Red Hat Advanced Cluster Management for Kubernetes 2.2

アプリケーションの管理

詳細は、Git リポジトリー、Helm リポジトリー、およびオブジェクトストレージリポジトリーを使用してアプリケーションを作成する方法について参照してください。

Last Updated: 2023-02-22

Red Hat Advanced Cluster Management for Kubernetes 2.2 アプリケーションの管理

詳細は、Git リポジトリー、Helm リポジトリー、およびオブジェクトストレージリポジトリーを使用してアプリケーションを作成する方法について参照してください。

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

法律上の通知

概要

詳細は、Git リポジトリー、Helm リポジトリー、およびオブジェクトストレージリポジトリーを使用してアプリケーションを作成する方法について参照してください。

目次

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。

以下のトピックを参照してください。

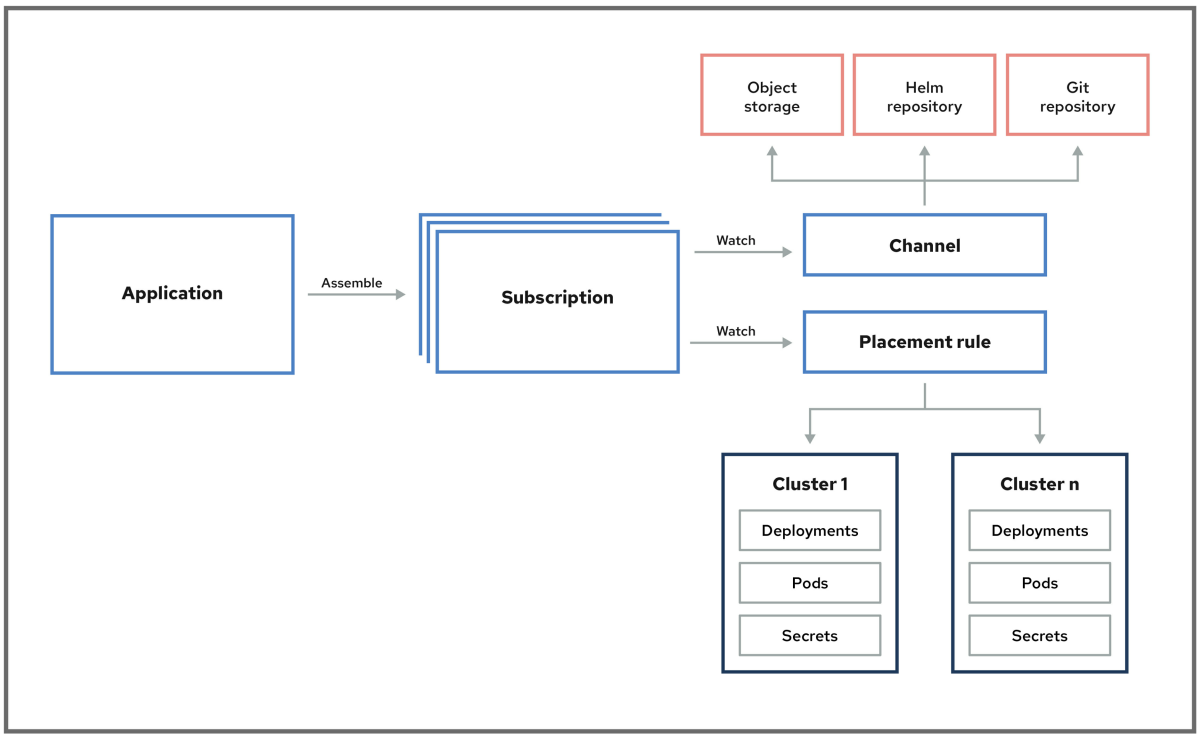
- アプリケーションモデルおよび定義
- アプリケーションコンソール
- アプリケーションリソースの管理
- Git リポジトリーを使用したアプリケーションの管理
- Helm リポジトリーを使用したアプリケーションの管理
- Object Storage リポジトリーを使用したアプリケーションの管理
- アプリケーションの詳細設定
- Git リソースのサブスクライブ
- セキュアな Git 接続用のアプリケーションチャネルおよびサブスクリプションの設定
- Ansible Tower タスクの設定
- Argo CD のマネージドクラスターの設定
- デプロイメントのスケジュール
- パッケージの上書きの設定
- チャネルの例
- サブスクリプションの例
- 配置ルールの例
- 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ー (**チャネル** 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、以下の画像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。

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構成する 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。

1.1.2. チャネル

チャネル (**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ーを定義します。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ー、ハブクラスター上にあるリソーステンプレートです。

認可が必要なチャネルから Kurbernetes 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 リポジトリー) がある場合には、これらのチャ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 に各チャネルを作成することを推奨します。ただし、Git チャネルは、Git、Helm、オブジェクトストレージなどの別のチャネルタイプで namespace を共有できます。

チャネル内の deployable には、そのチャ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す。

1.1.2.1. サポート対象の Git リポジトリーサーバー

- GitHub
- GitLab
- Bitbucket
- Gogs

1.1.3. 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 リポジトリー、Helm リリースリポジトリー、またはオブジェクトストレージリポジトリーなどのソースリポジトリー (チャネル) にサブスクライブできます。

注記: リソースがハブクラスターに影響を及ぼす可能性があるため、ハブクラスターの自己管理は推奨されません。

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。次に、トポロジーで **local-cluster** 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、チャネルを監視できます。

1.1.4. 配置ルール

配置ルール (**placementrule.apps.open-cluster-management.io**) は、リソーステンプレート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。配置ポリシーは、ガバナンスポリシーおよびリスクポリシーにも使用されます。

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 リポジトリーを使用したアプリケーションの管理
- Helm リポジトリーを使用したアプリケーションの管理
- Object Storage リポジトリー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ライブ
- Ansible Tower タスクの設定

- チャネルの例
- サブスクリプションの例
- 配置ルールの例
- 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。

以下のアプリケーションコンソール機能を参照してください。

重要: 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ー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ライブを行います。デフォルトでは、ハブクラスターは自己管理ができ、**local cluster** の名前を指定します。このローカルクラスターに、アプリケーションリソースをデプロイできますが、ローカルクラスターへのアプリケーションのデプロイはベストプラクティスではありません。
- **詳細設定** を使用して、チャネル、サブスクリプション、および配置ルールを表示または編集します。
- リソースリポジトリー、サブスクリプション、配置ルール、Ansible Tower タスクを使用した (git リポジトリー用の) デプロイメント前後のフック (オプション) などのデプロイされたリソースを含む、アプリケーションリソースに対応するトポロジービューにアクセスする。
- デプロイメント、更新、およびサブスクリプションなど、アプリケーションのコンテキストで個別のステータスを表示する。

コンソールには、アプリケーション管理機能をそれぞれ提供する各種ツールが含まれます。このような機能を使用すると、アプリケーションリソースの作成、検索、更新、デプロイを簡単に行えます。

- アプリケーションの概要
- リソーストポロジー
- 検索
- 詳細設定

1.2.1. アプリケーションの概要

メインの **Overview** タブで、以下を参照してください。

- 全アプリケーションを表示するテーブル
- 一覧表示されたアプリケーションをフィルタリングする **Search** ボックス

- アプリケーション名と namespace
- サブスクリプションを使用してリソースをデプロイするリモートおよびローカルクラスターの数
- アプリケーションがデプロイしたリソースの定義が格納されているリポジトリーへのリンク
- 期間の制約の表示 (作成されている場合)
- アプリケーションの作成日
- **Delete application** などの他のアクションアクションは、ユーザーに実行権限がある場合に利用できます。

1.2.1.1. 単一アプリケーションの概要

テーブルのアプリケーション名をクリックし、1 つのアプリケーションの詳細を表示します。以下を参照してください。

- リソースのステータスなどのクラスターの情報
- サブスクリプションの情報
- リソーストポロジー

Editor タブをクリックし、アプリケーションと関連リソースを編集します。

1.2.2. リソーストポロジー

トポロジーでは、ターゲットクラスターのアプリケーションでデプロイしたリソースなど、選択したアプリケーションを視覚的に表示できます。

- トポロジービューからコンポーネントを選択し、詳細情報を表示できます。
- リソースノードをクリックしてプロパティービューを開き、このアプリケーションがデプロイしたリソースのデプロイメント情報を表示します。
- プロパティーダイアログで、クラスターノードからクラスターの CPU およびメモリーを表示します。
  注記: クラスターの CPU およびメモリーの割合では、現在使用中の % が表示されます。この値は切り捨てられるため、非常に小さい値は **0** と表示されます。

  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」を参照して適切な **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。
- 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 Tower デプロイメントを表示します。
  リソースノードの名前をクリックするか、**Actions** > **View application** をクリックして、Ansible Tower ジョブの URL やテンプレート名など、Ansible タスクデプロイメントの詳細を表示します。また、Ansible Tower のデプロイメントに失敗すると、エラーが表示される可能性があります。
- **Launch resource in Search** をクリックし、関連リソースを検索します。

1.2.3. 検索

コンソール **Search** ページでは、各リソースの component **kind** によるアプリケーションリソースの検索をサポートします。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**
チャネル	**Channel**
Secret	**Secret**
配置ルール	**PlacementRule**
アプリケーション	**Application**

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。

検索結果から、名前、namespace、クラスター、ラベル、作成日など、各リソースの識別情報を確認できます。

アクセス権がある場合には、検索結果で **Actions** をクリックして選択し、そのリソースを削除することも可能です。

検索結果でリソース名をクリックし、YAML エディターを開き、変更を加えます。変更は保存すると、すぐにリソースに適用されます。

検索の使用方法は「コンソールでの検索」を参照してください。

1.2.4. 詳細設定

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、配置ルール、チャネルをフィルタリングできます。アクセス権がある場合には、編集、検索、削除などの **Actions** も複数、クリックできます。

YAML を表示または編集するリソースを選択します。

1.3. アプリケーションリソースの管理

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

重要: Git チャネルは、他のすべてのチャネルタイプ (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 リポジトリーを使用したアプリケーションの管理
- Helm リポジトリーを使用したアプリケーションの管理
- Object Storage リポジトリーを使用したアプリケーションの管理

1.3.1. Git リポジトリー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ーに配置されます。以下の手順で、Git リポジトリーからリソースをデプロイする方法を説明します。詳細は、「 アプリケーションモデルおよび定義 」を参照してください。

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 **Manage applications** をクリックします。
2. **Create application** をクリックします。
   以下の手順では、**YAML: On** を選択し、アプリケーションの作成時にコンソールで YAML を表示します。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. 以下の値を正しいフィールドに入力します。
   - Name: アプリケーションの有効な Kubernetes 名を入力します。
   - namespace: 一覧から namespace を選択します。正しいアクセスロールが割り当てられている場合は、有効な Kubernetes 名を使用して namespace を作成することもできます。
4. 使用できるリポジトリーの一覧から **Git** を選択します。
5. 必要な URL パスを入力するか、既存のパスを選択します。
   既存の Git リポジトリーパスを選択し、そのリポジトリーがプライベートの場合には、接続情報を指定する必要はありません。接続情報が事前設定されているので、これらの値を確認する必要はありません。

   新しい Git リポジトリーパスを入力し、その Git リポジトリーがプライベートの場合には、オプションで Git 接続情報を入力できます。
6. ブランチやパスなど、オプションでフィールドに値を入力します。
7. 調整オプションに注意してください。**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。
8. 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。
   テクノロジープレビュー: 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後に実行する Ansible Tower ジョブがある場合には、Ansible Tower のシークレットを設定します。Ansible ジョブを定義する Ansible Tower タスクは、このリポジトリーの **prehook** および **posthook** フォルダー内に配置する必要があります。

   シークレットをアプリケーション namespace に作成した場合は、ドロップダウンメニューから Ansible Tower シークレットを選択できます。この例では、接続情報は設定されており、これらの値を表示する必要はありません。

   新しい Ansible Tower シークレット名を入力して新しいシークレットを作成する場合は、Ansible Tower ホストおよびトークンを入力する必要があります。
9. **Select clusters to deploy** で、アプリケーションの配置ルールの情報を更新できます。以下から選択します。
   - ローカルクラスターへのデプロイ

- すべてのオンラインクラスターおよびローカルクラスターへのデプロイ
- 指定されたラベルに一致するクラスターのみへのアプリケーションリソースのデプロイ
- 配置ルールが定義済みの既存の namespace にアプリケーションを作成した場合には、**既存の配置設定を選択する** こともできます。

10. **Settings** から、アプリケーションの動作を指定できます。特定の期間にデプロイメントへの変更をブロックまたは有効にするには、**Deployment window** のオプションおよび **Time window configuration** を選択します。
11. 別のリポジトリーを選択するか、または **Save** をクリックします。
12. **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。

1.3.1.1. GitOps パターン

Git リポジトリーを編成してクラスターを管理するベストプラクティスについて説明します。

1.3.1.1.1. GitOps の例

この例のフォルダーは定義されて名前が付けられ、各フォルダーには、マネージドクラスターで実行されるアプリケーションまたは設定が含まれます。

- root フォルダー **managed-subscriptions**: **common-managed** フォルダーをターゲットとするサブスクリプションが含まれます。
- サブフォルダー **apps**/: **managed-clusters** に対する配置で **common-managed** フォルダーでアプリケーションをサブスクライブするために使用されます。
- サブフォルダー **config**/: **managed-clusters** に対する配置で **common-managed** フォルダーで設定をサブスクライブするために使用されます。
- サブフォルダー **policies**/: **managed-clusters** に対する配置でポリシーを適用するために使用します。
- フォルダー **root-subscription**/: **managed-subscriptions** フォルダーをサブスクライブするハブクラスターの最初のサブスクリプション。

ディレクトリーの例を参照してください。

```
common-managed/
   apps/
     app-name-0/
     app-name-1/
   config/
     config001/
     config002/

managed-subscriptions
   apps/
   config/
   policies/

root-subscription/
```

1.3.1.1.2. GitOps フロー

ディレクトリー構造は、**root-subscription** > **managed-subscriptions** > **common-managed** のサブスクリプションフロー用に作成されます。

1. **root-subscription/** の単一サブスクリプションは、CLI ターミナルからハブクラスターに適用されます。
2. サブスクリプションとポリシーは、**managed-subscription** フォルダーからハブクラスターにダウンロードされ、適用されます。
   - **managed-subscription** フォルダーのサブスクリプションおよびポリシーは、配置に基づいてマネージドクラスターで機能します。
   - 配置は、各サブスクリプションまたはポリシーが影響を及ぼす **managed-clusters** を決定します。
   - サブスクリプションまたはポリシーは、配置に一致するクラスター上のものを定義します。
3. サブスクリプションは、**common-managed** フォルダーのコンテンツを、配置ルールに一致する **managed-clusters** に適用します。また、配置ルールに一致するすべての **managed-clusters** に対して、一般的なアプリケーションおよび設定も適用されます。

1.3.1.1.3. その他の例

- **root-subscription/** の例については、application-subscribe-all を参照してください。
- 同じリポジトリー内の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 リポジトリー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。

1.3.2. Helm リポジトリー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ーに配置されます。以下の手順で、Helm リポジトリーからリソースをデプロイする方法を説明します。詳細は、「 アプリケーションモデルおよび定義 」を参照してください。

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 **Manage applications** をクリックします。
2. **Create application** をクリックします。
   以下の手順では、**YAML: On** を選択し、アプリケーションの作成時にコンソールで YAML を表示します。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. 以下の値を正しいフィールドに入力します。
   - Name: アプリケーションの有効な Kubernetes 名を入力します。

- namespace: 一覧から namespace を選択します。正しいアクセスロールが割り当てられている場合は、有効な Kubernetes 名を使用して namespace を作成することもできます。

4. 使用できるリポジトリーの一覧から **Helm** を選択します。

5. 必要な URL パスを入力するか、既存のパスを選択してから、パッケージバージョンを入力します。
   既存の Helm リポジトリーパスを選択し、そのリポジトリーがプライベートの場合には、接続情報を指定する必要はありません。接続情報が事前設定されているので、これらの値を確認する必要はありません。

   新しい Helm リポジトリーパスを入力し、その Helm リポジトリーがプライベートの場合には、オプションで Helm 接続情報を入力できます。

6. **Select clusters to deploy** で、アプリケーションの配置ルールの情報を更新できます。以下から選択します。

   - ローカルクラスターへのデプロイ
   - すべてのオンラインクラスターおよびローカルクラスターへのデプロイ
   - 指定されたラベルに一致するクラスターのみへのアプリケーションリソースのデプロイ
   - 配置ルールが定義済みの既存の namespace にアプリケーションを作成した場合には、**既存の配置設定を選択する** こともできます。

7. **Settings** から、アプリケーションの動作を指定できます。特定の期間にデプロイメントへの変更をブロックまたは有効にするには、**Deployment window** のオプションおよび **Time window configuration** を選択します。

8. 別のリポジトリーを選択するか、または **Save** をクリックします。

9. **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。

1.3.2.1. YAML 例

以下のチャネル定義例では Helm リポジトリーをチャ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
```

以下のチャネル定義は、Helm リポジトリーチャ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」を使用します。

1.3.3. Object Storage リポジトリー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ーに配置されます。以下の手順で、オブジェクトストレージリポジトリーからリソースをデプロイする方法を説明します。詳細は、「 アプリケーションモデルおよび定義 」を参照してください。

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。

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ーに配置されます。以下の手順で、Git リポジトリーからリソースをデプロイする方法を説明します。

1. コンソールのナビゲーションメニューから **Manage applications** をクリックします。
2. **Create application を** クリックします。
   以下の手順では、**YAML: On** を選択し、アプリケーションの作成時にコンソールで YAML を表示します。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. 以下の値を正しいフィールドに入力します。
   - Name: アプリケーションの有効な Kubernetes 名を入力します。
   - namespace: 一覧から namespace を選択します。正しいアクセスロールが割り当てられている場合は、有効な Kubernetes 名を使用して namespace を作成することもできます。
4. 使用できるリポジトリーの一覧から **オブジェクトストレージ** を選択します。
5. 必要な URL パスを入力するか、既存のパスを選択します。
   既存のオブジェクトストレージのリポジトリーパスを選択し、そのリポジトリーがプライベートの場合には、接続情報を指定する必要はありません。接続情報が事前設定されているので、これらの値を確認する必要はありません。

   新しいオブジェクトストレージリポジトリーパスを入力し、そのオブジェクトストレージリポジトリーがプライベートの場合には、オプションでオブジェクトストレージ接続情報を入力できます。
6. オプションのフィールドに値を入力します。
7. 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。

8. **Select clusters to deploy** で、アプリケーションの配置ルールの情報を更新できます。以下から選択します。
   - ローカルクラスターへのデプロイ
   - すべてのオンラインクラスターおよびローカルクラスターへのデプロイ
   - 指定されたラベルに一致するクラスターのみへのアプリケーションリソースのデプロイ
   - 配置ルールが定義済みの既存の namespace にアプリケーションを作成した場合には、**既存の配置設定を選択する** こともできます。
9. **Settings** から、アプリケーションの動作を指定できます。特定の期間にデプロイメントへの変更をブロックまたは有効にするには、**Deployment window** のオプションおよび **Time window configuration** を選択します。
10. 別のリポジトリーを選択するか、または **Save** をクリックします。
11. **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。

1.3.3.1. YAML 例

以下のチャネル定義例では、オブジェクトストレージをチャ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://9.28.236.243:31311/dev] # URL is appended with the valid bucket name, which matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

注記: REST API を確認するには、「API」を使用します。

1.4. 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構成されています。チャネル、サブスクリプション、および配置ルールリソースを使用すると、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
- セキュアな Git 接続用のアプリケーションチャネルおよびサブスクリプションの設定
- Ansible Tower タスクの設定
- Argo CD のマネージドクラスターの設定
- パッケージの上書きの設定
- チャネルの例
- サブスクリプションの例
- 配置ルールの例
- アプリケーションの例

1.4.1. 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。このトピックで説明するように、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には、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

1.4.1.1. ユーザーとグループのサブスクリプション管理権限の付与

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。

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。
   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. 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
```

1.4.1.2. 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。

1.4.1.3. アプリケーション namespace の例

この例では、サブスクリプション管理者としてログインします。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 リポジトリー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には、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。

1.4.1.4. 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)

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 リポジトリーから、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
```

1.4.1.4.1. 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 リポジトリー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には、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。

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場合は、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。このような場合には、この方法を使用しないでください。

1.4.1.4.2. 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には、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
```

1.4.1.4.3. 調整オプション

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することもできます。

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 リポジトリーのリソース 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。

1.4.1.4.3.1. 調整頻度

チャネル設定で、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。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**: ソースの Git リポジトリー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ーの最新コミット ID 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。リポジトリー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 リポジトリー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 リポジトリー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。

以下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、サブスクリプション CR 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。

以下のサンプル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。

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

Git チャ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 リポジトリーに接続します。以下のアプリケーションチャネル設定は、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ーへの接続
- 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

1.4.2.1. 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ーへの接続

チャ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ーに接続するには、以下の手順を参照してください。

1. チャネルと同じ 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 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
```

1.4.2.2. 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。

チャネルの仕様で **insecureSkipVerify: true** を指定します。それ以外の場合には、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
ind: Channel
metadata:
labels:
  name: sample-channel
  namespace: sample
spec:
  type: GitHub
  pathname: <Git HTTPS URL>
  insecureSkipVerify: true
```

1.4.2.3. セキュアな HTTPS 接続でのカスタム CA 証明書の使用

こ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 サーバーに安全に接続できます。

1. Git サーバーの root および中間 CA 証明書を PEM 形式で含む ConfigMap を作成します。ConfigMap はチャネル CR と同じ namespace にある必要があります。フィールド名は **caCerts** で、| を使用する必要があります。以下の例で、**caCerts** には root や中間 CA などの複数の証明書を含めることができる点に留意してください。

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

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

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
    LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
    hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH


VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBgNVBA
```

MM
PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWIxMy5kZXYwNi5yZWQtY2hl
c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcmVkaGF0LmNvbTCC
AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0Ic3
U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
6OS3gj738F635inz1KdyhKtlWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNIdRHcpQP
upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWIwNsoL5xt0Lw4mSyhlEip/t8lU
xn2y8qhm7MiIUpXuwWhSYgCrEVqmTcB70Pc2YRZdSFolMN9Et70MjQN0TXjoktH8
PyASJIKIRd+48yROIbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
+SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcnQVFmvaoM
gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
diNXs4Zn3QSStC2x2hKnK+Q1rwCSEg/lBawgxGUslTboFH77a+Kwu4Oug9ibtm5z
ISs/JY4Kiy4C2XJOltOR2XZYkdKaX4x3ctbrGaD8Bj+QHiSAxaaSXIX+VbzkHF2N
aD5ijFUopjQEKFrYh3O93DB/URIQ+wHVa6+Kvu3uqE0cg6pQsLpbFVQ/l8xHvt9L
kYy6z6V/nj9ZYKQbq/kPAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAKZuc+lewYAv
jaaSeRDRoToTb/yN0Xsi69UfK0aBdvhCa7/0rPHcv8hmUBH3YgkZ+CSA5ygajtL4
g2E8CwIO9ZjZ6l+pHCuqmNYoX1wdjaaDXlpwk8hGTSgy1LsOoYrC5ZysCi9Jilu9
PQVGs/vehQRqLV9uZBigG6oZqdUqEimaLHrOcEAHB5RVcnFurz0qNbT+UySjsD63
9yJdCeQbeKAR9SC4hG13EbM/RZh0lgFupkmGts7QYULzT+oA0cCJpPLQl6m6qGyE
kh9aBB7FLykK1TeXVuANlNU4EMyJ/e+uhNkS9ubNJ3vuRuo+ECHsha058yi16JC9
NkZqP+df4Hp85sd+xhrgYieq7QGX2KOXAjqAWo9htoBhOyW3mm783A7WcOiBMQv0
2UGZxMsRjlP6UqB08LsV5ZBAefElR344sokJR1de/Sx2J9J/am7yOoqbtKpQotIA
XSUkATuuQw4ctyZLDkUpzrDzgd2Bt+aawF6sD2YqycaGFwv2YD9t1YlD6F4Wh8Mc
20Qu5EGrkQTCWZ9pOHNSa7YQdmJzwbxJC4hqBpBRAJFl2fAlqFtyum6/8ZN9nZ9K
FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6ladliUOST3w5t92cB
M26t73UCExXMXTCQvnp0ki84PeR1kRk4
-----END CERTIFICATE-----

Git server intermediate CA 1

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

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH

VG9yb250bzEPMA0GA1UECgwGUmVkSGF0MQwwCgYDVQQLDANBQ00xRTBDBgNVBA
MM
PGdvZ3Mtc3ZjLWRlZmF1bHQuYXBwcy5yanVuZy1odWIxMy5kZXYwNi5yZWQtY2hl
c3RlcmZpZWxkLmNvbTEfMB0GCSqGSIb3DQEJARYQcm9rZWpAcmVkaGF0LmNvbTCC
AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAM3nPK4mOQzaDAo6S3ZJ0Ic3
U9p/NLodnoTIC+cn0q8qNCAjf13zbGB3bfN9Zxl8Q5fv+wYwHrUOReCp6U/InyQy
6OS3gj738F635inz1KdyhKtlWW2p9Ye9DUtx1IlfHkDVdXtynjHQbsFNIdRHcpQP
upM5pwPC3BZXqvXChhlfAy2m4yu7vy0hO/oTzWIwNsoL5xt0Lw4mSyhlEip/t8lU
xn2y8qhm7MiIUpXuwWhSYgCrEVqmTcB70Pc2YRZdSFolMN9Et70MjQN0TXjoktH8
PyASJIKIRd+48yROIbUn8rj4aYYBsJuoSCjJNwujZPbqseqUr42+v+Qp2bBj1Sjw
+SEZfHTvSv8AqX0T6eo6njr578+DgYlwsS1A1zcAdzp8qmDGqvJDzwcnQVFmvaoM
gGHCdJihfy3vDhxuZRDse0V4Pz6tl6iklM+tHrJL/bdL0NdfJXNCqn2nKrM51fpw
diNXs4Zn3QSStC2x2hKnK+Q1rwCSEg/lBawgxGUslTboFH77a+Kwu4Oug9ibtm5z

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

    # Git server intermediate CA 2

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

Q0ExCzAJBgNVBAgMAk9OMRAwDgYDVQQHDAdUb3JvbnRvMQ8wDQYDVQQKDAZSZW
RI

YXQxDDAKBgNVBAsMA0FDTTFFMEMGA1UEAww8Z29ncy1zdmMtZGVmYXVsdC5hcHBz
    LnJqdW5nLWh1YjEzLmRldjA2LnJlZC1jaGVzdGVyZmllbGQuY29tMR8wHQYJKoZI
    hvcNAQkBFhByb2tlakByZWRoYXQuY29tMB4XDTIwMTIwMzE4NTMxMloXDTIzMDky

MzE4NTMxMlowgbMxCzAJBgNVBAYTAkNBMQswCQYDVQQIDAJPTjEQMA4GA1UEBwwH


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

```
    FSEKdlu+xeb6Y6xYt0mJJWF6mCRi4i7IL74EU/VNXwFmfP6ladliUOST3w5t92cB
    M26t73UCExXMXTCQvnp0ki84PeR1kRk4
    -----END CERTIFICATE-----
```

2. この ConfigMap でチャネルを設定します。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
```

1.4.2.4. Git サーバーへの SSH 接続の設定

1. **data** の **sshKey** フィールドに SSH 秘密キーを含むためのシークレットを作成します。キーがパスフレーズ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、チャネル CRと同じ namespace にある必要があります。**kubectl** 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 **パスフレーズ** 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
RqRUFBQUFBQ21GbGN6STFOaTFqZEhJQUFBQUdZbU55ZVhCMEFBQUFHQUFBQUJD
K3YySHhWSIwCm8zejh1endzV3NWODMvSFVkOEtGeVBmWk5OeE5TQUgcFA3Yk1yR2tlRF
FPd3J6MGIKOUlRM0tKVXQzWEE0Zmd6NVlrVFVhcTJsZWxxVk1HcXI2WHF2UVJ5Mkc0NkRI
RVlYUGpabVZMcGVuaGtRYU5HYmpaMmZOdQpWUGpiOVhZRmd4bTNnYUpJU3BNeTFL
WjQ5MzJvOFByaDZEdzRYVUF1a28wZGdBaDdndVpPaE53b0pVYnNmYlZRc0xMS1RrCnQw
blZ1anRvd2NEVGx4TlpIUjcwbGVUSHdGQTYwekM0elpMNkRPc3RMYjV2LzZhMjFHRlMwVm
VXQ3YvMlpMOE1sbjVUZWwKSytoUWtxRnJBL3BUc1ozVXNjSG1GUi9PV25FPQotLS0tLUVO
RCBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0K
  passphrase: cGFzc3cwcmQK
type: Opaque
```

2. シークレットでチャネルを設定します。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:
  configMapRef:
```

```
    name: git-known-hosts
  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** 一覧を構築します。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
```

1.4.2.5. 証明書および SSH キーの更新

Git チャネル接続設定で 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、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。

1.4.3. Ansible Tower タスクの設定 (テクノロジープレビュー)

Red Hat Advanced Cluster Management は Ansible Tower 自動化と統合されるので、Git サブスクリプションのアプリケーション管理の prehook および posthook AnsibleJob インスタンスを作成できます。Ansible Tower 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 リポジトリーリソースの root パスには、アプリのデプロイ、更新、クラスターからの削除の一環として実行される Ansible Tower ジョブの **prehook** と **posthook** ディレクトリーが含まれます。

必要なアクセス権限: クラスターの管理者

1.4.3.1. 前提条件

- OpenShift Container Platform 4.5 以降
- Ansible Tower バージョン 3.7.3 以降がインストールされていること。Ansible Tower の最新のサポートバージョンをインストールすることがベストプラクティスです。詳細は、Red Hat AnsibleTower 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 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しておくこと。AnsibleJob を使用した Ansible Tower ジョブの実行時に最善の結果を得るには、実行時に Ansible Tower ジョブテンプレートが冪等でなければなりません。

テンプレートの **PROMPT ON LAUNCH** に INVENTORY と EXTRA VARIABLES の両方の有無を確認します。詳細は、「ジョブテンプレート」を参照してください。

1.4.3.2. Ansible Automation Platform Resource Operator をインストールします。

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。

1.4.3.3. Ansible Tower の URL およびトークンの取得

Ansible Tower の URL は、Tower へのログインに使用される URL と同じです。これは、Ansible prehook および posthook を使用してアプリケーションを設定する場合に、**アプリケーションコンソール** または Tower アクセスシークレットで必要になります。

URL は **https://ansible-tower-web-svc-tower.apps.my-openshift-cluster.com** のようになります。

1.4.3.4. トークンの取得

1. Ansible Tower 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 **Users** をクリックします。
3. 正しいユーザーを検索します。
4. ユーザーの **編集アイコン** をクリックします。
5. ユーザーセクションの **TOKENS** をクリックします。
6. **+** ボタンをクリックしてトークンを追加します。
7. **APPLICATION** フィールドを空白のままにします。
8. **DESCRIPTION** フィールドに、このトークンに使用する目的を指定します。
9. **SCOPE** ドロップダウンメニューで **Write** を選択します。
10. **SAVE** をクリックし、表示される **TOKEN** を記録します。

1.4.3.5. Ansible の統合

Ansible Tower ジョブは、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には、Ansible Tower の Ansible ジョブ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。また、アプリケーションは、Git サブスクリプションで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 リポジトリーリソースのルートパスにあり、それぞれ Ansible ジョブの prehook および posthook 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、フック前後の 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、インスタンス実行前および実行後の AnsibleJob インスタンスを作成するタイミングを決定します。

1.4.3.6. Ansible Operator のコンポーネント

サブスクリプション CR を作成すると、Git ブランチおよび Git パスは Git リポジトリー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。

1.4.3.6.1. Prehook

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、prehook Ansible オブジェクトとして prehook フォルダー内の **Kind:AnsibleJob** CR すべてを検索してから、新たに prehook AnsibleJob インスタンスを生成します。新しいインスタンス名には、prethook AnsibleJob のオブジェクト名、その後に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: **database-sync-1-2913063** を参照してください。

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、調整要求を 1 分間のループで再度キューに追加します。その時に、prehook AnsibleJob **status.ansibleJobResult** を確認します。prehook **status.ansibleJobResult.status** が **successful** となると、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のデプロイを続行します。

1.4.3.6.2. posthook

アプリケーションの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のステータスが Subscribed か、ステータスが Subscribed の全ターゲットクラスターに伝播された場合に、アプリケーションのサブスクリプションコントロは posthook AnsibleJob オブジェクトとして、posthook フォルダーの全 **AnsibleJob Kind** CR を検索します。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、その後に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。

1.4.3.6.3. Ansible 配置ルール

有効な prehook AnsibleJob では、サブスクリプションは配置ルールの決定事項に関係なく prehook AnsibleJob を起動します。たとえば、配置ルールサブスクリプションの伝播に失敗した prehook AnsibleJob を起動できます。配置ルールの意思決定が変更されると、新しい prehook および posthook AnsibleJob インスタンスが作成されます。

1.4.3.7. Ansible の設定

Ansible Tower 設定は、以下のタスクで指定できます。

1.4.3.7.1. Ansible シークレット

Ansible Tower シークレット CR は、同じサブスクリプション namespace 内に作成する必要があります。Ansible Tower シークレットは、同じサブスクリプション namespace に限定されます。

コンソールからシークレットを作成するには、**Ansible Tower のシークレット名** セクションに入力します。ターミナルを使用してシークレットを作成するには、以下の **yaml** を編集して適用します。

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
```

以下の YAML 例を参照してください。

注記: namespace は、サブスクリプションの namspace と同じにします。 **stringData:token** および **host** は Ansible Tower から取得します。

```
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
```

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を使用して prehook および posthook AnsibleJobs を作成する時に、サブスクリプションの **spec.hooksecretref** からのシークレットが利用できる場合には、AnsibleJob CR **spec.tower_auth_secret** に送信され、AnsibleJob が Ansible Tower にアクセスできるようになります。

1.4.3.8. シークレット調整の設定

prehook と posthook AnsibleJobs がある main-sub おサブスクリプションの場合には、メイン/サブのサブスクリプションは prehook と posthook AnsibleJobs すべてまたはメインサブスクリプションが Git リポジトリーで更新されてから調整する必要があります。

Prehook AnsibleJobs と main サブスクリプションは継続的に調整し、新しい pre-AnsibleJob インスタンスを起動し直します。

1. pre-AnsibleJob の実行後に、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の使用が変更されると、サブスクリプションは再デプロイされます。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す。ターゲットクラスターで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. メインサブスクリプションが完了したら、新しい posthook AnsibleJob インスタンスが再起動されます。
5. ステータスが Done の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 CR の作成時に、Kubernetes ジョブの CR が作成され、ターゲットの Ansible Tower に通信して Ansible Tower ジョブを起動します。ジョブが完了すると、ジョブの最終ステータスが AnsibleJob **status.ansibleJobResult** に戻ります。

注記:

AnsibleJob status.conditions は、Kubernetes ジョフの結果作成の保存用に Ansible Job operator により予約されています。status.conditions には、実際の Ansible Tower ジョブのステータスは反映されません。

サブスクリプションコントローラーは、Ansible Tower ジョブのステータスを **AnsibleJob.status.conditions** ではなく、**AnsibleJob.status.ansibleJobResult** で確認します。

前述の prehook および posthook AnsibleJob ワークフローで説明されているように、Git リポジトリーでメインサブスクリプションを更新すると、新しい prehook および posthook AnsibleJob インスタンスが作成されます。これにより、1 つのメインサブスクリプションに複数の AnsibleJob インスタンスをリンクできます。

subscription.status.ansibleJobs で 4 つのフィールドが定義されます。

- lastPrehookJobs: 最新の prehook AnsibleJobs
- prehookJobsHistory: prehook AnsibleJobs の全履歴
- lastPosthookJobs: 最新の posthook AnsibleJobs
- PosthookJobsHistory: posthook AnsibleJobs 全履歴

1.4.3.9. Ansible のサンプル YAML

以下の Git prehook および posthook フォルダーの AnsibleJob **.yaml** ファイルの例を参照してください。

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

1.4.4. Argo CD のマネージドクラスターの設定

サポートされるマネージドクラスターのタイプを手動で同期して、Argo CD クラスターコレクションを有効にし、アプリケーションを Argo CD から ACM マネージドクラスターにデプロイできるようにします。

1.4.4.1. 前提条件

- Red Hat Advanced Cluster Management for Kubernetes に、Argo CD をインストール する必要があります。
- 1 つ以上のマネージドクラスターが必要です。

1.4.4.2. Argo CD の設定

1 つ以上のマネージドクラスターの ArgoCD クラスターコレクションを有効または無効にすることができます。**cluster1** マネージドクラスターには、以下の **KlusterletAddonConfig** のサンプルリソースを参照してください。**spec.applicationManager.argocdCluster**の設定は、true または false に設定されています。

```
apiVersion: agent.open-cluster-management.io/v1
kind: KlusterletAddonConfig
metadata:
  name: cluster1
  namespace: cluster1
spec:
  applicationManager:
    argocdCluster: <true/false>
```

Argo CD クラスターコレクションを有効にすると、マネージドクラスターのシークレットは、ハブのマネージドクラスター namespace に自動作成されます。クラスターシークレットが **cluster1** namespace にある以下の例を参照してください。

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-cluster-secret
  namespace: cluster1
  labels:
    apps.open-cluster-management.io/secret-type: acm-cluster
type: Opaque
stringData:
  name: cluster1
  server: https://<url-name-here>
  config: |
    {
      "bearerToken": "<the bear token>",
      "tlsClientConfig": {
        "insecure": true
      }
    }
```

マネージドクラスターのシークレットが Argo CD namespace に同期される場合、クラスターシークレットはラベルが Argo CD **secret-type** に固有の以下の例のようになり、namespace は **argocd** に変更されます。

```
apiVersion: v1
kind: Secret
metadata:
  labels:
    argocd.argoproj.io/secret-type: cluster
    apps.open-cluster-management.io/acm-cluster: "true"
  name: cluster1-cluster-secret
```

```
  namespace: argocd
type: Opaque
stringData:
  name: cluster1
  server: https://<url-name-here>
  config: |
    {
      "bearerToken": "<bearer token>",
      "tlsClientConfig": {
        "insecure": true
      }
    }
```

1.4.5. 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には、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が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拘らず、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には、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には、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定します。
- 期間タイプが **active** の場合には、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のみ、デプロイメントを行う場合に、この設定を使用できます。
- 期間タイプが **block** の場合は、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、セキュリティー関連の更新を午前 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。

1.4.6. 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には、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
```

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 リリースの場合には、**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には、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には、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には、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
```

1.4.7. チャネルの例

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。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 (CICD) を提供します。

Kubernetes 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。

```
kubectl 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。

```
kubectl get Application
```

注記: Kubernetes namespace (Namespace) チャネルは本リリースでは使用できません。

1.4.7.1. チャネル YAML の構造

以下の YAML 構造は、チャ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する必要がある場合があります。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: Channel
metadata:
  name:
  namespace: # Each channel needs a unique namespace, except Git channel.
spec:
  sourceNamespaces:
  type:
```

```
  pathname:
  secretRef:
    name:
  gates:
    annotations:
  labels:
```

1.4.7.2. チャネル YAML 表

フィールド	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、リソースがチャネルであることを指定します。
metadata.name	必須。チャネルの名前。
metadata.namespace	必須。チャネルの namespace。各チャネルには Git チャネルを除き、一意の namespace が必要です。
spec.sourceNamespaces	任意。チャネルコントローラーが取得してチャ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** チャネルには必須。**HelmRepo** チャネルの場合は、値を Helm リポジトリーの URL に設定します。**ObjectBucket** チャネルの場合は、値をオブジェクトストレージの URL に設定します。**Git** チャネルの場合は、値を Git リポジトリーの HTTPS URL に設定します。
spec.secretRef.name	任意。リポジトリー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** タイプのチャネルでのみ認証に使用できます。
spec.gates	任意。チャネル内での 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には、チャネルの namespace またはソースに追加された deployable がそのチャネルにプロモートされます。**gates** は、**ObjectBucket** チャネルタイプだけに適用され **HelmRepo** や **Git** チャネルタイプに適用されません。

フィールド	説明
spec.gates.annotations	任意。チャネルのアノテーション。チャネル内では deployable に同じアノテーションを追加する必要があります。
metadata.labels	任意。チャネルのラベル。

チャネルの定義構造は、以下の YAML コンテンツの例のようになります。

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

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

以下のチャネル定義例では、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://9.28.236.243:31311/dev] # URL is appended with the valid bucket name, which matches the channel name.
 secretRef:
   name: miniosecret
 gates:
   annotations:
     dev-ready: true
```

1.4.7.4. Helm リポジトリー (HelmRepo) チャネル

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

非推奨に関する注記: 2.2 では、チャネルの **ConfigMap** 参照に **insecureSkipVerify: "true"** を指定して Helm リポジトリーの SSL 証明書を省略することが非推奨になりました。以下に例を示します。

```
apiVersion: v1
data:
```

```
  insecureSkipVerify: "true" # deprecated
kind: ConfigMap
metadata:
  name: insecure-skip-verify
  namespace: hub-repo
```

以下のサンプルで、チャ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。

```
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 リポジトリーチャネルの別の例を示しています。

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

1.4.7.5. Git (Git) リポジトリーチャネル

以下のチャネル定義例は、Git リポジトリー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 リポジトリー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ーを使用する場合は、**secretRef** 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/stolostron/gitops-clusters.git
```

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

1.4.8. シークレットの例

シークレット (**Secret**) は、パスワード、OAuth トークンや SSH キーなどの機密情報や認可の保存に使用可能な Kubernetes リソースです。この情報をシークレットとして保存すると、データセキュリティーの向上にこの情報を必要とするアプリケーションコンポーネントと、この情報を切り離すことができます。

Kubernetes 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。

```
kubectl 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。

```
kubectl get Application
```

シークレットの定義構造は、以下の YAML コンテンツのようになります。

1.4.8.1. シークレット YAML の構造

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

1.4.9. サブスクリプションの例

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。チャネルと同様に、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 (CICD) を提供します。

Kubernetes 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 apiserver に適用します。**filename** は、使用するファイル名に置き換えます。

```
kubectl 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。

```
kubectl get Application
```

1.4.9.1. 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する必要がある場合があります。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: PlacementRule
  overrides:
    clusterName:
    clusterOverrides:
      path:
      value:
```

-

1.4.9.2. サブスクリプションの YAML 表

フィールド	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。
spec.channel	任意。サブスクリプションのチャネルを定義する namespace 名 ("Namespace/Name")。**channel**、**source** または **sourceNamespace** フィールドを定義します。通常、**source** 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には、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 チャネルにのみ使用してください。**channel**、**source** または **sourceNamespace** フィールドを定義します。通常、**source** 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 リポジトリーのパス名 ("URL")。このフィールドは、Helm リポジトリーチャネルにだけ使用します。**channel**、**source** または **sourceNamespace** フィールドを定義します。通常、**source** 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ネルを参照します。

フィールド	説明
spec.name	**HelmRepo** タイプのチャネルには必須ですが、**ObjectBucket** タイプのチャネルには任意です。チャ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。
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には、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。ただし、上書き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 を配置する必要の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	説明
spec.placement.local	任意。ただし、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。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。サブスクリプションと、指定のチャネルを同期させるには、値を **true** に設定します。指定のチャ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。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。
spec.placement.clusters	任意。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には、**local cluster** も使用できます。
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。
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には、**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には、**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。

フィールド	説明
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 **PlacementRule** 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。
spec.timeWindow.type	任意。ただし、期間の設定には必須です。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。ただし、期間の設定には必須です。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。ただし、期間の設定には必須です。期間の作成時に時間の範囲を適用する場合には、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。ただし、期間の設定には必須です。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。ただし、期間の設定には必須です。期間の開始を定義するタイムスタンプ。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。

フィールド	説明
spec.timeWindow.hours.end	任意。ただし、期間の設定には必須です。期間の終了を定義するタイムスタンプ。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、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はできません。実際の時間の範囲は、**subscription-controller** のコンテナー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には、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: PlacementRule
      name: towhichcluster
  overrides:
  - clusterName: "/"
```

```
    clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.4.9.3. サブスクリプションファイルの例

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

1.4.9.3.1. 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: PlacementRule
      name: towhichcluster
  timewindow:
    windowtype: "active" #Enter active or blocked depending on the purpose of the type.
    location: "America/Los_Angeles"
    daysofweek: ["Monday", "Wednesday", "Friday"]
    hours:
      - start: "10:20AM"
        end: "10:30AM"
      - start: "12:40PM"
        end: "1:40PM"
```

1.4.9.3.2. 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、

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
```

1.4.9.3.3. Helm リポジトリー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 リポジトリーで新規バージョンが利用できる場合には、Helm リリースの deployable は **my-development-cluster-1** クラスターに配置されます。

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
```

1.4.9.3.4. Git リポジトリーのサブスクリプションの例

1.4.9.3.4.1. Git リポジトリーの特定ブランチおよびディレクトリーのサブスクライブ

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

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、チャネルに指定されている Git リポジトリーの **sample_app_1/dir1** ディレクトリーにある Helm チャートと Kubernetes リソースすべてを、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、リポジトリーの **branch1** ブランチをサブスクライブしています。

1.4.9.3.4.2. .kubernetesignore ファイルの追加

Git リポジトリーの 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リポジトリーから Kubernetes リソースか、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ー、あるいは両方を指定することができます。

また、**.kubernetesignore** ファイいるを使用して、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可能です。**.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 アノテーション外定義されていない場合には、サブスクリプションは、リポジトリーの root ディレクトリーで **.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場合には、サブスクリプションは **apps.open-cluster-management.io/github-path** ディレクトリーで **.kubernetesignore** ファイルを検索します。サブスクリプションは、他のディレクトリーでは **.kubernetesignore** ファイルの検索は行いません。

1.4.9.3.4.3. Kustomize の適用

サブスクライブする Git のフォルダーに **kustomization.yaml** または **kustomization.yml** ファイルがある場合には、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、新規エントリーを追加するか、既存のエントリーを更新します。既存のエントリーは削除されません。

1.4.9.3.4.4. GitHub Webhook の有効化

デフォルトでは、Git チャネルのサブスクリプションは、チャネルで指定されている GitHub リポジトリーを 1 分毎にクローンし、コミット ID が変更されたら、変更が適用されます。または、リポジトリーのプッシュまたはプルの Webhook イベント通知を Git リポジトリー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 リポジトリーで 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。

1.4.9.3.4.4.1. 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。

1.4.9.3.4.4.2. Webhook シークレット

Webhook シークレットは任意です。チャネル 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 リポジトリーごとに一意のシークレットを使用してください。

1.4.9.3.4.4.3. Git リポジトリーでの 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 リポジトリーで Webhook を設定します。

1.4.9.3.4.4.4. チャネルでの 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には、これについても、チャ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>"
```

1.4.9.3.4.4.5. Webhook 対応のチャネルのサブスクリプション

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。

1.4.10. 配置ルールの例

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。

Kubernetes 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。

```
kubectl 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。

```
kubectl get Application
```

1.4.10.1. 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する必要がある場合があります。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: PlacementRule
  name:
  namespace:
  resourceVersion:
  labels:
    app:
    chart:
    release:
    heritage:
  selfLink:
  uid:
spec:
  clusterSelector:
    matchLabels:
      datacenter:
      environment:
  clusterReplicas:
  clusterConditions:
  ResourceHint:
    type:
    order:
  Policies:
```

1.4.10.2. 配置ルールの YAML 値の表

フィールド	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** n い設定して、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。

フィールド	説明
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には、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。
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。

1.4.10.3. 配置ルールファイルの例

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。

フィールド	説明
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: towhichcluster
  namespace: ns-sub-1
  labels:
    app: nginx-app-details
spec:
  clusterReplicas: 1
  clusterConditions:
    - type: ManagedClusterConditionAvailable
      status: "True"
  clusterSelector:
    matchExpressions:
    - key: environment
      operator: In
      values:
      - dev
```

1.4.11. 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。

Kubernetes 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。

```
kubectl 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。

```
kubectl get Application
```

1.4.11.1. 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する必要がある場合があります。

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
```

1.4.11.2. 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	**Application**	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		

フィールド	値	説明
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。

1.4.11.3. アプリケーションファイルの例

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
```