Red Hat Advanced Cluster Management for Kubernetes 2.1

管理应用程序

管理应用程序

Last Updated: 2022-01-31

Red Hat Advanced Cluster Management for Kubernetes 2.1 管理应用程序

管理应用程序

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Manage_applications.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

管理 Red Hat Advanced Cluster Management for Kubernetes 中的应用程序

目录

第 1 章 管理应用程序 **4**
1.1. 应用程序模型和定义 4
1.1.1. 应用程序 5
1.1.2. Channels 5
1.1.3. 订阅（Subscription） 5
1.1.4. 放置规则 6
1.2. 应用程序控制台 6
1.2.1. 应用程序概述 7
1.2.1.1. 单个应用程序概述 7
1.2.2. 资源拓扑 7
1.2.3. 搜索 8
1.2.4. 高级配置 8
1.3. 管理应用程序资源 8
1.3.1. 使用 Git 存储库管理应用程序 9
1.3.1.1. YAML 示例 10
1.3.1.2. 应用程序 Git Ops 10
1.3.1.2.1. GitOps 的 Repo 示例 10
1.3.1.2.2. GitOps 根路径 11
1.3.1.2.3. 受管集群的 GitOps 应用程序 11
1.3.1.2.4. hub 集群的 GitOps 应用程序 11
1.3.1.2.5. 应用 GitOps 12
1.3.2. 使用 Helm 仓库管理应用程序 12
1.3.2.1. YAML 示例 13
1.3.3. 使用对象存储存储库管理应用程序 13
1.3.3.1. YAML 示例 14
1.4. 应用程序高级配置 15
1.4.1. 订阅 Git 资源 15
1.4.1.1. 授予用户和组订阅管理员特权 15
1.4.1.2. 应用程序命名空间示例 16
1.4.1.3. 资源覆盖示例 17
1.4.1.3.1. 协调选项 18
1.4.2. 配置软件包覆盖 19
1.4.3. 设置 Ansible Tower 任务（技术预览） 19
1.4.3.1. 先决条件 19
1.4.3.2. 安装 Ansible Automation Platform Resource Operator： 20
1.4.3.3. 获取 Ansible Tower URL 和令牌 20
1.4.3.4. 获取令牌 20
1.4.3.5. Ansible 集成 20
1.4.3.6. Ansible Operator 组件 21
1.4.3.6.1. Prehook 21
1.4.3.6.2. Posthook 21
1.4.3.6.3. Ansible 放置规则 21
1.4.3.7. Ansible 配置 21
1.4.3.7.1. Ansible secret 21
1.4.3.8. 设置 secret 协调 22
1.4.3.9. Ansible YAML 示例 23
1.4.4. 频道示例 23
1.4.4.1. 频道 YAML 结构 23
1.4.4.2. 频道 YAML 表 24
1.4.4.3. Object storage bucket (ObjectBucket) 频道 25
1.4.4.4. Helm 仓库 (HelmRepo) 频道 25

1.4.4.5. Git（Git）仓库频道 26
1.4.5. Secret 示例 27
1.4.5.1. Secret YAML 结构 27
1.4.6. 订阅示例 27
1.4.6.1. 订阅 YAML 结构 28
1.4.6.2. 订阅 YAML 表 28
1.4.6.3. 订阅文件示例 32
1.4.6.3.1. 订阅时间窗示例 33
1.4.6.3.2. 带有覆盖的订阅示例 33
1.4.6.3.3. Helm 仓库订阅示例 34
1.4.6.3.4. Git 仓库订阅示例 34
1.4.7. 放置规则示例 36
1.4.7.1. 放置规则 YAML 结构 37
1.4.7.2. 放置规则 YAML 值表 37
1.4.7.3. 放置规则示例文件 38
1.4.8. 应用程序示例 39
1.4.8.1. 应用程序 YAML 结构 40
1.4.8.2. 应用程序 YAML 表 40
1.4.8.3. 应用程序文件示例 41

第 1 章 管理应用程序

查看以下主题以了解更多有关创建、部署和管理应用程序的信息。本指南假定您对 Kubernetes 概念和术语有一定的了解。Kubernetes 关键术语和组件在此文档中并没有详细定义。有关 Kubernetes 概念的更多信息，请参阅 Kubernetes 文档。

应用程序管理功能为您提供了构建和部署应用程序及应用程序更新的统一和简化的选项。通过这些功能，开发人员和运维（DevOps）人员可通过基于频道和订阅的自动化功能在环境之间创建和管理应用程序。

请参见以下主题：

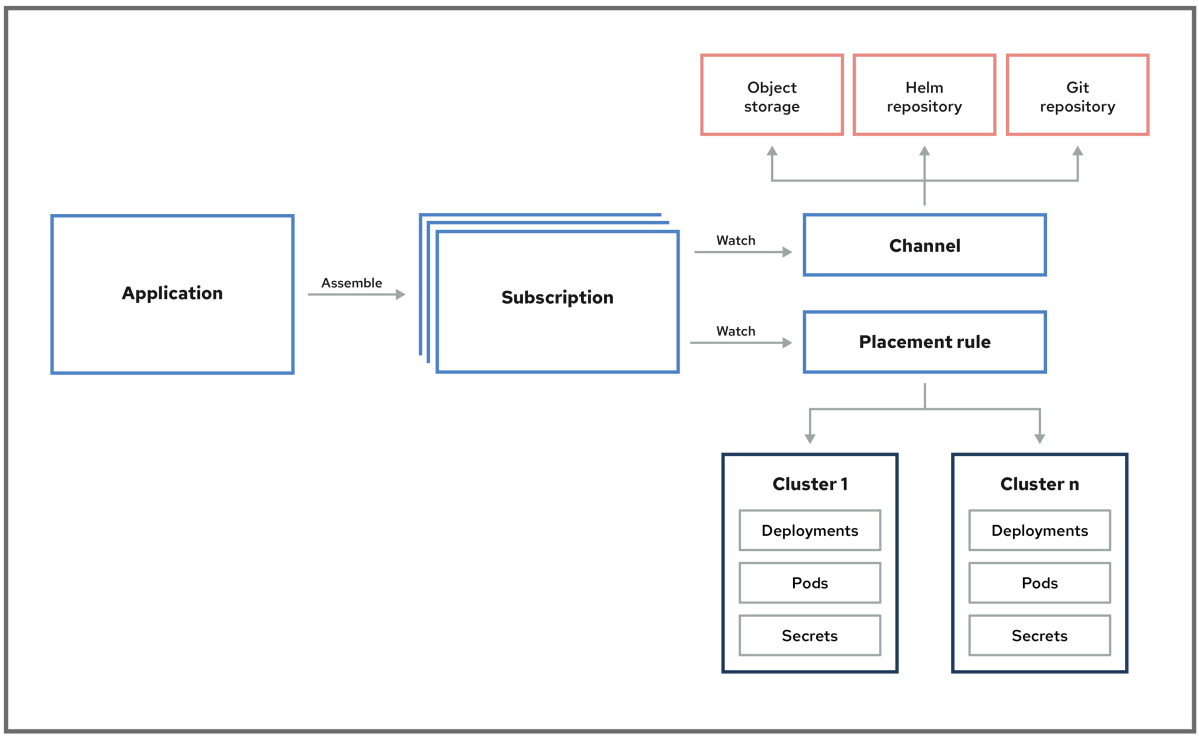
- 应用程序模型和定义
- 应用程序控制台
- 管理应用程序资源
- 使用 Git 存储库管理应用程序
- 使用 Helm 仓库管理应用程序
- 使用对象存储存储库管理应用程序
- 应用程序高级配置
- 订阅 Git 资源
- 配置软件包覆盖
- 设置 Ansible Tower 任务
- 频道示例
- 订阅示例
- 放置规则示例
- 应用程序示例

1.1. 应用程序模型和定义

应用程序模型基于订阅一个或多个 Kubernetes 资源仓库（repository）（*频道*资源），其中包含部署在受管集群上的资源。单集群和多集群应用程序使用相同的 Kubernetes 规格，但多集群应用程序涉及更多的部署和应用程序管理生命周期自动化。

请参阅以下镜像以了解更多有关应用程序模型的信息：

APPLICATION SUBSCRIPTION MODEL



查看以下应用程序资源部分：

1.1.1. 应用程序

Red Hat Advanced Cluster Management for Kubernetes 中的应用程序 **(application.app.k8s.io)** 用于对组成应用程序的 Kubernetes 资源进行分组。

Red Hat Advanced Cluster Management for Kubernetes 应用程序的所有应用程序组件资源都在 YAML 文件 spec 部分中定义。当需要创建或更新应用程序组件资源时，需要创建或编辑正确的 spec 部分，使其包含用于定义资源的标签。

1.1.2. Channels

频道(**channel.apps.open-cluster-management.io**)定义集群可通过订阅来订阅的源仓库，可以是以下类型：hub 集群上的 Git、Helm 发行版本和对象存储仓库以及资源模板。

如果您的应用程序需要的 Kubernetes 资源或 Helm chart 来自需要授权的频道，如授权 Git 仓库，您可以使用 secret 提供对这些频道的访问。您的订阅可以在保持数据安全的同时访问从这些频道部署的 Kubernetes 资源及 Helm chart。

频道使用 hub 集群中的一个命名空间，并指向存储了用于部署的资源的物理位置。集群可以到订阅频道，以标识要部署到每个集群的资源。

备注：最佳实践是在每个命名空间中创建一个频道。Git 频道可以与其他类型的频道（包括 Git、Helm 和 Object 存储）共享命名空间。

频道中的资源只能供订阅该频道的集群访问。

1.1.3. 订阅（Subscription）

订阅(**subscription.apps.open-cluster-management.io**)允许集群订阅源仓库（频道），可以是以下类型：Git 仓库、Helm 发行 registry 或对象存储仓库。

如果 hub 集群是自助管理的，订阅可以在本地将应用程序资源部署到 hub 集群。然后您可以在拓扑中查看 **local-cluster** 订阅。

订阅可以指向某个频道或存储位置，以标识新的或更新的资源模板。订阅 operator 可以在不先检查 hub 集群的情况下直接从存储位置下载并部署到目标受管集群。通过订阅，订阅 operator 可以监控该频道是否有新的或已更新的资源，而不是监控 Hub 集群。

1.1.4. 放置规则

放置规则（**placementrule.apps.open-cluster-management.io**）定义了可部署资源模板的目标集群。使用放置规则帮助您促进可部署资源的多集群部署。放置策略也用于监管和风险策略。

从以下文档了解更多相关信息：

- 应用程序控制台
- 管理应用程序资源
- 使用 Git 存储库管理应用程序
- 使用 Helm 仓库管理应用程序
- 使用对象存储存储库管理应用程序
- 应用程序高级配置
- 订阅 Git 资源
- 设置 Ansible Tower 任务
- 频道示例
- 订阅示例
- 放置规则示例
- 应用程序示例

1.2. 应用程序控制台

控制台包括用于管理应用程序生命周期的仪表板。您可以使用控制台仪表板来创建和管理应用程序，并查看应用程序的状态。增强的功能可帮助开发人员和操作人员在集群中创建、部署、更新、管理和视觉化应用程序。

请参阅以下应用程序控制台功能：

重要：操作基于您分配的角色。如需更多关于访问要求的信息，请参阅基于角色的访问控制文档。

- 可视化集群中部署的应用程序，包括任何关联的资源存储库、订阅和放置配置。
- 创建并编辑应用程序，并订阅资源。默认情况下，hub 集群可以自己管理，并命名为 **local cluster（本地集群）**。您可以选择将应用程序资源部署到这个本地集群，但在本地集群中部署应用程序并非是最佳做法。

- 使用 **Advanced configuration** 查看或编辑频道、订阅和放置规则。
- 访问一个拓扑视图，它代表了应用程序资源的情况，包括资源存储库、订阅、放置规则以及部署的资源，包括使用 Ansible Tower 任务的可选的部署前和部署后 hook（用于 Git 仓库）。
- 查看应用程序上下文中的单个状态，包括部署、更新和订阅。

控制台包括不同的工具，它们各自提供不同的应用程序管理功能。通过这些功能，您可以轻松地创建、查找、更新和部署应用程序资源。

- 应用程序概述
- 资源拓扑
- 搜索
- 高级配置

1.2.1. 应用程序概述

在主 *Overview* 选项卡中包括以下内容：

- 列出所有应用程序的表
- 使用 *Find resources* 过滤列出的应用程序
- 应用程序名称和命名空间
- 通过一个订阅部署的资源的远程和本地集群数量
- 到应用程序部署资源的定义所在仓库的链接
- 显示 Time 窗口约束（如果创建了）
- 应用程序的创建日期
- 更多操作，如 **Delete application**。如果用户有权限，可以执行的操作。

1.2.1.1. 单个应用程序概述

点击表中的应用程序名称查看单个应用程序的详情。请参见以下信息：

- 集群详情，如资源状态。
- 订阅详情
- 资源拓扑

点 **Editor** 选项卡编辑应用程序和相关资源。

1.2.2. 资源拓扑

拓扑可视化显示所选应用程序，包括此应用程序在目标集群中部署的资源。

- 您可以在拓扑视图中选择任何组件来查看更多详情。
- 点资源节点打开属性视图，查看此应用程序部署的任何资源的部署详情。

- 在属性对话框中查看集群节点中的集群 CPU 和内存。
  备注：显示的集群 CPU 和内存百分比是当前使用的百分比。这个值会被进行舍入处理，因此一个很小的值可能会显示为 **0**。

  对于 Helm 订阅，请参阅配置软件包覆盖以定义正确的 **packageName** 和 **packageAlias**，以获得准确的拓扑显示。

- 查看成功的 Ansible Tower 部署，如果使用 Ansible 任务作为部署的应用程序的 prehook 或 posthook。
  点 **Actions** 查看 Ansible 任务部署的详情，包括 Ansible Tower Job URL 和模板名称。另外，如果 Ansible Tower 部署失败，您可以看到错误。

- 点 **Launch resource in Search** 搜索相关资源。

1.2.3. 搜索

控制台 *Search* 页面支持按每个资源的组件 **kind** 搜索应用程序资源。要搜索资源，请使用以下值：

应用程序资源	类型（搜索参数）
Subscription	**Subscription**
Channel	**Channel**
Secret	**Secret**
放置规则	**PlacementRule**
Application	**Application**

您还可以按其他字段搜索，包括名称、命名空间、集群、标签等。

在搜索结果中，您可以查看每个资源的标识详情，包括名称、命名空间、集群、标签和创建日期。

如果您有访问权限，还可以点击搜索结果中的 **Actions**，然后选择删除该资源。

点击搜索结果中的资源名称打开 YAML 编辑器并进行修改。您保存的更改会立即应用到资源。

有关使用搜索的更多信息，请参阅在控制台中进行搜索。

1.2.4. 高级配置

点 **Advanced configuration** 选项卡查看所有应用程序的术语和资源表。您可以查找资源，并过滤订阅、放置规则和频道。如果您有访问权限，还可以点多个 **Actions**，如 Edit、Search 和 Delete。

选择要查看或编辑 YAML 的资源。

1.3. 管理应用程序资源

通过控制台，您可以使用 Git 仓库、Helm 仓库和对象存储库创建应用程序。

重要：Git 频道可以与所有其他频道类型共享命名空间：Helm、对象存储和其他 Git 命名空间。

请参阅以下主题以开始管理应用程序：

- 使用 Git 存储库管理应用程序
- 使用 Helm 仓库管理应用程序
- 使用对象存储存储库管理应用程序

1.3.1. 使用 Git 存储库管理应用程序

当您使用应用程序部署 Kubernetes 资源时，这些资源位于特定的存储库中。了解如何在以下流程中从 Git 存储库部署资源。如需了解更多有关应用程序模型的信息，请参阅应用程序模型和定义：

用户需要的访问权限：可创建应用的用户角色。您只能执行分配了相关角色的操作。如需更多关于访问要求的信息，请参阅基于角色的访问控制文档。

1. 在控制台导航菜单中点 **Manage application**。
2. 点 **Create application**。
   对于以下步骤，选择 **YAML:在** 上，在控制台中创建应用程序时查看 YAML。请参阅后面的 YAML 示例。
3. 在正确的字段中输入以下值：
   - name：输入应用程序的有效 Kubernetes 名称。
   - Namespace：从列表中选择一个命名空间。如果分配了正确的访问角色，您还可以使用有效的 Kubernetes 名称创建命名空间。
4. 从可以使用的存储库列表中选择 **Git**。
5. 输入所需的 URL 路径或选择现有路径。
   如果选择了现有的 Git 存储库路径，则不需要指定连接信息（如果这是私有存储库）。连接信息是预先设置的，您不需要查看这些值。

   如果输入了新的 Git 存储库路径，则可以选择性地输入 Git 连接信息（如果这是一个私有 Git 存储库）。
6. 输入可选字段的值，如分支和文件夹。
7. 设置任何可选的部署前和部署后的任务。
   技术预览：如果您有要在订阅部署应用程序资源之前或之后运行的 Ansible Tower 作业，则设置 Ansible Tower secret。定义 Ansible 作业的 Ansible Tower 任务必须放置在此存储库的 prehook 和 posthook 文件夹中。

   如果在应用程序命名空间中创建了 secret，可以从下拉菜单中选择 Ansible Tower secret。在这个实例中，连接信息是预先设置的，您不需要查看这些值。

   如果输入了新的 Ansible Tower secret 名称以创建新 secret，则需要输入 Ansible Tower 主机和令牌。
8. 在 *Select clusters to deploy* 中，您可以更新应用程序的放置规则信息。从中选择：
   - 在本地集群中部署
   - 部署到所有在线集群和本地集群

- 仅在与指定标签匹配的集群上部署应用程序资源
- 如果在现有的、并已定义了放置规则的命名空间中创建应用程序，则可以选择 *Select existing placement configuration* 选项。

9. 在 *Settings* 中，您可以指定应用程序的行为。要在指定时间窗内阻止或激活对部署的更改，为 *Deployment window* 和您的 *Time window configuration* 选择一个选项。
10. 您可以选择另一个存储库或点 **Save**。
11. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.3.1.1. YAML 示例

以下示例频道定义显示了 Git 仓库的频道示例。在以下示例中，**secretRef** 是指用于访问 **pathname** 中指定的 Git 仓库的用户身份。如果您有一个公共仓库库，则不需要 **secretRef**：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: hive-cluster-gitrepo
  namespace: gitops-cluster-lifecycle
spec:
  type: Git
  pathname: https://github.com/open-cluster-management/gitops-clusters.git
  secretRef:
    name: github-gitops-clusters
---
apiVersion: v1
kind: Secret
metadata:
  name: github-gitops-clusters
  namespace: gitops-cluster-lifecycle
data:
  user: dXNlcgo=          # Value of user and accessToken is Base 64 coded.
  accessToken: cGFzc3dvcmQ
```

备注：要查看 REST API，请使用 API。

1.3.1.2. 应用程序 Git Ops

使用 **local-cluster** 放置规则，您可以使用订阅向 hub 集群提供与配置相关的资源。

这些资源可以是用来配置应用程序和策略、运行 Ansible 作业、在置备或导入后配置集群的订阅。

1.3.1.2.1. GitOps 的 Repo 示例

在以下示例仓库中，您会看到每个 hub 集群的文件夹。您还可以为每个 hub 集群或每个 hub 集群创建一个仓库。

在 hub 集群上定义的单个订阅将拉取所有其他配置资源，如 hub 集群配置、策略和将配置和部署到 hub 和受管集群的通用应用程序。

用于存储这类信息的 Git 仓库类似以下示例文件和目录结构。参阅 root 路径、受管集群和 hub 集群部分：

1.3.1.2.2. GitOps 根路径

仓库根路径中的这些文件会创建一个订阅，该订阅会引用此 Git 仓库并应用所有 YAML，但 **.kubernetesignore** 中指定的除外。它包括这四个订阅文件和 **./managed-cluster-common** 目录。

```
/   # Repository root directory

 # The subscription that delivers all the previous content to the hub cluster:

 hub-application.yaml   # This represents the hub cluster configuration in the console
 hub-channels.yaml      # This points to `rhacm-hub-cluster` Git repository
 hub-subscriptions.yaml # This defines the time window, branch to be used, and defines which
directories containing appropriate configs, such as `hub-policies`, should be used (can be all)
 hub-placement.yaml     # Points back to the `local-cluster` (hub cluster that is managed)
 .kubernetesignore      # Tells the subscription to ignore hub-application.yaml, hub-channels.yaml,
hub-subscription.yaml & hub-placement.yaml
```

1.3.1.2.3. 受管集群的 GitOps 应用程序

以下目录包含了将应用程序应用到 *managed* 集群的订阅。这些订阅通过 root 目录中的订阅应用到 hub 集群。

在以下示例中，您会看到一个订阅到另一个订阅的订阅：

```
common-managed/
   apps/
     app-name-0/
       application.yaml
       subscription.yaml
       channel.yaml       # This points to a repository named `app-name-0`, of type Git, Helm, or Object
Storage
       placementrule.yaml
     app-name-1/
       application.yaml
       subscription.yaml
       channel.yaml       # This points to a repository named `app-name-0`, of type Git, Helm, or Object
Storage
       placementrule.yaml
   config/
     application.yaml      # named: `day2-config`
     subscription.yaml     # Points to the `managed-cluster-common/config` parent directory
     channel.yaml          # Can point to this Git repository or a different repository with the day-two
configuration
     placementrule.yaml    # Defines the clusters to target
managed-cluster-common/
  configs/                 # These configurations are referenced through the `config` subscription
   certmanagement.yaml
   auth-oidc.yaml
   autoscaler.yaml
   descheduler.yaml
```

1.3.1.2.4. hub 集群的 GitOps 应用程序

以下策略应用于 hub 集群，为 hub 集群以及远程集群提供策略配置。

这些是通过 root 订阅交付的，如下例所示：

```
managed-cluster-common/
  policies/
    policy-0.yaml
    policy-1.yaml
  hub-policies/
    policy-0.yaml
    vault.yaml
    operators.yaml
```

1.3.1.2.5. 应用 GitOps

使用前面的组合示例，您可以指定以下内容：

1. 可使用 CLI 命令应用一个根订阅。根订阅将返回此仓库，将所有 YAML 应用到 hub 集群。
2. 第 1 步中的订阅，它会应用应用程序并从 **common-managed/** 配置订阅。
3. 第 2 步的配置订阅，它应用 **managed-cluster-common/** 中定义的资源。
4. 在 **managed-cluster-common/** 中定义的策略也由第 1 步的订阅应用到 hub 集群。这些策略包括针对 hub 集群的策略，以及那些针对受管集群的策略。

1.3.2. 使用 Helm 仓库管理应用程序

当您使用应用程序部署 Kubernetes 资源时，这些资源位于特定的存储库中。了解如何在以下流程中从 Helm 仓库部署资源。如需了解更多有关应用程序模型的信息，请参阅应用程序模型和定义：

用户需要的访问权限：可创建应用的用户角色。您只能执行分配了相关角色的操作。如需更多关于访问要求的信息，请参阅基于角色的访问控制文档。

1. 在控制台导航菜单中点 **Manage application**。
2. 点 **Create application**。
   对于以下步骤，选择 **YAML:在** 上，在控制台中创建应用程序时查看 YAML。请参阅后面的 YAML 示例。
3. 在正确的字段中输入以下值：
   - name：输入应用程序的有效 Kubernetes 名称。
   - Namespace：从列表中选择一个命名空间。如果分配了正确的访问角色，您还可以使用有效的 Kubernetes 名称创建命名空间。
4. 从您可以使用的存储库列表中选择 **Helm**。
5. 输入所需 URL 路径，或者选择现有路径，然后输入软件包版本。
   如果您选择了现有的 Helm 仓库路径，则不需要指定连接信息（如果这是私有仓库）。连接信息是预先设置的，您不需要查看这些值。

   如果输入了新的 Helm 仓库路径，则可以输入 Helm 连接信息（如果这是私有 Helm 仓库）。
6. 在 *Select clusters to deploy* 中，您可以更新应用程序的放置规则信息。从中选择：
   - 在本地集群中部署

- 部署到所有在线集群和本地集群
- 仅在与指定标签匹配的集群上部署应用程序资源
- 如果在现有的、并已定义了放置规则的命名空间中创建应用程序，则可以选择 *Select existing placement configuration* 选项。

7. 在 *Settings* 中，您可以指定应用程序的行为。要在指定时间窗内阻止或激活对部署的更改，为 *Deployment window* 和您的 *Time window configuration* 选择一个选项。
8. 您可以选择另一个存储库或点 **Save**。
9. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.3.2.1. YAML 示例

以下示例频道定义将 Helm 仓库抽象为频道：

备注：对于 Helm，Helm chart 中包含的所有 Kubernetes 资源都必须具有标签 release。 **{{ .Release.Name }}'** 才能正确显示应用程序拓扑。

```
apiVersion: v1
kind: Namespace
metadata:
  name: hub-repo
---
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: helm
  namespace: hub-repo
spec:
    pathname: [https://kubernetes-charts.storage.googleapis.com/] # URL points to a valid chart URL.
    type: HelmRepo
```

以下频道定义显示了 Helm 仓库频道的另一个示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
```

备注：要查看 REST API，请使用 API。

1.3.3. 使用对象存储存储库管理应用程序

当您使用应用程序部署 Kubernetes 资源时，这些资源位于特定的存储库中。了解如何按照以下流程从对象存储存储库部署资源。如需了解更多有关应用程序模型的信息，请参阅应用程序模型和定义：

用户需要的访问权限：可创建应用的用户角色。您只能执行分配了相关角色的操作。如需更多关于访问要求的信息，请参阅基于角色的访问控制文档。

当您使用应用程序部署 Kubernetes 资源时，这些资源位于特定的存储库中。了解如何在以下流程中从 Git 存储库部署资源。

1. 在控制台导航菜单中点 **Manage application**。
2. 点 **Create application**。
   对于以下步骤，选择 **YAML:在** 上，在控制台中创建应用程序时查看 YAML。请参阅后面的 YAML 示例。
3. 在正确的字段中输入以下值：
   - name：输入应用程序的有效 Kubernetes 名称。
   - Namespace：从列表中选择一个命名空间。如果分配了正确的访问角色，您还可以使用有效的 Kubernetes 名称创建命名空间。
4. 从可以使用的存储库列表中选择 **Object storage**。
5. 输入所需的 URL 路径或选择现有路径。
   如果您选择了现有对象存储存储库路径，则不需要指定连接信息（如果这是私有存储库）。连接信息是预先设置的，您不需要查看这些值。

   如果您输入了新的对象存储存储库路径，则可以输入对象存储连接信息（如果这是私有对象存储存储库）。
6. 为可选字段输入值。
7. 设置任何可选的部署前和部署后的任务。
8. 在 *Select clusters to deploy* 中，您可以更新应用程序的放置规则信息。从中选择：
   - 在本地集群中部署
   - 部署到所有在线集群和本地集群
   - 仅在与指定标签匹配的集群上部署应用程序资源
   - 如果在现有的、并已定义了放置规则的命名空间中创建应用程序，则可以选择 *Select existing placement configuration* 选项。
9. 在 *Settings* 中，您可以指定应用程序的行为。要在指定时间窗内阻止或激活对部署的更改，为 *Deployment window* 和您的 *Time window configuration* 选择一个选项。
10. 您可以选择另一个存储库或点 **Save**。
11. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.3.3.1. YAML 示例

以下示例频道定义将对象存储抽象为频道：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
```

```
  name: dev
  namespace: ch-obj
spec:
  type: Object storage
  pathname: [http://9.28.236.243:31311/dev] # URL is appended with the valid bucket name, which
matches the channel name.
  secretRef:
    name: miniosecret
  gates:
    annotations:
      dev-ready: true
```

备注：要查看 REST API，请使用 API。

1.4. 应用程序高级配置

在 Red Hat Advanced Cluster Management for Kubernetes 中，应用程序由多个应用程序资源组成。您可以使用频道、订阅和放置规则资源来帮助部署、更新和管理整个应用程序。

单集群和多集群应用程序使用相同的 Kubernetes 规格，但多集群应用程序涉及更多的部署和应用程序管理生命周期自动化。

Red Hat Advanced Cluster Management for Kubernetes 应用程序的所有应用程序组件资源都在 YAML 文件 spec 部分中定义。当需要创建或更新应用程序组件资源时，需要创建或编辑正确的 spec 部分，使其包含用于定义资源的标签。

查看以下与应用程序高级配置相关的内容：

- 订阅 Git 资源
- 设置 Ansible Tower 任务
- 频道示例
- 订阅示例
- 放置规则示例
- 应用程序示例

1.4.1. 订阅 Git 资源

订阅管理员可以更改默认行为。默认情况下，当订阅将订阅的应用程序部署到目标集群时，即使应用程序资源与其他命名空间关联，应用程序也会部署到该订阅命名空间中。

另外，如果应用程序资源存在于集群中，且不是由订阅创建，订阅就不能在该现有资源上应用新资源。作为订阅管理员，请查看以下流程来更改默认设置。

需要的访问权限：集群管理员

1.4.1.1. 授予用户和组订阅管理员特权

了解如何授予订阅管理员访问权限。

1. 从控制台登录到 Red Hat OpenShift Container Platform 集群。

2. 创建一个或多个用户。有关创建用户的信息，请参阅准备用户。
   您创建的用户是 **app.open-cluster-management.io/subscription** 应用程序的管理员。在 OpenShift Container Platform 中，订阅管理员可以更改默认行为。您可以将这些用户组成一个组来代表订阅管理组群（在稍后会进行演示）。

3. 在终端中登录 Red Hat Advanced Cluster Management 集群。

4. 使用以下命令在 **open-cluster-management:subscription-admin** ClusterRoleBinding 中添加以下 subjects 内容：

   ```
   oc edit clusterrolebinding open-cluster-management:subscription-admin
   ```

   备注：最初，open **-cluster-management:subscription-admin** ClusterRoleBinding 没有 subject。

   您的 subject 可能如下所示：

   ```
   subjects:
   - apiGroup: rbac.authorization.k8s.io
     kind: User
     name: example-name
   - apiGroup: rbac.authorization.k8s.io
     kind: Group
     name: example-group-name
   ```

接下来，请参阅订阅管理员如何更改默认行为的更多示例。

1.4.1.2. 应用程序命名空间示例

在这个示例中，以订阅管理员身份登录。创建订阅以从 Git 存储库订阅示例资源 YAML 文件。示例文件包含位于以下不同命名空间中的订阅：

适用的频道类型：Git

- ConfigMap **test-configmap-1** 在 **multins** 命名空间中创建。
- ConfigMap **test-configmap-2** 在 **default** 命名空间中创建。
- ConfigMap **test-configmap-3** 在 **subscription** 命名空间中创建。

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: multins
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: multins
data:
  path: resource1
---
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: test-configmap-2
  namespace: default
data:
  path: resource2
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-3
data:
  path: resource3
```

如果订阅由其他用户创建，则所有 ConfigMap 都会在与订阅相同的命名空间中创建。

1.4.1.3. 资源覆盖示例

适用的频道类型：Git、ObjectBucket（控制台中的对象存储）

在本例中，目标集群中已存在以下 ConfigMap。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: sub-ns
data:
  name: user1
  age: 19
```

从 Git 存储库订阅以下示例资源 YAML 文件，并替换现有的 ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: sub-ns
data:
  age: 20
```

您可以以订阅管理员身份登录，并创建带有**apps.open-cluster-management.io/reconcile-option: replace** 注解的订阅。请参见以下示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: subscription-example
  namespace: sub-ns
  annotations:
    apps.open-cluster-management.io/git-path: sample-resources
    apps.open-cluster-management.io/reconcile-option: replace
spec:
  channel: channel-ns/somechannel
```

```
  placement:
    placementRef:
      name: dev-clusters
```

当此订阅由订阅管理员创建并订阅 ConfigMap 资源时，现有 ConfigMap 将替换为以下内容：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: sub-ns
data:
  age: 20
```

如果要从 Git 存储库订阅以下示例资源 YAML 文件，并使用现有 ConfigMap **合并**，使用 **apps.open-cluster-management.io/reconcile-option: merge** 注解。请参见以下示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: subscription-example
  namespace: sub-ns
  annotations:
    apps.open-cluster-management.io/git-path: sample-resources
    apps.open-cluster-management.io/reconcile-option: merge
spec:
  channel: channel-ns/somechannel
  placement:
    placementRef:
      name: dev-clusters
```

当此订阅由订阅管理员创建并订阅 ConfigMap 资源时，现有 ConfigMap 会被合并，如下例所示：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: sub-ns
data:
  name: user1
  age: 20
```

当使用 **merge** 选项时，订阅的资源中的条目会在现有资源中创建或更新。没有条目会从现有资源中删除。

重要：如果您想要覆盖订阅的资源由其他操作员或控制器自动协调，资源配置由订阅以及控制器或操作员更新。在这种情况下不要使用这个方法。

1.4.1.3.1. 协调选项

您还可以在单个资源中使用 **apps.open-cluster-management.io/reconcile-option** 注解来覆盖订阅级别的协调选项。

例如，如果您在订阅中添加 **apps.open-cluster-management.io/reconcile-option: replace** 注解，在订

例如，如果您在订阅中添加 **apps.open-cluster-management.io/reconcile-option: replace** 注解，在订阅的 Git 仓库的一个资源 YAML 中添加 **apps.open-cluster-management.io/reconcile-option: merge** 注解，则资源将在目标集群中合并，其他资源则在替换其他资源时合并。

1.4.2. 配置软件包覆盖

为订阅中所订阅的 Helm chart 或 Kubernetes 资源的订阅覆盖值配置软件包覆盖。

要配置软件包覆盖，请将 Kubernetes 资源 **spec** 中要覆盖的字段指定为 **path** 字段的值。将替换值指定为 **value** 字段的值。

例如，如果您需要在订阅的 Helm Chart 的 Helm 发行版本的 **spec** 中覆盖 values 字段，则需要将订阅定义中的 **path** 字段的值设置为 **spec**。

```
packageOverrides:
- packageName: nginx-ingress
  packageOverrides:
  - path: spec
    value: my-override-values
```

value 字段的内容用于覆盖 **Helm** spec 的 **spec** 字段中的值。

- 在 Helm 发行版本中，**spec** 字段的覆盖值合并到发行版本 **values.yaml** 文件中，以覆盖现有的值。此文件用于检索 Helm 发行版本的可配置变量。
- 如果您需要覆盖 Helm 发行版本的发行版本名称，请在定义中包含 **packageOverride** 部分。通过包含以下字段为 Helm 发行版本定义 **packageAlias**:
  - 用于标识 Helm chart 的 **packageName**。
  - 用于表示您将覆盖发行版本名称的 **packageAlias**。

  默认情况下，如果没有指定 Helm 发行版本名称，则使用 Helm chart 名称来标识该发行版本。在某些情况下，比如有多个发行版本订阅了同一 chart 时，可能会发生冲突。发行版本名称在命名空间中的不同订阅之间必须是唯一的。如果您创建的订阅的发行版本名称不是唯一的，则会出现错误。您必须通过定义 **packageOverride** 为您的订阅设置不同的发行版本名称。如果要在现有订阅中更改名称，必须首先删除该订阅，然后使用首选发行版本名称重新创建订阅。

  +

  ```
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: my-helm-release-name
  ```

1.4.3. 设置 Ansible Tower 任务（技术预览）

Red Hat Advanced Cluster Management 与 Ansible Tower 自动化集成，以便您可以为 Git 订阅应用程序管理创建 prehook 和 posthook AnsibleJob 实例。通过 Ansible Tower 作业，您可以自动完成任务并与外部服务（如 Slack 和 PagerDuty 服务）集成。您的 Git 仓库资源根路径将包含用于部署应用程序、更新应用程序或从集群中删除应用程序一部分的 Ansible Tower 作业的 **prehook** 和 **posthook** 目录。

需要的访问权限：集群管理员

1.4.3.1. 先决条件

- OpenShift Container Platform 4.5 或更高版本
- 已安装 Ansible Tower 版本 3.7.3 或更高版本。安装最新版本的 Ansible Tower 是最佳实践方案。如需更多详细信息，请参阅 Red Hat AnsibleTower 文档 。
- 安装 Ansible Automation Platform Resource Operator，将 Ansible 作业连接到 Git 订阅的生命周期。为了获得最佳结果，在使用 AnsibleJob 启动 Ansible Tower 作业时，Ansible Tower 作业模板在运行时应该是等价的。

在 INVENTORY 和 EXTRA VARIABLES 模板中检查 **PROMPT ON LAUNCH**。如需更多信息，请参阅 作业模板。

1.4.3.2. 安装 Ansible Automation Platform Resource Operator：

1. 登录您的 OpenShift Container Platform 集群控制台。
2. 在控制台导航中点 **OperatorHub**。
3. 搜索并安装 *Ansible Automation Platform Resource Operator* 。

1.4.3.3. 获取 Ansible Tower URL 和令牌

Ansible Tower URL 是用于登录到 Tower 的 URL。在使用 Ansible prehook 和 posthook 配置应用程序时，*应用程序控制台*或 Tower 访问 secret 需要此项。

请参见以下示例 URL: **https://ansible-tower-web-svc-tower.apps.my-openshift-cluster.com**。

1.4.3.4. 获取令牌

1. 登录到您的 Ansible Tower 控制台。
2. 在控制台导航中点 **Users**。
3. 搜索正确的用户。
4. 点 **Edit user** 图标。
5. 在 user 部分点 **TOKENS**。
6. 点 **+** 按钮添加令牌。
7. 将 **APPLICATION** 字段留空。
8. 在 **DESCRIPTION** 字段中，提供您要用于此令牌的预期用途。
9. 在 **SCOPE** 下拉菜单中选择 **Write**。
10. 点 **SAVE** 并记录所提供的 **TOKEN**。

1.4.3.5. Ansible 集成

您可以将 Ansible Tower 任务集成到 Git 订阅中。例如，对于数据库前端和后端应用程序，数据库需要使用带有 Ansible 作业的 Ansible Tower 进行实例化，应用程序由 Git 订阅安装。在使用订阅部署前端和后端应用程序 *前*，数据库会被实例化。

应用程序订阅 Operator 被改进，以定义两个子文件夹： **prehook** 和 **posthook**。这两个文件夹都位于 Git 仓库资源根路径中，并分别包含所有 prehook 和 posthook Ansible 作业。

创建 Git 订阅时，所有 pre AnsibleJob 和 post AnsibleJob 资源都会作为对象解析并存储在内存中。应用程序订阅控制器决定何时创建 AnsibleJob 实例。

1.4.3.6. Ansible Operator 组件

在创建订阅 CR 时，Git-branch 和 Git-path 会指向 Git 存储库根位置。在 Git root 位置中，两个子文件夹 **prehook** 和 **posthook** 应该至少包含一个 **Kind:AnsibleJob** 资源。

1.4.3.6.1. Prehook

应用程序订阅控制器在 prehook 文件夹中搜索所有 **Kind:AnsibleJob** CR，作为 prehook AnsibleJob 对象，然后生成新的 prehook AnsibleJob 实例。新实例名称是 prehook AnsibleJob 对象名称和随机后缀字符串。

一个实例名称示例： **database-sync-1-2913063**。

应用程序订阅控制器在 1 分钟循环中再次对协调请求进行队列，它会在 prehook AnsibleJob **status.anisibleJobResult** 中检查。当 prehook **status.ansibleJobResult.status** 的状态为 **successful** 时，应用程序订阅将继续部署主订阅。

1.4.3.6.2. Posthook

更新应用程序订阅状态时，如果订阅状态被订阅或传播到订阅状态的所有目标集群，应用程序订阅控制器会将 posthook 文件夹中的所有 **AnsibleJob Kind** CR 搜索为 posthook AnsibleJob 对象。然后，它会生成新的 posthook **AnsibleJob** 实例。新实例名称是 posthook **AnsibleJob** 对象名称以及一个随机的后缀字符串。

一个实例名称示例： **service-ticket-1-2913849**。

1.4.3.6.3. Ansible 放置规则

通过有效的 prehook AnsibleJob，订阅会启动 prehook AnsibleJob 而不受放置规则的影响。例如，您可以有一个 prehook AnsibleJob，它无法传递放置规则订阅。当放置规则更改时，会创建新的 prehook 和 posthook AnsibleJob 实例。

1.4.3.7. Ansible 配置

您可以使用以下任务配置 Ansible Tower 配置：

1.4.3.7.1. Ansible secret

您必须在同一订阅命名空间中创建一个 Ansible Tower secret CR。Ansible Tower secret 仅限于相同的订阅命名空间。

在控制台的 **Ansible Tower secret name** 部分填充相关的数据来创建 secret。要使用终端创建 secret，编辑并应用以下 **yaml**:

运行以下命令来添加 YAML 文件：

```
oc apply -f
```

请参见以下 YAML 示例：

备注：命名空间 与订阅命名空间相同。**stringData:token** 和 **host** 来自 Ansible Tower。

```
apiVersion: v1
kind: Secret
metadata:
  name: toweraccess
  namespace: same-as-subscription
type: Opaque
stringData:
  token: ansible-tower-api-token
  host: https://ansible-tower-host-url
```

当应用程序订阅控制器创建 prehook 和 posthook AnsibleJobs 时，如果 subscription **spec.hooksecretref** 中的 secret 可用，则会将其发送到 AnsibleJob CR **spec.tower_auth_secret**，AnsibleJob 可以访问 Ansible Tower。

1.4.3.8. 设置 secret 协调

对于带有 prehook 和 posthook AnsibleJobs 的主订阅，在 Git 存储库中更新所有 prehook 和 posthook AnsibleJobs 或主订阅后，应当协调主订阅。

Prehook AnsibleJobs 和主订阅持续协调并重新启动新的 AnsibleJob 实例。

1. 完成 pre-AnsibleJob 后，重新运行主订阅。
2. 如果主订阅中有任何规格更改，请重新部署订阅。应更新主要的订阅状态，使其与重新部署过程保持一致。
3. 将 hub 订阅的状态重置为 **nil**。订阅会与目标集群上的订阅部署一起刷新。
   当目标集群上的部署完成时，目标集群上的订阅状态会更新为 **"subscribed"** 或 **"failed"**，并同步到 hub 集群订阅状态。
4. 完成主订阅后，重新启动一个新的 AnsibleJob 实例。
5. 验证 DONE 订阅已更新。请参见以下输出：
   - subscription.status == **"subscribed"**
   - subscription.status == **"propagated"** 带有所有目标集群 **"subscribed"**

创建 AnsibleJob CR 时，会创建一个 Kubernetes 作业 CR，通过与目标 Ansible Tower 通信来启动 Ansible Tower 作业。作业完成后，作业的最终状态将返回 AnsibleJob **status.ansibleJobResult**。

备注：

Ansible Job operator 保留 AnsibleJob status.conditions 以存储 Kubernetes 作业结果的创建。status.conditions 并不反映实际 Ansible Tower 作业状态。

订阅控制器会根据 **AnsibleJob.status.ansibleJobResult** 而不是 **AnsibleJob.status.conditions** 检查 Ansible Tower 作业状态。

如 prehook 和 posthook AnsibleJob 工作流中所述，当 Git 存储库中更新了主订阅时，会创建一个新的 prehook 和 posthook AnsibleJob 实例。因此，一个主订阅可以链接到多个 AnsibleJob 实例。

subscription.status.ansibleJobs 中定义了四个字段：

- lastPrehookJobs:最新的 prehook AnsibleJobs

- prehookJobsHistory:所有 prehook AnsibleJobs 历史记录
- lastPosthookJobs:最新的 posthook AnsibleJobs
- posthookJobsHistory:所有 posthook AnsibleJobs 历史记录

1.4.3.9. Ansible YAML 示例

请参阅 Git prehook 和 posthook 文件夹中的 AnsibleJob **.yaml** 文件示例：

```
apiVersion: tower.ansible.com/v1alpha1
kind: AnsibleJob
metadata:
  generateName: demo-job-001
  namespace: default
spec:
  tower_auth_secret: toweraccess
  job_template_name: Demo Job Template
  extra_vars:
    cost: 6.88
    ghosts: ["inky","pinky","clyde","sue"]
    is_enable: false
    other_variable: foo
    pacman: mrs
    size: 8
    targets_list:
    - aaa
    - bbb
    - ccc
    version: 1.23.45
```

1.4.4. 频道示例

查看可以用来构建文件的示例和 YAML 定义。频道 (**channel.apps.open-cluster-management.io**) 为您提供改进的持续集成以及持续交付功能，以便创建和管理 Red Hat Advanced Cluster Management for Kubernetes 应用程序。

要使用 Kubernetes CLI 工具，请参阅以下流程：

a. 使用您首选的编辑工具编写并保存您的应用程序 YAML 文件。

b. 运行以下命令，将文件应用到 API 服务器。将 **filename** 替换为您的文件名称：

```
kubectl apply -f filename.yaml
```

c. 运行以下命令验证您的应用程序资源是否已创建：

```
kubectl get Application
```

备注：此发行版本不提供 Kubernetes 命名空间(Namespace)频道。

1.4.4.1. 频道 YAML 结构

以下 YAML 结构显示频道的必需字段，以及一些常见可选字段。您的 YAML 结构需要包含一些必需字段和值。根据应用程序管理要求，您可能需要包含其他可选字段和值。您可以使用任何工具编写您自己的 YAML 内容。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name:
  namespace: # Each channel needs a unique namespace, except Git channel.
spec:
  sourceNamespaces:
  type:
  pathname:
  secretRef:
    name:
  gates:
    annotations:
  labels:
```

1.4.4.2. 频道 YAML 表

字段	描述
apiVersion	必需。将值设为 **apps.open-cluster-management.io/v1**。
kind	必需。将值设为 **Channel** 以表示资源是频道。
metadata.name	必需。频道的名称。
metadata.namespace	必需。频道的命名空间；除 Git 频道外，每个频道都需要一个唯一的命名空间。
spec.sourceNamespaces	可选。标识频道控制器监控的命名空间，看是否有新的或已更新的可部署资源以检索并提升到频道。
spec.type	必需。频道类型。支持的类型有：**HelmRepo**、**Git** 和 **ObjectBucket** （控制台中的对象存储）
spec.pathname	对于 **HelmRepo**、**Git** 和 **ObjectBucket** 频道是必需的。对于 **HelmRepo** 频道，将值设置为 Helm 仓库的 URL。对于 **ObjectBucket** 频道，将值设置为对象存储的 URL。对于 **Git** 频道，将值设置为 Git 仓库的 HTTPS URL。
spec.secretRef.name	可选。标识用于身份验证的 Kubernetes Secret 资源，如用于访问仓库或 chart。您只能对 **HelmRepo**、**ObjectBucket** 和 **Git** 类型的频道使用 secret 进行身份验证。

字段	描述
spec.gates	可选。定义在频道中提升可部署资源的要求。如果没有设置任何要求，则会将添加到频道命名空间或源的任何可部署资源提升到频道。**gates** 不适用于 **HelmRepo** 和 **Git** 频道类型，仅适用于 **ObjectBucket** 频道类型。
spec.gates.annotations	可选。频道注解。可部署资源必须具有匹配的注解才能包含在频道中。
metadata.labels	可选。频道的标签。

频道的定义结构类似以下 YAML 内容：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
```

1.4.4.3. Object storage bucket (ObjectBucket) 频道

以下示例频道定义将对象存储桶抽象为频道：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
 name: dev
 namespace: ch-obj
spec:
 type: ObjectBucket
 pathname: [http://9.28.236.243:31311/dev] # URL is appended with the valid bucket name, which matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

1.4.4.4. Helm 仓库 (HelmRepo) 频道

以下示例频道定义将 Helm 仓库抽象为频道：

```
apiVersion: v1
kind: Namespace
metadata:
  name: hub-repo
---
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: Helm
  namespace: hub-repo
spec:
    pathname: [https://9.21.107.150:8443/helm-repo/charts] # URL points to a valid chart URL.
    configMapRef:
      name: insecure-skip-verify
    type: HelmRepo
---
apiVersion: v1
data:
  insecureSkipVerify: "true"
kind: ConfigMap
metadata:
  name: insecure-skip-verify
  namespace: hub-repo
```

以下频道定义显示了 Helm 仓库频道的另一个示例：

备注：对于 Helm，Helm chart 中包含的所有 Kubernetes 资源都必须具有标签 release。 **{{ .Release.Name }}'** 才能正确显示应用程序拓扑。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: predev-ch
  namespace: ns-ch
  labels:
    app: nginx-app-details
spec:
  type: HelmRepo
  pathname: https://kubernetes-charts.storage.googleapis.com/
```

1.4.4.5. Git（Git）仓库频道

以下示例频道定义显示了 Git 仓库的频道示例。在以下示例中， **secretRef** 是指用于访问 **pathname** 中指定的 Git 仓库的用户身份。如果您有一个公共仓库库，则不需要 **secretRef**：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: hive-cluster-gitrepo
  namespace: gitops-cluster-lifecycle
spec:
  type: Git
  pathname: https://github.com/open-cluster-management/gitops-clusters.git
  secretRef:
```

```
    name: github-gitops-clusters
---
apiVersion: v1
kind: Secret
metadata:
  name: github-gitops-clusters
  namespace: gitops-cluster-lifecycle
data:
  user: dXNlcgo=            # Value of user and accessToken is Base 64 coded.
  accessToken: cGFzc3dvcmQ
```

1.4.5. Secret 示例

Secret (**Secret**) 属于 Kubernetes 资源，可用于存储授权和其他敏感信息，如密码、OAuth 令牌和 SSH 密钥。通过将这些信息存储为 secret，您可以将信息与需要相应信息的应用程序组件分开以提高数据安全性。

要使用 Kubernetes CLI 工具，请参阅以下流程：

a. 使用您首选的编辑工具编写并保存您的应用程序 YAML 文件。

b. 运行以下命令，将文件应用到 API 服务器。将 **filename** 替换为您的文件名称：

```
kubectl apply -f filename.yaml
```

c. 运行以下命令验证您的应用程序资源是否已创建：

```
kubectl get Application
```

Secret 的定义结构类似以下 YAML 内容：

1.4.5.1. Secret YAML 结构

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
     apps.open-cluster-management.io/deployables: "true"
  name: [secret-name]
  namespace: [channel-namespace]
data:
  AccessKeyID: [ABCdeF1=] #Base64 encoded
  SecretAccessKey: [gHIjk2lmnoPQRST3uvw==] #Base64 encoded
```

1.4.6. 订阅示例

查看可以用来构建文件的示例和 YAML 定义。和频道一样，订阅 (**subscription.apps.open-cluster-management.io**) 为您提供改进的持续集成和持续交付功能以用于应用程序管理。

要使用 Kubernetes CLI 工具，请参阅以下流程：

a. 使用您首选的编辑工具编写并保存您的应用程序 YAML 文件。

b. 运行以下命令，将您的文件应用到 apiserver。将 **filename** 替换为您的文件名称：

```
kubectl apply -f filename.yaml
```

c. 运行以下命令验证您的应用程序资源是否已创建：

```
kubectl get Application
```

1.4.6.1. 订阅 YAML 结构

以下 YAML 结构显示订阅的必需字段，以及一些常见可选字段。您的 YAML 结构需要包含某些必需字段和值。

根据应用程序管理要求，您可能需要包含其他可选字段和值。您可以使用任何工具编写您自己的 YAML 内容：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name:
  namespace:
  labels:
spec:
  sourceNamespace:
  source:
  channel:
  name:
  packageFilter:
    version:
    labelSelector:
      matchLabels:
        package:
        component:
    annotations:
  packageOverrides:
  - packageName:
    packageAlias:
    - path:
      value:
  placement:
    local:
    clusters:
      name:
    clusterSelector:
    placementRef:
      name:
      kind: PlacementRule
  overrides:
    clusterName:
    clusterOverrides:
      path:
      value:
```

1.4.6.2. 订阅 YAML 表

字段	描述
apiVersion	必需。将值设为 **apps.open-cluster-management.io/v1**。
kind	必需。将值设为 **Subscription** 以表示资源是订阅。
metadata.name	必需。用于标识订阅的名称。
metadata.namespace	必需。用于订阅的命名空间资源。
metadata.labels	可选。订阅的标签。
spec.channel	可选。为订阅定义频道的命名空间名称（"Namespace/Name"）。定义 **channel**、**source** 或 **sourceNamespace** 字段。通常，使用 **channel** 字段来指向频道，而不要使用 **source** 或 **sourceNamespace** 字段。如果定义了多个字段，则会使用定义的第一个字段。
spec.sourceNamespace	可选。将可部署资源存储在 hub 集群上的源命名空间。仅将该字段用于命名空间频道。定义 **channel**、**source** 或 **sourceNamespace** 字段。通常，使用 **channel** 字段来指向频道，而不要使用 **source** 或 **sourceNamespace** 字段。
spec.source	可选。存储可部署资源的 Helm 仓库的路径名称 ("URL")。仅将该字段用于 Helm 仓库频道。定义 **channel**、**source** 或 **sourceNamespace** 字段。通常，使用 **channel** 字段来指向频道，而不要使用 **source** 或 **sourceNamespace** 字段。
spec.name	对于 **HelmRepo** 类型的频道是必需的，但对于 **ObjectBucket** 类型的频道是可选的。目标 Helm chart 或可部署资源在频道中的具体名称。对于该字段为可选的频道类型，如果没有定义 **name** 或 **packageFilter**，则会找到所有可部署资源，并检索每个可部署资源的最新版本。
spec.packageFilter	可选。定义用来查找目标可部署资源或一个可部署资源子集的参数。如果定义了多个过滤器条件，则可部署资源必须满足所有过滤器条件。
spec.packageFilter.version	可选。可部署资源的一个或多个版本。您可以使用 **>1.0** 或 **<3.0** 的形式来指定版本范围。默认情况下会使用带有最新 "creationTimestamp" 值的版本。
spec.packageFilter.annotations	可选。可部署资源的注解。

字段	描述
spec.packageOverrides	可选。用于为订阅中所订阅的 Kubernetes 资源（如 Helm chart、可部署资源或频道中的其他 Kubernetes 资源）定义覆盖的部分。
spec.packageOverrides.packageName	可选，但在设置覆盖时是必需的。标识将被覆盖的 Kubernetes 资源。
spec.packageOverrides.packageAlias	可选。为将被覆盖的 Kubernetes 资源指定别名。
spec.packageOverrides.packageOverrides	可选。用于覆盖 Kubernetes 资源的参数和替换值配置。
spec.placement	必需。标识需要放置可部署资源的订阅集群，或标识定义集群的放置规则。使用放置配置为多集群部署定义值。
spec.placement.local	可选，但对于独立集群或者您想要直接管理的集群是必需的。定义是否必须在本地部署订阅。将值设为 **true** 可将订阅与指定频道进行同步。将值设为 **false** 可防止在订阅中订阅指定频道中的任何资源。当集群属于独立集群或您将要直接管理此集群时，请使用此字段。如果您的集群是多集群的一部分，并且不想直接管理集群，则只使用 **clusters**、**clusterSelector** 或 **placementRef** 中的一个来定义您的订阅要放置的位置。如果您的集群是多集群的 Hub，而您想要直接管理集群，则在订阅 operator 可以在本地订阅资源前，必需将 Hub 注册为受管集群。
spec.placement.clusters	可选。定义要放置订阅的集群。仅使用 **clusters**、**clusterSelector** 或 **placementRef** 中的一个来为多集群定义要在哪里放置订阅。如果集群是一个不属于 hub 集群的独立集群，您也可以使用 **本地集群**。
spec.placement.clusters.name	可选，但在定义订阅集群时是必需的。订阅集群的一个或多个名称。
spec.placement.clusterSelector	可选。定义标签选择器，用于标识要放置订阅的集群。仅使用 **clusters**、**clusterSelector** 或 **placementRef** 中的一个来为多集群定义要在哪里放置订阅。如果集群是一个不属于 hub 集群的独立集群，您也可以使用 **本地集群**。
spec.placement.placementRef	可选。定义用于订阅的放置规则。仅使用 **clusters**、**clusterSelector** 或 **placementRef** 中的一个来为多集群定义要在哪里放置订阅。如果集群是一个不属于 Hub 集群的独立集群，您也可以使用 **本地集群**。

字段	描述
spec.placement.placementRef.name	可选，但在使用放置规则时是必需的。订阅的放置规则名称。
spec.placement.placementRef.kind	可选，但在使用放置规则时是必需的。将值设为 **PlacementRule** 以表示将使用放置规则以通过订阅进行部署。
spec.overrides	可选。需要覆盖的任何参数和值，如特定于集群的设置。
spec.overrides.clusterName	可选。参数和值将被覆盖的一个或多个集群的名称。
spec.overrides.clusterOverrides	可选。要覆盖的参数和值的配置。
spec.timeWindow	可选。定义用于在订阅处于活跃状态或受阻时配置时间窗的设置。
spec.timeWindow.type	可选，但在配置时间窗时是必需的。表示订阅在配置的时间窗内是处于活跃状态还是受阻。只有在订阅处于活跃状态时才会进行订阅的部署。
spec.timeWindow.location	可选，但在配置时间窗时是必需的。为时间窗配置的时间范围的时区。所有时区都必须使用 Time Zone (tz) 数据库名称格式。如需更多信息，请参阅Time Zone 数据库。
spec.timeWindow.daysofweek	可选，但在配置时间窗时是必需的。表示当使用了时间范围来创建时间窗时的每周天数。每周天数列表必须定义为数组，如 **daysofweek: ["Monday", "Wednesday", "Friday"]**。
spec.timeWindow.hours	可选，但在配置时间窗时是必需的。定义时间窗的时间范围。必须为每个时间窗定义小时范围的开始时间和结束时间。您可以为订阅定义多个时间窗范围。
spec.timeWindow.hours.start	可选，但在配置时间窗时是必需的。定义时间窗开始的时间戳。时间戳必须使用 Go 编程语言 Kitchen 格式 **"hh:mmpm"**。如需更多信息，请参阅 Constants。
spec.timeWindow.hours.end	可选，但在配置时间窗时是必需的。定义时间窗结束的时间戳。时间戳必须使用 Go 编程语言 Kitchen 格式 **"hh:mmpm"**。如需更多信息，请参阅 Constants。

备注：

- 在定义 YAML 时，订阅可以使用 **packageFilters** 来指向多个 Helm chart、可部署资源或其他 Kubernetes 资源。但是，订阅只会部署一个 chart、可部署资源或其他资源的最新版本。

- 对于时间窗，当您定义一个时间窗的时间范围时，必须将开始时间设置在结束时间之前。如果您要为订阅定义多个时间窗，时间窗的时间范围不能重叠。实际时间范围基于 **subscription-controller** 容器时间，可以设置为与您所在的时间和位置不同的时间和位置。

- 在订阅 spec 中，您还可以定义 Helm 发行版本的放置位置作为订阅定义的一部分。每个订阅都可以引用现有的放置规则，或者在订阅定义中直接定义放置规则。

- 当您要在 **spec.placement** 部分中定义订阅的放置位置时，对于多集群环境仅使用 **cclusters**、**clusterSelector** 或 **placementRef** 中的一个。

- 如果您包含多个放置设置，则会使用一个设置，其他设置将被忽略。以下优先级用于决定订阅 operator 使用哪个设置：

  a. **placementRef**

  b. **clusters**

  c. **clusterSelector**

您的订阅类似以下 YAML 内容：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement:
    placementRef:
      kind: PlacementRule
      name: towhichcluster
  overrides:
  - clusterName: "/"
    clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.4.6.3. 订阅文件示例

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
```

1.4.6.3.1. 订阅时间窗示例

以下示例订阅包含多个配置的时间窗。一个时间窗在每个周一、周三和周五的 10:20 AM 和 10:30 AM 之间发生。一个时间窗也在每周、周三和周五的 12:40 PM 到 1:40 PM 之间发生。订阅只在这 6 个每周时间窗内开始部署。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement:
    placementRef:
      kind: PlacementRule
      name: towhichcluster
  timewindow:
    windowtype: "active" #Enter active or blocked depending on the purpose of the type.
    location: "America/Los_Angeles"
    daysofweek: ["Monday", "Wednesday", "Friday"]
    hours:
      - start: "10:20AM"
        end: "10:30AM"
      - start: "12:40PM"
        end: "1:40PM"
```

1.4.6.3.2. 带有覆盖的订阅示例

以下示例包含软件包覆盖，以定义 Helm chart 的 Helm 发行版本的不同版本名称。软件包覆盖设置用于将名称 **my-nginx-ingress-releaseName** 设置为 **nginx-ingress** Helm 发行版本的不同发行版本名称。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: simple
  namespace: default
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: my-nginx-ingress-releaseName
    packageOverrides:
    - path: spec
      value:
        defaultBackend:
          replicaCount: 3
  placement:
    local: false
```

1.4.6.3.3. Helm 仓库订阅示例

以下订阅会自动拉取版本 **1.36.x** 的最新 **nginx** Helm 发行版本。当源 Helm 仓库中有新版本可用时，Helm 发行版本可部署资源会放置在 **my-development-cluster-1** 集群上。

spec.packageOverrides 部分显示了用于覆盖 Helm 发行版本值的可选参数。覆盖值合并到 Helm 发行版本 **values.yaml** 文件中，用于检索 Helm 发行版本的可配置变量。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
  placement:
    clusters:
    - name: my-development-cluster-1
  packageOverrides:
  - packageName: my-server-integration-prod
    packageOverrides:
    - path: spec
      value:
        persistence:
          enabled: false
          useDynamicProvisioning: false
        license: accept
        tls:
          hostname: my-mcm-cluster.icp
        sso:
          registrationImage:
            pullSecret: hub-repo-docker-secret
```

1.4.6.3.4. Git 仓库订阅示例

1.4.6.3.4.1. 订阅 Git 仓库的特定分支和目录

```
  apiVersion: apps.open-cluster-management.io/v1
  kind: Subscription
  metadata:
    name: sample-subscription
    namespace: default
    annotations:
      apps.open-cluster-management.io/git-path: sample_app_1/dir1
      apps.open-cluster-management.io/git-branch: branch1
  spec:
    channel: default/sample-channel
    placement:
```

```
    placementRef:
      kind: PlacementRule
      name: dev-clusters
```

在本示例订阅中，注解 **apps.open-cluster-management.io/git-path** 表示订阅中订阅了频道中指定的 Git 仓库 **sample_app_1/dir1** 目录中的所有 Helm chart 和 Kubernetes 资源。默认情况下会在订阅中订阅 **master** 分支（branch）。在本示例订阅中，指定了注解 **apps.open-cluster-management.io/git-branch: branch1** 来订阅仓库的 **branch1** 分支。

1.4.6.3.4.2. 添加 .kubernetesignore 文件

您可以在 Git 仓库根目录中，或者在订阅注解中指定的 **apps.open-cluster-management.io/git-path** 目录中包含 **.kubernetesignore** 文件。

您可以使用此 **.kubernetesignore** 文件指定文件和/或子目录模式，以在订阅部署仓库中的 Kubernetes 资源或 Helm chart 时忽略它们。

您还可以使用 **.kubernetesignore** 文件进行精细过滤，以选择性地应用 Kubernetes 资源。**.kubernetesignore** 文件的特征格式与 **.gitignore** 文件相同。

如果没有定义 **apps.open-cluster-management.io/git-path** 注解，订阅会在仓库根目录中查找 **.kubernetesignore** 文件。如果定义了 **apps.open-cluster-management.io/git-path** 字段，订阅会在 **apps.open-cluster-management.io/git-path** 目录中查找 **.kubernetesignore** 文件。订阅不会在任何其他目录中搜索 **.kubernetesignore** 文件。

1.4.6.3.4.3. 应用 Kustomize

如果订阅的 Git 文件夹中有 **kustomization.yaml** 或 **kustomization.yml** 文件，则会应用 kustomize。

您可以使用 **spec.packageOverrides** 在订阅部署时覆盖 **kustomization**。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: example-subscription
  namespace: default
spec:
  channel: some/channel
  packageOverrides:
  - packageName: kustomization
    packageOverrides:
    - value: |
patchesStrategicMerge:
- patch.yaml
```

为了覆盖 **kustomization.yaml** 文件，需要在 **packageOverrides** 中使用 **packageName: kustomization**。覆盖操作会添加新条目或更新现有条目。它不会移除现有的条目。

1.4.6.3.4.4. 启用 Git Webhook

默认情况下，Git 频道订阅每分钟克隆频道中指定的 Git 仓库，并在提交 ID 发生更改时应用更改。另外，您还可以将订阅配置为仅在 Git 仓库发送存储库 PUSH 和 PULL webhook 事件通知时应用更改。

要在 Git 仓库中配置 webhook，需要一个目标 webhook 有效负载 URL 和可选的 secret。

1.4.6.3.4.4.1. 有效负载（Payload） URL

在 hub 集群中创建路由（ingress），以公开订阅 operator 的 webhook 事件监听程序服务。

```
  oc create route passthrough --service=multicluster-operators-subscription -n open-cluster-
management
```

然后，使用 **oc get route multicluster-operators-subscription -n open-cluster-management** 命令查找外部可访问的主机名。webhook 有效负载 URL 是 **https://<externally-reachable hostname>/webhook**。

1.4.6.3.4.4.2. Webhook secret

Webhook secret 是可选的。在频道命名空间中创建 Kubernetes secret。secret 必须包含 **data.secret**。请参见以下示例：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-github-webhook-secret
data:
  secret: BASE64_ENCODED_SECRET
```

data.secret 的值是您要使用的 base-64 编码 WebHook secret。

最佳实践：为每个 Git 存储库使用唯一的 secret。

1.4.6.3.4.4.3. 在 Git 仓库中配置 WebHook

使用有效负载 URL 和 webhook secret 在 Git 仓库中配置 WebHook。

1.4.6.3.4.4.4. 在频道中启用 WebHook 事件通知

为订阅频道添加注解。请参见以下示例：

```
oc annotate channel.apps.open-cluster-management.io <channel name> apps.open-cluster-
management.io/webhook-enabled="true"
```

如果您使用了一个 secret 来配置 WebHook，也需要为频道添加该注解，其中 **<the_secret_name>** 是包含 webhook secret 的 kubernetes secret 名称。

```
oc annotate channel.apps.open-cluster-management.io <channel name> apps.open-cluster-
management.io/webhook-secret="<the_secret_name>"
```

1.4.6.3.4.4.5. 启用了 webhook 的频道的订阅

订阅中不需要特定于 webhook 的配置。

1.4.7. 放置规则示例

放置规则 (**placementrule.apps.open-cluster-management.io**) 定义的是可以部署可部署资源的目标集群。使用放置规则帮助您促进可部署资源的多集群部署。

要使用 Kubernetes CLI 工具，请参阅以下流程：

a. 使用您首选的编辑工具编写并保存您的应用程序 YAML 文件。

b. 运行以下命令，将文件应用到 API 服务器。将 **filename** 替换为您的文件名称：

```
kubectl apply -f filename.yaml
```

c. 运行以下命令验证您的应用程序资源是否已创建：

```
kubectl get Application
```

1.4.7.1. 放置规则 YAML 结构

以下 YAML 结构显示放置规则的必需字段，以及一些常见可选字段。您的 YAML 结构需要包含一些必需字段和值。根据应用程序管理要求，您可能需要包含其他可选字段和值。您可以使用任何工具编写您自己的 YAML 内容。

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
  name:
  namespace:
  resourceVersion:
  labels:
    app:
    chart:
    release:
    heritage:
  selfLink:
  uid:
spec:
  clusterSelector:
    matchLabels:
      datacenter:
      environment:
  clusterReplicas:
  clusterConditions:
  ResourceHint:
    type:
    order:
  Policies:
```

1.4.7.2. 放置规则 YAML 值表

字段	描述
apiVersion	必需。将值设为 **apps.open-cluster-management.io/v1**。
kind	必需。将值设为 **PlacementRule** 以表示资源是放置规则。

字段	描述
metadata.name	必需。用于标识放置规则的名称。
metadata.namespace	必需。用于放置规则的命名空间资源。
metadata.resourceVersion	可选。放置规则资源的版本。
metadata.labels	可选。放置规则的标签。
spec.clusterSelector	可选。用于标识目标集群的标签
spec.clusterSelector.matchLabels	可选。目标集群必须存在的标签。
status.decisions	可选。定义放置可部署资源的目标集群。
status.decisions.clusterName	可选。目标集群的名称
status.decisions.clusterNamespace	可选。目标集群的命名空间。
spec.clusterReplicas	可选。要创建的副本数量。
spec.clusterConditions	可选。为集群定义任何条件。
spec.ResourceHint	可选。如果多个集群与您在前面字段中提供的标签和值匹配，您可以指定特定于资源的条件来选择集群。例如，您可以选择包含最多可用 CPU 内核的集群。
spec.ResourceHint.type	可选。将值设为 **cpu** 以根据可用 CPU 内核选择集群，或者设为 **memory** 以根据可用内存资源选择集群。
spec.ResourceHint.order	可选。将值设为 **asc** 表示升序，或者设为 **desc** 表示降序。
spec.Policies	可选。放置规则的策略过滤器。

1.4.7.3. 放置规则示例文件

现有放置规则可以包括以下字段来指示放置规则的状态。此状态部分附加在规则的 YAML 结构中的 **spec** 部分后面。

```
status:
  decisions:
    clusterName:
    clusterNamespace:
```

字段	描述
status	放置规则的状态信息。
status.decisions	定义放置可部署资源的目标集群。
status.decisions.clusterName	目标集群的名称
status.decisions.clusterNamespace	目标集群的命名空间。

- 示例 1

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: gbapp-gbapp
  namespace: development
  labels:
    app: gbapp
spec:
  clusterSelector:
    matchLabels:
      environment: Dev
  clusterReplicas: 1
status:
  decisions:
    - clusterName: local-cluster
      clusterNamespace: local-cluster
```

- 示例 2

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: towhichcluster
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  clusterReplicas: 1
  clusterConditions:
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterSelector:
    matchExpressions:
    - key: environment
      operator: In
      values:
      - dev
```

1.4.8. 应用程序示例

查看可以用来构建文件的示例和 YAML 定义。Red Hat Advanced Cluster Management for Kubernetes 中的应用程序 (**Application.app.k8s.io**) 用于查看应用程序组件。

要使用 Kubernetes CLI 工具，请参阅以下流程：

a. 使用您首选的编辑工具编写并保存您的应用程序 YAML 文件。

b. 运行以下命令，将文件应用到 API 服务器。将 **filename** 替换为您的文件名称：

```
kubectl apply -f filename.yaml
```

c. 运行以下命令验证您的应用程序资源是否已创建：

```
kubectl get Application
```

1.4.8.1. 应用程序 YAML 结构

要编写用于创建或更新应用程序资源的应用程序定义 YAML 内容，YAML 结构需要包含一些必需字段和值。根据应用程序要求或应用程序管理要求，您可能需要包含其他可选字段和值。

以下 YAML 结构显示应用程序的必需字段，以及一些常见可选字段。

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name:
  namespace:
spec:
  selector:
    matchLabels:
      label_name: label_value
```

1.4.8.2. 应用程序 YAML 表

字段	值	描述
apiVersion	**app.k8s.io/v1beta1**	必需
kind	**Application**	必需
metadata		
	name ：用于标识应用程序资源的名称。	必需
	namespace ：用于应用程序的命名空间资源。	
spec		

字段	值	描述
selector.matchLabels	**key:value** 对，这是此应用程序关联的订阅或订阅上找到的 Kubernetes 标签和值。该标签允许应用程序资源通过执行标签名称和值匹配来查找相关订阅。	必需

定义这些应用程序的 spec 是基于 Kubernetes 特别兴趣小组 (SIG) 提供的应用程序元数据描述符自定义资源定义。只有在表中显示的值才是必需的。

您可以使用此定义来帮助编写自己的应用程序 YAML 内容。有关此定义的更多信息，请参阅 Kubernetes SIG 应用程序 CRD 社区规格。

1.4.8.3. 应用程序文件示例

应用程序的定义结构类似以下示例 YAML 内容：

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: my-application
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      my-label: my-label-value
```