Red Hat Advanced Cluster Management for Kubernetes 2.10

应用程序

应用程序管理

Last Updated: 2024-07-01

Red Hat Advanced Cluster Management for Kubernetes 2.10 应用程序

应用程序管理

法律通告

Copyright © 2024 Red Hat, Inc.

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

了解如何使用 Git 仓库、Helm 仓库和对象存储存储库创建应用程序。

目录

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。

重要：应用程序名称不能超过 37 个字符。

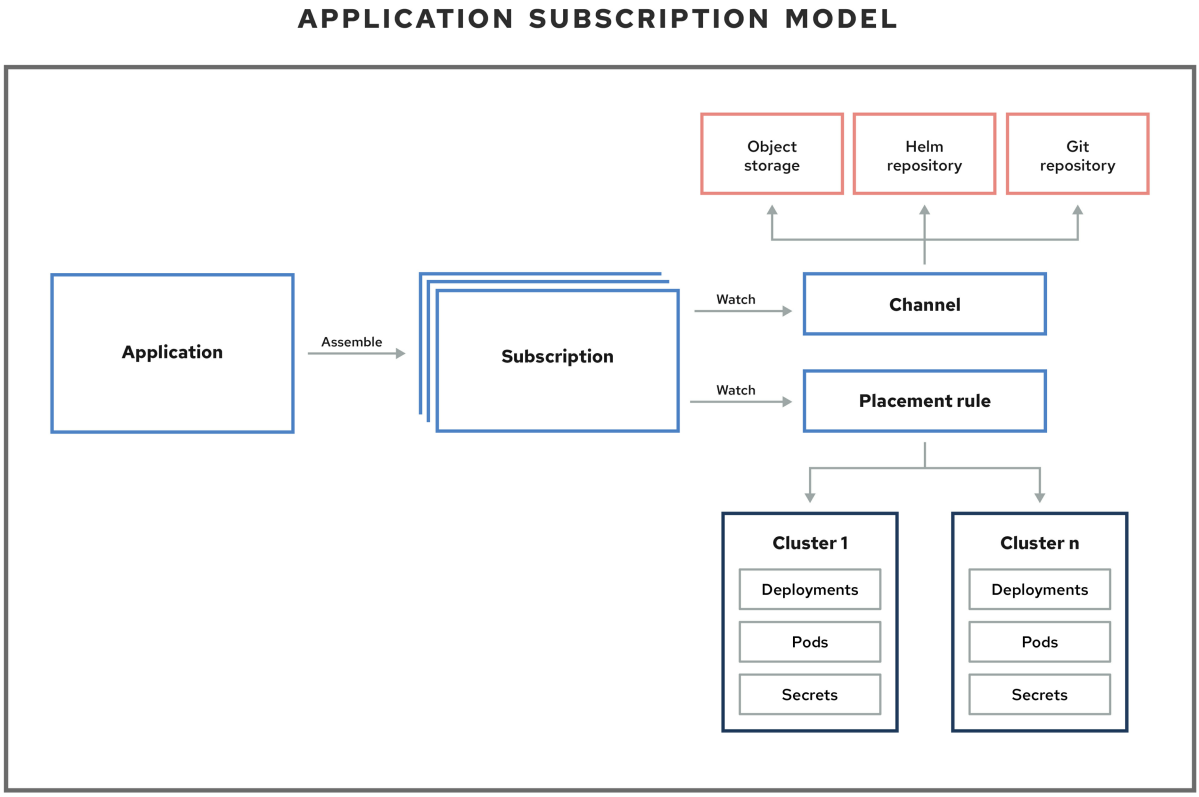
请参见以下主题：

- 应用程序模型和定义
  - 应用程序控制台
  - 订阅报告
  - 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
  - 授予订阅 admin 权限
  - 以订阅管理员身份创建允许和拒绝列表
  - 添加协调选项
  - 为安全 Git 连接配置应用程序频道和订阅
  - 设置 Ansible Automation Platform 任务
  - 调度部署
  - 配置软件包覆盖
  - 频道示例
  - 订阅示例
  - 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：



查看以下应用程序资源部分：

- 应用程序
- 订阅
- ApplicationSet
- 应用程序文档

最佳实践： 使用 GitOps Operator 或 Argo CD 集成，而不是 *Channel* 和 *Subscription* 模型。从 GitOps 概述 了解更多。

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，需要创建或编辑相应的部分，使其包含用于定义资源的标签。

您还可以使用 *发现的*应用程序，它们是 OpenShift Container Platform GitOps 或集群中安装的 Argo CD Operator 的应用程序。共享同一存储库的应用程序在此视图中分组在一起。

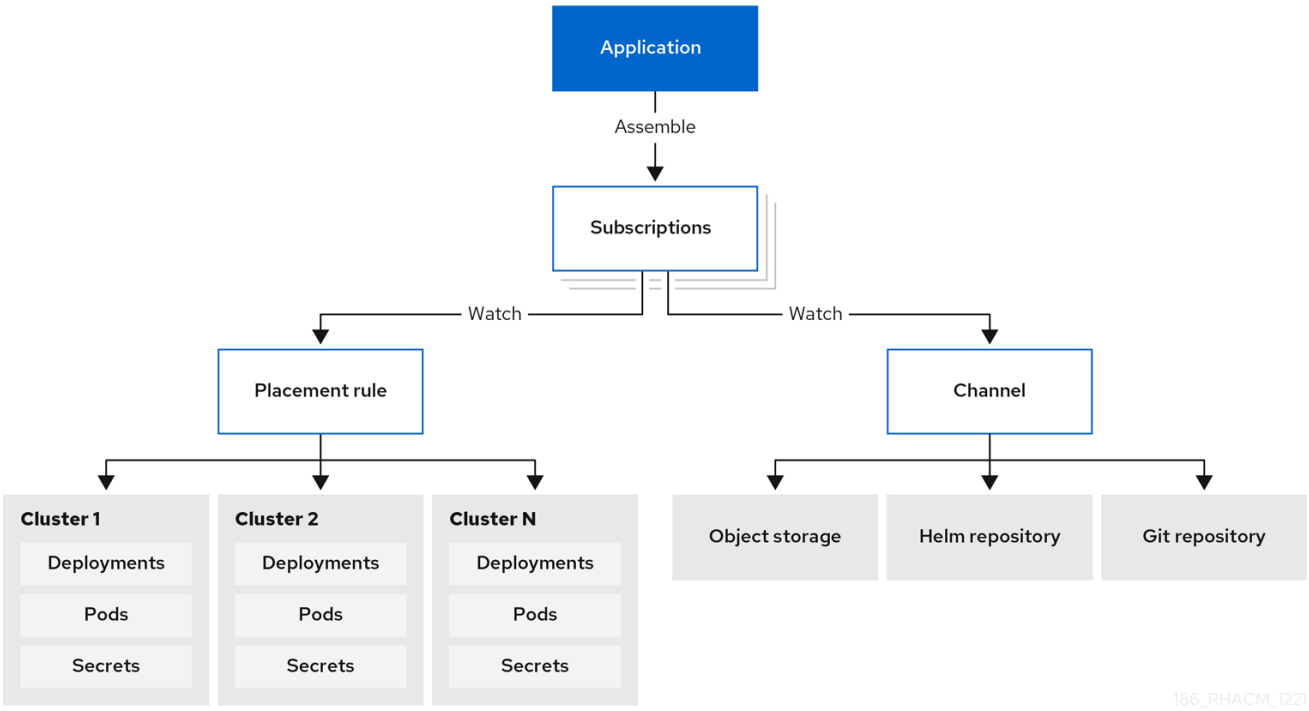
1.1.2. 订阅

订阅（**subscription.apps.open-cluster-management.io**）允许集群订阅到一个源仓库（频道），它可以是以下类型：Git 仓库、Helm 发行 registry 或 Object Storage 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** （自管理的 hub 集群）订阅。资源要求可能会对 hub 集群性能造成负面影响。

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。

请参阅以下订阅构架镜像：



1.1.2.1. Channels

频道（**channel.apps.open-cluster-management.io**）定义了集群可通过一个订阅来进行订阅的源仓库，它可以是以下类型：Git、Helm release 和 Object storage 仓库，以及 hub 集群上的资源模板。

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。

注： 最佳做法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。

1.1.2.1.1. 支持的 Git 存储库服务器

- GitHub
- GitLab

- Bitbucket
- Gogs

1.1.3. ApplicationSet

ApplicationSet 是 Argo CD 的子项目，由 GitOps Operator 支持。 **ApplicationSet** 添加了对 Argo CD 应用程序的多集群支持。您可从 Red Hat Advanced Cluster Management 控制台创建应用程序设置。

注：有关部署 **ApplicationSet** 的先决条件的详情，请参阅将受管集群注册到 GitOps。

OpenShift Container Platform GitOps 使用 Argo CD 来维护集群资源。Argo CD 是一个开源声明工具，用于应用程序的持续集成和持续部署（CI/CD）。OpenShift Container Platform GitOps 将 Argo CD 实现为一个控制器(OpenShift Container Platform GitOps Operator)，以便持续监控 Git 仓库中定义的应用程序定义和配置。然后，Argo CD 将这些配置的指定状态与集群中的实时状态进行比较。

ApplicationSet 控制器通过 GitOps operator 实例在集群中安装，并通过添加额外的功能来补充它，从而支持集群管理员的场景。**ApplicationSet** 控制器提供以下功能：

- 使用单个 Kubernetes 清单为使用 GitOps operator 的目标多个 Kubernetes 集群的功能。
- 可以使用单个 Kubernetes 清单使用 GitOps operator 从一个或多个 Git 存储库部署多个应用程序。
- 改进了对 monorepo 的支持，它位于 Argo CD 的上下文中，多个 Argo CD Application 资源在单个 Git 仓库中定义的多个 Argo CD Application 资源中。
- 在多租户集群中，改进了单个集群租户使用 Argo CD 部署应用程序的能力，而无需涉及特权集群管理员启用目标集群/命名空间。

ApplicationSet operator 利用集群决策生成器来接口 Kubernetes 自定义资源，这些资源使用自定义资源特定逻辑来决定要部署到的受管集群。集群决策资源会生成受管集群列表，然后呈现到 **ApplicationSet** 资源的 template 字段。这使用 duck-typing 来完成，不需要了解所引用的 Kubernetes 资源的完整知识。

请参阅 **ApplicationSet** 中的 **生成器.clusterDecisionResource** 值示例：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: sample-application-set
  namespace: sample-gitops-namespace
spec:
  generators:
    - clusterDecisionResource:
        configMapRef: acm-placement
        labelSelector:
          matchLabels:
            cluster.open-cluster-management.io/placement: sample-application-placement
        requeueAfterSeconds: 180
  template:
    metadata:
      name: sample-application-{{name}}
    spec:
      project: default
      sources: [
      {
```

```
      repoURL: https://github.com/sampleapp/apprepo.git
      targetRevision: main
      path: sample-application
       }
    ]
    destination:
      namespace: sample-application
      server: "{{server}}"
    syncPolicy:
      syncOptions:
        - CreateNamespace=true
        - PruneLast=true
        - Replace=true
        - ApplyOutOfSyncOnly=true
        - Validate=false
      automated:
        prune: true
        allowEmpty: true
        selfHeal: true
```

请参阅以下**放置** ：

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: sample-application-placement
  namespace: sample-gitops-namespace
spec:
  clusterSets:
    - sampleclusterset
```

如果要了解更多有关 **ApplicationSets** 的信息，请参阅 Cluster Decision Resource Generator。

1.1.4. 应用程序文档

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 Automation Platform 任务
- 频道示例
- 订阅示例

- 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。

请参阅以下列表中的一些控制台功能，并查看控制台以获取有关术语、操作以及如何阅读拓扑的指导信息：

重要：可用的操作基于您分配的角色。了解 基于角色的访问控制文档中的访问要求。

- 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。在 *Actions* 菜单中，您可以搜索、编辑或删除。在更新字段时，请确保选择 **YAML:On** 查看并编辑 YAML。
- 在主 *Overview* 选项卡中，您可以点应用程序名称来查看详情和应用程序资源，包括资源存储库、订阅、放置和部署的资源，如使用 Ansible Automation Platform 任务（用于 Git 存储库）的可选部署前和部署后 hook。您还可以从概述中创建应用程序。
- 创建并查看应用程序，如 *ApplicationSet*、*Subscription*、*OpenShift*、*Flux* 和 *Argo CD* 类型。**ApplicationSet** 代表从控制器生成的 Argo 应用程序。
  - 要创建 ArgoCD **ApplicationSet**，您需要从 **Sync 策略**中启用 **Automatically sync when cluster state changes**。
  - 对于带有 **kustomization** 控制器的 Flux，找到带有标签 **kustomize.toolkit.fluxcd.io/name=<app_name>** 的 Kubernetes 资源。
  - 对于带有 **helm** 控制器的 Flux，找到带有标签 **helm.toolkit.fluxcd.io/name=<app_name>** 的 Kubernetes 资源。
- 在主 *Overview* 中，当您点表中的应用程序名称来查看单个应用程序概述时，您可以查看以下信息：
- 集群详情，如资源状态
- 资源拓扑
- 订阅详情
- 访问 Editor 选项卡编辑
- 点 *Topology* 标签页显示项目中的所有应用程序和资源。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。
- 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、放置和频道。如果您有访问权限，还可以点多个 **Actions**，如 Edit、Search 和 Delete。
- 查看成功的 Ansible Automation Platform 部署，如果使用 Ansible 任务作为部署的应用程序的 prehook 或 posthook。
- 点 **Launch resource in Search** 搜索相关资源。
- 使用 *Search* 根据每个资源的组件 **kind** 查找应用程序资源。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**
Placement	**Placement**
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。有关使用搜索的更多信息，请参阅在控制台中进行搜索简介。

1.3. 订阅报告

订阅报告是来自您的所有受管集群的应用程序状态集合。具体来说，父应用程序资源可以从可扩展的受管集群保存报告。

受管集群的详细应用程序状态可用，而 hub 集群上的 **subscriptionReports** 则轻便且更具扩展性。请参阅以下三种类型的子状态报告：

- 软件包级的 **SubscriptionStatus** ：这是受管集群上的应用程序软件包状态，且应用程序在 **appsub** 命名空间中部署的所有资源的详细状态。
- 集群级 **SubscriptionReport** ：这是与特定集群部署的所有应用程序的整体状态报告。
- 应用程序级 **SubscriptionReport** ：这是部署特定应用程序的所有受管集群的总体状态报告。
  - SubscriptionStatus 软件包级别
  - SubscriptionReport 集群级别
  - SubscriptionReport 应用程序级别
  - managedClusterView
  - CLI 应用级别状态
  - CLI 最后更新时间

1.3.1. SubscriptionStatus 软件包级别

软件包级别的受管集群状态位于受管集群上的 **<namespace:<your-appsub-namespace>** 中，其中包含应用程序部署的所有资源的详细状态。对于部署到受管集群的每个 **appsub**，在受管集群的 **appsub** 命名空间中创建了 **SubscriptionStatus** CR。如果存在错误，则会报告每个资源，并带有详细错误。

软件包状态仅指示单个软件包的状态。您可以通过引用字段 **.status.subscription** 来查看整个订阅状态。

请参阅以下 **SubscriptionStatus** 示例 YAML 文件：

```
apiVersion: apps.open-cluster-management.io/v1alpha1
```

```
kind: SubscriptionStatus
metadata:
  labels:
    apps.open-cluster-management.io/cluster: <your-managed-cluster>
    apps.open-cluster-management.io/hosting-subscription: <your-appsub-namespace>.<your-
appsub-name>
  name: <your-appsub-name>
  namespace: <your-appsub-namespace>
statuses:
  packages:
  - apiVersion: v1
    kind: Service
    lastUpdateTime: "2021-09-13T20:12:34Z"
    Message: <detailed error. visible only if the package fails>
    name: frontend
    namespace: test-ns-2
    phase: Deployed
  - apiVersion: apps/v1
    kind: Deployment
    lastUpdateTime: "2021-09-13T20:12:34Z"
    name: frontend
    namespace: test-ns-2
    phase: Deployed
  - apiVersion: v1
    kind: Service
    lastUpdateTime: "2021-09-13T20:12:34Z"
    name: redis-master
    namespace: test-ns-2
    phase: Deployed
  - apiVersion: apps/v1
    kind: Deployment
    lastUpdateTime: "2021-09-13T20:12:34Z"
    name: redis-master
    namespace: test-ns-2
    phase: Deployed
  - apiVersion: v1
    kind: Service
    lastUpdateTime: "2021-09-13T20:12:34Z"
    name: redis-slave
    namespace: test-ns-2
    phase: Deployed
  - apiVersion: apps/v1
    kind: Deployment
    lastUpdateTime: "2021-09-13T20:12:34Z"
    name: redis-slave
    namespace: test-ns-2
    phase: Deployed
subscription:
  lastUpdateTime: "2021-09-13T20:12:34Z"
  phase: Deployed
```

1.3.2. SubscriptionReport 集群级别

集群级状态位于 hub 集群上的 & **lt;namespace:<your-managed-cluster-** 1> 中，仅包含该受管集群上每个应用程序的整体状态。hub 集群上的每个集群命名空间中的 **subscriptionReport** 报告以下状态之一：

- **Deployed**
- **Failed**
- **propagationFailed**

请参阅以下 **SubscriptionStatus** 示例 YAML 文件：

```
apiVersion: apps.open-cluster-management.io/v1alpha1
kind: subscriptionReport
metadata:
  labels:
    apps.open-cluster-management.io/cluster: "true"
  name: <your-managed-cluster-1>
  namespace: <your-managed-cluster-1>
reportType: Cluster
results:
- result: deployed
  source: appsub-1-ns/appsub-1                    // appsub 1 to <your-managed-cluster-1>
  timestamp:
    nanos: 0
    seconds: 1634137362
- result: failed
  source: appsub-2-ns/appsub-2                    // appsub 2 to <your-managed-cluster-1>
  timestamp:
    nanos: 0
    seconds: 1634137362
- result: propagationFailed
  source: appsub-3-ns/appsub-3                    // appsub 3 to <your-managed-cluster-1>
  timestamp:
    nanos: 0
    seconds: 1634137362
```

1.3.3. SubscriptionReport 应用程序级别

每个应用程序有一个应用程序级 **subscriptionReport**，位于 hub 集群的 **appsub** 命名空间中的 **<namespace:<your-appsub-namespace>**，并包含以下信息：

- 每个受管集群的应用程序的整体状态
- 应用程序所有资源的列表
- 带有集群总数的报告概述
- 一个报告摘要，其中包含应用程序处于状态的集群总数：**deployed**, **failed**, **propagationFailed**, 和 **inProgress**。

备注： inProcess 状态是总数减 **deployed**, 减 **failed `, 并减 `propagationFailed**。

请参阅以下 **SubscriptionStatus** 示例 YAML 文件：

```
apiVersion: apps.open-cluster-management.io/v1alpha1
kind: subscriptionReport
metadata:
  labels:
```

```
    apps.open-cluster-management.io/hosting-subscription: <your-appsub-namespace>.<your-
appsub-name>
  name: <your-appsub-name>
  namespace: <your-appsub-namespace>
reportType: Application
resources:
- apiVersion: v1
  kind: Service
  name: redis-master2
  namespace: playback-ns-2
- apiVersion: apps/v1
  kind: Deployment
  name: redis-master2
  namespace: playback-ns-2
- apiVersion: v1
  kind: Service
  name: redis-slave2
  namespace: playback-ns-2
- apiVersion: apps/v1
  kind: Deployment
  name: redis-slave2
  namespace: playback-ns-2
- apiVersion: v1
  kind: Service
  name: frontend2
  namespace: playback-ns-2
- apiVersion: apps/v1
  kind: Deployment
  name: frontend2
  namespace: playback-ns-2
results:
- result: deployed
  source: cluster-1                    //cluster 1 status
  timestamp:
    nanos: 0
    seconds: 0
- result: failed
  source: cluster-3                    //cluster 2 status
  timestamp:
    nanos: 0
    seconds: 0
- result: propagationFailed
  source: cluster-4                    //cluster 3 status
  timestamp:
    nanos: 0
    seconds: 0
summary:
  deployed: 8
  failed: 1
  inProgress: 0
  propagationFailed: 1
  clusters: 10
```

1.3.4. ManagedClusterView

在第一个失败的集群上报告 **ManagedClusterView** CR。如果应用程序在带有资源部署失败的多个集群中部署，则只会为 hub 集群上第一个失败的集群命名空间创建一个 **managedClusterView** CR。**managedClusterView** CR 从失败集群中检索详细的订阅状态，因此应用程序所有者不需要访问失败的远程集群。

运行以下命令，您可以运行以下命令获取状态：

```
% oc get managedclusterview -n <failing-clusternamespace> "<app-name>-<app name>"
```

1.3.5. CLI 应用级别状态

如果无法访问受管集群来获取订阅状态，您可以使用 CLI。集群级别或应用程序级别订阅报告提供了整个状态，而不是应用程序的详细错误消息。

1. 从 multicloud-operators-subscription 下载 CLI。
2. 运行以下命令来创建 **managedClusterView** 资源来查看受管集群应用程序 **SubscriptionStatus**，以便您可以识别错误：

```
% getAppSubStatus.sh -c <your-managed-cluster> -s <your-appsub-namespace> -n <your-appsub-name>
```

1.3.6. CLI 最后更新时间

当给定受管集群上无法登录到每个受管集群以检索这些信息时，您还可以获得给定受管集群上 AppSub 的 Last Update Time。因此，创建了一个实用程序脚本，以简化受管集群中 AppSub 的 Last Update Time 的检索。这个脚本设计为在 Hub 集群上运行。它会创建一个 managedClusterView 资源，以从受管集群获取 AppSub，并解析数据以获取 Last Update Time。

1. 从 multicloud-operators-subscription 下载 CLI。
2. 运行以下命令，以检索受管集群中 **AppSub** 的 **Last Update Time**。这个脚本设计为在 hub 集群上运行。它会创建一个 **managedClusterView** 资源，以从受管集群获取 AppSub，并解析数据以获取 Last Update Time：

```
% getLastUpdateTime.sh -c <your-managed-cluster> -s <your-appsub-namespace> -n <your-appsub-name>
```

1.4. 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

1.4.1. 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。

用户需要访问权限： 可以创建应用程序的用户角色。您只能执行分配了相关角色的操作。了解 基于角色的访问控制文档中的访问要求。

1. 在控制台导航菜单中点 **Applications** 来查看列出的应用程序并创建新应用。
2. **可选：** 在选择要创建的应用程序类型后，您可以选择 **YAML: On** 在控制台上按照创建和编辑应用程序查看 YAML。请参阅后面的 YAML 示例。
3. 从可以使用的存储库列表中选择 **Git**，并在正确的字段中输入值。按照控制台中的指南，根据您的输入查看 YAML 编辑器更改值。
   备注：
   - 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）。
   - 请注意协调选项。**merge** 选项是默认选择，这代表要添加新字段，并在资源中更新现有字段。您可以选择 **replace**。使用 **replace** 选项，现有资源被替换为 Git 源。当订阅协调率被设置为 **low** 时，订阅的应用程序资源最多可能需要一小时才能完成协调。在单个应用程序视图的卡上，单击 **Sync** 以手动协调。如果设为 **off**，则永远不会协调。
4. 设置任何可选的部署前和部署后的任务。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 Automation Platform 作业，则设置 Ansible Automation Platform secret。定义作业的 Ansible Automation Platform 任务必须放置在此存储库的 **prehook** 和 **posthook** 文件夹中。
5. 如果需要使用控制台添加凭证，可以点 **Add credential**。按照控制台中的指示进行操作。请参阅管理凭证概述。
6. 点 **Create**。
7. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.4.1.1. 更多示例

- 有关 **root-subscription/** 的示例，请参阅 **application-subscribe-all**。
- 有关指向同一仓库中其他文件夹的订阅示例，请参阅 **subscribe-all**。
- 请参阅 **nginx-apps** 存储库中带有应用程序工件的 **common-managed** 文件夹示例。
- 请参阅策略集合中的策略示例。

1.4.1.2. 使用 Git 删除订阅后保留部署的资源

使用 Git 存储库创建订阅时，您可以添加 **do-not-delete** 注解，以便在删除订阅后保留特定的部署资源。**do-not-delete** 注解仅适用于顶级部署资源。要添加 **do-not-delete** 注解，请完成以下步骤：

1. 创建至少部署一个资源的订阅。
2. 在您要保留的资源或资源中添加以下注解，即使您删除订阅：
   apps.open-cluster-management.io/do-not-delete: 'true'

请参见以下示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    apps.open-cluster-management.io/do-not-delete: 'true'
    apps.open-cluster-management.io/hosting-subscription: sub-ns/subscription-example
    apps.open-cluster-management.io/reconcile-option: merge
    pv.kubernetes.io/bind-completed: "yes"
```

删除订阅后，带有 **do-not-delete** 注解的资源仍然存在，而其他资源也会被删除。

注： 使用 **do-not-delete** 注解绑定到命名空间来保持部署的资源。因此，在删除剩余的资源前，您无法删除命名空间。

1.4.2. 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。

用户需要访问权限： 可以创建应用程序的用户角色。您只能执行分配了相关角色的操作。了解 基于角色的访问控制文档中的访问要求。

1. 在控制台导航菜单中点 **Applications** 来查看列出的应用程序并创建新应用。
2. **可选：** 在选择要创建的应用程序类型后，您可以选择 **YAML: On** 在控制台上按照创建和编辑应用程序查看 YAML。请参阅后面的 YAML 示例。
3. 从可以使用的存储库列表中选择 **Helm**，并在正确的字段中输入值。按照控制台中的指南，根据您的输入查看 YAML 编辑器更改值。
4. 点 **Create**。
5. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.4.2.1. YAML 示例

以下示例频道定义将 Helm 仓库抽象为频道：

注： 对于 Helm，Helm chart 中包含的所有 Kubernetes 资源都必须具有标签发行版本。 **{{ .Release.Name }}`** 用于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/
```

注： 要查看 REST API，使用 API。

1.4.2.2. 使用 Helm 删除订阅后保留部署的资源

Helm 提供了一个注解，用于在删除订阅后保留特定部署的资源。如需更多信息，请参阅 Tell Helm Not Uninstall a Resource。

注： 注解必须位于 Helm chart 中。

1.4.3. 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。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：

用户需要访问权限： 可以创建应用程序的用户角色。您只能执行分配了相关角色的操作。了解 基于角色的访问控制文档中的访问要求。

1. 在控制台导航菜单中点 **Applications** 来查看列出的应用程序并创建新应用。
2. **可选：** 在选择要创建的应用程序类型后，您可以选择 **YAML: On** 在控制台上按照创建和编辑应用程序查看 YAML。请参阅后面的 YAML 示例。
3. 从可以使用的存储库列表中选择 **Object store**，并在正确的字段中输入值。按照控制台中的指南，根据您的输入查看 YAML 编辑器更改值。
4. 点 **Create**。
5. 您会被重定向到 *Overiew* 页，可以在其中查看详情和拓扑。

1.4.3.1. 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://sample-ip:#####/dev] # URL is appended with the valid bucket name, which
matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

注： 要查看 REST API，使用 API。

1.4.3.2. 创建 Amazon Web Services (AWS) S3 对象存储存储桶

您可以设置订阅来订阅在 Amazon Simple Storage Service (Amazon S3) 对象存储服务中定义的资源。请参见以下步骤：

1. 使用您的 AWS 帐户、用户名和密码登录到 AWS 控制台。
2. 进入 **Amazon S3** > **Buckets** 到存储桶主页。
3. 点击 **Create Bucket** 创建存储桶。
4. 选择 **AWS region**，这对连接 AWS S3 对象存储桶至关重要。
5. 创建存储桶访问令牌。
6. 导航到导航栏中的用户名，然后从下拉菜单中选择 **My Security Credentials**。
7. 进入 *AWS IAM credentials* 标签页中的 *Access keys for CLI, SDK, & API access*，点 **Create access key**。
8. 保存您的 *Access key ID*、*Secret access key*。
9. 将对象 YAML 文件上传到存储桶。

1.4.3.3. 订阅 AWS 存储桶中的对象

1. 创建一个带有 secret 的对象存储桶类型频道，以指定用于连接 AWS bucket 的 *AccessKeyID*、*SecretAccessKey* 和 *Region*。创建 AWS 存储桶时会创建这三个字段。
2. 添加 URL。如果 URL 包含 **s3://** 或 **s3 and aws** 关键字，则 URL 用来标识 AWS S3 存储桶中的频道。例如，请查看以下所有存储桶 URL 都有 AWS s3 存储桶标识符：

   ```
   https://s3.console.aws.amazon.com/s3/buckets/sample-bucket-1
   s3://sample-bucket-1/
   https://sample-bucket-1.s3.amazonaws.com/
   ```

   注： 不需要 AWS S3 对象存储桶 URL 来将存储桶与 AWS S3 API 连接。

1.4.3.4. AWS 订阅示例

请参阅以下完整的 AWS S3 对象存储桶频道示例 YAML 文件：

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

```
kind: Channel
metadata:
  name: object-dev
  namespace: ch-object-dev
spec:
  type: ObjectBucket
  pathname: https://s3.console.aws.amazon.com/s3/buckets/sample-bucket-1
  secretRef:
    name: secret-dev
---
apiVersion: v1
kind: Secret
metadata:
  name: secret-dev
  namespace: ch-object-dev
stringData:
  AccessKeyID: <your AWS bucket access key id>
  SecretAccessKey: <your AWS bucket secret access key>
  Region: <your AWS  bucket region>
type: Opaque
```

弃用： 您可以继续创建其他 AWS 订阅和放置规则对象，如以下示例 YAML 添加了 **kind: PlacementRule** 和 **kind: Subscription**：

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: towhichcluster
  namespace: obj-sub-ns
spec:
  clusterSelector: {}
---
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: obj-sub
  namespace: obj-sub-ns
spec:
  channel: ch-object-dev/object-dev
  placement:
    placementRef:
      kind: PlacementRule
      name: towhichcluster
```

您还可以订阅对象存储桶中特定子文件夹内的对象。将 **subfolder** 注解添加到订阅中，它会强制对象存储桶订阅仅应用子文件夹路径中的所有资源。

请参阅带有 **subfolder-1** 的注解作为 **bucket-path**：

```
annotations:
  apps.open-cluster-management.io/bucket-path: <subfolder-1>
```

有关子文件夹，请参见以下完整示例：

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

```
kind: Subscription
metadata:
  annotations:
    apps.open-cluster-management.io/bucket-path: subfolder1
  name: obj-sub
  namespace: obj-sub-ns
  labels:
    name: obj-sub
spec:
  channel: ch-object-dev/object-dev
  placement:
    placementRef:
      kind: PlacementRule
      name: towhichcluster
```

1.4.3.5. 使用对象存储删除订阅后保留部署的资源

使用对象存储存储库创建订阅时，您可以添加 **do-not-delete** 注解，以便在删除订阅后保留特定的部署资源。**do-not-delete** 注解仅适用于顶级部署资源。要添加 **do-not-delete** 注解，请完成以下步骤：

1. 创建至少部署一个资源的订阅。
2. 在您要保留的资源或资源中添加以下注解，即使您删除订阅：
   apps.open-cluster-management.io/do-not-delete: 'true'

   请参见以下示例：

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     annotations:
       apps.open-cluster-management.io/do-not-delete: 'true'
       apps.open-cluster-management.io/hosting-subscription: sub-ns/subscription-example
       apps.open-cluster-management.io/reconcile-option: merge
       pv.kubernetes.io/bind-completed: "yes"
   ```

删除订阅后，带有 **do-not-delete** 注解的资源仍然存在，而其他资源也会被删除。

注： 使用 **do-not-delete** 注解绑定到命名空间来保持部署的资源。因此，在删除剩余的资源前，您无法删除命名空间。

1.5. ANSIBLE AUTOMATION PLATFORM 集成和简介

Red Hat Advanced Cluster Management 与 Red Hat Ansible Automation Platform 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。了解组件以及如何配置 Ansible Automation Platform。

需要的访问权限：集群管理员

1.5.1. 集成和组件

您可以将 Ansible Automation Platform 作业集成到 Git 订阅中。例如，对于数据库前端和后端应用程序，需要使用带有 Ansible Automation Platform 作业的 Ansible Automation Platform 来实例化数据库。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，以定义两个名为 **prehook** 和 **posthook** 的子文件夹。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 Automation Platform 作业。

创建 Git 订阅时，所有 prehook 和 posthook **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 prehook 和 posthook **AnsibleJob** 实例。

当您创建订阅自定义资源时，Git 分支和 Git 路径指向 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**。

1.5.1.1. 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。

请参见以下实例名称示例：**database-sync-1-2913063**。

应用程序订阅控制器在一分钟循环中再次对协调请求进行队列，它会在 prehook **AnsibleJob** **status.AnsibleJobResult** 中检查。当 prehook 的状态为 **successful** 时，应用程序订阅将继续部署主订阅。

1.5.1.2. Posthook

更新应用程序订阅状态时，如果订阅状态已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** 自定义资源作为 posthook **AnsibleJob** 对象搜索。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。

请参见以下实例名称示例：**service-ticket-1-2913849**。

请参阅以下主题以启用 {aap_short}：

- 设置 Ansible Automation Platform
- 配置 Ansible Automation Platform

1.5.2. 设置 Ansible Automation Platform

使用 Ansible Automation Platform 作业，您可以自动执行任务并与外部服务（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 Automation Platform 作业的 **prehook** 和 **posthook** 目录。

需要的访问权限：集群管理员

- 先决条件
- 安装 Ansible Automation Platform Resource Operator

1.5.2.1. 先决条件

- 安装 OpenShift Container Platform 4.13 或更高版本。
- 安装 Ansible Automation Platform。请参阅 Red Hat Ansible Automation Platform 文档来安装最新支持的版本。

- 安装 Ansible Automation Platform Resource Operator，将 Ansible Automation Platform 作业连接到 Git 订阅的生命周期。**最佳实践：** Ansible Automation Platform 作业模板在运行时应该是幂等的。
- 在模板上为 **INVENTORY** 和 **EXTRA VARIABLES** 选择 **PROMPT ON LAUNCH**。如需更多信息，请参阅 作业模板。

1.5.2.2. 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*。**备注：**要提交 prehook 和 posthook **AnsibleJobs**，请使用以下 OpenShift Container Platform 版本上的相应版本安装 Red Hat Ansible Automation Platform Resource Operator：
   - OpenShift Container Platform 4.8 需要(AAP)Resource Operator early-access、stable-2.1、stable-2.2
   - OpenShift Container Platform 4.9 需要(AAP)Resource Operator early-access、stable-2.1、stable-2.2
   - OpenShift Container Platform 4.10 及更新的版本需要(AAP) Resource Operator stable-2.1, stable-2.2

然后，您可以从控制台的 *Credentials* 页面创建凭证。点 **Add credential**，或者从导航中访问页面。如需更多信息，请参阅为 Ansible Automation Platform 创建凭证。

1.5.3. 配置 Ansible Automation Platform

> With {aap-short} jobs, you can automate tasks and integrate with external services, such as Slack and PagerDuty services. Your Git repository resource root path will contain `prehook` and `posthook` directories for {aap-short} jobs that run as part of deploying the application, updating the application, or removing the application from a cluster.

需要的访问权限：集群管理员

- 设置 Ansible Automation Platform secret
- 设置 secret 协调
- 使用 Ansible Automation Platform 示例 YAML 文件
- 启动工作流

您可以使用以下任务配置 Ansible Automation Platform 配置：

1.5.3.1. 设置 Ansible Automation Platform secret

您必须在同一订阅命名空间中创建 Ansible Automation Platform secret 自定义资源。Ansible Automation Platform secret 仅限于相同的订阅命名空间。

1. 通过填写 **Ansible Automation Platform secret name** 部分，从控制台创建 secret。要使用终端创建 secret，请编辑并应用示例 **yaml** 文件：

注： namespace 与订阅命名空间相同。**stringData:token** 和 **host** 来自 Ansible Automation Platform。

```
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
```

2. 运行以下命令来添加 YAML 文件：

```
oc apply -f
```

当应用程序订阅控制器创建 prehook 和 posthook Ansible 作业时，如果订阅 **spec.hooksecretref** 中的 secret 可用，则会将其发送到 **AnsibleJob** 自定义资源 **spec.tower_auth_secret**，**AnsibleJob** 可以访问 Ansible Automation Platform。

1.5.3.2. 设置 secret 协调

对于使用 prehook 和 posthook **AnsibleJob** 的主要订阅，在 Git 存储库中更新所有 prehook 和 posthook **AnsibleJob** 或主订阅后，应协调主订阅。

prehook **AnsibleJob** 和主订阅持续协调并重新启动新的 pre **AnsibleJob** 实例。

1. 在 pre **AnsibleJob** 完成后，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 集群订阅状态重置为 **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. 主订阅完成后，重新启动一个新的 post-**AnsibleJob** 实例。
5. 验证订阅是否已更新。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** 自定义资源时，会创建一个 Kubernetes 作业自定义资源，以通过与目标 Ansible Automation Platform 通信来启动 Ansible Automation Platform 作业。作业完成后，作业的最终状态将返回到 **AnsibleJob status.AnsibleJob Result**。

备注：

Ansible Automation Platform Job operator 保留 **AnsibleJob status.conditions**，以存储 Kubernetes 作业结果的创建。**status.conditions** 不会反映实际 Ansible Automation Platform 作业状态。

订阅控制器根据 **AnsibleJob.status.AnsibleJob.Result** 而不是 **AnsibleJob.status.conditions** 检查 Ansible Automation Platform 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 Ansible 作业
- **prehookJobsHistory** ：所有 prehook Ansible 作业历史记录
- **lastPosthookJobs** ：最新的 posthook Ansible 作业
- **posthookJobsHistory** ：所有 posthook Ansible 作业历史记录

1.5.3.3. 使用 Ansible Automation Platform 示例 YAML 文件

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:
  name: 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
  job_tags: "provision,install,configuration"
  skip_tags: "configuration,restart"
```

1.5.3.4. 启动工作流

要使用 **AnsibleJob** 自定义资源启动 Ansible Automation Platform 工作流，请将 **job_template_name** 字段替换为 **workflow_template_name** 字段，如下例所示。

1.5.3.5. 使用 Ansible Automation Platform 示例 YAML 工作流

请参阅 Git prehook 和 Git posthook 文件夹中的工作流 **AnsibleJob** YAML 文件示例：

```
apiVersion: tower.ansible.com/v1alpha1
kind: AnsibleJob
metadata:
  name: demo-job-001
  namespace: default
spec:
```

```
tower_auth_secret: toweraccess
workflow_template_name: Demo Workflow 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
```

请参阅工作流以了解更多有关 Ansible 工作流的信息。

1.6. 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，需要创建或编辑相应的部分，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
- 授予订阅 admin 权限
- 以订阅管理员身份创建允许和拒绝列表
- 添加协调选项
- 配置领导选举机制
- 为安全 Git 连接配置应用程序频道和订阅
- 设置 Ansible Automation Platform 任务
- 配置 Helm 以监视命名空间资源
- 配置软件包覆盖
- 频道示例概述
- 订阅示例概述
- 应用程序示例概述

1.6.1. 订阅 Git 资源

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。*订阅*管理员可以更改默认行为，如 授予订阅管理员权限所述。

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

- 在 Git 中创建应用程序资源
- 订阅特定的 Git 元素
- 应用程序命名空间示例
- 资源覆盖示例

1.6.1.1. 在 Git 中创建应用程序资源

您需要在订阅时在资源 YAML 中指定 **apiVersion** 的完整组和版本。例如，如果订阅 **apiVersion: v1**，订阅控制器无法验证订阅，您收到以下错误信息：**Resource /v1, Kind=ImageStream is not supported**。

如果 **apiVersion** 被改为 **image.openshift.io/v1**，如以下示例所示，它会在订阅控制器中传递验证，并且资源被成功应用。

```
apiVersion: `image.openshift.io/v1`
kind: ImageStream
metadata:
  name: default
  namespace: default
spec:
  lookupPolicy:
    local: true
  tags:
    - name: 'latest'
      from:
        kind: DockerImage
        name: 'quay.io/repository/open-cluster-management/multicluster-operators-subscription:community-latest'
```

接下来，请参阅订阅管理员如何更改默认行为的更多示例。

1.6.1.2. 应用程序命名空间示例

在以下示例中，您以订阅管理员身份登录。

1.6.1.2.1. 应用到不同的命名空间

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。

1.6.1.2.2. 应用到同一命名空间

作为订阅管理员，您可能希望将所有应用程序资源部署到同一命名空间中。

您可以通过创建一个允许和拒绝列表作为订阅管理员，将所有应用程序资源部署到订阅命名空间中。

将 **apps.open-cluster-management.io/current-namespace-scoped: true** 注解添加到订阅 YAML。例如，当订阅管理员创建以下订阅时，上例中的所有三个 ConfigMap 都在 **subscription-ns** 命名空间中创建。

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

1.6.1.3. 资源覆盖示例

适用的频道类型： Git、ObjectBucket（控制台中的对象存储）

注： 资源覆盖选项不适用于 Git 存储库中的 **helm** chart，因为 **helm** chart 资源由 Helm 管理。

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。查看 **data** 规格中的变化：

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

1.6.1.3.1. 默认合并选项

请参阅带有默认 **apps.open-cluster-management.io/reconcile-option: merge** 注解的 Git 存储库中的以下示例资源 YAML 文件。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。

1.6.1.3.2. mergeAndOwn 选项

使用 **mergeAndOwn** 时，订阅的资源中的条目会在现有资源中创建或更新。作为订阅管理员登录，并使用 **apps.open-cluster-management.io/reconcile-option: mergeAndOwn** 注解创建一个订阅。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: mergeAndOwn
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
  annotations:
    apps.open-cluster-management.io/hosting-subscription: sub-ns/subscription-example
data:
  name: user1
  age: 20
```

如前所述，当使用 **mergeAndOwn** 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。它还添加了 **apps.open-cluster-management.io/hosting-subscription** 注解，以指示资源现在归订阅所有。删除订阅会删除 ConfigMap。

1.6.1.3.3. 替换选项

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
```

1.6.1.4. 订阅特定的 Git 元素

您可以订阅特定 Git 分支、提交或标签。

1.6.1.4.1. 订阅特定分支

multicloud-operators-subscription 仓库中包含的订阅 operator 订阅 Git 仓库的默认分支。如果要订阅到不同的分支，则需要在订阅中指定分支名称注解。

以下示例 YAML 文件演示了如何通过 **apps.open-cluster-management.io/git-branch: <branch1>** 指定不同的分支：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: git-mongodb-subscription
  annotations:
    apps.open-cluster-management.io/git-path: stable/ibm-mongodb-dev
    apps.open-cluster-management.io/git-branch: <branch1>
```

1.6.1.4.2. 订阅特定提交

multicloud-operators-subscription 仓库中包含的订阅 operator 默认订阅 Git 仓库的指定分支的最新提交。如果要订阅特定提交，则需要使用订阅中的提交散列（commit hash）指定所需的提交注解。

在以下示例中，YAML 文件演示了如何通过 **apps.open-cluster-management.io/git-desired-commit: <full commit number>** 指定不同的提交：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: git-mongodb-subscription
  annotations:
```

```
    apps.open-cluster-management.io/git-path: stable/ibm-mongodb-dev
    apps.open-cluster-management.io/git-desired-commit: <full commit number>
    apps.open-cluster-management.io/git-clone-depth: 100
```

git-clone-depth 注解是可选的，默认设置为 **20**。这意味着订阅控制器会从 Git 存储库检索前 20 个提交历史记录。如果您指定了一个旧的 **git-desired-commit**，则需要为所需的提交相应地指定 **git-clone-depth**。

1.6.1.4.3. 订阅特定标签

multicloud-operators-subscription 仓库中包含的订阅 operator 默认订阅 Git 仓库的指定分支的最新提交。如果要订阅特定标签，则需要在订阅中指定标签注解。

以下示例显示，YAML 文件显示如何通过 **apps.open-cluster-management.io/git-tag: <v1.0>** 指定不同的标签：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: git-mongodb-subscription
  annotations:
    apps.open-cluster-management.io/git-path: stable/ibm-mongodb-dev
    apps.open-cluster-management.io/git-tag: <v1.0>
    apps.open-cluster-management.io/git-clone-depth: 100
```

注： 如果 Git 所需的提交和标签注解都被指定，标签将被忽略。

git-clone-depth 注解是可选的，默认设置为 **20**。这意味着订阅控制器从 Git 仓库检索前 **20** 个提交历史记录。如果指定了旧的 **git-tag**，则需要为标签的所需提交相应地指定 **git-clone-depth**。

1.6.2. 授予订阅管理员权限

了解如何授予订阅管理员访问权限。*订阅管理员*可以更改默认行为。在以下过程中了解更多信息：

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。您还可以准备组或服务帐户。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. 如果 **open-cluster-management:subscription-admin** ClusterRoleBinding 不存在，您需要创建它。请参见以下示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: open-cluster-management:subscription-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: open-cluster-management:subscription-admin
```

5. 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
```

注： 在初始情况下， **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
- kind: ServiceAccount
  name: my-service-account
  namespace: my-service-account-namespace
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: 'system:serviceaccount:my-service-account-namespace:my-service-account'
```

服务帐户可用作用户主题。

1.6.3. 以订阅管理员身份创建允许和拒绝列表

作为订阅管理员，您可以从 Git 存储库应用程序订阅创建应用程序，该订阅中包含一个**允许**列表只允许部署指定 Kubernetes **kind** 资源。您还可以在应用程序订阅中创建 **deny** 列表，以拒绝部署特定 Kubernetes **kind** 资源。

默认情况下，**policy.open-cluster-management.io/v1** 资源不会被应用程序订阅部署。为避免这种默认行为，应用程序订阅需要由订阅管理员部署。

请参阅以下 **allow** 和 **deny** 规格示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  annotations:
    apps.open-cluster-management.io/github-path: sub2
  name: demo-subscription
  namespace: demo-ns
spec:
  channel: demo-ns/somechannel
  allow:
  - apiVersion: policy.open-cluster-management.io/v1
    kinds:
    - Policy
  - apiVersion: v1
    kinds:
    - Deployment
  deny:
  - apiVersion: v1
```

```
    kinds:
    - Service
    - ConfigMap
  placement:
    local: true
```

以下应用程序订阅 YAML 指定当应用程序从源存储库的 **myapplication** 目录部署时，它将仅部署 **v1/Deployment** 资源，即使源存储库中有其他资源：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  annotations:
    apps.open-cluster-management.io/github-path: myapplication
  name: demo-subscription
  namespace: demo-ns
spec:
  channel: demo-ns/somechannel
  deny:
  - apiVersion: v1
      kinds:
        - Service
        - ConfigMap
  placement:
    placementRef:
      name: demo-placement
      kind: Placement
```

这个示例应用程序订阅 YAML 指定 **v1/Service** 和 **v1/ConfigMap** 资源以外的所有有效资源部署。您可以添加 **"*"** 来允许或拒绝 API 组中的所有资源类型，而不是列出 API 组中单独资源类型。

1.6.4. 添加协调选项

您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。

1.6.4.1. 协调频率 Git 频道

您可以选择在频道配置中选择协调频率选项： **high**、**medium**、**low** 和 **off**，以避免不必要的资源协调，并因此防止订阅 operator 的超载。

需要的访问权限： 管理员和集群管理员

请参阅以下的 **settings:attribute:<value>** 定义：

- **Off**：不自动协调部署的资源。**Subscription** 自定义资源的更改会启动协调。您可以添加或更新标签或注解。
- **Low**：部署的资源会每小时自动协调，即使源 Git 存储库没有改变。

- **Medium**：这是默认设置。订阅 operator 每 3 分钟将当前部署的提交 ID 与源存储库的最新提交 ID 进行比较，并将更改应用到目标集群。每 15 分钟，所有资源都会从源 Git 存储库重新应用到目标集群，即使存储库没有改变。

- **High**：部署的资源每两分钟自动协调，即使源 Git 存储库没有改变。

您可以使用订阅引用的频道自定义资源中的 **apps.open-cluster-management.io/reconcile-rate** 注解进行设置。

请参阅以下 **name: git-channel** 示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: git-channel
  namespace: sample
  annotations:
    apps.open-cluster-management.io/reconcile-rate: <value from the list>
spec:
  type: GitHub
  pathname: <Git URL>
---
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: git-subscription
  annotations:
    apps.open-cluster-management.io/git-path: <application1>
    apps.open-cluster-management.io/git-branch: <branch1>
spec:
  channel: sample/git-channel
  placement:
    local: true
```

在上例中，所有使用 **sample/git-channel** 的订阅都被分配了 **low** 协调频率。

a. 当订阅协调率被设置为 **low** 时，订阅的应用程序资源最多可能需要一小时才能完成协调。在单个应用程序视图的卡上，单击 **Sync** 以手动协调。如果设为 **off**，则永远不会协调。

无论频道中的 **reconcile-rate** 设置是什么，订阅都可以通过在 **Subscription 自定义资源中指定 apps.open-cluster-management.io/reconcile-rate: off** 注解来关闭自动协调。

请参阅以下 **git-channel** 示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: git-channel
  namespace: sample
  annotations:
    apps.open-cluster-management.io/reconcile-rate: high
spec:
  type: GitHub
  pathname: <Git URL>
---
apiVersion: apps.open-cluster-management.io/v1
```

```
kind: Subscription
metadata:
  name: git-subscription
  annotations:
    apps.open-cluster-management.io/git-path: application1
    apps.open-cluster-management.io/git-branch: branch1
    apps.open-cluster-management.io/reconcile-rate: "off"
spec:
  channel: sample/git-channel
  placement:
    local: true
```

git-subscription 部署的资源永远不会自动协调，即使 **reconcile-rate** 在频道中被设置为 **high**。

1.6.4.2. 协调频率 Helm 频道

每 15 分钟，订阅 operator 将当前部署 Helm chart 的哈希值与源仓库的哈希值进行比较。更改被应用到目标集群。资源协调的频率会影响其他应用程序部署和更新的性能。

例如，如果存在数百个应用程序订阅，并且您希望更频繁地协调所有订阅，则协调的响应时间会较慢。

根据应用程序的 Kubernetes 资源，适当的协调频率可以提高性能。

- **Off**：不自动协调部署的资源。Subscription 自定义资源的更改会启动协调。您可以添加或更新标签或注解。
- **Low** ：订阅 operator 每小时将当前部署的哈希值与源存储库的哈希进行比较，并在发生更改时对目标集群应用更改。
- **Medium**：这是默认设置。订阅 operator 每 15 分钟将当前部署的哈希值与源存储库的哈希进行比较，并在发生更改时对目标集群应用更改。
- **High** ：订阅 operator 每 2 分钟将当前部署的哈希值与源存储库的哈希进行比较，并在有更改时对目标集群应用更改。

您可以使用订阅引用的 **Channel** 自定义资源中的 **apps.open-cluster-management.io/reconcile-rate** 注解进行设置。请参阅以下 **helm-channel** 示例：

请参阅以下 **helm-channel** 示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: helm-channel
  namespace: sample
  annotations:
    apps.open-cluster-management.io/reconcile-rate: low
spec:
  type: HelmRepo
  pathname: <Helm repo URL>
---
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: helm-subscription
spec:
```

```
  channel: sample/helm-channel
  name: nginx-ingress
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: nginx-ingress-simple
    packageOverrides:
    - path: spec
      value:
        defaultBackend:
          replicaCount: 3
  placement:
    local: true
```

在本例中，所有使用 **sample/helm-channel** 的订阅都会被分配一个 **low** 协调频率。

无论频道中的 reconcile-rate 设置是什么，订阅都可以通过在 **Subscription** 自定义资源中指定 **apps.open-cluster-management.io/reconcile-rate: off** 注解来关闭自动协调，如下例所示：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: helm-channel
  namespace: sample
  annotations:
    apps.open-cluster-management.io/reconcile-rate: high
spec:
  type: HelmRepo
  pathname: <Helm repo URL>
---
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: helm-subscription
  annotations:
    apps.open-cluster-management.io/reconcile-rate: "off"
spec:
  channel: sample/helm-channel
  name: nginx-ingress
  packageOverrides:
  - packageName: nginx-ingress
    packageAlias: nginx-ingress-simple
    packageOverrides:
    - path: spec
      value:
        defaultBackend:
          replicaCount: 3
  placement:
    local: true
```

在本例中，**helm-subscription** 部署的资源永远不会自动协调，即使 **reconcile-rate** 在频道中被设置为 **high**。

1.6.5. 配置领导选举机制

借助 **LeaderElection**，您可以更改控制器如何在出现故障时请求选择新领导，这样可保证一次仅处理协调。您可以增加或减少控制器获取 **LeaderElection** 所需的时间。随着时间减少，故障期间会更快地选择新的领导。

注： 更改控制器的默认值可能会影响该任务中的系统性能。您可以通过更改 **leaseDuration**、**renewDeadline** 或 **retryPeriod** 控制器的默认值来减少 **etcd** 负载。

需要的访问权限：集群管理员

1.6.5.1. 编辑控制器标记

要配置 **LeaderElection**，您可以更改以下默认值：

- **leader-election-lease-duration: 137 seconds**
- **renew-deadline: 107 seconds**
- **retry-period: 26 seconds**

请参阅以下步骤来更改 **multicluster-operators-application**、**multicluster-operators-channel**、**multicluster-operators-standalone-subscription** 或 **multicluster-operators-hub-subscription** 控制器：

1. 运行以下命令以暂停 **multiclusterhub** ：

```
oc annotate mch -n open-cluster-management multiclusterhub mch-pause=true --overwrite=true
```

2. 通过在 **oc edit** 命令中添加控制器名称来编辑 **deployment** 文件。请参见以下示例命令：

```
oc edit deployment -n open-cluster-management multicluster-operators-hub-subscription
```

3. 通过搜索 **- command** 来查找控制器命令标志。
4. 在控制器中的 containers 部分中，插入 **- command** 标记。例如，插入 **RetryPeriod**。
5. 保存该文件。控制器会自动重启以应用标志。
6. 对您要更改的每个控制器重复此步骤。
7. 运行以下命令以恢复 **multiclusterhub** ：

```
oc annotate mch -n open-cluster-management multiclusterhub mch-pause=false --overwrite=true
```

请参阅以下成功编辑到 **-command** 的输出示例，其中 **retryPeriod** 标志会将前面提到的默认时间加倍到 **52**，这被分配来重试合并 **leaderElection** ：

```
command:
- /usr/local/bin/multicluster-operators-subscription
- --sync-interval=60
- --retry-period=52
```

1.6.6. 为安全 Git 连接配置应用程序频道和订阅

Git 频道和订阅通过 HTTPS 或 SSH 连接到指定的 Git 存储库。以下应用程序频道配置可用于安全 Git 连接：

- 使用用户和访问令牌连接到私有仓库
- 将不安全的 HTTPS 连接至 Git 服务器
- 在安全 HTTPS 连接中使用自定义 CA 证书
- 生成到 Git 服务器的 SSH 连接
- 更新证书和 SSH 密钥

1.6.6.1. 使用用户和访问令牌连接到私有仓库

您可以使用频道和订阅连接到 Git 服务器。有关连接到具有用户和访问令牌的私有存储库，请参阅以下步骤：

1. 在与频道相同的命名空间中创建 secret。将 **user** 字段设置为 Git 用户 ID，将 **accessToken** 字段设置为 Git 个人访问令牌。值应该采用 base64 编码。请参阅以下填充的用户和 accessToken 示例：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-git-secret
  namespace: channel-ns
data:
  user: dXNlcgo=
  accessToken: cGFzc3dvcmQK
```

2. 使用 secret 配置频道。请参阅以下带有 **secretRef** 填充的示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: sample-channel
  namespace: channel-ns
spec:
    type: Git
    pathname: <Git HTTPS URL>
    secretRef:
      name: my-git-secret
```

1.6.6.2. 将不安全的 HTTPS 连接至 Git 服务器

您可以使用开发环境中的以下连接方法使用由自定义或自签名证书颁发机构签名的 SSL 证书，连接到私有托管 Git 服务器。但是，不建议在生产环境中使用这个步骤：

在频道规格中指定 **insecureSkipVerify: true**。否则，与 Git 服务器的连接会失败，并显示类似如下的错误：

```
x509: certificate is valid for localhost.com, not localhost
```

关于这个方法，请查看以下频道规格添加的示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
labels:
  name: sample-channel
  namespace: sample
spec:
  type: GitHub
  pathname: <Git HTTPS URL>
  insecureSkipVerify: true
```

1.6.6.3. 在安全 HTTPS 连接中使用自定义 CA 证书

您可以使用这个连接方法使用由自定义或自签名证书认证机构签名的 SSL 证书安全地连接到托管的 Git 服务器。

1. 创建包括 PEM 格式的 Git 服务器 root 和中间 CA 证书的 ConfigMap。ConfigMap 必须与频道 CR 位于同一命名空间中。字段名称必须是 **caCerts** 并使用 |。在以下示例中，可以注意到 **caCerts** 可包含多个证书，如 root 和中间 CA：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: git-ca
  namespace: channel-ns
data:
  caCerts: |
    # Git server root CA

    -----BEGIN CERTIFICATE-----
    MIIF5DCCA8wCCQDInYMol7LSDTANBgkqhkiG9w0BAQsFADCBszELMAkGA1UEBhMC

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
    LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
    hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH


VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBgNVBA
MM
    PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWIxMy5kZXYwNi5yZWQtY2hl
    c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcmVkaGF0LmNvbTCC
    AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0Ic3
    U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
    6OS3gj738F635inz1KdyhKtlWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNIdRHcpQP
    upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWIwNsoL5xt0Lw4mSyhlEip/t8lU
    xn2y8qhm7MiIUpXuwWhSYgCrEVqmTcB70Pc2YRZdSFoIMN9Et70MjQN0TXjoktH8
    PyASJIKIRd+48yROIbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
    +SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcnQVFmvaoM
    gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
    diNXs4Zn3QSStC2x2hKnK+Q1rwCSEg/lBawgxGUslTboFH77a+Kwu4Oug9ibtm5z
```

```
ISs/JY4Kiy4C2XJOltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSAxaaSXIX+VbzkHF2N
aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/I8xHvt9L
kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
jaaSeRDRoToTb/yN0Xsi69UfK0aBdvhCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
PQVGs/vehQRqLV9uZBigG6oZqdUqEimaLHrOcEAHB5RVcnFurz0qNbT+UySjsD63
9yJdCeQbeKAR9SC4hG13EbM/RZh0lgFupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
kh9aBB7FLykK1TeXVuANlNU4EMyJ/e+uhNkS9ubNJ3vuRuo+ECHsha058yi16JC9
NkZqP+df4Hp85sd+xhrgYieq7QGX2KOXAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
2UGZxMsRjlP6UqB08LsV5ZBAefElR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
XSUkATuuQw4ctyZLDkUpzrDzgd2Bt+aawF6sD2YqycaGFwv2YD9t1YlD6F4Wh8Mc
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFl2fAIqFtyum6/8ZN9nZ9K
FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6IadliUOST3w5t92cB
M26t73UCExXMXTCQvnp0ki84PeR1kRk4
-----END CERTIFICATE-----

# Git server intermediate CA 1

-----BEGIN CERTIFICATE-----
MIIF5DCCA8wCCQDInYMol7LSDTANBgkqhkiG9w0BAQsFADCBszELMAkGA1UEBhMC

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH


VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBgNVBA
MM
PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWIxMy5kZXYwNi5yZWQtY2hl
c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcmVkaGF0LmNvbTCC
AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0Ic3
U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
6OS3gj738F635inz1KdyhKtlWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNIdRHcpQP
upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWIwNsoL5xt0Lw4mSyhlEip/t8lU
xn2y8qhm7MiIUpXuwWhSYgCrEVqmTcB70Pc2YRZdSFolMN9Et70MjQN0TXjoktH8
PyASJIKIRd+48yROIbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
+SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcnQVFmvaoM
gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
diNXs4Zn3QSStC2x2hKnK+Q1rwCSEg/lBawgxGUslTboFH77a+Kwu4Oug9ibtm5z
ISs/JY4Kiy4C2XJOltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSAxaaSXIX+VbzkHF2N
aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/I8xHvt9L
kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
jaaSeRDRoToTb/yN0Xsi69UfK0aBdvhCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
PQVGs/vehQRqLV9uZBigG6oZqdUqEimaLHrOcEAHB5RVcnFurz0qNbT+UySjsD63
9yJdCeQbeKAR9SC4hG13EbM/RZh0lgFupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
kh9aBB7FLykK1TeXVuANlNU4EMyJ/e+uhNkS9ubNJ3vuRuo+ECHsha058yi16JC9
NkZqP+df4Hp85sd+xhrgYieq7QGX2KOXAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
2UGZxMsRjlP6UqB08LsV5ZBAefElR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
XSUkATuuQw4ctyZLDkUpzrDzgd2Bt+aawF6sD2YqycaGFwv2YD9t1YlD6F4Wh8Mc
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFl2fAIqFtyum6/8ZN9nZ9K
```

```
    FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6IadliUOST3w5t92cB
    M26t73UCExXMXTCQvnp0ki84PeR1kRk4
    -----END CERTIFICATE-----

    # Git server intermediate CA 2

    -----BEGIN CERTIFICATE-----
    MIIF5DCCA8wCCQDInYMol7LSDTANBgkqhkiG9w0BAQsFADCBszELMAkGA1UEBhMC

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
    LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
    hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH


VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBgNVBA
MM
    PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWIxMy5kZXYwNi5yZWQtY2hl
    c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcmVkaGF0LmNvbTCC
    AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0Ic3
    U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
    6OS3gj738F635inz1KdyhKtlWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNIdRHcpQP
    upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWIwNsoL5xt0Lw4mSyhlEip/t8lU
    xn2y8qhm7MiIUpXuwWhSYgCrEVqmTcB70Pc2YRZdSFolMN9Et70MjQN0TXjoktH8
    PyASJIKIRd+48yROIbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
    +SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcnQVFmvaoM
    gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
    diNXs4Zn3QSStC2x2hKnK+Q1rwCSEg/lBawgxGUslTboFH77a+Kwu4Oug9ibtm5z
    ISs/JY4Kiy4C2XJOltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSAxaaSXIX+VbzkHF2N
    aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/I8xHvt9L
    kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
    jaaSeRDRoToTb/yN0Xsi69UfK0aBdvhCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
    g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
    PQVGs/vehQRqLV9uZBigG6oZqdUqEimaLHrOcEAHB5RVcnFurz0qNbT+UySjsD63
    9yJdCeQbeKAR9SC4hG13EbM/RZh0lgFupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
    kh9aBB7FLykK1TeXVuANlNU4EMyJ/e+uhNkS9ubNJ3vuRuo+ECHsha058yi16JC9
    NkZqP+df4Hp85sd+xhrgYieq7QGX2KOXAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
    2UGZxMsRjlP6UqB08LsV5ZBAefElR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
    XSUkATuuQw4ctyZLDkUpzrDzgd2Bt+aawF6sD2YqycaGFwv2YD9t1YlD6F4Wh8Mc
    20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFl2fAlqFtyum6/8ZN9nZ9K
    FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6IadliUOST3w5t92cB
    M26t73UCExXMXTCQvnp0ki84PeR1kRk4
    -----END CERTIFICATE-----
```

2. 使用此 ConfigMap 配置频道。参阅上一步中的 **git-ca** 名称的以下示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: my-channel
  namespace: channel-ns
spec:
```

```
  configMapRef:
    name: git-ca
  pathname: <Git HTTPS URL>
  type: Git
```

1.6.6.4. 生成到 Git 服务器的 SSH 连接

1. 在 **data** 的 **sshKey** 字段中创建一个 secret 来包含您的私有 SSH 密钥。如果密钥需要使用密码短语进行保护，在 **passphrase** 字段中指定密码。此 secret 必须与频道 CR 位于同一命名空间中。使用 **oc** 命令创建此 secret，以创建 secret generic **git-ssh-key --from-file=sshKey=./.ssh/id_rsa**，然后添加以 base64 编码的 **passphrase**。请参见以下示例：

```
apiVersion: v1
kind: Secret
metadata:
  name: git-ssh-key
  namespace: channel-ns
data:
  sshKey:
LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdG
RqRUFBQUFBQ21GbGN6STFOaTFqZEhJQUFBQUdZbU55ZVhCMEFBQUFHQUFBQUJD
K3YySHhWSIwCm8zejh1endzV3NWODMvSFVkOEtGeVBmWk5OeE5TQUgcFA3Yk1yR2tlRF
FPd3J6MGIKOUlRM0tKVXQzWEE0Zmd6NVlrVFVhcTJsZWxxVk1HcXI2WHF2UVJ5Mkc0NkRI
RVlYUGpabVZMcGVuaGtRYU5HYmpaMmZOdQpWUGpiOVhZRmd4bTNnYUpJU3BNeTFL
WjQ5MzJvOFByaDZEdzRYVUF1a28wZGdBaDdndVpPaE53b0pVYnNmYlZRc0xMS1RrCnQw
blZ1anRvd2NEVGx4TlpIUjcwbGVUSHdGQTYwekM0elpMNkRPc3RMYjV2LzZhMjFHRlMwVm
VXQ3YvMlpMOE1sbjVUZWwKSytoUWtxRnJBL3BUc1ozVXNjSG1GUi9PV25FPQotLS0tLUVO
RCBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0K
  passphrase: cGFzc3cwcmQK
type: Opaque
```

2. 使用 secret 配置频道。请参见以下示例：

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: my-channel
  namespace: channel-ns
spec:
  secretRef:
    name: git-ssh-key
  pathname: <Git SSH URL>
  type: Git
```

订阅控制器使用提供的 Git 主机名执行 **ssh-keyscan** 来构建 **known_hosts** 列表，以防止 SSH 连接中的 Man-in-the-middle（MITM）攻击。如果要跳过此步骤并使用不安全的连接，请在频道配置中使用 **insecureSkipVerify: true**。这不是最佳方案，特别是在生产环境中。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: my-channel
  namespace: channel-ns
spec:
  secretRef:
```

```
  name: git-ssh-key
pathname: <Git SSH URL>
type: Git
insecureSkipVerify: true
```

1.6.6.5. 更新证书和 SSH 密钥

如果 Git 频道连接配置需要更新，如 CA 证书、凭证或 SSH 密钥，则需要在同一命名空间中创建新 secret 和 ConfigMap，并更新频道以引用新的 secret 和 ConfigMap。如需更多信息，请参阅使用自定义 CA 证书进行安全 HTTPS 连接。

1.6.7. 配置 Helm 以监视命名空间资源

默认情况下，当订阅将订阅的 Helm 资源部署到目标集群时，应用程序资源会被监视。您可以配置 Helm 频道类型来监视命名空间范围的资源。启用后，会恢复对这些监视命名空间范围的资源的手动更改。

1.6.7.1. 配置

需要的访问权限：集群管理员

要将 Helm 应用程序配置为监视命名空间范围的资源，请将订阅定义中的 **watchHelmNamespaceScopedResources** 字段的值设置为 **true**。请参见以下示例。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: nginx
  namespace: ns-sub-1
spec:
  watchHelmNamespaceScopedResources: true
  channel: ns-ch/predev-ch
  name: nginx-ingress
  packageFilter:
    version: "1.36.x"
```

1.6.8. 调度部署

如果需要只在特定时间部署新的 Helm Chart 或其他资源，或更改 Helm Chart 或其他资源，您可以为这些资源定义订阅，以便只在特定时间才进行部署。另外，您可以限制部署。

例如，您可以把每周五的晚上 10 点到 11 点期间定义为调度的维护窗口期，只在此期间对集群应用补丁或进行其他应用程序的更新。

您可以限制或阻止在一个指定的时间窗期间开始部署，比如避免在商业高峰期内进行部署。例如，为了避免高峰期，您可以定义一个时间窗以避免在早上 8 点到下午 8 点间进行部署。

通过为订阅定义时间窗，您可以协调所有应用程序和集群的更新。例如，您可以定义订阅以只在 6:01 PM 和 11:59 PM 之间部署新的应用程序资源，并定义其他订阅仅在 12:00 AM 到 7:59 AM 之间部署这些资源的更新版本。

当为订阅定义一个时间窗时，则定义了订阅处于活跃变化的时间范围。作为定义时间窗的一部分，您可以指定在该窗口中订阅是 *active（活跃）* 或 *blocked（阻止）* 状态。

只有在订阅处于活跃状态时，才会开始部署新的或更改的资源。无论订阅是活跃还是被阻止，订阅都会继续监控任何新的或被更改的资源。活跃和阻止的设置仅会影响到部署。

当检测到新的或更改的资源时，时间窗定义决定订阅的下一步操作。

- 对于 **HelmRepo**、**ObjectBucket** 和 **Git** 类型的频道的订阅：
- 如果在订阅是 *active* 的时间范围内检测到资源，则开始部署资源。
- 如果在订阅无法运行部署时检测到资源，部署资源的请求会被缓存。当订阅在下一个活跃时间时，缓存的请求会被应用并开始相关的部署。
- 当时间窗为 *blocked* 时，之前由应用程序订阅部署的所有资源仍会保留。在窗口再次激活前，任何新的更新都会被阻止。

最终用户可能会错误地认为应用程序子时间窗被阻止，所有部署的资源都会被删除。当应用程序子时间窗再次处于活跃状态时，他们将返回。

如果部署在指定的时间窗内开始，并在定义的时间窗结束时仍在运行，部署将继续运行以完成。

要为订阅定义时间窗，您需要在订阅资源定义 YAML 中添加所需的字段和值。

- 作为定义时间窗的一部分，您可以使用天和小时定义时间窗。
- 您还可以定义时间窗类型，这决定了部署可以开始的时间窗是在指定的时间段内还是在指定的时间段外。
- 如果时间窗类型是 **active**，则部署只能在指定的时间范围内开始。当希望部署只在特定的维护窗口中进行时，您可以使用此设置。
- 如果时间窗类型是 **block**，则部署不能在指定的时间范围内启动，但可以在任何其他时间开始。当您有需要的关键更新，同时需要避免在指定时间范围内进行部署，则可以使用这个设置。例如，您可以使用这个设置类型来定义一个时间窗，允许在 10 AM 到 2:00 PM 这个时间段外随时应用与安全相关的更新。
- 您可以为订阅定义多个时间窗，如在每周一和周三定义时间窗。

1.6.9. 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 (1)
```

(1) **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
```

1.6.10. 频道示例概述

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 OpenShift 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：

```
oc 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：

```
oc get application.app
```

- 频道 YAML 结构
- 频道 YAML 表
- Object storage bucket (ObjectBucket) 频道
- Helm 仓库（**HelmRepo**）频道
- Git（**Git**）仓库频道

1.6.10.1. 频道 YAML 结构

有关您可以部署的应用程序示例，请参阅 **stolostron** 存储库。

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和产品控制台编写您自己的 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:
```

1.6.10.2. 频道 YAML 表

字段	可选或必需的	描述
apiVersion	必填	将值设为 **apps.open-cluster-management.io/v1**。
kind	必填	将值设为 **Channel** 以表示资源是频道。
metadata.name	必填	频道的名称。
metadata.namespace	必填	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	选填	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	必填	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**、**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。

字段	可选或必需的	描述
spec.secretRef.name	选填	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。
spec.gates	选填	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** 值仅适用于 **ObjectBucket** 频道类型，不适用于 **HelmRepo** 和 **Git** 频道类型。
spec.gates.annotations	选填	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	选填	频道的标签。
spec.insecureSkipVerify	选填	默认值为 **false**，如果设置 **true**，则通过跳过身份验证来构建频道连接

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/
```

1.6.10.3. 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:xxxx/dev] # URL is appended with the valid bucket name, which
matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

1.6.10.4. Helm 仓库 (HelmRepo) 频道

以下示例频道定义将 Helm 仓库抽象为频道：

弃用备注： 对于 2.10，在频道 **ConfigMap** 引用中指定 **insecureSkipVerify: "true"** 来跳过 Helm repo SSL 证书已被弃用。请参阅以下示例，它使用 **spec.insecureSkipVerify: true** 进行替换：

```
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.
    insecureSkipVerify: true
    type: HelmRepo
```

以下频道定义显示了 Helm 仓库频道的另一个示例：

注： 对于 Helm， Helm chart 中包含的所有 Kubernetes 资源必须具有标签发行版本 **{{ .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/
```

1.6.10.5. 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。如果您有一个公共存储库，则不需要 **secretRef** 标签和值：

```
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
```

1.6.11. 订阅示例概述

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 OpenShift 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 api 服务器。将 **filename** 替换为您的文件名称：

```
oc 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：

```
oc get application.app
```

- 订阅 YAML 结构
- 订阅 YAML 表
- 订阅文件示例
  - 订阅时间窗示例
  - 带有覆盖的订阅示例
  - Helm 仓库订阅示例
  - Git 仓库订阅示例

1.6.11.1. 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: Placement
  overrides:
    clusterName:
    clusterOverrides:
      path:
      value:
```

1.6.11.2. 订阅 YAML 表

字段	必需/可选	描述
apiVersion	必填	将值设为 **apps.open-cluster-management.io/v1**。
kind	必填	将值设为 **Subscription** 以表示资源是订阅。
metadata.name	必填	用于标识订阅的名称。
metadata.namespace	必填	用于订阅的命名空间资源。
metadata.labels	选填	订阅的标签。

字段	必需/可选	描述
spec.channel	选填	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	选填	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	选填	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	选填	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	选填	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	选填	可部署资源的注解。

字段	必需/可选	描述
spec.packageOverrides	选填	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	选填	为将被覆盖的 Kubernetes 资源指定别名。
spec.packageOverrides.packageOverrides	选填	用于覆盖 Kubernetes 资源的参数和替换值配置。
spec.placement	必填	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，但对于独立集群或您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	选填	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。

字段	必需/可选	描述
spec.placement.clusterSelector	选填	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	选填	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**。
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 **Placement** 以表示将放置规则用于通过订阅进行部署。
spec.overrides	选填	需要覆盖的任何参数和值，如特定于集群的设置。
spec.overrides.clusterName	选填	参数和值将被覆盖的一个或多个集群的名称。
spec.overrides.clusterOverrides	选填	要覆盖的参数和值的配置。
spec.timeWindow	选填	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。

字段	必需/可选	描述
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。
- 在订阅规格中，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: Placement
      name: towhichcluster
  overrides:
  - clusterName: "/"
    clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.6.11.3. 订阅文件示例

有关您可以部署的应用程序示例，请参阅 **stolostron** 存储库。

```
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
```

1.6.11.4. 二级频道示例

如果存在镜像的频道（应用程序源存储库），您可以在订阅 YAML 中指定 **secondaryChannel**。当应用程序订阅无法使用主频道连接到仓库服务器时，它会使用二级频道连接到仓库服务器。确保存储在二级频道中的应用程序清单与主频道同步。请参阅以下使用 **secondaryChannel** 的订阅 YAML 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
  secondaryChannel: ns-ch-2/predev-ch-2
  name: nginx-ingress
```

1.6.11.4.1. 订阅时间窗示例

以下示例订阅包含多个配置的时间窗。每个周一、周三和周五的 10:20 AM 到 10:30 AM 之间有一个时间窗。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: Placement
      name: towhichcluster
  timewindow:
    windowtype: "active"
    location: "America/Los_Angeles"
    daysofweek: ["Monday", "Wednesday", "Friday"]
    hours:
      - start: "10:20AM"
        end: "10:30AM"
      - start: "12:40PM"
        end: "1:40PM"
```

对于 **timewindow**，请输入 **active** 或 **blocked**，具体取决于类型的目的。

1.6.11.4.2. 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
```

1.6.11.4.3. 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
```

1.6.11.4.4. Git 仓库订阅示例

1.6.11.4.4.1. 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: Placement
      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。

备注：当您使用订阅 Helm chart 的 Git 频道订阅时，资源拓扑视图可能会显示额外的 **Helmrelease** 资源。此资源是内部应用管理资源，可以安全地忽略。

1.6.11.4.4.2. 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。

1.6.11.4.4.3. 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。

1.6.11.4.4.4. 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。

1.6.11.4.4.4.1. 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**。

1.6.11.4.4.4.2. 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 仓库使用唯一的 secret。

1.6.11.4.4.4.3. 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。

1.6.11.4.4.4.4. 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>"
```

订阅中不需要特定于 webhook 的配置。

1.6.12. 放置规则示例概述（已弃用）

弃用：PlacementRule 已被弃用。改为使用 **Placement**。

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 OpenShift 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：

```
oc 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：

```
oc get application.app
```

- 放置规则 YAML 结构
- 放置规则 YAML 值表
- 放置规则示例文件

1.6.12.1. 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和产品控制台编写您自己的 YAML 内容

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

1.6.12.2. 放置规则 YAML 值表

字段	必需/可选	描述
apiVersion	必填	将值设为 **apps.open-cluster-management.io/v1**。
kind	必填	将值设为 **PlacementRule** 以表示资源是放置规则。
metadata.name	必填	用于标识放置规则的名称。
metadata.namespace	必填	用于放置规则的命名空间资源。
metadata.resourceVersion	选填	放置规则资源的版本。
metadata.labels	选填	放置规则的标签。
spec.clusterSelector	选填	用于标识目标集群的标签
spec.clusterSelector.matchLabels	选填	目标集群必须存在的标签。
spec.clusterSelector.matchExpressions	选填	目标集群必须存在的标签。
status.decisions	选填	定义放置可部署资源的目标集群。
status.decisions.clusterName	选填	目标集群的名称
status.decisions.clusterNamespace	选填	目标集群的命名空间。
spec.clusterReplicas	选填	要创建的副本数。
spec.clusterConditions	选填	为集群定义任何条件。
spec.ResourceHint	选填	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	选填	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	选填	将值设为 **asc** 表示升序，或者设为 **desc** 表示降序。
spec.Policies	选填	放置规则的策略过滤器。

1.6.12.3. 放置规则示例文件

有关您可以部署的应用程序示例，请参阅 **stolostron** 存储库。

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
```

1.6.13. 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 OpenShift 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：

```
oc 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：

```
oc get application.app
```

- 应用程序 YAML 结构
- 应用程序 YAML 表
- 应用程序文件示例

1.6.13.1. 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
```

1.6.13.2. 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		
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。

1.6.13.3. 应用程序文件示例

有关您可以部署的应用程序示例，请参阅 **stolostron** 存储库。

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
```