Red Hat Advanced Cluster Management for Kubernetes 2.0

アプリケーションの管理

アプリケーションの管理

Last Updated: 2021-01-27

Red Hat Advanced Cluster Management for Kubernetes 2.0 アプリケーションの管理

アプリケーションの管理

法律上の通知

概要

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

目次

第1章 アプリケーションの管理 **3**
1.1. アプリケーション管理ライフサイクル 3

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。

- アプリケーション管理ライフサイクル
- アプリケーションモデルおよび定義
- アプリケーションリソース
- コンソールでのアプリケーションの管理
- チャネルの作成および管理
- サブスクリプションの作成および管理
- 配置ルールの作成および管理
- アプリケーションリソースの作成および管理
- アプリケーションリソースを使用したデプロイ
- アプリケーションリソースの例

1.1. アプリケーション管理ライフサイクル

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。

サブスクリプション コンポーネントは、**配置ルール** リソースを使用して、Kubernetes リソースのデプロイ先のマネージドクラスターを定義します。

アプリケーションモデルの最後のコンポーネントは、1 つまたは複数のリポジトリーサブスクリプションを参照する **アプリケーション** リソースです。アプリケーションの目的は、デプロイされた Kubernetes リソースをグループ化して、データを累積し、データの理解と管理を簡素化します。

以下のトピックで、アプリケーションライフサイクルの詳細を説明します。

- アプリケーションモデルおよび定義
- コンソールでのアプリケーションの管理

1.1.1. 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、チャネル、サブスクリプション、配置ルール、およびアプリケーションといった Kubernetes カスタムリソースで構成されます。

- 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 リポジトリー、Helm リリースレジストリー、オブジェクトストア、ハブクラスターにあるリソーステンプレート (deployable) の namespace です。
  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、Kubernetes namespace、オブジェクトストアなどの別のチャネルタイプで namespace を共有できます。

- 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、オブジェクトストア、またはリソーステンプレート (deployable) の namespace などのソースリポジトリーにサブスクライブできます。サブスクリプションは、ローカルでハブクラスターまたはマネージドクラスターに適用できます。

- 配置ルール (**placementrule.apps.open-cluster-management.io**) は、サブスクリプションが Kubernetes リソースをデプロイして管理する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、マルチクラスターのデプロイメントが容易になります。配置ルールは、サブスクリプション全体に共有できます。

- Red Hat Advanced Cluster Management for Kubernetes のアプリケーション (**application.app.k8s.io**) は、アプリケーションを構成する Kubernetes リソースのグループ化に使用します。

アプリケーションリソースの作成および管理に関する詳細は、以下を参照してください。

- チャネルの作成および管理
- サブスクリプションの作成および管理
- 配置ルールの作成および管理
- アプリケーションリソースの作成および管理

1.1.2. コンソールでのアプリケーションの管理

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、アプリケーションリソースを作成して管理できます。また、アプリケーションのステータスを表示することもできます。ダッシュボードには、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、可視化に使用できる強化機能が含まれています。

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。

- トポロジービューを使用して、関連するチャネルやサブスクリプションなど、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。

- チャネル、サブスクリプション、配置ルールなど、新しいアプリケーションリソース定義を含む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。

- チャネル、サブスクリプション、配置ルール、およびアプリケーションを追加および編集する。

全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

コンソールには、さまざまな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。

- アプリケーションダッシュボード
- 検索
- リソーストポロジー

1.1.2.1. アプリケーションダッシュボード

メインの **Applications** ダッシュボードから、すべてのアプリケーションの情報を表示して、特定のアプリケーションを選択し、表示できます。

このダッシュボードの **Overview** タブで、全アプリケーションに対する以下のタスクを実行できます。

- 全アプリケーションを一覧表示するテーブルを表示できます。
- **Find resouces** ボックスを使用して、一覧表示されたアプリケーションをフィルタリングします。
- アプリケーション名と namespace を表示します。
- サブスクリプションを使用してアプリケーションをデプロイする先のマネージドクラスターの数を表示します。
- アプリケーションのデプロイに使用する **ハブ** クラスター上のサブスクリプションの数と関連するステータスを表示します。
- アプリケーションの作成日を表示します。
- **Delete application** などの他のアクションについては、**Options** をクリックします。
- テーブルのアプリケーション名をクリックし、その特定のリソースに関する詳細情報を表示します。
- **Resources** タブからリソースパイプラインにアクセスします。
- **Resources** を確認するには、**Subscription** をクリックして **Search** ページを表示し、ハブクラスターのサブスクリプションとマネージドクラスターにデプロイされたすべてのサブスクリプションを確認します。
- さまざまなカードにあるすべてのアプリケーションリソースの概要を表示します。これらのリソースのサマリーカードには、以下のようなアプリケーションおよび関連するマネージドクラスターに関する情報が表示されます。
  - ハブクラスターのサブスクリプション数と、ハブクラスターへのデプロイに失敗したサブスクリプションの数。
  - マネージドクラスターの合計数。
  - これらのマネージドクラスター向けにアクティブなサブスクリプションの数。
  - 全アプリケーションのチャネルおよび配置ルールの合計数。

各サマリーカードをクリックして **Search** ページを開き、選択したリソースの詳細情報を表示できます。

- このページから、複数の **Create** オプションをクリックし、YAML エディターで追加のリソースを作成することもできます。
  これらのリソースを作成する場合には、リソースの定義に使用できる YAML エディターが開きます。デフォルトのサンプル YAML コンテンツの例は、テンプレートとして含まれており、必要なフィールドが示されているのでこのようなリソースタイプの作成に便利です。

  全アプリケーションのリソースを表示できます。このパイプラインには、アプリケーションや他のリソースの詳細を表示するテーブルが含まれます。

  - 各行には 1 つのアプリケーションの詳細が含まれます。行は全アプリケーションに含まれます。必要に応じて、検索ボックスを使用して、アプリケーションの行をフィルタリングし、一致するアプリケーションを検索できます。

  - 各アプリケーションの行を展開し、関連付けられたチャネル、サブスクリプション、配置ルールの詳細、関連するインシデント、リソース (Pod) デプロイメントのステータスを表示します。

  - サブスクライブしている deployable、関連のチャネル、namespace、ホストクラスター、関連リソース (Pod) デプロイメント、サブスクリプションで使用する配置設定など、チャネルおよびアプリケーションの関連するサブスクリプションを選択して、そのサブスクリプションの詳細を表示できます。

1.1.2.1.1. アプリケーションダッシュボード (単一アプリケーション)

1 つのアプリケーションのダッシュボードの **Overview** タブから、以下のタスクを実行できます。

- さまざまなカードにあるアプリケーションリソースのハイライトに関する概要を表示します。これらのリソースのサマリーカードには、以下の情報が表示されます。

  - アプリケーションのハブクラスターのサブスクリプション数と、ハブクラスターへのデプロイに失敗したサブスクリプションの数。

  - アプリケーションが使用されるマネージドクラスターの合計数。このカードでは、これらのマネージドクラスターにあるアプリケーションのサブスクリプションの合計数と、マネージドクラスターにデプロイできなかったサブスクリプション数も表示されます。

  - 実行中の Pod 数、失敗した Pod 数など、アプリケーションの Pod 数。

  - アプリケーションに関連するポリシー違反およびインシデントの数。各サマリーカードをクリックして **Search** ページを開き、選択したリソースの詳細情報を表示できます。

- アプリケーションの namespace、バージョン、作成日、関連するラベルおよびアノテーションなど、アプリケーションの他の情報を表示します。

- アプリケーションのリソーストポロジーを表示します。ここでは、チャネル、deployable、関連サービス、Pod、配置ルールなど、アプリケーションおよび関連コンポーネントが表示されます。トポロジーを編集し、リソースの追加、変更、または削除など、アプリケーションおよび関連リソースの YAML 定義を更新できます。

単一アプリケーションの **Resources** タブから、以下のタスクを実行できます。

- アプリケーションのリソース一覧を表示します。この一覧には、Pod、シークレット、サービスなど、アプリケーションに関連する全リソースが含まれます。この一覧は、各リソースの名前、namespace、kind、API グループ、およびステータスを示します。また、この一覧は、リ

ソースのデプロイ先のクラスター、リソースの作成日、リソースの最終更新日も指定します。

- アプリケーションのリソースパイプラインを表示します。これには、アプリケーションリソースの概要、リソースの作成オプション、およびそのリソースの詳細な情報を提供するテーブルが含まれます。単一アプリケーションのリソースパイプラインには、全アプリケーションのパイプラインと同様の情報が含まれます。この 2 つのパイプラインの相違点として、こちらのパイプラインは、選択したアプリケーションのみを対象として設定されています。

1.1.2.2. 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)
アプリケーション	**Application**
チャネル	**Channel**
Deployable	**Deployable**
シークレット	**Secret**
配置ルール	**PlacementRule**
サブスクリプション	**Subscription**

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。

必要に応じて、リソースの検索結果の **Options** メニューを展開して、対象のリソースを削除することもできます。

検索結果でリソース名をクリックすると、YAML エディターが開き、リソースの YAML 定義が表示されます。エディター内で定義を編集することもできます。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。

1.1.2.3. リソーストポロジー

アプリケーショントポロジーでは、視覚的にステータスを表示するアプリケーションカードが含まれます。各アプリケーションのトポロジービューには、そのアプリケーションのサービス、デプロイメント、チャート、および Pod が含まれます。

- 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。
- リソースの上にマウスをかざすと、component kind、名前、namespace、リソースまたは namespace の検索結果を表示するリンクを表示できます。

- Pod の詳細を表示します。その Pod のログを表示することもできます。
- 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。

  = アプリケーションリソース

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。Red Hat Advanced Cluster Management for Kubernetes アプリケーションの基本的なリソースは、**application** リソースと **deployable** リソースです。

さらに、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には、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。

シークレットリソースの操作に必要なアプリケーションがある場合には、サブスクリプションを使用して、リソースが必要なマネージドクラスターにシークレットをデプロイしてください。

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。サブスクリプション内にシークレットへの参照を追加するには、セキュアなチャネルへのアクセスに必要な認証情報を、サブスクリプションに指定します。このようなアクセス設定をすることで、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。

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。

1.1.3. チャネル

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。チャネルは、カスタムリソース定義のことで、これを使用してデプロイメントの合理化、クラスターアクセスの分離を容易にします。

チャネルの作成と管理の詳細は、「チャネルの作成および管理」を参照してください。YAML ファイルの例については、「アプリケーションリソースの例」を参照してください。

チャネルは、ハブクラスター内の namespace を定義して、オブジェクトストア、Kubernetes namespace、Helm リポジトリー、Git リポジトリーなど、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 deployable を特定できます。

- 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。

deployable の配置と上書き、依存関係は、deployable の仕様で指定されます。deployable の配置は、マルチクラスターデプロイメントの場合など、サブスクリプション内で定義することもできます。

- シークレットは、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。

「シークレットの管理」を参照してください。YAML ファイルの例については、「シークレットの例」を参照してください。

Namespace と **ObjectBucket** のチャネルタイプでは、チャネルごとの仕様で、チャネルに含めるために deployable が満たす必要のある条件を定義します。これらの条件は、ソース namespace、パッケージ名、ラベル、アノテーションなど、チャネルの Kubernetes ラベルとして定義されます。deployable は、チャネルに追加するために deployable と同じラベルを指定する必要があります。deployable は、チャネルと同じラベルを使用している場合のみ、チャネルに追加できます。

1.1.4. サブスクリプション

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。サブスクリプションの作成と管理の情報は、「サブスクリプションの作成および管理」を参照してください。YAML ファイルの例については、「アプリケーションリソースの例」を参照してください。

サブスクリプションは、アノテーション、ラベル、およびバージョンを使用して、チャネル内の Helm チャート、deployable、および他の Kubernetes リソースを特定する定義セットです。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 deployable を特定できます。サブスクリプション operator は、先にハブクラスターに確認しなくても、サブスクライブされている Helm チャート、deployable、またはシークレットを直接、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。

1.1.5. 配置ルール

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。配置ルールの作成および管理の詳細は、「配置ルールの作成および管理」を参照してください。YAML ファイルの例については、「配置ルールの例」を参照してください。

以前の Red Hat Advanced Cluster Management for Kubernetes バージョンのアプリケーションに使用されていた配置ポリシーが、配置ルールのカスタムリソース定義 (CRD) およびコントローラーに置き換えられました。ただし、配置ポリシーは、そのままセキュリティーポリシーおよびリスクポリシーに使用されます。

配置ルールは、サブスクリプションおよび deployable に定義できます。マルチクラスターデプロイメントのサブスクリプションレベルで配置ルールを定義します。単一クラスターデプロイメントの特定の deployable を指定する場合や、配置設定を上書きする場合に、配置ルールを定義します。

1.1.6. 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、アプリケーションコンポーネントの表示に使用します。YAML ファイルの例については、「アプリケーションリソースの例」を参照してください。

1.1.6.1. チャネルの作成および管理

チャネルを作成して使用し、Red Hat Advanced Cluster Management for Kubernetes アプリケーションの継続的インテグレーション/デリバリー機能にアプリケーションリソースを提供します。

チャネルは、マネージドクラスターに対するこれらのリソースのデプロイと分けてアプリケーションを管理できるカスタムリソースです。チャネルなどの全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

チャネル (**channel.apps.open-cluster-management.io**) は、デプロイメント用にリソースを保存する物理的な場所を参照します。チャネルは、ハブクラスターの namespace 内に作成し、ここにチャネルに関連する他のリソースも保存されます。

チャネルには 4 つのタイプがあります。リソースの保存先である、ソースのロケーションの種類をもとにチャネルはそれぞれ異なります。

- Kubernetes namespace (**Namespace**): deployable を使用して Kubernetes リソーステンプレートを保存します。このチャネルタイプのサブスクリプションは、テンプレートを取得してデプロイします。チャネルにより deployable が新たに追加されたり、更新されたりしていないかモニタリングする namespace はハブクラスター上になければなりません。

- オブジェクトストア (**ObjectBucket**): Kubernetes リソースの YAML ファイrを保存します。各 YAML ファイルには、完全な deployable オブジェクトではなく、リソース 1 つのテンプレートポーションが含まれます。オブジェクトストアは、ハブクラスター上にある deployable オブジェクトを使用するか、継続的インテグレーションパイプラインから直接か、または必要な YAML ファイルをオブジェクトストアに追加して、生成できます。

- Helm リポジトリー (**HelmRepo**): Helm チャートを保存します。チャートの構成に関する詳細は、Helm ドキュメント を参照してください。

- GitHub リポジトリー (**GitHub**): Kubernetes リソースの YAML ファイルと、パッケージされていない Helm チャートを保存します。これらのリソースは、deployable としてラップしたり、表示したりする必要はありません。チャネルコントローラーは、deployable としてリソースを自動的に同期します。
  サポート対象の **Git** ソースを確認してください。

  - GitHub
  - GitLab
  - Bitbucket
  - Gogs (webhook はサポート対象外)

チャネルは、リソースの保存場所を参照してハブクラスター上に作成されます。チャネルは、**Channel** のカスタムリソース (CR) で定義します。

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、Kubernetes namespace、オブジェクトストアなどの別のチャネルタイプで namespace を共有できます。

クラスターがチャネルをサブスクライブしている場合に、マネージドクラスター上のサブスクリプション operator は対象のチャネル namespace を利用できるようになります。その後、Operator はシークレットを取得して実際のチャネルにアクセスできます。Operator は、deployable がチャネル要件を満たしているかどうかを確認するのに、このアクセス権が必要です。

アプリケーションリソースの詳細は、「アプリケーションリソース」を参照してください。

チャネルの詳細は、以下のタスクを参照してください。

- チャネルの作成
- チャネルの更新
- チャネルの削除
- チャネルを使用したデプロイメントの管理
- チャネルのステータスと同期

1.1.6.1.1. チャネルの作成

1. チャネル定義 YAML コンテンツを作成します。チャネルリソースを作成または更新するには、まずリソース定義用の YAML ファイルを作成する必要があります。必要なフィールドなど YAML の構造に関する詳細は、「チャネル定義の YAML の構造」を参照してください。

2. 一意の namespace でチャネルを作成するようにしてください。すべてのチャネルには個別の namespace が必要ですが、Git チャネルは例外で、別のチャネルと namespace を共有できます。

3. 任意。**Namespace** タイプのチャネルを作成する場合には、ハブクラスターに namespace を作成します。
チャネルタイプは、チャネル仕様の **spec.sourceNamespaces** と **spec.type** フィールドで指定できます。チャネルを作成してソースを参照したら、チャネルコントローラーは、ソースの deployable をプロモーションを管理します。このコントローラーは、ソースとチャネルの namespace も同期します。

   YAML 定義の一部として namespace を定義したり、Kubernetes コマンドラインインターフェース (**kubectl**) ツールを使用して namespace を作成したりできます。Kubernetes CLI ツールを使用するには、以下のコマンドを実行します。**namespace name** は新しい namespace の名前に置き換えます。

   ```
   kubectl create namespace <namespace name>`
   ```

4. 任意。チャネルによるリポジトリーやチャートへのアクセスの認証に、Kubernetes Secret を使う必要がある場合には、Kubernetes Secret リソースを作成します。以下のコマンドを実行して、Kubernetes コマンドラインインターフェース (**kubectl**) ツールを使用して Kubernetes Secret を作成できます。

   ```
   kubectl create secret <secret name> generic --from-literal=user --from-literal=password=
   ```

   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。シークレットとチャネルを関連付けるには、チャネルの YAML 定義に **spec.secretRef.name** 設定を追加します。

5. Red Hat Advanced Cluster Management for Kubernetes でのチャネルの作成コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

   - コンソールを使用するには、以下を行います。

     i. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

ii. **Resources** タブをクリックします。

iii. **Resource pipeline** セクションまでスクロールします。ボタンの一覧から、**Channel** を探し出して、クリックします。**Create a Channel** エディターが表示されます。

iv. YAML コンテンツを入力してチャネルを定義するか、または要件に合わせてデフォルトの YAML テンプレートを直接更新します。

v. 完了したら、**Save** をクリックしてチャネルを作成します。新しいチャネルは、対応するアプリケーションの **Resource パイプライン** に表示されます。

または、特定のアプリケーションで作業する場合にチャネルを作成することもできます。

i. 全アプリケーションの **Overview** タブの **All applications** リストからアプリケーションをクリックします。選択したアプリケーションの **Overview** タブが開きます。

ii. そのアプリケーションの **Resources** タブをクリックします。

iii. **Resource pipeline** セクションまでスクロールします。リソースのサマリーカードの右側にあるボタン一覧から、**Channel** をクリックします。**Create a Channel** エディターが表示されます。

iv. YAML コンテンツを入力してチャネルを定義するか、または要件に合わせてデフォルトの YAML テンプレートを直接更新します。

v. 完了したら、**Save** をクリックしてチャネルを作成します。新しいチャネルは、アプリケーションの **Resource パイプライン** に表示されます。

- Kubernetes CLI ツールを使用するには、以下を実行します。

vi. 任意の編集ツールで、チャネルの YAML ファイルを作成して保存します。

vii. 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
```

viii. 以下のコマンドを実行して、チャネルリソースが作成されていることを確認します。

```
kubectl get Channel
```

新しいチャネルが出力に表示されていることを確認します。** REST API を使用するには、「API」を使用します。

1.1.6.1.2. チャネルの更新

1. チャネルの定義更新を作成します。必要なフィールドなど YAML の構造に関する詳細は、「チャネル定義の YAML の構造」を参照してください。

2. 定義を更新します。コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

- コンソールを使用するには、以下を行います。

i. コンソールを開きます。

ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

iii. **Resources** タブをクリックします。

iv. **Resource pipeline** セクションまで、ページを下方向にスクロールします。更新するチャネルの **YAML** 編集アイコンをクリックします。**Edit channel** ウィンドウが開きます。

v. チャネルの YAML を編集します。

vi. 完了したら、**Save** をクリックしてチャネルを更新します。

コンソール検索を使用して、チャネルの検索や編集も可能です。

i. ナビゲーションメニューから **Search** をクリックします。

ii. 検索ボックスで、**kind:chanel** でフィルタリングして全チャネルを表示します。

iii. すべてのチャネルの一覧で、更新するチャネルをクリックします。チャネルの YAML が表示されます。

iv. **Edit** をクリックして YAML コンテンツの編集を有効にします。

v. 編集が完了したら、**Save** をクリックします。変更が保存され、自動的に適用されます。

- Kubernetes CLI ツールの使用方法は、チャネルの作成の手順と同じです。

> To use REST API, use the link:../apis#apis[APIs].

1.1.6.1.2.1. チャネルの削除

チャネルの削除には、コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

コンソールを使用するには、以下の手順を実行します。

1. ナビゲーションから **Manage applications** をクリックします。
2. **Resources** タブをクリックします。
3. 削除するチャネルリソースカードを検索します。
4. 他のアクションについては、**Options** をクリックします。
5. **Delete channel** をクリックします。
6. 全チャネル一覧を更新すると、このチャネルが表示されなくなることを確認します。

Kubernetes CLI ツールを使用するには、以下を実行します。

1. 以下のコマンドを実行して、ターゲット namespace からチャネルを削除します。
2. **name** と **namespace** は、チャネル名と、ターゲットの namespace に置き換えます。

```
kubectl delete Channel <name> -n <namespace>
```

3. 以下のコマンドを実行してチャネルが削除されていることを確認します。

```
kubectl get Channel <name>
```

REST API を使用するには、「API」を使用します。

1.1.6.1.3. チャネルを使用したデプロイメントの管理

チャネルとサブスクリプションを使用して、Helm チャートや、Kubernetes deployable オブジェクトなど、マネージドクラスターや、他の namespace への deployable の継続的デリバリーを管理します。

チャネルが Helm リポジトリーを参照すると、チャネル Operator はリポジトリーにある各 Helm リリースを表す deployable を作成します。

Kubernetes deployable オブジェクトの場合は、deployable としてオブジェクトをラップしてチャネルに追加できます。

deployable の定義で、deployable のプロモート先のチャネルを直接指定できます。deployable を自動的にチャネルに追加できるようにゲート要件に一致する deployable の必須の Kubernetes ラベルを指定することも可能です。deployable に必要な Kubernetes ラベルが含まれていることをチャネルコントローラーが検出すると、このコントローラーは deployable をチャネルにプロモートします。

Namespace と **ObjectBucket** チャネルの作成時に、チャネル定義の **spec.gates** セクションでチャネルのゲート要件を設定できます。これらの要件は、deployable がチャネルにプロモートされる前に deployable に含める必要のある Kubernetes アノテーションです。たとえば、アノテーションを指定して開発の承認、テストおよび品質保証の承認をしたり、実稼働環境のクラスターへのデプロイメントの準備ができていることを示したりすることができます。これらのゲート要件には、ソースの namespace、パッケージ名、ラベル、アノテーションなどのフィールドおよび値などが含まれます。deployable の定義に一致するフィールドや値がある場合にのみ、deployable をチャネルにプロモートできます。deployable が定義済みの要件を満たす場合に、その deployable はチャネルに自動的にプロモートされ、そのチャネルにサブスクライブするマネージドクラスターにデプロイされます。ゲート要件は、**HelmRepo** と **Git** のチャネルタイプには適用されません。

チャネルにゲート要件が含まれない場合には、チャネルコントローラーは最新版の deployable をチャネルにプロモートします。

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 deployable を特定できます。チャネルにプロモートされた deployable には、そのチャネルのサブスクリプションでしかアクセスできません。チャネルとサブスクリプションが存在する場合には、チャネルとサブスクリプションが連携してチャネルソースから deployable を取得し、あて先に deployable を配置します。あて先は通常、namespace として抽象化された、マネージドクラスターです。マネージドクラスターまたは namespace は、複数のチャネルにサブスクライブして、クラスターにデプロイする deployable を特定することができます。チャネルを使用することで、適切な deployable が確実に取得されるようにします。また、サブスクリプションを使用することで、確実に deployable が取得され、あて先の namespace またはクラスターに配置されるようにします。サブスクリプション Operator は、deployable を取得する際に、アノテーションの制限を確認し、deployable を取得してマネージドクラスターに適用するかどうかを判断します。

1.1.6.1.4. チャネルのステータスと同期

チャネルにはステータスはありませんが、チャネルにサブスクライブされているサブスクリプションにはステータスがあります。サブスクリプションのステータスは、サブスクリプションがハブクラスター上で正常に伝搬されているかどうか、また、deployable テンプレートが正常に作成または適用されているかどうか、また helmRelease CR が作成されているかどうかを報告します。チャネルおよびサブスクリプションを使用してデプロイされる deployable のステータスは、サブスクリプションとは別に報告されます。

チャネルにはステータスがないため、チャネル namespace にプロモートされるリソースは、常に実際のチャネルストレージと同期されるわけではありません。

deployable がチャネルに追加されるか、更新されると、チャネルのサブスクリプション Operator は自動的に新規または更新された deployable を検出します。ハブクラスターからターゲットのマネージドクラスターに通知を発行する必要はありません。

namespace タイプのチャネルにサブスクライブしている場合には、クラスターがそのチャネルの namespace にアクセスできる場合のみ、リソースがマネージドクラスターに同期されます。チャネルソースはハブクラスターに存在するため、ハブクラスターにアクセスできる場合にのみ、サブスクリプションはソースからリソースをプルできます。

Helm リポジトリーとオブジェクトストアタイプのチャネルにサブスクライブしている場合には、サブスクリプションは、新規または更新された Helm チャートまたは deployable がないか、チャネルのソースリポジトリーを監視します。サブスクリプションは、ソースリポジトリーがハブクラスターにある限り、ハブクラスターと通信する必要はありません。サブスクリプションが Helm リリースまたは deployable オブジェクトの最新版をプルするように設定されていて、新規バージョンがそのリポジトリータイプに含まれている場合には、サブスクリプションでこれらのバージョンを取得できます。

1.1.6.1.5. シークレットの管理

サブスクリプションやアプリケーションコンポーネントの認可認証情報やその他の機密情報を保存する Kubernetes Secret リソースを作成します。

シークレットなどのリソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

1.1.6.1.5.1. Kubernetes Secret

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。

シークレットを使用すると、サブスクリプションで必要な場所に情報を送信しつつ、情報を安全に分離して保存できます。

たとえば、デプロイメントリソースが参照するプライベート Docker イメージのレジストリーの認証情報など、イメージのプルシークレットを使用できます。

Helm チャートまたはデプロイされた Pod へのサブスクリプションが含まれるマネージドクラスターの namespace にイメージプルシークレットを追加すると、Helm チャートまたは Pod はそのプルシークレットを namespace のスコープ内で使用できるようになります。

チャネルにシークレットを作成して追加し、マネージドクラスターにデプロイすると、そのシークレットは、ターゲットのマネージドクラスターにある、サブスクリプションと同じ namespace にだけデプロイできます (サブスクリプションはハブおよびマネージドクラスターにある同じ namespace を使用します)。

すべてのシークレットデータはエンドツーエンドで暗号化されます。ハブクラスターおよびマネージドクラスターでは、Kubernetes は使用中ではないシークレットを暗号化します。トランスポート中は、TLS 暗号化が使用されます。

1. ハブクラスターに **チャネル** および **シークレット** リソースの保存先の namespace を作成します。
   必要なアクセス権限: クラスターの管理者この手順を実行するのは一度だけです。以下のコマンドを実行してください。

```
oc new-project SECRET_NAMESPACE
```

または、**oc apply -f FILENAME.yaml** を実行して以下の YAML サンプルを適用することもできます。以下をコピーアンドペーストしてください。

```
apiVersion: v1
kind: Namespace
metadata:
  name: SECRET_NAMESPACE
```

2. チャネルリソースを作成します。シークレット namespace にはチャネルリソースも追加する必要があります。**oc apply -f FILENAME.yaml** コマンドを実行して、以下のサンプル YAML ファイルを適用します。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: CHANNEL_NAME
  namespace: SECRET_NAMESPACE
spec:
  type: Namespace
  pathname: SECRET_NAMESPACE
```

以下の YAML 定義の表には、指定する必要のある値/キーが含まれます。

キー	値
name	一意のチャネル名。dns 形式に制限されます。
namespace	任意。指定しない場合には、現在の namespace 環境にチャネルが作成されます。サブスクライブする必要のあるシークレットと同じである必要があります。
pathname	namespace と同じ値である必要があります。

3. **oc apply -f FILENAME.yaml** コマンドを実行してシークレットを作成して、以下のサンプル YAML ファイルを適用します。シークレットを必要な数だけ作成できます。

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
     apps.open-cluster-management.io/deployables: "true"
  name: SECRET_NAME
  namespace: SECRET_NAMESPACE
data:
```

以下の YAML 定義の表には、指定する必要のある値/キーが含まれます。

キー	値
name	一意名。dns 形式に制限されます。
namespace	任意。指定しない場合には、現在の namespace 環境にシークレットが作成されます。
data	これは YAML または、セキュアに保存する大きいサイズの文字列ブロックです。

4. サブスクリプションを作成します。サブスクリプションは、チャネルから 1 つ以上のシークレットをサブスクライブするために使用され、独自の namespace で作成する必要があります。この namespace は、シークレットが作成されたマネージドクラスターに伝播されます。
以下の例は、ローカルのサブスクリプションの配置を定義します。

```
placement:
  local: true
```

以下の例は、マネージドクラスター (リモート) のサブスクリプション配置を定義します。

```
placement:
  placementRef:
    name: PLACEMENT_NAME
    kind: PlacementRule
```

以下の例は、サブスクリプションを定義します。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: SUBSCRIPTION_NAME
  namespace: SUBSCRIPTION_NAMESPACE
spec:
  channel: CHANNEL_NAMESPACE/CHANNEL_NAME
  placement:
    local: true
```

5. チャネル内のシークレットがすべて適用されていることを確認するには、以下のコマンドを実行します。

```
oc -n SUBSCRIPTION_NAMESPACE describe appsub SUBSCRIPTION_NAME
```

1.1.6.2. サブスクリプションの作成および管理

サブスクリプションを作成して管理し、マネージドクラスターへの新規および更新リソースの特定、取得、デプロイを行うことができます。サブスクリプションを使用すると、アプリケーション管理の継続的デリバリー機能を向上できます。サブスクリプションなどの全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

サブスクリプションの詳細は、以下のタスクを参照してください。

- サブスクリプションの作成
- アプリケーションとサブスクリプションの一致
- サブスクリプションの更新
- デプロイメントのスケジュール
- Git リソースのサブスクライブ
- パッケージの上書きの設定
- サブスクリプションの削除

サブスクリプション (**subscription.apps.open-cluster-management.io**) は、定義セットとして機能する Kubernetes リソースで、アノテーション、ラベル、バージョンを使用してチャネル内の Helm チャートと、Kubernetes リソース (Git、Objectstores またはハブクラスターの deployable) を特定します。

サブスクリプションリソースは、チャネルを参照して、デプロイメント用に、新規および更新された Helm チャートまたは Kubernetes リソースを特定できます。次に、サブスクリプション Operator は、新規または更新されたチャートや deployable がないかどうかチャネルを監視します。

新規または更新された Helm チャートまたは Kubernetes リソースが検出されると、サブスクリプション operator は指定された Helm チャートバージョンのの Helm リリースバージョンまたは指定の Kubernetes リソースをダウンロードします。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、これらのオブジェクトを直接、または deployable として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できます。

サブスクリプションを使用すると、チャネルにプロモートされた deployable をフィルターして、特定の deployable を選択することができます。たとえば、サブスクリプションでは deployable をフィルターして、特定の deployable バージョンを選択できます。このような場合に、サブスクリプション operator は **version** パラメーターをチェックして、選択する deployable バージョンを特定します。

1.1.6.2.1. サブスクリプションの作成

1. サブスクリプションの定義 YAML コンテンツを作成します。YAML の構造および例の詳細は、「アプリケーションリソースの例」ドキュメントを参照してください。

2. Red Hat Advanced Cluster Management for Kubernetes でのサブスクリプションの作成コンソール、Kubernetes CLI (**kubectl**) ツール、または REST API を使用できます。

   - コンソールを使用するには、以下を行います。

     i. コンソールを開きます。

     ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

     iii. **Resources** タブをクリックします。

     iv. **Resource pipeline** セクションまでスクロールします。リソースのサマリーカードの右側にあるボタン一覧から、**Subscription** をクリックします。**Create a Subscription** エディターが表示されます。

     v. YAML コンテンツを入力してサブスクリプションを定義するか、または要件に合わせてデフォルトの YAML テンプレートを直接更新します。

vi. 完了したら、**Save** をクリックしてサブスクリプションを作成します。新しいサブスクリプションは、対応するアプリケーションおよびチャネルの **Resource パイプライン** に表示されます。

または、特定のアプリケーションで作業する場合にサブスクリプションを作成することもできます。

i. 全アプリケーションの **Overview** タブの **All applications** リストからアプリケーションをクリックします。選択したアプリケーションの **Overview** タブが開きます。

ii. そのアプリケーションの **Resources** タブをクリックします。

iii. **Resource pipeline** セクションまでスクロールします。リソースのサマリーカードの右側にあるボタン一覧から、**Subscription** をクリックします。**Create a Subscription** エディターが表示されます。

iv. YAML コンテンツを入力してサブスクリプションを定義するか、または要件に合わせてデフォルトの YAML テンプレートを直接更新します。

v. 完了したら、**Save** をクリックしてサブスクリプションを作成します。新しいサブスクリプションは、対応するアプリケーションおよびチャネルの **Resource パイプライン** に表示されます。

注記: アプリケーションのリソースパイプラインに、最初からサブスクリプションが表示されない場合があります。サブスクリプションとアプリケーションを関連付けるには、サブスクリプション定義に適切な値を追加して、アプリケーション定義に指定した値と一致させる必要があります。詳細は、「アプリケーションとサブスクリプションの一致」を参照してください。

- Kubernetes CLI ツールを使用するには、以下を実行します。

  i. 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
  ```

  ii. 以下のコマンドを実行して、サブスクリプ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 appsub
  ```

  新しいサブスクリプションが出力に一覧表示されていることを確認します。

- REST API を使用するには、「API」を使用します。

サブスクリプションの作成後に、サブスクリプションのステータスは以下のいずれかになります。

- **Subscribed** (マネージドクラスターのみ)
  特定のリソースにサブスクライブされて、リソースがデプロイされた。

- **Propagated** (ハブクラスターのみ)
  指定の **spec.placement** 設定をもとに、適切なマネージドクラスター (ある場合) にデプロイされるように、後続のサブスクリプションが生成される。ただし、サブスクリプションは実際にはデプロイされない場合があります。

- **Failed**: 特定のリソースへのサブスクライブに失敗した。

- **No status**

サブスクリプション operator がオンラインではないか、サブスクリプションの **spec.placement** の設定がない。

マルチクラスター環境のサブスクリプションを作成した場合には、サブスクリプションはハブクラスター上に作成されます。**spec.placement** の設定により、他のクラスターに配置される後続のサブスクリプションは異なります。**spec.placement** セクションに、複数の配置設定を追加した場合には、サブスクリプション operator は以下の優先度で、使用する配置設定を選択します。

- **spec.placement.placementRef**
  サブスクリプションは、指定の **PlacementRule** リソースにより特定されたクラスター上に配置されます。

- **spec.placement.clusters**
  サブスクリプションは、このフィールドの値として指定されたクラスターに配置されます。

- **spec.placement.clusterSelector**
  サブスクリプションは、指定のラベルセレクターと一致するクラスターに配置されます。ラベルセレクターは、このフィールドに値として定義します。

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サブスクリプションを作成した場合は、サブスクリプションはそのクラスターにのみ作成されます。サブスクリプション定義内の **spec.placement.local** フィールドに設定する値に応じて、サブスクリプションの動作は異なります。

- **true**
  サブスクリプションは、クラスターのローカルにある、指定のチャネルと同期します。

- **false**
  サブスクリプションは、指定のチャネルからリソースをサブスクライブしません。

1.1.6.2.2. アプリケーションとサブスクリプションの一致

サブスクリプションとアプリケーションを関連付けるには、サブスクリプションとアプリケーション両方を同じ namespace に配置して、チャネルからサブスクリプションが Helm チャート、deployable、他のリソースを取得できるようにします。

アプリケーションのリソース定義では、**spec.componentKinds** の設定を定義に含めて、アプリケーションがサブスクリプションを使用することを指定する必要があります。またこの定義には、**spec.selector** の設定を追加して、ラベル (**matchLabels**) または式 (**matchExpressions**) を定義して、サブスクリプションとアプリケーションを照合するのに使用する必要があります。

サブスクリプションリソースの定義には、アプリケーションが定義したラベルまたは式と一致する必須の値を追加する必要があります。

サブスクリプションがアプリケーションに関連付けられたら、 サブスクリプションは、サブスクリプションの **spec.placement** 設定または deployable を使用して、サブスクライブしているチャート、deployable、または他の Kubernetes リソースをデプロイします。

アプリケーションのリソース定義に関する詳細は、「アプリケーションリソースの作成および管理」を参照してください。

1.1.6.2.3. サブスクリプションの更新

1. サブスクリプションの定義 YAML コンテンツを作成します。

2. Red Hat Advanced Cluster Management for Kubernetes でのサブスクリプションの作成コンソール、Kubernetes CLI (**kubectl**) ツール、または REST API を使用できます。

- コンソールを使用するには、以下を行います。

  i. コンソールを開きます。

  ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

  iii. **Resources** タブをクリックします。

  iv. **Resource pipeline** セクションまで、ページを下方向にスクロールします。編集するサブスクリプションを使用するアプリケーションの行を展開します。

  v. 更新するサブスクリプションの YAML の**Edit** アイコンをクリックします。**Edit subscription** ウィンドウが開きます。

  vi. YAML を編集します。

  vii. 完了したら、**Save** をクリックしてサブスクリプションを更新します。

  または、特定のアプリケーションで作業する場合にサブスクリプションを更新することもできます。

  i. 全アプリケーションの **Overview** タブの **All applications** リストからアプリケーションをクリックします。選択したアプリケーションの **Overview** タブが開きます。

  ii. そのアプリケーションの **Resources** タブをクリックします。

  iii. **Resource pipeline** セクションまで、ページを下方向にスクロールします。

  iv. 更新するサブスクリプションの YAML の**Edit** アイコンをクリックします。**Edit subscription** ウィンドウが開きます。

  v. YAML を編集します。

  vi. 完了したら、**Save** をクリックしてサブスクリプションを更新します。

コンソール検索を使用して、サブスクリプションの検索や編集も可能です。

i. ヘッダーの **Search** アイコンをクリックします。

ii. 検索ボックスで、**kind:subscription** でフィルタリングして、全サブスクリプションを表示します。

iii. すべてのサブスクリプションの一覧で、更新するサブスクリプションをクリックします。サブスクリプションの YAML が表示されます。

iv. **Edit** をクリックして YAML コンテンツの編集を有効にします。

v. 編集が完了したら、**Save** をクリックします。変更が保存され、自動的に適用されます。

- Kubernetes CLI ツールの使用方法は、サブスクリプションの作成の手順と同じです。

> To use REST API, use the link:../apis#apis[APIs]

1.1.6.2.4. 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。また、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されないように制限できます。

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。

また、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され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。
- **Namespace** 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になると、サブスクリプションはチャネルと同期され、デプロイする必要があるリソースの最新バージョンのデプロイメントを開始します。
- サブスクリプションがブロックされている期間では、サブスクリプションは、リソースのデプロイメントチャネルと同期されません。

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。

1.1.6.2.4.1. Git リソースのサブスクライブ

Git リソースは、複数の namespace にサブスクライブできます。デフォルトでは、アプリケーションにサブスクライブすると、アプリケーションはそのサブスクリプションの namespace にデプロイされます。これらのリソースをサブスクリプション namespace 外の namespace に適用するには、以下の手順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. コンソールからハブ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。
これらのユーザーは、**app.open-cluster-management.io/subscription** アプリケーションの管理者を表します。OpenShift Container Platform では、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、本トピックで後ほど説明します。

3. ターミナルから、Red Hat Advanced Cluster Management ハブクラスターに再度ログインします。

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
```

サブスクリプション管理者としてログインしている場所の例を参照し、確認してください。この例では、Git リポジトリーからサンプルリソースの YAML ファイルをサブスクライブし、直前の手順に従い、更新された Configmap 設定を確認します。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。

- 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
```

1.1.6.2.5. 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 リソース仕様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 リリース仕様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、**HelmRelease** 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
  ```

1.1.6.2.6. サブスクリプションの削除

サブスクリプションの削除には、コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

コンソールを使用するには、以下の手順を実行します。

1. ナビゲーションから **Manage applications** をクリックします。
2. **Resources** タブをクリックします。
3. 削除するサブスクリプションリソースカードを見つけます。
4. 他のアクションについては、**Options** をクリックします。
5. **Delete subscription** をクリックします。
6. 全サブスクリプションの一覧を更新すると、サブスクリプションの表示がなくなることを確認します。

Kubernetes CLI ツールを使用するには、以下を実行します。

1. 以下のコマンドを実行して、ターゲット namespace からサブスクリプションを削除します。
2. **name** と **namespace** は、サブスクリプション名と、ターゲットの namespace に置き換えます。

   ```
   kubectl delete appsub <name> -n <namespace>
   ```

3. 以下のコマンドを実行してサブスクリプションが削除されていることを確認します。

   ```
   kubectl get appsub <name>
   ```

REST API を使用するには、「API」を使用します。

1.1.6.3. 配置ルールの作成および管理

配置ルールを作成して管理し、Helm チャートと deployable のデプロイ先とデプロイ方法を定義します。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。配置ルールなどの全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

- 配置ルールの作成
- 配置ルールの割り当て
- 配置ステータスの表示
- 配置ルールの更新
- 配置ルールの削除

以前の Red Hat Advanced Cluster Management for Kubernetes バージョンのアプリケーションに使用されていた配置ポリシーが、配置ルールのカスタムリソース定義 (CRD) およびコントローラーに置き換えられました。ただし、配置ポリシーは、そのままセキュリティーポリシーおよびリスクポリシーに使用されます。

配置ルールは、サブスクリプションおよび deployable に定義できます。マルチクラスターデプロイメントのサブスクリプションレベルで配置ルールを定義します。単一クラスターデプロイメントの特定の deployable を指定する場合や、配置設定を上書きする場合に、配置ルールを定義します。

1.1.6.3.1. 配置ルールの作成

配置ルールは、サブスクリプションおよび deployable に定義できます。マルチクラスターデプロイメントのサブスクリプションレベルで配置ルールを定義します。単一クラスターデプロイメントの特定の deployable を指定する場合や、配置設定を上書きする場合に、配置ルールを定義します。

前提条件: klusterlet-addon-appmgr Pod が実行されているようにしてください。**oc get pods -n open-cluster-management-agent-addon** を実行して、Pod の有無を確認します。

1. 配置ルールの定義 YAML コンテンツを作成します。配置ルールなどの全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。
2. Red Hat Advanced Cluster Management for Kubernetes に配置ルールを作成します。配置ルールを別のリソースとして定義するか、deployable またはサブスクリプションの定義にルールを定義できます。

ベストプラクティスとして、複数のリソースでルールを参照する必要がある場合には、配置ルールは個別のリソースとして定義します。

+ コンソール、Kubernetes CLI (**kubectl**) ツールまたは REST API を使用して、配置ルールを別のリソースとして作成できます。

- コンソールを使用するには、以下を行います。
  i. コンソールを開きます。
  ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

iii. **Resources** タブをクリックします。

iv. **Resource pipeline** セクションまでスクロールします。ボタンの一覧から検索し、**Placement Rule** をクリックします。**Create a placement rule**エディターが表示されます。

v. YAML コンテンツを入力して配置ルールを定義するか、または要件に合わせてデフォルトの YAML テンプレートを直接更新します。

vi. YAML の追加または編集が完了したら、**Save** をクリックして配置ルールを作成します。

vii. リソースのサマリーカードの一覧から、**PLACEMENT RULES** カードをクリックします。**Search** ダッシュボードが表示され、ここでは、**Applications** ダッシュボードで選択したアプリケーションが使用する配置ルールすべてが一覧表示されます。アプリケーションが使用するサブスクリプションで、ルールが参照されるまで、新しい配置ルールはこの一覧に表示されません。新しい配置ルールが作成されていることを確認するには、検索ボックスに **kind:placementrule** と入力し、検索を実行します。新しいルールが検索結果一覧に表示されることを確認します。

- Kubernetes CLI ツールを使用するには、以下を実行します。

  i. 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
  ```

  ii. 以下のコマンドを実行して、配置ルールが作成されていることを確認します。

  ```
  kubectl get PlacementRule
  ```

  新しい配置ルールが出力に一覧表示されていることを確認します。

- REST API を使用するには、「API」を使用します。

1.1.6.3.2. 配置ルールの割り当て

配置ルールは、deployable またはサブスクリプションに割り当てることができます。配置ルールを割り当てるには、deployable またはサブスクリプションの仕様を更新して配置ルールを参照する必要があります。

以下のように deployable またはサブスクリプション仕様の **placement** フィールドを追加して、配置ルールを参照するように値を指定します。

```
placement:
  placementRef:
    name:
    kind: PlacementRule
```

name フィールドの値として、配置ルールの名前を含めます。

配置ルールがサブスクリプションに割り当てられている場合は、以下の手順を使用してコンソール内のアプリケーションダッシュボードでその割り当てを表示できます。

1. コンソールを開きます。

2. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

3. **Resources** タブをクリックします。

4. **Resource pipeline** セクションまでスクロールします。アプリケーションを一覧表示するテーブルで、配置ルールを割り当てるサブスクリプションが含まれるアプリケーションの行を展開します。

5. アプリケーションの展開ビューから、チャネルごとに利用可能なサブスクリプションを確認できます。各サブスクリプションの詳細には、割り当てられた配置ルールが含まれます。必要に応じて、このリソースパイプラインテーブルから、配置ルール、サブスクリプション、およびチャネルの YAML を表示または編集できます。

1.1.6.3.3. 配置ルールの表示

配置ルールが作成され、使用中の場合には、ルールのステータスの詳細を表示できます。このステータスは、配置ルールの YAML 定義に追加します。このステータスでターゲットクラスターに対して、deployable の配置にルールを使用する先を示します。

配置ルールのステータスフィールドを表示するには、コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

- コンソールを使用するには、以下を行います。

  a. コンソールを開きます。

  b. ヘッダーの **Search** アイコンをクリックします。

  c. 検索ボックスで、 **kind:placementrule** でフィルタリングして、全配置ルールを表示します。

  d. すべての配置ルールの一覧から、レビューする配置ルールをクリックします。レビューするルールの YAML が表示されます。

  e. YAML コンテンツの **status** セクションでフィールドおよび値を確認します。

- Kubernetes CLI ツールを使用するには、以下のコマンドを実行します。**name** と **namespace** は、配置ルール名と、ターゲットの namespace に置き換えます。

  a. 以下のコマンドを実行します。

  ```
  kubectl get PlacementRule <name> -n <namespace>
  ```

  b. YAML コンテンツの **status** セクションでフィールドおよび値を確認します。

REST API を使用するには、「API」を使用します。

1.1.6.3.4. 配置ルールの更新

個別リソースである配置ルールを更新するには、コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

- コンソールを使用して配置ルールを編集するには、以下の手順を実行します。

  a. コンソールを開きます。

b. ヘッダーの **Search** アイコンをクリックします。

c. 検索ボックスで、 **kind:placementrule** でフィルタリングして、全配置ルールを表示します。

d. すべての配置ルールの一覧から、更新する配置ルールをクリックします。ルールの YAML が表示されます。

e. **Edit** をクリックして YAML コンテンツの編集を有効にします。

f. 編集が完了したら、**Save** をクリックします。変更が保存され、自動的に適用されます。

または、アプリケーションダッシュボードのパイプラインテーブルから YAML を編集することもできます。

a. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。

b. **Resources** タブをクリックします。

c. **Resource pipeline** セクションまでスクロールします。アプリケーションを一覧表示するテーブルで、配置ルールを割り当てるサブスクリプションが含まれるアプリケーションの行を展開します。

d. アプリケーションの展開ビューから、チャネルごとに利用可能なサブスクリプションを確認できます。各サブスクリプションの詳細には、割り当てられた配置ルールが含まれます。配置ルールのリンクをクリックして、**Edit placement rule** エディターを開きます。ルールの YAML が表示されます。

e. 編集が完了したら、**Save** をクリックします。変更が保存され、自動的に適用されます。

- Kubernetes CLI ツールの使用方法は、配置ルールの作成の手順と同じです。

REST API を使用するには、「API」を使用します。

deployable またはサブスクリプションの定義内に指定した配置ルールを更新する方法は、対象リソースの更新の手順と同じです。

1.1.6.3.5. 配置ルールの削除

コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用して、個別リソースである配置ルールを削除できます。

コンソールを使用するには、以下の手順を実行します。

1. ナビゲーションから **Manage applications** をクリックします。
2. **Resources** タブをクリックします。
3. 削除する配置ルールのサブスクリプションリソースカードを見つけます。
4. 他のアクションについては、**Options** をクリックします。
5. **Delete placement rule** をクリックします。
6. 全サブスクリプションの一覧を更新すると、配置ルールの表示がなくなることを確認します。

Kubernetes CLI ツールを使用するには、以下の手順を実行します。

1. 以下のコマンドを実行して、ターゲット namespace から配置ルールを削除します。
2. **name** と **namespace** は、配置ルール名と、ターゲットの namespace に置き換えます。

```
kubectl delete PlacementRule <name> -n <namespace>
```

3. 以下のコマンドを実行して、配置ルールリソースが削除されていることを確認します。

```
kubectl get PlacementRule <name>
```

REST API を使用するには、「API」を使用します。

1.1.6.4. アプリケーションリソースの作成および管理

アプリケーションリソースを作成し、Red Hat Advanced Cluster Management for Kubernetes のマルチクラスターアプリケーション全体を構成するアプリケーションコンポーネントをグループ化して表示します。全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。

- アプリケーションの作成
- アプリケーションとサブスクリプションの一致
- アプリケーションの更新
- アプリケーションの削除

1.1.6.4.1. アプリケーションの作成

1. アプリケーション定義の YAML コンテンツを作成します。アプリケーションリソースを作成または更新するには、まずリソース定義用の YAML ファイルを作成する必要があります。
2. Red Hat Advanced Cluster Management for Kubernetes でのアプリケーションの作成コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。
   - コンソールを使用するには、以下を行います。
     i. コンソールを開きます。
     ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。
     iii. **New application** をクリックします。**Create application** ウィンドウが開きます。
     iv. エディターでデフォルトの YAML コンテンツを更新または置き換えてアプリケーションを定義します。
     v. YAML の追加および編集が完了したら、**Save** をクリックしてアプリケーションを作成します。新規アプリケーションは **Overview** タブのアプリケーション一覧に表示されます。
   - Kubernetes CLI ツールを使用するには、以下を実行します。
     i. 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。

ii. 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
```

iii. 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
```

新しいアプリケーションが出力に一覧表示されていることを確認します。

- REST API を使用するには、「API」を使用します。

1.1.6.4.2. アプリケーションの更新

1. アプリケーション定義の更新を作成します。全リソースの例は、「アプリケーションリソースの例」のドキュメントで確認できます。
2. アプリケーション定義を更新します。コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。
   - コンソールを使用するには、以下の手順を実行します。
     i. コンソールを開きます。
     ii. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。
     iii. すべてのアプリケーションの一覧で、更新するアプリケーションの行を見つけます。その行で、**Options** メニューを展開し、**Edit application** をクリックします。**Edit application** ウィンドウが開きます。
     iv. アプリケーションの YAML コンテンツを更新します。
     v. 編集が完了したら、**Close editor** をクリックします。変更が保存され、自動的に適用されます。

   コンソール検索を使用して、アプリケーションの検索や編集も可能です。

   i. ヘッダーの **Search** アイコンをクリックして、**Search** ページを開きます。
   ii. 検索ボックスで **kind:application** でフィルタリングして、全アプリケーションを表示します。
   iii. すべてのアプリケーションの一覧で、更新するアプリケーションをクリックします。選択したアプリケーションの **Overview** ページが開きます。
   iv. **Edit app** をクリックします。**Edit application** ウィンドウが開きます。
   v. アプリケーションの YAML コンテンツを更新します。
   vi. 編集が完了したら、**Close editor** をクリックします。変更が保存され、自動的に適用されます。

   - Kubernetes CLI ツールの使用方法は、アプリケーションの作成の手順と同じです。

REST API を使用するには、「API」を使用します。

1.1.6.4.3. アプリケーションの削除

アプリケーションの削除には、コンソール、Kubernetes コマンドラインインターフェース (**kubectl**) ツールまたは REST API を使用できます。

コンソールを使用するには、以下の手順を実行します。

1. ナビゲーションメニューから **Manage applications** をクリックします。全アプリケーションの **Overview** タブが開きます。
2. すべてのアプリケーションの一覧で、削除するアプリケーションの行を見つけます。
3. その行で、**Options** メニューを展開し、**Delete application** をクリックします。確認ウィンドウが開きます。
4. アプリケーションを削除するには、削除を確定します。
5. 全アプリケーションの一覧を更新すると、アプリケーションが除外されます。

Kubernetes CLI ツールを使用するには、以下の手順を実行します。

1. 以下のコマンドを実行して、ターゲット namespace からアプリケーションを削除します。
2. **name** と **namespace** は、アプリケーション名と、ターゲットの namespace に置き換えます。

```
kubectl delete Application <name> -n <namespace>
```

3. 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削除されていることを確認します。

```
kubectl get Application <name>
```

REST API を使用するには、「API」を使用します。

1.1.6.4.4. アプリケーションリソースを使用したデプロイ

デプロイメントにチャネル、サブスクリプション、および配置ルールを設定して使用するには、以下の手順を実行します。

1. オブジェクトストア、Kubernetes namespace、または Helm リポジトリーを表すチャネルが存在しない場合は、チャネルを作成します。
   deployable が必要なラベルまたはアノテーションを定義してから、Deployable をチャネルにプロモートするようにしてください。詳細は、「チャネルの作成および管理」を参照してください。
2. ターゲットクラスターがチャネルにサブスクライブされていない場合は、サブスクリプションを作成します。詳細は、「サブスクリプションの作成および管理」を参照してください。
3. チャネルからデプロイする deployable の配置ルールを定義します。詳細は、「配置ルールの作成および管理」を参照してください。

配置ルールは、サブスクリプションおよび deployable に定義できます。**サブスクリプションの配置ルールを定義して、ルールを複数の deployable およびクラスターに適用できるようにします。** ルールがサブスクリプションの全 deployable に適用されないようにする場合や、またはサブスクリプションレベルの配置ルールを上書きする場合には、Deployable の配置ルールを定義します。配置ルールを参

照する場合は、deployable の上書き設定を追加して、deployable 固有の値で配置ルールの値を上書きすることもできます。任意。配置ルールをスタンドアロンリソースとして作成した場合には、サブスクリプションまたは deployable の定義を編集して、配置ルールを参照します。Deployable とチャネルの定義を編集して、deployable がチャネルにプロモートされるようにします。詳細は、「**チャネルへの deployable のプロモート**」を参照してください。コンソールを使用して、対象のチャネルにおけるターゲットクラスターのデプロイメントのステータスを監視します。

1.1.6.4.5. deployable のチャネルへのプロモート

deployable (**リソーステンプレート**) をチャネルにプロモートするには、以下の手法のいずれかを使用します。

- deployable 定義内にある **spec.channels** フィールドを正しいアノテーションで設定してチャネルを特定し、deployable が特定のチャネルを参照するようにします。
  spec.channels パラメーターは、deployable (**deployable.apps.open-cluster-management.io**) リソースが deployable のプロモート先のチャネルを特定するために利用できます。以下の例は、deployable が含まれるチャネルを定義する deployable を示しています。

```
apiVersion: apps.open-cluster-management.io/v1
kind: Deployable
metadata:
  name: deployable1
  namespace: default
  spec:
    template:
    placement:
    overrides:
    dependencies:
    channels:
    - mychannel
    - otherchannel
```

- チャネル定義を更新して、チャネルに追加する deployable を特定します。**spec.package** と **spec.packageFilter** フィールドを使用して、deployable を指定します。
  たとえば、以下のチャネルは、新規チャートがチャネルのソース Helm リポジトリーに公開されると、最新または最近の **nginx** チャートを自動的にプルします。チャート deployable には、チャネルにプロモートするバージョンと同じものを使用します (**1.x**)。

```
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/
---
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
```

- サブスクリプション定義を更新して、deployable を特定します。Deployable のチャネルへのプロモート設定は、サブスクリプション定義内で指定することもできます。

以下のサブスクリプションの例では、**nginx** の最新バージョン **1.x** のチャートが、対象のサブスクリプションでデプロイできるように、チャネルにプロモートされることを示します。

```
```yaml
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: mydevsub
  namespace: myspace
spec:
 source: https://kubernetes-charts.storage.googleapis.com/
 package: nginx
 packageFilter:
   version: 1.x
 placement:
   clusters:
   - name: mydevcluster1
```
```

- チャネル定義を更新してチャネルゲート要件を指定し、deployable の定義を更新して、ゲート要件を満たすフィールドおよび値を追加します。
  チャネルゲートの要件は、チャネル定義の **spec.gate** セクションで定義します。Deployable に、チャネル **spec.gate** 値に一致するフィールドがある場合は、deployable はチャネルにプロモートされます。このような場合に、deployable は **spec.channels** フィールドで特定のチャネルを参照する必要はありません。

  上記の例では、**packageFilter.version: "1.36.x"** は、サブスクリプションをデプロイメントするためにチャネルを介して特定の **nginx** バージョン **1.36.x** のチャートがプロモートされることを示しています。

1.1.6.4.5.1. 段階的ロールアウトを使用したデプロイ

すべてのターゲットクラスターにデプロイするのではなく、ターゲットのマネージドクラスターにデプロイメントをロールアウトする場合には、一度にデプロイするマネージドクラスターの割合だけに、deployable またはチャートのデプロイメントを設定できます。

たとえば、一度にすべてのクラスターに影響を与えずに更新をデプロイする必要がある場合に、デプロイメントをロールアウトできます。クラスターでデプロイメントが正常に行われると、デプロイメントは別のクラスターにロールアウトされます。

1.1.6.5. アプリケーションリソースの例

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。Red Hat Advanced Cluster Management for Kubernetes アプリケーションの基本的なリソースは、**application** リソースと **deployable** リソースです。

さらに、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には、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リソースの例を確認します。

- 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

1.1.6.5.1. 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。詳細は、「チャネルの作成および管理」を参照してください。

1.1.6.5.1.1. 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:
```

1.1.6.5.1.2. 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。**Namespace** チャネルの場合は、namespace はハブクラスターに配置する必要があります。
spec.type	必須。チャネルタイプ。サポート対象のタイプ: **Namespace**、**HelmRepo**、**Git**、および **ObjectBucket**
spec.pathname	**HelmRepo**、**Git**、**ObjectBucket**、**Namespace** 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アの 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。**Namespace** チャネルの場合は、チャネルが含まれる namespace に値を設定します。
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、**Namespace** および **ObjectBucket** チャネルタイプだけに適用され、**HelmRepo** や **Git** チャネルタイプには適用されません。
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。

フィールド	説明
spec.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/
```

1.1.6.5.1.3. Kubernetes namespace (Namespace) チャネル

チャネル namespace 内で deployable リソースを手作業で作成する必要があります。

deployable リソースを正しく作成するには、deployable に必要な以下のラベル 2 つをサブスクリプションコントローラーに追加して、このコントローラーで追加する deployable リソースを特定します。

```
labels:
    apps.open-cluster-management.io/channel: <channel name>
    apps.open-cluster-management.io/channel-type: Namespace
```

各 deployable の **spec.template.metadata.namespace** でテンプレートの namespace を指定しないでください。

namespace タイプのチャネルおよびサブスクリプションの場合は、deployable テンプレートがすべてマネージドクラスターのサブスクリプション namespace にデプロイされます。そのため、サブスクリプション namespace 以外で定義される deployable テンプレートは省略されます。

以下のチャネル定義の例は、deployable リソースを保持するチャネルとして namespace を抽象化します。この YAML が適用されると、namespace **ch-qa** は、**qa** という名前のチャネル用に作成されます。namespace ch-qa が作成されると、このチャネルは、deployable を特定するためにソースのデフォルト namespace を参照します。チャネルコントローラーは実際の namespace の場所にあるリソースを管理し、リソースが最新の状態に保たれるようにします。

```
  apiVersion: apps.open-cluster-management.io/v1
  kind: Channel
  metadata:
    name: qa
    namespace: ch-qa
  spec:
    sourceNamespaces:
    - default
    type: Namespace
    pathname: ch-qa
```

```
    gates:
      annotations:
        dev-ready: approved
```

```
apiVersion: apps.open-cluster-management.io/v1
kind: Deployable
metadata:
  labels:
    app: gbchn
    apps.open-cluster-management.io/channel: gbchn
    apps.open-cluster-management.io/channel-type: Namespace
    release: gbchn
  name: gbchn-service
  namespace: gbchn
spec:
  template:
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        app: gbchn
        release: gbchn
      name: gbchn
    spec:
      ports:
      - port: 80
      selector:
        app: gbchn
```

1.1.6.5.1.4. オブジェクトストアバケット (ObjectBucket) チャネル

以下のチャネル定義例では、オブジェクトストア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
```

1.1.6.5.1.5. 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。

```
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.
      configRef:
        name: insecure-skip-verify
      type: HelmRepo
  ---
  apiVersion: v1
  data:
    insecureSkipVerify: "true"
  kind: ConfigMap
  metadata:
    name: insecure-skip-verify
    namespace: hub-repo
```

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/
```

1.1.6.5.1.6. 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/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
```

1.1.6.5.1.7. 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。シークレットの詳細は、「シークレットの管理」を参照してください。

シークレット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。

1.1.6.5.1.7.1. 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
```

1.1.6.5.2. 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。詳細は、「サブスクリプションの作成および管理」を参照してください。

1.1.6.5.2.1. 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:
```

1.1.6.5.2.2. 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。	

フィールド	説明	
spec.channel	任意。サブスクリプションのチャネルを定義する NamespaceName ("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、**Namespace** と **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。	

フィールド	説明	
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。詳細については、「パッケージの上書きの設定」を参照してください。	
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指定します。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。	
spec.local	任意。ただし、直接管理するスタンドアロン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に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しない場合は、**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には、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。	

フィールド	説明	
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** 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** 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** 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。	
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。	

フィールド	説明	
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。	

フィールド	説明	
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。	
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。
- アノテーションは、**Namespace** タイプのチャネルが deployable のバージョンを検索するときにサブスクリプション Operator が使用します。サブスクリプション Operator は、バージョンを検索し、適切な deployable バージョンを検索して取得します。チャネルが **Namespace** チャネルである場合は、deployable のバージョンを識別するアノテーションを追加してください。
- 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 リリースまたは deployable の配置を定義することもできます。deployable の定義と同様に、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。**clusters**、**clusterSelector** または **placementRef** から複数を追加した場合には、サブスクリプション operator が使用する設定を判断する際に、以下の優先順位が使用されます。
  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: # Placement rules help you facilitate multi-cluster deployments, see placement rules documentation.
    placementRef:
      kind: PlacementRule
      name: towhichcluster
  overrides: # See Deployable documentation for more about overrides. Include overrides for any single cluster than requires some different settings
  - clusterName: "/"
    clusterOverrides:
    - path: "metadata.namespace"
      value: default
```

1.1.6.5.2.3. 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
```

1.1.6.5.2.3.1. 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"
```

1.1.6.5.2.3.2. 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、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
```

1.1.6.5.2.3.3. 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
```

1.1.6.5.2.3.4. Git リポジトリーのサブスクリプションの例

1.1.6.5.2.3.4.1. 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。

1.1.6.5.2.3.4.2. .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。

1.1.6.5.2.3.4.3. 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。

1.1.6.5.2.3.4.4. 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。

1.1.6.5.2.3.4.4.1. 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:

```
https://<externally-reachable hostname>/webhook`
```

1.1.6.5.2.3.4.4.2. 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。

1.1.6.5.2.3.4.4.3. 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。

1.1.6.5.2.3.4.4.4. 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>"
```

1.1.6.5.2.3.4.4.5. 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。

1.1.6.5.3. 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。

1.1.6.5.3.1. 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:
```

1.1.6.5.3.2. 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。

フィールド	説明
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。
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。

1.1.6.5.3.3. 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。

フィールド	説明
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: 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
```

1.1.6.5.4. 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。

アプリケーションの作成および管理の詳細は、「アプリケーションリソースの作成および管理」を参照してください。

1.1.6.5.4.1. 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:
  resourceVersion:
  annotations:
  labels:
    app:
    chart:
    heritage:
    name:
    release:
spec:
  componentKinds:
  - group:
    kind:
  descriptor:
  selector:
    matchExpressions:
    - key:
      operator:
      values:
```

1.1.6.5.4.2. アプリケーションの YAML 表

フィールド	説明
apiVersion	必須。この値は **app.k8s.io/v1beta1** に設定します。
kind	必須。この値は **Application** に設定して、リソースがアプリケーションリソースであることを指定します。
metadata.name	必須。アプリケーションリソースを識別する名前。

フィールド	説明
metadata.namespace	アプリケーションに使用する namespace リソース。
metadata.resourceVersion	アプリケーションリソースのバージョン。
metadata.annotations	任意。アプリケーションのアノテーション。
metadata.labels	任意。deployable のラベル。
spec.componentKinds	任意。アプリケーションに関連付けられるリソースの kind の一覧です。
spec.selector.matchExpressions	任意。アプリケーションと他の Kubernetesリソースのを関連付けるためのラベルセレクター。
spec.selector.matchExpressions.key	ラベルセレクターを定義する場合には必須です。リソースと一致する必要のある Kubernetes ラベルキー。
spec.selector.matchExpressions.values	ラベルセレクターを定義する場合には必須です。リソースと一致する必要のある Kubernetes ラベルの値。

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。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。

1.1.6.5.4.3. 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:
  labels:
    app: nginx-app-details
  name: nginx-app-3
  namespace: ns-sub-1
spec:
  componentKinds:
  - group: apps.open-cluster-management.io
    kind: Subscription
  selector:
    matchLabels:
      app: nginx-app-details
status: {}
```