联系给定地址的 gRPC API。在大多数情况下不应该使用这种类型。
   - **configmap**：OLM 解析配置映射数据，并运行一个可以为其提供 gRPC API 的 pod。

8. 指定 **legacy** 或 **restricted** 的值。如果没有设置该字段，则默认值为 **legacy**。在以后的 OpenShift Container Platform 发行版本中，计划默认值为 **restricted**。如果您的目录无法使用 **restricted** 权限运行，建议您手动将此字段设置为 **legacy**。

9. 可选： 对于 **grpc** 类型目录源，请覆盖在 **spec.image** 中提供内容的 pod 的默认节点选择器（如果定义）。

10. 可选： 对于 **grpc** 类型目录源，请覆盖在 **spec.image** 中提供内容的 pod 的默认优先级类名称（如果定义）。Kubernetes 默认提供 **system-cluster-critical** 和 **system-node-critical** 优先级类。将字段设置为空 (**""**) 可为 pod 分配默认优先级。可以手动定义其他优先级类。

11. 可选： 对于 **grpc** 类型目录源，请覆盖 **spec.image** 中提供内容的 pod 的默认容限（如果定义）。

12. 在指定的时间段内自动检查新版本以保持最新。

13. 目录连接的最后观察到状态。例如：

    - **READY** ：成功建立连接。
    - **CONNECTING** ：连接正在尝试建立。

- **TRANSIENT_FAILURE** ：尝试建立连接（如超时）时发生了临时问题。该状态最终将切回到 **CONNECTING**，然后重试。

如需了解更多详细信息，请参阅 gRPC 文档中的连接状态。

14 存储目录镜像的容器注册表最近轮询的时间，以确保镜像为最新版本。

15 目录的 Operator Registry 服务的状态信息。

在订阅中引用 **CatalogSource** 对象的**名称**会指示 OLM 搜索查找请求的 Operator 的位置：

例 2.10. 引用目录源的 Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

其他资源

- 了解 OperatorHub
- 红帽提供的 Operator 目录
- 在集群中添加目录源
- 目录优先级
- 使用 CLI 查看 Operator 目录源状态
- 了解并管理 pod 安全准入
- 目录源 pod 调度

2.4.1.2.2.1. 自定义目录源的镜像模板

与底层集群的 Operator 兼容性可以通过目录源以各种方式表示。其中一种用于红帽默认提供的目录源的方法是识别为特定平台发行版本（如 OpenShift Container Platform 4.19）特别创建的索引镜像的镜像标签。

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 4.18 升级到 4.19 的过程中，**redhat-operators** 目录的 **CatalogSource** 对象中的 **spec.image** 字段被更新：

```
registry.redhat.io/redhat/redhat-operator-index:v4.19
```

改为：

```
registry.redhat.io/redhat/redhat-operator-index:v4.19
```

但是，CVO 不会自动更新自定义目录的镜像标签。为确保用户在集群升级后仍然安装兼容并受支持的 Operator，还应更新自定义目录以引用更新的索引镜像。

从 OpenShift Container Platform 4.9 开始，集群管理员可以在自定义目录的 **CatalogSource** 对象中添加 **olm.catalogImageTemplate** 注解到包含模板的镜像引用。模板中支持使用以下 Kubernetes 版本变量：

- **kube_major_version**
- **kube_minor_version**
- **kube_patch_version**

注意

您必须指定 Kubernetes 集群版本，而不是 OpenShift Container Platform 集群版本，因为后者目前不适用于模板。

如果您已创建并推送了带有指定更新 Kubernetes 版本标签的索引镜像，设置此注解可使自定义目录中的索引镜像版本在集群升级后自动更改。注解值用于设置或更新 **CatalogSource** 对象的 **spec.image** 字段中的镜像引用。这有助于避免集群升级，从而避免在不受支持的状态或没有持续更新路径的情况下安装 Operator。

重要

您必须确保集群可在集群升级时访问带有更新标签的索引镜像（无论存储在哪一 registry 中）。

例 2.11. 带有镜像模板的目录源示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    olm.catalogImageTemplate:
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}"
spec:
  displayName: Example Catalog
  image: quay.io/example-org/example-catalog:v1.32
  priority: -400
  publisher: Example Org
```

注意

如果设置了 **spec.image** 字段和 **olm.catalogImageTemplate** 注解，则 **spec.image** 字段会被注解中的解析值覆盖。如果注解没有解析为可用的 pull spec，目录源会回退到设置的 **spec.image** 值。

如果没有设置 **spec.image** 字段，且注解没有解析为可用的 pull spec，OLM 会停止目录源的协调，并将其设置为人类可读的错误条件。

对于使用 Kubernetes 1.33 的 OpenShift Container Platform 4.19 集群，上例中的 **olm.catalogImageTemplate** 注解会解析为以下镜像引用：

```
quay.io/example-org/example-catalog:v1.32
```

对于将来的 OpenShift Container Platform 版本，您可以为自定义目录创建更新的索引镜像，该镜像以更新的 Kubernetes 版本为目标，供以后的 OpenShift Container Platform 版本使用。升级前设置了 **olm.catalogImageTemplate** 注解，将集群升级到更新的 OpenShift Container Platform 版本也会自动更新目录的索引镜像。

2.4.1.2.2.2. 目录健康要求

集群上的 Operator 目录可从安装解析视角进行交换；**Subscription** 对象可能会引用特定目录，但依赖项会根据集群中的所有目录解决。

例如，如果 Catalog A 不健康，则引用 Catalog A 的订阅可能会解析 Catalog B 中的依赖项，集群管理员可能还没有预期，因为 B 通常具有比 A 更低的目录优先级。

因此，OLM 要求所有具有给定全局命名空间的目录（例如，默认的 **openshift-marketplace** 命名空间或自定义全局命名空间）都健康。当目录不健康时，其共享全局命名空间中的所有 Operator 或更新操作都将因为 **CatalogSourcesUnhealthy** 条件而失败。如果这些操作处于不健康状态，OLM 可能会做出对集群管理员意外的解析和安装决策。

作为集群管理员，如果您观察一个不健康的目录，并希望将目录视为无效并恢复 Operator 安装，请参阅"删除自定义目录"或"Disabling the default OperatorHub 目录源"部分，以了解有关删除不健康目录的信息。

其他资源

- 删除自定义目录
- 禁用默认的 OperatorHub 目录源

2.4.1.2.3. 订阅

订阅（由一个 **Subscription** 对象定义）代表安装 Operator 的意图。它是将 Operator 与目录源关联的自定义资源。

Subscription 描述了要订阅 Operator 软件包的哪个频道，以及是自动还是手动执行更新。如果设置为自动，订阅可确保 Operator Lifecycle Manager（OLM）自动管理并升级 Operator，以确保集群中始终运行最新版本。

Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
```

```
kind: Subscription
metadata:
  name: example-operator
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

此 **Subscription** 对象定义了 Operator 的名称和命名空间，以及从中查找 Operator 数据的目录。频道（如 **alpha**、**beta** 或 **stable**）可帮助确定应从目录源安装哪些 Operator 流。

订阅中的频道名称可能会因 Operator 而异，但应遵守给定 Operator 中的常规约定。例如，频道名称可能会遵循 Operator 提供的应用程序的次发行版本更新流（**1.2**、**1.3**）或发行的频率（**stable**、**fast**）。

除了可从 OpenShift Container Platform Web 控制台轻松查看外，还可以通过检查相关订阅的状态来识别是否有较新版本的 Operator 可用。与 **currentCSV** 字段关联的值是 OLM 已知的最新版本，而 **installedCSV** 是集群中安装的版本。

其他资源

- 多租户和 Operator 共处
- 使用 CLI 查看 Operator 订阅状态

2.4.1.2.4. 安装计划

安装计划（由一个 **InstallPlan** 对象定义） 描述了 Operator Lifecycle Manager (OLM) 为安装或升级到 Operator 的特定版本而创建的一组资源。该版本由集群服务版本 (CSV) 定义。

要安装 Operator、集群管理员或被授予 Operator 安装权限的用户，必须首先创建一个 **Subscription** 对象。订阅代表了从目录源订阅 Operator 可用版本流的意图。然后，订阅会创建一个 **InstallPlan** 对象来方便为 Operator 安装资源。

然后，根据以下批准策略之一批准安装计划：

- 如果订阅的 **spec.installPlanApproval** 字段被设置为 **Automatic**，则会自动批准安装计划。
- 如果订阅的 **spec.installPlanApproval** 字段被设置为 **Manual**，则安装计划必须由集群管理员或具有适当权限的用户手动批准。

批准安装计划后，OLM 会创建指定的资源，并在订阅指定的命名空间中安装 Operator。

例 2.12. InstallPlan 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: InstallPlan
metadata:
  name: install-abcde
  namespace: operators
spec:
  approval: Automatic
  approved: true
  clusterServiceVersionNames:
```

```
    - my-operator.v1.0.1
  generation: 1
status:
  ...
  catalogSources: []
  conditions:
    - lastTransitionTime: '2021-01-01T20:17:27Z'
      lastUpdateTime: '2021-01-01T20:17:27Z'
      status: 'True'
      type: Installed
  phase: Complete
  plan:
    - resolving: my-operator.v1.0.1
      resource:
        group: operators.coreos.com
        kind: ClusterServiceVersion
        manifest: >-
        ...
        name: my-operator.v1.0.1
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1alpha1
      status: Created
    - resolving: my-operator.v1.0.1
      resource:
        group: apiextensions.k8s.io
        kind: CustomResourceDefinition
        manifest: >-
        ...
        name: webservers.web.servers.org
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1beta1
      status: Created
    - resolving: my-operator.v1.0.1
      resource:
        group: ''
        kind: ServiceAccount
        manifest: >-
        ...
        name: my-operator
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1
      status: Created
    - resolving: my-operator.v1.0.1
      resource:
        group: rbac.authorization.k8s.io
        kind: Role
        manifest: >-
        ...
        name: my-operator.v1.0.1-my-operator-6d7cbc6f57
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1
      status: Created
```

```
  - resolving: my-operator.v1.0.1
    resource:
      group: rbac.authorization.k8s.io
      kind: RoleBinding
      manifest: >-
      ...
      name: my-operator.v1.0.1-my-operator-6d7cbc6f57
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1
    status: Created
    ...
```

其他资源

- 多租户和 Operator 共处
- 允许非集群管理员安装 Operator

2.4.1.2.5. operator 组

由 **OperatorGroup** 资源定义的 *Operator 组*，为 OLM 安装的 Operator 提供多租户配置。Operator 组选择目标命名空间，在其中为其成员 Operator 生成所需的 RBAC 访问权限。

这一组目标命名空间通过存储在 CSV 的 **olm.targetNamespaces** 注解中的以逗号分隔的字符串来提供。该注解应用于成员 Operator 的 CSV 实例，并注入它们的部署中。

其他资源

- operator 组

2.4.1.2.6. Operator 条件

作为管理 Operator 生命周期的角色的一部分，Operator Lifecycle Manager（OLM）从定义 Operator 的 Kubernetes 资源状态中推断 Operator 状态。虽然此方法提供了一定程度的保证来确定 Operator 处于给定状态，但在有些情况下，Operator 可能需要直接向 OLM 提供信息，而这些信息不能被推断出来。这些信息可以被 OLM 使用来更好地管理 Operator 的生命周期。

OLM 提供了一个名为 **OperatorCondition** 的自定义资源定义（CRD），它允许 Operator 与 OLM 相互通信条件信息。当在一个 **OperatorCondition** 资源的 **Spec.Conditions** 数组中存在时，则代表存在一组会影响 OLM 管理 Operator 的支持条件。

注意

默认情况下，**OperatorCondition** 对象中没有 **Spec.Conditions** 数组，直到由用户添加或使用自定义 Operator 逻辑的结果为止。

其他资源

- Operator 条件

2.4.2. Operator Lifecycle Manager 架构

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager (OLM) 的组件架构。

2.4.2.1. 组件职责

Operator Lifecycle Manager (OLM) 由两个 Operator 组成，分别为：OLM Operator 和 Catalog Operator。

OLM 和 Catalog Operator 负责管理作为 OLM 框架基础的自定义资源定义(CRD)：

表 2.3. 由 OLM 和 Catalog Operator 管理的 CRD

资源	短名称	所有者	描述
ClusterServiceVersion (CSV)	**csv**	OLM	应用程序元数据：名称、版本、图标、所需资源、安装等。
InstallPlan	**ip**	Catalog	为自动安装或升级 CSV 而需创建的资源的计算列表。
CatalogSource	**catsrc**	Catalog	定义应用程序的 CSV、CRD 和软件包存储库。
Subscription	**sub**	Catalog	用于通过跟踪软件包中的频道来保持 CSV 最新。
OperatorGroup	**og**	OLM	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自定义资源 (CR)。

每个 Operator 还负责创建以下资源：

表 2.4. 由 OLM 和 Catalog Operator 创建的资源

资源	所有者
部署	OLM
ServiceAccounts	
(Cluster)Roles	
(Cluster)RoleBindings	
CustomResourceDefinitions (CRD)	Catalog
ClusterServiceVersions	

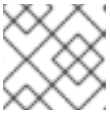
2.4.2.2. OLM Operator

集群中存在 CSV 中指定需要的资源后，OLM Operator 将负责部署由 CSV 资源定义的应用程序。

OLM Operator 不负责创建所需资源；用户可选择使用 CLI 手动创建这些资源，也可选择使用 Catalog Operator 来创建这些资源。这种关注点分离的机制可以使得用户逐渐增加他们选择用于其应用程序的 OLM 框架量。

OLM Operator 使用以下工作流：

1. 观察命名空间中的集群服务版本（CSV），并检查是否满足要求。
2. 如果满足要求，请运行 CSV 的安装策略。



注意

CSV 必须是 Operator 组的活跃成员，才可运行该安装策略。

2.4.2.3. Catalog Operator

Catalog Operator 负责解析和安装集群服务版本（CSV）以及它们指定的所需资源。另外还负责监视频道中的目录源中是否有软件包更新，并将其升级（可选择自动）至最新可用版本。

要跟踪频道中的软件包，您可以创建一个 **Subscription** 对象来配置所需的软件包、频道和 **CatalogSource** 对象，以便拉取更新。在找到更新后，便会代表用户将一个适当的 **InstallPlan** 对象写入命名空间。

Catalog Operator 使用以下工作流：

1. 连接到集群中的每个目录源。
2. 监视是否有用户创建的未解析安装计划，如果有：
   a. 查找与请求名称相匹配的 CSV，并将此 CSC 添加为已解析的资源。
   b. 对于每个受管或所需 CRD，将其添加为已解析的资源。
   c. 对于每个所需 CRD，找到管理相应 CRD 的 CSV。
3. 监视是否有已解析的安装计划并为其创建已发现的所有资源（用户批准或自动）。
4. 观察目录源和订阅并根据它们创建安装计划。

2.4.2.4. Catalog Registry

Catalog Registry 存储 CSV 和 CRD 以便在集群中创建，并存储有关软件包和频道的元数据。

package manifest 是 Catalog Registry 中的一个条目，用于将软件包标识与 CSV 集相关联。在软件包中，频道指向特定 CSV。因为 CSV 明确引用了所替换的 CSV，软件包清单向 Catalog Operator 提供了将 CSV 更新至频道中最新版本所需的信息，逐步安装和替换每个中间版本。

2.4.3. Operator Lifecycle Manager 工作流

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager (OLM) 的工作流。

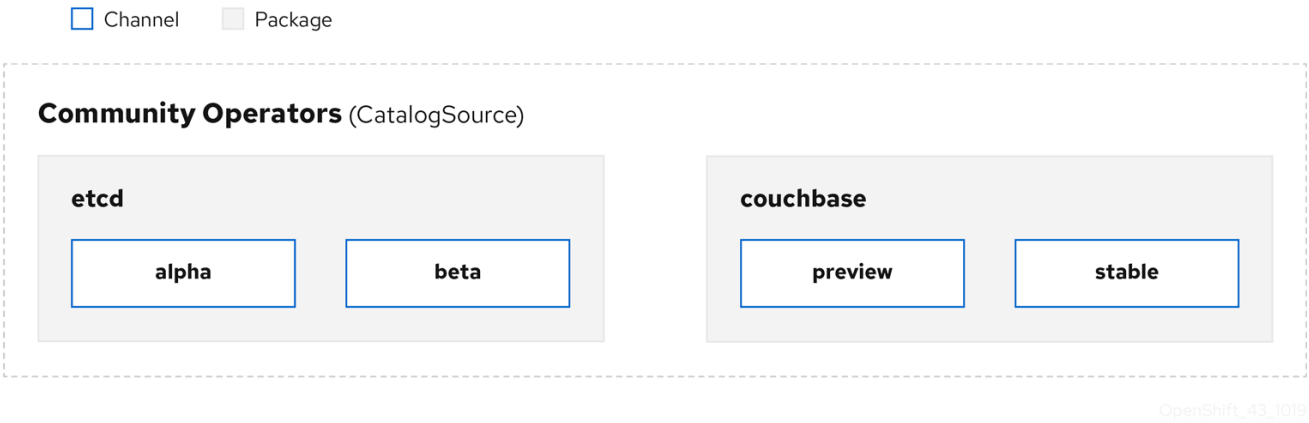
2.4.3.1. OLM 中的 Operator 安装和升级工作流

在 Operator Lifecycle Manager (OLM) 生态系统中，以下资源用于解决 Operator 的安装和升级问题：

- **ClusterServiceVersion** (CSV)
- **CatalogSource**
- **Subscription**

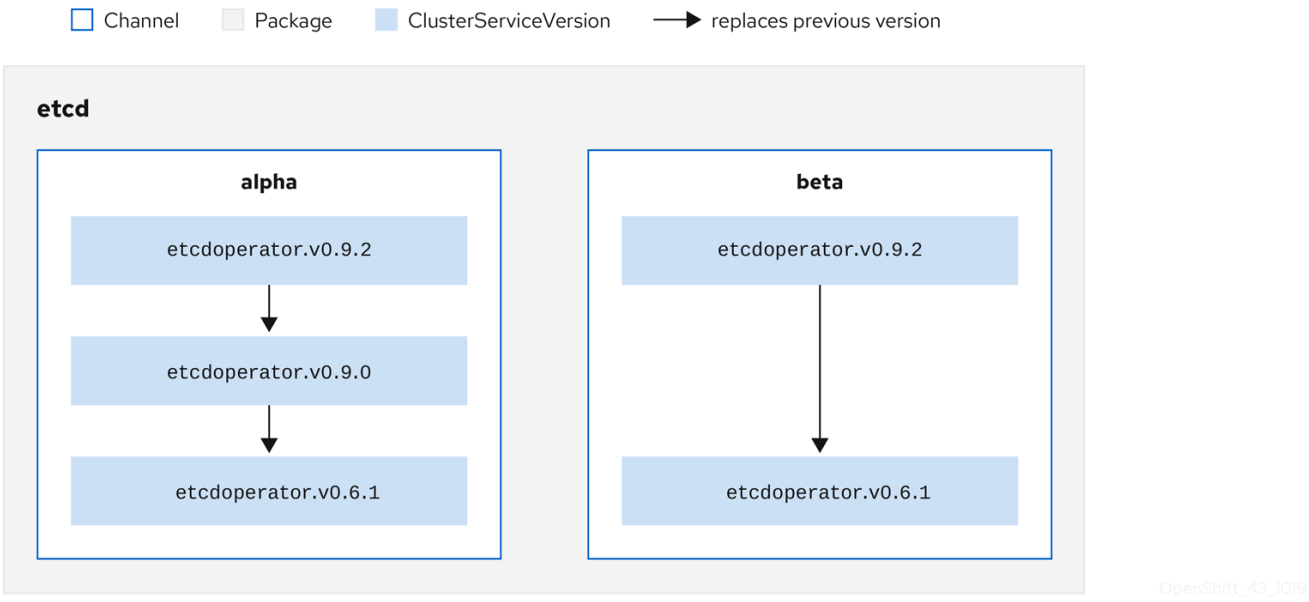
CSV 中定义的 Operator 元数据可保存在一个称为目录源的集合中。目录源使用 Operator Registry API，OLM 又使用目录源来查询是否有可用的 Operator 及已安装 Operator 是否有升级版本。

图 2.3. 目录源概述

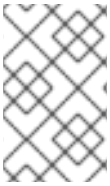


在目录源中，Operator 被整合为更新*软件包*和更新流，我们称为*频道*，这应是 OpenShift Container Platform 或其他软件（如 Web 浏览器）在持续发行周期中的常见更新模式。

图 2.4. 目录源中的软件包和频道



用户在*订阅*中的特定目录源中指示特定软件包和频道，如 **etcd** 包及其 **alpha** 频道。如果订阅了命名空间中尚未安装的软件包，则会安装该软件包的最新 Operator。

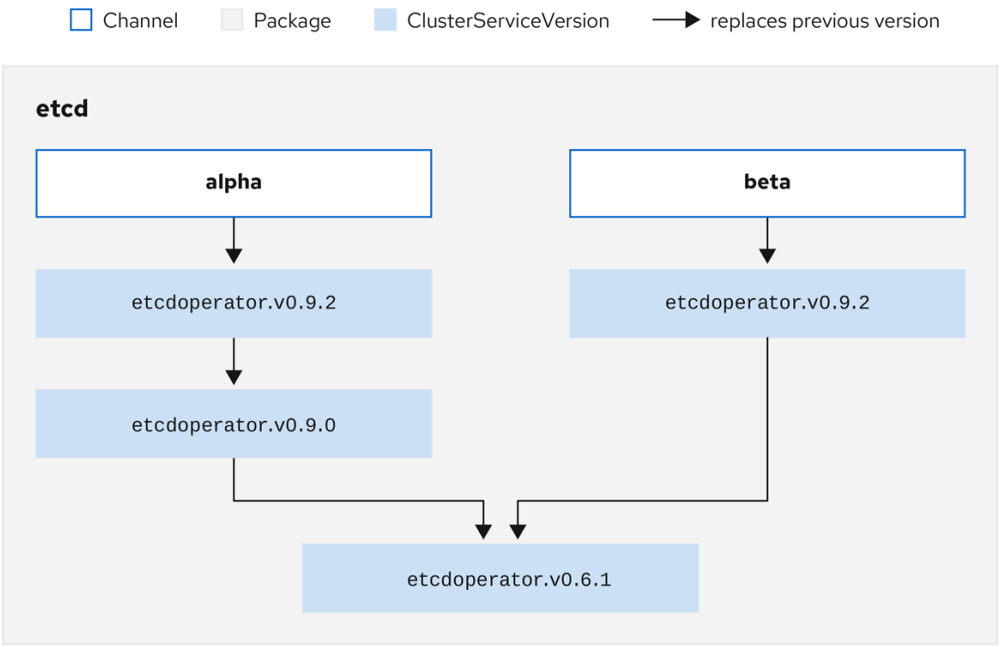


注意

OLM 会刻意避免版本比较，因此给定 *catalog* → *channel* → *package* 路径提供的“latest”或“newest”Operator 不一定是最高版本号。更应将其视为频道的 *head* 引用，类似 Git 存储库。

每个 CSV 均有一个 **replaces** 参数，指明所替换的是哪个 Operator。这样便构建了一个可通过 OLM 查询的 CSV 图，且不同频道之间可共享更新。可将频道视为更新图表的入口点：

图 2.5. OLM 的可用频道更新图表



软件包中的频道示例

```
packageName: example
channels:
- name: alpha
  currentCSV: example.v0.1.2
- name: beta
  currentCSV: example.v0.1.3
defaultChannel: alpha
```

为了让 OLM 成功查询更新、给定一个目录源、软件包、频道和 CSV，目录必须能够明确无误地返回替换输入 CSV 的单个 CSV。

2.4.3.1.1. 升级路径示例

对于示例升级场景，假设安装的 Operator 对应于 **0.1.1** 版 CSV。OLM 查询目录源，并在订阅的频道中检测升级，新的 **0.1.3** 版 CSV 替换了旧的但未安装的 **0.1.2** 版 CSV，后者又取代了较早且已安装的 **0.1.1** 版 CSV。

OLM 通过 CSV 中指定的 **replaces** 字段从频道头倒退至之前的版本，以确定升级路径为 **0.1.3** → **0.1.2** → **0.1.1**，其中箭头代表前者取代后者。OLM 一次仅升级一个 Operator 版本，直至到达频道头。

对于该给定场景，OLM 会安装 **0.1.2** 版 Operator 来取代现有的 **0.1.1** 版 Operator。然后再安装 **0.1.3** 版 Operator 来取代之前安装的 **0.1.2** 版 Operator。至此，所安装的 **0.1.3** 版 Operator 与频道头相匹配，意味着升级已完成。

2.4.3.1.2. 跳过升级

OLM 中升级的基本路径是：

- 通过对 Operator 的一个或多个更新来更新目录源。
- OLM 会遍历 Operator 的所有版本，直到到达目录源包含的最新版本。

但有时这不是一种安全操作。某些情况下，已发布但尚未就绪的 Operator 版本不可安装至集群中，如版本中存在严重漏洞。

这种情况下，OLM 必须考虑两个集群状态，并提供支持这两个状态的更新图：

- 集群发现并安装了"不良"中间 Operator。
- "不良"中间 Operator 尚未安装至集群中。

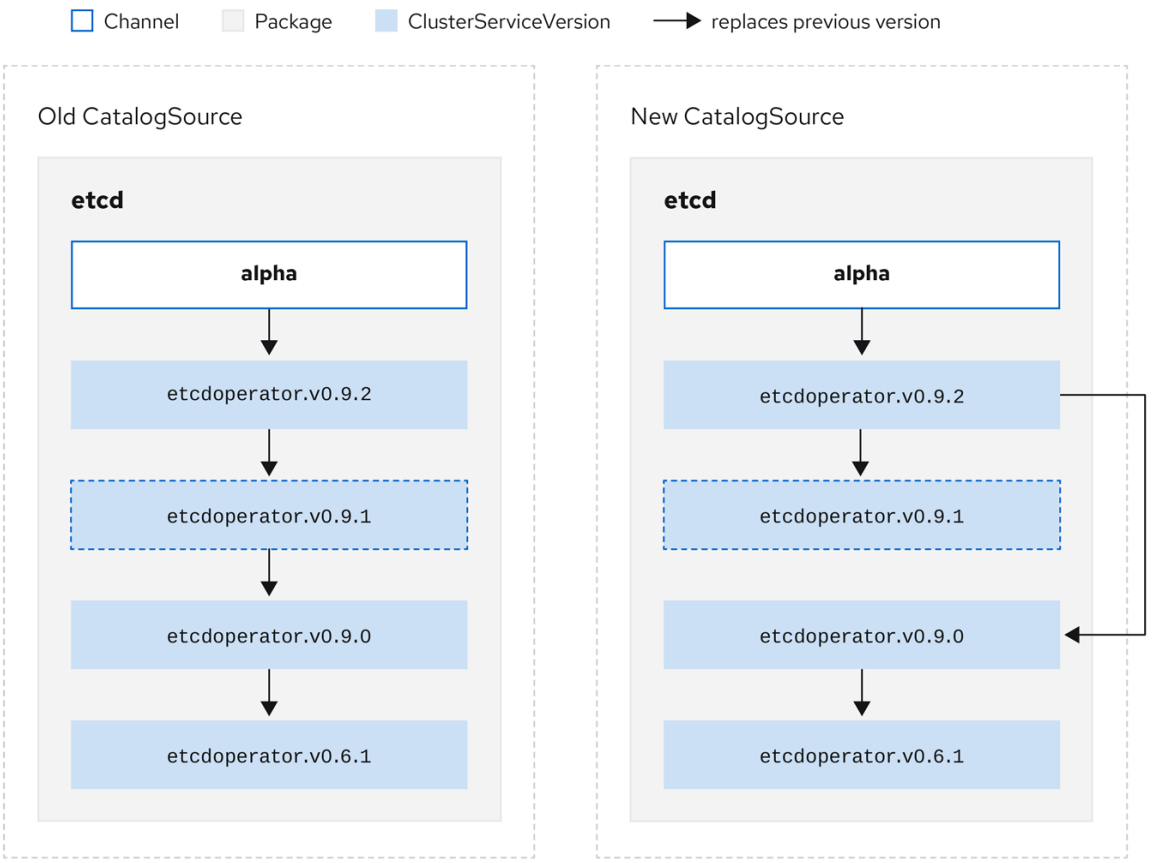
通过发送新目录并添加*跳过的*发行版本，可保证无论集群状态如何以及是否发现了不良更新，OLM 总能获得单个唯一更新。

带有跳过发行版本的 CSV 示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: etcdoperator.v0.9.2
  namespace: placeholder
  annotations:
spec:
    displayName: etcd
    description: Etcd Operator
    replaces: etcdoperator.v0.9.0
    skips:
    - etcdoperator.v0.9.1
```

考虑以下 **Old CatalogSource** 和 **New CatalogSource** 示例。

图 2.6. 跳过更新



该图表明：

- **Old CatalogSource** 中的任何 Operator 在 **New CatalogSource** 中均有单一替换项。
- **New CatalogSource** 中的任何 Operator 在 **New CatalogSource** 中均有单一替换项。
- 如果未曾安装不良更新，将来也绝不会安装。

2.4.3.1.3. 替换多个 Operator

按照描述创建 **New CatalogSource** 需要发布 CSV 来**替换**单个 Operator，但可**跳过**多个。该操作可通过 **skipRange** 注解来完成：

```
olm.skipRange: <semver_range>
```

其中 **<semver_range>** 具有 semver library 所支持的版本范围格式。

当在目录中搜索更新时，如果某个频道头提供一个 **skipRange** 注解，且当前安装的 Operator 的版本字段在该范围内，则 OLM 会更新至该频道中的最新条目。

先后顺序：

1. Subscription 上由 **sourceName** 指定的源中的频道头（满足其他跳过条件的情况下）。
2. 在 **sourceName** 指定的源中替换当前 Operator 的下一 Operator。
3. 对 Subscription 可见的另一个源中的频道头（满足其他跳过条件的情况下）。

4. 在对 Subscription 可见的任何源中替换当前 Operator 的下一 Operator。

带有 skipRange 的 CSV 示例

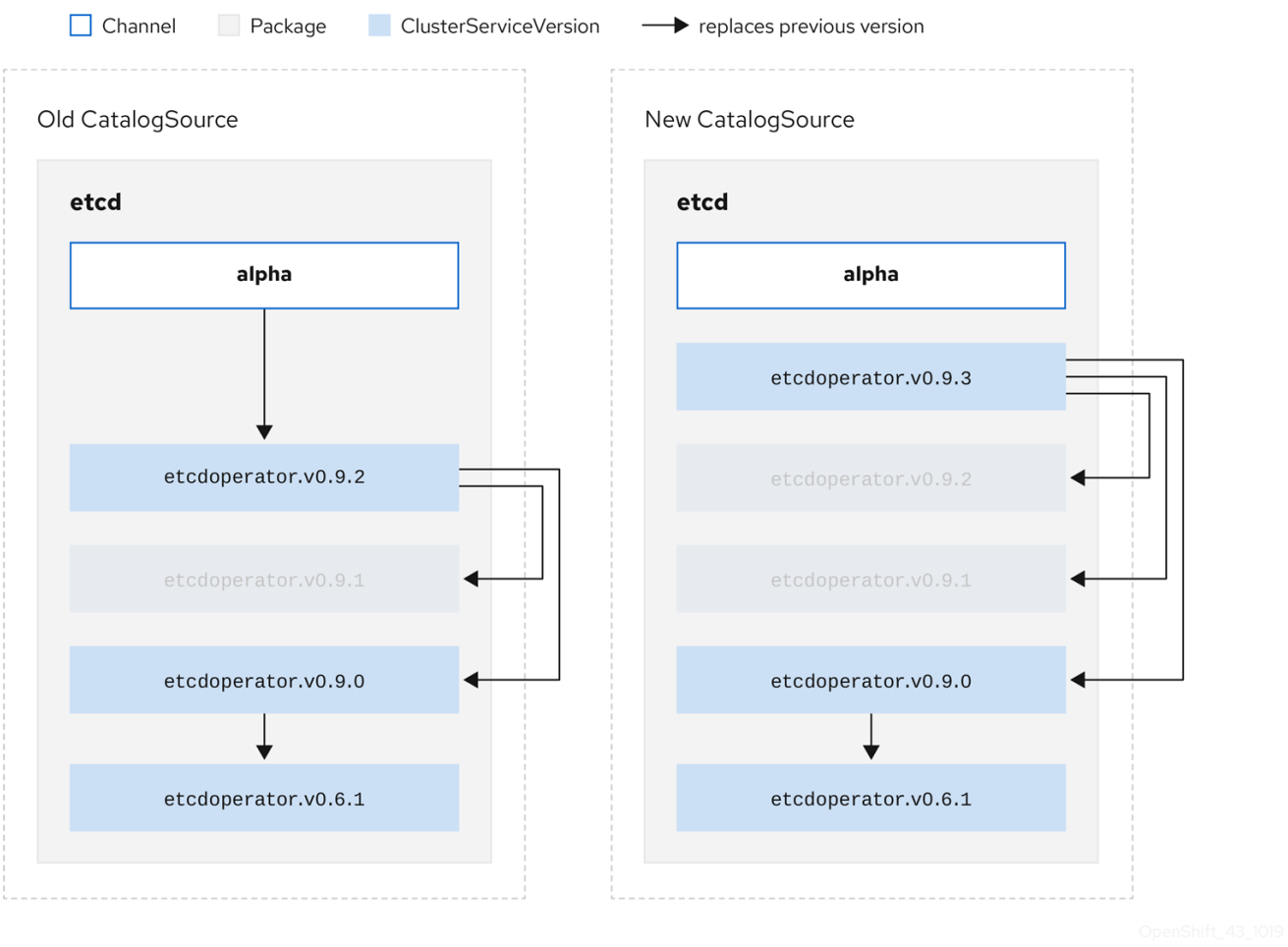
```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: elasticsearch-operator.v4.1.2
  namespace: <namespace>
  annotations:
    olm.skipRange: '>=4.1.0 <4.1.2'
```

2.4.3.1.4. Z-stream 支持

对于相同从版本，*z-stream* 或补丁版本必须取代所有先前 z-stream 版本。OLM 不考虑主版本、次版本或补丁版本，只需要在目录中构建正确的图表。

换句话说，OLM 必须能够像在 **Old CatalogSource** 中一样获取一个图表，像在 **New CatalogSource** 中一样生成一个图表：

图 2.7. 替换多个 Operator



该图表明：

- **Old CatalogSource** 中的任何 Operator 在 **New CatalogSource** 中均有单一替换项。
- **New CatalogSource** 中的任何 Operator 在 **New CatalogSource** 中均有单一替换项。

- **Old CatalogSource** 中的所有 z-stream 版本均会更新至 **New CatalogSource** 中最新 z-stream 版本。
- 不可用版本可被视为"虚拟"图表节点；它们的内容无需存在，注册表只需像图表看上去这样响应即可。

2.4.4. Operator Lifecycle Manager 依赖项解析

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager (OLM) 内的依赖项解析和自定义资源定义 (CRD) 升级生命周期。

2.4.4.1. 关于依赖项解析

Operator Lifecycle Manager(OLM)管理运行 Operator 的依赖项解析和升级生命周期。在很多方面，OLM 的问题与其他系统或语言软件包管理器类似，如 **yum** 和 **rpm**。

但其中有一个限制是相似系统一般不存在而 OLM 存在的，那就是：因为 Operator 始终在运行，所以 OLM 会努力确保您所接触的 Operator 组始终相互兼容。

因此，OLM 不得创建以下情况：

- 安装一组需要无法提供的 API 的 Operator
- 更新某个 Operator 之时导致依赖该 Operator 的另一 Operator 中断

这可以通过两种类型的数据：

Properties	在依赖项解析器中输入构成了公共接口的 Operator 元数据。示例包括 Operator 提供的 API 的 group/version/kind(GVK)，以及 Operator 的语义版本(semver)。
约束或依赖项	应该对可能或还没有在目标集群中安装的其他 Operator 满足 Operator 的要求。它们充当所有可用 Operator 的查询或过滤，并在依赖项解析和安装过程中限制选择。例如，需要特定的 API 在集群中可用，或希望安装带有特定版本的特定 Operator。

OLM 将这些属性和约束转换为布尔值公式系统，并将其传递给 SAT solver，SAT solver 是一个处理布尔值的程序，用于确定应该安装哪些 Operator。

2.4.4.2. Operator 属性

目录中的所有 Operator 均具有以下属性：

olm.package

包括软件包和 Operator 版本的名称

olm.gvk

集群服务版本(CSV)中每个提供的 API 的单个属性

Operator 作者也可以在 Operator 捆绑包的 **metadata/** 目录中包括 **properties.yaml** 文件来直接声明其他属性。

任意（arbitrary）属性示例

```
properties:
```

```
- type: olm.kubeversion
  value:
    version: "1.16.0"
```

2.4.4.2.1. 任意属性

Operator 作者可在 Operator 捆绑包的 **metadata/** 目录中的 **properties.yaml** 文件中声明任意属性。这些属性转换为映射数据结构，该结构用作运行时 Operator Lifecycle Manager(OLM)解析器的输入。

这些属性对解析器不理解属性而不理解这些属性，但可以针对这些属性评估通用限制，以确定约束是否可以满足给定的属性列表。

任意属性示例

```
properties:
  - property:
      type: color
      value: red
  - property:
      type: shape
      value: square
  - property:
      type: olm.gvk
      value:
        group: olm.coreos.io
        version: v1alpha1
        kind: myresource
```

此结构可用于为通用限制构建通用表达式语言(CEL)表达式。

其他资源

- 常见表达式语言(CEL)约束

2.4.4.3. Operator 依赖项

Operator 的依赖项列在捆绑包的 **metadata/** 目录中的 **dependencies.yaml** 文件中。此文件是可选的，目前仅用于指明 Operator-version 依赖项。

依赖项列表中，每个项目包含一个 **type** 字段，用于指定这一依赖项的类型。支持以下 Operator 依赖项：

olm.package

这个类型表示特定 Operator 版本的依赖项。依赖项信息必须包含软件包名称以及软件包的版本，格式为 semver。例如，您可以指定具体版本，如 **0.5.2**，也可指定一系列版本，如 **>0.5.1**。

olm.gvk

使用这个类型，作者可以使用 group/version/kind(GVK)信息指定依赖项，类似于 CSV 中现有 CRD 和基于 API 的使用量。该路径使 Operator 作者可以合并所有依赖项、API 或显式版本，使它们处于同一位置。

olm.constraint

这个类型在任意 Operator 属性上声明通用限制。

在以下示例中，为 Prometheus Operator 和 etcd CRD 指定依赖项：

dependencies.yaml 文件示例

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
      group: etcd.database.coreos.com
      kind: EtcdCluster
      version: v1beta2
```

2.4.4.4. 通用限制

olm.constraint 属性声明特定类型的依赖项约束，区分非约束和约束属性。其 **value** 字段是一个包含 **failureMessage** 字段的对象，其中包含约束消息的字符串表。如果约束在运行时不满意，则这一消息被作为信息性提供给用户使用。

以下键表示可用的约束类型：

gvk

其值及对其的解释与 **olm.gvk** 类型相同的类型

package

其值及对其的解释与 **olm.package** 类型相同的类型

cel

Operator Lifecycle Manager(OLM)解析程序通过任意捆绑包属性和集群信息在运行时评估的通用表达式语言(CEL)表达式

all, any, not

分别为 Conjunction, disjunction, 和 negation 约束，包括一个或多个 concrete 约束，如 **gvk** 或一个嵌套的 compound 约束

2.4.4.4.1. 常见表达式语言(CEL)约束

cel 约束类型支持将通用表达式语言(CEL) 用作表达式语言。**cel** struct 有一个 **rule** 字段，其中包含在运行时针对 Operator 属性评估的 CEL 表达式字符串，以确定 Operator 是否满足约束。

cel 约束示例

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified"'
  cel:
    rule: 'properties.exists(p, p.type == "certified")'
```

CEL 语法支持广泛的逻辑运算符，如 **AND** 和 **OR**。因此，单个 CEL 表达式可以具有多个规则，这些条件由这些逻辑运算符链接在一起。这些规则针对来自捆绑包或任何给定源的多个不同属性的数据评估，输出可以解决单一约束内满足所有这些规则的捆绑包或 Operator。

使用多个规则的 cel 约束示例

■

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified" and "stable" properties'
  cel:
    rule: 'properties.exists(p, p.type == "certified") && properties.exists(p, p.type == "stable")'
```

2.4.4.4.2. Compound 约束 (all, any, not)

复合约束类型按照其逻辑定义进行评估。

以下是两个软件包的 conjunctive 约束（**all**）的示例，以及一个 GVK。这代表，安装捆绑包都必须满足它们：

all 约束示例

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: All are required for Red because...
    all:
      constraints:
      - failureMessage: Package blue is needed for...
        package:
          name: blue
          versionRange: '>=1.0.0'
      - failureMessage: GVK Green/v1 is needed for...
        gvk:
          group: greens.example.com
          version: v1
          kind: Green
```

以下是同一个 GVK 的三个版本的 disjunctive 约束 (**any**) 示例。这代表，安装捆绑包必须至少满足其中一项：

any 约束示例

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Any are required for Red because...
    any:
      constraints:
      - gvk:
          group: blues.example.com
          version: v1beta1
          kind: Blue
      - gvk:
          group: blues.example.com
          version: v1beta2
          kind: Blue
      - gvk:
```

```
        group: blues.example.com
        version: v1
        kind: Blue
```

以下是 GVK 的一个版本的 negation 约束（**not**）的示例。这代表，此 GVK 无法由结果集中的任何捆绑包提供：

not 约束示例

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
  all:
    constraints:
    - failureMessage: Package blue is needed for...
      package:
        name: blue
        versionRange: '>=1.0.0'
    - failureMessage: Cannot be required for Red because...
      not:
        constraints:
        - gvk:
            group: greens.example.com
            version: v1alpha1
            kind: greens
```

对于 **not** 约束，其中的负语义可能并不明确。这里的负语义代表指示解析器删除所有可能的解决方案，这些解决方案包括特定 GVK、特点版本的软版本，或满足结果集中的一些子复合约束。

not compound 约束不应该和 **all** 或 **any** 一起使用，因为这里的负语言在没有先选择一组可能的依赖项时是并没有意义。

2.4.4.4.3. 嵌套复合限制

一个嵌套复合约束（包括最少一个子复合约束以及零个或更多简单约束）会从底向上的顺序被评估，并根据每个前面描述的约束类型的过程进行。

以下是一个 disjunction 的 conjunctions 示例，其中一个、另一个、或两者都能满足约束：

嵌套复合约束示例

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Required for Red because...
    any:
      constraints:
      - all:
          constraints:
          - package:
              name: blue
```

```
        versionRange: '>=1.0.0'
    - gvk:
        group: blues.example.com
        version: v1
        kind: Blue
  - all:
      constraints:
      - package:
          name: blue
          versionRange: '<1.0.0'
      - gvk:
          group: blues.example.com
          version: v1beta1
          kind: Blue
```

注意

olm.constraint 类型的最大原始大小为 64KB，用于限制资源耗尽的情况。

2.4.4.5. 依赖项首选项

有很多选项同样可以满足 Operator 的依赖性。Operator Lifecycle Manager（OLM）中的依赖项解析器决定哪个选项最适合所请求 Operator 的要求。作为 Operator 作者或用户，了解这些选择非常重要，以便明确依赖项解析。

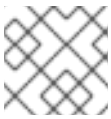
2.4.4.5.1. 目录优先级

在 OpenShift Container Platform 集群中，OLM 会读取目录源以了解哪些 Operator 可用于安装。

CatalogSource 对象示例

```
apiVersion: "operators.coreos.com/v1alpha1"
kind: "CatalogSource"
metadata:
  name: "my-operators"
  namespace: "operators"
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 1
  image: example.com/my/operator-index:v1
  displayName: "My Operators"
  priority: 100
```

1 指定 **legacy** 或 **restricted** 的值。如果没有设置该字段，则默认值为 **legacy**。在以后的 OpenShift Container Platform 发行版本中，计划默认值为 **restricted**。



注意

如果您的目录无法使用 **restricted** 权限运行，建议您手动将此字段设置为 **legacy**。

CatalogSource 有一个 **priority** 字段，解析器使用它来知道如何为依赖关系设置首选选项。

目录首选项有两个规则：

- 优先级较高目录中的选项优先于较低优先级目录的选项。
- 与依赖项相同的目录里的选项优先于其它目录。

2.4.4.5.2. 频道排序

目录中的 Operator 软件包是用户可在 OpenShift Container Platform 集群中订阅的更新频道集合。可使用频道为次发行版本（**1.2**, **1.3**）或者发行的频率（**stable**, **fast**）提供特定的更新流。

同一软件包中的 Operator 可能会满足依赖项，但可能会在不同的频道。例如，Operator 版本 **1.2** 可能存在于 **stable** 和 **fast** 频道中。

每个软件包都有一个默认频道，该频道总是首选非默认频道。如果默认频道中没有选项可以满足依赖关系，则会在剩余的频道中按频道名称的字母顺序考虑这些选项。

2.4.4.5.3. 频道中的顺序

一般情况下，总会有多个选项来满足单一频道中的依赖关系。例如，一个软件包和频道中的 Operator 提供了相同的 API 集。

当用户创建订阅时，它们会指示要从哪个频道接收更新。这会立即把搜索范围限制在那个频道。但是在频道中，可以会有许多 Operator 可以满足依赖项。

在频道中，应该首选考虑使用更新图中位置较高的较新的 Operator。如果某个频道的头满足依赖关系，它将会被首先尝试。

2.4.4.5.4. 其他限制

除了软件包依赖关系的限制外，OLM 还添加了其他限制来代表所需用户状态和强制实施解析变量。

2.4.4.5.4.1. 订阅约束

一个订阅（Subscription）约束会过滤可满足订阅的 Operator 集合。订阅是对依赖项解析程序用户提供的限制。它们会声明安装一个新的 Operator（如果还没有在集群中安装），或对现有 Operator 进行更新。

2.4.4.5.4.2. 软件包约束

在命名空间中，不同的两个 Operator 不能来自于同一软件包。

2.4.4.5.5. 其他资源

- 目录健康要求

2.4.4.6. CRD 升级

如果自定义资源定义（CRD）属于单一集群服务版本（CSV），OLM 会立即对其升级。如果某个 CRD 被多个 CSV 拥有，则当该 CRD 满足以下所有向后兼容条件时才会升级：

- 所有已存在于当前 CRD 中的服务版本都包括在新 CRD 中。
- 在根据新 CRD 的验证模式（schema）进行验证后，与 CRD 的服务版本关联的所有现有实例或自定义资源均有效。

2.4.4.7. 依赖项最佳实践

在指定依赖项时应该考虑的最佳实践。

依赖于 API 或 Operator 的特定版本范围

操作员可以随时添加或删除 API ; 始终针对 Operator 所需的任何 API 指定 **olm.gvk** 依赖项。例外情况是，指定 **olm.package** 约束来替代。

设置最小版本

Kubernetes 文档中与 API 的改变相关的部分描述了 Kubernetes 风格的 Operator 允许进行哪些更改。只要 API 向后兼容，Operator 就允许 Operator 对 API 进行更新，而不需要更改 API 的版本。
对于 Operator 依赖项，这意味着了解依赖的 API 版本可能不足以确保依赖的 Operator 正常工作。

例如：

- TestOperator v1.0.0 提供 **MyObject** 资源的 v1alpha1 API 版本。
- TestOperator v1.0.1 为 **MyObject** 添加了一个新的 **spec.newfield** 字段，但仍是 v1alpha1。

您的 Operator 可能需要将 **spec.newfield** 写入 **MyObject** 资源。仅使用 **olm.gvk** 约束还不足以让 OLM 决定您需要 TestOperator v1.0.1 而不是 TestOperator v1.0.0。

如果事先知道提供 API 的特定 Operator，则指定额外的 **olm.package** 约束来设置最小值。

省略一个最大版本，或允许一个广泛的范围

因为 Operator 提供了集群范围的资源，如 API 服务和 CRD，所以如果一个 Operator 为依赖项指定了一个小的窗口，则可能会对依赖项的其他用户的更新产生不必要的约束。
在可能的情况下，尽量不要设置最大版本。或者，设置一个非常宽松的语义范围，以防止与其他 Operator 冲突。例如：**>1.0.0 <2.0.0**。

与传统的软件包管理器不同，Operator 作者显性地对更新通过 OLM 中的频道进行编码。如果现有订阅有可用更新，则假定 Operator 作者表示它可以从上一版本更新。为依赖项设置最大版本会绕过作者的更新流，即不必要的将它截断到特定的上限。

注意

集群管理员无法覆盖 Operator 作者设置的依赖项。

但是，如果已知有需要避免的不兼容问题，就应该设置最大版本。通过使用版本范围语法，可以省略特定的版本，如 **> 1.0.0 !1.2.1**。

其他资源

- Kubernetes 文档： 更改 API

2.4.4.8. 依赖项注意事项

当指定依赖项时，需要考虑一些注意事项。

没有捆绑包约束（AND）

目前还没有方法指定约束间的 AND 关系。换句话说，无法指定一个 Operator，它依赖于另外一个 Operator，它提供一个给定的 API 且版本是 **>1.1.0**。
这意味着，在指定依赖项时，如：

```
dependencies:
- type: olm.package
  value:
    packageName: etcd
    version: ">3.1.0"
- type: olm.gvk
  value:
    group: etcd.database.coreos.com
    kind: EtcdCluster
    version: v1beta2
```

OLM 可以通过两个 Operator 来满足这个要求：一个提供 EtcdCluster，另一个有版本 **>3.1.0**。是否发生了这种情况，或者选择某个 Operator 是否满足这两个限制，这取决于是否准备了潜在的选项。依赖项偏好和排序选项被明确定义并可以指定原因，但为了谨慎起见，Operator 应该遵循一种机制或其他机制。

跨命名空间兼容性

OLM 在命名空间范围内执行依赖项解析。如果更新某个命名空间中的 Operator 会对另一个命名空间中的 Operator 造成问题，则可能会造成更新死锁。

2.4.4.9. 依赖项解析方案示例

在以下示例中，*provider（供应商）* 是指"拥有" CRD 或 API 服务的 Operator。

2.4.4.9.1. 示例：弃用从属 API

A 和 B 是 API（CRD）：

- A 的供应商依赖 B。
- B 的供应商有一个订阅。
- B 更新供应商提供 C，但弃用 B。

结果：

- B 不再有供应商。
- A 不再工作。

这是 OLM 通过升级策略阻止的一个案例。

2.4.4.9.2. 示例：版本死锁

A 和 B 均为 API：

- A 的供应商需要 B。
- B 的供应商需要 A。
- A 更新的供应商到（提供 A2，需要 B2）并弃用 A。
- B 更新的供应商到（提供 B2,需要 A2）并弃用 B。

如果 OLM 试图在更新 A 的同时不更新 B，或更新 B 的同时不更新 A，则无法升级到新版 Operator，即使可找到新的兼容集也无法更新。

这是 OLM 通过升级策略阻止的另一案例。

2.4.5. operator 组

本指南概述了 OpenShift Container Platform 中 Operator Lifecycle Manager（OLM）的 Operator 组使用情况。

2.4.5.1. 关于 Operator 组

由 **OperatorGroup** 资源定义的 *Operator 组*，为 OLM 安装的 Operator 提供多租户配置。Operator 组选择目标命名空间，在其中为其成员 Operator 生成所需的 RBAC 访问权限。

这一组目标命名空间通过存储在 CSV 的 **olm.targetNamespaces** 注解中的以逗号分隔的字符串来提供。该注解应用于成员 Operator 的 CSV 实例，并注入它们的部署中。

2.4.5.2. Operator 组成员

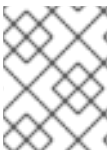
满足以下任一条件，Operator 即可被视为 Operator 组的 *member*：

- Operator 的 CSV 与 Operator 组位于同一命名空间中。
- Operator CSV 中的安装模式支持 Operator 组的目标命名空间集。

CSV 中的安装模式由 **InstallModeType** 字段和 **Supported** 的布尔值字段组成。CSV 的 spec 可以包含一组由四个不同 **InstallModeTypes** 组成的安装模式：

表 2.5. 安装模式和支持的 Operator 组

InstallModeType	描述
OwnNamespace	Operator 可以是选择其自有命名空间的 Operator 组的成员。
SingleNamespace	Operator 可以是选择一个命名空间的 Operator 组的成员。
MultiNamespace	Operator 可以是选择多个命名空间的 Operator 组的成员。
AllNamespaces	Operator 可以是选择所有命名空间的 Operator 组的成员（目标命名空间集为空字符串 **""**）。



注意

如果 CSV 的 spec 省略 **InstallModeType** 条目，则该类型将被视为不受支持，除非可通过隐式支持的现有条目推断出支持。

2.4.5.3. 目标命名空间选择

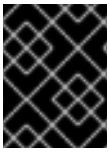
您可以使用 **spec.targetNamespaces** 参数为 Operator 组显式命名目标命名空间：

```
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  targetNamespaces:
  - my-namespace
```

您还可以使用带有 **spec.selector** 参数的标签选择器指定命名空间：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  selector:
    cool.io/prod: "true"
```



重要

不建议通过 **spec.targetNamespaces** 列出多个命名空间，或通过 **spec.selector** 使用标签选择器，因为在以后的版本中可能会删除对 Operator 组中多个目标命名空间的支持。

如果 **spec.targetNamespaces** 和 **spec.selector** 均已定义，则会忽略 **spec.selector**。另外，您可以省略 **spec.selector** 和 **spec.targetNamespaces** 来指定一个 *全局* Operator 组，该组选择所有命名空间：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
```

Opeator 组的 **status.namespaces** 参数中会显示所选命名空间的解析集合。全局 OperatorGroup 的 **status.namespace** 包含空字符串 (**""**)，而该字符串会向正在使用的 Operator 发出信号，要求其监视所有命名空间。

2.4.5.4. operator 组 CSV 注解

Operator 组的成员 CSV 具有以下注解：

注解	描述
olm.operatorGroup=<group_name>	包含 Operator 组的名称。
olm.operatorNamespace=<group_namespace>	包含 Operator 组的命名空间。
olm.targetNamespaces=<target_namespaces>	包含以逗号分隔的字符串，列出 Operator 组的目标命名空间选择。

注意

除 **olm.targetNamespaces** 以外的所有注解均包含在复制的 CSV 中。在复制的 CSV 上省略 **olm.targetNamespaces** 注解可防止租户之间目标命名空间出现重复。

2.4.5.5. 所提供的 API 注解

group/version/kind (GVK) 是 Kubernetes API 的唯一标识符。 **olm.providedAPIs** 注解中会显示有关 Operator 组提供哪些 GVK 的信息。该注解值为一个字符串，由用逗号分隔的 **<kind>.<version>.<group>** 组成。其中包括由 Operator 组的所有活跃成员 CSV 提供的 CRD 和 APIService 的 GVK。

查看以下 **OperatorGroup** 示例，该 OperatorGroup 带有提供 **PackageManifest** 资源的单个活跃成员 CSV：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: PackageManifest.v1alpha1.packages.apps.redhat.com
  name: olm-operators
  namespace: local
  ...
spec:
  selector: {}
  serviceAccountName:
    metadata:
      creationTimestamp: null
  targetNamespaces:
  - local
status:
  lastUpdated: 2019-02-19T16:18:28Z
  namespaces:
  - local
```

2.4.5.6. 基于角色的访问控制

创建 Operator 组时，会生成三个集群角色。当生成集群角色时，会自动在最后添加一个哈希值，以确保每个集群角色都是唯一的。每个 Operator 组均包含一个聚会规则，后者带有一个选择器以匹配标签，如下所示：

集群角色	要匹配的标签
olm.og.<operatorgroup_name>-admin-<hash_value>	**olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>**
olm.og.<operatorgroup_name>-edit-<hash_value>	**olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>**
olm.og.<operatorgroup_name>-view-<hash_value>	**olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

注意

要使用 Operator 组的集群角色为资源分配基于角色的访问控制 (RBAC)，请运行以下命令获取包括集群角色和哈希值的完整名称：

```
$ oc get clusterroles | grep <operatorgroup_name>
```

因为哈希值是在创建 Operator 组时生成的，所以在查找集群角色的完整名称前，需要先创建 Operator 组。

当 CSV 成为 Operator 组的活跃成员时，只要该 CSV 正在使用 **AllNamespaces** 安装模式来监视所有命名空间，且没有因 **InterOperatorGroupOwnerConflict** 原因处于故障状态，便会生成以下 RBAC 资源：

- 来自 CRD 的每个 API 资源的集群角色
- 来自 API 服务的每个 API 资源的集群角色
- 其他角色和角色绑定

表 2.6. 来自 CRD 的为每个 API 资源生成的集群角色

集群角色	设置
<kind>.<group>-<version>-admin	**<kind>** 上的操作动词： • ***** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-admin: true** • **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>**
<kind>.<group>-<version>-edit	**<kind>** 上的操作动词： • **create** • **update** • **patch** • **delete** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-edit: true** • **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>**

集群角色	设置
<kind>.<group>-<version>-view	**<kind>** 上的操作动词： • **get** • **list** • **watch** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-view: true** • **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**
<kind>.<group>-<version>-view-crdview	**apiextensions.k8s.io customresourcedefinitions <crd-name>** 上的操作动词： • **get** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-view: true** • **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

表 2.7. 来自 API 服务的为每个 API 资源生成的集群角色

集群角色	设置
<kind>.<group>-<version>-admin	**<kind>** 上的操作动词： • *** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-admin: true** • **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>**

集群角色	设置
<kind>.<group>-<version>-edit	**<kind>** 上的操作动词： • **create** • **update** • **patch** • **delete** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-edit: true** • **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>**
<kind>.<group>-<version>-view	**<kind>** 上的操作动词： • **get** • **list** • **watch** 聚合标签： • **rbac.authorization.k8s.io/aggregate-to-view: true** • **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

其他角色和角色绑定

- 如果 CSV 定义了一个目标命名空间，其中包括 *，则会针对 CSV 权限字段中定义的每个 **permissions** 生成集群角色和对应集群角色绑定。所有生成的资源均会标上 **olm.owner: <csv_name>** 和 **olm.owner.namespace: <csv_namespace>** 标签。
- 如果 CSV *没有*定义一个包含 * 的目标命名空间，则 Operator 命名空间中的所有角色和角色绑定都使用 **olm.owner: <csv_name>** 和 **olm.owner.namespace: <csv_namespace>** 标签复制到目标命名空间中。

2.4.5.7. 复制的 CSV

OLM 会在 Operator 组的每个目标命名空间中创建 Operator 组的所有活跃成员 CSV 的副本。复制 CSV 的目的在于告诉目标命名空间的用户，特定 Operator 已配置为监视在此创建的资源。

复制的 CSV 会**复制**状态原因，并会更新以匹配其源 CSV 的状态。在集群上创建复制的 CSV 之前，会从这些 CSV 中分离 **olm.targetNamespaces** 注解。省略目标命名空间选择可避免租户之间存在目标命名空间重复的现象。

当所复制的 CSV 的源 CSV 不存在或其源 CSV 所属的 Operator 组不再指向复制 CSV 的命名空间时，会删除复制的 CSV。

注意

默认情况下禁用 **disableCopiedCSVs** 字段。启用 **disableCopiedCSVs** 字段后，OLM 会删除集群中的现有复制的 CSV。当 **disableCopiedCSVs** 字段被禁用时，OLM 会再次添加复制的 CSV。

- 禁用 **disableCopiedCSVs** 字段：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: false
EOF
```

- 启用 **disableCopiedCSVs** 字段：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: true
EOF
```

2.4.5.8. 静态 Operator 组

如果 Operator 组的 **spec.staticProvidedAPIs** 字段被设置为 **true**，则 Operator 组为*静态*。因此，OLM 不会修改 Operator 组的 **olm.providedAPIs** 注解，这意味着可以提前设置它。如果一组命名空间没有活跃的成员 CSV 来为资源提供 API，而用户想使用 Operator 组来防止命名空间集中发生资源争用，则这一操作十分有用。

以下是一个 Operator 组示例，它使用 **something.cool.io/cluster-monitoring: "true"** 注解来保护所有命名空间中的 **Prometheus** 资源：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-monitoring
  namespace: cluster-monitoring
  annotations:
    olm.providedAPIs:
Alertmanager.v1.monitoring.coreos.com,Prometheus.v1.monitoring.coreos.com,PrometheusRule.v1.monitoring.coreos.com,ServiceMonitor.v1.monitoring.coreos.com
spec:
  staticProvidedAPIs: true
```

```
selector:
  matchLabels:
    something.cool.io/cluster-monitoring: "true"
```

2.4.5.9. operator 组交集

如果两个 Operator 组的目标命名空间集的交集不是空集，且根据 **olm.providedAPIs** 注解的定义，所提供的 API 集的交集也不是空集，则称这两个 OperatorGroup 的*提供的 API 有交集*。

一个潜在问题是，提供的 API 有交集的 Operator 组可能在命名空间交集中竞争相同资源。

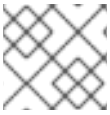
注意

在检查交集规则时，Operator 组的命名空间始终包含在其所选目标命名空间中。

2.4.5.9.1. 交集规则

每次活跃成员 CSV 同步时，OLM 均会查询集群，以获取 CSV 组和其他所有 CSV 组之间提供的 API 交集。然后 OLM 会检查该交集是否为空集：

- 如果结果为 **true**，且 CSV 提供的 API 是 Operator 组提供的 API 的子集：
  - 继续转变。
- 如果结果为 **true**，且 CSV 提供的 API *不是* Operator 组提供的 API 的子集：
  - 如果 Operator 组是静态的：
    - 则清理属于 CSV 的所有部署。
    - 将 CSV 转变为故障状态，状态原因为：**CannotModifyStaticOperatorGroupProvidedAPIs**。
  - 如果 Operator 组*不是*静态的：
    - 将 Operator 组的 **olm.providedAPIs** 注解替换为其本身与 CSV 提供的 API 的并集。
- 如果结果为 **false**，且 CSV 提供的 API *不是* Operator 组提供的 API 的子集：
  - 则清理属于 CSV 的所有部署。
  - 将 CSV 转变为故障状态，状态原因为：**InterOperatorGroupOwnerConflict**。
- 如果结果为 **false**，且 CSV 提供的 API 是 Operator 组提供的 API 的子集：
  - 如果 Operator 组是静态的：
    - 则清理属于 CSV 的所有部署。
    - 将 CSV 转变为故障状态，状态原因为：**CannotModifyStaticOperatorGroupProvidedAPIs**。
  - 如果 Operator 组*不是*静态的：
    - 将 Operator 组的 **olm.providedAPIs** 注解替换为其本身与 CSV 提供的 API 的差集。



注意

Operator 组所造成的故障状态不是终端状态。

每次 Operator 组同步时都会执行以下操作：

- 来自活跃成员 CSV 的提供的 API 集是通过集群计算出来的。注意，复制的 CSV 会被忽略。
- 将集群集与 **olm.providedAPIs** 进行比较，如果 **olm.providedAPIs** 包含任何额外 API，则将删除这些 API。
- 在所有命名空间中提供相同 API 的所有 CSV 均会重新排序。这样可向交集组中的冲突 CSV 发送通知，表明可能已通过调整大小或删除冲突的 CSV 解决了冲突。

2.4.5.10. 多租户 Operator 管理的限制

OpenShift Container Platform 支持在同一集群中安装不同版本的 Operator。Operator Lifecycle Manager (OLM) 会在不同的命名空间中多次安装 Operator。其中一个限制是 Operator 的 API 版本必须相同。

Operator 是控制平面的扩展，因为它们使用了 **CustomResourceDefinition** 对象 (CRD)，它们是 Kubernetes 中的全局资源。一个 Operator 的不同主版本通常具有不兼容的 CRD。这使得它们不兼容，可以在集群中的不同命名空间中安装。

所有租户或命名空间共享同一集群的 control plane。因此，多租户集群中的租户也共享全局 CRD，这限制同一集群中可以并行使用同一 Operator 实例的不同 Operator 实例。

支持的场景包括：

- 提供相同 CRD 定义的不同版本的 Operator （如果版本化 CRD，则完全相同的版本）
- 没有提供 CRD 的不同版本的 Operator，并在 OperatorHub 上的单独捆绑包中提供它们的 CRD

不支持所有其他场景，因为如果不同 Operator 版本中的多个竞争或重叠 CRD 在同一集群中协调，则无法保证集群数据的完整性。

其他资源

- Operator Lifecycle Manager (OLM)→ Multitenancy 和 Operator colocation
- 多租户集群中的 Operator
- 允许非集群管理员安装 Operator

2.4.5.11. 对 Operator 组进行故障排除

2.4.5.11.1. 成员资格

- 安装计划的命名空间必须只包含一个 Operator 组。当尝试在命名空间中生成集群服务版本（CSV）时，安装计划会认为一个 Operator 组在以下情况下无效：
  - 安装计划的命名空间中没有 Operator 组。
  - 安装计划的命名空间中存在多个 Operator 组。
  - 在 Operator 组中指定不正确或不存在的服务帐户名称。

如果安装计划遇到无效的 Operator 组，则不会生成 CSV，**InstallPlan** 资源将继续使用相关消息进行安装。例如，如果同一命名空间中存在多个 Operator 组，则会提供以下信息：

```
attenuated service account query failed - more than one operator group(s) are managing this namespace count=2
```

其中 **count=** 指定命名空间中的 Operator 组数量。

- 如果 CSV 的安装模式不支持其命名空间中 Operator 组的目标命名空间选择，CSV 会转变为故障状态，原因为 **UnsupportedOperatorGroup**。处于故障状态的 CSV 会在 Operator 组的目标命名空间选择变为受支持的配置后转变为待处理，或者 CSV 的安装模式被修改来支持目标命名空间选择。

2.4.6. 多租户和 Operator 共处

本指南概述了 Operator Lifecycle Manager (OLM) 中的多租户和 Operator 共处。

2.4.6.1. 命名空间中的 Operator 共处

Operator Lifecycle Manager (OLM) 处理在同一命名空间中安装的 OLM 管理的 Operator，这意味着其 **Subscription** 资源与相关 Operator 位于同一个命名空间中。即使它们实际不相关，OLM 会在更新其中任何一个时考虑其状态，如它们的版本和更新策略。

这个默认行为清单可以通过两种方式：

- 待处理的更新的 **InstallPlan** 资源包括同一命名空间中的所有其他 Operator 的 **ClusterServiceVersion** (CSV) 资源。
- 同一命名空间中的所有 Operator 都共享相同的更新策略。例如，如果一个 Operator 设置为手动更新，则所有其他 Operator 更新策略也会设置为 manual。

这些场景可能会导致以下问题：

- 很难了解有关 Operator 更新安装计划的原因，因为它们中定义了除更新 Operator 以外的更多资源。
- 在命名空间更新中，无法自动更新一些 Operator，而其他 Operator 也无法手动更新，这对集群管理员来说很常见。

这些问题通常是在使用 OpenShift Container Platform Web 控制台安装 Operator 时，默认行为会将支持 **All namespaces** 安装模式的 Operator 安装到默认的 **openshift-operators** 全局命名空间中。

作为集群管理员，您可以使用以下工作流手动绕过此默认行为：

1. 为 Operator 的安装创建一个命名空间。
2. 创建自定义 *全局 Operator 组*，这是监视所有命名空间的 Operator 组。通过将此 Operator 组与您刚才创建的命名空间关联，从而使安装命名空间成为全局命名空间，从而使 Operator 在所有命名空间中都可用。
3. 在安装命名空间中安装所需的 Operator。

如果 Operator 具有依赖项，依赖项会在预先创建的命名空间中自动安装。因此，它对依赖项 Operator 有效，使其具有相同的更新策略和共享安装计划。具体步骤，请参阅“在自定义命名空间中安装全局 Operator"。

其他资源

- 在自定义命名空间中安装全局 Operator
- 多租户集群中的 Operator

2.4.7. Operator 条件

本指南概述了 Operator Lifecycle Manager（OLM）如何使用 Operator 条件。

2.4.7.1. 关于 Operator 条件

作为管理 Operator 生命周期的角色的一部分，Operator Lifecycle Manager（OLM）从定义 Operator 的 Kubernetes 资源状态中推断 Operator 状态。虽然此方法提供了一定程度的保证来确定 Operator 处于给定状态，但在有些情况下，Operator 可能需要直接向 OLM 提供信息，而这些信息不能被推断出来。这些信息可以被 OLM 使用来更好地管理 Operator 的生命周期。

OLM 提供了一个名为 **OperatorCondition** 的自定义资源定义（CRD），它允许 Operator 与 OLM 相互通信条件信息。当在一个 **OperatorCondition** 资源的 **Spec.Conditions** 数组中存在时，则代表存在一组会影响 OLM 管理 Operator 的支持条件。

注意

默认情况下，**OperatorCondition** 对象中没有 **Spec.Conditions** 数组，直到由用户添加或使用自定义 Operator 逻辑的结果为止。

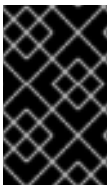
2.4.7.2. 支持的条件

Operator Lifecycle Manager（OLM）支持以下 Operator 条件。

2.4.7.2.1. Upgradeable（可升级）条件

Upgradeable Operator 条件可防止现有集群服务版本（CSV）被 CSV 的新版本替换。这一条件在以下情况下很有用：

- Operator 即将启动关键进程，不应在进程完成前升级。
- Operator 正在执行一个自定义资源（CR）迁移，这个迁移必须在 Operator 准备进行升级前完成。



重要

将 **Upgradeable** Operator 条件设置为 **False** 值不会避免 pod 中断。如果需要确保 pod 没有中断，请参阅"使用 pod 中断预算来指定必须在线的 pod 数量，以及 "Additional resources" 部分的 "Graceful termination"。

Upgradeable Operator 条件

```
apiVersion: operators.coreos.com/v1
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
```

```
conditions:
- type: Upgradeable 1
  status: "False" 2
  reason: "migration"
  message: "The Operator is performing a migration."
  lastTransitionTime: "2020-08-24T23:15:55Z"
```

1 条件的名称。

2 **False** 值表示 Operator 未准备好升级。OLM 可防止替换 Operator 现有 CSV 的 CSV 离开 **Pending** 状态。**False** 值不会阻止集群升级。

2.4.7.3. 其他资源

- 管理 Operator 条件
- 使用 pod 中断预算指定必须在线的 pod 数量
- 恰当终止

2.4.8. Operator Lifecycle Manager 指标数据

2.4.8.1. 公开的指标

Operator Lifecycle Manager（OLM）会公开某些 OLM 特定资源，供基于 Prometheus 的 OpenShift Container Platform 集群监控堆栈使用。

表 2.8. OLM 公开的指标

名称	描述
catalog_source_count	目录源数量。
catalogsource_ready	目录源的状态。值 **1** 表示目录源处于 **READY** 状态。**0** 表示目录源没有处于 **READY** 状态。
csv_abnormal	在协调集群服务版本（CSV）时，每当 CSV 版本处于 **Succeeded** 以外的任何状态时（如没有安装它时）就会存在。包括 **name**、**namespace**、**phase**、**reason** 和 **version** 标签。当存在此指标数据时会创建一个 Prometheus 警报。
csv_count	成功注册的 CSV 数量。
csv_succeeded	在协调 CSV 时，代表 CSV 版本处于 **Succeeded** 状态（值为 **1**）或没有处于这个状态（值为 **0**）。包含 **name**、**namespace** 和 **version** 标签。
csv_upgrade_count	CSV 升级的 Monotonic 计数。

名称	描述
install_plan_count	安装计划的数量。
installplan_warnings_total	由资源生成的警告数量（如已弃用资源）包含在安装计划中。
olm_resolution_duration_seconds	依赖项解析尝试的持续时间。
subscription_count	订阅数。
subscription_sync_total	订阅同步的单调计数。包括 **channel**、**installed** CSV 和订阅 **name** 标签。

2.4.9. Operator Lifecycle Manager 中的 Webhook 管理

Webhook 允许 Operator 作者在资源被保存到对象存储并由 Operator 控制器处理之前，拦截、修改、接受或拒绝资源。当 webhook 与 Operator 一同提供时，Operator Lifecycle Manager（OLM）可以管理这些 webhook 的生命周期。

2.4.9.1. 其他资源

- Webhook 准入插件类型
- Kubernetes 文档：
  - 验证准入 webhook
  - 变异准入 webhook
  - webhook 转换

2.5. 了解 OPERATORHUB

2.5.1. 关于 OperatorHub

OperatorHub 是集群管理员用来发现和安装 Operator 的 OpenShift Container Platform 中的 Web 控制台界面。只需单击一次，即可从其非集群源拉取 Operator，并将其安装和订阅至集群中，为工程团队使用 Operator Lifecycle Manager (OLM) 在部署环境中自助管理产品做好准备。

集群管理员可从划分为以下类别的目录进行选择：

类别	描述
红帽 Operator	已由红帽打包并提供的红帽产品。受红帽支持。

类别	描述
经认证的 Operator	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。
社区 Operator	由 redhat-openshift-ecosystem/community-operators-prod/operators GitHub 存储库中相关代表维护的可选可见软件。无官方支持。
自定义 Operator	您自行添加至集群的 Operator。如果您尚未添加任何自定义 Operator，则您的 OperatorHub 上 Web 控制台中便不会出现**自定义**类别。

OperatorHub 上的操作员被打包在 OLM 上运行。这包括一个称为集群服务版本（CSV）的 YAML 文件，其中包含安装和安全运行 Operator 所需的所有 CRD、RBAC 规则、Deployment 和容器镜像。它还包含用户可见的信息，如功能描述和支持的 Kubernetes 版本。

2.5.2. OperatorHub 架构

OperatorHub UI 组件默认由 **openshift-marketplace** 命名空间中 OpenShift Container Platform 上的 Marketplace Operator 驱动。

2.5.2.1. OperatorHub 自定义资源

Marketplace Operator 管理名为 **cluster** 的 **OperatorHub** 自定义资源（CR），用于管理 OperatorHub 提供的默认 **CatalogSource** 对象。您可以修改此资源以启用或禁用默认目录，这在受限网络环境中配置 OpenShift Container Platform 时非常有用。

OperatorHub 自定义资源示例

```
apiVersion: config.openshift.io/v1
kind: OperatorHub
metadata:
  name: cluster
spec:
  disableAllDefaultSources: true (1)
  sources: [ (2)
    {
      name: "community-operators",
      disabled: false
    }
  ]
```

1. **disableAllDefaultSources** 是一个覆盖，用于控制在 OpenShift Container Platform 安装期间默认配置的所有默认目录的可用性。
2. 通过更改每个源的 **disabled** 参数值来分别禁用默认目录。

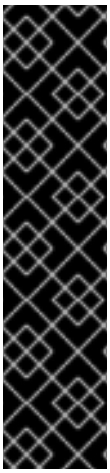
2.5.3. 其他资源

- 目录源
- OLM 中的 Operator 安装和升级工作流

- Red Hat Partner Connect

2.6. 红帽提供的 OPERATOR 目录

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



重要

从 OpenShift Container Platform 4.11 开始，默认的红帽提供的 Operator 目录以基于文件的目录格式发布。通过以过时的 SQLite 数据库格式发布的 4.10，用于 OpenShift Container Platform 4.6 的默认红帽提供的 Operator 目录。

与 SQLite 数据库格式相关的 **opm** 子命令、标志和功能已被弃用，并将在以后的版本中删除。功能仍被支持，且必须用于使用已弃用的 SQLite 数据库格式的目录。

许多 **opm** 子命令和标志都用于 SQLite 数据库格式，如 **opm index prune**，它们无法使用基于文件的目录格式。有关使用基于文件的目录的更多信息，请参阅管理自定义目录、Operator Framework 打包格式，以及使用 oc-mirror 插件为断开连接的安装 mirror 镜像。

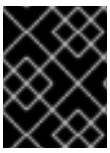
2.6.1. 关于 Operator 目录

Operator 目录是 Operator Lifecycle Manager（OLM）可以查询的元数据存储库，以在集群中发现和安装 Operator 及其依赖项。OLM 始终从目录的最新版本安装 Operator。

基于 Operator Bundle Format 的索引镜像是目录的容器化快照。这是一个不可变的工件，包含指向一组 Operator 清单内容的指针数据库。目录可以引用索引镜像来获取集群中 OLM 的内容。

随着目录的更新，Operator 的最新版本会发生变化，旧版本可能会被删除或修改。另外，当 OLM 在受限网络环境中的 OpenShift Container Platform 集群上运行时，它无法直接从互联网访问目录来拉取最新内容。

作为集群管理员，您可以根据红帽提供的目录或从头创建自己的自定义索引镜像，该镜像可用于提供集群中的目录内容。创建和更新您自己的索引镜像提供了一种方法来自定义集群上可用的一组 Operator，同时避免了上面提到的受限网络环境中的问题。



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。

注意

OpenShift Container Platform 4.8 及之后的版本中删除了对 Operator 的*传统软件包清单格式*的支持，包括使用传统格式的自定义目录。

在创建自定义目录镜像时，在以前的 OpenShift Container Platform 4 版本中需要使用 **oc adm catalog build** 命令，这个命令已在多个版本中被弃用，现在已被删除。从 OpenShift Container Platform 4.6 开始，红帽提供的索引镜像可用后,目录构建器必须使用 **opm index** 命令来管理索引镜像。

其他资源

- 管理自定义目录

- 打包格式
- 在断开连接的环境中使用 Operator Lifecycle Manager。

2.6.2. 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
```

2.7. 多租户集群中的 OPERATOR

Operator Lifecycle Manager (OLM) 的默认行为旨在简化 Operator 的安装过程。但是，此行为可能会缺少灵活性，特别是在多租户集群中。为了让 OpenShift Container Platform 集群上的多个租户使用 Operator，OLM 的默认行为要求管理员以 **All namespaces** 模式安装 Operator，这可能被视为违反最小特权的原则。

请考虑以下场景，以确定哪个 Operator 安装工作流最适合您的环境和要求。

其他资源

- 常见术语：多租户（Multitenant）

- 多租户 Operator 管理的限制

2.7.1. 默认 Operator 安装模式和行为

当以管理员身份使用 Web 控制台安装 Operator 时，通常会根据 Operator 的功能，对安装模式有两个选择：

单个命名空间

在所选命名空间中安装 Operator，并发出 Operator 请求在该命名空间中提供的所有权限。

所有命名空间

将 Operator 安装至默认**openshift-operators** 命名空间，以便供集群中的所有命名空间监视和使用。进行所有命名空间中 Operator 请求的所有权限。在某些情况下，Operator 作者可以定义元数据，为用户授予该 Operator 建议的命名空间的第二个选项。

此选择还意味着受影响命名空间中的用户可以访问 Operator API，该 API 可以利用他们拥有的自定义资源 (CR)，具体取决于命名空间中的角色：

- **namespace-admin** 和 **namespace-edit** 角色可以对 Operator API 进行读/写，这意味着他们可以使用它们。
- **namespace-view** 角色可以读取该 Operator 的 CR 对象。

对于 **Single namespace** 模式，因为 Operator 本身安装在所选命名空间中，所以其 pod 和服务帐户也位于那里。对于 **All namespaces** 模式，Operator 的权限会自动提升到集群角色，这意味着 Operator 在所有命名空间中都有这些权限。

其他资源

- 在集群中添加 Operator
- 安装模式类型

2.7.2. 多租户集群的建议解决方案

虽然 **Multinamespace** 安装模式存在，但只有少数 Operator 支持它。作为标准 **All namespaces** 和 **Single namespace** 安装模式之间的中间解决方案，您可以使用以下工作流安装同一 Operator 的多个实例，每个租户一个实例：

1. 为租户 Operator 创建命名空间，与租户的命名空间分开。
2. 为租户 Operator 创建 Operator 组，范围仅限于租户的命名空间。
3. 在租户 Operator 命名空间中安装 Operator。

因此，Operator 驻留在租户 Operator 命名空间中，并监视租户命名空间，但 Operator 的 pod 及其服务帐户都无法被租户可见或可用。

此解决方案以资源使用量成本提供更好的租户分离，以及确保满足约束的额外编配功能。如需详细步骤，请参阅"为多租户集群准备 Operator 的多个实例"。

限制和注意事项

只有在满足以下限制时，这个解决方案才可以正常工作：

- 同一 Operator 的所有实例都必须是相同的版本。

- Operator 无法依赖于其他 Operator。
- Operator 无法提供 CRD 转换 Webhook。

重要

您不能在同一集群中使用相同的 Operator 的不同版本。最后，当 Operator 的安装满足以下条件时，会阻断另一个 Operator 实例：

- 实例不是 Operator 的最新版本。
- 该实例提供了一个较老的 CRD 修订，它缺少新修订版本已在集群中使用的信息或版本。

警告

作为管理员，在允许非集群管理员自行安装 Operator 时请小心，如"允许非集群管理员安装 Operator"中所述。这些租户应只能访问已知没有依赖项的 Operator 策展目录。这些租户也必须强制使用 Operator 的同一版本行，以确保 CRD 不会改变。这需要使用命名空间范围的目录，并可能会禁用全局默认目录。

其他资源

- 为多租户集群准备多个 Operator 实例
- 允许非集群管理员安装 Operator
- 禁用默认的 OperatorHub 目录源

2.7.3. Operator 共处和 Operator 组

Operator Lifecycle Manager (OLM) 处理在同一命名空间中安装的 OLM 管理的 Operator，这意味着其 **Subscription** 资源与相关 Operator 位于同一个命名空间中。即使它们实际不相关，OLM 会在更新其中任何一个时考虑其状态，如它们的版本和更新策略。

如需有关 Operator 共处和使用 Operator 组的更多信息，请参阅 Operator Lifecycle Manager (OLM)→ Multitenancy 和 Operator colocation。

2.8. CRD

2.8.1. 使用自定义资源定义来扩展 Kubernetes API

Operator 使用 Kubernetes 扩展机制（自定义资源定义（CRD）），以便使由 Operator 管理的自定义类似于内置的原生 Kubernetes 对象。本指南介绍了集群管理员如何通过创建和管理 CRD 来扩展其 OpenShift Container Platform 集群。

2.8.1.1. 自定义资源定义

在 Kubernetes API 中，*resource（资源）*是存储某一类 API 对象集的端点。例如，内置 **Pod** 资源包含一组 **Pod** 对象。

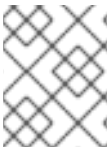
自定义资源定义（CRD）对象在集群中定义一个新的、唯一的对象类型，称为 *kind*，并允许 Kubernetes API 服务器处理其整个生命周期。

自定义资源 (CR) 对象由集群管理员通过集群中已添加的 CRD 创建，并支持所有集群用户在项目中增加新的资源类型。

当集群管理员增加新 CRD 至集群中时，Kubernetes API 服务器的回应方式是新建一个可由整个集群或单个项目（命名空间）访问的 RESTful 资源路径，并开始服务于指定的 CR。

集群管理员如果要向其他用户授予 CRD 访问权限，可使用集群角色聚合来向用户授予 **admin**、**edit** 或 **view** 默认集群角色访问权限。集群角色聚合支持将自定义策略规则插入到这些集群角色中。此行为将新资源整合到集群的 RBAC 策略中，就像内置资源一样。

Operator 会通过将 CRD 与任何所需 RBAC 策略和其他软件特定逻辑打包到一起来利用 CRD。集群管理员也可以手动将 CRD 添加到 Operator 生命周期之外的集群中，供所有用户使用。



注意

虽然只有集群管理员可创建 CRD，但具有 CRD 读写权限的开发人员也可通过现有 CRD 来创建 CR。

2.8.1.2. 创建自定义资源定义

要创建自定义资源 (CR) 对象，集群管理员首先必须创建一个自定义资源定义 (CRD)。

先决条件

- 以 **cluster-admin** 用户身份访问 OpenShift Container Platform 集群。

流程

要创建 CRD：

1. 先创建一个包含以下字段类型的 YAML 文件：

CRD 的 YAML 文件示例

```
apiVersion: apiextensions.k8s.io/v1 1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com 2
spec:
  group: stable.example.com 3
  versions:
    - name: v1 4
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
```

```
            type: object
            properties:
              cronSpec:
                type: string
              image:
                type: string
              replicas:
                type: integer
  scope: Namespaced (5)
  names:
    plural: crontabs (6)
    singular: crontab (7)
    kind: CronTab (8)
    shortNames:
    - ct (9)
```

1. 使用 **apiextensions.k8s.io/v1** API。
2. 为定义指定名称。这必须采用 **<plural-name>.<group>** 格式，并使用来自 **group** 和 **plural** 字段的值。
3. 为 API 指定组名。API 组是一个逻辑上相关的对象集。例如，**Job** 或 **ScheduledJob** 等所有批处理对象，均可添加至批处理 API 组（如 **batch.api.example.com**）中。最好使用您机构的完全限定域名（FQDN）。
4. 指定 URL 中要用的版本名称。每个 API 组都可能存在于多个版本中，例如 **v1alpha**、**v1beta**、**v1**。
5. 指定自定义对象可用于某一个项目 (**Namespaced**) 还是集群中的所有项目 (**Cluster**)。
6. 指定 URL 中要用的复数名称。**plural** 字段与 API URL 网址中的资源相同。
7. 指定将在 CLI 上用作别名并用于显示的单数名称。
8. 指定可创建的对象类型。类型可以采用 CamelCase。
9. 指定与 CLI 中的资源相匹配的较短字符串。

注意

默认情况下，CRD 的覆盖范围为整个集群，适用于所有项目。

2. 创建 CRD 对象：

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

在以下位置新建一个 RESTful API 端点：

```
/apis/<spec:group>/<spec:version>/<scope>/*/<names-plural>/...
```

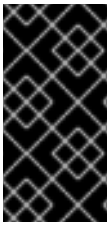
例如，以下端点便是通过示例文件创建的：

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

- 现在，您即可使用该端点 URL 来创建和管理 CR。对象类型基于您创建的 CRD 对象的 **spec.kind** 字段。

2.8.1.3. 为自定义资源定义创建集群角色

集群管理员可向现有集群范围的自定义资源定义 (CRD) 授予权限。如果使用 **admin**、**edit** 和 **view** 默认集群角色，请利用集群角色聚合来制定规则。



重要

您必须为每个角色明确分配权限。权限更多的角色不会继承权限较少角色的规则。如果要为某个角色分配规则，还必须将该操作动词分配给具有更多权限的角色。例如，如果要向 view 角色授予 **get crontabs** 的权限，也必须向 **edit** 和 **admin** 角色授予该权限。**admin** 或 **edit** 角色通常会分配给通过项目模板创建项目的用户。

先决条件

- 创建 CRD。

流程

1. 为 CRD 创建集群角色定义文件。集群角色定义是一个 YAML 文件，其中包含适用于各个集群角色的规则。OpenShift Container Platform 控制器会将您指定的规则添加到默认集群角色中。

集群角色定义的 YAML 文件示例

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1 (1)
metadata:
  name: aggregate-cron-tabs-admin-edit (2)
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin: "true" (3)
    rbac.authorization.k8s.io/aggregate-to-edit: "true" (4)
rules:
- apiGroups: ["stable.example.com"] (5)
  resources: ["crontabs"] (6)
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete", "deletecollection"] (7)
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: aggregate-cron-tabs-view (8)
  labels:
    # Add these permissions to the "view" default role.
    rbac.authorization.k8s.io/aggregate-to-view: "true" (9)
    rbac.authorization.k8s.io/aggregate-to-cluster-reader: "true" (10)
rules:
- apiGroups: ["stable.example.com"] (11)
  resources: ["crontabs"] (12)
  verbs: ["get", "list", "watch"] (13)
```

1. 使用 **rbac.authorization.k8s.io/v1** API。
2. 8. 为定义指定名称。
3. 指定该标签向 admin 默认角色授予权限。
4. 指定该标签向 edit 默认角色授予权限。
5. 11. 指定 CRD 的组名。
6. 12. 指定适用于这些规则的 CRD 的复数名称。
7. 13. 指定代表角色所获得的权限的操作动词。例如，对 **admin** 和 **edit** 角色应用读写权限，对 **view** 角色应用只读权限。
9. 指定该标签向 **view** 默认角色授予权限。
10. 指定该标签向 **cluster-reader** 默认角色授予权限。

2. 创建集群角色：

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

2.8.1.4. 通过文件创建自定义资源

将自定义资源定义 (CRD) 添加至集群后，可使用 CLI 按照自定义资源 (CR) 规范通过文件创建 CR。

先决条件

- 集群管理员已将 CRD 添加至集群中。

流程

1. 为 CR 创建 YAML 文件。在下面的定义示例中，**cronSpec** 和 **image** 自定义字段在 **Kind: CronTab** 的 CR 中设定。**Kind** 来自 CRD 对象的 **spec.kind** 字段：

   CR 的 YAML 文件示例

```
apiVersion: "stable.example.com/v1" 1
kind: CronTab 2
metadata:
  name: my-new-cron-object 3
  finalizers: 4
  - finalizer.stable.example.com
spec: 5
  cronSpec: "* * * * /5"
  image: my-awesome-cron-image
```

1. 指定 CRD 中的组名称和 API 版本（名称/版本）。
2. 指定 CRD 中的类型。
3. 指定对象的名称。

4 指定对象的结束程序（如有）。结束程序可让控制器实现在删除对象之前必须完成的条件。

5 指定特定于对象类型的条件。

2. 创建完文件后，再创建对象：

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

2.8.1.5. 检查自定义资源

您可使用 CLI 检查集群中存在的自定义资源 (CR) 对象。

先决条件

- 您有权访问的命名空间中已存在 CR 对象。

流程

1. 要获取特定类型的 CR 的信息，请运行：

```
$ oc get <kind>
```

例如：

```
$ oc get crontab
```

输出示例

```
NAME                 KIND
my-new-cron-object   CronTab.v1.stable.example.com
```

资源名称不区分大小写，您既可使用 CRD 中定义的单数或复数形式，也可使用简称。例如：

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. 还可查看 CR 的原始 YAML 数据：

```
$ oc get <kind> -o yaml
```

例如：

```
$ oc get ct -o yaml
```

输出示例

```
apiVersion: v1
```

```
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-new-cron-object
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

1 2 显示用于创建对象的 YAML 的自定义数据。

2.8.2. 管理自定义资源定义中的资源

本指南向开发人员介绍了如何管理来自自定义资源定义 (CRD) 的自定义资源 (CR)。

2.8.2.1. 自定义资源定义

在 Kubernetes API 中，*resource（资源）*是存储某一类 API 对象集的端点。例如，内置 **Pod** 资源包含一组 **Pod** 对象。

自定义资源定义（CRD）对象在集群中定义一个新的、唯一的对象类型，称为 *kind*，并允许 Kubernetes API 服务器处理其整个生命周期。

自定义资源 (CR) 对象由集群管理员通过集群中已添加的 CRD 创建，并支持所有集群用户在项目中增加新的资源类型。

Operator 会通过将 CRD 与任何所需 RBAC 策略和其他软件特定逻辑打包到一起来利用 CRD。集群管理员也可以手动将 CRD 添加到 Operator 生命周期之外的集群中，供所有用户使用。

注意

虽然只有集群管理员可创建 CRD，但具有 CRD 读写权限的开发人员也可通过现有 CRD 来创建 CR。

2.8.2.2. 通过文件创建自定义资源

将自定义资源定义 (CRD) 添加至集群后，可使用 CLI 按照自定义资源 (CR) 规范通过文件创建 CR。

先决条件

- 集群管理员已将 CRD 添加至集群中。

流程

1. 为 CR 创建 YAML 文件。在下面的定义示例中，**cronSpec** 和 **image** 自定义字段在 **Kind: CronTab** 的 CR 中设定。**Kind** 来自 CRD 对象的 **spec.kind** 字段：

   CR 的 YAML 文件示例

   ```
   apiVersion: "stable.example.com/v1" 1
   kind: CronTab 2
   metadata:
     name: my-new-cron-object 3
     finalizers: 4
     - finalizer.stable.example.com
   spec: 5
     cronSpec: "* * * * /5"
     image: my-awesome-cron-image
   ```

   1 指定 CRD 中的组名称和 API 版本（名称/版本）。

   2 指定 CRD 中的类型。

   3 指定对象的名称。

   4 指定对象的结束程序（如有）。结束程序可让控制器实现在删除对象之前必须完成的条件。

   5 指定特定于对象类型的条件。

2. 创建完文件后，再创建对象：

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

2.8.2.3. 检查自定义资源

您可使用 CLI 检查集群中存在的自定义资源 (CR) 对象。

先决条件

- 您有权访问的命名空间中已存在 CR 对象。

流程

1. 要获取特定类型的 CR 的信息，请运行：

   ```
   $ oc get <kind>
   ```

   例如：

   ```
   $ oc get crontab
   ```

   输出示例

   ```
   NAME                 KIND
   my-new-cron-object   CronTab.v1.stable.example.com
   ```

资源名称不区分大小写，您既可使用 CRD 中定义的单数或复数形式，也可使用简称。例如：

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. 还可查看 CR 的原始 YAML 数据：

```
$ oc get <kind> -o yaml
```

例如：

```
$ oc get ct -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-new-cron-object
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

1 2 显示用于创建对象的 YAML 的自定义数据。

第 3 章 用户任务

3.1. 从已安装的 OPERATOR 创建应用程序

本指南向开发人员介绍了如何使用 OpenShift Container Platform Web 控制台从已安装的 Operator 创建应用程序。

3.1.1. 使用 Operator 创建 etcd 集群

本流程介绍了如何通过由 Operator Lifecycle Manager (OLM) 管理的 etcd Operator 来新建一个 etcd 集群。

先决条件

- 访问 OpenShift Container Platform 4.19 集群。
- 管理员已在集群范围内安装了 etcd Operator。

流程

1. 针对此流程在 OpenShift Container Platform Web 控制台中新建一个项目。这个示例使用名为 **my-etcd** 的项目。
2. 导航至 **Operators → Installed Operators** 页面。由集群管理员安装到集群且可供使用的 Operator 将以集群服务版本（CSV）列表形式显示在此处。CSV 用于启动和管理由 Operator 提供的软件。

   提示

   使用以下命令从 CLI 获得该列表：

   ```
   $ oc get csv
   ```

3. 在 **Installed Operators** 页面中，点 etcd Operator 查看更多详情和可用操作。
   正如 **Provided API** 下所示，该 Operator 提供了三类新资源，包括一种用于 **etcd Cluster** 的资源（**EtcdCluster** 资源）。这些对象的工作方式与内置的原生 Kubernetes 对象（如 **Deployment** 或 **ReplicaSet**）相似，但包含特定于管理 etcd 的逻辑。
4. 新建 etcd 集群：
   a. 在 **etcd Cluster** API 框中，点 **Create instance**。
   b. 在下一页中，您可对 **EtcdCluster** 对象的最小起始模板进行任何修改，比如集群大小。现在，点击 **Create** 即可完成。点击后即可触发 Operator 启动 pod、服务和新 etcd 集群的其他组件。
5. 点 **example** etcd 集群，然后点 **Resources** 选项卡，您可以看到项目现在包含很多由 Operator 自动创建和配置的资源。
   验证已创建了支持您从项目中的其他 pod 访问数据库的 Kubernetes 服务。
6. 给定项目中具有 **edit** 角色的所有用户均可创建、管理和删除应用程序实例（本例中为 etcd 集群），这些实例由已在项目中创建的 Operator 以自助方式管理，就像云服务一样。如果要赋予其他用户这一权利，项目管理员可使用以下命令添加角色：

```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

现在您有了一个 etcd 集群，当 pod 运行不畅，或在集群中的节点之间迁移时，该集群将对故障做出反应并重新平衡数据。最重要的是，具有适当访问权限的集群管理员或开发人员现在可轻松将该数据库用于其应用程序。

3.2. 在命名空间中安装 OPERATOR

如果集群管理员将 Operator 安装权限委托给您的帐户，您可以以自助服务的方式将 Operator 安装并订阅到命名空间中。

3.2.1. 先决条件

- 集群管理员必须在 OpenShift Container Platform 用户帐户中添加某些权限，以便允许将自助服务 Operator 安装到命名空间。详情请参阅允许非集群管理员安装 Operator。

3.2.2. 关于使用 OperatorHub 安装 Operator

OperatorHub 是一个发现 Operator 的用户界面，它与 Operator Lifecycle Manager（OLM）一起工作，后者在集群中安装和管理 Operator。

作为具有适当权限的用户，您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装来自 OperatorHub 的 Operator。

安装过程中，您必须为 Operator 确定以下初始设置：

安装模式

选择要在其中安装 Operator 的特定命名空间。

更新频道

如果某个 Operator 可通过多个频道获得，则可任选您想要订阅的频道。例如，要通过 **stable** 频道部署（如果可用），则从列表中选择这个选项。

批准策略

您可以选择自动或者手动更新。
如果选择自动更新某个已安装的 Operator，则当所选频道中有该 Operator 的新版本时，Operator Lifecycle Manager（OLM）将自动升级 Operator 的运行实例，而无需人为干预。

如果选择手动更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。

- 了解 OperatorHub

3.2.3. 使用 Web 控制台，从 OperatorHub 安装

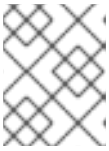
您可以使用 OpenShift Container Platform Web 控制台从 OperatorHub 安装并订阅 Operator。

先决条件

- 使用具有 Operator 安装权限的账户访问 OpenShift Container Platform 集群。

流程

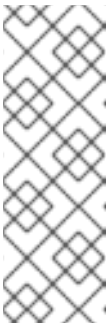
1. 在 Web 控制台中导航至 **Operators → OperatorHub** 页面。
2. 找到您需要的 Operator（滚动页面会在 **Filter by keyword** 框中输入查找关键字）。例如，输入 **advanced** 来查找 Advanced Cluster Management for Kubernetes Operator。
   您还可以根据**基础架构功能**过滤选项。例如，如果您希望 Operator 在断开连接的环境中工作，请选择 **Disconnected**。
3. 选择要显示更多信息的 Operator。



注意

选择 Community Operator 会警告红帽没有认证社区 Operator；您必须确认该警告方可继续。

4. 阅读 Operator 信息并点 **Install**。
5. 在 **Install Operator** 页面中，配置 Operator 安装：
   a. 如果要安装 Operator 的特定版本，请从列表中选择 **Update channel** 和 **Version**。您可以在可能具有的任何频道中浏览 Operator 的不同版本，查看该频道和版本的元数据，然后选择您要安装的确切版本。

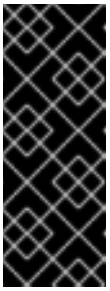


注意

版本选择默认为所选频道的最新版本。如果选择了频道的最新版本，则默认启用 **Automatic** 批准策略。如果没有为所选频道安装最新版本，则需要**手动**批准。

使用**手动**批准安装 Operator 会导致命名空间中安装的所有 Operator 并使用 **Manual** 批准策略和所有 Operator 一起更新。如果要独立更新 Operator，将 Operator 安装到单独的命名空间中。

   b. 选择要在其中安装 Operator 的特定单一命名空间。该 Operator 仅限在该单一命名空间中监视和使用。
   c. 对于启用了令牌身份验证的云供应商上的集群：
      - 如果集群在 web 控制台中使用 AWS 安全令牌服务(**STS Mode**)，在角色 ARN 字段中输入服务帐户的 AWS IAM 角色的 Amazon Resource Name (**ARN**)。要创建角色的 ARN，请按照 准备 AWS 帐户 中所述的步骤进行操作。
      - 如果集群使用 Microsoft Entra Workload ID (Web 控制台中的**Workload Identity / Federated Identity Mode**)，请在适当的项中添加客户端 ID、租户 ID 和订阅 ID。
      - 如果集群使用 Google Cloud Platform Workload Identity (**GCP Workload Identity / Federated Identity Mode**)，请在适当的字段中添加项目号、池 ID、供应商 ID 和服务帐户电子邮件。
   d. 对于 **更新批准**，请选择 **Automatic** 或 **Manual** 批准策略。



重要

如果 Web 控制台显示集群使用 AWS STS、Microsoft Entra Workload ID 或 GCP Workload Identity，您必须将 **Update approval** 设置为 **Manual**。

不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

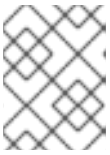
6. 点 **Install** 使 Operator 可供 OpenShift Container Platform 集群上的所选命名空间使用：

   a. 如果选择了**手动**批准策略，订阅的升级状态将保持在 **Upgrading** 状态，直至您审核并批准安装计划。
   在 **Install Plan** 页面批准后，订阅的升级状态将变为 **Up to date**。

   b. 如果选择了 **Automatic** 批准策略，升级状态会在不用人工参与的情况下变为 **Up to date**。

验证

- 在订阅的升级状态变为 **Up to date** 后，选择 **Operators → Installed Operators** 来验证已安装 Operator 的集群服务版本(CSV)是否最终出现了。**状态** 最终会在相关命名空间中变为 **Succeeded**。



注意

对于 **All namespaces...** 安装模式，状态会在 **openshift-operators** 命名空间中解析为 **Succeeded**，但如果检查其他命名空间，则状态为 **Copied**。

如果没有：

  - 检查 **openshift-operators** 项目（如果选择了 **A specific namespace...** 安装模式）中的 openshift-operators 项目中的 pod 的日志，这会在 **Workloads → Pods** 页面中报告问题以便进一步排除故障。

- 安装 Operator 时，元数据会指示安装了哪个频道和版本。

注意

此目录上下文中仍可使用 **Channel** 和 **Version** 下拉菜单查看其他版本元数据。

3.2.4. 使用 CLI 从 OperatorHub 安装

您可以使用 CLI 从 OperatorHub 安装 Operator，而不必使用 OpenShift Container Platform Web 控制台。使用 **oc** 命令来创建或更新一个**订阅**对象。

对于 **SingleNamespace** 安装模式，还必须确保相关命名空间中存在适当的 Operator 组。一个 Operator 组（由 **OperatorGroup** 对象定义），在其中选择目标命名空间，在其中为与 Operator 组相同的命名空间中的所有 Operator 生成所需的 RBAC 访问权限。

提示

在大多数情况下，使用 Web 控制台是首选的方法，因为它会在后台自动执行任务，比如在选择 **SingleNamespace** 模式时自动处理 **OperatorGroup** 和 **Subscription** 对象的创建。

先决条件

- 使用具有 Operator 安装权限的账户访问 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 查看 OperatorHub 中集群可用的 Operator 列表：

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

例 3.1. 输出示例

```
NAME                              CATALOG               AGE
3scale-operator                   Red Hat Operators     91m
advanced-cluster-management       Red Hat Operators     91m
amq7-cert-manager                 Red Hat Operators     91m
# ...
couchbase-enterprise-certified    Certified Operators   91m
crunchy-postgres-operator         Certified Operators   91m
mongodb-enterprise                Certified Operators   91m
# ...
etcd                              Community Operators   91m
jaeger                            Community Operators   91m
kubefed                           Community Operators   91m
# ...
```

记录下所需 Operator 的目录。

2. 检查所需 Operator，以验证其支持的安装模式和可用频道：

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

例 3.2. 输出示例

```
# ...
Kind:         PackageManifest
# ...
      Install Modes: (1)
        Supported:  true
        Type:       OwnNamespace
        Supported:  true
        Type:       SingleNamespace
        Supported:  false
        Type:       MultiNamespace
        Supported:  true
        Type:       AllNamespaces
# ...
    Entries:
      Name:       example-operator.v3.7.11
      Version:    3.7.11
      Name:       example-operator.v3.7.10
```

```
    Version:    3.7.10
    Name:       stable-3.7 2
# ...
  Entries:
    Name:       example-operator.v3.8.5
    Version:    3.8.5
    Name:       example-operator.v3.8.4
    Version:    3.8.4
  Name:         stable-3.8 3
Default Channel: stable-3.8 4
```

1 指明支持哪些安装模式。

2 3 频道名称示例。

4 在没有指定频道时默认选择的频道。

提示

您可以运行以下命令来以 YAML 格式打印 Operator 的版本和频道信息：

```
$ oc get packagemanifests <operator_name> -n <catalog_namespace> -o yaml
```

3. 如果在命名空间中安装多个目录，请运行以下命令从特定目录中查找 Operator 的可用版本和频道：

```
$ oc get packagemanifest \
  --selector=catalog=<catalogsource_name> \
  --field-selector metadata.name=<operator_name> \
  -n <catalog_namespace> -o yaml
```

重要

如果没有指定 Operator 的目录，运行 **oc get packagemanifest** 和 **oc describe packagemanifest** 命令可能会在满足以下条件时从一个意料外的目录中返回一个软件包：

- 在同一命名空间中安装多个目录。
- 目录包含具有相同名称的相同 Operator 或 Operator。

4. 如果要安装的 Operator 支持 **AllNamespaces** 安装模式，而您选择使用这个模式，请跳过这一步，因为 **openshift-operators** 命名空间默认有一个适当的 Operator 组，称为 **global-operators**。
如果要安装的 Operator 支持 **SingleNamespace** 安装模式，而选择使用此模式，您必须确保相关命名空间中存在适当的 Operator 组。如果不存在，您可以按照以下步骤创建一个：

重要

每个命名空间只能有一个 Operator 组。如需更多信息，请参阅 "Operator 组"。

a. 为 **SingleNamespace** 安装模式创建一个 **OperatorGroup** 对象 YAML 文件，如 **operatorgroup.yaml** ：

SingleNamespace 安装模式的 OperatorGroup 对象示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace> 1
spec:
  targetNamespaces:
  - <namespace> 2
```

1 2 对于 **SingleNamespace** 安装模式，对于 **metadata.namespace** 和 **spec.targetNamespaces** 字段使用相同的 **<namespace>** 值。

b. 创建 **OperatorGroup** 对象：

```
$ oc apply -f operatorgroup.yaml
```

5. 创建一个 **Subscription** 对象来订阅一个命名空间到 Operator：

a. 为 **Subscription** 对象创建一个 YAML 文件，如 **subscription.yaml** ：

注意

如果要订阅 Operator 的特定版本，请将 **startCSV** 字段设置为所需的版本，并将 **installPlanApproval** 字段设置为 **Manual**，以防止 Operator 在目录中有更新的版本时自动升级。详情请参阅以下 "Example **Subscription** object with a specific starting Operator version"。

例 3.3. Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: <namespace_per_install_mode> 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: <catalog_name> 4
  sourceNamespace: <catalog_source_namespace> 5
  config:
    env: 6
    - name: ARGS
      value: "-v=10"
    envFrom: 7
    - secretRef:
        name: license-secret
```

```
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1. 对于默认的 **AllNamespaces** 安装模式用法，请指定 **openshift-operators** 命名空间。另外，如果创建了自定义全局命名空间，您可以指定一个自定义全局命名空间。对于 **SingleNamespace** 安装模式的使用，请指定相关的单一命名空间。
2. 要订阅的频道的名称。
3. 要订阅的 Operator 的名称。
4. 提供 Operator 的目录源的名称。
5. 目录源的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub 目录源。
6. **env** 参数定义必须存在于由 OLM 创建的 pod 中所有容器中的环境变量列表。
7. **envFrom** 参数定义要在容器中填充环境变量的源列表。
8. **volumes** 参数定义了由 OLM 创建的、在 pod 上必须存在的卷列表。
9. **volumeMounts** 参数定义由 OLM 创建的 pod 中必须存在的卷挂载列表。如果 **volumeMount** 引用不存在的 **卷**，OLM 无法部署 Operator。
10. **tolerations** 参数为 OLM 创建的 pod 定义容限列表。
11. **resources** 参数为 OLM 创建的 pod 中所有容器定义资源限制。
12. **nodeSelector** 参数为 OLM 创建的 pod 定义 **NodeSelector**。

例 3.4. 带有特定启动 Operator 版本的 Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-operator
```

```
spec:
  channel: stable-3.7
  installPlanApproval: Manual (1)
  name: example-operator
  source: custom-operators
  sourceNamespace: openshift-marketplace
  startingCSV: example-operator.v3.7.10 (2)
```

1. 如果您指定的版本会被目录中的更新版本取代，则将批准策略设置为 **Manual**。此计划阻止自动升级到更新的版本，且需要在启动 CSV 可以完成安装前手动批准。
2. 设置 Operator CSV 的特定版本。

b. 对于启用了令牌身份验证的云供应商的集群，如 Amazon Web Services (AWS) 安全令牌服务 (STS)、Microsoft Entra Workload ID 或 Google Cloud Platform Workload Identity，按照以下步骤配置您的 **Subscription** 对象：

i. 确保 **Subscription** 对象被设置为手动更新批准：

例 3.5. 带有手动更新批准的 Subscription 对象示例

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual (1)
```

1. 不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

ii. 在 **Subscription** 对象的 **config** 部分包括相关的云供应商相关字段：
如果集群处于 AWS STS 模式，请包含以下字段：

例 3.6. 使用 AWS STS 变量的 Subscription 对象示例

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" (1)
```

1. 包含角色 ARN 详情。

如果集群处于 Workload ID 模式，请包含以下字段：

例 3.7. 带有 Workload ID 变量的 Subscription 对象示例

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" 1
   - name: TENANTID
     value: "<tenant_id>" 2
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" 3
```

1 包含客户端 ID。

2 包含租户 ID。

3 包括订阅 ID。

如果集群处于 GCP Workload Identity 模式，请包括以下字段：

例 3.8. 带有 GCP Workload Identity 变量的Subscription 对象示例

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: AUDIENCE
     value: "<audience_url>" 1
   - name: SERVICE_ACCOUNT_EMAIL
     value: "<service_account_email>" 2
```

其中：

<audience>

当管理员设置 GCP Workload Identity 时，在 Google Cloud 中创建，**AUDIENCE** 值必须是以下格式的 URL：

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

<service_account_email>

SERVICE_ACCOUNT_EMAIL 值是在 Operator 操作过程中模拟的 Google Cloud 服务帐户电子邮件，例如：

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

c. 运行以下命令来创建 **Subscription** 对象：

```
$ oc apply -f subscription.yaml
```

6. 如果将 **installPlanApproval** 字段设置为 **Manual**，请手动批准待处理的安装计划以完成 Operator 安装。如需更多信息，请参阅"手动批准待处理的 Operator 更新"。

此时，OLM 已了解所选的 Operator。Operator 的集群服务版本（CSV）应出现在目标命名空间中，由 Operator 提供的 API 应可用于创建。

验证

1. 运行以下命令，检查已安装的 Operator 的 **Subscription** 对象状态：

```
$ oc describe subscription <subscription_name> -n <namespace>
```

2. 如果您为 **SingleNamespace** 安装模式创建了 Operator 组，请运行以下命令检查 **OperatorGroup** 对象的状态：

```
$ oc describe operatorgroup <operatorgroup_name> -n <namespace>
```

其他资源

- operator 组
- 频道名称

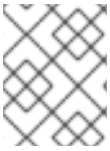
其他资源

- 手动批准待处理的 Operator 更新

第 4 章 管理员任务

4.1. 在集群中添加 OPERATOR

通过使用 Operator Lifecycle Manager (OLM)，集群管理员可将基于 OLM 的 Operator 安装到 OpenShift Container Platform 集群。



注意

如需有关 OLM 如何处理在同一命名空间中并置安装的 Operator 的更新，以及使用自定义全局 Operator 组安装 Operator 的替代方法，请参阅多租户和 Operator 共处。

4.1.1. 关于使用 OperatorHub 安装 Operator

OperatorHub 是一个发现 Operator 的用户界面，它与 Operator Lifecycle Manager（OLM）一起工作，后者在集群中安装和管理 Operator。

作为集群管理员，您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装来自 OperatorHub 的 Operator。将 Operator 订阅到一个或多个命名空间，供集群上的开发人员使用。

安装过程中，您必须为 Operator 确定以下初始设置：

安装模式

选择 **All namespaces on the cluster (default)**将 Operator 安装至所有命名空间；或选择单个命名空间（如果可用），仅在选定命名空间中安装 Operator。本例选择 **All namespaces...** 使 Operator 可供所有用户和项目使用。

更新频道

如果某个 Operator 可通过多个频道获得，则可任选您想要订阅的频道。例如，要通过 **stable** 频道部署（如果可用），则从列表中选择这个选项。

批准策略

您可以选择自动或者手动更新。
如果选择自动更新某个已安装的 Operator，则当所选频道中有该 Operator 的新版本时，Operator Lifecycle Manager（OLM）将自动升级 Operator 的运行实例，而无需人为干预。

如果选择手动更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。

其他资源

- 了解 OperatorHub

4.1.2. 使用 Web 控制台，从 OperatorHub 安装

您可以使用 OpenShift Container Platform Web 控制台从 OperatorHub 安装并订阅 Operator。

先决条件

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

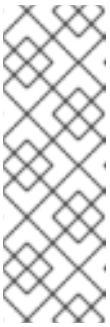
流程

1. 在 Web 控制台中导航至 **Operators → OperatorHub** 页面。
2. 找到您需要的 Operator（滚动页面会在 **Filter by keyword** 框中输入查找关键字）。例如，输入 **advanced** 来查找 Advanced Cluster Management for Kubernetes Operator。
   您还可以根据**基础架构功能**过滤选项。例如，如果您希望 Operator 在断开连接的环境中工作，请选择 **Disconnected**。
3. 选择要显示更多信息的 Operator。

注意

选择 Community Operator 会警告红帽没有认证社区 Operator ; 您必须确认该警告方可继续。

4. 阅读 Operator 信息并点 **Install**。
5. 在 **Install Operator** 页面中，配置 Operator 安装：
   a. 如果要安装 Operator 的特定版本，请从列表中选择 **Update channel** 和 **Version**。您可以在可能具有的任何频道中浏览 Operator 的不同版本，查看该频道和版本的元数据，然后选择您要安装的确切版本。



注意

版本选择默认为所选频道的最新版本。如果选择了频道的最新版本，则默认启用 **Automatic** 批准策略。如果没有为所选频道安装最新版本，则需要**手动**批准。

使用**手动**批准安装 Operator 会导致命名空间中安装的所有 Operator 并使用 **Manual** 批准策略和所有 Operator 一起更新。如果要独立更新 Operator，将 Operator 安装到单独的命名空间中。

   b. 确认 Operator 的安装模式：
      - **All namespaces on the cluster (default)** 选择该项会将 Operator 安装至默认 **openshift-operators** 命名空间，以便供集群中的所有命名空间监视和使用。该选项并非始终可用。
      - **A specific namespace on the cluster**，该项支持您选择单一特定命名空间来安装 Operator。该 Operator 仅限在该单一命名空间中监视和使用。
   c. 对于启用了令牌身份验证的云供应商上的集群：
      - 如果集群在 web 控制台中使用 AWS 安全令牌服务(**STS Mode**)，在角色 ARN 字段中输入服务帐户的 AWS IAM 角色的 Amazon Resource Name (**ARN**)。要创建角色的 ARN，请按照 准备 AWS 帐户 中所述的步骤进行操作。
      - 如果集群使用 Microsoft Entra Workload ID (Web 控制台中的**Workload Identity / Federated Identity Mode**)，请在适当的项中添加客户端 ID、租户 ID 和订阅 ID。
      - 如果集群使用 Google Cloud Platform Workload Identity (**GCP Workload Identity / Federated Identity Mode**)，请在适当的字段中添加项目号、池 ID、供应商 ID 和服务帐户电子邮件。
   d. 对于 **更新批准**，请选择 **Automatic** 或 **Manual** 批准策略。

重要

如果 Web 控制台显示集群使用 AWS STS、Microsoft Entra Workload ID 或 GCP Workload Identity，您必须将 **Update approval** 设置为 **Manual**。

不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

6. 点 **Install** 使 Operator 可供 OpenShift Container Platform 集群上的所选命名空间使用：

   a. 如果选择了**手动**批准策略，订阅的升级状态将保持在 **Upgrading** 状态，直至您审核并批准安装计划。
   在 **Install Plan** 页面批准后，订阅的升级状态将变为 **Up to date**。

   b. 如果选择了 **Automatic** 批准策略，升级状态会在不用人工参与的情况下变为 **Up to date**。

验证

- 在订阅的升级状态变为 **Up to date** 后，选择 **Operators → Installed Operators** 来验证已安装 Operator 的集群服务版本(CSV)是否最终出现了。**状态** 最终会在相关命名空间中变为 **Succeeded**。

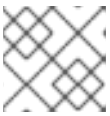
注意

对于 **All namespaces...** 安装模式，状态会在 **openshift-operators** 命名空间中解析为 **Succeeded**，但如果检查其他命名空间，则状态为 **Copied**。

如果没有：

  - 检查 **openshift-operators** 项目（如果选择了 **A specific namespace...** 安装模式）中的 openshift-operators 项目中的 pod 的日志，这会在 **Workloads → Pods** 页面中报告问题以便进一步排除故障。

- 安装 Operator 时，元数据会指示安装了哪个频道和版本。



注意

此目录上下文中仍可使用 **Channel** 和 **Version** 下拉菜单查看其他版本元数据。

其他资源

- 手动批准待处理的 Operator 更新

4.1.3. 使用 CLI 从 OperatorHub 安装

您可以使用 CLI 从 OperatorHub 安装 Operator，而不必使用 OpenShift Container Platform Web 控制台。使用 **oc** 命令来创建或更新一个**订阅**对象。

对于 **SingleNamespace** 安装模式，还必须确保相关命名空间中存在适当的 Operator 组。一个 Operator 组（由 **OperatorGroup** 对象定义），在其中选择目标命名空间，在其中为与 Operator 组相同的命名空间中的所有 Operator 生成所需的 RBAC 访问权限。

提示

在大多数情况下，使用 Web 控制台是首选的方法，因为它会在后台自动执行任务，比如在选择 **SingleNamespace** 模式时自动处理 **OperatorGroup** 和 **Subscription** 对象的创建。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群
- 已安装 OpenShift CLI(**oc**)。

流程

1. 查看 OperatorHub 中集群可用的 Operator 列表：

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

例 4.1. 输出示例

```
NAME                               CATALOG               AGE
3scale-operator                    Red Hat Operators     91m
advanced-cluster-management        Red Hat Operators     91m
amq7-cert-manager                  Red Hat Operators     91m
# ...
couchbase-enterprise-certified     Certified Operators   91m
crunchy-postgres-operator          Certified Operators   91m
mongodb-enterprise                 Certified Operators   91m
# ...
etcd                               Community Operators   91m
jaeger                             Community Operators   91m
kubefed                            Community Operators   91m
# ...
```

记录下所需 Operator 的目录。

2. 检查所需 Operator，以验证其支持的安装模式和可用频道：

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

例 4.2. 输出示例

```
# ...
Kind:         PackageManifest
# ...
    Install Modes: (1)
      Supported:  true
      Type:       OwnNamespace
      Supported:  true
      Type:       SingleNamespace
      Supported:  false
      Type:       MultiNamespace
      Supported:  true
      Type:       AllNamespaces
```

```
# ...
   Entries:
     Name:      example-operator.v3.7.11
     Version:   3.7.11
     Name:      example-operator.v3.7.10
     Version:   3.7.10
    Name:       stable-3.7 2
# ...
   Entries:
     Name:      example-operator.v3.8.5
     Version:   3.8.5
     Name:      example-operator.v3.8.4
     Version:   3.8.4
    Name:       stable-3.8 3
  Default Channel:  stable-3.8 4
```

1 1 指明支持哪些安装模式。

2 2 3 频道名称示例。

4 在没有指定频道时默认选择的频道。

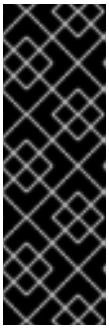
提示

您可以运行以下命令来以 YAML 格式打印 Operator 的版本和频道信息：

```
$ oc get packagemanifests <operator_name> -n <catalog_namespace> -o yaml
```

3. 如果在命名空间中安装多个目录，请运行以下命令从特定目录中查找 Operator 的可用版本和频道：

```
$ oc get packagemanifest \
  --selector=catalog=<catalogsource_name> \
  --field-selector metadata.name=<operator_name> \
  -n <catalog_namespace> -o yaml
```



重要

如果没有指定 Operator 的目录，运行 **oc get packagemanifest** 和 **oc describe packagemanifest** 命令可能会在满足以下条件时从一个意料外的目录中返回一个软件包：

- 在同一命名空间中安装多个目录。
- 目录包含具有相同名称的相同 Operator 或 Operator。

4. 如果要安装的 Operator 支持 **AllNamespaces** 安装模式，而您选择使用这个模式，请跳过这一步，因为 **openshift-operators** 命名空间默认有一个适当的 Operator 组，称为 **global-operators**。

如果要安装的 Operator 支持 **SingleNamespace** 安装模式，而选择使用此模式，您必须确保相关命名空间中存在适当的 Operator 组。如果不存在，您可以按照以下步骤创建一个：

重要

每个命名空间只能有一个 Operator 组。如需更多信息，请参阅 "Operator 组"。

a. 为 **SingleNamespace** 安装模式创建一个 **OperatorGroup** 对象 YAML 文件，如 **operatorgroup.yaml** ：

SingleNamespace 安装模式的 OperatorGroup 对象示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace> 1
spec:
  targetNamespaces:
  - <namespace> 2
```

1 2 对于 **SingleNamespace** 安装模式，对于 **metadata.namespace** 和 **spec.targetNamespaces** 字段使用相同的 **<namespace>** 值。

b. 创建 **OperatorGroup** 对象：

```
$ oc apply -f operatorgroup.yaml
```

5. 创建一个 **Subscription** 对象来订阅一个命名空间到 Operator：

a. 为 **Subscription** 对象创建一个 YAML 文件，如 **subscription.yaml** ：

注意

如果要订阅 Operator 的特定版本，请将 **startCSV** 字段设置为所需的版本，并将 **installPlanApproval** 字段设置为 **Manual**，以防止 Operator 在目录中有更新的版本时自动升级。详情请参阅以下 "Example **Subscription** object with a specific starting Operator version"。

例 4.3. Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: <namespace_per_install_mode> 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: <catalog_name> 4
  sourceNamespace: <catalog_source_namespace> 5
```

```
config:
  env: 6
  - name: ARGS
    value: "-v=10"
  envFrom: 7
  - secretRef:
      name: license-secret
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1. 对于默认的 **AllNamespaces** 安装模式用法，请指定 **openshift-operators** 命名空间。另外，如果创建了自定义全局命名空间，您可以指定一个自定义全局命名空间。对于 **SingleNamespace** 安装模式的使用，请指定相关的单一命名空间。
2. 要订阅的频道的名称。
3. 要订阅的 Operator 的名称。
4. 提供 Operator 的目录源的名称。
5. 目录源的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub 目录源。
6. **env** 参数定义必须存在于由 OLM 创建的 pod 中所有容器中的环境变量列表。
7. **envFrom** 参数定义要在容器中填充环境变量的源列表。
8. **volumes** 参数定义了由 OLM 创建的、在 pod 上必须存在的卷列表。
9. **volumeMounts** 参数定义由 OLM 创建的 pod 中必须存在的卷挂载列表。如果 **volumeMount** 引用不存在的 **卷**，OLM 无法部署 Operator。
10. **tolerations** 参数为 OLM 创建的 pod 定义容限列表。
11. **resources** 参数为 OLM 创建的 pod 中所有容器定义资源限制。
12. **nodeSelector** 参数为 OLM 创建的 pod 定义 **NodeSelector**。

例 4.4. 带有特定启动 Operator 版本的 Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-operator
spec:
  channel: stable-3.7
  installPlanApproval: Manual (1)
  name: example-operator
  source: custom-operators
  sourceNamespace: openshift-marketplace
  startingCSV: example-operator.v3.7.10 (2)
```

1. 如果您指定的版本会被目录中的更新版本取代，则将批准策略设置为 **Manual**。此计划阻止自动升级到更新的版本，且需要在启动 CSV 可以完成安装前手动批准。
2. 设置 Operator CSV 的特定版本。

b. 对于启用了令牌身份验证的云供应商的集群，如 Amazon Web Services (AWS) 安全令牌服务 (STS)、Microsoft Entra Workload ID 或 Google Cloud Platform Workload Identity，按照以下步骤配置您的 **Subscription** 对象：

i. 确保 **Subscription** 对象被设置为手动更新批准：

例 4.5. 带有手动更新批准的 Subscription 对象示例

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual (1)
```

1. 不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

ii. 在 **Subscription** 对象的 **config** 部分包括相关的云供应商相关字段：
如果集群处于 AWS STS 模式，请包含以下字段：

例 4.6. 使用 AWS STS 变量的Subscription 对象示例

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" (1)
```

1 包含角色 ARN 详情。

如果集群处于 Workload ID 模式，请包含以下字段：

例 4.7. 带有 Workload ID 变量的Subscription 对象示例

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" 1
   - name: TENANTID
     value: "<tenant_id>" 2
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" 3
```

1 包含客户端 ID。

2 包含租户 ID。

3 包括订阅 ID。

如果集群处于 GCP Workload Identity 模式，请包括以下字段：

例 4.8. 带有 GCP Workload Identity 变量的Subscription 对象示例

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: AUDIENCE
     value: "<audience_url>" 1
   - name: SERVICE_ACCOUNT_EMAIL
     value: "<service_account_email>" 2
```

其中：

<audience>

当管理员设置 GCP Workload Identity 时，在 Google Cloud 中创建，**AUDIENCE** 值必须是以下格式的 URL：

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

<service_account_email>

SERVICE_ACCOUNT_EMAIL 值是在 Operator 操作过程中模拟的 Google Cloud 服务帐户电子邮件，例如：

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

c. 运行以下命令来创建 **Subscription** 对象：

```
$ oc apply -f subscription.yaml
```

6. 如果将 **installPlanApproval** 字段设置为 **Manual**，请手动批准待处理的安装计划以完成 Operator 安装。如需更多信息，请参阅"手动批准待处理的 Operator 更新"。

此时，OLM 已了解所选的 Operator。Operator 的集群服务版本（CSV）应出现在目标命名空间中，由 Operator 提供的 API 应可用于创建。

验证

1. 运行以下命令，检查已安装的 Operator 的 **Subscription** 对象状态：

```
$ oc describe subscription <subscription_name> -n <namespace>
```

2. 如果您为 **SingleNamespace** 安装模式创建了 Operator 组，请运行以下命令检查 **OperatorGroup** 对象的状态：

```
$ oc describe operatorgroup <operatorgroup_name> -n <namespace>
```

其他资源

- 关于 Operator 组
- 在自定义命名空间中安装全局 Operator
- 手动批准待处理的 Operator 更新

4.1.4. 为多租户集群准备多个 Operator 实例

作为集群管理员，您可以添加多个 Operator 实例以用于多租户集群。这是使用标准 **All namespaces** 安装模式的替代解决方案，可被视为违反最小特权的原则，或 **Multinamespace** 模式（没有被广泛采用）。如需更多信息，请参阅"多租户集群中的Operator"。

在以下步骤中，*租户* 是为一组部署的工作负载共享通用访问和特权的用户或组。*租户 Operator* 是 Operator 实例，仅用于由该租户使用。

先决条件

- 您要安装的 Operator 的所有实例都必须在给定集群中相同。

重要

有关这个限制和其他限制的更多信息，请参阅"多租户集群中的Operator"。

流程

1. 在安装 Operator 前，为租户 Operator 创建一个命名空间，该 Operator 与租户的命名空间分开。例如，如果租户的命名空间是 **team1**，您可以创建一个 **team1-operator** 命名空间：

   a. 定义 **Namespace** 资源并保存 YAML 文件，如 **team1-operator.yaml** ：

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: team1-operator
   ```

   b. 运行以下命令创建命名空间：

   ```
   $ oc create -f team1-operator.yaml
   ```

2. 为租户 Operator 创建 Operator 组，范围到租户的命名空间，只有 **spec.targetNamespaces** 列表中有一个命名空间条目：

   a. 定义 **OperatorGroup** 资源并保存 YAML 文件，如 **team1-operatorgroup.yaml** ：

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: team1-operatorgroup
     namespace: team1-operator
   spec:
     targetNamespaces:
     - team1 1
   ```

   1 1 仅在 **spec.targetNamespaces** 列表中定义租户的命名空间。

   b. 运行以下命令来创建 Operator 组：

   ```
   $ oc create -f team1-operatorgroup.yaml
   ```

后续步骤

- 在租户 Operator 命名空间中安装 Operator。通过在 Web 控制台中使用 OperatorHub 而不是 CLI 来更轻松地执行此任务；有关详细的流程，参阅"使用 Web 控制台从 OperatorHub 安装"。

注意

完成 Operator 安装后，Operator 驻留在租户 Operator 命名空间中，并监视租户命名空间，但 Operator 的 pod 及其服务帐户都无法被租户可见或可用。

其他资源

- 多租户集群中的 Operator

4.1.5. 在自定义命名空间中安装全局 Operator

在使用 OpenShift Container Platform Web 控制台安装 Operator 时，默认行为会将支持 All

namespaces 安装模式的 Operator 安装到默认的 **openshift-operators** 全局命名空间中。这可能导致与命名空间中所有 Operator 共享安装计划和更新策略相关的问题。有关这些限制的详情，请参阅 "Multitenancy 和 Operator colocation"。

作为集群管理员，您可以通过创建自定义全局命名空间并使用该命名空间安装单个或范围 Operator 及其依赖项来手动绕过此默认行为。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 在安装 Operator 前，为所需 Operator 安装创建一个命名空间。此安装命名空间将成为自定义全局命名空间：
   a. 定义 **Namespace** 资源并保存 YAML 文件，如 **global-operators.yaml** ：

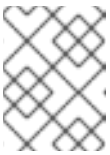
```
apiVersion: v1
kind: Namespace
metadata:
  name: global-operators
```

   b. 运行以下命令创建命名空间：

```
$ oc create -f global-operators.yaml
```

2. 创建自定义 *全局 Operator 组*，这是监视所有命名空间的 Operator 组：
   a. 定义 **OperatorGroup** 资源并保存 YAML 文件，如 **global-operatorgroup.yaml**。省略 **spec.selector** 和 **spec.targetNamespaces** 字段，使其成为一个 *全局 Operator 组*，该组选择所有命名空间：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: global-operatorgroup
  namespace: global-operators
```



注意

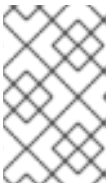
创建的全局 OperatorGroup 的 **status.namespace** 包含空字符串 (**""**)，而该字符串会向正在使用的 Operator 发出信号，要求其监视所有命名空间。

   b. 运行以下命令来创建 Operator 组：

```
$ oc create -f global-operatorgroup.yaml
```

后续步骤

- 在自定义全局命名空间中安装所需的 Operator。因为 Web 控制台没有在 Operator 安装过程中使用自定义全局命名空间填充 **Installed Namespace** 菜单，所以此任务只能使用 OpenShift CLI (**oc**) 执行。如需详细的安装过程，请参阅"使用 CLI 安装 OperatorHub"。



注意

启动 Operator 安装时，如果 Operator 有依赖项，依赖项也会自动安装到自定义全局命名空间中。因此，它对依赖项 Operator 有效，使其具有相同的更新策略和共享安装计划。

其他资源

- 多租户和 Operator 共处

4.1.6. Operator 工作负载的 Pod 放置

默认情况下，Operator Lifecycle Manager（OLM）在安装 Operator 或部署 Operand 工作负载时，会将 pod 放置到任意 worker 节点上。作为管理员，您可以使用节点选择器、污点和容限组合使用项目来控制将 Operator 和 Operands 放置到特定节点。

控制 Operator 和 Operand 工作负载的 pod 放置有以下先决条件：

1. 根据您的要求，确定 pod 的目标节点或一组节点。如果可用，请注意现有标签，如 **node-role.kubernetes.io/app**，用于标识节点。否则，使用计算机器集或直接编辑节点来添加标签，如 **myoperator**。您将在以后的步骤中使用此标签作为项目上的节点选择器。

2. 如果要确保只有具有特定标签的 pod 才能在节点上运行，同时将不相关的工作负载加载到其他节点，通过使用一个计算机器集或直接编辑节点为节点添加污点。使用一个效果来确保与污点不匹配的新 pod 不能调度到节点上。例如，**myoperator:NoSchedule** 污点确保与污点不匹配的新 pod 不能调度到该节点上，但节点上现有的 pod 可以保留。

3. 创建使用默认节点选择器配置的项目，如果您添加了污点，则创建一个匹配的容限。

此时，您创建的项目可在以下情况下用于将 pod 定向到指定节点：

对于 Operator pod

管理员可以在项目中创建 **Subscription** 对象，如以下部分所述。因此，Operator pod 放置在指定的节点上。

对于 Operand pod

通过使用已安装的 Operator，用户可以在项目中创建一个应用程序，这样可将 Operator 拥有的自定义资源（CR）放置到项目中。因此，Operand pod 放置到指定节点上，除非 Operator 在其他命名空间中部署集群范围对象或资源，在这种情况下，不会应用这个自定义的 pod 放置。

其他资源

- 手动或通过计算机器集向节点添加污点和容限
- 创建项目范围节点选择器
- 使用节点选择器和容限创建项目

4.1.7. 控制安装 Operator 的位置

默认情况下，当安装 Operator 时，OpenShift Container Platform 会随机将 Operator pod 安装到其中一个 worker 节点。然而，在某些情况下，您可能希望该 pod 调度到特定节点或一组节点上。

以下示例描述了您可能希望将 Operator pod 调度到特定节点或一组节点的情况：

- 如果 Operator 需要特定的平台，如 **amd64** 或 **arm64**
- 如果 Operator 需要特定的操作系统，如 Linux 或 Windows
- 如果您希望 Operator 在同一个主机上或位于同一机架的主机上工作
- 如果您希望 Operator 在整个基础架构中分散，以避免因为网络或硬件问题而停机

您可以通过在 Operator 的 **Subscription** 对象中添加节点关联性、pod 关联性或 pod 反关联性限制来控制 Operator pod 的安装位置。节点关联性是由调度程序用来确定 pod 的可放置位置的一组规则。pod 关联性允许您确保将相关的 pod 调度到同一节点。通过 Pod 反关联性，您可以防止 pod 调度到节点上。

以下示例演示了如何使用节点关联性或 pod 反关联性将自定义 Metrics Autoscaler Operator 实例安装到集群中的特定节点：

将 Operator pod 放置到特定节点的节点关联性示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - ip-10-0-163-94.us-west-2.compute.internal
#...
```

1 要求 Operator 的 pod 调度到名为 **ip-10-0-163-94.us-west-2.compute.internal** 的节点关联性。

将 Operator pod 放置到带有特定平台的节点关联性示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: 1
```

```
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/arch
              operator: In
              values:
              - arm64
            - key: kubernetes.io/os
              operator: In
              values:
              - linux
#...
```

1 要求 Operator 的 pod 调度到具有 **kubernetes.io/arch=arm64** 和 **kubernetes.io/os=linux** 标签的节点上。

将 Operator pod 放置到一个或多个特定节点的 Pod 关联性示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - test
          topologyKey: kubernetes.io/hostname
#...
```

1 将 Operator 的 pod 放置到具有 **app=test** 标签的 pod 的节点上的 pod 关联性。

防止 Operator pod 来自一个或多个特定节点的 Pod 反关联性示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
```

```
  config:
    affinity:
      podAntiAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: cpu
              operator: In
              values:
              - high
          topologyKey: kubernetes.io/hostname
#...
```

1 一个 pod 反关联性，它可防止 Operator 的 pod 调度到具有 **cpu=high** 标签的 pod 的节点上。

流程

要控制 Operator pod 的放置，请完成以下步骤：

1. 照常安装 Operator。
2. 如果需要，请确保您的节点已标记为正确响应关联性。
3. 编辑 Operator **Subscription** 对象以添加关联性：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity: 1
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - ip-10-0-185-229.ec2.internal
#...
```

1 添加 **nodeAffinity**、**podAffinity** 或 **podAntiAffinity**。有关创建关联性的详情，请参考下面的附加资源部分。

验证

- 要确保 pod 部署到特定的节点上，请运行以下命令：

```
$ oc get pods -o wide
```

-

输出示例

```
NAME                                        READY   STATUS    RESTARTS   AGE   IP
NODE                          NOMINATED NODE   READINESS GATES
custom-metrics-autoscaler-operator-5dcc45d656-bhshg   1/1     Running   0          50s
10.131.0.20   ip-10-0-185-229.ec2.internal   <none>           <none>
```

其他资源

- 了解 pod 关联性
- 了解节点关联性
- 了解如何更新节点上的标签

4.2. 更新安装的 OPERATOR

作为集群管理员，您可以升级以前使用 OpenShift Container Platform 集群上的 Operator Lifecycle Manager（OLM）安装的 Operator。

注意

如需有关 OLM 如何处理在同一命名空间中并置安装的 Operator 的更新，以及使用自定义全局 Operator 组安装 Operator 的替代方法，请参阅多租户和 Operator 共处。

4.2.1. 准备 Operator 更新

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。您可以更改更新频道，以开始跟踪并从更新频道接收更新。

订阅中更新频道的名称可能会因 Operator 而异，但应遵守给定 Operator 中的常规约定。例如，频道名称可能会遵循 Operator 提供的应用程序的次发行版本更新流（**1.2**、**1.3**）或发行的频率（**stable**、**fast**）。

注意

您不能将已安装的 Operator 更改为比当前频道旧的频道。

红帽客户门户网站 Labs 包括以下应用程序，可帮助管理员准备更新其 Operator：

- Red Hat OpenShift Container Platform Operator Update Information Checker

您可以使用应用程序搜索基于 Operator Lifecycle Manager 的 Operator，并在不同版本的 OpenShift Container Platform 中验证每个更新频道的可用 Operator 版本。不包含基于 Cluster Version Operator 的 Operator。

4.2.2. 更改 Operator 的更新频道

您可以使用 OpenShift Container Platform Web 控制台更改 Operator 的更新频道。

提示

如果订阅中的批准策略被设置为 **Automatic**，则更新过程会在所选频道中提供新的 Operator 版本时立即启动。如果批准策略设为 **Manual**，则必须手动批准待处理的更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**。
2. 点击您要更改更新频道的 Operator 名称。
3. 点 **Subscription** 标签页。
4. 点 **Update channel** 下的更新频道名称。
5. 点要更改的更新频道，然后点 **Save**。
6. 对于带有 **自动批准策略** 的订阅，更新会自动开始。返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。
   对于采用**手动**批准策略的订阅，您可以从 **Subscription** 选项卡中手动批准更新。

4.2.3. 手动批准待处理的 Operator 更新

如果已安装的 Operator 的订阅被设置为 **Manual**，则当其当前更新频道中发布新更新时，在开始安装前必须手动批准更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，进入 **Operators → Installed Operators**。
2. 处于待定更新的 Operator 会显示 **Upgrade available** 状态。点您要更新的 Operator 的名称。
3. 点 **Subscription** 标签页。任何需要批准的更新都会在 **Upgrade status** 旁边显示。例如：它可能会显示 **1 requires approval**。
4. 点 **1 requires approval**，然后点 **Preview Install Plan**。
5. 检查列出可用于更新的资源。在满意后，点 **Approve**。
6. 返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。

4.2.4. 其他资源

- 在断开连接的环境中使用 Operator Lifecycle Manager。

4.3. 从集群中删除 OPERATOR

下面介绍如何删除或卸载以前使用 OpenShift Container Platform 集群上的 Operator Lifecycle Manager（OLM）安装的 Operator。

重要

在尝试重新安装同一 Operator 前，您必须已成功并完全卸载了 Operator。没有正确地完全卸载 Operator 可能会留下一些资源，如项目或命名空间，处于"Terminating"状态，并导致尝试重新安装 Operator 时观察到 "error resolving resource" 消息。

如需更多信息，请参阅卸载失败后重新安装 Operator。

4.3.1. 使用 Web 控制台从集群中删除 Operator

集群管理员可以使用 Web 控制台从所选命名空间中删除已安装的 Operator。

先决条件

- 您可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群 Web 控制台。

流程

1. 进入到 **Operators → Installed Operators** 页面。
2. 在 **Filter by name** 字段中滚动或输入关键字以查找您要删除的 Operator。然后点它。
3. 在 **Operator Details** 页面右侧，从 **Actions** 列表中选择 **Uninstall Operator**。
   此时会显示 **Uninstall Operator?** 对话框。
4. 选择 **Uninstall** 来删除 Operator、Operator 部署和 pod。按照此操作，Operator 将停止运行，不再接收更新。

注意

此操作不会删除 Operator 管理的资源，包括自定义资源定义 (CRD) 和自定义资源 (CR) 。Web 控制台和继续运行的集群资源启用的仪表板和导航项可能需要手动清理。要在卸载 Operator 后删除这些，您可能需要手动删除 Operator CRD。

4.3.2. 使用 CLI 从集群中删除 Operator

集群管理员可以使用 CLI 从所选命名空间中删除已安装的 Operator。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- OpenShift CLI (**oc**)安装在您的工作站上。

流程

1. 确保在 **currentCSV** 字段中标识了订阅 Operator 的最新版本（如 **serverless-operator**）。

```
$ oc get subscription.operators.coreos.com serverless-operator -n openshift-serverless -o
yaml | grep currentCSV
```

输出示例

```
  currentCSV: serverless-operator.v1.28.0
```

2. 删除订阅（如 **serverless-operator**）：

```
$ oc delete subscription.operators.coreos.com serverless-operator -n openshift-serverless
```

输出示例

```
subscription.operators.coreos.com "serverless-operator" deleted
```

3. 使用上一步中的 **currentCSV** 值来删除目标命名空间中相应 Operator 的 CSV：

```
$ oc delete clusterserviceversion serverless-operator.v1.28.0 -n openshift-serverless
```

输出示例

```
clusterserviceversion.operators.coreos.com "serverless-operator.v1.28.0" deleted
```

4.3.3. 刷新失败的订阅

在 Operator Lifecycle Manager（OLM）中，如果您订阅的是引用网络中无法访问的镜像的 Operator，您可以在 **openshift-marketplace** 命名空间中找到带有以下错误的作业：

输出示例

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-
bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

输出示例

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get
"https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

因此，订阅会处于这个失败状态，Operator 无法安装或升级。

您可以通过删除订阅、集群服务版本（CSV）及其他相关对象来刷新失败的订阅。重新创建订阅后，OLM 会重新安装 Operator 的正确版本。

先决条件

- 您有一个失败的订阅，无法拉取不能访问的捆绑包镜像。
- 已确认可以访问正确的捆绑包镜像。

流程

1. 从安装 Operator 的命名空间中获取 **Subscription** 和 **ClusterServiceVersion** 对象的名称：

```
$ oc get sub,csv -n <namespace>
```

输出示例

```
NAME                                         PACKAGE               SOURCE             CHANNEL
subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-operators   5.0

NAME                                                  DISPLAY                  VERSION    REPLACES   PHASE
clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift Elasticsearch Operator   5.0.0-65              Succeeded
```

2. 删除订阅：

```
$ oc delete subscription <subscription_name> -n <namespace>
```

3. 删除集群服务版本：

```
$ oc delete csv <csv_name> -n <namespace>
```

4. 在 **openshift-marketplace** 命名空间中获取所有失败的作业的名称和相关配置映射：

```
$ oc get job,configmap -n openshift-marketplace
```

输出示例

```
NAME                                                              COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1           26s        9m30s

NAME                                                               DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3      9m30s
```

5. 删除作业：

```
$ oc delete job <job_name> -n openshift-marketplace
```

这样可确保尝试拉取无法访问的镜像的 Pod 不会被重新创建。

6. 删除配置映射：

```
$ oc delete configmap <configmap_name> -n openshift-marketplace
```

7. 在 Web 控制台中使用 OperatorHub 重新安装 Operator。

验证

- 检查是否已成功重新安装 Operator:

```
$ oc get sub,csv,installplan -n <namespace>
```

4.4. 配置 OPERATOR LIFECYCLE MANAGER 功能

Operator Lifecycle Manager(OLM)控制器由名为 **cluster** 的 **OLMConfig** 自定义资源(CR)进行配置。集群管理员可以修改此资源以启用或禁用某些功能。

本文档概述了由 **OLMConfig** 资源配置的 OLM 当前支持的功能。

4.4.1. 禁用复制的 CSV

当 Operator Lifecycle Manager (OLM) 安装 Operator 时，默认情况下会在 Operator 配置为监视的每个命名空间中创建其集群服务版本 (CSV) 的简化副本。这些 CSV 称为 *复制的 CSV*，并告知用户控制器在给定命名空间中主动协调资源事件。

当 Operator 配置为使用 **AllNamespaces** 安装模式时，与将单个或指定命名空间集为目标时，会在集群中的每个命名空间中创建 Operator 的复制的 CSV。在大型集群中，带有命名空间和安装的 Operator 可能存在于数百个或数千种 CSV 中，复制的 CSV 消耗大量资源，如 OLM 的内存用量、集群 etcd 限值和网络带宽。

为了支持这些较大的集群，集群管理员可以为使用 **AllNamespaces** 模式全局安装的 Operator 禁用复制的 CSV。

注意

如果您禁用复制的 CSV，则在 **AllNamespaces** 模式中安装的 Operator 只将其 CSV 复制到 **openshift** 命名空间，而不是集群中的每个命名空间。在禁用复制的 CSV 模式中，Web 控制台和 CLI 的行为有所不同：

- 在 web 控制台中，默认行为会被修改为显示每个命名空间中的 **openshift** 命名空间中复制的 CSV，即使 CSV 不会实际复制到每个命名空间中。这允许常规用户仍可在其命名空间中查看这些 Operator 的详情，并创建相关的自定义资源 (CR)。
- 在 OpenShift CLI (**oc**) 中，常规用户可以使用 **oc get csvs** 命令查看直接安装在其命名空间中的 Operator，但 **openshift** 命名空间中的复制的 CSV 无法在其命名空间中可见。受此限制影响的 Operator 仍然可用，并继续协调用户命名空间中的事件。
  要查看安装的全局 Operator 的完整列表，类似于 Web 控制台行为，所有经过身份验证的用户都可以运行以下命令：

  ```
  $ oc get csvs -n openshift
  ```

流程

- 编辑名为 **cluster** 的 **OLMConfig** 对象，将 **spec.features.disableCopiedCSVs** 字段设置为 **true**：

```
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
```

```
  features:
    disableCopiedCSVs: true (1)
EOF
```

(1) 为 **AllNamespaces** 安装模式 Operator 禁用复制的 CSV

验证

- 当禁用复制的 CSV 时，OLM 会在 Operator 命名空间中捕获这些信息：

```
$ oc get events
```

输出示例

```
LAST SEEN   TYPE      REASON              OBJECT                                MESSAGE
85s         Warning   DisabledCopiedCSVs  clusterserviceversion/my-csv.v1.0.0   CSV
copying disabled for operators/my-csv.v1.0.0
```

当 **spec.features.disableCopiedCSVs** 字段缺失或设置为 **false** 时，OLM 会为使用 **AllNamespaces** 模式安装的所有 Operator 重新创建复制的 CSV，并删除前面提到的事件。

其他资源

- 安装模式

4.5. 在 OPERATOR LIFECYCLE MANAGER 中配置代理支持

如果在 OpenShift Container Platform 集群中配置了全局代理，Operator Lifecycle Manager（OLM）会自动配置使用集群范围代理管理的 Operator。但是，您也可以配置已安装的 Operator 来覆盖全局代理服务器或注入自定义 CA 证书。

其他资源

- 配置集群范围代理
- 配置自定义 PKI （自定义 CA 证书）

4.5.1. 覆盖 Operator 的代理设置

如果配置了集群范围的出口代理，使用 Operator Lifecycle Manager（OLM）运行的 Operator 会继承其部署上的集群范围代理设置。集群管理员还可以通过配置 Operator 的订阅来覆盖这些代理设置。

重要

操作员必须为任何受管 Operands 处理 pod 中的代理设置环境变量。

先决条件

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

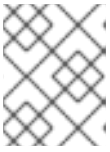
流程

1. 在 Web 控制台中导航至 **Operators → OperatorHub** 页面。
2. 选择 Operator 并点 **Install**。
3. 在 **Install Operator** 页面中，修改 **Subscription** 对象，使其在 **spec** 部分中包含一个或多个以下环境变量：
   - **HTTP_PROXY**
   - **HTTPS_PROXY**
   - **NO_PROXY**

   例如：

   带有代理设置的Subscription 对象覆盖

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: etcd-config-test
  namespace: openshift-operators
spec:
  config:
    env:
    - name: HTTP_PROXY
      value: test_http
    - name: HTTPS_PROXY
      value: test_https
    - name: NO_PROXY
      value: test
  channel: clusterwide-alpha
  installPlanApproval: Automatic
  name: etcd
  source: community-operators
  sourceNamespace: openshift-marketplace
  startingCSV: etcdoperator.v0.9.4-clusterwide
```



注意

这些环境变量也可以使用空值取消设置，以删除所有之前设定的集群范围或自定义代理设置。

   OLM 将这些环境变量作为一个单元处理; 如果至少设置了一个环境变量，则所有三个变量都将被视为覆盖，并且集群范围的默认值不会用于订阅的 Operator 部署。

4. 点击 **Install** 使 Operator 可供所选命名空间使用。
5. 当 Operator 的 CSV 出现在相关命名空间中后，您可以验证部署中是否设置了自定义代理环境变量。例如，使用 CLI：

```
$ oc get deployment -n openshift-operators \
    etcd-operator -o yaml \
    | grep -i "PROXY" -A 2
```

输出示例

```
        - name: HTTP_PROXY
          value: test_http
        - name: HTTPS_PROXY
          value: test_https
        - name: NO_PROXY
          value: test
        image: quay.io/coreos/etcd-
operator@sha256:66a37fd61a06a43969854ee6d3e21088a98b93838e284a6086b13917f96b0
d9c
...
```

4.5.2. 注入自定义 CA 证书

当集群管理员使用配置映射向集群添加自定义 CA 证书时，Cluster Network Operator 会将用户提供的证书和系统 CA 证书合并为一个捆绑包（bundle）。您可以将这个合并捆绑包注入 Operator Lifecycle Manager (OLM) 上运行的 Operator 中，如果您有一个中间人（man-in-the-middle）HTTPS 代理，这将会有用。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 使用配置映射添加自定义 CA 证书至集群。
- 在 OLM 上安装并运行所需的 Operator。

流程

1. 在存在 Operator 订阅的命名空间中创建一个空配置映射，并包括以下标签：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: trusted-ca 1
  labels:
    config.openshift.io/inject-trusted-cabundle: "true" 2
```

1 配置映射的名称。

2 请求 Cluster Network Operator 注入合并的捆绑包。

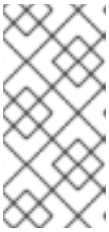
创建此配置映射后，它会立即使用合并捆绑包的证书内容填充。

2. 更新您的 **Subscription** 对象，使其包含 **spec.config** 部分，该部分可将 **trusted-ca** 配置映射作为卷挂载到需要自定义 CA 的 pod 中的每个容器：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-operator
spec:
```

```
package: etcd
channel: alpha
config: 1
  selector:
    matchLabels:
      <labels_for_pods> 2
  volumes: 3
  - name: trusted-ca
    configMap:
      name: trusted-ca
      items:
        - key: ca-bundle.crt 4
          path: tls-ca-bundle.pem 5
  volumeMounts: 6
  - name: trusted-ca
    mountPath: /etc/pki/ca-trust/extracted/pem
    readOnly: true
```

1. 如果不存在，请添加 **config** 部分。
2. 指定标签以匹配 Operator 拥有的 pod。
3. 创建一个 **trusted-ca** 卷。
4. **ca-bundle.crt** 需要作为配置映射键。
5. **tls-ca-bundle.pem** 需要作为配置映射路径。
6. 创建一个 **trusted-ca** 卷挂载。



注意

Operator 的部署可能无法验证颁发机构，并显示 **x509 certificate signed by unknown authority** 错误。即使在使用 Operator 订阅时注入自定义 CA，也会发生这个错误。在这种情况下，您可以使用 Operator 的订阅将 **mountPath** 设置为 trusted-ca 的 **/etc/ssl/certs**。

4.5.3. 其他资源

- 代理证书
- 替换默认入口证书
- 更新 CA 捆绑包

4.6. 查看 OPERATOR 状态

了解 Operator Lifecycle Manager (OLM) 中的系统状态，对于决定和调试已安装 Operator 的问题来说非常重要。OLM 可让您了解订阅和相关目录源的状态以及执行的操作。这样有助于用户更好地理解 Operator 的运行状况。

4.6.1. operator 订阅状况类型

订阅可报告以下状况类型：

表 4.1. 订阅状况类型

状况	描述
CatalogSourcesUnhealthy	用于解析的一个或多个目录源不健康。
InstallPlanMissing	缺少订阅的安装计划。
InstallPlanPending	订阅的安装计划正在安装中。
InstallPlanFailed	订阅的安装计划失败。
ResolutionFailed	订阅的依赖项解析失败。

注意

默认 OpenShift Container Platform 集群 Operator 由 Cluster Version Operator（CVO）管理，它们没有 **Subscription** 对象。应用程序 Operator 由 Operator Lifecycle Manager（OLM）管理，它们具有 **Subscription** 对象。

其他资源

- 刷新失败的订阅

4.6.2. 使用 CLI 查看 Operator 订阅状态

您可以使用 CLI 查看 Operator 订阅状态。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 列出 Operator 订阅：

```
$ oc get subs -n <operator_namespace>
```

2. 使用 **oc describe** 命令检查 **Subscription** 资源：

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. 在命令输出中，找到 Operator 订阅状况类型的 **Conditions** 部分。在以下示例中，**CatalogSourcesUnhealthy** 条件类型具有 **false** 状态，因为所有可用目录源都健康：

输出示例

```
-
```

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```

注意

默认 OpenShift Container Platform 集群 Operator 由 Cluster Version Operator（CVO）管理，它们没有 **Subscription** 对象。应用程序 Operator 由 Operator Lifecycle Manager（OLM）管理，它们具有 **Subscription** 对象。

4.6.3. 使用 CLI 查看 Operator 目录源状态

您可以使用 CLI 查看 Operator 目录源的状态。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 列出命名空间中的目录源。例如，您可以检查 **openshift-marketplace** 命名空间，该命名空间用于集群范围的目录源：

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

输出示例

```
NAME                  DISPLAY               TYPE   PUBLISHER   AGE
certified-operators   Certified Operators   grpc   Red Hat     55m
community-operators   Community Operators   grpc   Red Hat     55m
example-catalog       Example Catalog       grpc   Example Org 2m25s
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. 使用 **oc describe** 命令获取有关目录源的详情和状态：

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

输出示例

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
    Protocol:           grpc
    Service Name:       example-catalog
    Service Namespace:  openshift-marketplace
# ...
```

在上例的输出中，最后观察到的状态是 **TRANSIENT_FAILURE**。此状态表示目录源建立连接时出现问题。

3. 列出创建目录源的命名空间中的 pod：

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

输出示例

```
NAME                                  READY   STATUS             RESTARTS   AGE
certified-operators-cv9nn             1/1     Running            0          36m
community-operators-6v8lp             1/1     Running            0          36m
marketplace-operator-86bfc75f9b-jkgbc 1/1     Running            0          42m
example-catalog-bwt8z                 0/1     ImagePullBackOff   0          3m55s
redhat-operators-smxx8                1/1     Running            0          36m
```

在命名空间中创建目录源时，会在该命名空间中为目录源创建一个 pod。在前面的示例中，**example-catalog-bwt8z** pod 的状态是 **ImagePullBackOff**。此状态表示拉取目录源的索引镜像存在问题。

4. 使用 **oc describe** 命令检查 pod 以获取更多详细信息：

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

输出示例

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age                From               Message
```

```
----     ------          ----               ----               -------
  Normal   Scheduled       48s                default-scheduler  Successfully assigned openshift-
marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s                multus             Add eth0 [10.131.0.40/23] from
openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet            Back-off pulling image
"quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet            Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet            Pulling image "quay.io/example-
org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet            Failed to pull image
"quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading
manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested
resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet            Error: ErrImagePull
```

在前面的示例输出中，错误消息表示目录源的索引镜像因为授权问题而无法成功拉取。例如，索引镜像可能存储在需要登录凭证的 registry 中。

其他资源

- Operator Lifecycle Manager 概念和资源 → Catalog 源
- gRPC 文档：连接状态
- 从私有 registry 访问 Operator 的镜像

4.7. 管理 OPERATOR 条件

作为集群管理员，您可以使用 Operator Lifecycle Manager（OLM）来管理 Operator 状况。

4.7.1. 覆盖 Operator 条件

作为集群管理员，您可能想要忽略由 Operator 报告的、支持的 Operator 条件。当存在时，**Spec.Overrides** 阵列中的 Operator 条件会覆盖 **Spec.Conditions** 阵列中的条件，以便集群管理员可以处理 Operator 向 Operator Lifecycle Manager（OLM）报告了不正确状态的情况。

注意

默认情况下，**OperatorCondition** 对象中不存在 **Spec.Overrides** 数组，直到集群管理员添加为止。**Spec.Conditions** 数组还不存在，直到被用户添加或因为自定义 Operator 逻辑而添加为止。

例如，一个 Operator 的已知版本，它始终会告知它是不可升级的。在这种情况下，尽管报告是不可升级的，您仍然希望升级 Operator。这可以通过在 **OperatorCondition** 对象的 **Spec.Overrides** 阵列中添加 **type** 和 **status** 来覆盖 Operator 条件来实现。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 具有 **OperatorCondition** 对象的 Operator，使用 OLM 安装。

流程

1. 编辑 Operator 的 **OperatorCondition** 对象：

```
$ oc edit operatorcondition <name>
```

2. 在对象中添加 **Spec.Overrides** 数组：

Operator 条件覆盖示例

```
apiVersion: operators.coreos.com/v2
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
  overrides:
  - type: Upgradeable 1
    status: "True"
    reason: "upgradeIsSafe"
    message: "This is a known issue with the Operator where it always reports that it cannot be upgraded."
  conditions:
  - type: Upgradeable
    status: "False"
    reason: "migration"
    message: "The operator is performing a migration."
    lastTransitionTime: "2020-08-24T23:15:55Z"
```

1 允许集群管理员将升级就绪状态更改为 **True**。

4.7.2. 更新 Operator 以使用 Operator 条件

Operator Lifecycle Manager（OLM）会自动为每个它所协调的 **ClusterServiceVersion** 资源创建一个 **OperatorCondition** 资源。CSV 中的所有服务帐户都会被授予 RBAC，以便与 Operator 拥有的 **OperatorCondition** 交互。

Operator 作者可开发其自己的 Operator 来使用 **operator-lib** 库，以便在由 OLM 部署 Operator 后，它可以设置自己的条件。有关将 Operator 条件设置为 Operator 作者的更多信息，请参阅启用 Operator 条件页面。

4.7.2.1. 设置默认值

为了保持向后兼容，OLM 认为在没有 **OperatorCondition** 时代表不使用条件。因此，要使用 Operator 条件的 Operator，在将 pod 的就绪探测设置为 **true** 前应设置默认条件。这为 Operator 提供了一个宽限期，用于将条件更新为正确的状态。

4.7.3. 其他资源

- Operator 条件

4.8. 允许非集群管理员安装 OPERATOR

集群管理员可以使用 *Operator 组*来允许常规用户安装 Operator。

其他资源

- operator 组

4.8.1. 了解 Operator 安装策略

Operator 可能需要广泛权限才可运行，且不同版本需要的权限也可能不同。Operator Lifecycle Manager (OLM) 需要 **cluster-admin** 权限才可运行。默认情况下，Operator 作者可在集群服务版本（CSV）中指定任意权限集，OLM 之后会将其授予 Operator。

为确保 Operator 无法获得集群范围的权限，并且用户无法使用 OLM 升级权限，集群管理员可在将 Operator 添加到集群前手动审核 Operator。集群管理员还可获得一些工具来决定和限制在使用服务账户安装或升级 Operator 期间允许的操作。

集群管理员可以将 Operator 组与赋予了一组权限的服务账户关联。服务帐户在 Operator 上设置策略，通过使用基于角色的访问控制 (RBAC) 规则来确保它们仅在预先确定的边界内运行。因此，Operator 无法执行这些规则未明确允许的任何操作。

通过使用 Operator 组，具有足够权限的用户可以安装具有有限范围的 Operator。因此，更多 Operator Framework 工具可以安全地提供给更多用户，为使用 Operator 构建应用程序提供更丰富的体验。

注意

Subscription 对象的基于角色的访问控制 (RBAC) 会自动授予命名空间中具有 **edit** 或 **admin** 角色的用户。但是，**OperatorGroup** 对象不存在 RBAC；没有什么情况可防止常规用户安装 Operator。预安装 Operator 组实际上会提供安装权限。

在将 Operator 组与服务帐户关联时请注意以下几点：

- **APIService** 和 **CustomResourceDefinition** 资源都由 OLM 使用 **cluster-admin** 角色来创建。不应向与 Operator 组相关联的服务账户授予写入这些资源的权限。
- 与该 Operator 组相关联的所有 Operator 现已被限制在指定服务账户获得的权限范围内。如果 Operator 请求了超出服务账户范围的权限，安装会失败，并显示适当的错误，以便集群管理员能够排除故障并解决问题。

4.8.1.1. 安装场景

在确定是否可在集群上安装或升级 Operator 时，Operator Lifecycle Manager（OLM）会考虑以下情况：

- 集群管理员新建了一个 Operator 组并指定了服务账户。已安装与该 Operator 组关联的所有 Operator，并根据相应服务账户获得的权限运行。
- 集群管理员新建了一个 Operator 组，且不指定任何服务帐户。OpenShift Container Platform 保持向后兼容性，因此会保留默认行为，并允许安装和升级 Operator。
- 对于未指定服务账户的现有 Operator 组，会保留默认行为,并允许安装和升级 Operator。
- 集群管理员更新了现有 Operator 组并指定了服务帐户。OLM 支持现有 Operator 继续根据当前权限运行。现有 Operator 升级后，它会重新安装并根据相应服务账户获得的权限运行，与新 Operator 一样。

- 由 Operator 组指定的服务帐户通过添加或删除权限来更改，或者现有服务账户被换为新服务帐户。现有 Operator 升级后，它会重新安装并根据更新后的服务账户获得的权限运行，与新 Operator 一样。
- 集群管理员从 Operator 组中删除服务账户。默认行为保留，并允许安装和升级 Operator。

4.8.1.2. 安装工作流

当 Operator 组与服务账户绑定，并且安装或升级了 Operator 时，Operator Lifecycle Manager（OLM）会使用以下工作流：

1. OLM 会提取给定订阅对象。
2. OLM 获取与该订阅相关联的 Operator 组。
3. OLM 确定 Operator 组是否指定了服务帐户。
4. OLM 在服务账户范围内创建一个客户端，并使用该范围内客户端来安装 Operator。这样可确保 Operator 请求的任何权限始终限制在 Operator 组中服务账户的权限范围内。
5. OLM 新建一个服务账户，在 CSV 中指定其权限集，并将其分配至 Operator。Operator 将根据所分配的服务账户运行。

4.8.2. 限定 Operator 安装范围

要为 Operator Lifecycle Manager（OLM）上的 Operator 安装和升级提供范围规则，请将服务帐户与 Operator 组关联。

集群管理员可借鉴本例，将一组 Operator 限制到指定命名空间中。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 新建命名空间：

   例 4.9. 创建 Namespace 对象的命令示例

   ```
   $ cat <<EOF | oc create -f -
   apiVersion: v1
   kind: Namespace
   metadata:
     name: scoped
   EOF
   ```

2. 分配 Operator 的权限范围。这包括在新创建的、指定命名空间中创建新服务帐户、相关角色和角色绑定：

   a. 运行以下命令来创建服务帐户：

      例 4.10. 创建 ServiceAccount 对象的命令示例

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: scoped
  namespace: scoped
EOF
```

b. 运行以下命令来创建 secret：

例 4.11. 创建长期 API 令牌 Secret 对象的命令示例

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token (1)
metadata:
  name: scoped
  namespace: scoped
  annotations:
    kubernetes.io/service-account.name: scoped
EOF
```

(1) secret 必须是长期的 API 令牌，供服务帐户使用。

c. 运行以下命令来创建角色。

警告

在本例中，角色授予了在特定的命名空间中进行仍会操作的服务帐户权限。这仅用于演示目的。在生产环境中，您应该创建更为精细的权限集。如需更多信息，请参阅"细粒度权限"。

例 4.12. 创建 Role 和 RoleBinding 对象的命令示例

```
$ cat <<EOF | oc create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: scoped
  namespace: scoped
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: scoped-bindings
  namespace: scoped
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: scoped
subjects:
- kind: ServiceAccount
  name: scoped
  namespace: scoped
EOF
```

3. 运行以下命令，在指定的命名空间中创建 **OperatorGroup** 对象。该 Operator 组以指定的命名空间为目标，以确保其租期仅限于该命名空间。另外，Operator 组允许用户指定服务帐户。

例 4.13. 创建 OperatorGroup 对象的命令示例

```
$ cat <<EOF | oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: scoped
  namespace: scoped
spec:
  serviceAccountName: scoped (1)
  targetNamespaces:
  - scoped
EOF
```

(1) 指定上一步中创建的服务帐户。在指定命名空间中安装的任何 Operator 均会关联至此 Operator 组，因此也会关联到指定的服务账户。

4. 在指定命名空间中创建 **Subscription** 对象以安装 Operator:

例 4.14. 创建 Subscription 对象的命令示例

```
$ cat <<EOF | oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: scoped
spec:
  channel: stable-v1
  name: openshift-cert-manager-operator
  source: <catalog_source_name> (1)
  sourceNamespace: <catalog_source_namespace> (2)
EOF
```

1. 指定已存在于指定命名空间中或位于全局目录命名空间中的目录源，如 **redhat-operators**。
2. 指定创建目录源的命名空间，如 **redhat-operators** 目录的 **openshift-marketplace**。

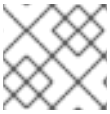
与该 Operator 组相关联的所有 Operator 都仅限于为指定服务账户授予的权限。如果 Operator 请求的权限超出服务账户范围，安装会失败并显示相关错误。

4.8.2.1. 细粒度权限

Operator Lifecycle Manager（OLM）使用 Operator 组中指定的服务账户来创建或更新与正在安装的 Operator 相关的以下资源：

- **ClusterServiceVersion**
- **Subscription**
- **Secret**
- **ServiceAccount**
- **Service**
- **ClusterRole** 和 **ClusterRoleBinding**
- **Role** 和 **RoleBinding**

要将 Operator 限制到指定命名空间，集群管理员可以首先向服务账户授予以下权限：



注意

以下角色只是一个通用示例，具体 Operator 可能需要额外规则。

```
kind: Role
rules:
- apiGroups: ["operators.coreos.com"]
  resources: ["subscriptions", "clusterserviceversions"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: [""]
  resources: ["services", "serviceaccounts"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: ["apps"] 1
  resources: ["deployments"]
  verbs: ["list", "watch", "get", "create", "update", "patch", "delete"]
- apiGroups: [""] 2
  resources: ["pods"]
  verbs: ["list", "watch", "get", "create", "update", "patch", "delete"]
```

1 2 增加创建其他资源的权限，如此处显示的部署和 pod。

另外，如果任何 Operator 指定了 pull secret，还必须增加以下权限：

```
kind: ClusterRole 1
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
---
kind: Role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "update", "patch"]
```

1 需要从 OLM 命名空间中获取 secret。

4.8.3. Operator 目录访问控制

当在全局目录命名空间 **openshift-marketplace** 中创建 Operator 目录时，目录的 Operator 会提供给所有命名空间。在其他命名空间中创建的目录仅使其 Operator 在目录的同一命名空间中可用。

在非集群管理员用户委托了 Operator 安装权限的集群上，集群管理员可能需要进一步控制或限制允许用户安装的 Operator 集合。这可以通过以下操作来实现：

1. 禁用所有默认全局目录。
2. 在预安装相关 Operator 组的同一命名空间中启用自定义、策展的目录。

其他资源

- 禁用默认的 OperatorHub 目录源
- 在集群中添加目录源

4.8.4. 故障排除权限失败

如果因为缺少权限而导致 Operator 安装失败，请按照以下流程找出错误。

流程

1. 查看 **Subscription** 对象。其状态中有一个指向 **InstallPlan** 对象的对象引用 **installPlanRef**，该对象试图为 Operator 创建需要的 **[Cluster]Role[Binding]**：

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: etcd
  namespace: scoped
status:
  installPlanRef:
    apiVersion: operators.coreos.com/v1
    kind: InstallPlan
    name: install-4plp8
    namespace: scoped
```

```
    resourceVersion: "117359"
    uid: 2c1df80e-afea-11e9-bce3-5254009c9c23
```

2. 检查 **InstallPlan** 对象的状态中的任何错误：

```
apiVersion: operators.coreos.com/v1
kind: InstallPlan
status:
  conditions:
  - lastTransitionTime: "2019-07-26T21:13:10Z"
    lastUpdateTime: "2019-07-26T21:13:10Z"
    message: 'error creating clusterrole etcdoperator.v0.9.4-clusterwide-dsfx4:
clusterroles.rbac.authorization.k8s.io
      is forbidden: User "system:serviceaccount:scoped:scoped" cannot create resource
      "clusterroles" in API group "rbac.authorization.k8s.io" at the cluster scope'
    reason: InstallComponentFailed
    status: "False"
    type: Installed
  phase: Failed
```

错误信息中会显示：

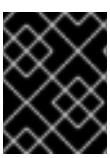
- 创建失败的资源类型，包括资源的 API 组。本例中为 **rbac.authorization.k8s.io** 组中的 **clusterroles**。
- 资源名称。
- 错误类型：**is forbidden** 表明相应用户没有足够权限来执行这一操作。
- 试图创建或更新资源的用户名称。本例中指的是 Operator 组中指定的服务账户。
- 操作范围：**集群范围内**或范围外。
  用户可在服务账户中增加所缺权限，然后迭代操作。

注意

Operator Lifecycle Manager（OLM）目前未提供第一次尝试的完整错误列表。

4.9. 管理自定义目录

集群管理员和 Operator 目录维护人员可以使用 OpenShift Container Platform 的 Operator Lifecycle Manager (OLM) 上的捆绑包格式创建和管理打包的自定义目录。



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。

其他资源

- 红帽提供的 Operator 目录

4.9.1. 先决条件

- 已安装 **opm** CLI。

4.9.2. 基于文件的目录

*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

从 OpenShift Container Platform 4.11 开始，默认的红帽提供的 Operator 目录以基于文件的目录格式发布。通过以过时的 SQLite 数据库格式发布的 4.10，用于 OpenShift Container Platform 4.6 的默认红帽提供的 Operator 目录。

与 SQLite 数据库格式相关的 **opm** 子命令、标志和功能已被弃用，并将在以后的版本中删除。功能仍被支持，且必须用于使用已弃用的 SQLite 数据库格式的目录。

许多 **opm** 子命令和标志都用于 SQLite 数据库格式，如 **opm index prune**，它们无法使用基于文件的目录格式。有关使用基于文件的目录的更多信息，请参阅 Operator Framework 打包格式以及使用 oc-mirror 插件为断开连接的安装 mirror 镜像 。

4.9.2.1. 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

4.9.2.2. 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

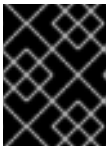
1. 以 YAML 格式将目录镜像的内容提取到 catalog 目录中的 **index.yaml** 文件中：

```
$ 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 4.15. 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>
```

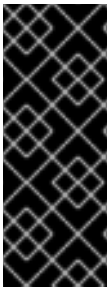
验证

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

4.9.3. 基于 SQLite 的目录



重要

Operator 目录的 SQLite 数据库格式是一个弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

4.9.3.1. 创建基于 SQLite 的索引镜像

您可以使用 **opm** CLI 根据 SQLite 数据库格式创建索引镜像。

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. 启动一个新的索引：

```
$ opm index add \
    --bundles <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
    --tag <registry>/<namespace>/<index_image_name>:<tag> \ 2
    [--binary-image <registry_base_image>] 3
```

1. 要添加到索引中的捆绑包镜像以逗号分隔的列表。
2. 希望索引镜像具有的镜像标签。
3. 可选：用于为目录提供服务的备选 registry 基础镜像。

2. 将索引镜像推送到 registry。

   a. 如果需要，与目标 registry 进行身份验证：

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

   b. 推送索引镜像：

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

4.9.3.2. 更新基于 SQLite 的索引镜像

在将 OperatorHub 配置为使用引用自定义索引镜像的目录源后，集群管理员可通过将捆绑包镜像添加到索引镜像来保持其集群上的可用 Operator 最新状态。

您可以使用 **opm index add** 命令来更新存在的索引镜像。

先决条件

- 已安装 **opm** CLI。
- 您有 **podman** 版本 1.9.3+。
- 构建并推送到 registry 的索引镜像。
- 引用索引镜像的现有目录源。

流程

1. 通过添加捆绑包镜像来更新现有索引：

```
$ opm index add \
    --bundles <registry>/<namespace>/<new_bundle_image>@sha256:<digest> \ 1
    --from-index <registry>/<namespace>/<existing_index_image>:<existing_tag> \ 2
    --tag <registry>/<namespace>/<existing_index_image>:<updated_tag> \ 3
    --pull-tool podman 4
```

1. **--bundles** 标志指定要添加到索引中的、以逗号分隔的额外捆绑包镜像列表。
2. **--from-index** 标志指定之前推送的索引。
3. **--tag** 标志指定要应用到更新的索引镜像的镜像标签。
4. **--pull-tool** 标志指定用于拉取容器镜像的工具。

其中：

<registry>
指定 registry 的主机名，如 **quay.io** 或 **mirror.example.com**。

<namespace>
指定 registry 的命名空间，如 **ocs-dev** 或 **abc**。

<new_bundle_image>
指定要添加到 registry 的新捆绑包镜像，如 **ocs-operator**。

<digest>
指定捆绑包镜像的 SHA 镜像 ID 或摘要，如 **c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a41**。

<existing_index_image>
指定之前推送的镜像，如 **abc-redhat-operator-index**。

<existing_tag>
指定之前推送的镜像标签，如 **4.19**。

<updated_tag>
指定要应用到更新的索引镜像的镜像标签，如 **4.19.1**。

示例命令

```
$ opm index add \
```

```
    --bundles quay.io/ocs-dev/ocs-
operator@sha256:c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a
41 \
    --from-index mirror.example.com/abc/abc-redhat-operator-index:4.19 \
    --tag mirror.example.com/abc/abc-redhat-operator-index:4.19.1 \
    --pull-tool podman
```

2. 推送更新的索引镜像：

```
$ podman push <registry>/<namespace>/<existing_index_image>:<updated_tag>
```

3. Operator Lifecycle Manager（OLM）会在常规时间段内自动轮询目录源中引用的索引镜像，验证是否已成功添加新软件包：

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

4.9.3.3. 过滤基于 SQLite 的索引镜像

基于 Operator Bundle Format 的索引镜像是 Operator 目录的容器化快照。您可以过滤或 *prune（修剪）*除指定的软件包列表以外的所有索引，创建只包含您想要的 Operator 的源索引副本。

先决条件

- 您有 **podman** 版本 1.9.3+。
- **grpcurl**（第三方命令行工具）
- 已安装 **opm** CLI。
- 访问支持 Docker v2-2 的 registry

流程

1. 通过目标 registry 进行身份验证：

```
$ podman login <target_registry>
```

2. 确定您要包括在您的修剪索引中的软件包列表。

   a. 运行您要修剪容器中的源索引镜像。例如：

```
$ podman run -p50051:50051 \
    -it registry.redhat.io/redhat/redhat-operator-index:v4.19
```

   输出示例

```
Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4.19...
Getting image source signatures
Copying blob ae8a0c23f5b1 done
...
INFO[0000] serving registry                 database=/database/index.db port=50051
```

   b. 在一个单独的终端会话中，使用 **grpcurl** 命令获取由索引提供的软件包列表：

```
$ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
```

c. 检查 **package.out** 文件，确定要保留在此列表中的哪个软件包名称。例如：

软件包列表片断示例

```
...
{
  "name": "advanced-cluster-management"
}
...
{
  "name": "jaeger-product"
}
...
{
{
  "name": "quay-operator"
}
...
```

d. 在您执行 **podman run** 命令的终端会话中，按 **Ctrl** 和 **C** 停止容器进程。

3. 运行以下命令来修剪指定软件包以外的所有源索引：

```
$ opm index prune \
   -f registry.redhat.io/redhat/redhat-operator-index:v4.19 \ 1
   -p advanced-cluster-management,jaeger-product,quay-operator \ 2
   [-i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4.19] \ 3
   -t <target_registry>:<port>/<namespace>/redhat-operator-index:v4.19 4
```

1. 到修剪的索引。
2. 要保留的软件包用逗号隔开。
3. 只适用于 IBM Power® 和 IBM Z® 镜像： Operator Registry 基础镜像和与目标 OpenShift Container Platform 集群主版本和次版本匹配的标签。
4. 用于正在构建新索引镜像的自定义标签。

4. 运行以下命令将新索引镜像推送到目标 registry:

```
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4.19
```

其中 **<namespace>** 是 registry 上的任何现有命名空间。

4.9.4. 目录源和 pod 安全准入

OpenShift Container Platform 4.11 中引入了 *Pod 安全准入*，以确保 pod 安全标准。使用基于 SQLite 的目录格式构建的目录源以及在 OpenShift Container Platform 4.11 无法运行受限 pod 安全强制前发布的 **opm** CLI 工具的版本。

在 OpenShift Container Platform 4.19 中，命名空间默认没有受限 pod 安全强制，默认的目录源安全模式设置为 **legacy**。

计划在以后的 OpenShift Container Platform 发行版本中包括所有命名空间的默认限制强制。当发生受限强制时，目录源 pod 规格的安全上下文必须与受限 pod 安全标准匹配。如果您的目录源镜像需要不同的 pod 安全标准，则必须明确设置命名空间的 pod 安全准入标签。

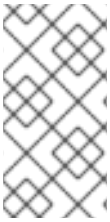
注意

如果您不想以受限方式运行基于 SQLite 的目录源 pod，则不需要在 OpenShift Container Platform 4.19 中更新目录源。

但是，建议您采取措施来确保目录源在受限 pod 安全强制下运行。如果您不采取措施来确保目录源在受限 pod 安全强制下运行，您的目录源可能不会在以后的 OpenShift Container Platform 版本中运行。

作为目录作者，您可以通过完成以下任一操作来启用与受限 pod 安全强制的兼容性：

- 将您的目录迁移到基于文件的目录格式。
- 使用 OpenShift Container Platform 4.11 或更高版本发布的 **opm** CLI 工具版本更新您的目录镜像。



注意

SQLite 数据库目录格式已弃用，但仍然被红帽支持。在以后的发行版本中，不支持 SQLite 数据库格式，目录将需要迁移到基于文件的目录格式。从 OpenShift Container Platform 4.11 开始，默认的红帽提供的 Operator 目录以基于文件的目录格式发布。基于文件的目录与受限 pod 安全强制兼容。

如果您不想更新 SQLite 数据库目录镜像，或将目录迁移到基于文件的目录格式，您可以将目录配置为使用升级的权限运行。

其他资源

- 了解并管理 pod 安全准入

4.9.4.1. 将 SQLite 数据库目录迁移到基于文件的目录格式

您可以将已弃用的 SQLite 数据库格式目录更新为基于文件的目录格式。

先决条件

- SQLite 数据库目录源
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 在工作站上具有 OpenShift Container Platform 4.19 发布的最新版本 **opm** CLI 工具。

流程

1. 运行以下命令，将 SQLite 数据库目录迁移到基于文件的目录：

```
$ opm migrate <registry_image> <fbc_directory>
```

2. 运行以下命令，为您的基于文件的目录生成 Dockerfile：

```
$ opm generate dockerfile <fbc_directory> \
  --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4.19
```

后续步骤

- 生成的 Dockerfile 可以构建、标记并推送到 registry。

其他资源

- 在集群中添加目录源

4.9.4.2. 重建 SQLite 数据库目录镜像

您可以使用 OpenShift Container Platform 版本发布的 **opm** CLI 工具的最新版本重建 SQLite 数据库目录镜像。

先决条件

- SQLite 数据库目录源
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 在工作站上具有 OpenShift Container Platform 4.19 发布的最新版本 **opm** CLI 工具。

流程

- 运行以下命令，使用 **opm** CLI 工具的最新版本重建目录：

```
$ opm index add --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4.19 \
  --from-index <your_registry_image> \
  --bundles "" -t \<your_registry_image>
```

4.9.4.3. 配置目录以使用升级的权限运行

如果您不想更新 SQLite 数据库目录镜像，或将目录迁移到基于文件的目录格式，您可以执行以下操作以确保目录源在默认 pod 安全强制更改为受限时运行：

- 在目录源定义中手动将目录安全模式设置为 legacy。此操作可确保您的目录使用旧权限运行，即使默认目录安全模式更改为 restricted。
- 为基准或特权 pod 安全强制标记目录源命名空间。

注意

SQLite 数据库目录格式已弃用，但仍然被红帽支持。在以后的发行版本中，不支持 SQLite 数据库格式，目录将需要迁移到基于文件的目录格式。基于文件的目录与受限 pod 安全强制兼容。

先决条件

- SQLite 数据库目录源
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 支持运行带有升级 pod 安全准入标准 **baseline** 或 **privileged** 的 pod 的目标命名空间

流程

1. 通过将 **spec.grpcPodConfig.securityContextConfig** 标签设置为 **legacy** 来编辑 **CatalogSource** 定义，如下例所示：

   CatalogSource 定义示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-catsrc
  namespace: my-ns
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: legacy
  image: my-image:latest
```

   提示

   在 OpenShift Container Platform 4.19 中，**spec.grpcPodConfig.securityContextConfig** 字段默认设置为 **legacy**。在以后的发行版本中，计划默认设置将更改为 **restricted**。如果您的目录无法在受限强制下运行，建议您手动将此字段设置为 **legacy**。

2. 编辑 **<namespace>.yaml** 文件，将升级的 pod 安全准入标准添加到目录源命名空间中，如下例所示：

   <namespace>.yaml 文件示例

```
apiVersion: v1
kind: Namespace
metadata:
...
  labels:
    security.openshift.io/scc.podSecurityLabelSync: "false" 1
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: baseline 2
  name: "<namespace_name>"
```

   1. 通过在命名空间中添加 **security.openshift.io/scc.podSecurityLabelSync=false** 标签来关闭 pod 安全标签同步。
   2. 应用 pod 安全准入 **pod-security.kubernetes.io/enforce** 标签。将标签设置为 **baseline** 或 **privileged**。使用 **baseline** pod 安全配置集，除非命名空间中的其他工作负载需要 **privileged** 配置集。

4.9.5. 在集群中添加目录源

将目录源添加到 OpenShift Container Platform 集群可为用户发现和安装 Operator。集群管理员可以创建一个 **CatalogSource** 对象来引用索引镜像。OperatorHub 使用目录源来填充用户界面。

提示

或者，您可以使用 Web 控制台管理目录源。在 **Administration → Cluster Settings → Configuration → OperatorHub** 页面中，点 **Sources** 选项卡，您可以在其中创建、更新、删除、禁用和启用单独的源。

先决条件

- 构建并推送索引镜像到 registry。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建一个 **CatalogSource** 对象来引用索引镜像。
   a. 根据您的规格修改以下内容，并将它保存为 **catalogSource.yaml** 文件：

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace (1)
  annotations:
    olm.catalogImageTemplate: (2)
      "<registry>/<namespace>/<index_image_name>:v{kube_major_version}.
{kube_minor_version}.{kube_patch_version}"
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> (3)
  image: <registry>/<namespace>/<index_image_name>:<tag> (4)
  displayName: My Operator Catalog
  publisher: <publisher_name> (5)
  updateStrategy:
    registryPoll: (6)
      interval: 30m
```

(1) 如果您希望目录源对所有命名空间中的用户全局可用，请指定 **openshift-marketplace** 命名空间。否则，您可以指定一个不同的命名空间来对目录进行作用域并只对该命名空间可用。

(2) 可选：将 **olm.catalogImageTemplate** 注解设置为索引镜像名称，并使用一个或多个 Kubernetes 集群版本变量，如为镜像标签构建模板时所示。

(3) 指定 **legacy** 或 **restricted** 的值。如果没有设置该字段，则默认值为 **legacy**。在以后的 OpenShift Container Platform 发行版本中，计划默认值为 **restricted**。

注意

如果您的目录无法使用 **restricted** 权限运行，建议您手动将此字段设置为 **legacy**。

4 指定索引镜像。如果您在镜像名称后指定了标签，如 **:v4.19**，则目录源 Pod 会使用镜像 pull 策略 **Always**，这意味着 pod 始终在启动容器前拉取镜像。如果您指定了摘要，如 **@sha256:<id>**，则镜像拉取策略为 **IfNotPresent**，这意味着仅在节点上不存在的镜像时才拉取镜像。

5 指定发布目录的名称或机构名称。

6 目录源可以自动检查新版本以保持最新。

b. 使用该文件创建 **CatalogSource** 对象：

```
$ oc apply -f catalogSource.yaml
```

2. 确定成功创建以下资源。

a. 检查 pod:

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

输出示例

```
NAME                                  READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6             1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. 检查目录源：

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

输出示例

```
NAME              DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog  My Operator Catalog  grpc          5s
```

c. 检查软件包清单：

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

输出示例

```
NAME                  CATALOG            AGE
jaeger-product          My Operator Catalog  93s
```

现在，您可以在 OpenShift Container Platform Web 控制台中通过 **OperatorHub** 安装 Operator。

其他资源

- Operator Lifecycle Manager 概念和资源 → Catalog 源
- 从私有 registry 访问 Operator 的镜像
- 镜像拉取（pull）策略

4.9.6. 从私有 registry 访问 Operator 的镜像

如果与 Operator Lifecycle Manager（OLM）管理的 Operator 相关的某些镜像托管在需要经过身份验证的容器镜像 registry（也称为私有 registry）中时，在默认情况下，OLM 和 OperatorHub 将无法拉取镜像。要启用访问权限，可以创建一个包含 registry 验证凭证的 pull secret。通过引用一个或多个目录源的 pull secret，OLM 可以处理将 secret 放置到 Operator 和目录命名空间中以允许进行安装。

Operator 或其 Operands 所需的其他镜像可能会需要访问私有 registry。对于这种情况，OLM 不处理将 secret 放置到目标租户命名空间中，但身份验证凭证可以添加到全局范围集群 pull secret 中，或单独的命名空间服务帐户中，以启用所需的访问权限。

在决定由 OLM 管理的 Operator 是否有适当的拉取访问权限时，应该考虑以下类型的镜像：

索引镜像

CatalogSource 对象可以引用索引镜像，该镜像使用 Operator 捆绑包格式，并作为托管在镜像 registry 中的容器镜像的目录源打包。如果索引镜像托管在私有 registry 中，可以使用 secret 来启用拉取访问。

捆绑包镜像

Operator 捆绑包镜像是元数据和清单，并被打包为容器镜像，代表 Operator 的一个特定版本。如果目录源中引用的任何捆绑包镜像托管在一个或多个私有 registry 中，可以使用一个 secret 来启用拉取（pull）访问。

Operator 和 Operand 镜像

如果从目录源安装的 Operator 使用私有镜像，对于 Operator 镜像本身或它监视的 Operand 镜像之一，Operator 将无法安装，因为部署无法访问所需的 registry 身份验证。在目录源中引用 secret 不会使 OLM 将 secret 放置到安装 Operands 的目标租户命名空间中。
相反，身份验证详情被添加到 **openshift-config** 命名空间中的全局集群 pull secret 中，该 secret 提供对集群上所有命名空间的访问。另外，如果不允许访问整个集群，则可将 pull secret 添加到目标租户命名空间的 **default** 服务帐户中。

您可以通过为 registry 凭证创建 secret 并添加用于相关目录的 secret，从 Operator 访问镜像。

先决条件

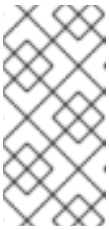
- 您至少有一个托管在私有 registry 中的以下之一：
  - 一个索引镜像或目录镜像。
  - 一个 Operator 捆绑包镜像。
  - 一个 Operator 或 Operand 镜像。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 为每个必需的私有 registry 创建 secret。

a. 登录到私有 registry 以创建或更新 registry 凭证文件：

```
$ podman login <registry>:<port>
```



注意

registry 凭证的文件路径可能会根据用于登录到 registry 的容器工具的不同而有所不同。对于 **podman** CLI，默认位置为 **${XDG_RUNTIME_DIR}/containers/auth.json**。对于 **docker** CLI，默认位置为 **/root/.docker/config.json**。

b. 建议您为每个 secret 只包含一个 registry 的凭证，并在单独的 secret 中为多个 registry 管理凭证。后续步骤中的 **CatalogSource** 可包括多个 secret，OpenShift Container Platform 会将这些 secret 合并为一个单独的虚拟凭证文件，以便在镜像拉取过程中使用。
默认情况下，registry 凭证文件可以在一个 registry 中存储多个 registry 或多个存储库的详细信息。确认您的文件的当前内容。例如：

为多个 registry 存储凭证的文件

```
{
   "auths": {
      "registry.redhat.io": {
         "auth": "FrNHNydQXdzclNqdg=="
      },
      "quay.io": {
         "auth": "fegdsRib21iMQ=="
      },
      "https://quay.io/my-namespace/my-user/my-image": {
         "auth": "eWfjwsDdfsa221=="
      },
      "https://quay.io/my-namespace/my-user": {
         "auth": "feFweDdscw34rR=="
      },
      "https://quay.io/my-namespace": {
         "auth": "frwEews4fescyq=="
      }
   }
}
```

由于此文件用于后续步骤中创建 secret，请确保为每个文件只存储一个 registry 的详情。这可使用以下方法之一完成：

- 使用 **podman logout <registry>** 命令为额外 registry 删除凭证，直到您只保留一个 registry。
- 编辑 registry 凭证文件，将 registry 详情分开以存储在多个文件中。例如：

  为一个 registry 存储凭证的文件

```
{
        "auths": {
                "registry.redhat.io": {
                        "auth": "FrNHNydQXdzclNqdg=="
```

```
        }
    }
}
```

为另一个 registry 存储凭证的文件

```
{
    "auths": {
        "quay.io": {
            "auth": "Xd2lhdsbnRib21iMQ=="
        }
    }
}
```

c. 在 **openshift-marketplace** 命名空间中创建 secret，其中包含私有 registry 的身份验证凭证：

```
$ oc create secret generic <secret_name> \
  -n openshift-marketplace \
  --from-file=.dockerconfigjson=<path/to/registry/credentials> \
  --type=kubernetes.io/dockerconfigjson
```

重复此步骤，为任何其他需要的私有 registry 创建额外的 secret，更新 **--from-file** 标志以指定另一个 registry 凭证文件路径。

2. 创建或更新现有的 **CatalogSource** 对象以引用一个或多个 secret：

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  secrets: 1
  - "<secret_name_1>"
  - "<secret_name_2>"
  grpcPodConfig:
    securityContextConfig: <security_mode> 2
  image: <registry>:<port>/<namespace>/<image>:<tag>
  displayName: My Operator Catalog
  publisher: <publisher_name>
  updateStrategy:
    registryPoll:
      interval: 30m
```

1 添加 **spec.secrets** 部分并指定任何所需 secret。

2 指定 **legacy** 或 **restricted** 的值。如果没有设置该字段，则默认值为 **legacy**。在以后的 OpenShift Container Platform 发行版本中，计划默认值为 **restricted**。

注意

如果您的目录无法使用 **restricted** 权限运行，建议您手动将此字段设置为 **legacy**。

3. 如果订阅的 Operator 引用的任何 Operator 或 Operand 镜像需要访问私有 registry，您可以提供对集群中的所有命名空间或单独的目标租户命名空间的访问。

- 要访问集群中的所有命名空间，请将身份验证详情添加到 **openshift-config** 命名空间中的全局集群 pull secret 中。

警告

集群资源必须调整为新的全局 pull secret，这样可暂时限制集群的可用性。

a. 从全局 pull secret 中提取 **.dockerconfigjson** 文件：

```
$ oc extract secret/pull-secret -n openshift-config --confirm
```

b. 使用所需私有 registry 或 registry 的身份验证凭证更新 **.dockerconfigjson** 文件，并将它保存为新文件：

```
$ cat .dockerconfigjson | \
    jq --compact-output '.auths["<registry>:<port>/<namespace>/"] |= . + {"auth":"
<token>"}' \ 1
    > new_dockerconfigjson
```

1 将 **<registry>:<port> /<namespace>** 替换为私有 registry 的详情，将 **<token>** 替换为您的身份验证凭证。

c. 使用新文件更新全局 pull secret：

```
$ oc set data secret/pull-secret -n openshift-config \
    --from-file=.dockerconfigjson=new_dockerconfigjson
```

- 要更新单个命名空间，请向 Operator 的服务帐户添加一个 pull secret，以便在目标租户命名空间中访问该 Operator。

a. 在租户命名空间中为 **openshift-marketplace** 重新创建 secret：

```
$ oc create secret generic <secret_name> \
    -n <tenant_namespace> \
    --from-file=.dockerconfigjson=<path/to/registry/credentials> \
    --type=kubernetes.io/dockerconfigjson
```

b. 通过搜索租户命名空间来验证 Operator 的服务帐户名称：

```
$ oc get sa -n <tenant_namespace> 1
```

1 如果 Operator 安装在单独的命名空间中，请搜索该命名空间。如果 Operator 已为所有命名空间安装，请搜索 **openshift-operators** 命名空间。

输出示例

```
NAME          SECRETS   AGE
builder       2         6m1s
default       2         6m1s
deployer      2         6m1s
etcd-operator 2         5m18s 1
```

1 已安装的 etcd Operator 的服务帐户。

c. 将 secret 链接到 Operator 的服务帐户：

```
$ oc secrets link <operator_sa> \
    -n <tenant_namespace> \
     <secret_name> \
    --for=pull
```

其他资源

- 如需了解更多与 secret 相关的信息，包括用于 registry 凭证的 secret 类型的信息，请参阅什么是 secret?。
- 如需了解有关更改此 secret 的影响的更多详细信息，请参阅更新全局集群 pull secret。
- 如需了解更多有关将 pull secret 链接到每个命名空间的服务帐户的详情，请参阅允许 Pod 引用其他安全 registry 中的镜像。

4.9.7. 禁用默认的 OperatorHub 目录源

在 OpenShift Container Platform 安装过程中，默认为 OperatorHub 配置由红帽和社区项目提供的源内容的 operator 目录。作为集群管理员，您可以禁用默认目录集。

流程

- 通过在 **OperatorHub** 对象中添加 **disableAllDefaultSources: true** 来禁用默认目录的源：

```
$ oc patch OperatorHub cluster --type json \
    -p '[{"op": "add", "path": "/spec/disableAllDefaultSources", "value": true}]'
```

提示

或者，您可以使用 Web 控制台管理目录源。在 **Administration → Cluster Settings → Configuration → OperatorHub** 页面中，点 **Sources** 选项卡，您可以在其中创建、更新、删除、禁用和启用单独的源。

4.9.8. 删除自定义目录

作为集群管理员，您可以删除之前添加到集群中的自定义 Operator 目录，方法是删除相关的目录源。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 在 Web 控制台的 **Administrator 视角中**，导航到 **Administration → Cluster Settings**。
2. 点 **Configuration** 选项卡，然后点 **OperatorHub**。
3. 点 **Sources** 选项卡。
4. 选择您要删除的目录的 Options 菜单  ，然后点 **Delete CatalogSource**。

4.10. 在断开连接的环境中使用 OPERATOR LIFECYCLE MANAGER。

对于在断开连接的环境中的 OpenShift Container Platform 集群，Operator Lifecycle Manager (OLM) 默认无法访问托管在远程 registry 上的红帽提供的 OperatorHub 源，因为这些远程源需要足够的互联网连接。

但是，作为集群管理员，如果您有一个有完全互联网访问的工作站，则仍可以让集群在断开连接的环境中使用 OLM。工作站需要完全访问互联网来拉取远程 OperatorHub 内容，用于准备远程源的本地镜像，并将内容推送到镜像 registry。

镜像 registry 可以位于堡垒主机上，它需要连接到您的工作站和断开连接的集群，或者一个完全断开连接的或 *airgapped* 主机，这需要可移动介质物理将镜像内容移到断开连接的环境中。

本指南描述了在断开连接的环境中启用 OLM 所需的流程：

- 为 OLM 禁用默认远程 OperatorHub 源。
- 使用有完全互联网访问的工作站来创建并推送 OperatorHub 内容的本地镜像到镜像 registry。
- 将 OLM 配置为从镜像 registry 上的本地源而不是默认的远程源安装和管理 Operator。

在断开连接的环境中启用 OLM 后，您可以继续使用不受限制的工作站在发布较新版本的 Operator 时保持本地 OperatorHub 源更新。

如需更多信息，请参阅断开连接环境部分中的在断开连接的环境中使用 Operator Lifecycle Manager。

4.11. 目录源 POD 调度

当源类型 **grpc** 的 Operator Lifecycle Manager (OLM) 目录源定义 **spec.image** 时，Catalog Operator 会创建一个提供定义的镜像内容的 pod。默认情况下，此 pod 在规格中定义以下内容：

- 只有 **kubernetes.io/os=linux** 节点选择器。
- 默认优先级类名称：**system-cluster-critical**。
- 没有容限。

作为管理员，您可以通过修改 **CatalogSource** 对象的可选 **spec.grpcPodConfig** 部分中的字段来覆盖这些值。

重要

Marketplace Operator **openshift-marketplace** 负责管理默认的 **OperatorHub** 自定义资源 (CR)。此 CR 管理 **CatalogSource** 对象。如果您试图修改 **CatalogSource** 对象的 **spec.grpcPodConfig** 部分中的字段，Marketplace Operator 会自动恢复这些修改。默认情况下，如果您修改 **CatalogSource** 对象的 **spec.grpcPodConfig** 部分中的字段，Marketplace Operator 会自动恢复这些更改。

要将持久性更改应用到 **CatalogSource** 对象，您必须首先禁用一个默认的 **CatalogSource** 对象。

其他资源

- OLM 概念和资源 → Catalog source

4.11.1. 在本地级别禁用默认 CatalogSource 对象

您可以通过禁用默认的 **CatalogSource** 对象，对 **CatalogSource** 对象（如目录源 pod）应用到本地级别。当默认 **CatalogSource** 对象的配置不符合您的机构需求时，请考虑默认配置。默认情况下，如果您修改 **CatalogSource** 对象的 **spec.grpcPodConfig** 部分中的字段，Marketplace Operator 会自动恢复这些更改。

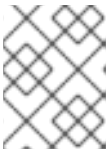
Marketplace Operator **openshift-marketplace** 负责管理 **OperatorHub** 的默认自定义资源 (CR)。**OperatorHub** 管理 **CatalogSource** 对象。

要将持久性更改应用到 **CatalogSource** 对象，您必须首先禁用一个默认的 **CatalogSource** 对象。

流程

- 要在本地级别禁用所有默认 **CatalogSource** 对象，请输入以下命令：

```
$ oc patch operatorhub cluster -p '{"spec": {"disableAllDefaultSources": true}}' --type=merge
```



注意

您还可以将默认 **OperatorHub** CR 配置为禁用所有 **CatalogSource** 对象或禁用特定对象。

其他资源

- OperatorHub 自定义资源
- 禁用默认的 OperatorHub 目录源

4.11.2. 覆盖目录源 pod 的节点选择器

先决条件

- 源类型的 **CatalogSource** 对象， 定义了 **spec.image**

流程

- 编辑 **CatalogSource** 对象并添加或修改 **spec.grpcPodConfig** 部分，使其包含以下内容：

```
grpcPodConfig:
  nodeSelector:
    custom_label: <label>
```

其中 **<label>** 是您希望目录源 pod 用于调度的节点选择器的标签。

其他资源

- 使用节点选择器将 pod 放置到特定节点

4.11.3. 覆盖目录源 pod 的优先级类名称

先决条件

- 源类型的 **CatalogSource** 对象， 定义了 **spec.image**

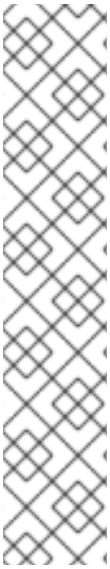
流程

- 编辑 **CatalogSource** 对象并添加或修改 **spec.grpcPodConfig** 部分，使其包含以下内容：

```
grpcPodConfig:
  priorityClassName: <priority_class>
```

其中 **<priority_class>** 是以下之一：

  - Kubernetes 提供的默认优先级类之一：**system-cluster-critical** 或 **system-node-critical**
  - 用于分配默认优先级的空集合 (**""**)
  - 预先存在的和自定义优先级类



注意

在以前的版本中，唯一可以被覆盖的 pod 调度参数是 **priorityClassName**。这可以通过将 **operatorframework.io/priorityclass** 注解添加到 **CatalogSource** 对象来实现。例如：

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    operatorframework.io/priorityclass: system-cluster-critical
```

如果 **CatalogSource** 对象同时定义了注解和 **spec.grpcPodConfig.priorityClassName**，注解优先于配置参数。

其他资源

- Pod 优先级类

4.11.4. 覆盖目录源 pod 的容限

先决条件

- 源类型的 **CatalogSource** 对象， 定义了 **spec.image**

流程

- 编辑 **CatalogSource** 对象并添加或修改 **spec.grpcPodConfig** 部分，使其包含以下内容：

```
grpcPodConfig:
  tolerations:
    - key: "<key_name>"
      operator: "<operator_type>"
      value: "<value>"
      effect: "<effect>"
```

其他资源

- 了解污点和容限

4.12. TROUBLESHOOTING OPERATOR 的问题

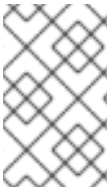
如果遇到 Operator 问题，请验证 Operator 订阅状态。检查集群中的 Operator pod 健康状况，并收集 Operator 日志以进行诊断。

4.12.1. operator 订阅状况类型

订阅可报告以下状况类型：

表 4.2. 订阅状况类型

状况	描述
CatalogSourcesUnhealthy	用于解析的一个或多个目录源不健康。
InstallPlanMissing	缺少订阅的安装计划。
InstallPlanPending	订阅的安装计划正在安装中。
InstallPlanFailed	订阅的安装计划失败。
ResolutionFailed	订阅的依赖项解析失败。



注意

默认 OpenShift Container Platform 集群 Operator 由 Cluster Version Operator（CVO）管理，它们没有 **Subscription** 对象。应用程序 Operator 由 Operator Lifecycle Manager（OLM）管理，它们具有 **Subscription** 对象。

其他资源

- 目录健康要求

4.12.2. 使用 CLI 查看 Operator 订阅状态

您可以使用 CLI 查看 Operator 订阅状态。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 列出 Operator 订阅：

```
$ oc get subs -n <operator_namespace>
```

2. 使用 **oc describe** 命令检查 **Subscription** 资源：

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. 在命令输出中，找到 Operator 订阅状况类型的 **Conditions** 部分。在以下示例中，**CatalogSourcesUnhealthy** 条件类型具有 **false** 状态，因为所有可用目录源都健康：

输出示例

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```

注意

默认 OpenShift Container Platform 集群 Operator 由 Cluster Version Operator（CVO）管理，它们没有 **Subscription** 对象。应用程序 Operator 由 Operator Lifecycle Manager（OLM）管理，它们具有 **Subscription** 对象。

4.12.3. 使用 CLI 查看 Operator 目录源状态

您可以使用 CLI 查看 Operator 目录源的状态。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 列出命名空间中的目录源。例如，您可以检查 **openshift-marketplace** 命名空间，该命名空间用于集群范围的目录源：

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

输出示例

```
NAME                  DISPLAY              TYPE   PUBLISHER   AGE
certified-operators   Certified Operators  grpc   Red Hat     55m
community-operators   Community Operators  grpc   Red Hat     55m
example-catalog       Example Catalog      grpc   Example Org 2m25s
redhat-operators      Red Hat Operators    grpc   Red Hat     55m
```

2. 使用 **oc describe** 命令获取有关目录源的详情和状态：

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

输出示例

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
```

```
    Protocol:          grpc
    Service Name:      example-catalog
    Service Namespace: openshift-marketplace
# ...
```

在上例的输出中，最后观察到的状态是 **TRANSIENT_FAILURE**。此状态表示目录源建立连接时出现问题。

3. 列出创建目录源的命名空间中的 pod：

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

输出示例

```
NAME                                  READY   STATUS             RESTARTS   AGE
certified-operators-cv9nn             1/1     Running            0          36m
community-operators-6v8lp             1/1     Running            0          36m
marketplace-operator-86bfc75f9b-jkgbc 1/1     Running            0          42m
example-catalog-bwt8z                 0/1     ImagePullBackOff   0          3m55s
redhat-operators-smxx8                1/1     Running            0          36m
```

在命名空间中创建目录源时，会在该命名空间中为目录源创建一个 pod。在前面的示例中，**example-catalog-bwt8z** pod 的状态是 **ImagePullBackOff**。此状态表示拉取目录源的索引镜像存在问题。

4. 使用 **oc describe** 命令检查 pod 以获取更多详细信息：

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

输出示例

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age                From               Message
  ----     ------          ----               ----               -------
  Normal   Scheduled       48s                default-scheduler  Successfully assigned openshift-marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s                multus             Add eth0 [10.131.0.40/23] from openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet            Back-off pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet            Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet            Pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet            Failed to pull image "quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet            Error: ErrImagePull
```

在前面的示例输出中，错误消息表示目录源的索引镜像因为授权问题而无法成功拉取。例如，索引镜像可能存储在需要登录凭证的 registry 中。

其他资源

- Operator Lifecycle Manager 概念和资源 → Catalog 源
- gRPC 文档：连接状态
- 从私有 registry 访问 Operator 的镜像

4.12.4. 查询 Operator pod 状态

您可以列出集群中的 Operator pod 及其状态。您还可以收集详细的 Operator pod 概述。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- API 服务仍然可以正常工作。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 列出集群中运行的 Operator。输出包括 Operator 版本、可用性和运行时间信息：

```
$ oc get clusteroperators
```

2. 列出在 Operator 命名空间中运行的 Operator pod，以及 pod 状态、重启和年龄：

```
$ oc get pod -n <operator_namespace>
```

3. 输出详细的 Operator pod 概述：

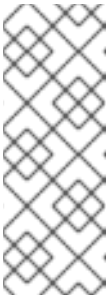
```
$ oc describe pod <operator_pod_name> -n <operator_namespace>
```

4. 如果 Operator 问题特定于某个节点，则在该节点上查询 Operator 容器状态。
   a. 为节点启动 debug pod：

```
$ oc debug node/my-node
```

   b. 将 **/host** 设为 debug shell 中的根目录。debug pod 在 pod 中的 **/host** 中挂载主机的 root 文件系统。将根目录改为 **/host**，您可以运行主机可执行路径中包含的二进制文件：

```
# chroot /host
```



注意

运行 Red Hat Enterprise Linux CoreOS (RHCOS) 的 OpenShift Container Platform 4.19 集群节点不可变，它依赖于 Operator 来应用集群更改。不建议使用 SSH 访问集群节点。但是，如果 OpenShift Container Platform API 不可用，或 kubelet 在目标节点上无法正常工作, **oc** 操作将会受到影响。在这种情况下，可以使用 **ssh core@<node>.<cluster_name>.<base_domain>** 来访问节点。

c. 列出节点容器的详细信息，包括状态和关联的 pod ID:

```
# crictl ps
```

d. 列出节点上特定 Operator 容器的信息。以下示例列出了 **network-operator** 容器的信息：

```
# crictl ps --name network-operator
```

e. 退出 debug shell。

4.12.5. 收集 Operator 日志

如果遇到 Operator 问题，您可以从 Operator pod 日志中收集详细诊断信息。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- API 服务仍然可以正常工作。
- 已安装 OpenShift CLI(**oc**)。
- 您有 control plane 或 control plane 机器的完全限定域名。

流程

1. 列出在 Operator 命名空间中运行的 Operator pod，以及 pod 状态、重启和年龄：

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

2. 检查 Operator pod 的日志：

```
$ oc logs pod/<pod_name> -n <operator_namespace>
```

如果 Operator pod 具有多个容器，则上述命令将会产生一个错误，其中包含每个容器的名称。从独立容器查询日志：

```
$ oc logs pod/<operator_pod_name> -c <container_name> -n <operator_namespace>
```

3. 如果 API 无法正常工作，请使用 SSH 来查看每个 control plane 节点上的 Operator pod 和容器日志。将 **<master-node>.<cluster_name>.<base_domain>** 替换为适当的值。

a. 列出每个 control plane 节点上的 pod：

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl pods
```

b. 对于任何未显示 **Ready** 状态的 Operator pod，详细检查 Pod 的状态。将 **<operator_pod_id>** 替换为上一命令输出中列出的 Operator pod ID:

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspectp
<operator_pod_id>
```

c. 列出与 Operator pod 相关的容器：

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl ps --pod=
<operator_pod_id>
```

d. 对于任何未显示 **Ready** 状态的 Operator 容器，请详细检查容器的状态。将 **<container_id>** 替换为上一命令输出中列出的容器 ID:

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspect
<container_id>
```

e. 检查任何未显示 **Ready** 状态的 Operator 容器的日志。将 **<container_id>** 替换为上一命令输出中列出的容器 ID:

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl logs -f
<container_id>
```

注意

运行 Red Hat Enterprise Linux CoreOS (RHCOS) 的 OpenShift Container Platform 4.19 集群节点不可变，它依赖于 Operator 来应用集群更改。不建议使用 SSH 访问集群节点。在尝试通过 SSH 收集诊断数据前，请运行 **oc adm must gather** 和其他 **oc** 命令看它们是否可以提供足够的数据。但是，如果 OpenShift Container Platform API 不可用，或 kubelet 在目标节点上无法正常工作, **oc** 操作将会受到影响。在这种情况下,可以使用 **ssh core@<node>.<cluster_name>.<base_domain>** 来访问节点。

4.12.6. 禁用 Machine Config Operator 自动重新引导

当 Machine Config Operator（MCO）进行配置更改时，Red Hat Enterprise Linux CoreOS（RHCOS）必须重启才能使更改生效。无论配置更改是自动还是手动的，RHCOS 节点都会自动重启，除非它已被暂停。

注意

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

为了避免不必要的中断，您可以修改机器配置池（MCP）以防止在 Operator 更改机器配置后自动重启。

4.12.6.1. 使用控制台禁用 Machine Config Operator 自动重新引导

为了避免对 Machine Config Operator（MCO）所做的更改造成不必要的中断，您可以使用 OpenShift Container Platform Web 控制台修改机器配置池（MCP），以防止 MCO 在那个池中对节点进行任何更改。这会防止任何通常属于 MCO 更新过程一部分的重启。

注意

请参阅第二个有关 禁用 Machine Config Operator 自动重新引导 的**备注**。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

要暂停或取消暂停自动 MCO 更新重新引导：

- 暂停自动引导过程：
  1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
  2. 点 **Compute → MachineConfigPools**。
  3. 在 **MachineConfigPools** 页面中，点击 **master** 或 **worker**，具体取决于您要暂停重新引导的节点。
  4. 在 **master** 或 **worker** 页面中，点 **YAML**。

5. 在 YAML 中，将 **spec.paused** 字段更新为 **true**。

   MachineConfigPool 对象示例

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   # ...
   spec:
   # ...
     paused: true (1)
   # ...
   ```

   (1) 将 **spec.paused** 字段更新为 **true** 以暂停重新引导。

6. 要验证 MCP 是否已暂停，请返回到 **MachineConfigPools** 页面。
   在 **MachineConfigPools** 页面中，您修改的 MCP 报告了 **Paused** 列中为 **True**。

   如果 MCP 在暂停时有待处理的变化，**Updated** 列为 **False**，**Updating** 为 **False**。当 **Updated** 为 **True** 且 **Updating** 为 **False** 时，代表没有待处理的更改。

   重要

   如果有尚未进行的更改（ **Updated** 和 **Updating** 字段都是 **False**）,建议您尽快调度一个维护窗口用于重启。使用以下步骤取消暂停自动引导过程，以应用上一次重启后排队的更改。

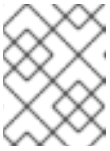
- 取消暂停自动引导过程：

  1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
  2. 点 **Compute → MachineConfigPools**。
  3. 在 **MachineConfigPools** 页面中，点击 **master** 或 **worker**，具体取决于您要暂停重新引导的节点。
  4. 在 **master** 或 **worker** 页面中，点 **YAML**。
  5. 在 YAML 中，将 **spec.paused** 字段更新为 **false**。

     MachineConfigPool 对象示例

     ```
     apiVersion: machineconfiguration.openshift.io/v1
     kind: MachineConfigPool
     # ...
     spec:
     # ...
       paused: false (1)
     # ...
     ```

     (1) 将 **spec.paused** 字段更新为 **false** 以允许重启。



注意

通过取消暂停 MCP，MCO 应用所有暂停的更改，根据需要重启 Red Hat Enterprise Linux CoreOS（RHCOS）。

6. 要验证 MCP 是否已暂停，请返回到 **MachineConfigPools** 页面。
在 **MachineConfigPools** 页面中，您修改的 MCP 报告 **Paused** 列为 **False**。

   如果 MCP 正在应用任何待处理的更改，**Updated** 列为 **False**，**Updating** 列为 **True**。当 **Updated** 为 **True** 且 **Updating** 为 **False** 时，不会再进行任何更改。

4.12.6.2. 使用 CLI 禁用 Machine Config Operator 自动重新引导

为了避免对 Machine Config Operator（MCO）所做的更改造成不必要的中断，您可以使用 OpenShift CLI（oc）来修改机器配置池（MCP），以防止 MCO 在那个池中对节点进行任何更改。这会防止任何通常属于 MCO 更新过程一部分的重启。

注意

请参阅第二个有关 禁用 Machine Config Operator 自动重新引导 的备注。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。

流程

要暂停或取消暂停自动 MCO 更新重新引导：

- 暂停自动引导过程：

  1. 更新 **MachineConfigPool** 自定义资源，将 **spec.paused** 字段设置为 **true**。

     Control plane（master）节点

     ```
     $ oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/master
     ```

     Worker 节点

     ```
     $ oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/worker
     ```

  2. 验证 MCP 是否已暂停：

     Control plane（master）节点

     ```
     $ oc get machineconfigpool/master --template='{{.spec.paused}}'
     ```

     Worker 节点

     ```
     $ oc get machineconfigpool/worker --template='{{.spec.paused}}'
     ```

输出示例

```
true
```

spec.paused 字段为 **true**，MCP 暂停。

3. 确定 MCP 是否有待处理的更改：

```
# oc get machineconfigpool
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING
master   rendered-master-33cf0a1254318755d7b48002c597bf91   True      False
worker   rendered-worker-e405a5bdb0db1295acea08bcca33fa60   False     False
```

如果 **UPDATED** 列是 **False**，**UPDATING** 为 **False**，则有待处理的更改。当 **UPDATED** 为 **True** 且 **UPDATING** 为 **False** 时，没有待处理的更改。在上例中，worker 节点有待处理的变化。control plane 节点没有任何待处理的更改。

重要

如果有尚未进行的更改（ **Updated** 和 **Updating** 字段都是 **False**）,建议您尽快调度一个维护窗口用于重启。使用以下步骤取消暂停自动引导过程，以应用上一次重启后排队的更改。

- 取消暂停自动引导过程：

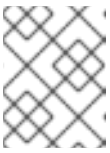
1. 更新 **MachineConfigPool** 自定义资源，将 **spec.paused** 字段设置为 **false**。

Control plane（master）节点

```
$ oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/master
```

Worker 节点

```
$ oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/worker
```



注意

通过取消暂停 MCP，MCO 应用所有暂停的更改，根据需要重启 Red Hat Enterprise Linux CoreOS（RHCOS）。

2. 验证 MCP 是否已取消暂停：

Control plane（master）节点

```
$ oc get machineconfigpool/master --template='{{.spec.paused}}'
```

Worker 节点

```
$ oc get machineconfigpool/worker --template='{{.spec.paused}}'
```

输出示例

```
false
```

spec.paused 字段为 **false**，MCP 被取消暂停。

3. 确定 MCP 是否有待处理的更改：

```
$ oc get machineconfigpool
```

输出示例

```
NAME     CONFIG                                UPDATED  UPDATING
master   rendered-master-546383f80705bd5aeaba93   True     False
worker   rendered-worker-b4c51bb33ccaae6fc4a6a5   False    True
```

如果 MCP 正在应用任何待处理的更改，**UPDATED** 列为 **False**，**UPDATING** 列为 **True**。当 **UPDATED** 为 **True** 且 **UPDATING** 为 **False** 时，没有进一步的更改。在上例中，MCO 正在更新 worker 节点。

4.12.7. 刷新失败的订阅

在 Operator Lifecycle Manager（OLM）中，如果您订阅的是引用网络中无法访问的镜像的 Operator，您可以在 **openshift-marketplace** 命名空间中找到带有以下错误的作业：

输出示例

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-
bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

输出示例

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get
"https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

因此，订阅会处于这个失败状态，Operator 无法安装或升级。

您可以通过删除订阅、集群服务版本（CSV）及其他相关对象来刷新失败的订阅。重新创建订阅后，OLM 会重新安装 Operator 的正确版本。

先决条件

- 您有一个失败的订阅，无法拉取不能访问的捆绑包镜像。
- 已确认可以访问正确的捆绑包镜像。

流程

1. 从安装 Operator 的命名空间中获取 **Subscription** 和 **ClusterServiceVersion** 对象的名称：

```
$ oc get sub,csv -n <namespace>
```

输出示例

```
NAME                                         PACKAGE                SOURCE                CHANNEL
subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-operators   5.0

NAME                                                  DISPLAY                            VERSION    REPLACES   PHASE
clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift Elasticsearch Operator   5.0.0-65              Succeeded
```

2. 删除订阅：

```
$ oc delete subscription <subscription_name> -n <namespace>
```

3. 删除集群服务版本：

```
$ oc delete csv <csv_name> -n <namespace>
```

4. 在 **openshift-marketplace** 命名空间中获取所有失败的作业的名称和相关配置映射：

```
$ oc get job,configmap -n openshift-marketplace
```

输出示例

```
NAME                                                              COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1           26s        9m30s

NAME                                                              DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3      9m30s
```

5. 删除作业：

```
$ oc delete job <job_name> -n openshift-marketplace
```

这样可确保尝试拉取无法访问的镜像的 Pod 不会被重新创建。

6. 删除配置映射：

```
$ oc delete configmap <configmap_name> -n openshift-marketplace
```

7. 在 Web 控制台中使用 OperatorHub 重新安装 Operator。

验证

- 检查是否已成功重新安装 Operator:

```
$ oc get sub,csv,installplan -n <namespace>
```

4.12.8. 卸载失败后重新安装 Operator

在尝试重新安装同一 Operator 前，您必须已成功并完全卸载了 Operator。无法正确卸载 Operator 可以正确地保留资源，如项目或命名空间，处于"Terminating"状态，并导致"错误解析资源"信息。例如：

Project 资源描述示例

```
...
  message: 'Failed to delete all resource types, 1 remaining: Internal error occurred:
    error resolving resource'
...
```

这些类型的问题可能会阻止 Operator 成功重新安装。

警告

强制删除命名空间可能无法解决 "Terminating" 状态问题，并可能导致不稳定或无法预计的集群行为，因此最好尝试查找可能会阻止命名空间被删除的相关资源。如需更多信息，请参阅红帽知识库解决方案 #4165791，请专注于注意和警告部分。

以下流程演示了如何在无法重新安装 Operator 时进行故障排除，因为之前安装 Operator 中的自定义资源定义 (CRD) 会阻止相关的命名空间成功删除。

流程

1. 检查是否有与 Operator 相关的命名空间是否处于"Terminating"状态：

```
$ oc get namespaces
```

输出示例

```
operator-ns-1                                Terminating
```

2. 检查卸载失败后是否存在与 Operator 相关的 CRD：

```
$ oc get crds
```

注意

CRD 是全局集群定义；与 CRD 相关的实际自定义资源 (CR) 实例可能位于其他命名空间中，也可以是全局集群实例。

3. 如果有任何 CRD 由 Operator 提供或管理，并在卸载后删除 CRD：

```
$ oc delete crd <crd_name>
```

4. 检查卸载后是否仍然存在与 Operator 相关的剩余 CR 实例，如果存在，请删除 CR：

a. 要搜索的 CR 类型可能很难确定卸载后，需要了解 Operator 管理哪些 CRD。例如，如果您要对提供 **EtcdCluster** CRD 的 etcd Operator 卸载进行故障排除，您可以在命名空间中搜索剩余的 **EtcdCluster** CR：

```
$ oc get EtcdCluster -n <namespace_name>
```

另外，您还可以在所有命名空间中搜索：

```
$ oc get EtcdCluster --all-namespaces
```

b. 如果有任何剩余的 CR 应该被删除，请删除实例：

```
$ oc delete <cr_name> <cr_instance_name> -n <namespace_name>
```

5. 检查命名空间删除是否已成功解决：

```
$ oc get namespace <namespace_name>
```

重要

如果命名空间或其他 Operator 资源仍然没有完全卸载，请联系红帽支持。

6. 在 Web 控制台中使用 OperatorHub 重新安装 Operator。

验证

- 检查是否已成功重新安装 Operator:

```
$ oc get sub,csv,installplan -n <namespace>
```

其他资源

- 从集群中删除 Operator
- 在集群中添加 Operator

第 5 章 开发 OPERATOR

5.1. 令牌身份验证

5.1.1. 云供应商上的 Operator 的令牌身份验证

许多云提供商可以通过使用提供短期、有限权限安全凭据的帐户令牌来启用身份验证。

OpenShift Container Platform 包含 Cloud Credential Operator (CCO)，用于将云供应商凭证作为自定义资源定义(CRD)进行管理。**CredentialsRequest** 自定义资源(CR)的 CCO 同步，允许 OpenShift Container Platform 组件使用所需的特定权限请求云供应商凭证。

在以前的版本中，在 CCO 处于 *手动模式* 的集群中，由 Operator Lifecycle Manager (OLM)管理的 Operator 通常会在 OperatorHub 中提供详细说明，供用户手动置备任何所需的云凭证。

从 OpenShift Container Platform 4.14 开始，CCO 可以在启用的集群中检测它，以便在某些云供应商中使用短期凭证。然后，它可以半自动置备某些凭证，只要 Operator 作者启用了其 Operator 来支持更新的 CCO。

其他资源

- 关于 Cloud Credential Operator
- 使用 AWS STS 的 OLM 管理的 Operator 基于 CCO 的工作流
- 使用 Microsoft Entra Workload ID 为 OLM 管理的 Operator 基于 CCO 的工作流
- 使用 GCP Workload Identity 的 OLM 管理的 Operator 基于 CCO 的工作流

5.1.2. 使用 AWS STS 的 OLM 管理的 Operator 基于 CCO 的工作流

当在 AWS 上运行的 OpenShift Container Platform 集群处于 Security Token Service (STS) 模式时，这意味着集群会利用 AWS 和 OpenShift Container Platform 的功能在应用程序级别使用 IAM 角色。STS 使应用程序能够提供可假定 IAM 角色的 JSON Web Token (JWT)。

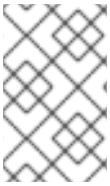
JWT 包含用于 **sts:AssumeRoleWithWebIdentity** IAM 操作的 Amazon 资源名称 (ARN)，以允许服务帐户临时获得权限。JWT 包含 AWS IAM 可验证的 **ProjectedServiceAccountToken** 的签名密钥。服务帐户令牌本身（已签名）用作假定 AWS 角色所需的 JWT。

Cloud Credential Operator (CCO) 是在云供应商上运行的 OpenShift Container Platform 集群中默认安装的集群 Operator。对于 STS，CCO 提供以下功能：

- 检测它是否在启用了 STS 的集群中运行的
- 检查 **CredentialsRequest** 对象是否有字段，它们提供授予 Operator 对 AWS 资源的访问权限所需的信息

即使处于手动模式，CCO 也会执行此检测。正确配置后，CCO 将带有所需访问信息的 **Secret** 对象项目到 Operator 命名空间中。

从 OpenShift Container Platform 4.14 开始，CCO 可以通过扩大使用 **CredentialsRequest** 对象来自动处理此任务，该对象可请求创建包含 STS 工作流所需的信息的 **Secret**。通过 Web 控制台或 CLI 安装 Operator 时，用户可以提供角色 ARN。



注意

不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

作为 Operator 作者准备一个 Operator 以用于 OpenShift Container Platform 4.14 或更高版本中更新的 CCO，您应该指示用户并添加代码来处理之前 CCO 版本的比较，除了处理 STS 令牌身份验证外（如果您的 Operator 还没有启用 STS）。推荐的方法是为 **CredentialsRequest** 对象提供正确填充的 STS 相关字段，并让 CCO 为您的 Secret 创建 **Secret**。

重要

如果您计划支持早于版本 4.14 的 OpenShift Container Platform 集群，请考虑为用户提供有关如何使用 CCO 实用程序(**ccoctl**)手动创建带有 STS 额外信息的 secret 的说明。早期 CCO 版本不知道集群中的 STS 模式，且无法为您创建 secret。

您的代码应检查永远不会出现的 secret，并警告用户以遵循您提供的回退指令。如需更多信息，请参阅"Alternative method"子部分。

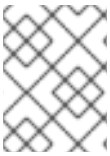
其他资源

- OLM 管理的 Operator 支持使用 AWS STS 进行身份验证
- 使用 Web 控制台从 OperatorHub 安装
- 使用 CLI 从 OperatorHub 安装

5.1.2.1. 启用 Operator 以支持使用 AWS STS 的基于 CCO 的工作流

作为 Operator 作者设计在 Operator Lifecycle Manager (OLM)上运行的项目，您可以通过自定义项目来支持 Cloud Credential Operator (CCO)，使 Operator 能够对启用了 STS 的 OpenShift Container Platform 集群上的 AWS 进行身份验证。

使用此方法，Operator 负责创建 **CredentialsRequest** 对象并读取生成的 **Secret** 对象，并需要 RBAC 权限。



注意

默认情况下，与 Operator 部署相关的 pod 会挂载 **serviceAccountToken** 卷，以便在生成的 **Secret** 对象中引用服务帐户令牌。

先决条件

- OpenShift Container Platform 4.14 或更高版本
- 处于 STS 模式的集群
- 基于 OLM 的 Operator 项目

流程

1. 更新 Operator 项目的 **ClusterServiceVersion** (CSV)对象：

a. 确保 Operator 有 RBAC 权限来创建 **CredentialsRequests** 对象：

例 5.1. clusterPermissions 列表示例

```
# ...
install:
  spec:
    clusterPermissions:
    - rules:
      - apiGroups:
        - "cloudcredential.openshift.io"
        resources:
        - credentialsrequests
        verbs:
        - create
        - delete
        - get
        - list
        - patch
        - update
        - watch
```

b. 添加以下注解来声明对使用 AWS STS 的基于 CCO 工作流的方法的支持：

```
# ...
metadata:
 annotations:
   features.operators.openshift.io/token-auth-aws: "true"
```

2. 更新 Operator 项目代码：

a. 从 pod 上由 **Subscription** 对象设置的环境变量获取角色 ARN。例如：

```
// Get ENV var
roleARN := os.Getenv("ROLEARN")
setupLog.Info("getting role ARN", "role ARN = ", roleARN)
webIdentityTokenPath := "/var/run/secrets/openshift/serviceaccount/token"
```

b. 确保具有 **CredentialsRequest** 对象已准备好修补并应用。例如：

例 5.2. CredentialsRequest 对象创建示例

```
import (
   minterv1 "github.com/openshift/cloud-credential-
operator/pkg/apis/cloudcredential/v1"
   corev1 "k8s.io/api/core/v1"
   metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

var in = minterv1.AWSProviderSpec{
   StatementEntries: []minterv1.StatementEntry{
     {
       Action: []string{
         "s3:*",
       },
```

```
        Effect:   "Allow",
        Resource: "arn:aws:s3:*:*:*",
      },
    },
 STSIAMRoleARN: "<role_arn>",
}

var codec = minterv1.Codec
var ProviderSpec, _ = codec.EncodeProviderSpec(in.DeepCopyObject())

const (
  name      = "<credential_request_name>"
  namespace = "<namespace_name>"
)

var CredentialsRequestTemplate = &minterv1.CredentialsRequest{
  ObjectMeta: metav1.ObjectMeta{
      Name:      name,
      Namespace: "openshift-cloud-credential-operator",
  },
  Spec: minterv1.CredentialsRequestSpec{
    ProviderSpec: ProviderSpec,
    SecretRef: corev1.ObjectReference{
      Name:      "<secret_name>",
      Namespace: namespace,
    },
    ServiceAccountNames: []string{
      "<service_account_name>",
    },
    CloudTokenPath:   "",
  },
}
```

另外，如果您从 YAML 表单的 **CredentialsRequest** 对象开始（例如，作为 Operator 项目代码的一部分），您可以以不同的方式处理它：

例 5.3. 以 YAML 格式创建 CredentialsRequest 对象示例

```
// CredentialsRequest is a struct that represents a request for credentials
type CredentialsRequest struct {
  APIVersion string `yaml:"apiVersion"`
  Kind       string `yaml:"kind"`
  Metadata   struct {
    Name      string `yaml:"name"`
    Namespace string `yaml:"namespace"`
  } `yaml:"metadata"`
  Spec struct {
    SecretRef struct {
      Name      string `yaml:"name"`
      Namespace string `yaml:"namespace"`
    } `yaml:"secretRef"`
    ProviderSpec struct {
      APIVersion     string `yaml:"apiVersion"`
      Kind           string `yaml:"kind"`
```

```
        StatementEntries []struct {
          Effect   string   `yaml:"effect"`
          Action   []string `yaml:"action"`
          Resource string   `yaml:"resource"`
        } `yaml:"statementEntries"`
        STSIAMRoleARN   string `yaml:"stsIAMRoleARN"`
      } `yaml:"providerSpec"`

      // added new field
      CloudTokenPath   string `yaml:"cloudTokenPath"`
  } `yaml:"spec"`
}

// ConsumeCredsRequestAddingTokenInfo is a function that takes a YAML filename
and two strings as arguments
// It unmarshals the YAML file to a CredentialsRequest object and adds the token
information.
func ConsumeCredsRequestAddingTokenInfo(fileName, tokenString, tokenPath
string) (*CredentialsRequest, error) {
  // open a file containing YAML form of a CredentialsRequest
  file, err := os.Open(fileName)
  if err != nil {
    return nil, err
  }
  defer file.Close()

  // create a new CredentialsRequest object
  cr := &CredentialsRequest{}

  // decode the yaml file to the object
  decoder := yaml.NewDecoder(file)
  err = decoder.Decode(cr)
  if err != nil {
    return nil, err
  }

  // assign the string to the existing field in the object
  cr.Spec.CloudTokenPath = tokenPath

  // return the modified object
  return cr, nil
}
```

注意

目前不支持在 Operator 捆绑包中添加 **CredentialsRequest** 对象。

c. 在凭证请求中添加角色 ARN 和 Web 身份令牌路径，并在 Operator 初始化过程中应用它：

例 5.4. 在 Operator 初始化过程中应用 CredentialsRequest 对象示例

```
// apply CredentialsRequest on install
credReq := credreq.CredentialsRequestTemplate
credReq.Spec.CloudTokenPath = webIdentityTokenPath
```

```
c := mgr.GetClient()
if err := c.Create(context.TODO(), credReq); err != nil {
  if !errors.IsAlreadyExists(err) {
    setupLog.Error(err, "unable to create CredRequest")
    os.Exit(1)
  }
}
```

d. 确保 Operator 可以等待 **Secret** 对象从 CCO 显示，如下例所示，以及您在 Operator 中协调的其他项目：

例 5.5. 等待 Secret 对象示例

```
// WaitForSecret is a function that takes a Kubernetes client, a namespace, and a v1
"k8s.io/api/core/v1" name as arguments
// It waits until the secret object with the given name exists in the given namespace
// It returns the secret object or an error if the timeout is exceeded
func WaitForSecret(client kubernetes.Interface, namespace, name string)
(*v1.Secret, error) {
  // set a timeout of 10 minutes
  timeout := time.After(10 * time.Minute) 1

  // set a polling interval of 10 seconds
  ticker := time.NewTicker(10 * time.Second)

  // loop until the timeout or the secret is found
  for {
    select {
    case <-timeout:
      // timeout is exceeded, return an error
      return nil, fmt.Errorf("timed out waiting for secret %s in namespace %s", name,
namespace)
        // add to this error with a pointer to instructions for following a manual path to a
Secret that will work on STS
    case <-ticker.C:
      // polling interval is reached, try to get the secret
      secret, err := client.CoreV1().Secrets(namespace).Get(context.Background(),
name, metav1.GetOptions{})
      if err != nil {
        if errors.IsNotFound(err) {
          // secret does not exist yet, continue waiting
          continue
        } else {
          // some other error occurred, return it
          return nil, err
        }
      } else {
        // secret is found, return it
        return secret, nil
      }
    }
  }
}
```

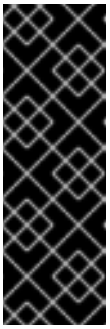
1 **timeout** 值基于 CCO 检测添加的 **CredentialsRequest** 对象并生成 **Secret** 对象的速度。您可能会考虑降低时间或为集群管理员创建自定义反馈，这可能会导致

e. 通过从凭证请求中读取 CCO 创建的 secret 并设置 AWS 配置，并创建包含该 secret 数据的 AWS 配置文件：

例 5.6. AWS 配置创建示例

```
func SharedCredentialsFileFromSecret(secret *corev1.Secret) (string, error) {
  var data []byte
  switch {
  case len(secret.Data["credentials"]) > 0:
     data = secret.Data["credentials"]
  default:
     return "", errors.New("invalid secret for aws credentials")
  }


  f, err := ioutil.TempFile("", "aws-shared-credentials")
  if err != nil {
     return "", errors.Wrap(err, "failed to create file for shared credentials")
  }
  defer f.Close()
  if _, err := f.Write(data); err != nil {
     return "", errors.Wrapf(err, "failed to write credentials to %s", f.Name())
  }
  return f.Name(), nil
}
```



重要

secret 被假定为存在，但在使用此 secret 时，您的 Operator 代码应等待和重试，以提供 CCO 创建 secret 的时间。

另外，等待周期最终应该超时，并警告用户 OpenShift Container Platform 集群版本，因此 CCO 可能会是一个较早的版本，它不支持使用 STS 检测的 **CredentialsRequest** 对象工作流。在这种情况下，指示用户必须使用其他方法添加 secret。

f. 配置 AWS SDK 会话，例如：

例 5.7. AWS SDK 会话配置示例

```
sharedCredentialsFile, err := SharedCredentialsFileFromSecret(secret)
if err != nil {
  // handle error
}
options := session.Options{
  SharedConfigState: session.SharedConfigEnable,
  SharedConfigFiles: []string{sharedCredentialsFile},
}
```

5.1.2.2. 角色规格

Operator 描述应包含在安装前创建的角色的具体信息，最好是管理员可以运行的脚本的形式。例如：

例 5.8. 角色创建脚本示例

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

AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
OIDC_PROVIDER=$(oc get authentication cluster -ojson | jq -r .spec.serviceAccountIssuer | sed -
e "s/^https:\/\///")
NAMESPACE=my-namespace
SERVICE_ACCOUNT_NAME="my-service-account"
POLICY_ARN_STRINGS="arn:aws:iam::aws:policy/AmazonS3FullAccess"


read -r -d '' TRUST_RELATIONSHIP <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": {
       "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDC_PROVIDER}"
     },
     "Action": "sts:AssumeRoleWithWebIdentity",
     "Condition": {
       "StringEquals": {
         "${OIDC_PROVIDER}:sub":
"system:serviceaccount:${NAMESPACE}:${SERVICE_ACCOUNT_NAME}"
       }
     }
   }
 ]
}
EOF

echo "${TRUST_RELATIONSHIP}" > trust.json

aws iam create-role --role-name "$SERVICE_ACCOUNT_NAME" --assume-role-policy-document
file://trust.json --description "role for demo"

while IFS= read -r POLICY_ARN; do
   echo -n "Attaching $POLICY_ARN ... "
   aws iam attach-role-policy \
       --role-name "$SERVICE_ACCOUNT_NAME" \
       --policy-arn "${POLICY_ARN}"
   echo "ok."
done <<< "$POLICY_ARN_STRINGS"
```

5.1.2.3. 故障排除

5.1.2.3.1. 身份验证失败

如果身份验证不成功，请确保您可以使用提供给 Operator 的令牌假设具有 Web 身份的角色。

流程

1. 从 pod 中提取令牌：

```
$ oc exec operator-pod -n <namespace_name> \
    -- cat /var/run/secrets/openshift/serviceaccount/token
```

2. 从 pod 中提取角色 ARN：

```
$ oc exec operator-pod -n <namespace_name> \
    -- cat /<path>/<to>/<secret_name> 1
```

1 不要将 root 用于路径。

3. 尝试使用 Web 身份令牌假定角色：

```
$ aws sts assume-role-with-web-identity \
    --role-arn $ROLEARN \
    --role-session-name <session_name> \
    --web-identity-token $TOKEN
```

5.1.2.3.2. Secret 无法正确挂载

以非 root 用户身份运行的 Pod 无法写入默认存在 AWS 共享凭证文件的 **/root** 目录。如果 secret 没有正确挂载到 AWS 凭证文件路径，请考虑将 secret 挂载到不同的位置，并在 AWS SDK 中启用共享凭证文件选项。

5.1.2.4. 其它方法

作为 Operator 作者的替代方法，您可以指定用户在安装 Operator 前负责为 Cloud Credential Operator (CCO) 创建 **CredentialsRequest** 对象。

Operator 指令必须向用户指示以下内容：

- 通过在说明中内联提供 YAML 或将用户指向下载位置，提供 **CredentialsRequest** 对象的 YAML 版本
- 指示用户创建 **CredentialsRequest** 对象

在 OpenShift Container Platform 4.14 及更高版本中，当 **CredentialsRequest** 对象出现在添加了适当 STS 信息的集群上后，Operator 可以读取 CCO 生成的 **Secret** 或挂载它，并在集群服务版本(CSV)中定义挂载。

对于早期版本的 OpenShift Container Platform，Operator 指令还必须向用户指示以下内容：

- 使用 CCO 实用程序(**ccoctl**)从 **CredentialsRequest** 对象生成 **Secret** YAML 对象
- 将 **Secret** 对象应用到适当的命名空间中的集群

Operator 仍然必须能够使用生成的 secret 与云 API 通信。因为在这种情况下，用户会在安装 Operator 前创建 secret，所以 Operator 可以执行以下操作之一：

- 在 CSV 中的 **Deployment** 对象中定义显式挂载
- 从 API 服务器以编程方式读取 **Secret** 对象，如推荐的"启用 Operator 以支持使用 AWS STS 的基于 CCO 的工作流"方法所示

5.1.3. 使用 Microsoft Entra Workload ID 为 OLM 管理的 Operator 基于 CCO 的工作流

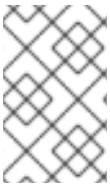
当在 Azure 上运行的 OpenShift Container Platform 集群处于 **Workload Identity / Federated Identity** 模式时，这意味着集群会利用 Azure 和 OpenShift Container Platform 的功能在应用程序级别应用*用户分配的管理的身份* 或 Microsoft Entra Workload ID 中的 *app 注册*。

Cloud Credential Operator (CCO) 是在云供应商上运行的 OpenShift Container Platform 集群中默认安装的集群 Operator。从 OpenShift Container Platform 4.14.8 开始，CCO 支持使用 Workload ID 的 OLM 管理的 Operator 工作流。

对于 Workload ID，CCO 提供以下功能：

- 检测在启用了 Workload ID 的集群中运行的时间
- 检查 **CredentialsRequest** 对象是否有字段，它们提供授予 Operator 对 Azure 资源的访问权限所需的信息

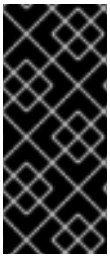
CCO 可以通过扩展 **CredentialsRequest** 对象来自动处理这个过程，该对象可以请求创建包含 Workload ID 工作流所需的信息的 **Secret**。



注意

不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

作为 Operator 作者准备一个 Operator 以用于 OpenShift Container Platform 4.14 及之后的版本中的更新的 CCO，您应该指示用户并添加代码来处理早期 CCO 版本的划分，除了处理 Workload ID token 身份验证（如果您的 Operator 还没有启用）。推荐的方法是使用正确的 Workload ID 相关字段提供 **CredentialsRequest** 对象，并让 CCO 为您创建 **Secret** 对象。



重要

如果您计划支持早于版本 4.14 的 OpenShift Container Platform 集群，请考虑为用户提供有关如何使用 CCO 实用程序(**ccoctl**)手动创建带有 Workload ID 启用信息的 secret 的说明。早期 CCO 版本不知道集群中的 Workload ID 模式，且无法为您创建 secret。

您的代码应检查永远不会出现的 secret，并警告用户以遵循您提供的回退指令。

使用 Workload ID 进行身份验证需要以下信息：

- **azure_client_id**
- **azure_tenant_id**
- **azure_region**

- **azure_subscription_id**
- **azure_federated_token_file**

Web 控制台中的 **Install Operator** 页面允许集群管理员在安装时提供此信息。然后，此信息会作为 Operator pod 上的环境变量传播到 **Subscription** 对象中。

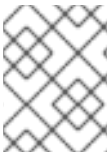
其他资源

- OLM 管理的 Operator 支持使用 Microsoft Entra Workload ID 进行身份验证
- 使用 Web 控制台从 OperatorHub 安装
- 使用 CLI 从 OperatorHub 安装

5.1.3.1. 启用 Operator 以支持使用 Microsoft Entra Workload ID 的基于 CCO 的工作流

作为 Operator 作者设计在 Operator Lifecycle Manager (OLM) 上运行的项目，您可以通过自定义项目来支持 Cloud Credential Operator (CCO)，使 Operator 能够对启用了 Microsoft Entra Workload ID 的 OpenShift Container Platform 集群进行身份验证。

使用此方法，Operator 负责创建 **CredentialsRequest** 对象并读取生成的 **Secret** 对象，并需要 RBAC 权限。



注意

默认情况下，与 Operator 部署相关的 pod 会挂载 **serviceAccountToken** 卷，以便在生成的 **Secret** 对象中引用服务帐户令牌。

先决条件

- OpenShift Container Platform 4.14 或更高版本
- 处于 Workload ID 模式的集群
- 基于 OLM 的 Operator 项目

流程

1. 更新 Operator 项目的 **ClusterServiceVersion** (CSV)对象：
   a. 确保 Operator 有 RBAC 权限来创建 **CredentialsRequests** 对象：

例 5.9. clusterPermissions 列表示例

```
# ...
install:
  spec:
    clusterPermissions:
    - rules:
      - apiGroups:
        - "cloudcredential.openshift.io"
        resources:
        - credentialsrequests
        verbs:
```

```
    - create
    - delete
    - get
    - list
    - patch
    - update
    - watch
```

b. 添加以下注解来声明对使用 Workload ID 的基于 CCO 工作流的方法的支持：

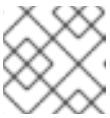
```
# ...
metadata:
 annotations:
   features.operators.openshift.io/token-auth-azure: "true"
```

2. 更新 Operator 项目代码：

a. 从由 **Subscription** 对象设置的环境变量中获取客户端 ID、租户 ID 和订阅 ID。例如：

```
// Get ENV var
clientID := os.Getenv("CLIENTID")
tenantID := os.Getenv("TENANTID")
subscriptionID := os.Getenv("SUBSCRIPTIONID")
azureFederatedTokenFile := "/var/run/secrets/openshift/serviceaccount/token"
```

b. 确保具有 **CredentialsRequest** 对象已准备好修补并应用。



注意

目前不支持在 Operator 捆绑包中添加 **CredentialsRequest** 对象。

c. 在凭证请求中添加 Azure 凭证信息和 Web 身份令牌路径，并在 Operator 初始化过程中应用它：

例 5.10. 在 Operator 初始化过程中应用 CredentialsRequest 对象示例

```
// apply CredentialsRequest on install
credReqTemplate.Spec.AzureProviderSpec.AzureClientID = clientID
credReqTemplate.Spec.AzureProviderSpec.AzureTenantID = tenantID
credReqTemplate.Spec.AzureProviderSpec.AzureRegion = "centralus"
credReqTemplate.Spec.AzureProviderSpec.AzureSubscriptionID = subscriptionID
credReqTemplate.CloudTokenPath = azureFederatedTokenFile

c := mgr.GetClient()
if err := c.Create(context.TODO(), credReq); err != nil {
    if !errors.IsAlreadyExists(err) {
        setupLog.Error(err, "unable to create CredRequest")
        os.Exit(1)
    }
}
```

d. 确保 Operator 可以等待 **Secret** 对象从 CCO 显示，如下例所示，以及您在 Operator 中协调的其他项目：

例 5.11. 等待 Secret 对象示例

```
// WaitForSecret is a function that takes a Kubernetes client, a namespace, and a v1
"k8s.io/api/core/v1" name as arguments
// It waits until the secret object with the given name exists in the given namespace
// It returns the secret object or an error if the timeout is exceeded
func WaitForSecret(client kubernetes.Interface, namespace, name string)
(*v1.Secret, error) {
  // set a timeout of 10 minutes
  timeout := time.After(10 * time.Minute) 1

  // set a polling interval of 10 seconds
  ticker := time.NewTicker(10 * time.Second)

  // loop until the timeout or the secret is found
  for {
    select {
    case <-timeout:
      // timeout is exceeded, return an error
      return nil, fmt.Errorf("timed out waiting for secret %s in namespace %s", name,
namespace)
        // add to this error with a pointer to instructions for following a manual path to a
Secret that will work on STS
    case <-ticker.C:
      // polling interval is reached, try to get the secret
      secret, err := client.CoreV1().Secrets(namespace).Get(context.Background(),
name, metav1.GetOptions{})
      if err != nil {
        if errors.IsNotFound(err) {
          // secret does not exist yet, continue waiting
          continue
        } else {
          // some other error occurred, return it
          return nil, err
        }
      } else {
        // secret is found, return it
        return secret, nil
      }
    }
  }
}
```

1 **timeout** 值基于 CCO 检测添加的 **CredentialsRequest** 对象并生成 **Secret** 对象的速度。您可能会考虑降低时间或为集群管理员创建自定义反馈，这可能会导致 Operator 尚未访问云资源的原因。

e. 从 **CredentialsRequest** 对象读取 CCO 创建的 secret，以与 Azure 进行身份验证并接收必要的凭证。

5.1.4. 使用 GCP Workload Identity 的 OLM 管理的 Operator 基于 CCO 的工作流

当 Google Cloud 上运行的 OpenShift Container Platform 集群位于 **GCP Workload Identity / Federated Identity** 模式中时，这意味着集群使用 Google Cloud 和 OpenShift Container Platform 的功能，以便在应用程序级别的 GCP Workload Identity 中应用权限。

Cloud Credential Operator (CCO) 是在云供应商上运行的 OpenShift Container Platform 集群中默认安装的集群 Operator。从 OpenShift Container Platform 4.17 开始，CCO 支持使用 GCP Workload Identity 的 OLM 管理的 Operator 的工作流。

对于 GCP Workload Identity，CCO 提供以下功能：

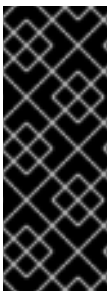
- 检测它在启用了 GCP Workload Identity 的集群中运行时
- 检查 **CredentialsRequest** 对象是否有提供授予 Operator 对 Google Cloud 资源访问权限所需的信息的字段

CCO 可以通过扩展使用 **CredentialsRequest** 对象来半自动化这个过程，这可以请求创建包含 GCP Workload Identity 工作流所需的信息的 **Secret**。

注意

不建议使用具有自动批准更新的订阅，因为在更新前可能会进行权限更改。具有手动批准更新的订阅可确保管理员有机会验证后续版本的权限，执行任何必要的步骤，然后进行更新。

作为 Operator 作者准备一个 Operator 以用于 OpenShift Container Platform 4.17 及之后的版本中的更新的 CCO，您应该指示用户并添加代码来处理早期 CCO 版本的划分，除了处理 GCP Workload Identity token 身份验证（如果您的 Operator 还没有启用）。推荐的方法是提供一个 **CredentialsRequest** 对象，带有正确填充的 GCP Workload Identity 字段，并让 CCO 为您创建 **Secret** 对象。



重要

如果您计划支持早于 4.17 的 OpenShift Container Platform 集群，请考虑为用户提供有关如何使用 CCO 实用程序(**ccoctl**)手动创建带有 GCP Workload Identity-enabling 信息的 secret 的说明。早期 CCO 版本不知道集群中的 GCP Workload Identity 模式，且无法为您创建 secret。

您的代码应检查永远不会出现的 secret，并警告用户以遵循您提供的回退指令。

要通过 Google Cloud Platform Workload Identity 使用简短令牌与 Google Cloud 进行身份验证，Operator 必须提供以下信息：

AUDIENCE

当管理员设置 GCP Workload Identity 时，在 Google Cloud 中创建，**AUDIENCE** 值必须是以下格式的 URL：

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

SERVICE_ACCOUNT_EMAIL

SERVICE_ACCOUNT_EMAIL 值是在 Operator 操作过程中模拟的 Google Cloud 服务帐户电子邮件，例如：

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

Web 控制台中的 **Install Operator** 页面允许集群管理员在安装时提供此信息。然后，此信息会作为 Operator pod 上的环境变量传播到 **Subscription** 对象中。

其他资源

- OLM 管理的 Operator 支持使用 GCP Workload Identity 进行身份验证
- 使用 Web 控制台从 OperatorHub 安装
- 使用 CLI 从 OperatorHub 安装

5.1.4.1. 启用 Operator 以支持使用 GCP Workload Identity 的基于 CCO 的工作流

作为 Operator 作者设计在 Operator Lifecycle Manager (OLM) 上运行的项目，您可以通过自定义项目来支持 Cloud Credential Operator (CCO)，使 Operator 能够对启用了 Google Cloud Platform Workload Identity-enabled OpenShift Container Platform 集群进行身份验证。

使用此方法，Operator 负责创建 **CredentialsRequest** 对象并读取生成的 **Secret** 对象，并需要 RBAC 权限。

注意

默认情况下，与 Operator 部署相关的 pod 会挂载 **serviceAccountToken** 卷，以便在生成的 **Secret** 对象中引用服务帐户令牌。

先决条件

- OpenShift Container Platform 4.17 或更高版本
- 在 **GCP Workload Identity / Federated Identity**模式中的集群
- 基于 OLM 的 Operator 项目

流程

1. 更新 Operator 项目的 **ClusterServiceVersion** (CSV)对象：
   a. 确保 CSV 中的 Operator 部署有以下 **volumeMounts** 和 **volumes** 字段，以便 Operator 可以假定角色具有 Web 身份：

例 5.12. volumeMounts 和 volumes 字段示例

```
# ...
    volumeMounts:

    - name: bound-sa-token
      mountPath: /var/run/secrets/openshift/serviceaccount
      readOnly: true
    volumes:
      # This service account token can be used to provide identity outside the cluster.
      - name: bound-sa-token
        projected:
          sources:
```

```
      - serviceAccountToken:
        path: token
        audience: openshift
```

b. 确保 Operator 有 RBAC 权限来创建 **CredentialsRequests** 对象：

例 5.13. clusterPermissions 列表示例

```
# ...
install:
  spec:
    clusterPermissions:
    - rules:
      - apiGroups:
        - "cloudcredential.openshift.io"
        resources:
        - credentialsrequests
        verbs:
        - create
        - delete
        - get
        - list
        - patch
        - update
        - watch
```

c. 添加以下注解来声明对使用 GCP Workload Identity 的基于 CCO 工作流的方法的支持：

```
# ...
metadata:
 annotations:
   features.operators.openshift.io/token-auth-gcp: "true"
```

2. 更新 Operator 项目代码：

a. 从由订阅配置在 pod 上设置的环境变量中获取 **audience** 和 **serviceAccountEmail** 值：

```
// Get ENV var
  audience := os.Getenv("AUDIENCE")
  serviceAccountEmail := os.Getenv("SERVICE_ACCOUNT_EMAIL")
  gcpIdentityTokenFile := "/var/run/secrets/openshift/serviceaccount/token"
```

b. 确保具有 **CredentialsRequest** 对象已准备好修补并应用。

注意

目前不支持在 Operator 捆绑包中添加 **CredentialsRequest** 对象。

c. 将 GCP Workload Identity 变量添加到凭证请求中，并在 Operator 初始化过程中应用它：

例 5.14. 在 Operator 初始化过程中应用 CredentialsRequest 对象示例

```
// apply CredentialsRequest on install
  credReqTemplate.Spec.GCPProviderSpec.Audience = audience
  credReqTemplate.Spec.GCPProviderSpec.ServiceAccountEmail =
serviceAccountEmail
  credReqTemplate.CloudTokenPath = gcpIdentityTokenFile


  c := mgr.GetClient()
  if err := c.Create(context.TODO(), credReq); err != nil {
     if !errors.IsAlreadyExists(err) {
        setupLog.Error(err, "unable to create CredRequest")
        os.Exit(1)
     }
  }
```

d. 确保 Operator 可以等待 **Secret** 对象从 CCO 显示，如下例所示，以及您在 Operator 中协调的其他项目：

例 5.15. 等待 Secret 对象示例

```
// WaitForSecret is a function that takes a Kubernetes client, a namespace, and a v1
"k8s.io/api/core/v1" name as arguments
// It waits until the secret object with the given name exists in the given namespace
// It returns the secret object or an error if the timeout is exceeded
func WaitForSecret(client kubernetes.Interface, namespace, name string)
(*v1.Secret, error) {
  // set a timeout of 10 minutes
  timeout := time.After(10 * time.Minute) (1)

  // set a polling interval of 10 seconds
  ticker := time.NewTicker(10 * time.Second)

  // loop until the timeout or the secret is found
  for {
    select {
    case <-timeout:
      // timeout is exceeded, return an error
      return nil, fmt.Errorf("timed out waiting for secret %s in namespace %s", name,
namespace)
// add to this error with a pointer to instructions for following a manual path to a Secret
that will work
    case <-ticker.C:
      // polling interval is reached, try to get the secret
      secret, err := client.CoreV1().Secrets(namespace).Get(context.Background(),
name, metav1.GetOptions{})
      if err != nil {
        if errors.IsNotFound(err) {
          // secret does not exist yet, continue waiting
          continue
        } else {
          // some other error occurred, return it
          return nil, err
        }
      } else {
        // secret is found, return it
```

```
            return secret, nil
        }
    }
}
}
```

1 **timeout** 值基于 CCO 检测添加的 **CredentialsRequest** 对象并生成 **Secret** 对象的速度。您可能会考虑降低时间或为集群管理员创建自定义反馈，这可能会导致 Operator 尚未访问云资源的原因。

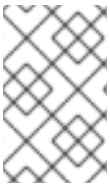
e. 从 secret 中读取 **service_account.json** 字段，并使用它来验证您的 Google Cloud 客户端：

```
service_account_json := secret.StringData["service_account.json"]
```

第 6 章 集群 OPERATOR 参考

本指南对红帽提供的集群 *Operator* 进行了索引，该 Operator 充当 OpenShift Container Platform 的架构基础。默认情况下会安装集群 Operator（除非另有说明），并由 Cluster Version Operator (CVO) 管理。如需有关 control plane 架构的更多信息，请参阅 OpenShift Container Platform 中的 Operator。

集群管理员可以通过 **Administration → Cluster Settings** 页面在 OpenShift Container Platform Web 控制台中查看集群 Operator。

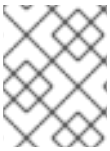


注意

Cluster Operator 不由 Operator Lifecycle Manager (OLM) 和 OperatorHub 管理。OLM 和 OperatorHub 是 OpenShift Container Platform 中用于安装和运行可选附加组件 Operator 的 Operator Framework 的一部分。

安装前可以禁用以下一些集群 Operator。如需更多信息，请参阅集群功能。

6.1. CLUSTER BAREMETAL OPERATOR



注意

Cluster Baremetal Operator 是一个可选的集群功能，可在安装过程中由集群管理员禁用。有关可选集群功能的更多信息，请参阅 *安装*中的"集群功能"。

Cluster Baremetal Operator (CBO) 会部署使裸机服务器成为一个可完全正常工作的节点以运行 OpenShift Container Platform 计算节点所需的所有组件。CBO 确保 metal3 部署（由 Bare Metal Operator (BMO) 和 Ironic 容器组成）在 OpenShift Container Platform 集群内的一个 control plane 节点上运行。CBO 还会侦听 OpenShift Container Platform 对资源的更新，它会监视并采取适当的操作。

6.1.1. 项目

cluster-baremetal-operator

其他资源

- 裸机功能

6.2. CLOUD CREDENTIAL OPERATOR

Cloud Credential Operator(CCO)将云供应商凭证作为 Kubernetes 自定义资源定义(CRD)进行管理。**CredentialsRequest** 自定义资源（CR）的 CCO 同步，允许 OpenShift Container Platform 组件使用集群运行所需的特定权限请求云供应商凭证。

通过在 **install-config.yaml** 文件中为 **credentialsMode** 参数设置不同的值，可将 CCO 配置为以几种不同模式操作。如果没有指定模式，或将 **credentialsMode** 参数被设置为空字符串（""）。

6.2.1. 项目

openshift-cloud-credential-operator

6.2.2. CRD

- **credentialsrequests.cloudcredential.openshift.io**
  - Scope: Namespaced
  - CR：**CredentialsRequest**
  - Validation: Yes

6.2.3. Configuration objects

不需要配置。

6.2.4. 其他资源

- 关于 Cloud Credential Operator
- **CredentialsRequest** 自定义资源

6.3. CLUSTER AUTHENTICATION OPERATOR

Cluster Authentication Operator 在集群中安装并维护 **Authentication** 自定义资源，并可以通过以下方式查看：

```
$ oc get clusteroperator authentication -o yaml
```

6.3.1. project

cluster-authentication-operator

6.4. CLUSTER AUTOSCALER OPERATOR

Cluster Autoscaler Operator 使用 **cluster-api** 供应商管理 OpenShift Cluster Autoscaler 的部署。

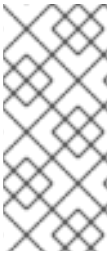
6.4.1. project

cluster-autoscaler-operator

6.4.2. CRD

- **ClusterAutoscaler**：这是一个单一的资源，用于控制集群自动扩展实例的配置。Operator 只响应受管命名空间中名为 **default** 的 **ClusterAutoscaler** 资源，即 **WATCH_NAMESPACE** 环境变量的值。
- **MachineAutoscaler**：此资源针对一个节点组，并管理注解来为那个组启用和配置自动扩展，**min** 和 **max** 的值。目前只能将 **MachineSet** 对象作为目标。

6.5. CLOUD CONTROLLER MANAGER OPERATOR



注意

对于 Amazon Web Services (AWS), Google Cloud, IBM Cloud®, global Microsoft Azure, Microsoft Azure Stack Hub, Nutanix, Red Hat OpenStack Platform (RHOSP), 和 VMware vSphere，这个 Operator 的状态是正式发布（GA）。

对于 IBM Power® Virtual Server，Operator 作为技术预览提供

Cloud Controller Manager Operator 管理并更新在 OpenShift Container Platform 上部署的云控制器管理器。Operator 基于 Kubebuilder 框架和 **controller-runtime** 库。您可以使用 Cluster Version Operator (CVO) 安装 Cloud Controller Manager Operator。

Cloud Controller Manager Operator 包括以下组件：

- Operator
- 云配置观察

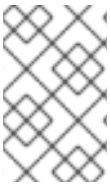
默认情况下，Operator 通过 **metrics** 服务公开 Prometheus 指标数据。

6.5.1. 项目

cluster-cloud-controller-manager-operator

6.6. CLUSTER CAPI OPERATOR

Cluster CAPI Operator 维护 Cluster API 资源的生命周期。此 Operator 负责在 OpenShift Container Platform 集群中部署 Cluster API 项目的所有管理任务。



注意

对于 Amazon Web Services (AWS)、Google Cloud、Microsoft Azure、Red Hat OpenStack Platform (RHOSP) 和 VMware vSphere 集群，这个 Operator 作为技术预览提供。

6.6.1. 项目

cluster-capi-operator

6.6.2. CRD

- **awsmachines.infrastructure.cluster.x-k8s.io**
  - Scope: Namespaced
  - CR: **awsmachine**
- **gcpmachines.infrastructure.cluster.x-k8s.io**
  - Scope: Namespaced
  - CR: **gcpmachine**
- **azuremachines.infrastructure.cluster.x-k8s.io**

    - Scope: Namespaced
    - CR: **azuremachine**
- **openstackmachines.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **openstackmachine**
- **vspheremachines.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **vspheremachine**
- **metal3machines.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **metal3machine**
- **awsmachinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **awsmachinetemplate**
- **gcpmachinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **gcpmachinetemplate**
- **azuremachinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **azuremachinetemplate**
- **openstackmachinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **openstackmachinetemplate**
- **vspheremachinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **vspheremachinetemplate**
- **metal3machinetemplates.infrastructure.cluster.x-k8s.io**
    - Scope: Namespaced
    - CR: **metal3machinetemplate**

6.7. CLUSTER CONFIG OPERATOR

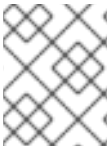
Cluster Config Operator 执行与 **config.openshift.io** 相关的以下任务：

- 创建 CRD。
- 呈现初始自定义资源。
- 处理迁移。

6.7.1. 项目

cluster-config-operator

6.8. CLUSTER CSI SNAPSHOT CONTROLLER OPERATOR



注意

Cluster CSI Snapshot Controller Operator 是一个可选集群功能，集群管理员可在安装过程中禁用。有关可选集群功能的更多信息，请参阅*安装*中的"集群功能"。

Cluster CSI Snapshot Controller Operator 安装和维护 CSI Snapshot Controller。CSI Snapshot Controller 负责监视 **VolumeSnapshot** CRD 对象，并管理卷快照的创建和删除生命周期。

6.8.1. 项目

cluster-csi-snapshot-controller-operator

其他资源

- CSI 快照控制器功能

6.9. CLUSTER IMAGE REGISTRY OPERATOR

Cluster Image Registry Operator 管理 OpenShift 镜像 registry 的单个实例。它管理 registry 的所有配置，包括创建存储。

在初始启动时，Operator 会基于集群中检测到的配置创建默认的 **image-registry** 资源实例。这代表了根据云供应商要使用的云存储类型。

如果没有足够的信息来定义完整的 **image-registry** 资源，则会定义一个不完整的资源，Operator 将更新资源状态以提供缺失的内容。

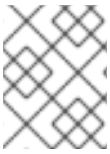
Cluster Image Registry Operator在**openshift-image-registry**命名空间中运行，并管理该位置中的 registry 实例。registry的所有配置和工作负载资源都位于该命名空间中。

6.9.1. project

cluster-image-registry-operator

6.10. CLUSTER MACHINE APPROVER OPERATOR

Cluster Machine Approver Operator 在集群安装后自动批准为新 worker 节点请求的 CSR。



注意

对于 control plane 节点，bootstrap 节点上的 **approve-csr** 服务会在集群引导阶段自动批准所有 CSR。

6.10.1. 项目

cluster-machine-approver-operator

6.11. CLUSTER MONITORING OPERATOR

Cluster Monitoring Operator (CMO) 管理并更新 OpenShift Container Platform 上部署的基于 Prometheus 的集群监控堆栈。

项目
openshift-monitoring

CRD

- **alertmanagers.monitoring.coreos.com**
  - Scope: Namespaced
  - CR: **alertmanager**
  - Validation: Yes
- **prometheuses.monitoring.coreos.com**
  - Scope: Namespaced
  - CR: **prometheus**
  - Validation: Yes
- **prometheusrules.monitoring.coreos.com**
  - Scope: Namespaced
  - CR: **prometheusrule**
  - Validation: Yes
- **servicemonitors.monitoring.coreos.com**
  - Scope: Namespaced
  - CR: **servicemonitor**
  - Validation: Yes

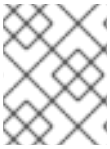
Configuration objects

```
$ oc -n openshift-monitoring edit cm cluster-monitoring-config
```

6.12. CLUSTER NETWORK OPERATOR

Cluster Network Operator 在 OpenShift Container Platform 集群上安装并升级网络组件。

6.13. CLUSTER SAMPLES OPERATOR



注意

Cluster Samples Operator 是一个可选集群功能，集群管理员可在安装过程中禁用。有关可选集群功能的更多信息，请参阅 *安装*中的"集群功能"。

Cluster Samples Operator 管理存储在 **openshift** 命名空间中的示例镜像流和模板。

在初始启动时，Operator 会创建默认样本配置资源来启动镜像流和模板的创建。配置对象是一个集群范围内的对象，它带有一个键 **cluster** 和类型 **configs.samples**。

镜像流是基于 Red Hat Enterprise Linux CoreOS (RHCOS) 的 OpenShift Container Platform 镜像流，指向 **registry.redhat.io** 上的镜像。同样，模板也被归类为 OpenShift Container Platform 模板。

Cluster Samples Operator 部署包含在 **openshift-cluster-samples-operator** 命名空间中。开始时，OpenShift 镜像 registry 中的镜像流导入逻辑和 API 服务器会使用安装 pull secret 与 **registry.redhat.io** 进行身份验证。如果管理员更改了用于示例镜像流的 registry，则管理员可在 **openshift** 命名空间中创建额外的 secret。如果创建，这些 secret 包含用于简化镜像导入所需的 **docker** 的 **config.json** 的内容。

Cluster Samples Operator 的镜像包含关联的 OpenShift Container Platform 发行版本的镜像流和模板定义。Cluster Samples Operator 创建示例后，它会添加一个注解，表示其兼容的 OpenShift Container Platform 版本。Operator 使用此注解来确保每个示例与兼容发行版本匹配。清单（inventory）以外的示例会与跳过的示例一样被忽略。

只要版本注解没有修改或删除，则允许对 Operator 管理的任何样本进行修改。但是，在升级中，当版本注解改变时，这些修改可能会被替换，因为样本会使用更新的版本进行更新。Jenkins 镜像是安装后镜像有效负载的一部分，并直接标记到镜像流中。

Samples Operator 配置资源包含一个终结器（finalizer），它会在删除时清除以下内容：

- Operator 管理的镜像流
- Operator 管理的模板
- Operator 生成的配置资源
- 集群状态资源

删除样本资源后，Samples Operator 会使用默认配置重新创建资源。

6.13.1. project

cluster-samples-operator

其他资源

- OpenShift 示例功能

6.14. CLUSTER STORAGE OPERATOR

注意

Cluster Storage Operator 是一个可选集群功能，集群管理员可在安装过程中禁用。有关可选集群功能的更多信息，请参阅*安装*中的"集群功能"。

Cluster Storage Operator 设置 OpenShift Container Platform 集群范围内的存储默认设置。它确保了 OpenShift Container Platform 集群存在默认**存储类**。它还安装 Container Storage Interface (CSI) 驱动程序，使集群能够使用各种存储后端。

6.14.1. 项目

cluster-storage-operator

6.14.2. Configuration

不需要配置。

6.14.3. 备注

- Operator 创建的存储类可以通过编辑其注解来实现非默认设置，但只要 Operator 运行，这个存储类就无法被删除。

其他资源

- 存储功能

6.15. CLUSTER VERSION OPERATOR

集群 Operator 管理集群功能的特定区域。Cluster Version Operator (CVO) 管理集群 Operator 的生命周期，其中许多默认安装在 OpenShift Container Platform 中。

CVO 还检查 OpenShift Update Service，以根据图中的当前组件版本和信息来查看有效的更新和更新路径，方法是收集集群版本及其集群 Operator 的状态。此状态包括条件类型，它告知您 OpenShift Container Platform 集群的健康状态和当前状态。

如需有关集群版本状况类型的更多信息，请参阅"了解集群版本状况类型"。

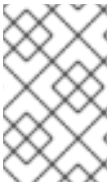
6.15.1. 项目

cluster-version-operator

其他资源

- 了解集群版本状况类型

6.16. CONSOLE OPERATOR



注意

Console Operator 是一个可选集群功能，集群管理员可在安装过程中禁用。如果您在安装时禁用了 Console Operator，您的集群仍被支持并可升级。有关可选集群功能的更多信息，请参阅 *安装*中的"集群功能"。

Console Operator 在集群中安装和维护 OpenShift Container Platform web 控制台。Console Operator 会被默认安装，并自动维护控制台。

6.16.1. 项目

console-operator

其他资源

- Web 控制台功能

6.17. CONTROL PLANE MACHINE SET OPERATOR

Control Plane Machine Set Operator 自动管理 OpenShift Container Platform 集群中的 control plane 机器资源。

注意

此 Operator 可用于 Amazon Web Services (AWS)、Google Cloud、Microsoft Azure、Nutanix 和 VMware vSphere。

6.17.1. 项目

cluster-control-plane-machine-set-operator

6.17.2. CRD

- **controlplanemachineset.machine.openshift.io**
  - Scope: Namespaced
  - CR: **ControlPlaneMachineSet**
  - Validation: Yes

6.17.3. 其他资源

- 关于 control plane 机器集
- **ControlPlaneMachineSet** 自定义资源

6.18. DNS OPERATOR

DNS Operator 部署并管理 CoreDNS，以为 pod 提供名称解析服务。它在 OpenShift Container Platform 中启用了基于 DNS 的 Kubernetes 服务发现。

Operator 根据集群的配置创建可正常工作的默认部署。

- 默认集群域是 **cluster.local**。
- 尚不支持配置 CoreDNS Corefile 或 Kubernetes 插件。

DNS Operator 把 CoreDNS 做为一个 Kubernetes 守护进程集进行管理。它会使用一个带有静态 IP 的服务向外界公开这个功能。CoreDNS 在集群中的所有节点上运行。

6.18.1. project

cluster-dns-operator

6.19. ETCD 集群 OPERATOR

etcd 集群 Operator 自动执行 etcd 集群扩展，启用 etcd 监控和指标，并简化灾难恢复流程。

6.19.1. project

cluster-etcd-operator

6.19.2. CRD

- **etcds.operator.openshift.io**
  - Scope: Cluster
  - CR: **etcd**
  - Validation: Yes

6.19.3. Configuration objects

```
$ oc edit etcd cluster
```

6.20. INGRESS OPERATOR

Ingress Operator 配置并管理 OpenShift Container Platform 路由。

6.20.1. 项目

openshift-ingress-operator

6.20.2. CRD

- **clusteringresses.ingress.openshift.io**
  - Scope: Namespaced
  - CR: **clusteringresses**
  - Validation: No

6.20.3. Configuration objects

- Cluster config
  - 类型名：**clusteringresses.ingress.openshift.io**
  - 实例名称： **default**
  - 查看命令：

```
$ oc get clusteringresses.ingress.openshift.io -n openshift-ingress-operator default -o yaml
```

6.20.4. 备注

Ingress Operator 在 **openshift-ingress** 项目中设置路由，并为路由创建部署：

```
$ oc get deployment -n openshift-ingress
```

Ingress Operator 使用来自 **network/cluster** 状态的 **clusterNetwork[].cidr** 来决定受管入口控制器（路由器）应该在其中操作的模式(IPv4、IPv6 或双堆栈）。例如，如果 **clusterNetwork** 只包含 v6 **cidr**，则 Ingress Controller 在只纯 IPv6 模式下运行。

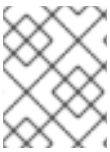
在以下示例中，Ingress Operator 管理的 ingress 控制器将以 IPv4 模式运行，因为只有一个集群网络存在，网络是 IPv4 **cidr**：

```
$ oc get network/cluster -o jsonpath='{.status.clusterNetwork[*]}'
```

输出示例

```
map[cidr:10.128.0.0/14 hostPrefix:23]
```

6.21. INSIGHTS OPERATOR



注意

Insights Operator 是一个可选集群功能，集群管理员可在安装过程中禁用。有关可选集群功能的更多信息，请参阅 *安装*中的"集群功能"。

Insights Operator 收集 OpenShift Container Platform 配置数据并将其发送到红帽。数据用于生成有关集群可能暴露的潜在问题的主动分析建议。这些 insights 通过 console.redhat.com 上的 Insights 公告服务与集群管理员进行交流。

6.21.1. 项目

insights-operator

6.21.2. 配置

不需要配置。

6.21.3. 备注

Insights Operator 补充 OpenShift Container Platform Telemetry。

其他资源

- Insights 功能
- 关于远程健康监控

6.22. KUBERNETES API SERVER OPERATOR

Kubernetes API Server Operator 管理并更新在 OpenShift Container Platform 上部署的 Kubernetes API 服务器。Operator 基于 OpenShift Container Platform **library-go** 框架，它与 Cluster Version Operator (CVO) 一起安装。

6.22.1. 项目

openshift-kube-apiserver-operator

6.22.2. CRD

- **kubeapiservers.operator.openshift.io**
  - Scope: Cluster
  - CR: **kubeapiserver**
  - Validation: Yes

6.22.3. Configuration objects

```
$ oc edit kubeapiserver
```

6.23. KUBERNETES CONTROLLER MANAGER OPERATOR

Kubernetes Controller Manager Operator 管理并更新在 OpenShift Container Platform 上部署的 Kubernetes Controller Manager。Operator 基于 OpenShift Container Platform **library-go** 框架，并通过 Cluster Version Operator (CVO) 安装。

它包含以下组件：

- Operator
- Bootstrap 清单解析器
- 基于静态 pod 的安装程序
- 配置观察

默认情况下，Operator 通过 **metrics** 服务公开 Prometheus 指标数据。

6.23.1. 项目

cluster-kube-controller-manager-operator

6.24. KUBERNETES SCHEDULER OPERATOR

Kubernetes Scheduler Operator 管理并更新在 OpenShift Container Platform 上部署的 Kubernetes 调度程序。Operator 基于 OpenShift Container Platform **library-go** 框架，它与 Cluster Version Operator (CVO) 一起安装。

Kubernetes Scheduler Operator 包含以下组件：

- Operator
- Bootstrap 清单解析器
- 基于静态 pod 的安装程序
- 配置观察

默认情况下，Operator 通过 metrics 服务公开 Prometheus 指标数据。

6.24.1. project

cluster-kube-scheduler-operator

6.24.2. Configuration

Kubernetes 调度程序的配置是以下合并的结果：

- 默认配置。
- 从 spec **schedulers.config.openshift.io** 获得的配置。

所有这些都是稀疏配置，无效的 JSON 片断会在结尾进行合并，以便形成有效的配置。

6.25. KUBERNETES STORAGE VERSION MIGRATOR OPERATOR

Kubernetes Storage Version Migrator Operator 检测到默认存储版本的更改，在存储版本更改时为资源类型创建迁移请求，并处理迁移请求。

6.25.1. 项目

cluster-kube-storage-version-migrator-operator

6.26. MACHINE API OPERATOR

Machine API Operator 管理用于扩展 Kubernetes API 的特定目的自定义资源定义（CRD）、控制器和 RBAC 对象的生命周期。它声明集群中机器的所需状态。

6.26.1. project

machine-api-operator

6.26.2. CRD

- **MachineSet**

- **机器**
- **MachineHealthCheck**

6.27. MACHINE CONFIG OPERATOR

Machine Congig Operator 管理并应用基本操作系统和容器运行时的配置和更新，包括内核和 kubelet 之间的所有配置和更新。

有四个组件：

- **machine-config-server**：为加入集群的新机器提供 Ignition 配置。
- **machine-config-controller**：协调机器升级到 **MachineConfig** 对象定义的配置。提供用来控制单独一组机器升级的选项。
- **machine-config-daemon**：在更新过程中应用新机器配置。验证并验证机器的状态到请求的机器配置。
- **machine-config**：提供安装、首次启动和更新一个机器的完整机器配置源。

重要

目前，不支持阻止或限制机器配置服务器端点。机器配置服务器必须公开给网络，以便新置备的机器没有现有配置或状态，才能获取其配置。在这个模型中，信任的根是证书签名请求 (CSR) 端点，即 kubelet 发送其证书签名请求以批准加入集群。因此，机器配置不应用于分发敏感信息，如 secret 和证书。

为确保机器配置服务器端点，端口 22623 和 22624 在裸机场景中是安全的，客户必须配置正确的网络策略。

6.27.1. 项目

openshift-machine-config-operator

6.28. MARKETPLACE OPERATOR

注意

Marketplace Operator 是一个可选集群功能，如果不需要，集群管理员可以禁用它。有关可选集群功能的更多信息，请参阅 *安装*中的"集群功能"。

Marketplace Operator 通过使用集群中的一组默认 Operator Lifecycle Manager (OLM) 目录简化了将非集群 Operator 引入集群的过程。安装 Marketplace Operator 时，它会创建 **openshift-marketplace** 命名空间。OLM 确保在 **openshift-marketplace** 命名空间中安装的目录源可用于集群中的所有命名空间。

6.28.1. project

operator-marketplace

其他资源

- Marketplace 功能

6.29. NODE TUNING OPERATOR

Node Tuning Operator 可以帮助您通过编排 TuneD 守护进程来管理节点级别的性能优化，并使用 Performance Profile 控制器获得低延迟性能。大多数高性能应用程序都需要一定程度的内核级性能优化。Node Tuning Operator 为用户提供了一个统一的、节点一级的 sysctl 管理接口，并可以根据具体用户的需要灵活地添加自定义性能优化设置。

Operator 将为 OpenShift Container Platform 容器化 TuneD 守护进程作为一个 Kubernetes 守护进程集进行管理。它保证了自定义性能优化设置以可被守护进程支持的格式传递到在集群中运行的所有容器化的 TuneD 守护进程中。相应的守护进程会在集群的所有节点上运行，每个节点上运行一个。

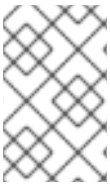
在发生触发配置集更改的事件时，或通过接收和处理终止信号安全终止容器化 TuneD 守护进程时，容器化 TuneD 守护进程所应用的节点级设置将被回滚。

Node Tuning Operator 使用 Performance Profile 控制器来实现自动性能优化，从而实现 OpenShift Container Platform 应用程序的低延迟性能。

集群管理员配置了性能配置集以定义节点级别的设置，例如：

- 将内核更新至 kernel-rt。
- 为内务选择 CPU。
- 为运行工作负载选择 CPU。

在版本 4.1 及更高版本中，OpenShift Container Platform 标准安装中包含了 Node Tuning Operator。



注意

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

6.29.1. 项目

cluster-node-tuning-operator

6.29.2. 其他资源

- 关于低延迟

6.30. OPENSHIFT API SERVER OPERATOR

OpenShift API Server Operator 在集群中安装和维护 **openshift-apiserver** 。

6.30.1. project

openshift-apiserver-operator

6.30.2. CRD

- **openshiftapiservers.operator.openshift.io**
  - Scope: Cluster

- CR: **openshiftapiserver**
- Validation: Yes

6.31. OPENSHIFT CONTROLLER MANAGER OPERATOR

OpenShift Controller Manager Operator 在集群中安装和维护 **OpenShiftControllerManager** 自定义资源，并可使用以下方法查看：

```
$ oc get clusteroperator openshift-controller-manager -o yaml
```

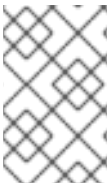
自定义资源（CRD） **openshiftcontrollermanagers.operator.openshift.io** 可以在具有以下内容的集群中查看：

```
$ oc get crd openshiftcontrollermanagers.operator.openshift.io -o yaml
```

6.31.1. 项目

cluster-openshift-controller-manager-operator

6.32. OPERATOR LIFECYCLE MANAGER (OLM) CLASSIC OPERATORS

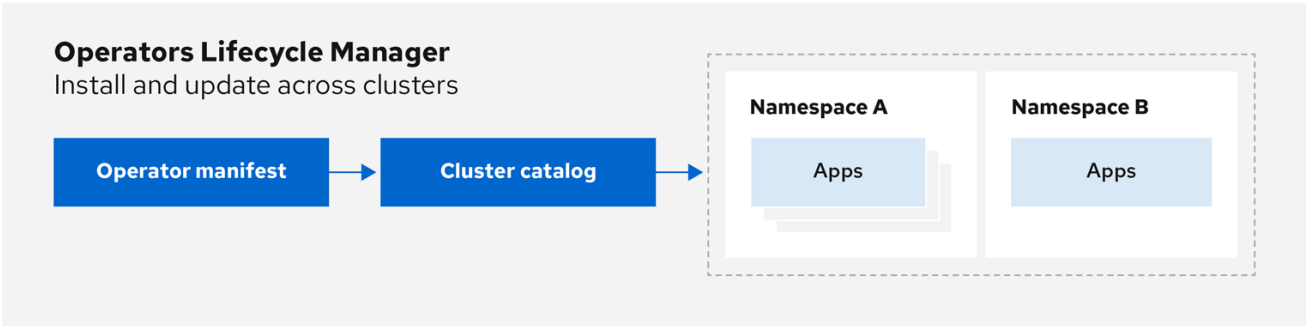


注意

以下章节与 OpenShift Container Platform 4 中包含的 Operator Lifecycle Manager (OLM) Classic 相关，自其初始发行版本起就包括在 OpenShift Container Platform 4 中。对于 OLM v1，请参阅 Operator Lifecycle Manager (OLM) v1 Operator。

Operator Lifecycle Manager (OLM) Classic 可帮助用户安装、更新和管理 Kubernetes 原生应用程序 (Operator)以及在 OpenShift Container Platform 集群中运行的关联服务的生命周期。它是 Operator Framework 的一部分，后者是一个开源工具包，用于以有效、自动化且可扩展的方式管理 Operator。

图 6.1. OLM (Classic) 工作流



OpenShift_43_1019

OLM 默认在 OpenShift Container Platform 4.19 中运行，辅助集群管理员对集群上运行的 Operator 进行安装、升级和授予访问权。OpenShift Container Platform Web 控制台提供一些管理界面，供集群管理员安装 Operator，以及为特定项目授权以便使用集群上的可用 Operator 目录。

开发人员通过自助服务体验，无需成为相关问题的专家也可自由置备和配置数据库、监控和大数据服务的实例，因为 Operator 已将相关知识融入其中。

6.32.1. OLM Operator

集群中存在 CSV 中指定需要的资源后，OLM Operator 将负责部署由 CSV 资源定义的应用程序。

OLM Operator 不负责创建所需资源；用户可选择使用 CLI 手动创建这些资源，也可选择使用 Catalog Operator 来创建这些资源。这种关注点分离的机制可以使得用户逐渐增加他们选择用于其应用程序的 OLM 框架量。

OLM Operator 使用以下工作流：

1. 观察命名空间中的集群服务版本（CSV），并检查是否满足要求。
2. 如果满足要求，请运行 CSV 的安装策略。

注意

CSV 必须是 Operator 组的活跃成员，才可运行该安装策略。

6.32.2. Catalog Operator

Catalog Operator 负责解析和安装集群服务版本（CSV）以及它们指定的所需资源。另外还负责监视频道中的目录源中是否有软件包更新，并将其升级（可选择自动）至最新可用版本。

要跟踪频道中的软件包，您可以创建一个 **Subscription** 对象来配置所需的软件包、频道和 **CatalogSource** 对象，以便拉取更新。在找到更新后，便会代表用户将一个适当的 **InstallPlan** 对象写入命名空间。

Catalog Operator 使用以下工作流：

1. 连接到集群中的每个目录源。
2. 监视是否有用户创建的未解析安装计划，如果有：
   a. 查找与请求名称相匹配的 CSV，并将此 CSC 添加为已解析的资源。
   b. 对于每个受管或所需 CRD，将其添加为已解析的资源。
   c. 对于每个所需 CRD，找到管理相应 CRD 的 CSV。
3. 监视是否有已解析的安装计划并为其创建已发现的所有资源（用户批准或自动）。
4. 观察目录源和订阅并根据它们创建安装计划。

6.32.3. Catalog Registry

Catalog Registry 存储 CSV 和 CRD 以便在集群中创建，并存储有关软件包和频道的元数据。

package manifest 是 Catalog Registry 中的一个条目，用于将软件包标识与 CSV 集相关联。在软件包中，频道指向特定 CSV。因为 CSV 明确引用了所替换的 CSV，软件包清单向 Catalog Operator 提供了将 CSV 更新至频道中最新版本所需的信息，逐步安装和替换每个中间版本。

6.32.4. CRD

OLM 和 Catalog Operator 负责管理作为 OLM 框架基础的自定义资源定义(CRD)：

表 6.1. 由 OLM 和 Catalog Operator 管理的 CRD

资源	短名称	所有者	描述
ClusterServiceVersion (CSV)	**csv**	OLM	应用程序元数据：名称、版本、图标、所需资源、安装等。
InstallPlan	**ip**	Catalog	为自动安装或升级 CSV 而需创建的资源的计算列表。
CatalogSource	**catsrc**	Catalog	定义应用程序的 CSV、CRD 和软件包存储库。
Subscription	**sub**	Catalog	用于通过跟踪软件包中的频道来保持 CSV 最新。
OperatorGroup	**og**	OLM	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自定义资源 (CR)。

每个 Operator 还负责创建以下资源：

表 6.2. 由 OLM 和 Catalog Operator 创建的资源

资源	所有者
部署	OLM
ServiceAccounts	
(Cluster)Roles	
(Cluster)RoleBindings	
CustomResourceDefinitions (CRD)	Catalog
ClusterServiceVersions	

6.32.5. Cluster Operators

在 OpenShift Container Platform 中，OLM 功能在一组集群 Operator 中提供：

operator-lifecycle-manager

提供 OLM Operator。另外，如果任何已安装的 Operator 会阻止集群升级，请通知集群管理员，根据其 **olm.maxOpenShiftVersion** 属性。如需更多信息，请参阅"控制与 OpenShift Container Platform 版本的 Operator 兼容性"。

operator-lifecycle-manager-catalog

提供 Catalog Operator。

operator-lifecycle-manager-packageserver

代表一个 API 扩展服务器，负责从集群中的所有目录收集元数据，并提供面向用户的 **PackageManifest** API。

6.32.6. 其他资源

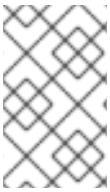
- 了解 Operator Lifecycle Manager (OLM)

6.33. OPERATOR LIFECYCLE MANAGER (OLM) V1 OPERATOR

从 OpenShift Container Platform 4.18 开始，OLM v1 与 OLM (Classic) 一起默认启用。这个下一代迭代提供了一个更新的框架，它改变了许多 OLM (Classic) 概念，使集群管理员能够为其用户扩展功能。

OLM v1 管理新 **ClusterExtension** 对象的生命周期，其中包括通过 **registry+v1** 捆绑包格式的 Operator，并控制集群中扩展的安装、升级和基于角色的访问控制(RBAC)。

在 OpenShift Container Platform 中，OLM v1 由 **olm** cluster Operator 提供。



注意

olm cluster Operator 会通知集群管理员，如果任何安装的扩展会阻止集群升级，根据其 **olm.maxOpenShiftVersion** 属性。如需更多信息，请参阅"与 OpenShift Container Platform 版本保持一致"。

6.33.1. 组件

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

Operator 控制器

OLM v1 的核心组件，使用 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

一个 Kubernetes 扩展，它解包基于文件的目录(FBC)内容，并在容器镜像中提供，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。

6.33.2. CRD

- **clusterextension.olm.operatorframework.io**
  - Scope: Cluster
  - CR: **ClusterExtension**
- **clustercatalog.olm.operatorframework.io**
  - Scope: Cluster
  - CR: **ClusterCatalog**

6.33.3. 项目

- operator-framework/operator-controller
- operator-framework/catalogd

6.33.4. 其他资源

- 扩展概述
- 与 OpenShift Container Platform 版本的兼容性

6.34. OPENSHIFT SERVICE CA OPERATOR

OpenShift Service CA Operator mint 并管理 Kubernetes 服务的服务证书。

6.34.1. 项目

openshift-service-ca-operator

6.35. VSPHERE 问题检测器（VSPHERE PROBLEM DETECTOR）OPERATOR

vSphere 问题检测器 Operator 会检查在 vSphere 上部署的集群，以获取与存储相关的常见安装和错误配置问题。

注意

只有 Cluster Storage Operator 检测到集群部署在 vSphere 上时，Cluster Storage Operator 才会启动 vSphere 问题检测器 Operator。

6.35.1. 配置

不需要配置。

6.35.2. 备注

- Operator 支持 vSphere 上的 OpenShift Container Platform 安装。
- Operator 使用 **vsphere-cloud-credentials** 与 vSphere 通信。
- Operator 会执行与存储相关的检查。

其他资源

- 使用 vSphere 问题检测器 Operator

第 7 章 OLM V1

7.1. 关于 OPERATOR LIFECYCLE MANAGER V1

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 4.18 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。

从 OpenShift Container Platform 4.17 开始，OLM v1 的文档已移至以下新指南：

- 扩展(OLM v1)