



Red Hat Advanced Cluster Management for Kubernetes 2.10

アプリケーション

アプリケーション管理

Red Hat Advanced Cluster Management for Kubernetes 2.10 アプリケーション

アプリケーション管理

法律上の通知

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.

概要

Git リポジトリ、Helm リポジトリ、およびオブジェクトストレージリポジトリを使用してアプリケーションを作成する方法は、こちらを参照してください。

目次

第1章 アプリケーションの管理	3
1.1. アプリケーションモデルおよび定義	3
1.2. アプリケーションコンソール	8
1.3. サブスクリプションレポート	10
1.4. アプリケーションリソースの管理	14
1.5. ANSIBLE AUTOMATION PLATFORM の統合と導入	21
1.6. アプリケーションの詳細設定	26

第1章 アプリケーションの管理

アプリケーションの作成、デプロイ、および管理に関する詳細は、以下のトピックを参照してください。本書では、Kubernetes の概念および用語に精通していることを前提としています。主要な Kubernetes の用語はコンポーネントについては、定義しません。Kubernetes の概念に関する情報は、[Kubernetes ドキュメント](#) を参照してください。

アプリケーション管理機能では、アプリケーションや、アプリケーションの更新を構築してデプロイするオプションが統一、簡素化されています。開発者および DevOps 担当者は、このアプリケーション管理機能を使用することで、チャンネルおよびサブスクリプションベースの自動化を使用し、環境全体でアプリケーションを作成して管理できます。

重要: アプリケーション名は 37 文字を超えることができません。

以下のトピックを参照してください。

- [アプリケーションモデルおよび定義](#)
 - [アプリケーションコンソール](#)
 - [サブスクリプションレポート](#)
 - [アプリケーションリソースの管理](#)
 - [Git リポジトリを使用したアプリケーションの管理](#)
 - [Helm リポジトリを使用したアプリケーションの管理](#)
 - [Object Storage リポジトリを使用したアプリケーションの管理](#)
- [アプリケーションの詳細設定](#)
 - [Git リソースのサブスクリプション](#)
 - [サブスクリプション特権の付与](#)
 - [サブスクリプション管理者としての許可リストの作成および拒否リストの作成](#)
 - [調整オプションの追加](#)
 - [セキュアな Git 接続用のアプリケーションチャンネルおよびサブスクリプションの設定](#)
 - [Ansible Automation Platform タスクのセットアップ](#)
 - [デプロイメントのスケジュール](#)
 - [パッケージの上書きの設定](#)
 - [チャンネルの例](#)
 - [サブスクリプションの例](#)
 - [アプリケーションの例](#)

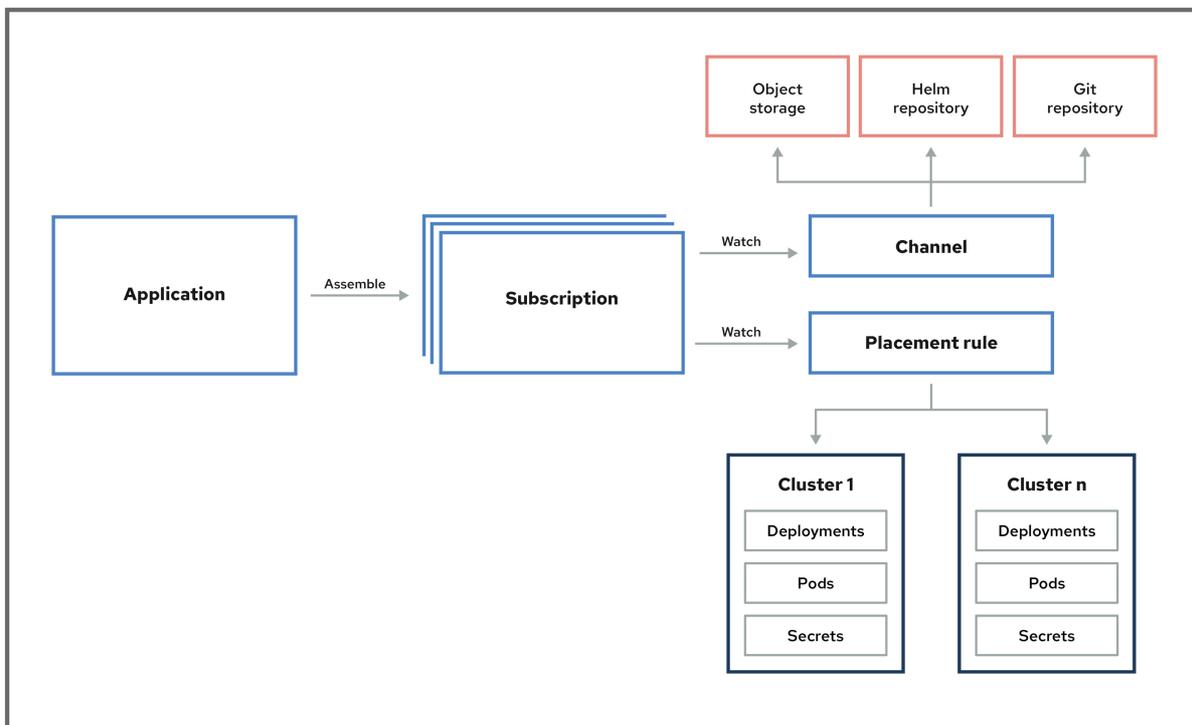
1.1. アプリケーションモデルおよび定義

アプリケーションモデルは、マネージドクラスターにデプロイされるリソースが含まれる1つまたは複

数の Kubernetes リソースリポジトリ (チャンネル リソース) にサブスクライブすることをベースとしています。単一クラスターアプリケーションもマルチクラスターアプリケーションも同じ Kubernetes 仕様を使用しますが、マルチクラスターアプリケーションでは、デプロイメントおよびアプリケーション管理ライフサイクルがさらに自動化されます。

アプリケーションモデルの詳細を理解するには、以下のイメージを参照してください。

APPLICATION SUBSCRIPTION MODEL



以下のアプリケーションリソースセクションを確認します。

- [アプリケーション](#)
- [サブスクリプション](#)
- [ApplicationSet](#)
- [アプリケーションドキュメント](#)

ベストプラクティス: チャンネルとサブスクリプションモデルの代わりに、GitOps Operator または Argo CD 統合を使用します。詳細は、[GitOps overview](#) を参照してください。

1.1.1. アプリケーション

Red Hat Advanced Cluster Management for Kubernetes のアプリケーション ([application.app.k8s.io](#)) は、アプリケーションを設定する Kubernetes リソースのグループ化に使用します。

Red Hat Advanced Cluster Management for Kubernetes アプリケーションのアプリケーションコンポーネントリソースはすべて、YAML ファイルの仕様セクションで定義します。アプリケーションコンポーネントリソースを作成または更新する必要がある場合は、適切な仕様セクションを作成してリソースを定義するラベルを追加する必要があります。

OpenShift Container Platform GitOps またはクラスターにインストールされている Argo CD Operator によって検出されるアプリケーションである **Discovered** アプリケーションと連携することもできます。同じリポジトリを共有するアプリケーションは、このビューでグループ化されます。

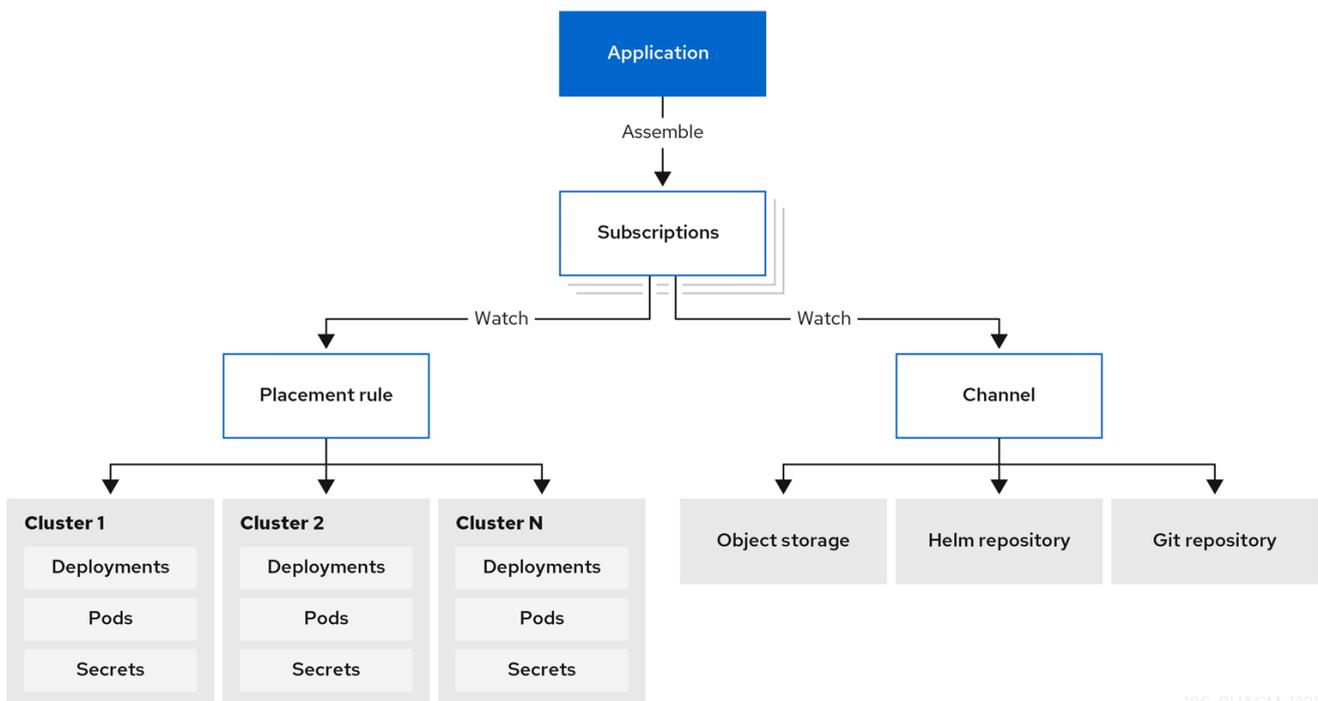
1.1.2. サブスクリプション

サブスクリプション (subscription.apps.open-cluster-management.io) により、クラスターは Git リポジトリ、Helm リリースリポジトリ、またはオブジェクトストレージリポジトリなどのソースリポジトリ (チャンネル) にサブスクライブできます。

ハブクラスターが自己管理の場合には、サブスクリプションはハブクラスターにローカルでアプリケーションリソースをデプロイできます。トポロジーで **local-cluster** (自己管理のハブクラスター) サブスクリプションを表示できます。リソース要件は、ハブクラスターのパフォーマンスに悪影響を与える可能性があります。

サブスクリプションは、チャンネルまたはストレージの場所を参照して、新規または更新したリソーステンプレートを特定できます。次に、サブスクリプション operator は、先にハブクラスターを確認しなくても、直接ストレージの場所からターゲットのマネージドクラスターにダウンロードしてデプロイできます。サブスクリプションを使用すると、サブスクリプション operator は、ハブクラスターの代わりに、新規または更新されたリソースがないか、チャンネルを監視できます。

以下のサブスクリプションアーキテクチャーイメージを参照してください。



186_RHACM_I221

1.1.2.1. チャンネル

チャンネル (channel.apps.open-cluster-management.io) は、クラスターがサブスクリプションを使用してサブスクライブ可能なソースリポジトリを定義します。許容タイプは、Git、Helm リリース、オブジェクトストレージリポジトリ、ハブクラスター上にあるリソーステンプレートです。

認可が必要なチャンネルから Kubernetes リソースまたは Helm チャートを必要とするアプリケーション (例: エンタイトルメントのある Git リポジトリ) がある場合は、これらのチャンネルにアクセスできるようにするシークレットを使用できます。お使いのサブスクリプションで、データセキュリティーを確保しつつも、これらのチャンネルからデプロイメント用の Kubernetes リソースおよび Helm チャートにアクセスできます。

チャンネルは、ハブクラスター内の namespace を使用して、リソースをデプロイメント用に保存する、物理的な場所を参照します。クラスターは、チャンネルにサブスクライブすることで、クラスターごとにデプロイするリソースを特定できます。

注記: 一意の namespace に各チャンネルを作成することを推奨します。ただし、Git チャンネルは、Git、Helm、オブジェクトストレージなどの別のチャンネルタイプで namespace を共有できます。

チャンネル内の deployable には、そのチャンネルにサブスクライブするクラスターのみがアクセスできません。

1.1.2.1.1. サポート対象の Git リポジトリサーバー

- GitHub
- GitLab
- Bitbucket
- Gogs

1.1.3. ApplicationSet

ApplicationSet は Argo CD のサブプロジェクトで、GitOps Operator によりサポートされています。**ApplicationSet** は Argo CD アプリケーションのマルチクラスターサポートを追加します。Red Hat Advanced Cluster Management コンソールからアプリケーションを作成できます。

注記: **ApplicationSet** をデプロイするための前提条件の詳細については、[マネージドクラスターの GitOps への登録](#) を参照してください。

OpenShift Container Platform GitOps は Argo CD を使用してクラスターリソースを維持します。Argo CD は、アプリケーションの継続的インテグレーションおよび継続的デプロイメント (CI/CD) のオープンソースの宣言型ツールです。OpenShift Container Platform GitOps は Argo CD をコントローラー (OpenShift Container Platform GitOps Operator) として実装し、Git リポジトリで定義されるアプリケーション定義および設定を継続的に監視できるようにします。次に、Argo CD は、これらの設定の指定された状態をクラスターのライブ状態と比較します。

ApplicationSet コントローラーは、GitOps Operator インスタンスを使用してクラスターにインストールされ、cluster-administrator に焦点を当てたシナリオをサポートする機能を追加することでクラスターを補完します。**ApplicationSet** コントローラーは以下の機能を提供します。

- 単一の Kubernetes マニフェストを使用して、GitOps Operator で複数の Kubernetes クラスターをターゲットにする機能。
- 単一の Kubernetes マニフェストを使用して、GitOps Operator を含む1つまたは複数の Git リポジトリから複数のアプリケーションをデプロイする機能。
- monorepo のサポートの改善。これは単一の Git リポジトリ内で定義されている複数の ArgoCD アプリケーションリソースである ArgoCD のコンテキストに含まれます。
- マルチクラスター内での個別クラスターテナントの機能の向上。特権のあるクラスター管理者を使用して宛先のクラスター/namespace を有効にする必要なく、Argo CD でアプリケーションをデプロイできます。

ApplicationSet Operator はクラスターデジモンジェネレーターを使用して、カスタムリソース固有のロジックでデプロイするマネージドクラスターを決定する Kubernetes カスタムリソースをインターフェイスします。クラスターデジモンリソースは、マネージドクラスターのリストを生成し、これを

ApplicationSet リソースの `template` フィールドにレンダリングします。これは、参照される Kubernetes リソースの完全な形状に関する知識を必要としないダックタイピングを使用して行われます。

ApplicationSet 内の `generators.clusterDecisionResource` 値の例を参照してください。

```
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
```

以下の **Placement** を参照してください。

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

```
spec:
  clusterSets:
    - sampleclusterset
```

ApplicationSets の詳細は、[Cluster Decision Resource Generator](#) を参照してください。

1.1.4. アプリケーションドキュメント

詳細情報は、以下のドキュメントを参照してください。

- [アプリケーションコンソール](#)
- [アプリケーションリソースの管理](#)
- [Git リポジトリを使用したアプリケーションの管理](#)
- [Helm リポジトリを使用したアプリケーションの管理](#)
- [Object Storage リポジトリを使用したアプリケーションの管理](#)
- [アプリケーションの詳細設定](#)
- [Git リソースのサブスクリプト](#)
- [Ansible Automation Platform タスクのセットアップ](#)
- [チャンネルの例](#)
- [サブスクリプションの例](#)
- [アプリケーションの例](#)

1.2. アプリケーションコンソール

コンソールには、アプリケーションライフサイクル管理用のダッシュボードが含まれます。コンソールダッシュボードを使用し、アプリケーションの作成および管理、アプリケーションステータスの表示が可能です。機能が強化され、開発者およびオペレーションスタッフは全クラスターのアプリケーションの作成、デプロイ、更新、管理、可視化がしやすくなります。

以下のリストのコンソール機能の一部を参照し、制約、アクション、トポロジーの読み取り方法に関するガイド付きの情報は、コンソールを参照してください。

重要: 利用可能なアクションは割り当てられたロールに基づきます。[ロールベースのアクセス制御](#) のドキュメントで、アクセス要件について確認してください。

- 関連するリソースリポジトリやサブスクリプションおよび配置設定など、クラスター全体にデプロイされたアプリケーションを可視化する。
- アプリケーションの作成および編集とリソースのサブスクリプトを行います。**Actions** メニューから、検索、編集、または削除が可能です。**YAML:On** を選択し、フィールドの更新時にYAMLを表示して編集します。
- メインの **Overview** タブから、アプリケーション名をクリックして、リソースリポジトリ、サブスクリプション、Ansible Automation Platform タスクを使用する (git リポジトリ用の) デプロイメント前後のフックなどのデプロイされたリソースなど、詳細やアプリケーションリソースを表示できます。概要からアプリケーションを作成することもできます。

- **ApplicationSet**、**Subscription**、**OpenShift**、**Flux**、および **Argo CD** タイプなどのアプリケーションを作成および表示します。**ApplicationSet** は、このコントローラーから生成される Argo アプリケーションを表します。
 - ArgoCD **ApplicationSet** を作成するには、**Sync policy** から **Automatically sync when cluster state changes** を有効にする必要があります。
 - **kustomization** コントローラーを使用する Flux の場合は、**kustomize.toolkit.fluxcd.io/name=<app_name>** ラベルが付いた Kubernetes リソースを見つけます。
 - **helm** コントローラーを使用する Flux の場合は、**helm.toolkit.fluxcd.io/name=<app_name>** ラベルが付いた Kubernetes リソースを見つけます。
- メインの **Overview** から、表のアプリケーション名をクリックして単一のアプリケーションの概要を表示すると、以下の情報を確認できます。
- リソースのステータスなどのクラスターの詳細
- リソーストポロジー
- サブスクリプションの情報
- Editor タブにアクセスして編集します。
- **Topology** タブをクリックして、プロジェクトのすべてのアプリケーションおよびリソースを視覚的に表示します。Helm サブスクリプションの場合は、[パッケージの上書きの設定](#) を参照し、適切な **packageName** および **packageAlias** を定義して、正確なトポロジー表示を取得します。
- **Advanced configuration** タブをクリックして、全アプリケーションのリソースの用語および表を表示します。リソースを特定し、サブスクリプション、配置、チャンネルをフィルタリングできます。アクセス権がある場合は、編集、検索、削除などの **Actions** も複数、クリックできます。
- Ansible タスクをデプロイしたアプリケーションの prehook または posthook として使用している場合に、成功した Ansible Automation Platform デプロイメントを表示します。
- **Launch resource in Search** をクリックし、関連リソースを検索します。
- **Search** を使用して、各リソースのコンポーネント **kind** 別にアプリケーションリソースを検索します。リソースの検索には、以下の値を使用します。

アプリケーションリソース	Kind (検索パラメーター)
サブスクリプション	Subscription
チャンネル	チャンネル
シークレット	Secret
Placement	Placement
アプリケーション	アプリケーション

また、名前、namespace、クラスター、ラベルなどの他のフィールドで検索することもできます。検索の使用方法は、[コンソールでの検索の概要](#) を参照してください。

1.3. サブスクリプションレポート

サブスクリプションレポートは、フリート内のすべてのマネージドクラスターからのアプリケーションステータスのコレクションです。具体的には、親アプリケーションリソースは、スケーラブルなマネージドクラスターからのレポートを保持できます。

詳細なアプリケーションステータスはマネージドクラスターで利用できますが、ハブクラスターの **subscriptionReports** は軽量でスケーラブルです。以下の3種類のサブ状態レポートを参照してください。

- パッケージレベルの **SubscriptionStatus**: これは、マネージドクラスター上のアプリケーションパッケージのステータスであり、アプリケーションによって **appsub** namespace にデプロイされるすべてのリソースの詳細なステータスが含まれます。
- クラスターレベルの **SubscriptionReport**: 特定のクラスターにデプロイされているすべてのアプリケーションに関する全体的なステータスレポートです。
- アプリケーションレベルの **SubscriptionReport**: これは、特定のアプリケーションがデプロイされたすべてのマネージドクラスターの全体的なステータスレポートです。
 - [SubscriptionStatus パッケージレベル](#)
 - [SubscriptionReport クラスターレベル](#)
 - [SubscriptionReport アプリケーションレベル](#)
 - [managedClusterView](#)
 - [CLI アプリケーションレベルのステータス](#)
 - [CLI 最終更新時間](#)

1.3.1. SubscriptionStatus パッケージレベル

パッケージレベルのマネージドクラスターのステータスは、マネージドクラスターの **<namespace: <your-appsub-namespace>** にあり、アプリケーションでデプロイされたすべてのリソースの詳細ステータスが含まれます。マネージドクラスターにデプロイされるすべての **appsub** に対して、マネージドクラスターの **appsub** namespace に **SubscriptionStatus** CR が作成されます。エラーが存在する場合は、すべてのリソースが詳細エラーと共に報告されます。

パッケージのステータスは、個々のパッケージのステータスのみを示します。**.status.subscription** のフィールドを参照して、サブスクリプション全体のステータスを表示できます。

以下の **SubscriptionStatus** サンプル YAML ファイルを参照してください。

```
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 クラスターレベル

クラスターレベルのステータスはハブクラスターの **<namespace:<your-managed-cluster-1>** にあり、そのマネージドクラスターの各アプリケーションの全体的なステータスのみが含まれます。ハブクラスターの各クラスター namespace の **subscriptionReport** は、以下のステータスのいずれかを報告します。

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

以下の **SubscriptionStatus** サンプル YAML ファイルを参照してください。

```

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 アプリケーションレベル

各アプリケーションのアプリケーションレベルの1つである **subscriptionReport** が、ハブクラスターの **appsub** namespace の **<namespace:<your-appsub-namespace>** にあり、以下の情報が含まれています。

- 各マネージドクラスターのアプリケーション全体のステータス
- アプリケーションのすべてのリソースのリスト
- クラスターの合計数を含むレポートサマリー
- アプリケーションがステータスにあるクラスターの合計数 (**deployed**、**failed**、**propagationFailed**、および **inProgress**) のレポートサマリー。

注記: **inProcess** ステータスは、合計マイナス **deployed**、さらに **failed**、と **propagationFailed** をマイナスします。

以下の **SubscriptionStatus** サンプル YAML ファイルを参照してください。

```

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

ManagedClusterView CR は、最初の **failed** クラスターについて報告されます。リソースのデプロイに失敗した複数のクラスターにアプリケーションがデプロイされた場合、ハブクラスターで最初に失敗したクラスター namespace に対して **managedClusterViewCR** 1つだけが作成されま

す。**managedClusterView** CR は、障害が発生したクラスターから詳細なサブスクリプションステータスを取得するため、アプリケーションの所有者は障害が発生したリモートクラスターにアクセスする必要があります。

以下のコマンドを実行してステータスを取得できます。

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

1.3.5. CLI アプリケーションレベルのステータス

マネージドクラスターにアクセスしてサブスクリプションステータスを取得できない場合は、CLI を使用できます。クラスターレベルまたはアプリケーションレベルのサブスクリプションレポートは、全体のステータスは含まれますが、アプリケーションの詳細なエラーメッセージは含まれません。

1. [multicloud-operators-subscription](#) から CLI をダウンロードします。
2. 以下のコマンドを実行して **managedClusterView** リソースを作成し、エラーを特定できるようにマネージドクラスターアプリケーションの **SubscriptionStatus** を表示します。

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

1.3.6. CLI 最終更新時間

また、各マネージドクラスターにログインしてこの情報を取得する場合は、指定のマネージドクラスターでの AppSub の最終更新時間を取得することもできます。そのため、マネージドクラスターの AppSub の最終更新時間の取得を簡素化するために、ユーティリティスクリプトが作成されました。このスクリプトは、ハブクラスターで実行するように設計されています。これは、マネージドクラスターから AppSub を取得するための **managedClusterView** リソースを作成し、データを解析して最終更新時刻を取得します。

1. [multicloud-operators-subscription](#) から CLI をダウンロードします。
2. 以下のコマンドを実行して、マネージドクラスターの **AppSub** の **最終更新時間** を取得します。このスクリプトは、ハブクラスターで実行するように設計されています。**managedClusterView** リソースを作成してマネージドクラスターから AppSub を取得し、データを解析して最終更新時間を取得します。

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

1.4. アプリケーションリソースの管理

コンソールから、Git リポジトリ、Helm リポジトリ、およびオブジェクトストレージリポジトリを使用してアプリケーションを作成できます。

重要: Git チャンネルは、他のすべてのチャンネルタイプ (Helm、オブジェクトストレージ、およびその他の Git namespace) と namespace を共有できます。

アプリケーションの管理を開始する場合は、以下のトピックを参照してください。

- [Git リポジトリを使用したアプリケーションの管理](#)
- [Helm リポジトリを使用したアプリケーションの管理](#)

- [Object Storage リポジトリを使用したアプリケーションの管理](#)

1.4.1. Git リポジトリを使用したアプリケーションの管理

アプリケーションを使用して Kubernetes リソースをデプロイする場合に、リソースは特定のリポジトリに配置されます。以下の手順で、Git リポジトリからリソースをデプロイする方法を説明します。[アプリケーションモデルおよび定義](#) でアプリケーションモデルについて確認してください。

ユーザーに必要なアクセス権: アプリケーションを作成できるユーザーロールが割り当てられているアクションのみを実行できます。[ロールベースのアクセス制御](#) のドキュメントで、アクセス要件について確認してください。

1. コンソールのナビゲーションメニューから **Applications** をクリックし、リスト表示されているアプリケーションを表示して新規アプリケーションを作成します。
2. **オプション:** 作成するアプリケーションの種類を選択した後に、**YAML: On** を選んで YAML: アプリケーションの作成および編集時に YAML をコンソールで表示できます。このトピックの後半にある YAML サンプルを参照してください。
3. 使用可能なリポジトリのリストから **Git** を選択し、正しいフィールドに値を入力します。コンソールのガイダンスに従い、入力に基づいて YAML エディターの変更値を確認します。

注記:

- 既存の Git リポジトリパスを選択し、そのリポジトリがプライベートの場合は、接続情報を指定する必要がありません。接続情報が事前設定されているため、これらの値を確認する必要はありません。
 - 新しい Git リポジトリパスを入力し、その Git リポジトリがプライベートの場合は、オプションで Git 接続情報を入力できます。
 - 調整オプションに着目します。**merge** オプションは、新規フィールドが追加され、既存フィールドがリソースで更新されることを意味するデフォルトの選択です。**replace** を選択することができます。**replace** オプションを指定すると、既存のリソースが Git ソースに置き換えられます。サブスクリプションの調整速度を **low** に設定すると、サブスクライブしているアプリケーションリソースの調整に最大1時間かかる場合があります。単一アプリケーションビューのカードで、**Sync** をクリックして手動で調整します。**off** に設定すると、調整はありません。
4. デプロイメントの前後のタスクをオプションで設定します。サブスクリプションがアプリケーションリソースをデプロイする前または後に実行する Ansible Automation Platform ジョブがある場合は、Ansible Automation Platform シークレットを設定します。ジョブを定義する Ansible Automation Platform タスクは、このリポジトリの **prehook** および **posthook** フォルダー内に配置する必要があります。
 5. コンソールを使用して認証情報を追加する必要がある場合は、**Add credential** をクリックします。コンソールの指示に従います。詳細は、[認証情報の管理の概要](#) を参照してください。
 6. **Create** をクリックします。
 7. **Overview** ページにリダイレクトされ、詳細とトポロジーを確認できます。

1.4.1.1. その他の例

- **root-subscription/** の例は、[application-subscribe-all](#) を参照してください。

- 同じリポジトリにある他のフォルダーを参照するサブスクリプションの例は、[subscribe-all](#) を参照してください。
- [nginx-apps](#) リポジトリのアプリケーションアーティファクトを含む **common-managed** フォルダーの例を参照してください。
- [Policy collection](#) のポリシーの例を参照してください。

1.4.1.2. Git でサブスクリプションを削除した後、デプロイされたリソースを保持する

Git リポジトリを使用してサブスクリプションを作成する場合、**do-not-delete** アノテーションを追加して、サブスクリプションを削除した後も特定のデプロイ済みリソースを保持できます。**do-not-delete** アノテーションは、最上位のデプロイリソースでのみ機能します。**do-not-delete** アノテーションを追加するには、次の手順を実行します。

1. 少なくとも1つのリソースをデプロイするサブスクリプションを作成します。
2. サブスクリプションを削除した後でも、保持するリソースに次のアノテーションを追加します。

```
apps.open-cluster-management.io/do-not-delete: 'true'
```

以下の例を参照してください。

```
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"
```

サブスクリプションを削除した後、**do-not-delete** アノテーションを持つリソースは引き続き存在しますが、他のリソースは削除されます。

注記: **do-not-delete** アノテーションを使用してデプロイされたままのリソースは、namespace にバインドされます。そのため、残りのリソースを削除するまで namespace を削除することはできません。

1.4.2. Helm リポジトリを使用したアプリケーションの管理

アプリケーションを使用して Kubernetes リソースをデプロイする場合に、リソースは特定のリポジトリに配置されます。以下の手順で、Helm リポジトリからリソースをデプロイする方法を説明します。[アプリケーションモデルおよび定義](#) でアプリケーションモデルについて確認してください。

ユーザーに必要なアクセス権: アプリケーションを作成できるユーザーロールが割り当てられているアクションのみを実行できます。[ロールベースのアクセス制御](#) のドキュメントで、アクセス要件について確認してください。

1. コンソールのナビゲーションメニューから **Applications** をクリックし、リスト表示されているアプリケーションを表示して新規アプリケーションを作成します。
2. **オプション:** 作成するアプリケーションの種類を選択した後に、**YAML: On** を選んで YAML: アプリケーションの作成および編集時に YAML をコンソールで表示できます。このトピックの後半にある YAML サンプルを参照してください。

3. 使用できるリポジトリのリストから **Helm** を選択し、正しいフィールドに値を入力します。コンソールのガイダンスに従い、入力に基づいて YAML エディターの変更値を確認します。
4. **Create** をクリックします。
5. **Overview** ページにリダイレクトされ、詳細とトポロジーを確認できます。

1.4.2.1. YAML 例

以下のチャンネル定義例では Helm リポジトリをチャンネルとして抽象化します。

注記: Helm では、Helm チャートに含まれる全 Kubernetes リソースにはラベルリリースが必要です。アプリケーショントポロジーの `{{ .Release.Name }}` が正しく表示されるようにします。

```
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
```

以下のチャンネル定義は、Helm リポジトリチャンネルの別の例を示しています。

```
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/
```

注記: REST API を確認するには、[API](#) を使用してください。

1.4.2.2. Helm でサブスクリプションを削除した後、デプロイされたリソースを保持する

Helm は、サブスクリプションを削除した後、特定のデプロイされたリソースを保持するためのアノテーションを提供します。詳細については、[Tell Helm Not To Uninstall a Resource](#) を参照してください。

注記: アノテーションは Helm チャートにある必要があります。

1.4.3. Object Storage リポジトリを使用したアプリケーションの管理

アプリケーションを使用して Kubernetes リソースをデプロイする場合に、リソースは特定のリポジトリに配置されます。[アプリケーションモデルおよび定義](#) でアプリケーションモデルについて確認してください。

ユーザーに必要なアクセス権: アプリケーションを作成できるユーザーロールが割り当てられているアクションのみを実行できます。[ロールベースのアクセス制御](#) のドキュメントで、アクセス要件について確認してください。

1. コンソールのナビゲーションメニューから **Applications** をクリックし、リスト表示されているアプリケーションを表示して新規アプリケーションを作成します。
2. **オプション:** 作成するアプリケーションの種類を選択した後に、**YAML: On** を選んで YAML: アプリケーションの作成および編集時に YAML をコンソールで表示できます。このトピックの後半にある YAML サンプルを参照してください。
3. 使用できるリポジトリのリストから **オブジェクトストア** を選択し、正しいフィールドに値を入力します。コンソールのガイダンスに従い、入力に基づいて YAML エディターの変更値を確認します。
4. **Create** をクリックします。
5. **Overview** ページにリダイレクトされ、詳細とトポロジを確認できます。

1.4.3.1. YAML 例

以下のチャンネル定義例では、オブジェクトストレージをチャンネルとして抽象化します。

```
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
```

注記: REST API を確認するには、[API](#) を使用します。

1.4.3.2. Amazon Web Services (AWS) S3 オブジェクトストレージバケットの作成

サブスクリプションを設定して、Amazon Simple Storage Service (Amazon S3) オブジェクトストレージサービスで定義したリソースをサブスクライブできます。以下の手順を参照してください。

1. AWS アカウント、ユーザー名、およびパスワードを使用して [AWS コンソール](#) にログインします。
2. **Amazon S3 > Buckets** に移動し、バケットホームページに移動します。
3. **Create Bucket** をクリックし、バケットを作成します。

4. **AWS リージョン** を選択します。AWS S3 オブジェクトバケットの接続に不可欠です。
5. バケットアクセストークンを作成します。
6. ナビゲーションバーのユーザー名に移動して、ドロップダウンメニューから **My Security Credentials** を選択します。
7. **AWS IAM credentials** タブで **Access keys for CLI, SDK, & API access** に移動し、**Create access key** をクリックします。
8. **Access key ID** と **Secret access key** を保存します。
9. オブジェクト YAML ファイルをバケットにアップロードします。

1.4.3.3. AWS バケットのオブジェクトへのサブスクリプト

1. シークレットでオブジェクトバケットタイプチャンネルを作成し、**AccessKeyID**、**SecretAccessKey**、および **リージョン** を指定して、AWS バケットに接続します。AWS バケットの作成時に、この3つのフィールドが作成されます。
2. URL を追加します。URL に **s3://** または **s3 and aws** のキーワードが含まれる場合は、そのURL で AWS S3 バケットのチャンネルを特定します。たとえば、以下のバケット URL にはすべて AWS s3 バケット識別子が含まれます。

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

注記: バケットを AWS S3 API に接続する場合に、AWS S3 オブジェクトバケット URL は必要ありません。

1.4.3.4. AWS サブスクリプションの例

以下の完全な AWS S3 オブジェクトバケットチャンネルのサンプル YAML ファイルを参照してください。

```
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
```

非推奨: 以下の YAML の例で **kind: PlacementRule** および **kind: Subscription added** と表示されているため、引き続き他の AWS サブスクリプションおよび配置ルールオブジェクトを作成できます。

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: towwhichcluster
  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: towwhichcluster
```

オブジェクトバケットの特定サブフォルダー内にあるオブジェクトをサブスクライブすることもできます。**subfolder** アノテーションをサブスクリプションに追加することで、オブジェクトバケットサブスクリプションがサブフォルダーパス内の全リソースのみを強制的に適用します。

subfolder-1 のアノテーションを **bucket-path** として参照してください。

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

サブフォルダーの完全なサンプルは、以下を参照してください。

```
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: towwhichcluster
```

1.4.3.5. オブジェクトストレージでサブスクリプションを削除した後、デプロイされたリソースを保持する

オブジェクトストレージリポジトリを使用してサブスクリプションを作成する場合、**do-not-delete** アノテーションを追加して、サブスクリプションを削除した後も特定のデプロイ済みリソースを保持できます。**do-not-delete** アノテーションは、最上位のデプロイリソースでのみ機能します。**do-not-delete** アノテーションを追加するには、次の手順を実行します。

1. 少なくとも1つのリソースをデプロイするサブスクリプションを作成します。
2. サブスクリプションを削除した後でも、保持するリソースに次のアノテーションを追加します。

```
apps.open-cluster-management.io/do-not-delete: 'true'
```

以下の例を参照してください。

```
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"
```

サブスクリプションを削除した後、**do-not-delete** アノテーションを持つリソースは引き続き存在しますが、他のリソースは削除されます。

注記: **do-not-delete** アノテーションを使用してデプロイされたままのリソースは、namespace にバインドされます。そのため、残りのリソースを削除するまで namespace を削除することはできません。

1.5. ANSIBLE AUTOMATION PLATFORM の統合と導入

Red Hat Advanced Cluster Management は Red Hat Ansible Automation Platform と統合されているため、Git サブスクリプションアプリケーション管理用のプリフックおよびポストフック **AnsibleJob** インスタンスを作成できます。コンポーネントと Ansible Automation Platform の設定方法について学びます。

必要なアクセス権限: クラスターの管理者

1.5.1. 統合とコンポーネント

Ansible Automation Platform ジョブを Git サブスクリプションに統合できます。たとえば、データベースのフロントエンドおよびバックエンドアプリケーションの場合、Ansible Automation Platform ジョブで Ansible Automation Platform を使用してデータベースをインスタンス化する必要があります。アプリケーションは、Git サブスクリプションによってインストールされます。サブスクリプションを使用してフロントエンドおよびバックエンドアプリケーションをデプロイする **前** に、データベースはインスタンス化されます。

アプリケーションのサブスクリプション Operator は、**prehook** および **posthook** の2つのサブフォルダーを定義できるように強化されています。いずれのフォルダーも Git リポジトリリソースのルートパスにあり、それぞれ Ansible Automation Platform ジョブのプリフックおよびポストフックがすべて含まれています。

Git サブスクリプションが作成されると、プリフックおよびポストフックのすべての **AnsibleJob** のリ

ソースがすべて解析され、オブジェクトとしてメモリーに保存されます。アプリケーションのサブスクリプションコントローラーは、プリフックおよびポストフックの **AnsibleJob** インスタンスを作成するタイミングを決定します。

サブスクリプションカスタムリソースを作成すると、Git ブランチおよび Git パスは Git リポジトリのルートの場所を参照します。Git のルート内の 2 つのサブフォルダー (**prehook** および **posthook**) には最低でも **Kind:AnsibleJob** リソース 1 つが含まれている必要があります。

1.5.1.1. Prehook

アプリケーションサブスクリプションコントローラーは、プリフックフォルダー内のすべての **kind:AnsibleJob** CR をプリフックの **AnsibleJob** オブジェクトとして検索し、新しいプリフックの **AnsibleJob** インスタンスを生成します。新しいインスタンス名はプリフックの **AnsibleJob** オブジェクト名とランダムな接尾辞文字列です。

インスタンス名の例 **database-sync-1-2913063** を参照してください。

アプリケーションサブスクリプションコントローラーは、調整要求を 1 分間のループで再度キューに追加します。プリフックの **AnsibleJob status.AnsibleJobResult** をチェックします。プリフックステータスが **successful** の場合、アプリケーションサブスクリプションは主要なサブスクリプションを引き続きデプロイします。

1.5.1.2. posthook

アプリケーションサブスクリプションステータスが更新されると、サブスクリプションステータスが Subscribed であるか、ステータスが Subscribed のすべてのターゲットクラスターに伝播されている場合、アプリケーションサブスクリプションコントローラーは、ポストフックフォルダー内のすべての **AnsibleJob kind** カスタムリソースをポストフックの **AnsibleJob** オブジェクトとして検索します。次に、posthook **AnsibleJob** インスタンスを新たに生成します。新しいインスタンス名には、**posthook AnsibleJob** のオブジェクト名、その後は無作為に指定した接尾辞の文字列を指定します。

インスタンス名の例 **service-ticket-1-2913849** を参照してください。

{aap_short} を有効にするには、次のトピックを参照してください。

- [Ansible Automation Platform のセットアップ](#)
- [Ansible Automation Platform の設定](#)

1.5.2. Ansible Automation Platform のセットアップ

Ansible Automation Platform ジョブを使用すると、タスクを自動化し、Slack や PagerDuty サービスなどの外部サービスと統合できます。Git リポジトリのリソースルートパスには、アプリケーションのデプロイ、アプリケーションの更新、またはクラスターからのアプリケーションの削除の一部として実行される Ansible Automation Platform ジョブの **prehook** および **posthook** ディレクトリーが含まれません。

必要なアクセス権限: クラスターの管理者

- [前提条件](#)
- [Ansible Automation Platform Resource Operator のインストール](#)

1.5.2.1. 前提条件

- OpenShift Container Platform 4.13 以降をインストールする。

- Ansible Automation Platform をインストールする。サポートされている最新バージョンをインストールするには、[Red Hat Ansible Automation Platform](#) ドキュメントを参照してください。
- Ansible Automation Platform Resource Operator をインストールして、Ansible Automation Platform ジョブを Git サブスクリプションのライフサイクルに接続する。**ベストプラクティス**: Ansible Automation Platform ジョブテンプレートは、実行時、べき等である必要があります。
- **INVENTORY** と **EXTRA VARIABLES** の両方について、テンプレートの **PROMPT ON LAUNCH** を確認する。詳細は、[Job templates](#) を参照してください。

1.5.2.2. Ansible Automation Platform Resource Operator のインストール

1. OpenShift Container Platform クラスターコンソールにログインします。
2. コンソールナビゲーションで **OperatorHub** をクリックします。
3. **Ansible Automation Platform Resource Operator** を検索し、インストールします。**注記**: プリフックおよびポストフックの **AnsibleJobs** を送信するには、Red Hat Ansible Automation Platform Resource Operator を以下の OpenShift Container Platform バージョンで利用可能な対応するバージョンとともにインストールします。
 - OpenShift Container Platform 4.8 には (AAP) リソース Operator の早期アクセス、stable-2.1、stable-2.2 が必要です
 - OpenShift Container Platform 4.9 には (AAP) リソース Operator の早期アクセス、stable-2.1、stable-2.2 が必要です
 - OpenShift Container Platform 4.10 以降には、(AAP) Resource Operator stable-2.1、stable-2.2 が必要です。

その後、コンソールの **Credentials** ページから認証情報を作成できます。**Add credential** をクリックするか、ナビゲーションからページにアクセスします。認証情報については、[Ansible Automation Platform の認証情報の作成](#) を参照してください。

1.5.3. Ansible Automation Platform の設定

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.

必要なアクセス権限: クラスターの管理者

- [Ansible Automation Platform シークレットの設定](#)
- [シークレット調整の設定](#)
- [Ansible Automation Platform サンプル YAML ファイルの使用](#)
- [ワークフローの起動](#)

Ansible Automation Platform 設定は、以下のタスクで行うことができます。

1.5.3.1. Ansible Automation Platform シークレットの設定

このドキュメントは、Red Hat Ansible Automation Platform のインストールと設定に関する情報を提供します。

Ansible Automation Platform シークレットカスタムリソースを同じサブスクリプション namespace に作成する必要があります。Ansible Automation Platform シークレットは、同じサブスクリプション namespace に制限されています。

1. **Ansible Automation Platform secret name** セクションに入力して、コンソールからシークレットを作成します。ターミナルを使用してシークレットを作成するには、サンプルの **yaml** ファイルを編集して適用します。
注記: **namespace** は、サブスクリプションの namespace と同じにします。 **stringData:token** と **host** は、Ansible Automation Platform のものです。

```
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. 以下のコマンドを実行して YAML ファイルを追加します。

```
oc apply -f
```

アプリケーションサブスクリプションコントローラーがプリフックおよびポストフックの Ansible ジョブを作成する際、サブスクリプション **spec.hooksecretref** からのシークレットが利用可能な場合、それは **AnsibleJob** カスタムリソース **spec.tower_auth_secret** に送信され、**AnsibleJob** は Ansible Automation Platform にアクセスできます。

1.5.3.2. シークレット調整の設定

プリフックおよびポストフックの **AnsibleJob** を使用するメインのサブサブスクリプションの場合、すべてのプリフックおよびポストフックの **AnsibleJob** またはメインのサブスクリプションが Git リポジトリで更新された後、メイン/サブのサブスクリプションを調整する必要があります。

プリフックの **AnsibleJob** とメインのサブスクリプションは新しいプリフックの **AnsibleJob** インスタンスを継続的に調整および再起動します。

1. プリフックの **AnsibleJob** が完了したら、メインのサブスクリプションを再実行します。
2. メインのサブスクリプションに仕様変更がある場合は、サブスクリプションを再デプロイします。再デプロイメントの手順に合わせて、メインのサブスクリプションステータスを更新する必要があります。
3. ハブクラスターのサブスクリプションステータスを **nil** にリセットします。サブスクリプションは、ターゲットクラスターにサブスクリプションをデプロイするときに合わせて更新されません。
ターゲットクラスターへのデプロイメントが終了すると、ターゲットクラスターのサブスクリプションステータスが **"subscribed"** または **"failed"** になり、ハブクラスターのサブスクリプションステータスに同期されます。
4. メインのサブスクリプションが完了したら、新しいポストフックの **AnsibleJob** インスタンスを再起動します。
5. サブスクリプションが更新されていることを確認します。以下の出力を参照してください。

- `subscription.status == "subscribed"`
- `subscription.status == "propagated"` with all of the target clusters **"subscribed"**

AnsibleJob カスタムリソースが作成されると、ターゲットの Ansible Automation Platform と通信することによって、Ansible Automation Platform ジョブを起動するために、Kubernetes ジョブのカスタムリソースが作成されます。ジョブが完了すると、ジョブの最終ステータスが **AnsibleJob status.AnsibleJob Result** に戻ります。

注記:

AnsibleJob status.conditions は、Kubernetes ジョブの結果の作成を保存するために、Ansible Automation Platform Job Operator によって予約されています。**status.conditions** は実際の Ansible Automation Platform ジョブのステータスを反映していません。

サブスクリプションコントローラーは、**AnsibleJob.status.conditions** ではなく、**AnsibleJob.status.AnsibleJob.Result** によって、Ansible Automation Platform ジョブのステータスをチェックします。

プリフックおよびポストフックの **AnsibleJob** ワークフローで前述したように、メインのサブスクリプションが Git リポジトリで更新されると、新しいプリフックおよびポストフックの **AnsibleJob** インスタンスが作成されます。その結果、1つのメインサブスクリプションが複数の **AnsibleJob** インスタンスにリンクできます。

subscription.status.ansiblejobs には、4つのフィールドが定義されています。

- **lastPrehookJobs**: 最新のプリフックの Ansible ジョブ
- **prehookJobsHistory**: プリフックの Ansible ジョブのすべての履歴
- **lastPosthookJobs**: 最新のポストフックの Ansible ジョブ
- **posthookJobsHistory**: ポストフックの Ansible ジョブのすべての履歴

1.5.3.3. Ansible Automation Platform サンプル YAML ファイルの使用

Git prehook および posthook フォルダ内の **AnsibleJob** YAML ファイルの次のサンプルを参照してください。

```
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. ワークフローの起動

AnsibleJob カスタムリソースを使用して Ansible Automation Platform ワークフローを起動するには、次の例に示すように、**job_template_name** フィールドを **workflow_template_name** に置き換えます。

1.5.3.5. Ansible Automation Platform のサンプル YAML ワークフローの使用

Git prehook および posthook フォルダ内の Workflow **AnsibleJob** YAML ファイルの次のサンプルを参照してください。

```

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

```

Ansible ワークフローの詳細については、[ワークフロー](#) を参照してください。

1.6. アプリケーションの詳細設定

Red Hat Advanced Cluster Management for Kubernetes では、アプリケーションは複数のアプリケーションリソースで設定されています。チャンネル、サブスクリプション、および配置を使用すると、アプリケーション全体のデプロイ、更新、および管理に役立ちます。

単一クラスターアプリケーションもマルチクラスターアプリケーションも同じ Kubernetes 仕様を使用しますが、マルチクラスターアプリケーションでは、デプロイメントおよびアプリケーション管理ライフサイクルがさらに自動化されます。

Red Hat Advanced Cluster Management for Kubernetes アプリケーションのアプリケーションコンポーネントリソースはすべて、YAML ファイルの仕様セクションで定義します。アプリケーションコンポーネントリソースを作成または更新する必要がある場合は、適切な仕様セクションを作成してリソースを定義するラベルを追加する必要があります。

以下のアプリケーション詳細設定のトピックを確認してください。

- [Git リソースのサブスクライブ](#)
- [サブスクリプション特権の付与](#)
- [サブスクリプション管理者としての許可リストの作成および拒否リストの作成](#)
- [調整オプションの追加](#)
- [リーダー選択の設定](#)
- [セキュアな Git 接続用のアプリケーションチャンネルおよびサブスクリプションの設定](#)
- [Ansible Automation Platform タスクのセットアップ](#)
- [namespace リソースを監視するように Helm を設定する](#)
- [パッケージの上書きの設定](#)
- [チャンネルサンプルの概要](#)
- [サブスクリプションの例の概要](#)
- [アプリケーションの例の概要](#)

1.6.1. Git リソースのサブスクライブ

デフォルトでは、サブスクリプションを使用してサブスクライブされているアプリケーションをターゲットクラスターにデプロイすると、アプリケーションリソースが別の namespace に関連付けられている場合でも、このアプリケーションはこのサブスクリプション namespace にデプロイされます。[サブスクリプションの管理権限の付与](#) で説明されているように、[サブスクリプション管理者](#) はデフォルトの動作を変更できます。

また、アプリケーションリソースがクラスターに存在しており、サブスクリプションを使用して作成されていない場合、このサブスクリプションではその既存リソースに対して新しいリソースを適用できません。サブスクリプション管理者としてデフォルト設定を変更するには、以下のプロセスを参照してください。

必要なアクセス権限: クラスターの管理者

- [Git でのアプリケーションリソースの作成](#)
- [特定の Git 要素のサブスクライブ](#)
- [アプリケーション namespace の例](#)
- [リソース上書きの例](#)

1.6.1.1. Git でのアプリケーションリソースの作成

サブスクライブ時に、リソース YAML で **apiVersion** の完全グループおよびバージョンを指定する必要があります。たとえば、**apiVersion: v1** にサブスクライブすると、サブスクリプションコントローラーがサブスクリプションの検証に失敗し、エラーメッセージ **Resource /v1, Kind=ImageStream is not supported** が表示されます。

以下の例にあるように、**apiVersion** を **image.openshift.io/v1** に変更すると、サブスクリプションコントローラーの検証を渡し、リソースが正常に適用されます。

```
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'
```

次に、サブスクリプション管理者がデフォルト動作を変更する有用な例を確認します。

1.6.1.2. アプリケーション namespace の例

以下の例では、サブスクリプション管理者としてログインします。

1.6.1.2.1. アプリケーションと異なる namespace

サブスクリプションを作成して、サンプルのリソース YAML ファイルを Git リポジトリからサブスクライブします。このサンプルファイルには、以下の異なる namespace にあるサブスクリプションが含まれます。

適用可能なチャネルタイプ: Git

- ConfigMap **test-configmap-1** は **multins** namespace に作成されます。
- ConfigMap **test-configmap-2** は **default** namespace に作成されます。
- ConfigMap **test-configmap-3** は **subscription** namespace に作成されます。

```
---
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

```

他のユーザーがサブスクリプションを作成した場合は、ConfigMap がすべて、サブスクリプションと同じ namespace に作成されます。

1.6.1.2.2. 同じ namespace へのアプリケーション

サブスクリプション管理者は、すべてのアプリケーションリソースを同じ namespace にデプロイする必要がある場合があります。

[サブスクリプション管理者として許可リストと拒否リストを作成する](#) ことにより、すべてのアプリケーションリソースをサブスクリプション namespace にデプロイできます。

apps.open-cluster-management.io/current-namespace-scoped: true アノテーションをサブスクリプションYAML に追加します。たとえば、サブスクリプション管理者が以下のサブスクリプションを作成すると、直前の例の3つの ConfigMap すべてが **subscription-ns** namespace に作成されます。

```

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. リソース上書きの例

適用可能なチャネルタイプ: Git、ObjectBucket (コンソールのオブジェクトストレージ)

注記: **helm** チャートリソースが Helm で管理されているため、リソースの上書きオプションは Git リポジトリから **helm** チャートには適用されません。

この例では、以下の ConfigMap はすでにターゲットクラスターにあります。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap-1
  namespace: sub-ns

```

```
data:
  name: user1
  age: 19
```

Git リポジトリから、以下のリソース YAML ファイル例をサブスクライブして、既存の ConfigMap を置き換えます。**data** 仕様の変更を参照してください。

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

1.6.1.3.1. デフォルトのマージオプション

デフォルトの **apps.open-cluster-management.io/reconcile-option: merge** アノテーションを使用して、Git リポジトリから以下のサンプルリソースの YAML ファイルを表示します。以下の例を参照してください。

```
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
```

サブスクリプション管理者がこのサブスクリプションを作成し、そのサブスクリプションで ConfigMap リソースをサブスクライブする場合は、以下の例のように既存の ConfigMap をマージします。

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

merge オプションを使用すると、サブスクライブしているリソースのエントリが、既存のリソースで作成または更新されます。既存のリソースからエントリは削除されません。

重要: サブスクリプションで上書きする既存のリソースが自動的に別の Operator またはコントローラーで調整されると、リソース設定はサブスクリプションとコントローラーの両方、または Operator により更新されます。このような場合は、この方法を使用しないでください。

1.6.1.3.2. mergeAndOwn オプション

mergeAndOwn では、サブスクライブしているリソースのエントリーが既存のリソースで作成または更新されます。サブスクリプション管理者としてログインして、**apps.open-cluster-management.io/reconcile-option: mergeAndOwn** アノテーションを付けてサブスクリプションを作成します。以下の例を参照してください。

```
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
```

サブスクリプション管理者がこのサブスクリプションを作成し、そのサブスクリプションで ConfigMap リソースをサブスクライブする場合は、以下の例のように既存の ConfigMap をマージします。

```
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
```

前述のように、**mergeAndOwn** オプションを使用すると、サブスクライブしているリソースのエントリーが既存のリソースで作成または更新されます。既存のリソースからエントリーは削除されません。また、**apps.open-cluster-management.io/hosting-subscription** アノテーションを追加して、リソースがサブスクリプションによって所有されていることを示します。サブスクリプションを削除すると、ConfigMap が削除されます。

1.6.1.3.3. replace オプション

サブスクリプション管理者としてログインして、**apps.open-cluster-management.io/reconcile-option: replace** アノテーションを付けてサブスクリプションを作成します。以下の例を参照してください。

```
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
```

サブスクリプション管理者がこのサブスクリプションを作成し、そのサブスクリプションで ConfigMap リソースをサブスクライブする場合は、既存の ConfigMap を以下に置き換えます。

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

1.6.1.4. 特定の Git 要素のサブスクライブ

特定の Git ブランチ、コミット、またはタグをサブスクライブできます。

1.6.1.4.1. 特定のブランチへのサブスクライブ

multicloud-operators-subscription リポジトリに含まれるサブスクリプション operator は、デフォルトで Git リポジトリのデフォルトのブランチにサブスクライブします。別のブランチにサブスクライブする場合は、そのサブスクリプションにブランチ名のアノテーションを指定する必要があります。

以下の YAML ファイルの例では **apps.open-cluster-management.io/git-branch: <branch1>** で異なるブランチを指定する方法を示しています。

```
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. 特定のコミットのサブスクライブ

multicloud-operators-subscription リポジトリに含まれるサブスクリプション operator は、デフォルトで Git リポジトリの指定のブランチに対する最新のコミットにサブスクライブします。特定のコミットにサブスクライブする場合は、サブスクリプションのコミットハッシュで、必要なコミットアノテーションを指定する必要があります。

以下の YAML ファイルの例では **apps.open-cluster-management.io/git-desired-commit: <full commit number>** で異なるコミットを指定する方法を示しています。

```
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 アノテーションは任意で、デフォルトでは **20** に設定されます。この値は、サブスクリプションコントローラーが Git リポジトリから最新のコミット履歴 20 回分を取得するという意味です。随分前の **git-desired-commit** を指定する場合は、必要なコミットに合った **git-clone-depth** を指定する必要があります。

1.6.1.4.3. 特定のタグへのサブスクリプション

multicloud-operators-subscription リポジトリに含まれるサブスクリプション operator は、デフォルトで Git リポジトリの指定のブランチに対する最新のコミットにサブスクリプションします。特定のタグをサブスクリプションする場合は、サブスクリプションにタグのアノテーションを指定する必要があります。

以下の YAML ファイルの例では **apps.open-cluster-management.io/git-tag: <v1.0>** で異なるタグを指定する方法を示しています。

```
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
```

注記: Git のコミットとタグアノテーションの両方が指定された場合は、タグが無視されます。

git-clone-depth アノテーションは任意で、デフォルトでは **20** に設定されます。この値は、サブスクリプションコントローラーが Git リポジトリから最新のコミット履歴 **20** 回分を取得するという意味です。随分前の **git-tag** を指定する場合は、必要なタグのコミットに合った **git-clone-depth** を指定する必要があります。

1.6.2. サブスクリプション管理者権限の付与

サブスクリプションの管理者権限を付与する方法について説明します。**サブスクリプション** 管理者は、デフォルトの動作を変更できます。詳細は、以下のプロセスを参照してください。

1. コンソールから OpenShift Container Platform クラスターにログインします。
2. ユーザーを1つ以上作成します。ユーザー作成に関する詳細は、[ユーザー向けの準備](#) を参照してください。グループまたはサービスアカウントを用意することもできます。
app.open-cluster-management.io/subscription アプリケーションの管理者として、ユーザーを作成します。OpenShift Container Platform では、**サブスクリプション** 管理者はデフォルトの動作を変更できます。これらのユーザーをグループ化してサブスクリプション管理グループを表すことができます。これについては、後ほど例説します。
3. ターミナルから、Red Hat Advanced Cluster Management クラスターにログインします。
4. **open-cluster-management:subscription-admin** ClusterRoleBinding が存在しない場合は作成する必要があります。以下の例を参照してください。

```
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
```

- 以下のコマンドで、次のサブジェクトを **open-cluster-management:subscription-admin** ClusterRoleBinding に追加します。

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

注記: **open-cluster-management:subscription-admin** ClusterRoleBinding にはサブジェクトは初期設定されていません。

サブジェクトは以下の例のように表示されます。

```
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'
```

Service Account は、ユーザーサブジェクトとして使用できます。

1.6.3. サブスクリプション管理者としての許可リストの作成および拒否リストの作成

サブスクリプション管理者は、Git リポジトリアプリケーションのサブスクリプションからアプリケーションを作成し、**allow** リストを使用して、指定された Kubernetes **kind** リソースのみのデプロイメントを可能にします。アプリケーションサブスクリプションに **deny** list を作成して、特定の Kubernetes **kind** リソースのデプロイメントを拒否することもできます。

デフォルトでは、**policy.open-cluster-management.io/v1** リソースはアプリケーションサブスクリプションによってデプロイされません。このデフォルトの動作を回避するには、サブスクリプション管理者がアプリケーションサブスクリプションをデプロイする必要があります。

以下の **allow** および **deny** 仕様の例を参照してください。

```
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

```

以下のアプリケーションサブスクリプション YAML は、ソースリポジトリの **myapplication** ディレクトリからアプリケーションがデプロイされると、ソースリポジトリに他のリソースがある場合でも **v1/Deployment** リソースのみをデプロイすることを指定します。

```

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

```

このアプリケーションのサブスクリプション YAML は、**v1/Service** および **v1/ConfigMap** リソース以外のすべての有効なリソースのデプロイを指定します。API グループ内に個別のリソースの種類をリスト表示する代わりに、**""** を追加して、API グループのすべてのリソースの種類を許可または拒否できます。

1.6.4. 調整オプションの追加

個々のリソースで **apps.open-cluster-management.io/reconcile-option** アノテーションを使用して、サブスクリプションレベルの調整オプションを上書きできます。

たとえば、**apps.open-cluster-management.io/reconcile-option: replace** アノテーションをサブスクリプションに追加し、サブスクライブされた Git リポジトリのリソース YAML に **apps.open-cluster-management.io/reconcile-option: merge** アノテーションを追加すると、そのリソースはターゲットクラスターにマージされます。その他のリソースは置き換えられます。

1.6.4.1. 調整頻度の Git チャネル

チャンネル設定で、不要なリソースの調整を回避するために調整頻度オプション (**high**、**medium**、**low**、および **off**) を選択できるようになりました。これにより、サブスクリプション Operator のオーバーロードを防ぐことができます。

必要なアクセス: 管理者およびクラスター管理者である必要があります。

settings:attribute:<value> に関する以下の定義を参照してください。

- **Off:** デプロイされたリソースは自動的に調整されません。**Subscription** カスタムリソースを変更すると、調整がトリガーされます。ラベルまたはアノテーションを追加または更新できません。
- **Low:** ソースの Git リポジトリに変更がない場合でも、デプロイされたリソースは1時間ごとに自動調整されます。
- **Medium:** これはデフォルトの設定です。サブスクリプション Operator は、3分ごとに現在デプロイされているコミット ID をソースリポジトリの最新コミット ID と比較し、ターゲットクラスターに変更を適用します。リポジトリに変更がない場合でも、すべてのリソースは15分ごとにソース Git リポジトリからターゲットクラスターに再適用されます。
- **High:** ソースの Git リポジトリに変更がない場合でも、デプロイされたリソースは2分ごとに自動調整されます。

これは、サブスクリプションによって参照されるチャンネルカスタムリソースの **apps.open-cluster-management.io/reconcile-rate** アノテーションを使用して設定できます。

name: git-channel の例を参照してください。

```
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
```

上記の例では、**sample/git-channel** を使用するすべてのサブスクリプションの調整頻度は **low** に割り当てられます。

- a. サブスクリプションの調整速度を **low** に設定すると、サブスクライフしているアプリケーションリソースの調整に最大1時間かかる場合があります。単一アプリケーションビューのカードで、**Sync** をクリックして手動で調整します。**off** に設定すると、調整はありません。

チャンネルの **reconcile-rate** 設定に関係なく、サブスクリプションは、**Subscription** カスタムリソースの **apps.open-cluster-management.io/reconcile-rate: off** アノテーションを指定して、自動調整を **off** に設定できます。

以下の **git-channel** の例を参照してください。

```
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
```

チャンネルで **reconcile-rate** が **high** に設定されている場合でも、**git-subscription** によってデプロイされたリソースが自動調整されないことを確認します。

1.6.4.2. Helm チャンネルの調整頻度

サブスクリプション Operator は 15 分ごとに、Helm チャートで現在デプロイされているハッシュを、ソースリポジトリからのハッシュと比較します。変更がターゲットクラスターに適用されます。リソース調整の頻度は、他のアプリケーションのデプロイメントおよび更新のパフォーマンスに影響します。

たとえば、数百のアプリケーションサブスクリプションがあり、すべてのサブスクリプションを頻繁に調整する場合は、調整の応答時間は遅くなります。

アプリケーションの Kubernetes リソースによっては、適切な調整頻度でパフォーマンスを向上できません。

- **Off**: デプロイされたリソースは自動的に調整されません。サブスクリプションカスタムリソースを変更すると、調整がトリガーされます。ラベルまたはアノテーションを追加または更新できます。

- **Low:** サブスクリプション Operator は、1時間ごとに現在デプロイされているハッシュと、ソースリポジトリのハッシュと比較し、変更がある場合はターゲットクラスターに変更を適用します。
- **Medium:** これはデフォルトの設定です。サブスクリプション Operator は、15分ごとに現在デプロイされているハッシュと、ソースリポジトリのハッシュと比較し、変更がある場合はターゲットクラスターに変更を適用します。
- **High:** サブスクリプション Operator は、2分ごとに現在デプロイされているハッシュと、ソースリポジトリのハッシュと比較し、変更がある場合はターゲットクラスターに変更を適用します。

これは、サブスクリプションによって参照される **Channel** カスタムリソースの **apps.open-cluster-management.io/reconcile-rate** アノテーションを使用して設定できます。以下の **helm-channel** の例を参照してください。

以下の **helm-channel** の例を参照してください。

```
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
```

この例では、**sample/helm-channel** を使用するすべてのサブスクリプションの調整頻度は **low** になります。

チャネルの **reconcile-rate** 設定に関係なく、サブスクリプションは、**Subscription** カスタムリソースの **apps.open-cluster-management.io/reconcile-rate: off** アノテーションを指定して、自動調整を **off** に設定できます。

```
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

```

この例では、チャンネルで **reconcile-rate** が **high** に設定されている場合でも、**helm-subscription** によってデプロイされたリソースが自動調整されないことを確認します。

1.6.5. リーダー選択の設定

LeaderElection を使用すると、障害が発生した場合にコントローラーが新しいリーダーを選択するようにリクエストする方法を変更できます。これにより、一度に1つのリーダーインスタンスのみが調整を処理することが保証されます。コントローラーが **LeaderElection** を取得するのにかかる時間を増減できます。時間が短縮されると、障害時に新しいリーダーがより迅速に選択されます。

注記: コントローラーのデフォルト値を変更すると、そのタスク中のシステムパフォーマンスに影響を与える可能性があります。コントローラーの **leaseDuration**、**renewDeadline**、または **retryPeriod** のデフォルト値を変更することで、**etcd** の負荷を減らすことができます。

必要なアクセス権限: クラスターの管理者

1.6.5.1. コントローラーフラグの編集

LeaderElection を設定するには、次のデフォルト値を変更します。

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

multicluster-operators-application、**multicluster-operators-channel**、**multicluster-operators-standalone-subscription**、または **multicluster-operators-hub-subscription** コントローラーを変更するには、次の手順を参照してください。

1. 次のコマンドを実行して、**multiclusterhub** を一時停止します。

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

2. コントローラー名を **oc edit** コマンドに追加して、**deployment** ファイルを編集します。次のコマンド例を参照してください。

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

3. **-command** を検索して、コントローラーコマンドフラグを見つけます。
4. コントローラーのコンテナーセクションから、**-command** フラグを挿入します。たとえば、**RetryPeriod** を挿入します。
5. ファイルを保存します。コントローラーは自動的に再起動してフラグを適用します。
6. 変更するコントローラーごとに、この手順を繰り返します。
7. 次のコマンドを実行して、**multiclusterhub** を再開します。

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

-command の編集が成功した場合の次の出力例を参照してください。ここで、**retryPeriod** フラグは前述のデフォルト時間を 2 倍にして、**52** にします。この値は **LeaderElection** の取得を再試行するために割り当てられます。

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

1.6.6. セキュアな Git 接続用のアプリケーションチャンネルおよびサブスクリプションの設定

Git チャンネルおよびサブスクリプションは、HTTPS または SSH を使用して指定された Git リポジトリに接続します。以下のアプリケーションチャンネル設定は、セキュアな Git 接続に使用できます。

- [ユーザーおよびアクセストークンを使用したプライベートリポジトリへの接続](#)
- [Git サーバーへのセキュアではない HTTPS 接続の設定](#)
- [セキュアな HTTPS 接続でのカスタム CA 証明書の使用](#)
- [Git サーバーへの SSH 接続の設定](#)
- [証明書および SSH キーの更新](#)

1.6.6.1. ユーザーおよびアクセストークンを使用したプライベートリポジトリへの接続

チャンネルおよびサブスクリプションを使用して Git サーバーに接続できます。ユーザーおよびアクセストークンを使用してプライベートリポジトリに接続するには、以下の手順を参照してください。

1. チャンネルと同じ namespace にシークレットを作成します。**user** フィールドを Git ユーザー ID に、**accessToken** フィールドを Git の個人アクセストークンに設定します。値は base64 でエンコードされる必要があります。user および accessToken が設定された以下の例を参照してください。

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

2. シークレットでチャンネルを設定します。**secretRef** が設定された以下の例を参照してください。

```
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. Git サーバーへのセキュアではない HTTPS 接続の設定

開発環境で以下の接続方法を使用し、カスタム署名または自己署名の認証局による SSL 証明書を使用して、プライベートにホストされている Git サーバーに接続できます。ただし、このソリューションは実稼働では推奨されません。

チャンネルの仕様で **insecureSkipVerify: true** を指定します。それ以外の場合は、Git サーバーへの接続が失敗し、以下のようなエラーが表示されます。

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

この方法のチャンネル仕様が追加された以下の例を参照してください。

```
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
```



```
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFI2fAlqFtyum6/8ZN9nZ9K
FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6ladliUOST3w5t92cB
M26t73UCEXMXTCQvnp0ki84PeR1kRk4
-----END CERTIFICATE-----
```

```
# Git server intermediate CA 1
```

```
-----BEGIN CERTIFICATE-----
```

```
MIIIF5DCCA8wCCQDInYMoI7LSDTANBgkqhkiG9w0BAQsFADCBszELMAkGA1UEBhMC
```

```
Q0ExCzAJBgNVBAGMAk9OMRAwDgYDVQQHDAUub3JvbnRvMQ8wDQYDVQQKDAZSZSW
RI
```

```
YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
hvcNAQkBFhByb2t1akByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky
```

```
MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMCswCQYDVQQIDAJPTjEQMA4GA1UEBwwH
```

```
VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBGNVBA
MM
```

```
PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWlxMy5kZXlYwNi5yZWQtY2hl
c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcnVkaGF0LmNvbTCC
AilwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0lc3
U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
6OS3gj738F635inz1KdyhKtIWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNldRHcpQP
upM5pwPC3BZXqvXChlfAy2m4yu7vy0hO/oTzWlwNsoL5xt0Lw4mSyhlEip/t8IU
xn2y8qhm7MilUpXuWWhSYgCrEVqmTcB70Pc2YRZdSFoIMN9Et70MjQN0TXjoktH8
PyASJKIRd+48yROlbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
+SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcNqVfMvaom
gGHCdJihfy3vDhXuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
diNXs4Zn3QSSStC2x2hKnK+Q1rwCSEg/IBawgxGUstBoFH77a+Kwu4Oug9ibtm5z
ISs/JY4Kiy4C2XJoltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSaxaaSXIX+VbzKHf2N
aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/I8xHvt9L
kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
jaaSeRDRoToTb/yN0Xsi69UfK0aBdvhCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
PQVGs/vehQRqLV9uZBigG6oZqdUqEimalHrOcEAHB5RVcnFurz0qNbT+UySjsD63
9yJdCeQbeKAR9SC4hG13EbM/RZhoIgfupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
kh9aBB7FLykK1TeXVuANINU4EMyJ/e+uhNks9ubNJ3vuRuo+ECHsha058yi16JC9
NkZqP+df4Hp85sd+xhrgYieq7QGx2KoxAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
2UGZxMsRjIP6UqB08LsV5ZBAefEIR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
XSUKATuuQw4ctyZLDkUpzrDzgd2Bt+aaWF6sD2YqycaGFwv2YD9t1YID6F4Wh8Mc
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFI2fAlqFtyum6/8ZN9nZ9K
FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6ladliUOST3w5t92cB
M26t73UCEXMXTCQvnp0ki84PeR1kRk4
-----END CERTIFICATE-----
```

```
# Git server intermediate CA 2
```

```
-----BEGIN CERTIFICATE-----
```

```
MIIIF5DCCA8wCCQDInYMoI7LSDTANBgkqhkiG9w0BAQsFADCBszELMAkGA1UEBhMC
```

```
Q0ExCzAJBgNVBAGMAk9OMRAwDgYDVQQHDAUub3JvbnRvMQ8wDQYDVQQKDAZSZSW
RI
```

```

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
LnJqdW5nLWWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmlibGQuY29tMR8wHQYJKoZI
hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMBQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH

VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQQLDANBQ00xRTBDBGNVBA
MM
PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWlxMy5kZXlYwNi5yZWQtY2hl
c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcnVkaGF0LmNvbTCC
AilwDQYJKoZIhvcNAQEBBQADggIPADCCAgocggIBAM3nPK4mOQzaDAo6S3ZJ0lc3
U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9ZxI8Q5fv+wYwHrUOReCp6U/InyQy
6OS3gj738F635inz1KdyhKtIWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNldRHcpQP
upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWlWnsoL5xt0Lw4mSyhlEip/t8IU
xn2y8qhm7MilUpXuwWhSYgCrEvmTcB70Pc2YRZdSFoIMN9Et70MjQN0TXjoktH8
PyASJKIRd+48yROlbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
+SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcNqVfMvaoM
gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
diNXs4Zn3QSSStC2x2hKnK+Q1rwCSEg/IBawgxGUsITboFH77a+Kwu4Oug9ibtm5z
ISs/JY4Kiy4C2XJOltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSaxaaSXIX+VbzkHF2N
aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/I8xHvt9L
kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
jaaSeRDRoToTb/yN0Xsi69UfK0aBdvCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
PQVGs/vehQRqLV9uZBigG6oZqdUqEimalHrOcEAHB5RVcnFurz0qNbT+UySjsD63
9yJdCeQbeKAR9SC4hG13EbM/RZ0IgfupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
kh9aBB7FLykK1TeXVuANINU4EMyJ/e+uhNks9ubNJ3vuRuo+ECHsha058yi16JC9
NkZqP+df4Hp85sd+xhrgYieq7QGx2KoxAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
2UGZxMsRjIP6UqB08LsV5ZBAefEIR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
XSUkATuuQw4ctyZLDkUpzrDzgd2Bt+aawF6sD2YqycaGFwv2YD9t1YID6F4Wh8Mc
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFI2fAlqFtyum6/8ZN9nZ9K
FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6ladliUOST3w5t92cB
M26t73UCExXMXTcQvnp0ki84PeR1kRk4
-----END CERTIFICATE-----

```

- この ConfigMap でチャンネルを設定します。直前の手順の **git-ca** 名を使用した以下の例を参照してください。

```

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. Git サーバーへの SSH 接続の設定

- data** の **sshKey** フィールドに SSH 秘密キーを追加するシークレットを作成します。キーがパスワードで保護されている場合は、**passphrase** フィールドにパスワードを指定します。こ

のシークレットは、チャンネル CR と同じ namespace にある必要があります。**oc** コマンドを使用してこのシークレットを作成し、generic のシークレット **git-ssh-key --from-file=sshKey=./.ssh/id_rsa** を作成してから、base64 でエンコードされた **passphrase** を追加します。以下のサンプルを参照してください。

```
apiVersion: v1
kind: Secret
metadata:
  name: git-ssh-key
  namespace: channel-ns
data:
  sshKey:
    LS0tLS1CRUdJTiBPUeVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdG
    RqRUFuBQUFBQ21GbGN6STFOaTFqZEhJQUFBQUdZbU55ZVhCMEFBQUFHQUFBQUJD
    K3YySHhWSlWcm8zejh1endzV3NWODMvSFVkeEhGeVBmWk5OeE5TQUgcFA3Yk1yR2tIRF
    FPd3J6MGIKOUIRM0tKVXQzWEE0Zmd6NVlrVfVhcTJsZWxxVk1HcXI2WHF2UVJ5Mkc0NkRI
    RVIYUGpabVZMcGVuaGtRYU5HYmpaMmZOdQpWUGpiOVhZRmd4bTNnYUpJU3BNeTFL
    WjQ5MzJvOFByaDZEdzRYVUF1a28wZGdBaDdndVpPaE53b0pVYnNmYIZRc0xMS1RrCnQw
    biZ1anRvd2NEVGx4TlplUjcwbgVUSHdGQTYwekM0elpMNkRPc3RMYjV2LzZhMjFHRIMwVm
    VXQ3YvMlpMOE1sbjVUZWwKSytoUWtxRnJBL3BUc1ozVXNjSG1GUi9PV25FPQotLS0tLUVO
    RCBPUeVOU1NIIFBSSVZBVEUgS0VZLS0tLS0K
  passphrase: cGFzc3cwcmQK
type: Opaque
```

- シークレットでチャンネルを設定します。以下のサンプルを参照してください。

```
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
```

サブスクリプションコントローラーは、提供される Git ホスト名で **ssh-keyscan** を行い、SSH 接続での MITM (Man-in-the-middle: 中間者) 攻撃を防ぐために **known_hosts** リストを構築します。これを省略して非セキュアな接続を確立する場合は、チャンネル設定で **insecureSkipVerify: true** を使用します。これは、特に実稼働環境でのベストプラクティスではありません。

```
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. 証明書および SSH キーの更新

Git チャンネル接続設定で CA 証明書、認証情報、または SSH キーなどの更新が必要な場合は、同じ namespace に新規のシークレットおよび ConfigMap を作成し、そのチャンネルを更新して新規のシークレットおよび ConfigMap を参照する必要があります。詳細は、[セキュアな HTTPS 接続でのカスタム CA 証明書の使用](#) を参照してください。

1.6.7. namespace リソースを監視するように Helm を設定する

デフォルトでは、サブスクリプションがサブスクライブされた Helm リソースをターゲットクラスターにデプロイするときに、アプリケーションリソースが監視されます。namespace スコープのリソースを監視するように Helm チャンネルタイプを設定できます。有効にすると、監視対象の namespace スコープのリソースに対する手動の変更が元に戻されます。

1.6.7.1. 設定

必要なアクセス権限: クラスターの管理者

namespace スコープのリソースを監視するように Helm アプリケーションを設定するには、サブスクリプション定義の **watchHelmNamespaceScopedResources** フィールドの値を **true** に設定します。以下のサンプルを参照してください。

```
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. デプロイメントのスケジュール

Helm チャートやその他のリソースを特定の時間にだけデプロイしたり、変更したりする必要がある場合は、このようなリソースにサブスクリプションを定義して、特定の時間にだけ、デプロイメントを開始することができます。あるいは、デプロイメントを制限することもできます。

たとえば、金曜の午後 10 時から午後 11 時の時間帯を、クラスターにパッチや他のアプリケーションの更新を適用する予定メンテナンス枠として定義できます。

ピークの営業時間に想定外のデプロイメントが実行されないように、特定の時間帯にデプロイメントを開始しないように制限またはブロックできます。たとえば、午前 8 時から午後 8 時までデプロイメントを開始しないように、サブスクリプションに時間帯を定義してピーク時を回避できます。

サブスクリプションに期間を定義することで、すべてのアプリケーションおよびクラスターの更新を調整できます。たとえば、午後 6 時 1 分から午後 11 時 59 分までの間に新しいアプリケーションリソースのみをデプロイするようにサブスクリプションを定義し、また別のサブスクリプションに対して、午前 12 時から午前 7 時 59 分までの間に既存のリソースの更新版のみをデプロイするように定義できます。

サブスクリプションに期間を定義すると、サブスクリプションがアクティブな期間が変わります。期間の定義の一部として、期間内のサブスクリプションを **active** または **blocked** に定義できます。

サブスクリプションがアクティブな場合にだけ、新規リソースまたは変更リソースのデプロイメントが開始されます。サブスクリプションがアクティブであるか、ブロックされているかに関わらず、サブスクリプションは引き続き、新規リソースや変更リソースがないかどうかを監視します。Active または Blocked の設定は、デプロイメントにだけ影響があります。

新しいリソースまたは変更されたリソースが検出されると、期間の定義をもとに、サブスクリプションの次のアクションが決まります。

- **HelmRepo**、**ObjectBucket**、および **Git** タイプのチャンネルに対するサブスクリプションの場合:
- サブスクリプションが **アクティブ** な期間にリソースが検出されると、リソースのデプロイメントが開始されます。
- サブスクリプションでのデプロイメントの実行がブロックされている期間外にリソースが検出された場合は、リソースのデプロイ要求がキャッシュされます。次回サブスクリプションがアクティブになると、キャッシュされた要求が適用され、関連のデプロイメントが開始されます。
- 期間が **ブロック** されると、アプリケーションサブスクリプションで以前にデプロイされたすべてのリソースが残ります。新しい更新は、期間が再度アクティブになるまでブロックされます。

アプリケーションの準期間がブロックされていると、デプロイされたリソースがすべて削除されるとエンドユーザーが誤って判断する場合があります。また、アプリの準期間が再度アクティブになると、元に戻ります。

定義された期間にデプロイメントが開始され、その定義期間終了時を超えてもデプロイメントが実行されている場合は、デプロイメントが完了するまで継続されます。

サブスクリプションの期間を定義するには、必要なフィールドおよび値をサブスクリプションリソース定義 YAML に追加する必要があります。

- 期間の定義では、日付と時間を定義できます。
- 期間タイプも定義できます。このタイプにより、指定の期間中または期間外にデプロイメントを開始できる期間かどうかが決まります。
- 期間タイプが **active** の場合、デプロイメントは、定義した期間中にのみ開始できます。特定のメンテナンス期間にデプロイメントを行う場合に限り、この設定を使用できます。
- 期間タイプが **block** の場合、デプロイメントは、定義した期間中に開始できませんが、それ以外の時間であればいつでも開始できます。この設定は、特定の時間帯のデプロイメントは回避しつつも、必須の重要な更新がある場合に、使用できます。たとえば、セキュリティ関連の更新を午前 10 時から午後 2 時の時間帯以外に実行できるように、期間を定義する場合に、このタイプを使用できます。
- 毎週月曜と水曜に期間を定義するなど、サブスクリプションの期間を複数定義できます。

1.6.9. パッケージの上書きの設定

パッケージが、サブスクリプションに登録されている Helm チャートまたは Kubernetes リソースのサブスクリプション上書き値より優先されるように設定します。

パッケージの上書きを設定するには、**path** フィールドの値として上書きするように、Kubernetes リソース **spec** のフィールドを指定します。**value** フィールドの値として、置き換える値を指定します。

たとえば、サブスクライブしている Helm チャートの Helm リリース **spec** 内の値フィールドを上書きする必要がある場合は、サブスクリプション定義の **path** フィールドを **spec** に設定する必要があります。

```
packageOverrides:
- packageName: nginx-ingress
packageOverrides:
- path: spec
  value: my-override-values ①
```

① **value** フィールドの内容は、Helm 仕様の **spec** フィールドの値を上書きするのに使用します。

- Helm リリースの場合は、**spec** フィールドの上書き値が Helm リリースの **values.yaml** ファイルにマージされ、既存の値を上書きします。このファイルを使用して、Helm リリースの設定可能な変数を取得します。
- Helm リリースのリリース名を上書きする必要がある場合は、定義に **packageOverride** セクションを追加します。以下のフィールドを追加して、Helm リリースの **packageAlias** を定義します。
 - **packageName** (Helm チャートを特定)
 - **packageAlias** (リリース名を上書きすることを指定)

デフォルトでは、Helm リリース名が指定されていない場合は、Helm チャート名を使用してリリースを特定します。同じチャートに複数のリリースがサブスクライブされている場合など、競合が発生する可能性があります。リリース名は、namespace 内の全サブスクリプションで一意である必要があります。作成するサブスクリプションのリリース名が一意でない場合は、エラーが発生します。**packageOverride** を定義して、サブスクリプションに異なるリリース名を設定する必要があります。既存のサブスクリプション内の名前を変更する場合は、先にサブスクリプションを削除してから、希望のリリース名でサブスクリプションを作り直す必要があります。

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

1.6.10. チャネルサンプルの概要

ファイルの構築に使用できる例および YAML 定義を確認します。チャネル (**channel.apps.open-cluster-management.io**) では、Red Hat Advanced Cluster Management for Kubernetes アプリケーションを作成して管理するための、向上された継続的インテグレーション/継続的デリバリー機能 (CI/CD) を提供します。

OpenShift CLI ツールを使用するには、以下の手順を参照します。

- 任意の編集ツールで、アプリケーションの YAML ファイルを作成して保存します。
- 以下のコマンドを実行してファイルを API サーバーに適用します。**filename** は、使用するファイル名に置き換えます。

```
oc apply -f filename.yaml
```

- 以下のコマンドを実行して、アプリケーションリソースが作成されていることを確認します。

-

```
oc get application.app
```

- [チャンネル YAML の構造](#)
- [チャンネル YAML 表](#)
- [オブジェクトストレージバケット \(ObjectBucket\) チャンネル](#)
- [Helm リポジトリ \(HelmRepo\) チャンネル](#)
- [Git \(Git\) リポジトリチャンネル](#)

1.6.10.1. チャンネル YAML の構造

デプロイできるアプリケーションサンプルについては、[Istolostron](#) リポジトリを参照してください。

以下の YAML 構造は、チャンネルの必須フィールドと、一般的な任意のフィールドの一部を示しています。YAML 構造には、必須なフィールドおよび値を追加する必要があります。アプリケーション管理要件によっては、他の任意のフィールドおよび値を追加しないといけない場合があります。独自の YAML コンテンツは、任意のツールや、製品コンソールで作成できます。

```
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. チャンネル YAML 表

フィールド	任意または必須	説明
apiVersion	必須	この値は apps.open-cluster-management.io/v1 に設定します。
kind	必須	この値は Channel に設定して、リソースがチャンネルであることを指定します。
metadata.name	必須	チャンネルの名前。

フィールド	任意または必須	説明
metadata.namespace	必須	チャンネルの namespace。各チャンネルには Git チャンネルを除き、一意の namespace が必要です。
spec.sourceNamespaces	任意	チャンネルコントローラーが取得してチャンネルにプロモートする新規または更新された deployable がないかを監視する namespace を指定してします。
spec.type	必須	チャンネルタイプ。サポート対象のタイプは、 HelmRepo 、 Git 、および ObjectBucket (コンソールのオブジェクトストレージ) です。
spec.pathname	HelmRepo 、 Git 、 ObjectBucket チャンネルには必須。	HelmRepo チャンネルの場合は、値を Helm リポジトリの URL に設定します。 ObjectBucket チャンネルの場合は、値をオブジェクトストレージの URL に設定します。 Git チャンネルの場合は、値を Git リポジトリの HTTPS URL に設定します。
spec.secretRef.name	任意	リポジトリまたはチャートへのアクセスなど、認証に使用する Kubernetes Secret リソースを指定します。シークレットは、 HelmRepo 、 ObjectBucket 、および Git タイプのチャンネルでのみ認証に使用できます。
spec.gates	任意	チャンネル内での deployable のプロモート要件を定義します。要件が設定されていない場合は、チャンネルの namespace またはソースに追加された deployable がそのチャンネルにプロモートされます。 gates は、 ObjectBucket チャンネルタイプだけに適用され、 HelmRepo や Git チャンネルタイプには適用されません。
spec.gates.annotations	任意	チャンネルのアノテーション。チャンネル内では deployable に同じアノテーションを追加する必要があります。

フィールド	任意または必須	説明
metadata.labels	任意	チャンネルのラベル。
spec.insecureSkipVerify	任意	デフォルト値は false で、 true に設定されると認証を省略してチャンネル接続が作成されます。

チャンネルの定義構造は、以下の YAML コンテンツのようになります。

```
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. オブジェクトストレージバケット (ObjectBucket) チャンネル

以下のチャンネル定義例では、オブジェクトストレージバケットをチャンネルとして抽象化します。

```
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 リポジトリ (HelmRepo) チャンネル

以下のチャンネル定義例では Helm リポジトリをチャンネルとして抽象化します。

非推奨に関する注記: 2.10 では、チャンネルの **ConfigMap** 参照に **insecureSkipVerify: "true"** を指定して Helm リポジトリの SSL 証明書を省略することが非推奨となりました。以下のサンプルで、チャンネルで代わりに使用される **spec.insecureSkipVerify: true** に置き換えられていることを確認してください

い。

```

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

```

以下のチャネル定義は、Helm リポジトリチャネルの別の例を示しています。

注記: Helm の場合は、アプリケーショントポロジーが正しく表示されるように、Helm チャートに含まれる Kubernetes リソースにはラベルリリース {{ **.Release.Name** }} を含める必要があります。

```

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) リポジトリチャネル

以下のチャネル定義例は、Git リポジトリのチャネルの例を示しています。以下の例では、**secretRef** は、**pathname** で指定されている Git リポジトリにアクセスするときに使用するユーザー ID を参照します。パブリックリポジトリを使用する場合は、**secretRef** ラベルと値は必要ありません。

```

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. サブスクリプションの例の概要

ファイルの構築に使用できる例および YAML 定義を確認します。チャンネルと同様に、サブスクリプション (**subscription.apps.open-cluster-management.io**) は、アプリケーション管理用に、向上された継続的インテグレーション/継続的デリバリー (CI/CD) 機能を提供します。

OpenShift CLI ツールを使用するには、以下の手順を参照します。

- 任意の編集ツールで、アプリケーションの YAML ファイルを作成して保存します。
- 以下のコマンドを実行してファイルを API サーバーに適用します。 **filename** は、使用するファイル名に置き換えます。

```
oc apply -f filename.yaml
```

- 以下のコマンドを実行して、アプリケーションリソースが作成されていることを確認します。

```
oc get application.app
```

- [サブスクリプションの YAML 構造](#)
- [サブスクリプションの YAML 表](#)
- [サブスクリプションファイルの例](#)
 - [サブスクリプションの時間枠の例](#)
 - [上書きを使用したサブスクリプションの例](#)
 - [Helm リポジトリのサブスクリプションの例](#)
 - [Git リポジトリのサブスクリプションの例](#)

1.6.11.1. サブスクリプションの YAML 構造

以下の YAML 構造は、サブスクリプションの必須フィールドと、一般的な任意のフィールドの一部を示しています。YAML 構造には、特定の必須フィールドおよび値を追加する必要があります。

アプリケーション管理要件によっては、他の任意のフィールドおよび値を追加しないといけない場合があります。独自の YAML コンテンツは、どのツールでも作成できます。

```
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. サブスクリプションのYAML表

フィールド	必須またはオプション	説明
apiVersion	必須	この値は apps.open-cluster-management.io/v1 に設定します。
kind	必須	この値は Subscription に設定して、リソースがチャンネルであることを指定します。
metadata.name	必須	サブスクリプションを識別する名前。
metadata.namespace	必須	サブスクリプションに使用する namespace リソース。
metadata.labels	任意	サブスクリプションのラベル。

フィールド	必須またはオプション	説明
spec.channel	任意	サブスクリプションのチャンネルを定義する namespace 名 ("Namespace/Name")。 channel フィールド、 source フィールド、または sourceNamespace フィールドを定義します。通常、 source フィールドまたは sourceNamespace フィールドを使用する代わりに、 channel フィールドを使用してチャンネルを参照します。複数のフィールドが定義されている場合は、定義されている最初のフィールドが使用されます。
spec.sourceNamespace	任意	Deployable を保存するハブクラスター上のソース namespace。このフィールドは namespace チャンネルにのみ使用してください。 channel フィールド、 source フィールド、または sourceNamespace フィールドを定義します。通常、 source フィールドまたは sourceNamespace フィールドを使用する代わりに、 channel フィールドを使用してチャンネルを参照します。
spec.source	任意	deployable の保存先である Helm リポジトリのパス名 ("URL")。このフィールドは、Helm リポジトリチャンネルにだけ使用します。 channel フィールド、 source フィールド、または sourceNamespace フィールドを定義します。通常、 source フィールドまたは sourceNamespace フィールドを使用する代わりに、 channel フィールドを使用してチャンネルを参照します。
spec.name	HelmRepo タイプのチャンネルには必須ですが、 ObjectBucket タイプのチャンネルには任意です。	チャンネル内にあるターゲットの Helm チャートまたは deployable の固有名。任意のフィールドである name や packageFilter が定義されていない場合には、すべての deployables が検出され、各 deployable の最新バージョンが取得されます。

フィールド	必須またはオプション	説明
spec.packageFilter	任意	ターゲットの deployable または deployable のサブセットを検索するのに使用するパラメーターを定義します。複数のフィルター条件が定義されている場合、deployable はすべてのフィルター条件を満たす必要があります。
spec.packageFilter.version	任意	deployable のバージョン。バージョンの範囲には >1.0 または <3.0 の形式を使用できます。デフォルトでは、"creationTimestamp" の値が最新のバージョンが使用されます。
spec.packageFilter.annotations	任意	deployable のアノテーション。
spec.packageOverrides	任意	チャンネル内の Helm チャート、deployable、他の Kubernetes リソースなど、サブスクリプションで取得する Kubernetes リソースの上書きを定義するセクションです。
spec.packageOverrides.packageName	任意ですが、オーバーライドの設定には必須です。	上書きされる Kubernetes リソースを特定します。
spec.packageOverrides.packageAlias	任意	上書きされる Kubernetes リソースにエイリアスを指定します。
spec.packageOverrides.packageOverrides	任意	Kubernetes リソースの上書きに使用するパラメーターおよび代替値の設定。
spec.placement	必須	deployable を配置する必要があるサブスクライブクラスター、またはクラスターを定義する配置ルールを特定します。配置設定を使用して、マルチクラスターデプロイメントの値を定義します。

フィールド	必須またはオプション	説明
spec.placement.local	任意ですが、スタンドアロンクラスターまたは直接管理するクラスターには必須です。	<p>サブスクリプションをローカルにデプロイする必要があるかどうかを定義します。</p> <p>サブスクリプションと、指定のチャンネルを同期させるには、値を true に設定します。</p> <p>指定のチャンネルからリソースをサブスクライブしないようにするには、この値を false に設定します。</p> <p>クラスターがスタンドアロンクラスターの場合や、このクラスターを直接管理している場合は、このフィールドを使用します。クラスターがマルチクラスターに含まれており、クラスターを直接管理する必要がない場合は、clusters、clusterSelector、または placementRef の1つだけを使用してサブスクリプションの配置先を定義します。クラスターがマルチクラスターのハブで、クラスターを直接管理する必要がある場合は、サブスクリプション Operator がローカルのリソースにサブスクライブする前に、ハブをマネージドクラスターとして登録しておく必要があります。</p>
spec.placement.clusters	任意	<p>サブスクリプションを配置するクラスターを定義します。clusters、clusterSelector、または placementRef の1つだけを使用して、サブスクリプションをマルチクラスターのどの部分に配置するかを定義します。クラスターが、ハブクラスターではないスタンドアロンクラスターの場合は、local cluster も使用できます。</p>
spec.placement.clusters.name	任意ですが、サブスクライブするクラスターを定義するには必須です。	サブスクライブするクラスターの名前です。

フィールド	必須またはオプション	説明
spec.placement.clusterSelector	任意	サブスクリプションを配置するクラスターを識別するために使用するラベルセレクターを定義します。 clusters 、 clusterSelector 、または placementRef の1つだけを使用して、サブスクリプションをマルチクラスターのどの部分に配置するかを定義します。クラスターが、ハブクラスターではないスタンドアロンクラスターの場合は、 local cluster も使用できます。
spec.placement.placementRef	任意	サブスクリプションに使用する配置ルールを定義します。 clusters 、 clusterSelector 、または placementRef の1つだけを使用して、サブスクリプションをマルチクラスターのどの部分に配置するかを定義します。クラスターが、ハブクラスターではないスタンドアロンクラスターの場合は、 local cluster も使用できます。
spec.placement.placementRef.name	任意ですが、配置ルールを使用するには必須です。	サブスクリプションの配置ルールの名前です。
spec.placement.placementRef.kind	任意ですが、配置ルールを使用するには必須です。	この値を Placement に設定して、サブスクリプションでのデプロイメントに使用する配置ルールを指定します。
spec.overrides	任意	クラスター固有の設定など、上書きする必要のあるパラメーターおよび値。
spec.overrides.clusterName	任意	パラメーターおよび値を上書きするクラスターの名前。
spec.overrides.clusterOverrides	任意	上書きするパラメーターおよび値の設定。
spec.timeWindow	任意	サブスクリプションがアクティブな期間、またはブロックされる期間の設定を定義します。

フィールド	必須またはオプション	説明
spec.timeWindow.type	任意ですが、期間の設定には必須	設定した期間中に、サブスクリプションがアクティブであるか、ブロックされるかを指定します。サブスクリプションのデプロイメントは、サブスクリプションがアクティブな場合にのみ行われます。
spec.timeWindow.location	任意ですが、期間の設定には必須	設定した期間のタイムゾーン。タイムゾーンはすべて Time Zone (tz) データベース名の形式を使用する必要があります。詳細は、 Time Zone Database を参照します。
spec.timeWindow.daysOfWeek	任意ですが、期間の設定には必須	期間の作成時に時間の範囲を適用する場合は、曜日を指定します。 daysOfWeek: ["Monday", "Wednesday", "Friday"] などのように、曜日は配列として定義する必要があります。
spec.timeWindow.hours	任意ですが、期間の設定には必須	期間の範囲を定義します。期間ごとに、開始時間と終了時間 (時間単位) を定義する必要があります。サブスクリプションには複数の期間を定義する必要があります。
spec.timeWindow.hours.start	任意ですが、期間の設定には必須	期間の開始を定義するタイムスタンプです。タイムスタンプには、Go プログラミング言語の Kitchen 形式 "hh:mmpm" を使用する必要があります。詳細は、 Constants を参照してください。
spec.timeWindow.hours.end	任意ですが、期間の設定には必須	期間の終了を定義するタイムスタンプです。タイムスタンプには、Go プログラミング言語の Kitchen 形式 "hh:mmpm" を使用する必要があります。詳細は、 Constants を参照してください。

注記:

- YAML の定義時には、サブスクリプションは **packageFilters** を使用して複数の Helm ダート、deployable、またはその他の Kubernetes リソースを参照できます。ただし、サブスクリプションは、チャート、deployable、その他のリソースの最新バージョンのみをデプロイします。

- 期間の範囲を定義する場合には、開始時間は、終了時間より前に設定する必要があります。サブスクリプションに複数の期間を定義する場合は、期間の範囲を重複させることができません。実際の時間の範囲は、**subscription-controller** のコンテナの時間をもとにしていますが、作業環境とは異なる時間および場所を設定することができます。
- サブスクリプション仕様では、サブスクリプションの定義の一部として Helm リリースの配置を定義することもできます。サブスクリプションごとに、既存の配置ルールを参照するか、サブスクリプション定義内に直接配置ルールを定義できます。
- **spec.placement** セクションに、サブスクリプションの配置先を定義する時には、マルチクラスター環境の **clusters**、**clusterSelector**、または **placementRef** の1つだけを使用します。
- 複数の配置設定を追加した場合は、1つの設定が使用され、他の設定は無視されます。サブスクリプション Operator が使用する設定を決定するために、以下の優先順位で使用されます。
 - a. **placementRef**
 - b. **clusters**
 - c. **clusterSelector**

サブスクリプションは、以下の YAML コンテンツのようになります。

```
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: towwhichcluster
  overrides:
    - clusterName: "/"
  clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.6.11.3. サブスクリプションファイルの例

デプロイできるアプリケーションサンプルについては、[Istolostron](#) リポジトリを参照してください。

```
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. セカンダリーチャンネルの例

ミラーリングされたチャンネル (アプリケーションソースリポジトリ) がある場合は、サブスクリプション YAML に **secondaryChannel** を指定できます。アプリケーションサブスクリプションがプライマリーチャンネルを使用してリポジトリサーバーへの接続に失敗した場合、セカンダリーチャンネルを使用してリポジトリサーバーに接続します。セカンダリーチャンネルに保存されるアプリケーション manifests がプライマリーチャンネルと同期されていることを確認します。**secondaryChannel** の以下のサブスクリプション YAML のサンプルを参照してください。

```

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. サブスクリプションの時間枠の例

以下のサブスクリプションの例には、設定された時間枠が複数含まれています。指定の時間枠は、毎週月曜、水曜、金曜の午前 10 時 20 分から午前 10 時半の間と、毎週月曜、水曜、金曜の午後 12 時 40 分から午後 1 時 40 分の間です。1 週間でこの 6 つの時間枠内にだけ、サブスクリプションがアクティブで、デプロイメントを開始できます。

```

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: towwhichcluster
  timewindow:
    windowtype: "active"
    location: "America/Los_Angeles"
    daysofweek: ["Monday", "Wednesday", "Friday"]
    hours:

```

```
- start: "10:20AM"
  end: "10:30AM"
- start: "12:40PM"
  end: "1:40PM"
```

timewindow には、タイプの目的に応じて、**active** または **blocked** を入力します。

1.6.11.4.2. 上書きを使用したサブスクリプションの例

以下の例には、パッケージの上書きが含まれており、Helm チャートの Helm リリースに異なるリリース名を定義します。パッケージの上書き設定は、**nginx-ingress** Helm リリースの別のリリース名として、**my-nginx-ingress-releaseName** の名前を設定するために使用します。

```
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 リポジトリのサブスクリプションの例

以下のサブスクリプションは、バージョンが **1.36.x** の最新の **nginx** Helm リリースを自動的にプルします。ソースの Helm リポジトリで新規バージョンが利用できる場合、Helm リリースの **deployable** は **my-development-cluster-1** クラスタに配置されます。

spec.packageOverrides セクションでは、Helm リリースの上書き値の任意パラメーターを指定します。上書き値は、Helm リリースの **values.yaml** ファイルにマージされ、このファイルを使用して Helm リリースの設定可能な値を取得します。

```
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 リポジトリのサブスクリプションの例

1.6.11.4.4.1. Git リポジトリの特定ブランチおよびディレクトリーのサブスクリプション

```

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

```

このサブスクリプションの例では、**apps.open-cluster-management.io/git-path** のアノテーションは、チャンネルに指定されている Git リポジトリの **sample_app_1/dir1** ディレクトリーにある Helm チャートおよび Kubernetes リソースのすべてをサブスクリプションがサブスクライブするように指定します。サブスクリプションは、デフォルトで **master** ブランチにサブスクライブします。このサブスクリプションの例では、**apps.open-cluster-management.io/git-branch: branch1** のアノテーションを指定して、リポジトリの **branch1** ブランチをサブスクライブしています。

注意: Helm チャートにサブスクライブする Git チャンネルサブスクリプションを使用している場合には、リソーストポロジビューには追加の **Helmrelease** リソースが表示される可能性があります。このリソースは内部アプリケーションの管理リソースであるため、無視しても問題はありません。

1.6.11.4.4.2. .kubernetesignore ファイルの追加

Git リポジトリの root ディレクトリー、またはサブスクリプションのアノテーションで指定した **apps.open-cluster-management.io/git-path** ディレクトリーに **.kubernetesignore** ファイルを追加できます。

この **.kubernetesignore** ファイルを使用して、サブスクリプションが、そのリポジトリから Kubernetes リソースまたは Helm チャートをデプロイするときに無視するファイルまたはサブディレクトリ、もしくはその両方を指定することができます。

また、**.kubernetesignore** ファイルを使用して、詳細に絞り込み、選択した Kubernetes リソースだけを適用することもできます。**.kubernetesignore** ファイルのパターン形式は、**.gitignore** ファイルと同じです。

apps.open-cluster-management.io/git-path アノテーションが定義されていないと、サブスクリプションは、リポジトリの root ディレクトリで **.kubernetesignore** ファイルを検索します。**apps.open-cluster-management.io/git-path** フィールドが定義されていると、サブスクリプションは **apps.open-cluster-management.io/github-path** ディレクトリで **.kubernetesignore** ファイルを検索します。サブスクリプションは、他のディレクトリでは **.kubernetesignore** ファイルの検索は行いません。

1.6.11.4.4.3. Kustomize の適用

サブスクライブする Git のフォルダーに **kustomization.yaml** ファイルまたは **kustomization.yml** ファイルがある場合は、**kustomize** が適用されます。**spec.packageOverrides** を使用して、サブスクリプションのデプロイメント時に **kustomization** を上書きできます。

```
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
```

kustomization.yaml ファイルを上書きするには、**packageOverrides** に **packageName: kustomization** が必要です。上書きは、新規エントリを追加するか、既存のエントリを更新します。既存のエントリは削除されません。

1.6.11.4.4.4. GitHub Webhook の有効化

デフォルトでは、Git チャンネルのサブスクリプションは、チャンネルで指定されている GitHub リポジトリを 1 分ごとにクローンし、コミット ID が変更されたら、変更が適用されます。または、リポジトリのプッシュまたはプル Webhook イベント通知を Git リポジトリが送信する場合にのみ、変更を適用するようにサブスクリプションを設定できます。

Git リポジトリで Webhook を設定するには、ターゲット Webhook ペイロード URL と、シークレット (任意) が必要です。

1.6.11.4.4.4.1. ペイロード URL

ハブクラスターでルート (ingress) を作成し、サブスクリプション Operator の Webhook イベントリスナーサービスを公開します。

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

次に、**oc get route multicluster-operators-subscription -n open-cluster-management** コマンドを使用して、外部からアクセスできるホスト名を見つけます。

webhook のペイロード URL は <https://<externally-reachable hostname>/webhook> です。

1.6.11.4.4.2. Webhook シークレット

Webhook シークレットは任意です。チャンネル namespace に Kubernetes Secret を作成します。シークレットには **data.secret** を含める必要があります。

以下の例を参照してください。

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

data.secret の値は、使用する base-64 でエンコードされた WebHook シークレットに置き換えます。

ベストプラクティス: Git リポジトリごとに一意のシークレットを使用してください。

1.6.11.4.4.3. Git リポジトリでの Webhook の設定

ペイロード URL および Webhook シークレットを使用して Git リポジトリで Webhook を設定します。

1.6.11.4.4.4. チャンネルでの Webhook イベント通知の有効化

サブスクリプションチャンネルにアノテーションを追加します。以下の例を参照してください。

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

WebHook の設定にシークレットを使用した場合は、これについても、チャンネルにアノテーションを付けます。**<the_secret_name>** は Webhook シークレットを含む Kubernetes Secret 名に置き換えます。

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

サブスクリプションには Webhook 固有の設定は必要ありません。

1.6.12. Placement ルールのサンプルの概要 (非推奨)

非推奨: PlacementRules は非推奨になりました。代わりに **Placement** を使用してください。

配置ルール (**placementrule.apps.open-cluster-management.io**) は、deployable をデプロイ可能なターゲットクラスターを定義します。配置ルールを使用すると、deployable のマルチクラスターでのデプロイメントが容易になります。

OpenShift CLI ツールを使用するには、以下の手順を参照します。

- a. 任意の編集ツールで、アプリケーションの YAML ファイルを作成して保存します。
- b. 以下のコマンドを実行してファイルを API サーバーに適用します。 **filename** は、使用するファイル名に置き換えます。

```
oc apply -f filename.yaml
```

- c. 以下のコマンドを実行して、アプリケーションリソースが作成されていることを確認します。

```
oc get application.app
```

- [配置ルールの YAML 構造](#)
- [配置ルールの YAML 値の表](#)
- [配置ルールファイルの例](#)

1.6.12.1. 配置ルールの YAML 構造

以下の YAML 構造は、配置ルールの必須フィールドと、一般的な任意のフィールドの一部を示しています。YAML 構造には、必須なフィールドおよび値を追加する必要があります。アプリケーション管理要件によっては、他の任意のフィールドおよび値を追加しないといけない場合があります。独自の YAML コンテンツは、任意のツールや、製品コンソールで作成できます。

```
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. 配置ルールの YAML 値の表

フィールド	必須またはオプション	説明
apiVersion	必須	この値は apps.open-cluster-management.io/v1 に設定します。
kind	必須	この値は PlacementRule に設定して、リソースが配置ルールであることを指定します。
metadata.name	必須	配置ルールを識別する名前。
metadata.namespace	必須	配置ルールに使用する namespace リソース。
metadata.resourceVersion	任意	配置ルールのリソースのバージョン。
metadata.labels	任意	配置ルールのラベル。
spec.clusterSelector	任意	ターゲットクラスターを特定するラベル。
spec.clusterSelector.matchLabels	任意	ターゲットクラスターに含める必要があるラベル。
spec.clusterSelector.matchExpressions	任意	ターゲットクラスターに含める必要があるラベル。
status.decisions	任意	Deployable を配置するターゲットクラスターを定義します。
status.decisions.clusterName	任意	ターゲットクラスターの名前。
status.decisions.clusterNamespace	任意	ターゲットクラスターの namespace。
spec.clusterReplicas	任意	作成するレプリカの数。
spec.clusterConditions	任意	クラスターの条件を定義します。
spec.ResourceHint	任意	以前のフィールドで指定したラベルと値に複数のクラスターが一致した場合は、リソース固有の基準を指定してクラスターを選択することができます。たとえば、利用可能な CPU コアの最大数で、クラスターを選択できます。

フィールド	必須またはオプション	説明
spec.ResourceHint.type	任意	この値を cpu に設定して、利用可能な CPU コア数をもとにクラスターを選択するか、 memory に設定して、利用可能なメモリーリソースをもとにクラスターを選択することができます。
spec.ResourceHint.order	任意	この値は、昇順の場合は asc に、または降順の場合は desc に設定します。
spec.Policies	任意	配置ルールのポリシーフィルター。

1.6.12.3. 配置ルールファイルの例

デプロイできるアプリケーションサンプルについては、[Istolostron](#) リポジトリを参照してください。

既存の配置ルールに、以下のフィールドを追加して、配置ルールのステータスを指定することができます。このステータスのセクションは、ルールの YAML 構造の **spec** セクションの後に追加できます。

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

フィールド	説明
status	配置ルールのステータス情報。
status.decisions	Deployable を配置するターゲットクラスターを定義します。
status.decisions.clusterName	ターゲットクラスターの名前。
status.decisions.clusterNamespace	ターゲットクラスターの namespace。

- 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
```

- 例 2

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: towwhichcluster
  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. アプリケーションの例

ファイルの構築に使用できる例および YAML 定義を確認します。Red Hat Advanced Cluster Management for Kubernetes のアプリケーション (**Application.app.k8s.io**) は、アプリケーションコンポーネントの表示に使用します。

OpenShift CLI ツールを使用するには、以下の手順を参照します。

- 任意の編集ツールで、アプリケーションの YAML ファイルを作成して保存します。
- 以下のコマンドを実行してファイルを API サーバーに適用します。**filename** は、使用するファイル名に置き換えます。

```
oc apply -f filename.yaml
```

- 以下のコマンドを実行して、アプリケーションリソースが作成されていることを確認します。

```
oc get application.app
```

- [アプリケーションの YAML 構造](#)
- [アプリケーションの YAML 表](#)
- [アプリケーションファイルの例](#)

1.6.13.1. アプリケーションの YAML 構造

アプリケーション定義 YAML コンテンツを作成して、アプリケーションリソースを作成または更新するには、YAML 構造に、必須のフィールドおよび値を追加する必要があります。アプリケーション要件やアプリケーション管理の要件によっては、他の任意のフィールドや値を追加しないといけない場合があります。

以下の YAML 構造は、アプリケーションの必須フィールドと、一般的な任意のフィールドの一部を示しています。

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

1.6.13.2. アプリケーションの YAML 表

フィールド	値	説明
apiVersion	app.k8s.io/v1beta1	必須
kind	アプリケーション	必須
metadata		
	name: アプリケーションリソースを識別する名前	必須
	namespace: アプリケーションに使用する namespace リソース	
spec		
selector.matchLabels	このアプリケーションを関連付けるサブスクリプションにある Kubernetes ラベルと値の key:value ペア。ラベルを使用すると、ラベル名と値を照合させることで、アプリケーションリソースは関連のあるサブスクリプションを検索できます。	必須

これらのアプリケーションの定義仕様は、Kubernetes Special Interest Group (SIG) が提供するアプリケーションメタデータ記述子のカスタムリソース定義が基になっています。テーブルに表示される値のみが必要です。

この定義を使用すると、独自のアプリケーションのYAMLコンテンツ作成に役立ちます。この定義の詳細は、[Kubernetes SIG Application CRD community specification](#) を参照してください。

1.6.13.3. アプリケーションファイルの例

デプロイできるアプリケーションサンプルについては、[Istio/istio](#) リポジトリを参照してください。

アプリケーションの定義構造は、以下のYAMLコンテンツの例のようになります。

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