# OpenShift Dedicated 4

## Operator

OpenShift Dedicated の Operator

Last Updated: 2025-12-01

# OpenShift Dedicated 4 Operator

OpenShift Dedicated の Operator

## Legal Notice



## Abstract

コントロールプレーンでのサービスのパッケージ化、デプロイメント、管理に Operator がどのように役立つかを説明します。

# Table of Contents

# 第1章 OPERATOR の概要

Operator は OpenShift Dedicated の最も重要なコンポーネントです。これらは、コントロールプレーン上でサービスをパッケージ化、デプロイ、および管理するための推奨される方法です。Operator の使用は、ユーザーが実行するアプリケーションにも各種の利点があります。

Operators は、**kubectl** や OpenShift CLI (**oc**) などの Kubernetes API および CLI ツールと統合します。Operator はアプリケーションの監視、ヘルスチェックの実行、OTA (over-the-air) 更新の管理を実行し、アプリケーションが指定した状態にあることを確認するための手段となります。

Operators は、Kubernetes ネイティブアプリケーション向けに特別に設計されており、インストールや設定などの一般的な Day 1 オペレーションを実装および自動化します。Operators は、自動スケールアップや自動スケールダウン、バックアップの作成など、Day 2 オペレーションを自動化することもできます。これらのアクティビティーは、クラスター上で実行されているソフトウェアによってすべて制御されます。

OpenShift Dedicated の Operator は、目的に応じて 2 つの異なるシステムによって管理されます。どちらも同様の Operator の概念と目標に準拠しています。

**クラスター Operator**

Cluster Version Operator (CVO) により管理され、クラスター機能を実行するためにデフォルトでインストールされます。

**オプションのアドオン Operators**

Operator Lifecycle Manager (OLM) により管理され、ユーザーがアプリケーションで実行するようにアクセスを可能にします。**OLM ベースの Operators** とも呼ばれます。

## 1.1. 開発者の場合

Operator 作成者は、OLM ベースの Operators に関して、次の開発タスクを実行できます。

- インストールされた Operator から Web コンソールを介してアプリケーションを作成 します。

## 1.2. 管理者の場合

**dedicated-admin** ロールを持つ管理者は、次の Operator タスクを実行できます。

- カスタムカタログを管理 します。
- ソフトウェアカタログから Operator をインストール します。
- オペレータのステータスを表示 します。
- Operator の状態を管理 します。
- インストールされている Operators をアップグレード します。
- インストールされている Operators を削除 します。
- プロキシーサポートを設定 します。

## 1.3. 次のステップ

- Operators について

# 第2章 OPERATORS について

## 2.1. OPERATORS について

概念的に言うと、**Operators** は人間の運用上のナレッジを使用し、これをコンシューマーと簡単に共有できるソフトウェアにエンコードします。

Operator は、ソフトウェアの他の部分を実行する運用上の複雑さを軽減するソフトウェアの特定の部分で設定されます。Operator はソフトウェアベンダーのエンジニアリングチームの一員のように機能し、Kubernetes 環境 (OpenShift Dedicated など) を監視し、その現在の状態を使用してリアルタイムで意思決定を行います。高度な Operator はアップグレードをシームレスに実行し、障害に自動的に対応するように設計されており、時間の節約のためにソフトウェアのバックアッププロセスを省略するなどのショートカットを実行することはありません。

技術的に言うと、Operators は Kubernetes アプリケーションをパッケージ化し、デプロイし、管理する方法です。

Kubernetes アプリケーションは、Kubernetes にデプロイされ、Kubernetes API および **kubectl** または **oc** ツールを使用して管理されるアプリケーションです。Kubernetes を最大限に活用するには、Kubernetes 上で実行されるアプリケーションを提供し、管理するために拡張できるように一連の総合的な API が必要です。Operators は、Kubernetes 上でこのタイプのアプリケーションを管理するランタイムと見なすことができます。

### 2.1.1. Operators を使用する理由

Operators は以下を提供します。

- インストールおよびアップグレードの反復性。
- すべてのシステムコンポーネントの継続的なヘルスチェック。
- OpenShift コンポーネントおよび ISV コンテンツの OTA (Over-the-air) 更新。
- フィールドエンジニアの知識を凝縮し、1 人や 2 人だけでなくすべてのユーザーに広める場所。

**Kubernetes にデプロイする理由**

Kubernetes (延長線上で考えると OpenShift Dedicated も含まれる) には、シークレットの処理、負荷分散、サービスの検出、自動スケーリングなどの、オンプレミスおよびクラウドプロバイダーで機能する、複雑な分散システムをビルドするために必要なすべてのプリミティブが含まれます。

**アプリケーションを Kubernetes API および kubectl ツールで管理する理由**

これらの API は機能的に充実しており、すべてのプラットフォームのクライアントを持ち、クラスターのアクセス制御/監査機能にプラグインします。Operator は Kubernetes の拡張メカニズム、カスタムリソース定義 (CRD、Custom Resource Definition ) を使用するため、**MongoDB** など のカスタムオブジェクトは、ビルトインされたネイティブ Kubernetes オブジェクトのように表示され、機能します。

**Operators とサービスブローカーとの比較**

サービスブローカーは、アプリケーションのプログラムによる検出およびデプロイメントを行うための 1 つの手段です。ただし、これは長期的に実行されるプロセスではないため、アップグレード、フェイルオーバー、またはスケーリングなどの Day 2 オペレーションを実行できません。カスタマイズおよびチューニング可能なパラメーターはインストール時に提供されるのに対し、Operator はクラスターの最新の状態を常に監視します。クラスター外のサービスを使用する場合は、Operators もこれらのクラスター外のサービスに使用できますが、これらをサービスブローカーで使用できます。

### 2.1.2. Operator Framework

Operator Framework は、上記のカスタマーエクスペリエンスに関連して提供されるツールおよび機能のファミリーです。これは、コードを作成するためだけにあるのではなく、Operators のテスト、実行、および更新などの重要な機能を実行します。Operator Framework コンポーネントは、これらの課題に対応するためのオープンソースツールで構成されています。

**Operator Lifecycle Manager**

Operator Lifecycle Manager (OLM) は、クラスター内の Operators のインストール、アップグレード、ロールベースのアクセス制御 (RBAC) を制御します。OpenShift Dedicated ではデフォルトでデプロイされます。

**Operator Registry**

Operator Registry は、クラスターで作成するためのクラスターサービスバージョン (Cluster Service Version、CSV) およびカスタムリソース定義 (CRD) を保存し、パッケージおよびチャネルに関する Operator メタデータを保存します。これは Kubernetes または OpenShift クラスターで実行され、この Operator カタログデータを OLM に指定します。

**ソフトウェアカタログ**

ソフトウェアカタログは、クラスター管理者がクラスターにインストールする Operator を検索して選択するための Web コンソールです。OpenShift Dedicated ではデフォルトでデプロイされます。

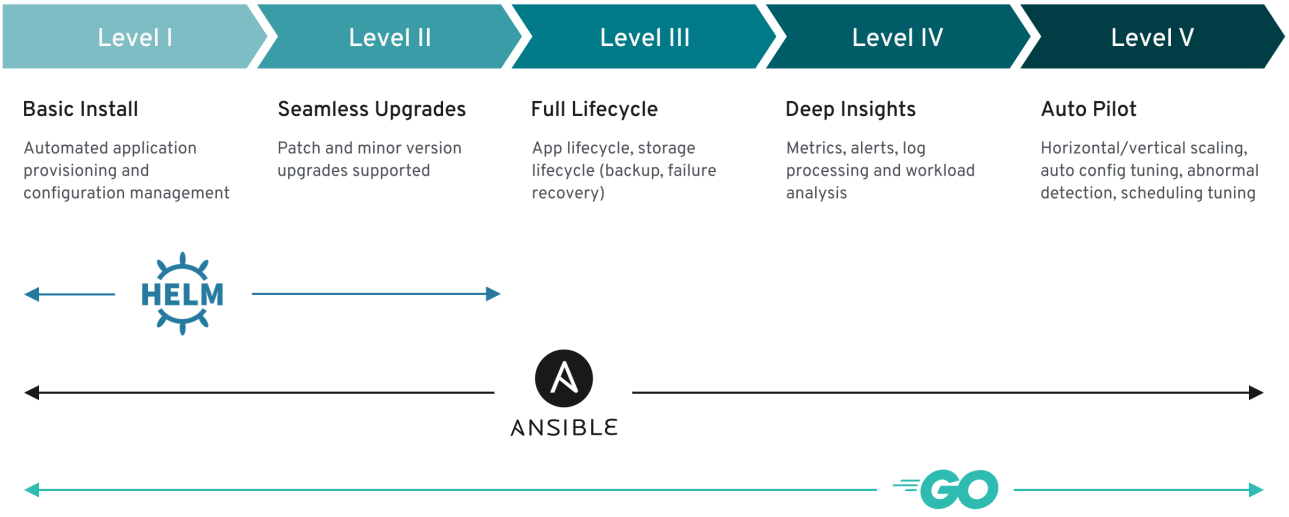
これらのツールは組み立て可能なツールとして設計されているため、役に立つと思われるツールを使用できます。

### 2.1.3. Operator 成熟度モデル

Operator 内にカプセル化されている管理ロジックの複雑さのレベルはさまざまです。また、このロジックは通常 Operator によって表されるサービスのタイプによって大きく変わります。

ただし、大半の Operators に含まれる特定の機能セットは、Operator のカプセル化された操作の成熟度の規模を一般化することができます。このため、以下の Operator 成熟度モデルは、Operator の一般的な Day 2 オペレーションに関する 5 つのフェーズの成熟度を定義しています。

**図2.1 Operator 成熟度モデル**




## 2.2. OPERATOR FRAMEWORK パッケージ形式

ここでは、OpenShift Dedicated の Operator Lifecycle Manager (OLM) によってサポートされる Operator のパッケージ形式の概要を説明します。

### 2.2.1. Bundle Format

Operators の **Bundle Format** は、Operator Framework によって導入されるパッケージ形式です。スケーラビリティーを向上させ、アップストリームユーザーがより効果的に独自のカタログをホストできるようにするために、Bundle Format 仕様は Operator メタデータのディストリビューションを単純化します。

Operator バンドルは、Operator の単一バージョンを表します。ディスク上の **バンドルマニフェスト** は、Kubernetes マニフェストおよび Operator メタデータを保存する実行不可能なコンテナーイメージである **バンドルイメージ** としてコンテナー化され、提供されます。次に、バンドルイメージの保存および配布は、**podman**、**docker**、および Quay などのコンテナーレジストリーを使用して管理されます。

Operator メタデータには以下を含めることができます。

- Operator を識別する情報 (名前およびバージョンなど)。
- UI を駆動する追加情報 (アイコンや一部のカスタムリソース (CR) など)。
- 必須および提供される API。
- 関連するイメージ。

マニフェストを Operator Registry データベースに読み込む際に、以下の要件が検証されます。

- バンドルには、アノテーションで定義された 1 つ以上のチャネルが含まれる必要がある。
- すべてのバンドルには、1 つのクラスターサービスバージョン (CSV) がある。
- CSV がクラスターリソース定義 (CRD) を所有する場合、その CRD はバンドルに存在する必要がある。

#### 2.2.1.1. マニフェスト

バンドルマニフェストは、Operator のデプロイメントおよび RBAC モデルを定義する Kubernetes マニフェストのセットを指します。

バンドルには、ディレクトリーごとに 1 つの CSV が含まれています。通常、その **/manifests** ディレクトリーには、CSV が所有する API を定義する CRD が格納されています。

**Bundle Format のレイアウトの例**

```
etcd
 ├──── manifests
 │     ├──── etcdcluster.crd.yaml
 │     └──── etcdoperator.clusterserviceversion.yaml
 │     └──── secret.yaml
 │     └──── configmap.yaml
 └──── metadata
       └──── annotations.yaml
       └──── dependencies.yaml
```

2.2.1.1.1. その他のサポート対象のオブジェクト

以下のオブジェクトタイプは、バンドルの **/manifests** ディレクトリーにオプションとして追加することもできます。

**サポート対象のオプションオブジェクトタイプ**

- **ClusterRole**
- **ClusterRoleBinding**
- **ConfigMap**
- **ConsoleCLIDownload**
- **ConsoleLink**
- **ConsoleQuickStart**
- **ConsoleYamlSample**
- **PodDisruptionBudget**
- **PriorityClass**
- **PrometheusRule**
- **Role**
- **RoleBinding**
- **Secret**
- **Service**
- **ServiceAccount**
- **ServiceMonitor**
- **VerticalPodAutoscaler**

これらのオプションオブジェクトがバンドルに含まれる場合、Operator Lifecycle Manager (OLM) はバンドルからこれらを作成し、CSV と共にそれらのライフサイクルを管理できます。

**オプションオブジェクトのライフサイクル**

- CSV が削除されると、OLM はオプションオブジェクトを削除します。
- CSV がアップグレードされると、以下を実行します。
  - オプションオブジェクトの名前が同じである場合、OLM はこれを更新します。
  - オプションオブジェクトの名前がバージョン間で変更された場合、OLM はこれを削除し、再作成します。

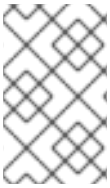
2.2.1.2. アノテーション

バンドルには、/**metadata** ディレクトリーに **annotations.yaml** ファイルも含まれています。このファイルは、バンドルをバンドルのインデックスに追加する方法に関する形式およびパッケージ情報の記述に役立つ高レベルの集計データを定義します。

**annotations.yaml の例**

```
annotations:
  operators.operatorframework.io.bundle.mediatype.v1: "registry+v1" 1
  operators.operatorframework.io.bundle.manifests.v1: "manifests/" 2
  operators.operatorframework.io.bundle.metadata.v1: "metadata/" 3
  operators.operatorframework.io.bundle.package.v1: "test-operator" 4
  operators.operatorframework.io.bundle.channels.v1: "beta,stable" 5
  operators.operatorframework.io.bundle.channel.default.v1: "stable" 6
```

1. Operator バンドルのメディアタイプまたは形式。**registry+v1** 形式の場合、これに CSV および関連付けられた Kubernetes オブジェクトが含まれることを意味します。
2. Operator マニフェストが含まれるディレクトリーへのイメージのパス。このラベルは今後使用するために予約され、現時点ではデフォの **manifests/** に設定されています。**manifests.v1** の値は、バンドルに Operator マニフェストが含まれることを示します。
3. バンドルに関するメタデータファイルが含まれるディレクトリーへのイメージのパス。このラベルは今後使用するために予約され、現時点ではデフォの **metadata/** に設定されています。**metadata.v1** の値は、このバンドルに Operator メタデータがあることを意味します。
4. バンドルのパッケージ名。
5. Operator Registry に追加される際にバンドルがサブスクライブするチャネルのリスト。
6. レジストリーからインストールされる場合に Operator がサブスクライブされるデフォルトチャネル。



**注記**

一致しない場合、**annotations.yaml** ファイルは、これらのアノテーションに依存するクラスター上の Operator Registry のみがこのファイルにアクセスできるために権威を持つファイルになります。

#### 2.2.1.3. Dependencies

Operator の依存関係は、バンドルの **metadata/** フォルダー内の **dependencies.yaml** ファイルに一覧表示されます。このファイルはオプションであり、現時点では明示的な Operator バージョンの依存関係を指定するためにのみ使用されます。

依存関係の一覧には、依存関係の内容を指定するために各項目の **type** フィールドが含まれます。次のタイプの Operator 依存関係がサポートされています。

**olm.package**

このタイプは、特定の Operator バージョンの依存関係であることを意味します。依存関係情報には、パッケージ名とパッケージのバージョンを semver 形式で含める必要があります。たとえば、**0.5.2** などの特定バージョンや **>0.5.1** などのバージョンの範囲を指定することができます。

**olm.gvk**

このタイプの場合、作成者は CSV の既存の CRD および API ベースの使用方法と同様に group/version/kind (GVK) 情報で依存関係を指定できます。これは、Operator の作成者がすべての依存関係、API または明示的なバージョンを同じ場所に配置できるようにするパスです。

**olm.constraint**

このタイプは、任意の Operator プロパティーに対するジェネリック制約を宣言します。

以下の例では、依存関係は Prometheus Operator および etcd CRD に指定されます。

**dependencies.yaml ファイルの例**

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
      group: etcd.database.coreos.com
      kind: EtcdCluster
      version: v1beta2
```

**関連情報**

- Operator Lifecycle Manager の依存関係の解決

#### 2.2.1.4. opm CLI について

**opm** CLI ツールは、Operator Bundle Format で使用するために Operator Framework によって提供されます。このツールを使用して、ソフトウェアリポジトリーに相当する Operator バンドルのリストから Operators のカタログを作成し、維持することができます。結果として、コンテナーイメージをコンテナーレジストリーに保存し、その後にクラスターにインストールできます。

カタログには、コンテナーイメージの実行時に提供される組み込まれた API を使用してクエリーできる、Operator マニフェストコンテンツへのポインターのデータベースが含まれます。OpenShift Dedicated では、Operator Lifecycle Manager (OLM) は、**CatalogSource** オブジェクトが定義したカタログソース内のイメージ参照できます。これにより、クラスター上にインストールされた Operator への頻度の高い更新を可能にするためにイメージを一定の間隔でポーリングできます。

- **opm** CLI のインストール手順は、CLI ツール を参照してください。

### 2.2.2. 主な特徴

**ファイルベースのカタログ** は、Operator Lifecycle Manager (OLM) の最新バージョンのカタログ形式です。この形式は、プレーンテキストベース (JSON または YAML) であり、以前の SQLite データベース形式の宣言的な設定の進化であり、完全な下位互換性があります。この形式の目標は、Operator のカタログ編集、設定可能性、および拡張性を有効にすることです。

**編集**

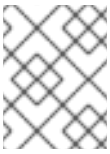
ファイルベースのカタログを使用すると、カタログの内容を操作するユーザーは、形式を直接変更し、変更が有効であることを確認できます。この形式はプレーンテキストの JSON または YAML であるため、カタログメンテナーは、一般的に知られている、サポート対象の JSON または YAML ツール (例: **jq** CLI) を使用して、手動でカタログメタデータを簡単に操作できます。

この編集機能により、以下の機能とユーザー定義の拡張が有効になります。

- 既存のバンドルの新規チャネルへのプロモート
- パッケージのデフォルトチャネルの変更
- アップグレードパスを追加、更新、および削除するためのカスタムアルゴリズム

**コンポーザービリティー**

ファイルベースのカタログは、任意のディレクトリー階層に保管され、カタログの作成が可能になります。たとえば、2 つのファイルベースのカタログディレクトリー (**catalogA** および **catalogB** ) を見てみましょう。カタログメンテナーは、新規のディレクトリー **catalogC** を作成して **catalogA** と **catalogB** をそのディレクトリーにコピーし、新しく結合カタログを作成できます。
このコンポーザービリティーにより、カタログの分散化が可能になります。この形式により、Operator の作成者は Operator 固有のカタログを維持でき、メンテナーは個別の Operator カタログで構成されるカタログを簡単にビルドできます。ファイルベースのカタログは、他の複数のカタログを組み合わせたり、1 つのカタログのサブセットを抽出したり、またはこれらの両方を組み合わせたりすることで作成できます。



**注記**

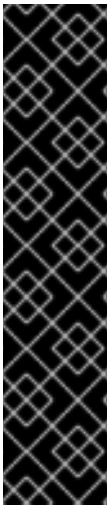
パッケージ内でパッケージおよびバンドルを重複できません。**opm validate** コマンドは、重複が見つかった場合はエラーを返します。

Operator の作成者は Operator、その依存関係およびそのアップグレードの互換性を最も理解しているので、Operator 固有のカタログを独自のカタログに維持し、そのコンテンツを直接制御できます。ファイルベースのカタログの場合に、Operator の作成者はカタログでパッケージをビルドして維持するタスクを所有します。ただし、複合カタログメンテナーは、カタログ内のパッケージのキュレートおよびユーザーにカタログを公開するタスクのみを所有します。

**拡張性**

ファイルベースのカタログ仕様は、カタログの低レベル表現です。これは低レベルの形式で直接保守できますが、カタログメンテナーは、このレベルの上に任意の拡張をビルドして、独自のカスタムツールを使用して任意数の変更を加えることができます。
たとえば、**(mode=semver)** などの高レベルの API を、アップグレードパス用に低レベルのファイルベースのカタログ形式にツールで変換できます。または、カタログメンテナーは、特定の条件を満たすバンドルに新規プロパティーを追加して、すべてのバンドルメタデータをカスタマイズする必要がある場合があります。

このような拡張性を使用すると、今後の OpenShift Dedicated リリース向けに、追加の正式なツールを低レベル API 上で開発できます。主な利点としては、カタログメンテナーもこの機能を利用できる点が挙げられます。



**重要**

OpenShift Dedicated 4.11 以降、デフォルトの Red Hat 提供の Operator カタログはファイルベースのカタログ形式でリリースされます。OpenShift Dedicated 4.6 から 4.10 までの Red Hat が提供するデフォルトの Operator カタログは、非推奨の SQLite データベース形式でリリースされました。

**opm** サブコマンド、フラグ、および SQLite データベース形式に関連する機能も非推奨となり、今後のリリースで削除されます。機能は引き続きサポートされており、非推奨の SQLite データベース形式を使用するカタログに使用する必要があります。

**opm index prune** などの SQLite データベース形式を使用する **opm** サブコマンドおよびフラグの多くは、ファイルベースのカタログ形式では機能しません。ファイルベースのカタログを使用する方法の詳細は、カスタムカタログの管理 を参照してください。

#### 2.2.2.1. ディレクトリー構造

ファイルベースのカタログは、ディレクトリーベースのファイルシステムから保存してロードできます。**opm** CLI は、root ディレクトリーを元に、サブディレクトリーに再帰してカタログを読み込みます。CLI は、検出されるすべてのファイルの読み込みを試行し、エラーが発生した場合には失敗します。

**.gitignore** ファイルとパターンと優先順位が同じ **.indexignore** ファイルを使用して、カタログ以外のファイルを無視できます。

**例: .indexignore ファイル**

```
# Ignore everything except non-object .json and .yaml files
**/*
!*.json
!*.yaml
**/objects/*.json
**/objects/*.yaml
```

カタログメンテナーは、必要なレイアウトを柔軟に選択できますが、各パッケージのファイルベースのカタログ Blob は別々のサブディレクトリーに保管することを推奨します。個々のファイルは JSON または YAML のいずれかをしようしてください。カタログ内のすべてのファイルが同じ形式を使用する必要はありません。

**推奨される基本構造**

```
catalog
├── packageA
│   └── index.yaml
├── packageB
│   ├── .indexignore
│   ├── index.yaml
│   └── objects
│       └── packageB.v0.1.0.clusterserviceversion.yaml
└── packageC
    └── index.json
    └── deprecations.yaml
```

この推奨の構造には、ディレクトリー階層内の各サブディレクトリーは自己完結型のカタログであるという特性があるため、カタログの作成、検出、およびナビゲーションなどのファイルシステムの操作が

簡素化されます。このカタログは、親カタログのルートディレクトリーにコピーして親カタログに追加することもできます。

#### 2.2.2.2. スキーマ

ファイルベースのカタログは、任意のスキーマで拡張できる CUE 言語仕様 に基づく形式を使用します。以下の **_Meta** CUE スキーマは、すべてのファイルベースのカタログ Blob が順守する必要のある形式を定義します。

**_Meta スキーマ**

```
_Meta: {
  // schema is required and must be a non-empty string
  schema: string & !=""

  // package is optional, but if it's defined, it must be a non-empty string
  package?: string & !=""

  // properties is optional, but if it's defined, it must be a list of 0 or more properties
  properties?: [... #Property]
}

#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}
```

**注記**

この仕様にリストされている CUE スキーマは網羅されていると見なされます。**opm validate** コマンドには、CUE で簡潔に記述するのが困難または不可能な追加の検証が含まれます。

Operator Lifecycle Manager(OLM) カタログは、現時点で OLM の既存のパッケージおよびバンドルの概念に対応する 3 つのスキーマ (**olm.package**、**olm.channel** および **olm.bundle**) を使用します。

カタログの各 Operator パッケージには、**olm.package** Blob が 1 つ (少なくとも **olm.channel** Blob 1 つ、および 1 つ以上の **olm.bundle** Blob) が必要です。

**注記**

**olm.*** スキーマは OLM 定義スキーマ用に予約されています。カスタムスキーマには、所有しているドメインなど、一意の接頭辞を使用する必要があります。

##### 2.2.2.2.1. olm.package スキーマ

**olm.package** スキーマは Operator のパッケージレベルのメタデータを定義します。これには、名前、説明、デフォルトのチャネル、およびアイコンが含まれます。

**例2.1 olm.package スキーマ**

```
#Package: {
  schema: "olm.package"

  // Package name
  name: string & !=""

  // A description of the package
  description?: string

  // The package's default channel
  defaultChannel: string & !=""

  // An optional icon
  icon?: {
    base64data: string
    mediatype:  string
  }
}
```

#### 2.2.2.2.2. olm.channel スキーマ

**olm.channel** スキーマは、パッケージ内のチャネル、チャネルのメンバーであるバンドルエントリー、およびそのバンドルのアップグレードパスを定義します。

バンドルエントリーが複数の **olm.channel** Blob 内のエッジを表す場合、バンドルエントリーはチャネルごとに 1 つだけ指定できます。

エントリーの **replaces** 値が、このカタログにも別のカタログにも存在しない別のバンドル名を参照していても、有効とされます。ただし、他のすべてのチャネルの普遍条件に該当する必要があります (チャネルに複数のヘッドがない場合など)。

**例2.2 olm.channel スキーマ**

```
#Channel: {
  schema: "olm.channel"
  package: string & !=""
  name: string & !=""
  entries: [...#ChannelEntry]
}

#ChannelEntry: {
  // name is required. It is the name of an `olm.bundle` that
  // is present in the channel.
  name: string & !=""

  // replaces is optional. It is the name of bundle that is replaced
  // by this entry. It does not have to be present in the entry list.
  replaces?: string & !=""

  // skips is optional. It is a list of bundle names that are skipped by
  // this entry. The skipped bundles do not have to be present in the
  // entry list.
  skips?: [...string & !=""]
```

```
  // skipRange is optional. It is the semver range of bundle versions
  // that are skipped by this entry.
  skipRange?: string & !=""
}
```

**警告**

**skipRange** フィールドを使用すると、スキップされた Operator バージョンが更新グラフからプルーニングされ、ユーザーが **Subscription** オブジェクトの **spec.startingCSV** プロパティーを使用してそのバージョンをインストールできなくなります。

**skipRange** フィールドと **replaces** フィールドの両方を使用すると、以前にインストールしたバージョンをユーザーが将来インストールできるように維持しながら、Operator を段階的に更新できます。**replaces** フィールドが当該 Operator バージョンの直前のバージョンを参照していることを確認してください。

#### 2.2.2.2.3. olm.bundle スキーマ

**例2.3 olm.bundle スキーマ**

```
#Bundle: {
  schema: "olm.bundle"
  package: string & !=""
  name: string & !=""
  image: string & !=""
  properties: [...#Property]
  relatedImages?: [...#RelatedImage]
}

#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}

#RelatedImage: {
  // image is the image reference
  image: string & !=""

  // name is an optional descriptive name for an image that
  // helps identify its purpose in the context of the bundle
  name?: string & !=""
}
```

#### 2.2.2.2.4. olm.deprecations スキーマ

オプションの **olm.deprecations** スキーマは、カタログ内のパッケージ、バンドル、チャネルの非推奨情報を定義します。Operator の作成者は、このスキーマを使用して、サポートステータスや推奨アップグレードパスなど、Operators に関する関連メッセージを、カタログから Operators を実行しているユーザーに提供できます。

このスキーマが定義されると、OpenShift Dedicated Web コンソールには、ソフトウェアカタログのインストール前ページとインストール後ページの両方で、カスタムの非推奨メッセージを含む Operator の影響を受ける要素の警告バッジが表示されます。

**olm.deprecations** スキーマエントリーには、非推奨の範囲を示す次の **reference** タイプが 1 つ以上含まれています。Operator がインストールされると、指定されたメッセージが、関連する **Subscription** オブジェクトのステータス状況として表示されます。

**表2.1 非推奨の reference タイプ**

| 型 | スコープ | ステータス状況 |
|---|---|---|
| **olm.package** | パッケージ全体を表します。 | **PackageDeprecated** |
| **olm.channel** | 1 つのチャネルを表します。 | **ChannelDeprecated** |
| **olm.bundle** | 1 つのバンドルバージョンを表します。 | **BundleDeprecated** |

次の例で詳しく説明するように、各 **reference** タイプには独自の要件があります。

**例2.4 各 reference タイプを使用した olm.deprecations スキーマの例**

```
schema: olm.deprecations
package: my-operator (1)
entries:
  - reference:
      schema: olm.package (2)
    message: | (3)
    The 'my-operator' package is end of life. Please use the
    'my-operator-new' package for support.
  - reference:
      schema: olm.channel
      name: alpha (4)
    message: |
    The 'alpha' channel is no longer supported. Please switch to the
    'stable' channel.
  - reference:
      schema: olm.bundle
      name: my-operator.v1.68.0 (5)
    message: |
    my-operator.v1.68.0 is deprecated. Uninstall my-operator.v1.68.0 and
    install my-operator.v1.72.0 for support.
```

1. 各非推奨スキーマには **package** 値が必要であり、そのパッケージ参照はカタログ全体で一意である必要があります。関連する **name** フィールドを含めることはできません。

2 **olm.package** スキーマに **name** フィールドを含めることはできません。このフィールドは、スキーマ内で前に定義した **package** フィールドによって決定されるためです。

3 すべての **message** フィールドは、**reference** タイプを問わず、長さが 0 以外である必要があり、不透明なテキスト Blob として表す必要があります。

4 **olm.channel** スキーマの **name** フィールドは必須です。

5 **olm.bundle** スキーマの **name** フィールドは必須です。

**注記**

非推奨機能では、パッケージ、チャネル、バンドルなど、重複する非推奨は考慮されません。

Operator の作成者は、**olm.deprecations** スキーマエントリーを **deprecations.yaml** ファイルとしてパッケージの **index.yaml** ファイルと同じディレクトリーに保存できます。

**非推奨を含むカタログのディレクトリー構造の例**

```
my-catalog
└── my-operator
    ├── index.yaml
    └── deprecations.yaml
```

**関連情報**

- ファイルベースのカタログイメージの更新またはフィルタリング

#### 2.2.2.3. プロパティー

プロパティーは、ファイルベースのカタログスキーマに追加できる任意のメタデータです。**type** フィールドは、**value** フィールドのセマンティックおよび構文上の意味を効果的に指定する文字列です。値には任意の JSON または YAML を使用できます。

OLM は、予約済みの **olm.*** 接頭辞をもう一度使用して、いくつかのプロパティータイプを定義します。

##### 2.2.2.3.1. olm.package プロパティー

**olm.package** プロパティーは、パッケージ名とバージョンを定義します。これはバンドルの必須プロパティーであり、これらのプロパティーが 1 つ必要です。**packageName** フィールドはバンドルのファーストクラス **package** フィールドと同じでなければならず、**version** フィールドは有効なセマンティクスバージョンである必要があります。

**例2.5 olm.package プロパティー**

```
#PropertyPackage: {
  type: "olm.package"
  value: {
    packageName: string & !=""
```

```
    version: string & !=""
  }
}
```

#### 2.2.2.3.2. olm.gvk プロパティー

**olm.gvk** プロパティーは、このバンドルで提供される Kubernetes API の group/version/kind(GVK) を定義します。このプロパティーは、OLM が使用して、必須の API と同じ GVK をリストする他のバンドルの依存関係として、このプロパティーでバンドルを解決します。GVK は Kubernetes GVK の検証に準拠する必要があります。

**例2.6 olm.gvk プロパティー**

```
#PropertyGVK: {
  type: "olm.gvk"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

#### 2.2.2.3.3. olm.package.required

**olm.package.required** プロパティーは、このバンドルが必要な別のパッケージのパッケージ名とバージョン範囲を定義します。バンドルにリストされている必要なパッケージプロパティーごとに、OLM は、リストされているパッケージのクラスターに必要なバージョン範囲で Operator がインストールされていることを確認します。**versionRange** フィールドは有効なセマンティクスバージョン (semver) の範囲である必要があります。

**例2.7 olm.package.required プロパティー**

```
#PropertyPackageRequired: {
  type: "olm.package.required"
  value: {
    packageName: string & !=""
    versionRange: string & !=""
  }
}
```

#### 2.2.2.3.4. olm.gvk.required

**olm.gvk.required** プロパティーは、このバンドルが必要とする Kubernetes API の group/version/kind(GVK) を定義します。バンドルにリストされている必要な GVK プロパティーごとに、OLM は、提供する Operator がクラスターにインストールされていることを確認します。GVK は Kubernetes GVK の検証に準拠する必要があります。

**例2.8 olm.gvk.required プロパティー**

```
#PropertyGVKRequired: {
  type: "olm.gvk.required"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

#### 2.2.2.4. カタログの例

ファイルベースのカタログを使用すると、カタログメンテナーは Operator のキュレーションおよび互換性に集中できます。Operator の作成者は Operators 用に Operator 固有のカタログをすでに生成しているので、カタログメンテナーは、各 Operator カタログをカタログのルートディレクトリーのサブディレクトリーにレンダリングしてビルドできます。

ファイルベースのカタログをビルドする方法は多数あります。以下の手順は、単純なアプローチの概要を示しています。

1. カタログの設定ファイルを 1 つ維持し、カタログ内に Operator ごとにイメージの参照を含めます。

   **カタログ設定ファイルのサンプル**

   ```
   name: community-operators
   repo: quay.io/community-operators/catalog
   tag: latest
   references:
   - name: etcd-operator
     image: quay.io/etcd-
   operator/index@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f
   6be03
   - name: prometheus-operator
     image: quay.io/prometheus-
   operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d84101
   eb317
   ```

2. 設定ファイルを解析し、その参照から新規カタログを作成するスクリプトを実行します。

   **スクリプトの例**

   ```
   name=$(yq eval '.name' catalog.yaml)
   mkdir "$name"
   yq eval '.name + "/" + .references[].name' catalog.yaml | xargs mkdir
   for l in $(yq e '.name as $catalog | .references[] | .image + "|" + $catalog + "/" + .name +
   "/index.yaml"' catalog.yaml); do
     image=$(echo $l | cut -d'|' -f1)
     file=$(echo $l | cut -d'|' -f2)
     opm render "$image" > "$file"
   done
   opm generate dockerfile "$name"
   ```

```
indexImage=$(yq eval '.repo + ":" + .tag' catalog.yaml)
docker build -t "$indexImage" -f "$name.Dockerfile" .
docker push "$indexImage"
```

#### 2.2.2.5. ガイドライン

ファイルベースのカタログを維持する場合には、以下のガイドラインを考慮してください。

##### 2.2.2.5.1. イミュータブルなバンドル

Operator Lifecycle Manager(OLM) に関する一般的なアドバイスとして、バンドルイメージとそのメタデータをイミュータブルとして処理する必要がある点があります。

破損したバンドルがカタログにプッシュされている場合には、少なくとも 1 人のユーザーがそのバンドルにアップグレードしたと想定する必要があります。このような想定に基づいて、破損したバンドルをインストールしたユーザーがアップグレードを確実に受け取れるように、破損したバンドルのアップグレードパスを使用して別のバンドルをリリースする必要があります。OLM は、カタログでバンドルの内容が更新された場合に、インストールされたバンドルは再インストールされません。

ただし、カタログメタデータの変更が推奨される場合があります。

- チャネルプロモーション: バンドルをすでにリリースし、後で別のチャネルに追加することにした場合は、バンドルのエントリーを別の **olm.channel** Blob に追加できます。
- 新規アップグレードパス: **1.2.z** バンドルバージョンを新たにリリースしたが (例:**1.2.4**)、**1.3.0** がすでにリリースされている場合は、**1.2.4** をスキップするように **1.3.0** のカタログメタデータを更新できます。

##### 2.2.2.5.2. ソース制御

カタログメタデータはソースコントロールに保存され、信頼できる情報源として処理される必要があります。以下の手順で、カタログイメージを更新する必要があります。

1. ソース制御されたカタログディレクトリーを新規コミットを使用して更新します。
2. カタログイメージをビルドし、プッシュします。ユーザーがカタログが利用可能になり次第更新を受信できるように、一貫性のあるタグ付け (**:latest** or **:<target_cluster_version>**) を使用します。

#### 2.2.2.6. CLI の使用

**opm** CLI を使用してファイルベースのカタログを作成する方法は、カスタムカタログの管理 を参照してください。

ファイルベースのカタログの管理に関連する **opm** CLI コマンドに関する参考情報は、CLI ツール を参照してください。

#### 2.2.2.7. 自動化

Operator の作成者およびカタログメンテナーは、CI/CD ワークフローを使用してカタログのメンテナンスを自動化することが推奨されます。カタログメンテナーは、GitOps 自動化をビルドして以下のタスクを実行し、これをさらに向上させることができます。

- パッケージのイメージ参照の更新など、プル要求 (PR) の作成者が要求された変更を実行できることを確認します。

- カタログの更新で **opm validate** コマンドが指定されていることを確認します。
- 更新されたバンドルまたはカタログイメージの参照が存在し、カタログイメージがクラスターで正常に実行され、そのパッケージの Operators が正常にインストールされることを確認します。
- 以前のチェックに合格した PR を自動的にマージします。
- カタログイメージを自動的にもう一度ビルドして公開します。

## 2.3. 一般的な OPERATOR FRAMEWORK 用語

このトピックでは、Operator Lifecycle Manager (OLM) を含む Operator Framework に関連する一般的な用語の用語集を提供します。

### 2.3.1. バンドル

Bundle Format では、**バンドル** は Operator CSV、マニフェスト、およびメタデータのコレクションです。さらに、それらはクラスターにインストールできる一意のバージョンの Operator を形成します。

### 2.3.2. バンドルイメージ

Bundle Format では、**バンドルイメージ** は Operator マニフェストからビルドされ、1 つのバンドルが含まれるコンテナーイメージです。バンドルイメージは、Quay.io または DockerHub などの Open Container Initiative (OCI) 仕様コンテナーレジストリーによって保存され、配布されます。

### 2.3.3. カタログソース

**カタログソース** は、OLM が Operators およびそれらの依存関係を検出し、インストールするためにクエリーできるメタデータのストアを表します。

### 2.3.4. チャネル

**チャネル** は Operator の更新ストリームを定義し、サブスクライバーの更新をロールアウトするために使用されます。ヘッドはそのチャネルの最新バージョンを参照します。たとえば **stable** チャネルには、Operator のすべての安定したバージョンが最も古いものから最新のものへと編成されます。

Operator には複数のチャネルを含めることができ、特定のチャネルへのサブスクリプションのバインドはそのチャネル内の更新のみを検索します。

### 2.3.5. チャネルヘッド

**チャネルヘッド** は、特定のチャネル内の最新の既知の更新を指します。

### 2.3.6. クラスターサービスバージョン

**クラスターサービスバージョン (CSV)** は、クラスターでの Operator の実行に使用される Operator メタデータから作成される YAML マニフェストです。これは、ユーザーインターフェイスにロゴ、説明、およびバージョンなどの情報を設定するために使用される Operator コンテナーイメージに伴うメタデータです。

CSV は、Operator が必要とする RBAC ルールやそれが管理したり、依存したりするカスタムリソース (CR) などの Operator の実行に必要な技術情報の情報源でもあります。

### 2.3.7. 依存関係

Operator はクラスターに存在する別の Operator への **依存関係** を持つ場合があります。たとえば、Vault Operator にはそのデータ永続層に etcd Operator への依存関係があります。

OLM は、インストールフェーズで指定されたすべてのバージョンの Operators および CRD がクラスターにインストールされていることを確認して依存関係を解決します。この依存関係は、必要な CRD API を満たすカタログの Operator を検索し、インストールすることで解決され、パッケージまたはバンドルには関連しません。

### 2.3.8. 拡張機能

拡張機能により、クラスター管理者は OpenShift Dedicated クラスター上のユーザーの機能を拡張できます。拡張機能は Operator Lifecycle Manager (OLM) v1 によって管理されます。

**ClusterExtension** API は、ユーザー向け API を単一のオブジェクトに統合することで、**registry+v1** バンドル形式を使用した Operators を含むインストール済み拡張機能の管理を効率化します。管理者と SRE は、この API を使用してプロセスを自動化し、GitOps の原則に基づき望ましい状態を定義できます。

### 2.3.9. インデックスイメージ

Bundle Format で、**インデックスイメージ** は、すべてのバージョンの CSV および CRD を含む Operator バンドルに関する情報が含まれるデータベースのイメージ (データベーススナップショット) を指します。このインデックスは、クラスターで Operators の履歴をホストでき、**opm** CLI ツールを使用して Operators を追加または削除することで維持されます。

### 2.3.10. インストール計画

**インストール計画** は、CSV を自動的にインストールするか、アップグレードするために作成されるリソースの計算された一覧です。

### 2.3.11. マルチテナンシー

OpenShift Dedicated の **テナント** は、デプロイされた一連のワークロードに対する共通のアクセス権と権限を共有するユーザーまたはユーザーのグループであり、通常は namespace またはプロジェクトで表されます。テナントを使用して、異なるグループまたはチーム間に一定レベルの分離を提供できます。

クラスターが複数のユーザーまたはグループによって共有されている場合、**マルチテナント** クラスターと見なされます。

### 2.3.12. Operator

Operators は、Kubernetes アプリケーションをパッケージ化し、デプロイし、管理する方法です。Kubernetes アプリケーションは、Kubernetes にデプロイされ、Kubernetes API および **kubectl** または **oc** ツールを使用して管理されるアプリケーションです。

Operator Lifecycle Manager (OLM) v1 では、**ClusterExtension** API により、**registry+v1** バンドル形式を使用した Operators を含むインストール済み拡張機能の管理が効率化されます。

### 2.3.13. Operator グループ

**Operator グループ** は、**OperatorGroup** オブジェクトと同じ namespace にデプロイされたすべての Operators を、namespace のリストまたはクラスター全体でそれらの CR を監視できるように設定します。

### 2.3.14. Package

Bundle Format で、**パッケージ** は Operator のリリースされたすべての履歴をそれぞれのバージョンで囲むディレクトリーです。Operator のリリースされたバージョンは、CRD と共に CSV マニフェストに記述されます。

### 2.3.15. レジストリー

**レジストリー** は、Operators のバンドルイメージを保存するデータベースで、それぞれにすべてのチャネルの最新バージョンおよび過去のバージョンすべてが含まれます。

### 2.3.16. サブスクリプション

**サブスクリプション** は、パッケージのチャネルを追跡して CSV を最新の状態に保ちます。

### 2.3.17. 更新グラフ

**更新グラフ** は、他のパッケージ化されたソフトウェアの更新グラフと同様に、CSV の複数のバージョンを 1 つにまとめます。Operators を順番にインストールすることも、特定のバージョンを省略することもできます。更新グラフは、新しいバージョンが追加されている状態でヘッドでのみ拡張することが予想されます。

**更新エッジ** または **更新パス** とも呼ばれます。

## 2.4. OPERATOR LIFECYCLE MANAGER (OLM)

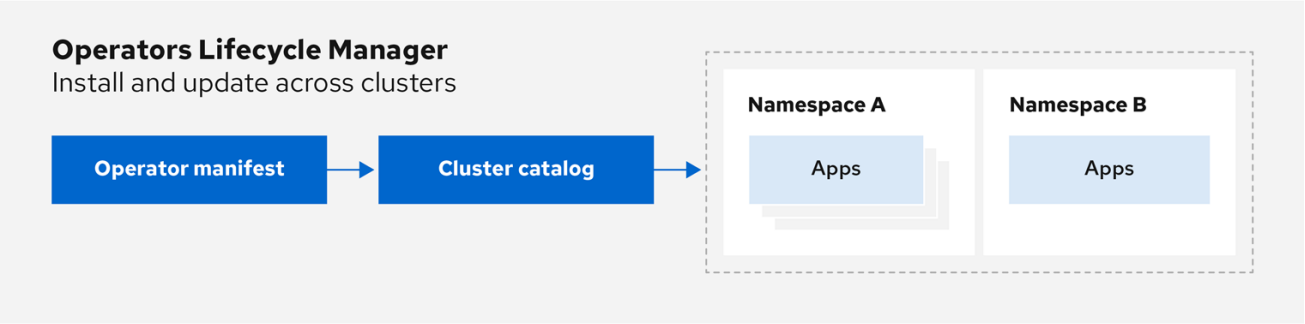
### 2.4.1. Operator Lifecycle Manager の概念およびリソース

このガイドでは、OpenShift Dedicated の Operator Lifecycle Manager (OLM) を支える概念の概要を説明します。

#### 2.4.1.1. Operator Lifecycle Manager (OLM) Classic とは

Operator Lifecycle Manager (OLM) Classic を使用することにより、ユーザーは Kubernetes ネイティブアプリケーション (Operator) および OpenShift Dedicated クラスター全体で実行される関連サービスに対してインストール、更新、およびそのライフサイクルの管理を実行できます。これは、Operator を効果的かつ自動化された拡張可能な方法で管理するために設計されたオープンソースツールキットの Operator Framework の一部です。

図2.2 OLM (Classic) ワークフロー




OLM は OpenShift Dedicated でデフォルトで実行されます。これは、**dedicated-admin** ロールを持つ管理者が、クラスターで実行されている Operator のインストール、アップグレード、アクセス付与を行う際に役立ちます。OpenShift Dedicated Web コンソールは、**dedicated-admin** 管理者が Operator をインストールしたり、クラスターで利用可能な Operator のカタログを使用できるように特定のプロジェクトアクセスを付与したりするのに使用する管理画面を提供します。

開発者の場合は、セルフサービスを使用することで、専門的な知識がなくてもデータベースのインスタンスのプロビジョニングや設定、またモニタリング、ビッグデータサービスなどを実行できます。Operator にそれらに関するナレッジが織り込まれているためです。

#### 2.4.1.2. OLM リソース

以下のカスタムリソース定義 (CRD) は Operator Lifecycle Manager (OLM) によって定義され、管理されます。

表2.2 OLM およびカタログ Operators で管理される CRD

| リソース | 短縮名 | 説明 |
|---|---|---|
| **ClusterServiceVersion** (CSV) | **csv** | アプリケーションメタデータ:例: 名前、バージョン、アイコン、必須リソース。 |
| **CatalogSource** | **catsrc** | CSV、CRD、およびアプリケーションを定義するパッケージのリポジトリー。 |
| **Subscription** | **sub** | パッケージのチャネルを追跡して CSV を最新の状態に保ちます。 |
| **InstallPlan** | **ip** | CSV を自動的にインストールするか、アップグレードするために作成されるリソースの計算された一覧。 |
| **OperatorGroup** | **og** | **OperatorGroup** オブジェクトと同じ namespace にデプロイされたすべての Operators を、namespace のリストまたはクラスター全体でカスタムリソース (CR) を監視できるように設定します。 |
| **OperatorConditions** | - | OLM とそれが管理する Operator との間で通信チャネルを作成します。Operators は **Status.Conditions** 配列に書き込みを行い、複雑な状態を OLM と通信できます。 |

#### 2.4.1.2.1. クラスターサービスバージョン

**クラスターサービスバージョン** (CSV) は、OpenShift Dedicated クラスター上で実行中の Operator の特定バージョンを表します。これは、クラスターでの Operator Lifecycle Manager (OLM) の Operator の実行に使用される Operator メタデータから作成される YAML マニフェストです。

OLM は Operator に関するこのメタデータを要求し、これがクラスターで安全に実行できるようにし、Operator の新規バージョンが公開される際に更新を適用する方法に関する情報を提供します。これは従来のオペレーティングシステムのソフトウェアのパッケージに似ています。OLM のパッケージ手順を、**rpm**、**deb**、または **apk** バンドルを作成するステージとして捉えることができます。

CSV には、ユーザーインターフェイスに名前、バージョン、説明、ラベル、リポジトリーリンクおよびロゴなどの情報を設定するために使用される Operator コンテナーイメージに伴うメタデータが含まれます。

CSV は、Operator が管理したり、依存したりするカスタムリソース (CR)、RBAC ルール、クラスター要件、およびインストールストラテジーなどの Operator の実行に必要な技術情報の情報源でもあります。この情報は OLM に対して必要なリソースの作成方法と、Operator をデプロイメントとしてセットアップする方法を指示します。

#### 2.4.1.2.2. カタログソース

**カタログソース** は、通常コンテナーレジストリーに保存されている **インデックスイメージ** を参照してメタデータのストアを表します。Operator Lifecycle Manager(OLM) はカタログソースをクエリーし、Operators およびそれらの依存関係を検出してインストールします。OpenShift Dedicated Web コンソールのソフトウェアカタログには、カタログソースによって提供される Operator も表示されます。

**ヒント**

クラスター管理者は、Web コンソールの **Administration → Cluster Settings → Configuration → OperatorHub** ページを使用して、クラスターで有効なログソースにより提供される Operators の詳細一覧を表示できます。

**CatalogSource** オブジェクトの **spec** は、Pod の構築方法、または Operator Registry gRPC API を提供するサービスとの通信方法を示します。

**例2.9 CatalogSource オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog 1
  namespace: openshift-marketplace 2
  annotations:
    olm.catalogImageTemplate: 3
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}.
{kube_patch_version}"
spec:
  displayName: Example Catalog 4
  image: quay.io/example-org/example-catalog:v1 5
  priority: -400 6
  publisher: Example Org
  sourceType: grpc 7
```

```
  grpcPodConfig:
    securityContextConfig: <security_mode> 8
    nodeSelector: 9
      custom_label: <label>
    priorityClassName: system-cluster-critical 10
    tolerations: 11
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
  updateStrategy:
    registryPoll: 12
      interval: 30m0s
status:
  connectionState:
    address: example-catalog.openshift-marketplace.svc:50051
    lastConnect: 2021-08-26T18:14:31Z
    lastObservedState: READY 13
  latestImageRegistryPoll: 2021-08-26T18:46:25Z 14
  registryService: 15
    createdAt: 2021-08-26T16:16:37Z
    port: 50051
    protocol: grpc
    serviceName: example-catalog
    serviceNamespace: openshift-marketplace
```

1. **CatalogSource** オブジェクトの名前。この値は、要求された namespace で作成される、関連の Pod 名の一部としても使用されます。

2. カタログを作成する namespace。カタログを全 namespace のクラスター全体で利用可能にするには、この値を **openshift-marketplace** に設定します。Red Hat が提供するデフォルトのカタログソースも **openshift-marketplace** namespace を使用します。それ以外の場合は、値を特定の namespace に設定し、Operator をその namespace でのみ利用可能にします。

3. 任意: クラスターのアップグレードにより、Operator のインストールがサポートされていない状態になったり、更新パスが継続されなかったりする可能性を回避するために、クラスターのアップグレードの一環として、Operator カタログのインデックスイメージのバージョンを自動的に変更するように有効化することができます。

   **olm.catalogImageTemplate** アノテーションをインデックスイメージ名に設定し、イメージタグのテンプレートを作成する際に、1 つ以上の Kubernetes クラスターバージョン変数を使用します。アノテーションは、実行時に **spec.image** フィールドを上書きします。詳細は、「カスタムカタログソースのイメージテンプレート」のセクションを参照してください。

4. Web コンソールおよび CLI でのカタログの表示名。

5. カタログのインデックスイメージ。オプションで、**olm.catalogImageTemplate** アノテーションを使用して実行時のプル仕様を設定する場合には、省略できます。

6. カタログソースの重み。OLM は重みを使用して依存関係の解決時に優先順位付けします。重みが大きい場合は、カタログが重みの小さいカタログよりも優先されることを示します。

7. ソースタイプには以下が含まれます。

- **image** 参照のある **grpc**: OLM はイメージをポーリングし、Pod を実行します。これにより、準拠 API が提供されることが予想されます。
- **address** フィールドのある **grpc**: OLM は所定アドレスでの gRPC API へのアクセスを試行します。これはほとんどの場合使用することができません。
- **configmap**: OLM は設定マップデータを解析し、gRPC API を提供できる Pod を実行します。

8 **legacy** または **restricted** の値を指定します。フィールドが設定されていない場合、デフォルト値は **legacy** です。今後の OpenShift Dedicated リリースでは、デフォルト値が **restricted** になる予定です。**restricted** 権限でカタログを実行できない場合は、このフィールドを手動で **legacy** に設定することを推奨します。

9 オプション: **grpc** タイプのカタログソースの場合は、**spec.image** でコンテンツを提供する Pod のデフォルトのノードセレクターをオーバーライドします (定義されている場合)。

10 オプション: **grpc** タイプのカタログソースの場合は、**spec.image** でコンテンツを提供する Pod のデフォルトの優先度クラス名をオーバーライドします (定義されている場合)。Kubernetes は、デフォルトで優先度クラス **system-cluster-critical** および **system-node-critical** を提供します。フィールドを空 (**""**) に設定すると、Pod にデフォルトの優先度が割り当てられます。他の優先度クラスは、手動で定義できます。

11 オプション: **grpc** タイプのカタログソースの場合は、**spec.image** でコンテンツを提供する Pod のデフォルトの Toleration をオーバーライドします (定義されている場合)。

12 最新の状態を維持するために、特定の間隔で新しいバージョンの有無を自動的にチェックします。

13 カタログ接続が最後に監視された状態。以下に例を示します。

- **READY**: 接続が正常に確立されました。
- **CONNECTING**: 接続が確立中です。
- **TRANSIENT_FAILURE**: タイムアウトなど、接続の確立時一時的な問題が発生しました。状態は最終的に **CONNECTING** に戻り、再試行されます。

詳細は、gRPC ドキュメントの 接続の状態 を参照してください。

14 カタログイメージを保存するコンテナーレジストリーがポーリングされ、イメージが最新の状態であることを確認します。

15 カタログの Operator Registry サービスのステータス情報。

サブスクリプションの **CatalogSource** オブジェクトの **name** を参照すると、要求された Operator を検索する場所を、OLM に指示します。

**例2.10 カタログソースを参照する Subscription オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
```

```
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

**関連情報**

- ソフトウェアカタログの概要
- Red Hat が提供する Operator カタログ
- クラスターへのカタログソースの追加
- カタログの優先順位
- CLI を使用した Operator カタログソースのステータス表示
- カタログソース Pod のスケジューリング

2.4.1.2.2.1. カスタムカタログソースのイメージテンプレート

基礎となるクラスターとの Operator との互換性は、さまざまな方法でカタログソースにより表現できます。1 つの方法は、特定のプラットフォームリリース (OpenShift Dedicated など) 用に特別に作成されたインデックスイメージのイメージタグを特定することです。この方法は、Red Hat 提供のデフォルトのカタログソースに使用されています。

クラスターのアップグレード時に、Red Hat が提供するデフォルトのカタログソースのインデックスイメージのタグは、Operator Lifecycle Manager (OLM) が最新版のカタログをプルするように、Cluster Version Operator (CVO) により自動更新されます。たとえば、OpenShift Dedicated 4.19 から 4 へのアップグレード時に、**redhat-operators** カタログの **CatalogSource** オブジェクトの **spec.image** フィールドは次のように更新されます。

```
registry.redhat.io/redhat/redhat-operator-index:v4.20
```

更新後は次のようになります。

```
registry.redhat.io/redhat/redhat-operator-index:v4.20
```

ただし、CVO ではカスタムカタログのイメージタグは自動更新されません。クラスターのアップグレード後、ユーザーが互換性があり、サポート対象の Operator のインストールを確実に行えるようにするには、カスタムカタログも更新して、更新されたインデックスイメージを参照する必要があります。

OpenShift Dedicated 4.9 以降、クラスター管理者は、カスタムカタログの **CatalogSource** オブジェクトの **olm.catalogImageTemplate** アノテーションを、テンプレートを含むイメージ参照に追加できます。以下の Kubernetes バージョン変数は、テンプレートで使用できるようにサポートされています。

- **kube_major_version**
- **kube_minor_version**
- **kube_patch_version**

**注記**

OpenShift Dedicated クラスターのバージョンではなく、Kubernetes クラスターのバージョンを指定する必要があります。OpenShift Dedicated クラスターのバージョンは、現在テンプレートに使用できないためです。

更新された Kubernetes バージョンを指定するタグでインデックスイメージを作成してプッシュしている場合に、このアノテーションを設定すると、カスタムカタログのインデックスイメージのバージョンがクラスターのアップグレード後に自動的に変更されます。アノテーションの値は、**CatalogSource** オブジェクトの **spec.image** フィールドでイメージ参照を設定したり、更新したりするために使用されます。こうすることで、サポートなしの状態や、継続する更新パスなしの状態で Operator がインストールされないようにします。

**重要**

格納されているレジストリーがどれであっても、クラスターのアップグレード時に、クラスターが、更新されたタグを含むインデックスイメージにアクセスできるようにする必要があります。

**例2.11 イメージテンプレートを含むカタログソースの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    olm.catalogImageTemplate:
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}"
spec:
  displayName: Example Catalog
  image: quay.io/example-org/example-catalog:v1.33
  priority: -400
  publisher: Example Org
```

**注記**

**spec.image** フィールドおよび **olm.catalogImageTemplate** アノテーションの両方が設定されている場合には、**spec.image** フィールドはアノテーションから解決された値で上書きされます。アノテーションが使用可能なプル仕様に対して解決されない場合は、カタログソースは **spec.image** 値にフォールバックします。

**spec.image** フィールドが設定されていない場合に、アノテーションが使用可能なプル仕様に対して解決されない場合は、OLM はカタログソースの調整を停止し、人間が判読できるエラー条件に設定します。

Kubernetes 1.33 を使用する OpenShift Dedicated クラスターの場合、前の例の **olm.catalogImageTemplate** アノテーションは次のイメージ参照に解決されます。

```
quay.io/example-org/example-catalog:v1.33
```

OpenShift Dedicated の今後のリリースに備えて、新しい OpenShift Dedicated バージョンで使用される新しい Kubernetes バージョンを対象とした、カスタムカタログの更新済みインデックスイメージを作成できます。アップグレード前に **olm.catalogImageTemplate** アノテーションを設定してから、クラスターを新しい OpenShift Dedicated バージョンにアップグレードすると、カタログのインデックスイメージも自動的に更新されます。

#### 2.4.1.2.2.2. カタログの正常性要件

クラスター上の Operator カタログは、インストール解決の観点から相互に置き換え可能です。**Subscription** オブジェクトは特定のカタログを参照する場合がありますが、依存関係はクラスターのすべてのカタログを使用して解決されます。

たとえば、カタログ A が正常でない場合、カタログ A を参照するサブスクリプションはカタログ B の依存関係を解決する可能性があります。通常、B のカタログ優先度は A よりも低いため、クラスター管理者はこれおを想定していない可能性があります。

その結果、OLM では、特定のグローバル namespace (デフォルトの **openshift-marketplace** namespace やカスタムグローバル namespace など) を持つすべてのカタログが正常であることが必要になります。カタログが正常でない場合、その共有グローバル namespace 内のすべての Operator のインストールまたは更新操作は、**CatalogSourcesUnhealthy** 状態で失敗します。正常でない状態でこれらの操作が許可されている場合、OLM はクラスター管理者が想定しない解決やインストールを決定する可能性があります。

クラスター管理者として、正常でないカタログが見つかった際にそのカタログを無効とみなして Operator のインストールを再開する場合は、「カスタムカタログの削除」または「デフォルトのソフトウェアカタログソースの無効化」セクションで、正常でないカタログを削除する方法を確認してください。

#### 2.4.1.2.3. サブスクリプション

**サブスクリプション** は、**Subscription** オブジェクトによって定義され、Operator をインストールする意図を表します。これは、Operator をカタログソースに関連付けるカスタムリソースです。

サブスクリプションは、サブスクライブする Operator パッケージのチャネルや、更新を自動または手動で実行するかどうかを記述します。サブスクリプションが自動に設定された場合、Operator Lifecycle Manager (OLM) が Operator を管理し、アップグレードして、最新バージョンがクラスター内で常に実行されるようにします。

**Subscription オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

この **Subscription** オブジェクトは、Operator の名前と namespace、および Operator データのあるカタログを定義します。**alpha**、**beta**、または **stable** などのチャネルは、カタログソースからインストールする必要のある Operator ストリームを判別するのに役立ちます。

サブスクリプションのチャネルの名前は Operators 間で異なる可能性がありますが、命名スキームは指定された Operator 内の一般的な規則に従う必要があります。たとえば、チャネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に基づく可能性があります。

関連するサブスクリプションのステータスを調べると、Operator の新しいバージョンが利用可能になったことを確認できます。これは、OpenShift Dedicated Web コンソールからも簡単に確認できます。**currentCSV** フィールドに関連付けられる値は OLM に認識される最新のバージョンであり、**installedCSV** はクラスターにインストールされるバージョンです。

**関連情報**

- CLI を使用した Operator サブスクリプションステータスの表示

2.4.1.2.4. インストール計画

**InstallPlan** オブジェクトによって定義される **インストール計画** は、Operator Lifecycle Manager(OLM) が特定バージョンの Operator をインストールまたはアップグレードするために作成するリソースのセットを記述します。バージョンはクラスターサービスバージョン (CSV) で定義されます。

Operator、クラスター管理者、または Operator インストールパーミッションが付与されているユーザーをインストールするには、まず **Subscription** オブジェクトを作成する必要があります。サブスクリプションでは、カタログソースから利用可能なバージョンの Operator のストリームにサブスクライブする意図を表します。次に、サブスクリプションは **InstallPlan** オブジェクトを作成し、Operator のリソースのインストールを容易にします。

その後、インストール計画は、以下の承認ストラテジーのいずれかをもとに承認される必要があります。

- サブスクリプションの **spec.installPlanApproval** フィールドが **Automatic** に設定されている場合には、インストール計画は自動的に承認されます。
- サブスクリプションの **spec.installPlanApproval** フィールドが **Manual** に設定されている場合には、インストール計画はクラスター管理者または適切なパーミッションが割り当てられたユーザーによって手動で承認する必要があります。

インストール計画が承認されると、OLM は指定されたリソースを作成し、サブスクリプションで指定された namespace に Operator をインストールします。

**例2.12 InstallPlan オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: InstallPlan
metadata:
  name: install-abcde
  namespace: operators
spec:
  approval: Automatic
  approved: true
  clusterServiceVersionNames:
    - my-operator.v1.0.1
  generation: 1
status:
  ...
```

```
catalogSources: []
conditions:
  - lastTransitionTime: '2021-01-01T20:17:27Z'
    lastUpdateTime: '2021-01-01T20:17:27Z'
    status: 'True'
    type: Installed
phase: Complete
plan:
  - resolving: my-operator.v1.0.1
    resource:
      group: operators.coreos.com
      kind: ClusterServiceVersion
      manifest: >-
      ...
      name: my-operator.v1.0.1
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1alpha1
    status: Created
  - resolving: my-operator.v1.0.1
    resource:
      group: apiextensions.k8s.io
      kind: CustomResourceDefinition
      manifest: >-
      ...
      name: webservers.web.servers.org
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1beta1
    status: Created
  - resolving: my-operator.v1.0.1
    resource:
      group: ''
      kind: ServiceAccount
      manifest: >-
      ...
      name: my-operator
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1
    status: Created
  - resolving: my-operator.v1.0.1
    resource:
      group: rbac.authorization.k8s.io
      kind: Role
      manifest: >-
      ...
      name: my-operator.v1.0.1-my-operator-6d7cbc6f57
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1
    status: Created
  - resolving: my-operator.v1.0.1
    resource:
      group: rbac.authorization.k8s.io
      kind: RoleBinding
```

```
    manifest: >-
    ...
    name: my-operator.v1.0.1-my-operator-6d7cbc6f57
    sourceName: redhat-operators
    sourceNamespace: openshift-marketplace
    version: v1
  status: Created
  ...
```

#### 2.4.1.2.5. Operator グループ

**Operator グループ** は、**OperatorGroup** リソースによって定義され、マルチテナント設定を OLM でインストールされた Operators に提供します。Operator グループは、そのメンバー Operators に必要な RBAC アクセスを生成するために使用するターゲット namespace を選択します。

ターゲット namespace のセットは、クラスターサービスバージョン (CSV) の **olm.targetNamespaces** アノテーションに保存されるコンマ区切りの文字列によって指定されます。このアノテーションは、メンバー Operator の CSV インスタンスに適用され、そのデプロイメントに反映されます。

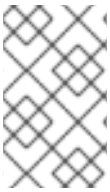
**関連情報**

- Operator グループ

#### 2.4.1.2.6. Operator 条件

Operator のライフサイクル管理のロールの一部として、Operator Lifecycle Manager (OLM) は、Operator を定義する Kubernetes リソースの状態から Operator の状態を推測します。このアプローチでは、Operator が特定の状態にあることをある程度保証しますが、推測できない情報を Operator が OLM と通信して提供する必要がある場合も多々あります。続いて、OLM がこの情報を使用して、Operator のライフサイクルをより適切に管理することができます。

OLM は、Operators が OLM に条件を通信できる **OperatorCondition** というカスタムリソース定義 (CRD) を提供します。**OperatorCondition** リソースの **Spec.Conditions** 配列にある場合に、OLM による Operator の管理に影響するサポートされる条件のセットがあります。



**注記**

デフォルトでは、**Spec.Conditions** 配列は、ユーザーによって追加されるか、カスタム Operator ロジックの結果として追加されるまで、**OperatorCondition** オブジェクトに存在しません。

**関連情報**

- Operator 条件

### 2.4.2. Operator Lifecycle Manager アーキテクチャー

ここでは、OpenShift Dedicated における Operator Lifecycle Manager (OLM) のコンポーネントアーキテクチャーの概要を説明します。

#### 2.4.2.1. コンポーネントの役割

Operator Lifecycle Manager (OLM) は、OLM Operator および Catalog Operator の 2 つの Operator で構成されています。

OLM Operator と Catalog Operator は、それぞれ OLM フレームワークの基礎となるカスタムリソース定義 (CRD) を管理します。

**表2.3 OLM およびカタログ Operators で管理される CRD**

| リソース | 短縮名 | 所有者 | 説明 |
|---|---|---|---|
| **ClusterServiceVersion** (CSV) | **csv** | OLM | アプリケーションのメタデータ: 名前、バージョン、アイコン、必須リソース、インストールなど。 |
| **InstallPlan** | **ip** | Catalog | CSV を自動的にインストールするか、アップグレードするために作成されるリソースの計算された一覧。 |
| **CatalogSource** | **catsrc** | Catalog | CSV、CRD、およびアプリケーションを定義するパッケージのリポジトリー。 |
| **Subscription** | **sub** | Catalog | パッケージのチャネルを追跡して CSV を最新の状態に保つために使用されます。 |
| **OperatorGroup** | **og** | OLM | **OperatorGroup** オブジェクトと同じ namespace にデプロイされたすべての Operators を、namespace のリストまたはクラスター全体でカスタムリソース (CR) を監視できるように設定します。 |

これらの Operators のそれぞれは以下のリソースの作成も行います。

**表2.4 OLM およびカタログ Operators によって作成されるリソース**

| リソース | 所有者 |
|---|---|
| **Deployments** | OLM |
| **ServiceAccounts** | |
| **(Cluster)Roles** | |
| **(Cluster)RoleBindings** | |
| **CustomResourceDefinitions** (CRDs) | Catalog |
| **ClusterServiceVersions** | |

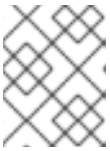
#### 2.4.2.2. OLM Operator

OLM Operator は、CSV で指定された必須リソースがクラスター内にあることが確認された後に CSV リソースで定義されるアプリケーションをデプロイします。

OLM Operator は必須リソースの作成には関与せず、ユーザーが CLI またはカタログ Operator を使用してこれらのリソースを手動で作成することを選択できます。このタスクの分離により、アプリケーションに OLM フレームワークをどの程度活用するかに関連してユーザーによる追加機能の購入を可能にします。

OLM Operator は以下のワークフローを使用します。

1. namespace でクラスターサービスバージョン (CSV) の有無を確認し、要件を満たしていることを確認します。
2. 要件が満たされている場合、CSV のインストールストラテジーを実行します。



**注記**

CSV は、インストールストラテジーの実行を可能にするために Operator グループのアクティブなメンバーである必要があります。

#### 2.4.2.3. Catalog Operator

Catalog Operator はクラスターサービスバージョン (CSV) およびそれらが指定する必須リソースを解決し、インストールします。また、カタログソースでチャネル内のパッケージへの更新の有無を確認し、必要な場合はそれらを利用可能な最新バージョンに自動的にアップグレードします。

チャネル内のパッケージを追跡するために、必要なパッケージ、チャネル、および更新のプルに使用する **CatalogSource** オブジェクトを設定して **Subscription** オブジェクトを作成できます。更新が見つかると、ユーザーに代わって適切な **InstallPlan** オブジェクトの namespace への書き込みが行われます。

Catalog Operator は以下のワークフローを使用します。

1. クラスターの各カタログソースに接続します。
2. ユーザーによって作成された未解決のインストール計画の有無を確認し、これがあった場合は以下を実行します。
   a. 要求される名前に一致する CSV を検索し、これを解決済みリソースとして追加します。
   b. マネージドまたは必須の CRD のそれぞれについて、これを解決済みリソースとして追加します。
   c. 必須 CRD のそれぞれについて、これを管理する CSV を検索します。
3. 解決済みのインストール計画の有無を確認し、それに関する検出されたすべてのリソースを作成します (ユーザーによって、または自動的に承認される場合)。
4. カタログソースおよびサブスクリプションの有無を確認し、それらに基づいてインストール計画を作成します。

#### 2.4.2.4. カタログレジストリー

カタログレジストリーは、クラスター内での作成用に CSV および CRD を保存し、パッケージおよびチャネルに関するメタデータを保存します。

**パッケージマニフェスト** は、パッケージアイデンティティーを CSV のセットに関連付けるカタログレ

ジストリー内のエントリーです。パッケージ内で、チャネルは特定の CSV を参照します。CSV は置き換え対象の CSV を明示的に参照するため、パッケージマニフェストは Catalog Operator に対し、CSV をチャネル内の最新バージョンに更新するために必要なすべての情報を提供します (各中間バージョンをステップスルー)。

### 2.4.3. Operator Lifecycle Manager ワークフロー

ここでは、OpenShift Dedicated における Operator Lifecycle Manager (OLM) のワークフローの概要を説明します。

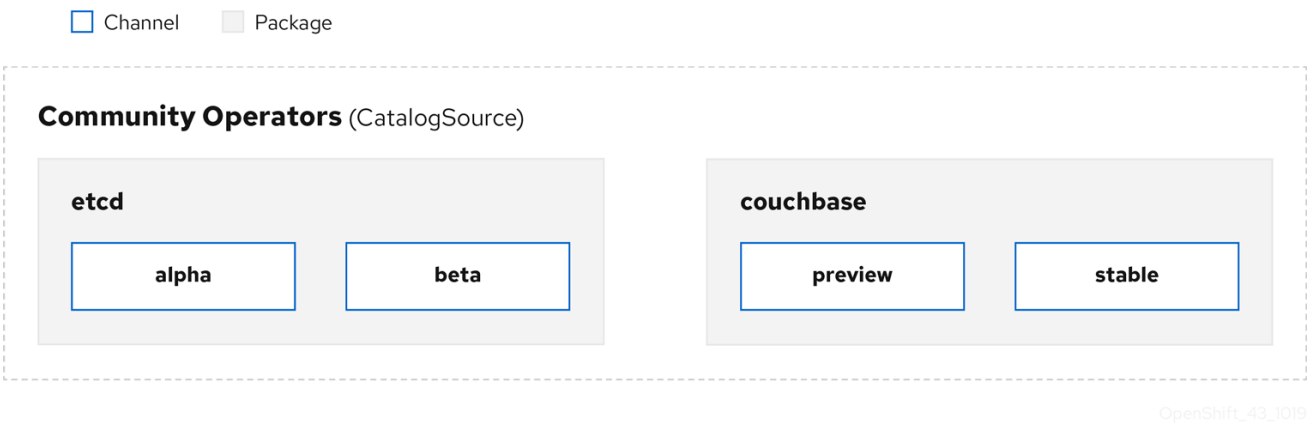
#### 2.4.3.1. OLM での Operator のインストールおよびアップグレードのワークフロー

Operator Lifecycle Manager (OLM) エコシステムでは、以下のリソースを使用して Operator インストールおよびアップグレードを解決します。

- **ClusterServiceVersion** (CSV)
- **CatalogSource**
- **Subscription**

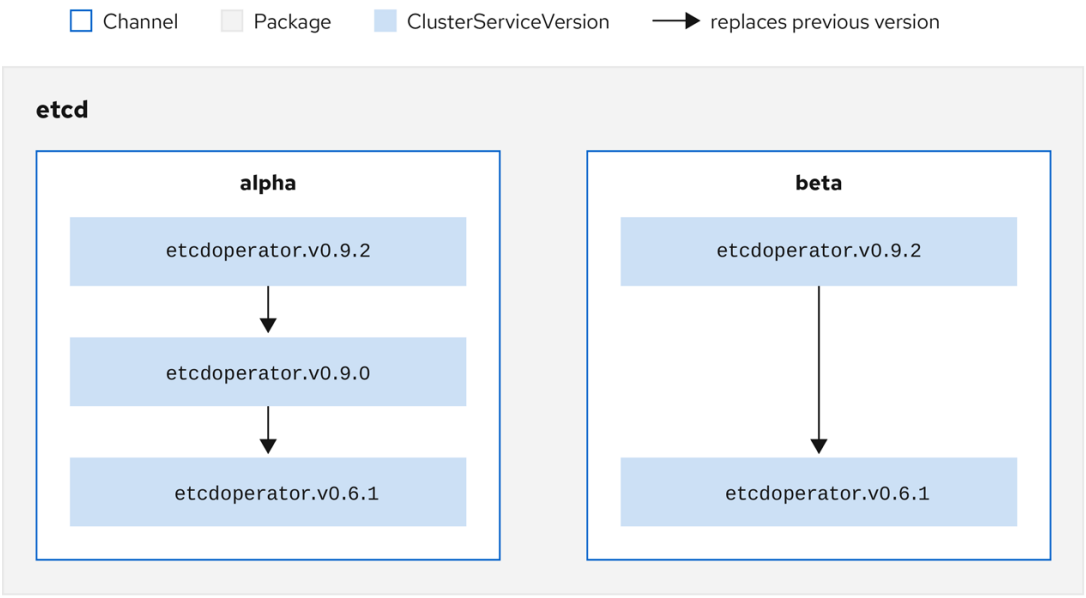
CSV で定義される Operator メタデータは、カタログソースというコレクションに保存できます。OLM はカタログソースを使用します。これは Operator Registry API を使用して利用可能な Operators やインストールされた Operators のアップグレードをクエリーします。

**図2.3 カタログソースの概要**

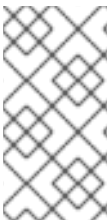



カタログソース内では、Operator は **パッケージ** と、**チャネル** と呼ばれる更新のストリームに編成されています。これは、OpenShift Dedicated や、Web ブラウザーなどの継続的なリリースサイクルを持つその他のソフトウェアでもよく見られる更新パターンです。

図2.4 カタログソースのパッケージおよびチャネル




ユーザーは **サブスクリプション** の特定のカタログソースの特定のパッケージおよびチャネルを指定できます (例: **etcd** パッケージおよびその **alpha** チャネル)。サブスクリプションが namespace にインストールされていないパッケージに対して作成されると、そのパッケージの最新 Operator がインストールされます。

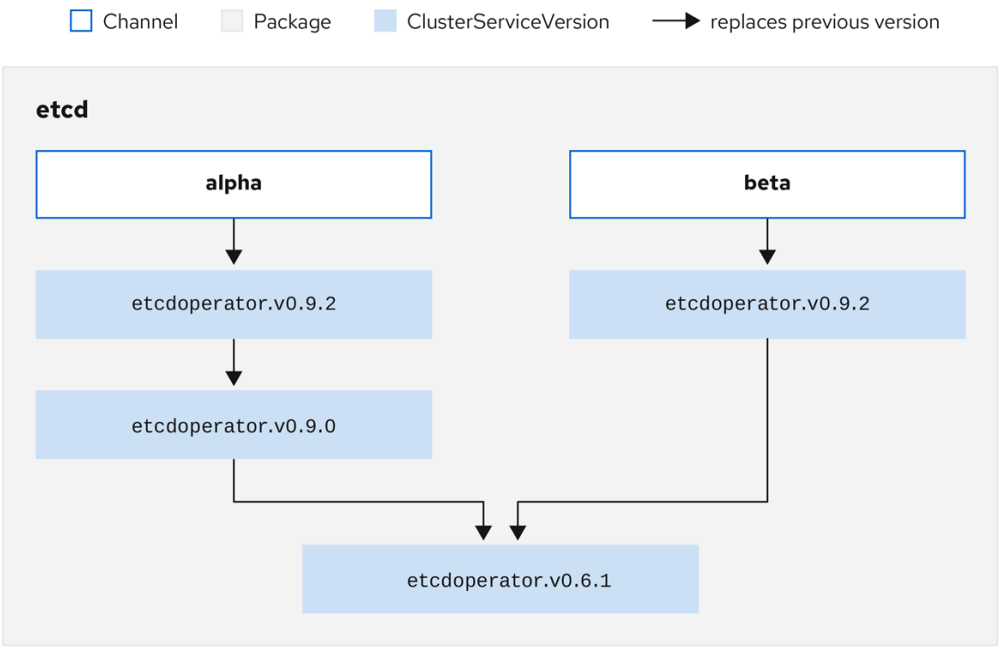


**注記**

OLM では、バージョンの比較が意図的に避けられます。そのため、所定の **catalog → channel → package** パスから利用可能な "latest" または "newest" Operator が必ずしも最も高いバージョン番号である必要はありません。これは Git リポジトリーの場合と同様に、チャネルの **head** リファレンスとして見なされます。

各 CSV には、これが置き換える Operator を示唆する **replaces** パラメーターがあります。これにより、OLM でクエリー可能な CSV のグラフが作成され、更新がチャネル間で共有されます。チャネルは、更新グラフのエントリーポイントと見なすことができます。

図2.5 利用可能なチャネル更新に関する OLM グラフ




パッケージのチャネルの例

```
packageName: example
channels:
- name: alpha
  currentCSV: example.v0.1.2
- name: beta
  currentCSV: example.v0.1.3
defaultChannel: alpha
```

カタログソース、パッケージ、チャネルおよび CSV がある状態で、OLM が更新のクエリーを実行できるようにするには、カタログが入力された CSV の置き換え (**replaces**) を実行する単一 CSV を明確にかつ確定的に返すことができる必要があります。

2.4.3.1.1. アップグレードパスの例

アップグレードシナリオのサンプルについて、CSV バージョン **0.1.1** に対応するインストールされた Operator を見てみましょう。OLM はカタログソースをクエリーし、新規 CSV バージョン **0.1.3** について、サブスクライブされたチャネルのアップグレードを検出します。これは、古いバージョンでインストールされていない CSV バージョン **0.1.2** を置き換えます。その後、さらに古いインストールされた CSV バージョン **0.1.1** を置き換えます。

OLM は、チャネルヘッドから CSV で指定された **replaces** フィールドで以前のバージョンに戻り、アップグレードパス **0.1.3** → **0.1.2** → **0.1.1** を判別します。矢印の方向は前者が後者を置き換えることを示します。OLM は、チャネルヘッドに到達するまで Operator を 1 バージョンずつアップグレードします。

このシナリオでは、OLM は Operator バージョン **0.1.2** をインストールし、既存の Operator バージョン **0.1.1** を置き換えます。その後、Operator バージョン **0.1.3** をインストールし、直前にインストールされた Operator バージョン **0.1.2** を置き換えます。この時点で、インストールされた Operator のバージョン **0.1.3** はチャネルヘッドに一致し、アップグレードは完了します。

#### 2.4.3.1.2. アップグレードの省略

OLM のアップグレードの基本パスは以下の通りです。

- カタログソースは Operator への 1 つ以上の更新によって更新されます。
- OLM は、カタログソースに含まれる最新バージョンに到達するまで、Operator のすべてのバージョンを横断します。

ただし、この操作の実行は安全でない場合があります。公開されているバージョンの Operator がクラスターにインストールされていない場合、そのバージョンによって深刻な脆弱性が導入される可能性があるなどの理由で、その Operator をクラスターにインストールできないことがあります。

この場合、OLM は以下の 2 つのクラスターの状態を考慮に入れて、それらの両方に対応する更新グラフを提供する必要があります。

- "問題のある" 中間 Operator がクラスターによって確認され、かつインストールされている。
- "問題のある" 中間 Operator がクラスターにまだインストールされていない。

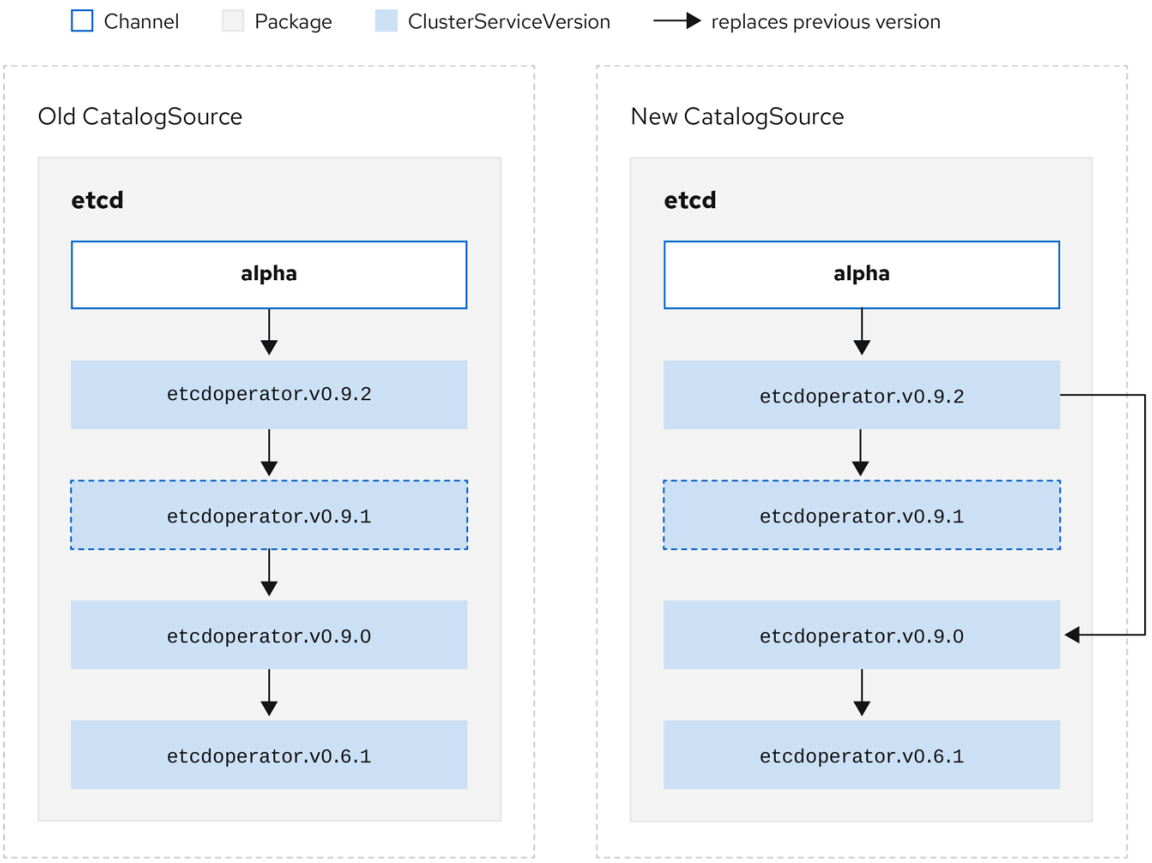
OLM は、新規カタログを送り、**省略された** リリースを追加することで、クラスターの状態や問題のある更新が発見されたかどうかにかかわらず、単一の固有の更新を常に取得することができます。

**省略されたリリースの CSV 例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: etcdoperator.v0.9.2
  namespace: placeholder
  annotations:
spec:
    displayName: etcd
    description: Etcd Operator
    replaces: etcdoperator.v0.9.0
    skips:
    - etcdoperator.v0.9.1
```

**古い CatalogSource** および **新規 CatalogSource** に関する以下の例を見てみましょう。

図2.6 更新のスキップ




このグラフは、以下を示しています。

- **古い CatalogSource** の Operator には、**新規 CatalogSource** の単一の置き換えがある。
- **新規 CatalogSource** の Operator には、**新規 CatalogSource** の単一の置き換えがある。
- 問題のある更新がインストールされていない場合、これがインストールされることはない。

#### 2.4.3.1.3. 複数の Operators の置き換え

前述の方法で **新規 CatalogSource** を作成するには、ある Operator を置き換えつつ (**replace**)、複数バージョンをスキップ (**skip**) できる CSV を公開する必要があります。これは、**skipRange** アノテーションを使用して実行できます。

```
olm.skipRange: <semver_range>
```

ここで **<semver_range>** には、semver ライブラリー でサポートされるバージョン範囲の形式が使用されます。

カタログで更新を検索する場合、チャネルのヘッドに **skipRange** アノテーションがあり、現在インストールされている Operator にその範囲内のバージョンフィールドがある場合、OLM はチャネル内の最新エントリーに対して更新されます。

以下は動作が実行される順序になります。

1. サブスクリプションの **sourceName** で指定されるソースのチャネルヘッド (省略する他の条件が満たされている場合)。
2. **sourceName** で指定されるソースの現行バージョンを置き換える次の Operator。
3. サブスクリプションに表示される別のソースのチャネルヘッド (省略する他の条件が満たされている場合)。
4. サブスクリプションに表示されるソースの現行バージョンを置き換える次の Operator。

**skipRange を含む CSV の例**

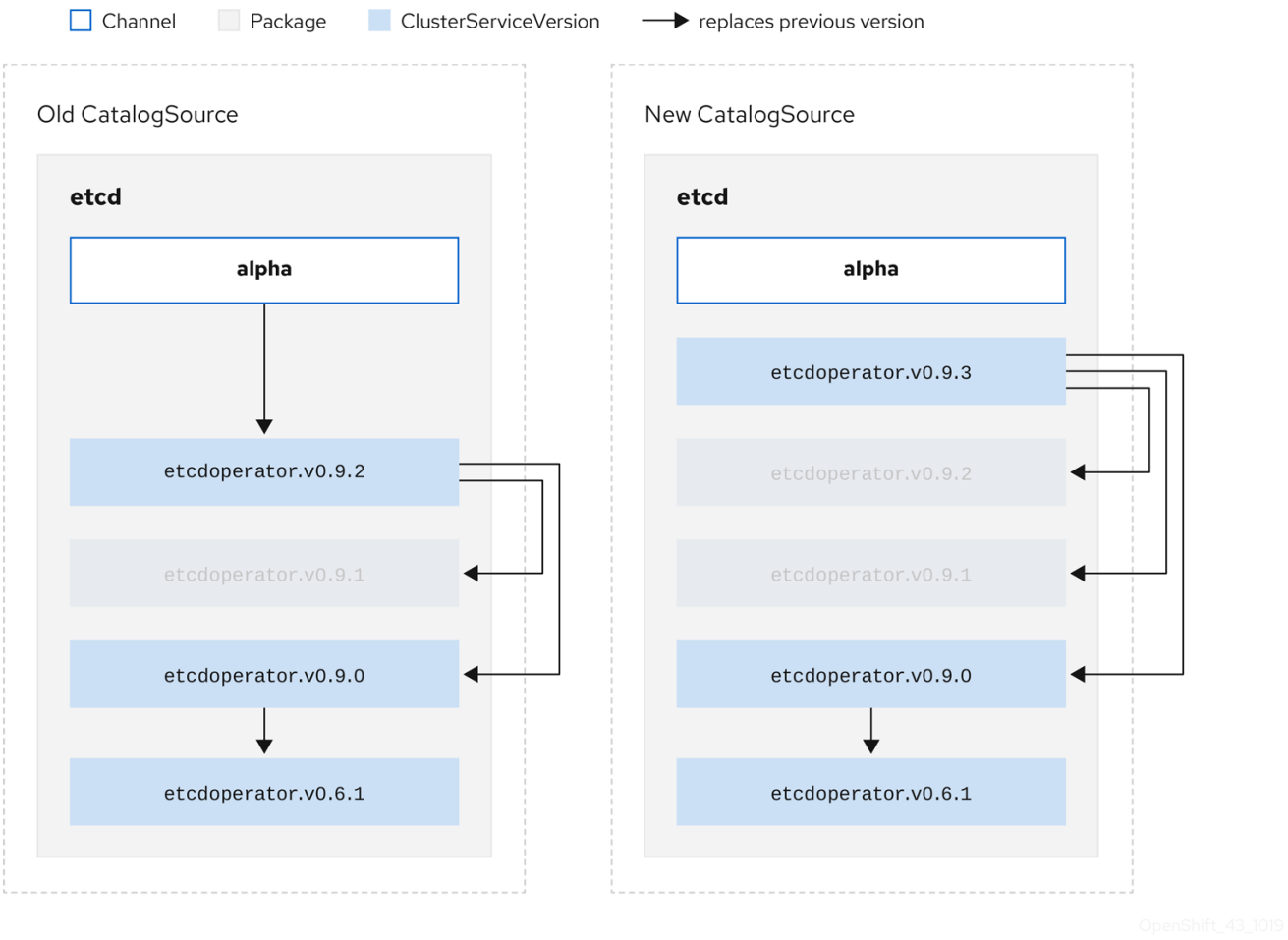
```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: elasticsearch-operator.v4.1.2
  namespace: <namespace>
  annotations:
    olm.skipRange: '>=4.1.0 <4.1.2'
```

#### 2.4.3.1.4. z-stream サポート

**z-stream** またはパッチリリースは、同じマイナーバージョンの以前のすべての z-stream リリースを置き換える必要があります。OLM は、メジャー、マイナーまたはパッチバージョンを考慮せず、カタログ内で正確なグラフのみを作成する必要があります。

つまり、OLM では **古い CatalogSource** のようにグラフを使用し、以前と同様に **新規 CatalogSource** にあるようなグラフを生成する必要があります。

図2.7 複数 Operators の置き換え




このグラフは、以下を示しています。

- **古い CatalogSource** の Operator には、**新規 CatalogSource** の単一の置き換えがある。
- **新規 CatalogSource** の Operator には、**新規 CatalogSource** の単一の置き換えがある。
- **古い CatalogSource** の z-stream リリースは、**新規 CatalogSource** の最新 z-stream リリースに更新される。
- 使用不可のリリースは "仮想" グラフノードと見なされる。それらのコンテンツは存在する必要がなく、レジストリーはグラフが示すように応答することのみが必要になります。

## 2.4.4. Operator Lifecycle Manager の依存関係の解決

ここでは、OpenShift Dedicated における Operator Lifecycle Manager (OLM) を使用した依存関係の解決とカスタムリソース定義 (CRD) のアップグレードライフサイクルの概要を説明します。

### 2.4.4.1. 依存関係の解決

Operator Lifecycle Manager (OLM) は、実行中の Operators の依存関係の解決とアップグレードのライフサイクルを管理します。多くの場合、OLM が直面する問題は、**yum** や **rpm** などの他のシステムまたは言語パッケージマネージャーと同様です。

ただし、OLM にはあるものの、通常同様のシステムにはない 1 つの制約があります。Operators は常に実行されており、OLM は相互に機能しない Operators のセットの共存を防ごうとします。

その結果、以下のシナリオで OLM を使用しないでください。

- 提供できない API を必要とする Operators のセットのインストール
- Operator と依存関係のあるものに障害を発生させる仕方での Operator の更新

これは、次の 2 種類のデータで可能になります。

| | |
|---|---|
| プロパティー | Operator に関する型付きのメタデータ。これは、依存関係のリゾルバーで Operator の公開インターフェイスを構成します。例としては、Operator が提供する API の group/version/kind (GVK) や Operator のセマンティックバージョン (semver) などがあります。 |
| 制約または依存関係 | ターゲットクラスターにすでにインストールされているかどうかに関係なく、他の Operators が満たす必要のある Operator の要件。これらは、使用可能なすべての Operators に対するクエリーまたはフィルターとして機能し、依存関係の解決およびインストール中に選択を制限します。クラスターで特定の API が利用できる状態にする必要がある場合や、特定のバージョンに特定の Operator をインストールする必要がある場合など、例として挙げられます。 |

OLM は、これらのプロパティーと制約をブール式のシステムに変換して SAT ソルバーに渡します。これは、ブールの充足可能性を確立するプログラムであり、インストールする Operators を決定する作業を行います。

#### 2.4.4.2. Operator のプロパティー

カタログ内の Operators にはすべて、次のプロパティーが含まれます。

**olm.package**

パッケージの名前と Operator のバージョンを含めます。

**olm.gvk**

クラスターサービスバージョン (CSV) から提供された API ごとに 1 つのプロパティー

追加のプロパティーは、Operator バンドルの **metadata/** ディレクトリーに **properties.yaml** ファイルを追加して、Operator 作成者が直接宣言することもできます。

**任意のプロパティーの例**

```
properties:
- type: olm.kubeversion
  value:
    version: "1.16.0"
```

##### 2.4.4.2.1. 任意のプロパティー

Operator の作成者は、Operator バンドルの **metadata/** ディレクトリーにある **properties.yaml** ファイルで任意のプロパティーを宣言できます。これらのプロパティーは、実行時に Operator Lifecycle Manager (OLM) リゾルバーへの入力として使用されるマップデータ構造に変換されます。

これらのプロパティーはリゾルバーには不透明です。リゾルバーはプロパティーを理解しませんが、これらのプロパティーに対する一般的な制約を評価して、プロパティーリストを指定することで制約を満たすことができるかどうかを判断します。

**任意のプロパティーの例**

```
properties:
  - property:
      type: color
      value: red
  - property:
      type: shape
      value: square
  - property:
      type: olm.gvk
      value:
        group: olm.coreos.io
        version: v1alpha1
        kind: myresource
```

この構造を使用して、ジェネリック制約の Common Expression Language (CEL) 式を作成できます。

**関連情報**

- Common Expression Language (CEL) の制約

#### 2.4.4.3. Operator の依存関係

Operator の依存関係は、バンドルの **metadata/** フォルダー内の **dependencies.yaml** ファイルに一覧表示されます。このファイルはオプションであり、現時点では明示的な Operator バージョンの依存関係を指定するためにのみ使用されます。

依存関係の一覧には、依存関係の内容を指定するために各項目の **type** フィールドが含まれます。次のタイプの Operator 依存関係がサポートされています。

**olm.package**

このタイプは、特定の Operator バージョンの依存関係であることを意味します。依存関係情報には、パッケージ名とパッケージのバージョンを semver 形式で含める必要があります。たとえば、**0.5.2** などの特定バージョンや **>0.5.1** などのバージョンの範囲を指定することができます。

**olm.gvk**

このタイプの場合、作成者は CSV の既存の CRD および API ベースの使用方法と同様に group/version/kind (GVK) 情報で依存関係を指定できます。これは、Operator の作成者がすべての依存関係、API または明示的なバージョンを同じ場所に配置できるようにするパスです。

**olm.constraint**

このタイプは、任意の Operator プロパティーに対するジェネリック制約を宣言します。

以下の例では、依存関係は Prometheus Operator および etcd CRD に指定されます。

**dependencies.yaml ファイルの例**

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
```

```
    group: etcd.database.coreos.com
    kind: EtcdCluster
    version: v1beta2
```

#### 2.4.4.4. 一般的な制約

**olm.constraint** プロパティーは、特定のタイプの依存関係制約を宣言し、非制約プロパティーと制約プロパティーを区別します。その **value** フィールドは、制約メッセージの文字列表現を保持する **failureMessage** フィールドを含むオブジェクトです。このメッセージは、実行時に制約が満たされない場合に、ユーザーへの参考のコメントとして表示されます。

次のキーは、使用可能な制約タイプを示します。

**gvk**

値と解釈が **olm.gvk** タイプと同じタイプ

**package**

値と解釈が **olm.package** タイプと同じタイプ

**cel**

任意のバンドルプロパティーとクラスター情報に対して Operator Lifecycle Manager (OLM) リゾルバーによって実行時に評価される Common Expression Language (CEL) 式

**all、any、not**

**gvk** やネストされた複合制約など、1 つ以上の具体的な制約を含む、論理積、論理和、否定の制約。

##### 2.4.4.4.1. Common Expression Language (CEL) の制約

**cel** 制約型は、式言語として Common Expression Language (CEL) をサポートしています。**cel** 構造には、Operator が制約を満たしているかどうかを判断するために、実行時に Operator プロパティーに対して評価される CEL 式文字列を含む **rule** フィールドがあります。

**cel 制約の例**

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified"'
  cel:
    rule: 'properties.exists(p, p.type == "certified")'
```

CEL 構文は、**AND** や **OR** などの幅広い論理演算子をサポートします。その結果、単一の CEL 式は、これらの論理演算子で相互にリンクされる複数の条件に対して複数のルールを含めることができます。これらのルールは、バンドルまたは任意のソースからの複数の異なるプロパティーのデータセットに対して評価され、出力は、単一の制約内でこれらのルールのすべてを満たす単一のバンドルまたは Operator に対して解決されます。

**複数のルールが指定された cel 制約の例**

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified" and "stable" properties'
  cel:
    rule: 'properties.exists(p, p.type == "certified") && properties.exists(p, p.type == "stable")'
```

#### 2.4.4.4.2. 複合制約 (all, any, not)

複合制約タイプは、論理定義に従って評価されます。

以下は、2 つのパッケージと 1 つの GVK の接続制約 (**all**) の例です。つまり、インストールされたバンドルがすべての制約を満たす必要があります。

**all 制約の例**

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: All are required for Red because...
    all:
      constraints:
      - failureMessage: Package blue is needed for...
        package:
          name: blue
          versionRange: '>=1.0.0'
      - failureMessage: GVK Green/v1 is needed for...
        gvk:
          group: greens.example.com
          version: v1
          kind: Green
```

以下は、同じ GVK の 3 つのバージョンの選言的制約 (**any**) の例です。つまり、インストールされたバンドルが少なくとも 1 つの制約を満たす必要があります。

**any 制約の例**

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Any are required for Red because...
    any:
      constraints:
      - gvk:
          group: blues.example.com
          version: v1beta1
          kind: Blue
      - gvk:
          group: blues.example.com
          version: v1beta2
          kind: Blue
      - gvk:
          group: blues.example.com
          version: v1
          kind: Blue
```

以下は、GVK の 1 つのバージョンの否定制約 (**not**) の例です。つまり、この結果セットのバンドルでは、この GVK を提供できません。

**not の制約例**

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
  all:
    constraints:
    - failureMessage: Package blue is needed for...
      package:
        name: blue
        versionRange: '>=1.0.0'
    - failureMessage: Cannot be required for Red because...
      not:
        constraints:
        - gvk:
            group: greens.example.com
            version: v1alpha1
            kind: greens
```

否定のセマンティクスは、**not** 制約のコンテキストで不明確であるように見える場合があります。つまり、この否定では、特定の GVK、あるバージョンのパッケージを含むソリューション、または結果セットからの子の複合制約を満たすソリューションを削除するように、リゾルバーに対して指示を出しています。

当然の結果として、最初に可能な依存関係のセットを選択せずに否定することは意味がないため、複合では **not** 制約は **all** または **any** 制約内でのみ使用する必要があります。

#### 2.4.4.4.3. ネストされた複合制約

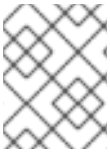
ネストされた複合制約 (少なくとも 1 つの子複合制約と 0 個以上の単純な制約を含む制約) は、前述の各制約タイプの手順に従って、下から上に評価されます。

以下は、接続詞の論理和の例で、one、the other、または both が制約を満たすことができます。

**ネストされた複合制約の例**

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Required for Red because...
    any:
      constraints:
      - all:
          constraints:
          - package:
              name: blue
              versionRange: '>=1.0.0'
          - gvk:
              group: blues.example.com
              version: v1
              kind: Blue
```

```
    - all:
        constraints:
        - package:
            name: blue
            versionRange: '<1.0.0'
        - gvk:
            group: blues.example.com
            version: v1beta1
            kind: Blue
```



**注記**

**olm.constraint** タイプの最大 raw サイズは 64KB に設定されており、リソース枯渇攻撃を制限しています。

#### 2.4.4.5. 依存関係の設定

Operator の依存関係を同等に満たすオプションが多数ある場合があります。Operator Lifecycle Manager (OLM) の依存関係リゾルバーは、要求された Operator の要件に最も適したオプションを判別します。Operator の作成者またはユーザーとして、依存関係の解決が明確になるようにこれらの選択方法を理解することは重要です。

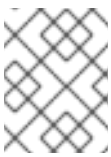
##### 2.4.4.5.1. カタログの優先順位

OpenShift Dedicated クラスターでは、OLM がカタログソースを読み取り、どの Operator がインストール可能であるかを確認します。

**CatalogSource オブジェクトの例**

```
apiVersion: "operators.coreos.com/v1alpha1"
kind: "CatalogSource"
metadata:
  name: "my-operators"
  namespace: "operators"
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 1
  image: example.com/my/operator-index:v1
  displayName: "My Operators"
  priority: 100
```

1 **legacy** または **restricted** の値を指定します。フィールドが設定されていない場合、デフォルト値は **legacy** です。今後の OpenShift Dedicated リリースでは、デフォルト値が **restricted** になる予定です。



**注記**

**restricted** 権限でカタログを実行できない場合は、このフィールドを手動で **legacy** に設定することを推奨します。

**CatalogSource** オブジェクトには **priority** フィールドがあります。このフィールドは、依存関係のオプションを優先する方法を把握するためにリゾルバーによって使用されます。

カタログ設定を規定する 2 つのルールがあります。

- 優先順位の高いカタログにあるオプションは、優先順位の低いカタログのオプションよりも優先されます。
- 依存オブジェクトと同じカタログにあるオプションは他のカタログよりも優先されます。

#### 2.4.4.5.2. チャネルの順序付け

カタログ内の Operator パッケージは、OpenShift Dedicated クラスターでユーザーがサブスクライブできる更新チャネルのコレクションです。チャネルは、マイナーリリース (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に関する特定の更新ストリームを提供するために使用できます。

同じパッケージの Operators によって依存関係が満たされる可能性がありますが、その場合、異なるチャネルの Operators のバージョンによって満たされる可能性があります。たとえば、Operator のバージョン **1.2** は **stable** および **fast** チャネルの両方に存在する可能性があります。

それぞれのパッケージにはデフォルトのチャネルがあり、これは常にデフォルト以外のチャネルよりも優先されます。デフォルトチャネルのオプションが依存関係を満たさない場合には、オプションは、チャネル名の辞書式順序 (lexicographic order) で残りのチャネルから検討されます。

#### 2.4.4.5.3. チャネル内での順序

ほとんどの場合、単一のチャネル内に依存関係を満たすオプションが複数あります。たとえば、1 つのパッケージおよびチャネルの Operators は同じセットの API を提供します。

ユーザーがサブスクリプションを作成すると、それらはどのチャネルから更新を受け取るかを示唆します。これにより、すぐにその 1 つのチャネルだけに検索が絞られます。ただし、チャネル内では、多くの Operators が依存関係を満たす可能性があります。

チャネル内では、更新グラフでより上位にある新規 Operators が優先されます。チャネルのヘッドが依存関係を満たす場合、これがまず試行されます。

#### 2.4.4.5.4. その他の制約

OLM には、パッケージの依存関係で指定される制約のほかに、必要なユーザーの状態を表し、常にメンテナンスする必要のある依存関係の解決を適用するための追加の制約が含まれます。

##### 2.4.4.5.4.1. サブスクリプションの制約

サブスクリプションの制約は、サブスクリプションを満たすことのできる Operators のセットをフィルターします。サブスクリプションは、依存関係リゾルバーに関するユーザー指定の制約です。それらは、クラスター上にない場合は新規 Operator をインストールすることを宣言するか、既存 Operator の更新された状態を維持することを宣言します。

##### 2.4.4.5.4.2. パッケージの制約

namespace 内では、2 つの Operators が同じパッケージから取得されることはありません。

#### 2.4.4.5.5. 関連情報

- カタログの正常性要件

#### 2.4.4.6. CRD のアップグレード

OLM は、単一のクラスターサービスバージョン (CSV) によって所有されている場合にはカスタムリソース定義 (CRD) をすぐにアップグレードします。CRD が複数の CSV によって所有されている場合、CRD は、以下の後方互換性の条件のすべてを満たす場合にアップグレードされます。

- 現行 CRD の既存の有効にされたバージョンすべてが新規 CRD に存在する。
- 検証が新規 CRD の検証スキーマに対して行われる場合、CRD の提供バージョンに関連付けられる既存インスタンスまたはカスタムリソースすべてが有効である。

#### 2.4.4.7. 依存関係のベストプラクティス

依存関係を指定する際には、ベストプラクティスを考慮する必要があります。

**Operators の API または特定のバージョン範囲によって異なります。**

Operators は API をいつでも追加または削除できます。Operators が必要とする API に **olm.gvk** 依存関係を常に指定できます。これの例外は、**olm.package** 制約を代わりに指定する場合です。

**最小バージョンの設定**

API の変更に関する Kubernetes ドキュメントでは、Kubernetes 形式の Operators で許可される変更を説明しています。これらのバージョン管理規則により、Operator は API バージョンに後方互換性がある限り、API バージョンに影響を与えずに API を更新することができます。
Operator の依存関係の場合、依存関係の API バージョンを把握するだけでは、依存する Operator が確実に意図された通りに機能することを確認できないことを意味します。

以下に例を示します。

- TestOperator v1.0.0 は、v1alpha1 API バージョンの **MyObject** リソースを提供します。
- TestOperator v1.0.1 は新しいフィールド **spec.newfield** を **MyObject** に追加しますが、v1alpha1 のままになります。

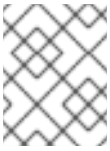
Operator では、**spec.newfield** を **MyObject** リソースに書き込む機能が必要になる場合があります。**olm.gvk** 制約のみでは、OLM で TestOperator v1.0.0 ではなく TestOperator v1.0.1 が必要であると判断することはできません。

可能な場合には、API を提供する特定の Operator が事前に分かっている場合、最小値を設定するために追加の **olm.package** 制約を指定します。

**最大バージョンを省略するか、幅広いバージョンを許可します。**

Operators は API サービスや CRD などのクラスタースコープのリソースを提供するため、依存関係に小規模な範囲を指定する Operator は、その依存関係の他のコンシューマーの更新に不要な制約を加える可能性があります。
可能な場合は、最大バージョンを設定しないでください。または、他の Operators との競合を防ぐために、幅広いセマンティクスの範囲を設定します。例: **>1.0.0 <2.0.0**

従来のパッケージマネージャーとは異なり、Operator の作成者は更新が OLM のチャネルで更新を安全に行われるように Operator を明示的にエンコードします。更新が既存のサブスクリプションで利用可能な場合、Operator の作成者がこれが以前のバージョンから更新できることを示唆していることが想定されます。依存関係の最大バージョンを設定すると、特定の上限で不必要な切り捨てが行われることにより、作成者の更新ストリームがオーバーライドされます。



**注記**

クラスター管理者は、Operator の作成者が設定した依存関係をオーバーライドできません。

ただし、回避する必要がある非互換性があることが分かっている場合は、最大バージョンを設定でき、およびこれを設定する必要があります。バージョン範囲の構文を使用すると、複数の特定のバージョンを省略できます (例: **> 1.0.0 !1.2.1**)。

**関連情報**

- Kubernetes ドキュメント: Changing the API

### 2.4.4.8. 依存関係に関する注意事項

依存関係を指定する際には、考慮すべき注意事項があります。

**複合制約がない (AND)**

現時点で、制約の間に AND 関係を指定する方法はありません。つまり、ある Operator が、所定の API を提供し、バージョン **>1.1.0** を持つ別の Operator に依存するように指定することはできません。
依存関係を指定すると、以下のようになります。

```
dependencies:
- type: olm.package
  value:
    packageName: etcd
    version: ">3.1.0"
- type: olm.gvk
  value:
    group: etcd.database.coreos.com
    kind: EtcdCluster
    version: v1beta2
```

OLM は EtcdCluster を提供する Operators とバージョン **>3.1.0** を持つ Operators の 2 つの Operators で、上記の依存関係の例の条件を満たすことができる可能性があります。その場合や、または両方の制約を満たす Operator が選択されるかどうかは、選択できる可能性のあるオプションが参照される順序によって変わります。依存関係の設定および順序のオプションは十分に定義され、理にかなったものであると考えられますが、Operators は継続的に特定のメカニズムをベースとする必要があります。

**namespace 間の互換性**

OLM は namespace スコープで依存関係の解決を実行します。ある namespace での Operator の更新が別の namespace の Operator の問題となる場合、更新のデッドロックが生じる可能性があります。

### 2.4.4.9. 依存関係解決のシナリオ例

以下の例で、**プロバイダー** は CRD または API サービスを "所有" する Operator です。

#### 2.4.4.9.1. 例: 依存 API を非推奨にする

A および B は API (CRD):

- A のプロバイダーは B によって異なる。
- B のプロバイダーにはサブスクリプションがある。
- B のプロバイダーは C を提供するように更新するが、B を非推奨にする。

この結果は以下のようになります。

- B にはプロバイダーがなくなる。
- A は機能しなくなる。

これは OLM がアップグレードストラテジーで回避するケースです。

#### 2.4.4.9.2. 例: バージョンのデッドロック

A および B は API である:

- A のプロバイダーは B を必要とする。
- B のプロバイダーは A を必要とする。
- A のプロバイダーは (A2 を提供し、B2 を必要とするように) 更新し、A を非推奨にする。
- B のプロバイダーは (B2 を提供し、A2 を必要とするように) 更新し、B を非推奨にする。

OLM が B を同時に更新せずに A を更新しようとする場合や、その逆の場合、OLM は、新しい互換性のあるセットが見つかったとしても Operators の新規バージョンに進むことができません。

これは OLM がアップグレードストラテジーで回避するもう 1 つのケースです。

### 2.4.5. Operator グループ

ここでは、OpenShift Dedicated における Operator Lifecycle Manager (OLM) での Operator グループの使用を概説します。

#### 2.4.5.1. Operator グループについて

**Operator グループ** は、**OperatorGroup** リソースによって定義され、マルチテナント設定を OLM でインストールされた Operators に提供します。Operator グループは、そのメンバー Operators に必要な RBAC アクセスを生成するために使用するターゲット namespace を選択します。

ターゲット namespace のセットは、クラスターサービスバージョン (CSV) の **olm.targetNamespaces** アノテーションに保存されるコンマ区切りの文字列によって指定されます。このアノテーションは、メンバー Operator の CSV インスタンスに適用され、そのデプロイメントに反映されます。

#### 2.4.5.2. Operator グループメンバーシップ

Operator は、以下の条件が true の場合に Operator グループの **メンバー** とみなされます。

- Operator の CSV が Operator グループと同じ namespace にある。
- Operator の CSV のインストールモードは Operator グループがターゲットに設定する namespace のセットをサポートする。

CSV のインストールモードは **InstallModeType** フィールドおよびブール値の **Supported** フィールドで構成されます。CSV の仕様には、4 つの固有の **InstallModeTypes** のインストールモードのセットを含めることができます。

**表2.5 インストールモードおよびサポートされる Operator グループ**

| InstallModeType | 説明 |
|---|---|
| **OwnNamespace** | Operator は、独自の namespace を選択する Operator グループのメンバーにすることができます。 |
| **SingleNamespace** | Operator は 1 つの namespace を選択する Operator グループのメンバーにすることができます。 |
| **MultiNamespace** | Operator は複数の namespace を選択する Operator グループのメンバーにすることができます。 |
| **AllNamespaces** | Operator はすべての namespace を選択する Operator グループのメンバーにすることができます (設定されるターゲット namespace は空の文字列 **""** です)。 |

**注記**

CSV の仕様が **InstallModeType** のエントリーを省略する場合、そのタイプは暗黙的にこれをサポートする既存エントリーによってサポートが示唆されない限り、サポートされないものとみなされます。

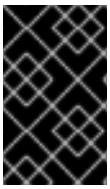
#### 2.4.5.3. ターゲット namespace の選択

**spec.targetNamespaces** パラメーターを使用して Operator グループのターゲット namespace に名前を明示的に指定することができます。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  targetNamespaces:
  - my-namespace
```

または、**spec.selector** パラメーターでラベルセレクターを使用して namespace を指定することもできます。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  selector:
    cool.io/prod: "true"
```



**重要**

**spec.targetNamespaces** で複数の namespace をリスト表示したり、**spec.selector** でラベルセレクターを使用したりすることは推奨されません。Operator グループの複数のターゲット namespace のサポートは今後のリリースで取り除かれる可能性があります。

**spec.targetNamespaces** と **spec.selector** の両方が定義されている場合、**spec.selector** は無視されます。または、**spec.selector** と **spec.targetNamespaces** の両方を省略し、**global** Operator グループを指定できます。これにより、すべての namespace が選択されます。

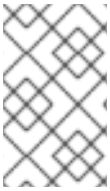
```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
```

選択された namespace の解決済みのセットは Operator グループの **status.namespaces** パラメーターに表示されます。グローバル Operator グループの **status.namespace** には空の文字列 (**""**) が含まれます。これは、消費する Operator に対し、すべての namespace を監視するように示唆します。

#### 2.4.5.4. Operator グループの CSV アノテーション

Operator グループのメンバー CSV には以下のアノテーションがあります。

| アノテーション | 説明 |
|---|---|
| **olm.operatorGroup=<group_name>** | Operator グループの名前が含まれます。 |
| **olm.operatorNamespace=<group_namespace>** | Operator グループの namespace が含まれます。 |
| **olm.targetNamespaces=<target_namespaces>** | Operator グループのターゲット namespace 選択をリスト表示するコンマ区切りの文字列が含まれます。 |



**注記**

**olm.targetNamespaces** 以外のすべてのアノテーションがコピーされた CSV と共に含まれます。**olm.targetNamespaces** アノテーションをコピーされた CSV で省略すると、テナント間のターゲット namespace の重複が回避されます。

#### 2.4.5.5. 提供される API アノテーション

**group/version/kind(GVK)** は Kubernetes API の一意の識別子です。Operator グループによって提供される GVK に関する情報が **olm.providedAPIs** アノテーションに表示されます。アノテーションの値は、コンマで区切られた **<kind>.<version>.<group>** で構成される文字列です。Operator グループのすべてのアクティブメンバーの CSV によって提供される CRD および API サービスの GVK が含まれます。

**PackageManifest** リースを提供する単一のアクティブメンバー CSV を含む **OperatorGroup** オブジェクトの以下の例を確認してください。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: PackageManifest.v1alpha1.packages.apps.redhat.com
  name: olm-operators
  namespace: local
  ...
spec:
  selector: {}
  serviceAccountName:
    metadata:
      creationTimestamp: null
  targetNamespaces:
  - local
status:
  lastUpdated: 2019-02-19T16:18:28Z
  namespaces:
  - local
```

#### 2.4.5.6. ロールベースのアクセス制御

Operator グループの作成時に、3 つのクラスタールールが生成されます。クラスターロールが生成されると、各クラスターロールが一意になるように、ハッシュ値が自動的に末尾に付加されます。各 Operator グループには、次の表に示すように、ラベルに一致するように設定されたクラスターロールセレクターを持つ 1 つの集約ルールが含まれています。

| クラスターロール | 一致するラベル |
| --- | --- |
| **olm.og.<operatorgroup_name>-admin-<hash_value>** | **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>** |
| **olm.og.<operatorgroup_name>-edit-<hash_value>** | **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>** |
| **olm.og.<operatorgroup_name>-view-<hash_value>** | **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>** |

**注記**

Operator グループのクラスターロールを使用してリソースにロールベースのアクセス制御 (RBAC) を割り当てるには、次のコマンドを実行して、クラスターロールの完全な名前とハッシュ値を取得します。

```
$ oc get clusterroles | grep <operatorgroup_name>
```

ハッシュ値は Operator グループの作成時に生成されるため、クラスターロールの完全な名前を検索するには、先に Operator グループを作成する必要があります。

以下の RBAC リソースは、CSV が **AllNamespaces** インストールモードのあるすべての namespace を監視しており、理由が **InterOperatorGroupOwnerConflict** の失敗状態にない限り、CSV が Operator グループのアクティブメンバーになる際に生成されます。

- CRD からの各 API リソースのクラスターロール
- API サービスからの各 API リソースのクラスターロール
- 追加のロールおよびロールバインディング

**表2.6 CRD からの各 API リソース用に生成されたクラスターロール**

| クラスターロール | 設定 |
| --- | --- |
| **<kind>.<group>-<version>-admin** | **<kind>** の動詞<br>• *****<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-admin: true**<br>• **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>** |
| **<kind>.<group>-<version>-edit** | **<kind>** の動詞<br>• **create**<br>• **update**<br>• **patch**<br>• **delete**<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-edit: true**<br>• **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>** |

| クラスターロール | 設定 |
| --- | --- |
| **<kind>.<group>-<version>-view** | **<kind>** の動詞<br>• **get**<br>• **list**<br>• **watch**<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-view: true**<br>• **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>** |
| **<kind>.<group>-<version>-view-crdview** | Verbs on **apiextensions.k8s.io customresourcedefinitions <crd-name>**:<br>• **get**<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-view: true**<br>• **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>** |

**表2.7 API サービスから各 API リソース用に生成されたクラスターロール**

| クラスターロール | 設定 |
| --- | --- |
| **<kind>.<group>-<version>-admin** | **<kind>** の動詞<br>• *<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-admin: true**<br>• **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>** |

| クラスターロール | 設定 |
| --- | --- |
| **<kind>.<group>-<version>-edit** | **<kind>** の動詞<br>• **create**<br>• **update**<br>• **patch**<br>• **delete**<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-edit: true**<br>• **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>** |
| **<kind>.<group>-<version>-view** | **<kind>** の動詞<br>• **get**<br>• **list**<br>• **watch**<br>集計ラベル:<br>• **rbac.authorization.k8s.io/aggregate-to-view: true**<br>• **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>** |

**追加のロールおよびロールバインディング**

- CSV が * が含まれる 1 つのターゲット namespace を定義する場合、クラスターロールと対応するクラスターロールバインディングが CSV の **permissions** フィールドに定義されるパーミッションごとに生成されます。生成されたすべてのリソースには **olm.owner: <csv_name>** および **olm.owner.namespace: <csv_namespace>** ラベルが付与されます。
- CSV が * が含まれる 1 つのターゲット namespace を定義 **しない** 場合、**olm.owner: <csv_name>** および **olm.owner.namespace: <csv_namespace>** ラベルの付いた Operator namespace にあるすべてのロールおよびロールバインディングがターゲット namespace にコピーされます。

#### 2.4.5.7. コピーされる CSV

OLM は、それぞれの Operator グループのターゲット namespace の Operator グループのすべてのアクティブな CSV のコピーを作成します。コピーされる CSV の目的は、ユーザーに対して、特定の Operator が作成されるリソースを監視するように設定されたターゲット namespace を通知することにあります。

コピーされる CSV にはステータスの理由 **Copied** があり、それらのソース CSV のステータスに一致するように更新されます。**olm.targetNamespaces** アノテーションは、クラスター上でコピーされる CSV が作成される前に取られます。ターゲット namespace 選択を省略すると、テナント間のターゲット namespace の重複が回避されます。

コピーされる CSV はそれらのソース CSV が存在しなくなるか、それらのソース CSV が属する Operator グループが、コピーされた CSV の namespace をターゲットに設定しなくなると削除されます。

**注記**

デフォルトでは、**disableCopiedCSVs** フィールドは無効になっています。**disableCopiedCSVs** フィールドを有効にすると、OLM はクラスター上の既存のコピーされた CSV を削除します。**disableCopiedCSVs** フィールドが無効になると、OLM はコピーされた CSV を再度追加します。

- **disableCopiedCSVs** フィールドを無効にします。

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: false
EOF
```

- **disableCopiedCSVs** フィールドを有効にします。

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: true
EOF
```

#### 2.4.5.8. 静的 Operator グループ

Operator グループはその **spec.staticProvidedAPIs** フィールドが **true** に設定されると **静的** になります。その結果、OLM は Operator グループの **olm.providedAPIs** アノテーションを変更しません。つまり、これを事前に設定することができます。これは、ユーザーが Operator グループを使用して namespace のセットでリソースの競合を防ぐ必要がある場合で、それらのリソースの API を提供するアクティブなメンバーの CSV がない場合に役立ちます。

以下は、**something.cool.io/cluster-monitoring: "true"** アノテーションのあるすべての namespace の **Prometheus** リソースを保護する Operator グループの例です。

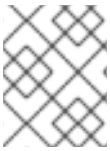
```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```

```
  name: cluster-monitoring
  namespace: cluster-monitoring
  annotations:
    olm.providedAPIs:
Alertmanager.v1.monitoring.coreos.com,Prometheus.v1.monitoring.coreos.com,PrometheusRule.v1.mo
nitoring.coreos.com,ServiceMonitor.v1.monitoring.coreos.com
spec:
  staticProvidedAPIs: true
  selector:
    matchLabels:
      something.cool.io/cluster-monitoring: "true"
```

#### 2.4.5.9. Operator グループの交差部分

2 つの Operator グループは、それらのターゲット namespace セットの交差部分が空のセットではなく、**olm.providedAPIs** アノテーションで定義されるそれらの指定 API セットの交差部分が空のセットではない場合に、**交差部分のある指定 API** があると見なされます。

これによって生じ得る問題として、交差部分のある指定 API を持つ複数の Operator グループは、一連の交差部分のある namespace で同じリソースに関して競合関係になる可能性があります。



**注記**

交差ルールを確認すると、Operator グループの namespace は常に選択されたターゲット namespace の一部として組み込まれます。

##### 2.4.5.9.1. 交差のルール

アクティブメンバーの CSV が同期する際はいつでも、OLM はクラスターで、CSV の Operator グループとそれ以外のすべての間での交差部分のある指定 API のセットをクエリーします。その後、OLM はそのセットが空のセットであるかどうかを確認します。

- **true** であり、CSV の指定 API が Operator グループのサブセットである場合:
  - 移行を継続します。
- **true** であり、CSV の指定 API が Operator グループのサブセット **ではない** 場合:
  - Operator グループが静的である場合:
    - CSV に属するすべてのデプロイメントをクリーンアップします。
    - ステータスの理由 **CannotModifyStaticOperatorGroupProvidedAPIs** のある失敗状態に CSV を移行します。
  - Operator グループが静的 **ではない** 場合:
    - Operator グループの **olm.providedAPIs** アノテーションを、それ自体と CSV の指定 API の集合に置き換えます。
- **false** であり、CSV の指定 API が Operator グループのサブセット **ではない** 場合:
  - CSV に属するすべてのデプロイメントをクリーンアップします。
  - ステータスの理由 **InterOperatorGroupOwnerConflict** のある失敗状態に CSV を移行します。

- **false** であり、CSV の指定 API が Operator グループのサブセットである場合:
  - Operator グループが静的である場合:
    - CSV に属するすべてのデプロイメントをクリーンアップします。
    - ステータスの理由 **CannotModifyStaticOperatorGroupProvidedAPIs** のある失敗状態に CSV を移行します。
  - Operator グループが静的 **ではない** 場合:
    - Operator グループの **olm.providedAPIs** アノテーションを、それ自体と CSV の指定 API 間の差異部分に置き換えます。

**注記**

Operator グループによって生じる失敗状態は非終了状態です。

以下のアクションは、Operator グループが同期するたびに実行されます。

- アクティブメンバーの CSV の指定 API のセットは、クラスターから計算されます。コピーされた CSV は無視されることに注意してください。
- クラスターセットは **olm.providedAPIs** と比較され、**olm.providedAPIs** に追加の API が含まれる場合は、それらの API がプルーニングされます。
- すべての namespace で同じ API を提供するすべての CSV は再びキューに入れられます。これにより、交差部分のあるグループ間の競合する CSV に対して、それらの競合が競合する CSV のサイズ変更または削除のいずれかによって解決されている可能性があることが通知されます。

#### 2.4.5.10. マルチテナント Operator 管理の制限事項

OpenShift Dedicated では、同じクラスターに異なる Operator のバージョンを同時にインストールした場合、サポートが限定的になります。Operator Lifecycle Manager (OLM) は、Operator を異なる namespace に複数回インストールします。その 1 つの制約として、Operator の API バージョンは同じである必要があります。

Operators は、Kubernetes のグローバルリソースである **CustomResourceDefinition** オブジェクト (CRD) を使用するため、コントロールプレーンの拡張機能です。多くの場合、Operator の異なるメジャーバージョンには互換性のない CRD があります。これにより、クラスター上の異なる namespace に同時にインストールするのに互換性がなくなります。

すべてのテナントまたは namespace がクラスターの同じコントロールプレーンを共有します。したがって、マルチテナントクラスター内のテナントはグローバル CRD も共有するため、同じクラスターで同じ Operator の異なるインスタンスを並行して使用できるシナリオが制限されます。

サポートされているシナリオは次のとおりです。

- まったく同じ CRD 定義を提供する異なるバージョンの Operators (バージョン管理された CRD の場合は、まったく同じバージョンのセット)
- CRD を同梱しておらず、代わりにソフトウェアカタログ内の別のバンドルで CRD を提供している異なるバージョンの Operator

他のすべてのシナリオはサポートされていません。これは、異なる Operator バージョンからの複数の競合または重複する CRD が同じクラスター上で調整される場合、クラスターデータの整合性が保証されないためです。

**関連情報**

- マルチテナントクラスター内の Operators

#### 2.4.5.11. Operator グループのトラブルシューティング

##### 2.4.5.11.1. メンバーシップ

- インストールプランの namespace には、Operator グループを 1 つだけ含める必要があります。namespace でクラスターサービスバージョン (CSV) を生成しようとすると、インストールプランでは、以下のシナリオの Operator グループが無効であると見なされます。
  - インストールプランの namespace に Operator グループが存在しない。
  - インストールプランの namespace に複数の Operator グループが存在する。
  - Operator グループに、正しくないサービスアカウント名または存在しないサービスアカウント名が指定されている。

  インストールプランで無効な Operator グループが検出された場合には、CSV は生成されず、**InstallPlan** リソースは関連するメッセージを出力して、インストールを続行します。たとえば、複数の Operator グループが同じ namespace に存在する場合に以下のメッセージが表示されます。

  ```
  attenuated service account query failed - more than one operator group(s) are managing this namespace count=2
  ```

  ここでは、**count=** は、namespace 内の Operator グループの数を指します。

- CSV のインストールモードがその namespace で Operator グループのターゲット namespace 選択をサポートしない場合、CSV は **UnsupportedOperatorGroup** の理由で失敗状態に切り替わります。この理由で失敗した状態にある CSV は、Operator グループのターゲット namespace の選択がサポートされる設定に変更されるか、CSV のインストールモードがターゲット namespace 選択をサポートするように変更される場合に、保留状態に切り替わります。

### 2.4.6. マルチテナント対応と Operator のコロケーション

このガイドでは、Operator Lifecycle Manager (OLM) のマルチテナント対応と Operator のコロケーションを説明します。

#### 2.4.6.1. namespace 内での Operators コロケーション

Operator Lifecycle Manager (OLM) は、同じ namespace にインストールされている OLM 管理 Operators を処理します。つまり、それらの **Subscription** リソースは、関連する Operators として同じ namespace に配置されます。それらが実際には関連していなくても、いずれかが更新されると、OLM はバージョンや更新ポリシーなどの状態を考慮します。

このデフォルトの動作は、次の 2 つの方法で現れます。

- 保留中の更新の **InstallPlan** リソースには、同じ namespace にある他のすべての Operators の **ClusterServiceVersion** (CSV) リソースが含まれます。

- 同じ namespace 内のすべての Operators は、同じ更新ポリシーを共有します。たとえば、1 つの Operator が手動更新に設定されている場合は、他のすべての Operators の更新ポリシーも手動に設定されます。

これらのシナリオは、次の問題につながる可能性があります。

- 更新された Operator だけでなく、より多くのリソースが定義されているため、Operator 更新のインストール計画を推論するのは難しくなります。
- namespace 内の一部の Operator を自動で更新し、他の Operator を手動で更新するという、多くのクラスター管理者が求めていることが不可能になります。

通常、これらの問題が明らかになるのは、OpenShift Dedicated を使用して Operator をインストールするときに、デフォルトの動作により、**All namespaces** インストールモードをサポートする Operator が、デフォルトの **openshift-operators** グローバル namespace にインストールされるためです。

**dedicated-admin** ロールを持つ管理者は、次のワークフローを使用して、このデフォルトの動作を手動で回避できます。

1. Operator をインストールするためのプロジェクトを作成します。
2. すべての namespace を監視する Operator グループである、カスタム **グローバル Operator グループ** を作成します。この Operator グループを作成した namespace に関連付けることで、インストール namespace がグローバル namespace になり、そこにインストールされた Operators がすべての namespace で使用できるようになります。
3. 必要な Operator をインストール namespace にインストールします。

Operator に依存関係がある場合、依存関係は事前に作成された namespace に自動的にインストールされます。その結果、依存関係 Operators が同じ更新ポリシーと共有インストールプランを持つことが有効になります。詳細な手順は、「カスタム namespace へのグローバル Operators のインストール」を参照してください。

**関連情報**

- カスタム namespace にグローバル Operators をインストールする
- マルチテナントクラスター内の Operators

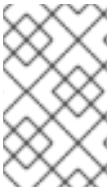
## 2.4.7. Operator 条件

以下では、Operator Lifecycle Manager (OLM) による Operator 条件の使用方法を説明します。

### 2.4.7.1. Operator 条件について

Operator のライフサイクル管理のロールの一部として、Operator Lifecycle Manager (OLM) は、Operator を定義する Kubernetes リソースの状態から Operator の状態を推測します。このアプローチでは、Operator が特定の状態にあることをある程度保証しますが、推測できない情報を Operator が OLM と通信して提供する必要がある場合も多々あります。続いて、OLM がこの情報を使用して、Operator のライフサイクルをより適切に管理することができます。

OLM は、Operators が OLM に条件を通信できる **OperatorCondition** というカスタムリソース定義 (CRD) を提供します。**OperatorCondition** リソースの **Spec.Conditions** 配列にある場合に、OLM による Operator の管理に影響するサポートされる条件のセットがあります。



**注記**

デフォルトでは、**Spec.Conditions** 配列は、ユーザーによって追加されるか、カスタム Operator ロジックの結果として追加されるまで、**OperatorCondition** オブジェクトに存在しません。

#### 2.4.7.2. サポートされる条件

Operator Lifecycle Manager (OLM) は、以下の Operator 条件をサポートします。

##### 2.4.7.2.1. アップグレード可能な条件

**Upgradeable** Operator 条件は、既存のクラスターサービスバージョン (CSV) が、新規の CSV バージョンに置き換えられることを阻止します。この条件は、以下の場合に役に立ちます。

- Operator が重要なプロセスを開始するところで、プロセスが完了するまでアップグレードしてはいけない場合
- Operator が、Operator のアップグレードの準備ができる前に完了する必要のあるカスタムリソース (CR) の移行を実行している場合

**重要**

**Upgradeable** Operator の条件を **False** 値に設定しても、Pod の中断は回避できません。Pod が中断されないようにする必要がある場合は、「追加リソース」セクションの「Pod 中断バジェットを使用して稼働させなければならない Pod の数を指定する」と「グレースフルな終了」を参照してください。

**Upgradeable Operator 条件の例**

```
apiVersion: operators.coreos.com/v1
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
  conditions:
  - type: Upgradeable (1)
    status: "False" (2)
    reason: "migration"
    message: "The Operator is performing a migration."
    lastTransitionTime: "2020-08-24T23:15:55Z"
```

(1) 条件の名前。

(2) **False** 値は、Operator のアップグレードの準備ができていないことを示します。OLM は、Operator の既存の CSV を置き換える CSV が **Pending** フェーズでなくなることを阻止します。**False** 値はクラスターのアップグレードをブロックしません。

#### 2.4.7.3. 関連情報

- Operator 条件の管理

## 2.4.8. Operator Lifecycle Manager メトリクス

### 2.4.8.1. 公開されるメトリクス

Operator Lifecycle Manager (OLM) は、Prometheus ベースの OpenShift Dedicated クラスターモニタリングスタックで使用される特定の OLM 固有のリソースを公開します。

**表2.8 OLM によって公開されるメトリクス**

| 名前 | 説明 |
| --- | --- |
| **catalog_source_count** | カタログソースの数。 |
| **catalogsource_ready** | カタログソースの状態。値 **1** は、カタログソースが **READY** 状態であることを示します。値 **0** は、カタログソースが **READY** 状態ではないことを示します。 |
| **csv_abnormal** | クラスターサービスバージョン (CSV) を調整する際に、(インストールされていない場合など) CSV バージョンが **Succeeded** 以外の状態にあることを表します。**name**、**namespace**、**phase**、**reason**、および **version** ラベルが含まれます。Prometheus アラートは、このメトリクスが存在する場合に作成されます。 |
| **csv_count** | 正常に登録された CSV の数。 |
| **csv_succeeded** | CSV を調整する際に、CSV バージョンが **Succeeded** 状態 (値 **1**) にあるか、そうでないか (値 **0**) を表します。**name**、**namespace**、および **version** ラベルが含まれます。 |
| **csv_upgrade_count** | CSV アップグレードの単調 (monotonic) カウント。 |
| **install_plan_count** | インストール計画の数。 |
| **installplan_warnings_total** | インストール計画に含まれる非推奨のリソースなど、リソースによって生成される警告の個数。 |
| **olm_resolution_duration_seconds** | 依存関係解決の試行期間。 |
| **subscription_count** | サブスクリプションの数。 |
| **subscription_sync_total** | サブスクリプション同期の単調 (monotonic) カウント。**channel**、**installed** CSV、およびサブスクリプション **name** ラベルが含まれます。 |

## 2.4.9. Operator Lifecycle Manager での Webhook の管理

Webhook により、リソースがオブジェクトストアに保存され、Operator コントローラーによって処理される前に、Operator の作成者はリソースのインターセプト、変更、許可、および拒否を実行することができます。Operator Lifecycle Manager (OLM) は、Operator と共に提供される際にこれらの Webhook のライフサイクルを管理できます。

#### 2.4.9.1. 関連情報

- Kubernetes ドキュメント:
  - 検証用の受付 Webhook
  - 変更用の受付 Webhook
  - 変換 Webhook

## 2.5. ソフトウェアカタログの概要

### 2.5.1. ソフトウェアカタログについて

**ソフトウェアカタログ** は、クラスター管理者が Operator を検出してインストールするために使用する OpenShift Dedicated の Web コンソールインターフェイスです。1 回のクリックで、Operator をクラスター外のソースからプルし、クラスター上でインストールおよびサブスクライブして、エンジニアリングチームが Operator Lifecycle Manager (OLM) を使用してデプロイメント環境全体で製品をセルフサービスで管理される状態にすることができます。

クラスター管理者は、以下のカテゴリーにグループ化されたカタログから選択することができます。

| カテゴリー | 説明 |
|---|---|
| Red Hat Operators | Red Hat によってパッケージ化され、出荷される Red Hat 製品。Red Hat によってサポートされます。 |
| Certified Operators | 大手独立系ソフトウェアベンダー (ISV) の製品。Red Hat は ISV とのパートナーシップにより、パッケージ化および出荷を行います。ISV によってサポートされます。 |
| Community Operators | redhat-openshift-ecosystem/community-operators-prod/operators GitHub リポジトリーで関連する担当者によって保守されているオプションで表示可能なソフトウェア。正式なサポートはありません。 |
| Custom Operators | 各自でクラスターに追加する Operator。カスタム Operator を追加していない場合、Web コンソールソフトウェアカタログに **Custom** カテゴリーは表示されません。 |

ソフトウェアカタログ内の Operator は、OLM 上で実行できるようにパッケージ化されています。これには、Operator のインストールおよびセキュアな実行に必要なすべての CRD、RBAC ルール、デプロイメント、およびコンテナーイメージが含まれるクラスターサービスバージョン (CSV) という YAML ファイルが含まれます。また、機能の詳細やサポートされる Kubernetes バージョンなどのユーザーに表示される情報も含まれます。

### 2.5.2. ソフトウェアカタログのアーキテクチャー

OpenShift Dedicated 上のソフトウェアカタログ UI コンポーネントは、デフォルトでは **openshift-marketplace** namespace の Marketplace Operator によって動作しています。

#### 2.5.2.1. OperatorHub カスタムリソース

Marketplace Operator は、ソフトウェアカタログで提供されるデフォルトの **CatalogSource** オブジェクトを管理する、**cluster** という名前の **OperatorHub** カスタムリソース (CR) を管理します。

### 2.5.3. 関連情報

- カタログソース
- OLM での Operator のインストールおよびアップグレードのワークフロー
- Red Hat Partner Connect

## 2.6. RED HAT が提供する OPERATOR カタログ

Red Hat は、OpenShift Dedicated にデフォルトで含まれる複数の Operator カタログを提供します。

**重要**

OpenShift Dedicated 4.11 以降、デフォルトの Red Hat 提供の Operator カタログはファイルベースのカタログ形式でリリースされます。OpenShift Dedicated 4.6 から 4.10 までの Red Hat が提供するデフォルトの Operator カタログは、非推奨の SQLite データベース形式でリリースされました。

**opm** サブコマンド、フラグ、および SQLite データベース形式に関連する機能も非推奨となり、今後のリリースで削除されます。機能は引き続きサポートされており、非推奨の SQLite データベース形式を使用するカタログに使用する必要があります。

**opm index prune** などの SQLite データベース形式を使用する **opm** サブコマンドおよびフラグの多くは、ファイルベースのカタログ形式では機能しません。ファイルベースのカタログを使用する方法の詳細は、カスタムカタログの管理 および Operator Framework パッケージ形式 を参照してください。

### 2.6.1. Operator カタログについて

Operator カタログは、Operator Lifecycle Manager (OLM) がクエリーを行い、Operators およびそれらの依存関係をクラスターで検出し、インストールできるメタデータのリポジトリーです。OLM は最新バージョンのカタログから Operators を常にインストールします。

Operator Bundle Format に基づくインデックスイメージは、カタログのコンテナー化されたスナップショットです。これは、Operator マニフェストコンテンツのセットへのポインターのデータベースが含まれるイミュータブルなアーティファクトです。カタログはインデックスイメージを参照し、クラスター上の OLM のコンテンツを調達できます。

カタログが更新されると、Operators の最新バージョンが変更され、それ以前のバージョンが削除または変更される可能性があります。また、制限されたネットワーク環境の OpenShift Dedicated クラスター上で OLM が実行されている場合、インターネットからカタログに直接アクセスして最新のコンテンツをプルすることはできません。

クラスター管理者は、Red Hat が提供するカタログをベースとして使用して、またはゼロから独自のカスタムインデックスイメージを作成できます。これを使用して、クラスターのカタログコンテンツを調達できます。独自のインデックスイメージの作成および更新により、クラスターで利用可能な Operators のセットをカスタマイズする方法が提供され、また前述のネットワークが制限された環境の問題を回避することができます。

**重要**

Kubernetes は定期的に特定の API を非推奨とし、後続のリリースで削除します。そのため、OpenShift Dedicated のバージョンで、API が削除された Kubernetes バージョンが採用されると、Operator がその API を使用できなくなります。

**注記**

Operator のレガシー **パッケージマニフェスト形式** (レガシー形式を使用していたカスタムカタログを含む) のサポートは、OpenShift Dedicated 4.8 以降で削除されました。

カスタムカタログイメージを作成する場合、OpenShift Dedicated 4 の以前のバージョンでは **oc adm catalog build** コマンドを使用する必要がありました。このコマンドはいくつかのリリースで非推奨となり、現在は削除されています。OpenShift Dedicated 4.6 以降、Red Hat 提供のインデックスイメージが利用可能になったため、カタログ作成者は **opm index** コマンドを使用してインデックスイメージを管理する必要があります。

**関連情報**

- カスタムカタログの管理
- パッケージ形式

### 2.6.2. Red Hat が提供する Operator カタログについて

Red Hat が提供するカタログソースは、デフォルトで **openshift-marketplace** namespace にインストールされます。これにより、すべての namespace でクラスター全体でカタログを利用できるようになります。

以下の Operator カタログは Red Hat によって提供されます。

| Catalog | インデックスイメージ | 説明 |
|---|---|---|
| **redhat-operators** | **registry.redhat.io/redhat/redhat-operator-index:v4** | Red Hat によってパッケージ化され、出荷される Red Hat 製品。Red Hat によってサポートされます。 |
| **certified-operators** | **registry.redhat.io/redhat/certified-operator-index:v4** | 大手独立系ソフトウェアベンダー (ISV) の製品。Red Hat は ISV とのパートナーシップにより、パッケージ化および出荷を行います。ISV によってサポートされます。 |

| Catalog | インデックスイメージ | 説明 |
| --- | --- | --- |
| **community-operators** | **registry.redhat.io/redhat/community-operator-index:v4** | redhat-openshift-ecosystem/community-operators-prod/operators GitHub リポジトリーで、関連する担当者によって保守されているソフトウェア。正式なサポートはありません。 |

クラスターのアップグレード時に、Red Hat が提供するデフォルトのカタログソースのインデックスイメージのタグは、Operator Lifecycle Manager (OLM) が最新版のカタログをプルするように、Cluster Version Operator (CVO) により自動更新されます。たとえば、OpenShift Dedicated 4.8 から 4.9 へのアップグレード時に、**redhat-operators** カタログの **CatalogSource** オブジェクトの **spec.image** フィールドは次のように更新されます。

```
registry.redhat.io/redhat/redhat-operator-index:v4.8
```

更新後は次のようになります。

```
registry.redhat.io/redhat/redhat-operator-index:v4.9
```

## 2.7. マルチテナントクラスター内の OPERATORS

Operator Lifecycle Manager (OLM) のデフォルトの動作は、Operator のインストール時に簡素化することを目的としています。ただし、この動作は、特にマルチテナントクラスターでは柔軟性に欠ける場合があります。OpenShift Dedicated クラスター上の複数のテナントが Operator を使用するには、OLM のデフォルトの動作では、管理者が Operator を **All namespaces** モードでインストールする必要がありますが、これは最小特権の原則に違反すると考えられます。

以下のシナリオを考慮して、環境と要件に最適な Operator インストールワークフローを決定してください。

**関連情報**

- Common terms: Multitenant
- マルチテナント Operator 管理の制限事項

### 2.7.1. デフォルトの Operator インストールモードと動作

管理者として Web コンソールを使用して Operators をインストールする場合、通常、Operators の機能に応じて、インストールモードに 2 つの選択肢があります。

**単一の namespace**

選択した単一の namespace に Operator をインストールし、Operator が要求するすべての権限をその namespace で使用できるようにします。

**すべての namespace**

デフォルトの **openshift-operators** namespace で Operator をインストールし、クラスターのすべての namespace を監視し、Operator をこれらの namespace に対して利用可能にします。Operator が要求するすべてのアクセス許可をすべての namespace で使用できるようにします。場合によっては、Operator の作成者はメタデータを定義して、その Operator が提案する namespace の 2 番目のオプションをユーザーに提供できます。

この選択は、影響を受ける namespace のユーザーが、namespace でのロールに応じて、所有するカスタムリソース (CR) を活用できる Operators API にアクセスできることも意味します。

- **namespace-admin** および **namespace-edit** ロールは、Operator API の読み取り/書き込みが可能です。つまり、Operator API を使用できます。
- **namespace-view** ロールは、その Operator の CR オブジェクトを読み取ることができます。

**Single namespace** モードの場合、Operator 自体が選択した namespace にインストールされるため、その Pod とサービスアカウントもそこに配置されます。**All namespaces** モードの場合、Operator の権限はすべて自動的にクラスターロールに昇格されます。つまり、Operator はすべての namespace でこれらの権限を持ちます。

**関連情報**

- クラスターへの Operators の追加
- Install modes types

### 2.7.2. マルチテナントクラスターの推奨ソリューション

**Multinamespace** インストールモードは存在しますが、サポートされている Operators はほとんどありません。標準 **All namespaces** と **Single namespace** インストールモードの中間的なソリューションとして、次のワークフローを使用して、テナントごとに 1 つずつ、同じ Operator の複数のインスタンスをインストールできます。

1. テナントの namespace とは別のテナント Operator の namespace を作成します。これは、プロジェクトを作成することで実行できます。
2. テナントの namespace のみを対象とするテナント Operator の Operator グループを作成します。
3. テナント Operator namespace に Operator をインストールします。

その結果、Operator はテナントの Operator namespace に存在し、テナントの namespace を監視しますが、Operator の Pod もそのサービスアカウントも、テナントによって表示または使用できません。

このソリューションは、より優れたテナント分離、リソースの使用を犠牲にした最小特権の原則、および制約が確実に満たされるようにするための追加のオーケストレーションを提供します。詳細な手順は、「マルチテナントクラスター用の Operator の複数インスタンスの準備」を参照してください。

**制限および考慮事項**

このソリューションは、次の制約が満たされている場合にのみ機能します。

- 同じ Operator のすべてのインスタンスは、同じバージョンである必要があります。
- Operator は、他の Operators に依存することはできません。
- Operator は CRD 変換 Webhook を出荷できません。

**重要**

同じクラスターで同じ Operator の異なるバージョンを使用することはできません。最終的に、Operator の別のインスタンスのインストールは、以下の条件を満たす場合にブロックされます。

- インスタンスは Operator の最新バージョンではありません。
- インスタンスは、クラスターですでに使用されている新しいリビジョンに含まれる情報またはバージョンを欠いている CRD の古いリビジョンを出荷します。

**関連情報**

- マルチテナントクラスター用の Operator の複数インスタンスの準備

### 2.7.3. Operator のコロケーションと Operator グループ

Operator Lifecycle Manager (OLM) は、同じ namespace にインストールされている OLM 管理 Operators を処理します。つまり、それらの **Subscription** リソースは、関連する Operators として同じ namespace に配置されます。それらが実際には関連していなくても、いずれかが更新されると、OLM はバージョンや更新ポリシーなどの状態を考慮します。

Operator のコロケーションと Operator グループの効果的な使用の詳細は、Operator Lifecycle Manager (OLM) → マルチテナント対応と Operator のコロケーション を参照してください。

## 2.8. CRD

### 2.8.1. カスタムリソース定義からのリソースの管理

以下では、開発者がカスタムリソース定義 (CRD) にあるカスタムリソース (CR) をどのように管理できるかを説明します。

#### 2.8.1.1. カスタムリソース定義

Kubernetes API では、**リソース** は特定の種類の API オブジェクトのコレクションを保管するエンドポイントです。たとえば、ビルトインされた **Pods** リソースには、**Pod** オブジェクトのコレクションが含まれます。

**カスタムリソース定義** (CRD) オブジェクトは、クラスター内に新規の固有オブジェクト **kind** を定義し、Kubernetes API サーバーにそのライフサイクル全体を処理させます。

**カスタムリソース** (CR) オブジェクトは、クラスター管理者によってクラスターに追加された CRD から作成され、すべてのクラスターユーザーが新規リソースタイプをプロジェクトに追加できるようにします。

Operators はとりわけ CRD を必要な RBAC ポリシーおよび他のソフトウェア固有のロジックでパッケージ化することで CRD を利用します。

#### 2.8.1.2. ファイルからのカスタムリソースの作成

カスタムリソース定義 (CRD) がクラスターに追加された後に、クラスターリソース (CR) は CR 仕様を使用するファイルを使って CLI で作成できます。

**手順**

1. CR の YAML ファイルを作成します。以下の定義例では、**cronSpec** と **image** のカスタムフィールドが **Kind: CronTab** の CR に設定されます。この **Kind** は、CRD オブジェクトの **spec.kind** フィールドから取得されます。

   **CR の YAML ファイルサンプル**

   ```
   apiVersion: "stable.example.com/v1" 1
   kind: CronTab 2
   metadata:
     name: my-new-cron-object 3
     finalizers: 4
     - finalizer.stable.example.com
   spec: 5
     cronSpec: "* * * * /5"
     image: my-awesome-cron-image
   ```

   1. CRD からグループ名および API バージョン (name/version) を指定します。
   2. CRD にタイプを指定します。
   3. オブジェクトの名前を指定します。
   4. オブジェクトの ファイナライザー を指定します (ある場合)。ファイナライザーは、コントローラーがオブジェクトの削除前に完了する必要のある条件を実装できるようにします。
   5. オブジェクトのタイプに固有の条件を指定します。

2. ファイルの作成後に、オブジェクトを作成します。

   ```
   $ oc create -f <file_name>.yaml
   ```

### 2.8.1.3. カスタムリソースの検査

CLI を使用してクラスターに存在するカスタムリソース (CR) オブジェクトを検査できます。

**前提条件**

- CR オブジェクトがアクセスできる namespace にあること。

**手順**

1. CR の特定の kind に関する情報を取得するには、以下を実行します。

   ```
   $ oc get <kind>
   ```

   以下に例を示します。

   ```
   $ oc get crontab
   ```

   **出力例**

```
NAME                 KIND
my-new-cron-object   CronTab.v1.stable.example.com
```

リソース名では大文字と小文字が区別されず、CRD で定義される単数形または複数形のいずれか、および任意の短縮名を指定できます。以下に例を示します。

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. CR の未加工の YAML データを確認することもできます。

```
$ oc get <kind> -o yaml
```

以下に例を示します。

```
$ oc get ct -o yaml
```

**出力例**

```
apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-new-cron-object
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

1 2 オブジェクトの作成に使用した YAML からのカスタムデータが表示されます。

# 第3章 ユーザータスク

## 3.1. インストールされた OPERATOR からのアプリケーションの作成

以下では、開発者を対象に、OpenShift Dedicated Web コンソールを使用して、インストールされた Operator からアプリケーションを作成する例を示します。

### 3.1.1. Operator を使用した etcd クラスターの作成

この手順では、Operator Lifecycle Manager (OLM) で管理される etcd Operator を使用した新規 etcd クラスターの作成を説明します。

**前提条件**

- OpenShift Dedicated クラスターへのアクセス。
- 管理者によってクラスター全体に etcd Operator がすでにインストールされている。

**手順**

1. この手順を実行するために OpenShift Dedicated Web コンソールで新規プロジェクトを作成します。この例では、**my-etcd** というプロジェクトを使用します。
2. **Ecosystem → Installed Operators** ページに移動します。このページには、dedicated-admin によってクラスターにインストールされた使用可能な Operators が、クラスターサービスバージョン (CSV) のリストの形で表示されます。CSV は Operator によって提供されるソフトウェアを起動し、管理するために使用されます。

   **ヒント**

   以下を使用して、CLI でこのリストを取得できます。

   ```
   $ oc get csv
   ```

3. **Installed Operators** ページで、etcd Operator をクリックして詳細情報および選択可能なアクションを表示します。
   **Provided APIs** に表示されているように、この Operator は 3 つの新規リソースタイプを利用可能にします。これには、**etcd クラスター** (**EtcdCluster** リソース) のタイプが含まれます。これらのオブジェクトは、**Deployment** または **ReplicaSet** などの組み込み済みのネイティブ Kubernetes オブジェクトと同様に機能しますが、これらには etcd を管理するための固有のロジックが含まれます。
4. 新規 etcd クラスターを作成します。
   a. **etcd Cluster** API ボックスで、**Create instance** をクリックします。
   b. 次のページでは、**EtcdCluster** オブジェクト (クラスターのサイズなど) のテンプレートを起動する最小条件を変更できます。ここでは **Create** をクリックして確定します。これにより、Operator がトリガーされ、Pod、サービス、および新規 etcd クラスターの他のコンポーネントが起動します。
5. **example** etcd クラスター、**Resources** タブの順にクリックし、Operator が自動的に作成および設定した多数のリソースが含まれていることを確認します。

Kubernetes サービスが作成され、プロジェクトの他の Pod からデータベースにアクセスできることを確認します。

6. 所定プロジェクトで **edit** ロールを持つすべてのユーザーは、クラウドサービスのようにセルフサービス方式でプロジェクトにすでに作成されている Operators によって管理されるアプリケーションのインスタンス (この例では etcd クラスター) を作成し、管理し、削除することができます。この機能を持つ追加のユーザーを有効にする必要がある場合、プロジェクト管理者は以下のコマンドを使用してこのロールを追加できます。

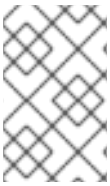
```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

これで、etcd クラスターは Pod が正常でなくなったり、クラスターのノード間で移行する際の障害に対応し、データのリバランスを行います。最も重要なことは、適切なアクセス権を持つ dedicated-admin または開発者が、アプリケーションでデータベースを簡単に使用できるようになった点です。

# 第4章 管理者タスク

## 4.1. OPERATOR のクラスターへの追加

**dedicated-admin** ロールを持つ管理者は、Operator Lifecycle Manager (OLM) を使用して、OLM ベースの Operator を OpenShift Dedicated クラスターにインストールできます。



**注記**

OLM が同一 namespace に配置されたインストール済み Operator の更新を処理する方法や、カスタムグローバル Operator グループで Operator をインストールする別の方法は、マルチテナント対応と Operator のコロケーション を参照してください。

### 4.1.1. ソフトウェアカタログからの Operator のインストールについて

ソフトウェアカタログは、Operator を検索するためのユーザーインターフェイスです。これは、クラスター上で Operator をインストールして管理する Operator Lifecycle Manager (OLM) と連携して動作します。

**dedicated-admin** として、OpenShift Dedicated Web コンソールまたは CLI を使用して、ソフトウェアカタログから Operator をインストールできます。Operator を 1 つまたは複数の namespace にサブスクライブし、Operator をクラスター上で開発者が使用できるようにできます。

インストール時に、Operator の以下の初期設定を判別する必要があります。

**インストールモード**

**All namespaces on the cluster (default)**を選択して Operator をすべての namespace にインストールするか、(利用可能な場合は) 個別の namespace を選択し、選択された namespace のみに Operator をインストールします。この例では、**All namespaces...** を選択し、Operator をすべてのユーザーおよびプロジェクトで利用可能にします。

**更新チャネル**

Operator が複数のチャネルで利用可能な場合、サブスクライブするチャネルを選択できます。たとえば、(利用可能な場合に) **stable** チャネルからデプロイするには、これをリストから選択します。

**承認ストラテジー**

自動 (Automatic) または手動 (Manual) のいずれかの更新を選択します。
インストールされた Operator に自動更新を選択する場合、Operator の新規バージョンが選択されたチャネルで利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。

手動更新を選択した場合、新しいバージョンの Operator が利用可能になると、OLM によって更新要求が作成されます。その後、**dedicated-admin** として、その更新リクエストを手動で承認して、Operator を新しいバージョンに更新する必要があります。

**関連情報**

- ソフトウェアカタログの概要

### 4.1.2. Web コンソールを使用してソフトウェアカタログからインストールする

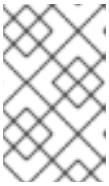
OpenShift Dedicated Web コンソールを使用して、ソフトウェアカタログから Operator をインストールし、サブスクライブできます。

**前提条件**

- **dedicated-admin** ロールを持つアカウントを使用して、OpenShift Dedicated クラスターにアクセスできる。

**手順**

1. Web コンソールで、**Ecosystem → Software Catalog** ページに移動します。
2. スクロールするか、キーワードを **Filter by keyword** ボックスに入力し、必要な Operator を見つけます。たとえば、Advanced Cluster Management for Kubernetes Operator を検索するには **advanced** を入力します。
   また、**インフラストラクチャー機能** でオプションをフィルターすることもできます。たとえば、非接続環境 (ネットワークが制限された環境ともしても知られる) で機能する Operators を表示するには、**Disconnected** を選択します。
3. Operator を選択して、追加情報を表示します。



**注記**

コミュニティー Operator を選択すると、Red Hat がコミュニティー Operator を認定していないことを警告します。続行する前に警告を確認する必要があります。

4. Operator の情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで、Operator のインストールを設定します。
   a. 特定のバージョンの Operator をインストールする場合は、リストから **Update channel** と **Version** を選択します。Operator のすべてのチャネルから Operator のさまざまなバージョンを参照し、そのチャネルとバージョンのメタデータを表示して、インストールする正確なバージョンを選択できます。

**注記**

バージョン選択のデフォルトは、選択したチャネルの最新バージョンです。チャネルの最新バージョンが選択されている場合は、**自動** 承認戦略がデフォルトで有効になります。それ以外の場合、選択したチャネルの最新バージョンをインストールしない場合は、**手動** による承認が必要です。

**手動** 承認を使用して Operator をインストールすると、namespace 内にインストールされたすべての Operators が **手動** 承認戦略で機能し、すべての Operators が一緒に更新されます。Operators を個別に更新する場合は、Operators を別の namespace にインストールします。

   b. Operator のインストールモードを確認します。
      - **All namespaces on the cluster (default)**は、デフォルトの **openshift-operators** namespace で Operator をインストールし、クラスターのすべての namespace を監視し、Operator をこれらの namespace に対して利用可能にします。このオプションは常に選択可能です。
      - **A specific namespace on the cluster**では、Operator をインストールする特定の単一 namespace を選択できます。Operator は監視のみを実行し、この単一 namespace で使用されるように利用可能になります。

c. トークン認証が有効になっているクラウドプロバイダー上のクラスターの場合:

- クラスターで AWS Security Token Service (Web コンソールの **STS Mode**) を使用する場合は、**role ARN** フィールドに、サービスアカウントの AWS IAM ロールの Amazon Resource Name (ARN) を入力します。ロールの ARN を作成するには、AWS アカウントの準備 で説明されている手順に従います。
- クラスターで Microsoft Entra Workload ID (Web コンソールの **Workload Identity / Federated Identity Mode**) を使用する場合は、適切なフィールドに、クライアント ID、テナント ID、サブスクリプション ID を追加します。
- クラスターで Google Cloud Platform Workload Identity (Web コンソールの **GCP Workload Identity / Federated Identity Mode**) を使用する場合は、適切なフィールドに、プロジェクト番号、プール ID、プロバイダー ID、サービスアカウントのメールアドレスを追加します。

d. **Update approval** で、承認ストラテジー **Automatic** または **Manual** を選択します。

**重要**

Web コンソールに、クラスターが AWS STS、Microsoft Entra Workload ID、または GCP Workload Identity を使用していることが示されている場合は、**Update approval** を **Manual** に設定する必要があります。

更新の自動承認を使用したサブスクリプションは推奨されません。更新前に権限の変更が必要な場合があるためです。更新の手動承認を使用したサブスクリプションであれば、管理者が新しいバージョンの権限を確認し、必要な手順を実行してから更新できます。

6. **Install** をクリックし、Operator をこの OpenShift Dedicated クラスターの選択した namespace で利用可能にします。

a. **手動** の承認ストラテジーを選択している場合、サブスクリプションのアップグレードステータスは、そのインストール計画を確認し、承認するまで **Upgrading** のままになります。
**Install Plan** ページでの承認後に、サブスクリプションのアップグレードステータスは **Up to date** に移行します。

b. **自動** の承認ストラテジーを選択している場合、アップグレードステータスは、介入なしに **Up to date** に解決するはずです。

**検証**

- サブスクリプションのアップグレードステータスが **Up to date** になったら、**Ecosystem → Installed Operator** を選択して、インストールされた Operator のクラスターサービスバージョン (CSV) が最終的に表示されることを確認します。**Status** は、関連する namespace で最終的に **Succeeded** に解決されるはずです。

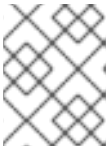
**注記**

**All namespaces...** インストールモードの場合、ステータスは **openshift-operators** namespace で **Succeeded** になりますが、他の namespace でチェックする場合、ステータスは **Copied** になります。

上記通りにならない場合、以下を実行します。

- さらにトラブルシューティングを行うために問題を報告している **Workloads → Pods** ページで、**openshift-operators** プロジェクト (または **A specific namespace...** インストールモードが選択されている場合は他の関連の namespace) の Pod のログを確認します。

- Operator をインストールすると、メタデータに、インストールされているチャネルとバージョンが示されます。



**注記**

ドロップダウンメニュー **Channel** および **Version** は、このカタログコンテキストで他のバージョンのメタデータを表示するために引き続き使用できます。

**関連情報**

- 保留中の Operator 更新の手動による承認

### 4.1.3. CLI を使用してソフトウェアカタログからインストールする

OpenShift Dedicated Web コンソールを使用する代わりに、CLI を使用してソフトウェアカタログから Operator をインストールできます。**oc** コマンドを使用して、**Subscription** オブジェクトを作成または更新します。

**SingleNamespace** インストールモードの場合は、関連する namespace に適切な Operator グループが存在することも確認する必要があります。**OperatorGroup** で定義される Operator グループは、Operator グループと同じ namespace 内のすべての Operators に必要な RBAC アクセスを生成するターゲット namespace を選択します。

**ヒント**

ほとんどの場合は、この手順の Web コンソール方式が推奨されます。これは、**SingleNamespace** モードを選択したときに **OperatorGroup** オブジェクトおよび **Subscription** オブジェクトの作成を自動的に処理するなど、バックグラウンドでタスクが自動化されるためです。

**前提条件**

- **dedicated-admin** ロールを持つアカウントを使用して、OpenShift Dedicated クラスターにアクセスできる。

- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. ソフトウェアカタログからクラスターで使用可能な Operator のリストを表示します。

```
$ oc get packagemanifests -n openshift-marketplace
```

**例4.1 出力例**

```
NAME                              CATALOG               AGE
3scale-operator                   Red Hat Operators     91m
advanced-cluster-management       Red Hat Operators     91m
amq7-cert-manager                 Red Hat Operators     91m
# ...
```

```
couchbase-enterprise-certified     Certified Operators   91m
crunchy-postgres-operator          Certified Operators   91m
mongodb-enterprise                 Certified Operators   91m
# ...
etcd                          Community Operators   91m
jaeger                        Community Operators   91m
kubefed                       Community Operators   91m
# ...
```

必要な Operator のカタログをメモします。

2. 必要な Operator を検査して、サポートされるインストールモードおよび利用可能なチャネルを確認します。

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

**例4.2 出力例**

```
# ...
Kind:         PackageManifest
# ...
      Install Modes: (1)
        Supported:  true
        Type:       OwnNamespace
        Supported:  true
        Type:       SingleNamespace
        Supported:  false
        Type:       MultiNamespace
        Supported:  true
        Type:       AllNamespaces
# ...
    Entries:
      Name:       example-operator.v3.7.11
      Version:    3.7.11
      Name:       example-operator.v3.7.10
      Version:    3.7.10
    Name:         stable-3.7 (2)
# ...
  Entries:
    Name:         example-operator.v3.8.5
    Version:      3.8.5
    Name:         example-operator.v3.8.4
    Version:      3.8.4
  Name:           stable-3.8 (3)
 Default Channel:  stable-3.8 (4)
```

(1) サポートされているインストールモードを示します。

(2) (3) チャネル名の例。

(4) 指定されていない場合にデフォルトで選択されるチャネル。

**ヒント**

次のコマンドを実行すると、Operator のバージョンとチャネル情報を YAML 形式で出力できます。

```
$ oc get packagemanifests <operator_name> -n <catalog_namespace> -o yaml
```

3. namespace に複数のカタログがインストールされている場合は、次のコマンドを実行して、特定のカタログから Operator の使用可能なバージョンとチャネルを検索します。

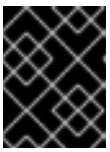
```
$ oc get packagemanifest \
  --selector=catalog=<catalogsource_name> \
  --field-selector metadata.name=<operator_name> \
  -n <catalog_namespace> -o yaml
```

**重要**

Operator のカタログを指定しない場合、**oc get packagemanifest** および **oc describe packagemanifest** コマンドを実行すると、次の条件が満たされると予期しないカタログからパッケージが返される可能性があります。

- 複数のカタログが同じ namespace にインストールされます。
- カタログには、同じ Operators、または同じ名前の Operators が含まれています。

4. インストールする Operator が **AllNamespaces** インストールモードをサポートしており、このモードを使用することを選択した場合は、**openshift-operators** namespace に **global-operators** と呼ばれる適切な Operator グループがデフォルトですでに配置されているため、この手順をスキップしてください。
インストールする Operator が **SingleNamespace** インストールモードをサポートしており、このモードを使用することを選択した場合は、関連する namespace に適切な Operator グループが存在することを確認する必要があります。存在しない場合は、次の手順に従って作成できます。



**重要**

namespace ごとに Operator グループを 1 つだけ持つことができます。詳細は、「Operator グループ」を参照してください。

a. **SingleNamespace** インストールモード用に、**OperatorGroup** オブジェクト YAML ファイル (例: **operatorgroup.yaml**) を作成します。

**SingleNamespace インストールモードの OperatorGroup オブジェクトの例**

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace> 1
```

```
spec:
  targetNamespaces:
  - <namespace> 2
```

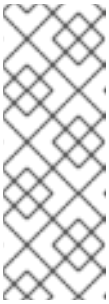
1 2 **SingleNamespace** インストールモードの場合は、**metadata.namespace** フィールドと **spec.targetNamespaces** フィールドの両方に同じ **<namespace>** 値を使用します。

b. **OperatorGroup** オブジェクトを作成します。

```
$ oc apply -f operatorgroup.yaml
```

5. namespace を Operator にサブスクライブするための **Subscription** オブジェクトを作成します。

a. **Subscription** オブジェクトの YAML ファイル (例: **subscription.yaml**) を作成します。



**注記**

特定のバージョンの Operator をサブスクライブする場合は、**startingCSV** フィールドを目的のバージョンに設定し、**installPlanApproval** フィールドを **Manual** に設定して、カタログに新しいバージョンが存在する場合に Operator が自動的にアップグレードされないようにします。詳細は、次の「特定の開始 Operator バージョンを持つ **Subscription** オブジェクトの例」を参照してください。

**例4.3 Subscription オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: <namespace_per_install_mode> 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: <catalog_name> 4
  sourceNamespace: <catalog_source_namespace> 5
  config:
    env: 6
    - name: ARGS
      value: "-v=10"
    envFrom: 7
    - secretRef:
        name: license-secret
    volumes: 8
    - name: <volume_name>
      configMap:
        name: <configmap_name>
    volumeMounts: 9
    - mountPath: <directory_name>
      name: <volume_name>
```

```
  tolerations: (10)
  - operator: "Exists"
  resources: (11)
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: (12)
    foo: bar
```

1. デフォルトの **AllNamespaces** インストールモードの使用は、**openshift-operators** namespace を指定します。カスタムグローバル namespace を作成している場合はこれを指定できます。**SingleNamespace** インストールモードを使用する場合は、関連する単一の namespace を指定します。
2. サブスクライブするチャネルの名前。
3. サブスクライブする Operator の名前。
4. Operator を提供するカタログソースの名前。
5. カタログソースの namespace。デフォルトのソフトウェアカタログソースには **openshift-marketplace** を使用します。
6. **env** パラメーターは、OLM によって作成される Pod のすべてのコンテナーに存在する必要がある環境変数の一覧を定義します。
7. **envFrom** パラメーターは、コンテナーの環境変数に反映するためのソースの一覧を定義します。
8. **volumes** パラメーターは、OLM によって作成される Pod に存在する必要があるボリュームの一覧を定義します。
9. **volumeMounts** パラメーターは、OLM によって作成される Pod のすべてのコンテナーに存在する必要があるボリュームマウントの一覧を定義します。存在しない **volume** を **volumeMount** が参照すると、OLM が Operator のデプロイに失敗します。
10. **tolerations** パラメーターは、OLM によって作成される Pod の toleration の一覧を定義します。
11. **resources** パラメーターは、OLM によって作成される Pod のすべてのコンテナーのリソース制約を定義します。
12. **nodeSelector** パラメーターは、OLM によって作成される Pod の **NodeSelector** を定義します。

**例4.4 特定の開始 Operator バージョンを持つSubscription オブジェクトの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: example-operator
  namespace: example-operator
spec:
  channel: stable-3.7
  installPlanApproval: Manual (1)
  name: example-operator
  source: custom-operators
  sourceNamespace: openshift-marketplace
  startingCSV: example-operator.v3.7.10 (2)
```

(1) 指定したバージョンがカタログの新しいバージョンに置き換えられる場合に備えて、承認ストラテジーを **Manual** に設定します。これにより、新しいバージョンへの自動アップグレードが阻止され、最初の CSV のインストールが完了する前に手動での承認が必要となります。

(2) Operator CSV の特定バージョンを設定します。

b. Amazon Web Services (AWS) Security Token Service (STS)、Microsoft Entra Workload ID、Google Cloud Platform Workload Identity など、トークン認証が有効なクラウドプロバイダー上のクラスターの場合は、次の手順に従って **Subscription** オブジェクトを設定します。

i. **Subscription** オブジェクトが手動更新承認に設定されていることを確認します。

**例4.5 更新の手動承認を使用した Subscription オブジェクトの例**

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual (1)
```

(1) 更新の自動承認を使用したサブスクリプションは推奨されません。更新前に権限の変更が必要な場合があるためです。更新の手動承認を使用したサブスクリプションであれば、管理者が新しいバージョンの権限を確認し、必要な手順を実行してから更新できます。

ii. 関連するクラウドプロバイダー固有のフィールドを **Subscription** オブジェクトの **config** セクションに含めます。
クラスターが AWS STS モードの場合は、次のフィールドを含めます。

**例4.6 AWS STS の変数を使用した Subscription オブジェクトの例**

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" (1)
```

1 ロール ARN の詳細を含めます。

クラスターが Workload ID モードの場合は、次のフィールドを含めます。

**例4.7 Workload ID の変数を使用したSubscription オブジェクトの例**

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" 1
   - name: TENANTID
     value: "<tenant_id>" 2
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" 3
```

1 クライアント ID を含めます。

2 テナント ID を含めます。

3 サブスクリプション ID を含めます。

クラスターが GCP Workload Identity モードの場合は、次のフィールドを含めます。

**例4.8 GCP Workload Identity の変数を使用したSubscription オブジェクトの例**

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: AUDIENCE
     value: "<audience_url>" 1
   - name: SERVICE_ACCOUNT_EMAIL
     value: "<service_account_email>" 2
```

ここでは、以下のようになります。

**<audience>**

**AUDIENCE** 値は、管理者が GCP Workload Identity を設定するときに Google Cloud で作成し、次の形式で事前にフォーマットした URL である必要があります。

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

**<service_account_email>**

**SERVICE_ACCOUNT_EMAIL** 値は、Operator 操作中に権限を借用する Google Cloud サービスアカウントのメールアドレスです。次に例を示します。

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

c. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

```
$ oc apply -f subscription.yaml
```

6. **installPlanApproval** フィールドを **Manual** に設定する場合は、保留中のインストールプランを手動で承認して Operator のインストールを完了します。詳細は、「保留中の Operator 更新の手動による承認」を参照してください。

この時点で、OLM は選択した Operator を認識します。Operator のクラスターサービスバージョン (CSV) はターゲット namespace に表示され、Operator で指定される API は作成用に利用可能になります。

**検証**

1. 次のコマンドを実行して、インストールされている Operator の **Subscription** オブジェクトのステータスを確認します。

```
$ oc describe subscription <subscription_name> -n <namespace>
```

2. **SingleNamespace** インストールモードの Operator グループを作成した場合は、次のコマンドを実行して **OperatorGroup** オブジェクトのステータスを確認します。

```
$ oc describe operatorgroup <operatorgroup_name> -n <namespace>
```

**関連情報**

- Operator グループについて
- カスタム namespace にグローバル Operators をインストールする
- 保留中の Operator 更新の手動による承認

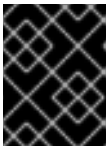
### 4.1.4. マルチテナントクラスター用の Operator の複数インスタンスの準備

**dedicated-admin** ロールを持つ管理者は、マルチテナントクラスターで使用する Operator のインスタンスを複数追加できます。これは、最小特権の原則に違反していると見なされる標準の **All namespaces** インストールモード、または広く採用されていない **Multinamespace** モードのいずれかを使用する代替ソリューションです。詳細は、「マルチテナントクラスター内の Operators」を参照してください。

次の手順では、**テナント** は、デプロイされた一連のワークロードに対する共通のアクセス権と特権を共有するユーザーまたはユーザーのグループです。**テナント Operator** は、そのテナントのみによる使用を意図した Operator のインスタンスです。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- インストールする Operator のすべてのインスタンスは、特定のクラスター全体で同じバージョンである必要があります。



**重要**

この制限およびその他の制限の詳細は、「マルチテナントクラスター内の Operators」を参照してください。

**手順**

1. Operator をインストールする前に、テナントの namespace とは別のテナント Operator の namespace を作成します。これは、プロジェクトを作成することで実行できます。たとえば、テナントの namespace が **team1** の場合、**team1-operator** プロジェクトを作成します。

```
$ oc new-project team1-operator
```

2. **spec.targetNamespaces** リストにその 1 つの namespace エントリーのみを使用して、テナントの namespace をスコープとするテナント Operator の Operator グループを作成します。

   a. **OperatorGroup** リソースを定義し、YAML ファイル (例: **team1-operatorgroup.yaml**) を保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: team1-operatorgroup
  namespace: team1-operator
spec:
  targetNamespaces:
  - team1 1
```

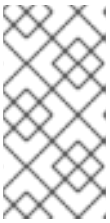
1 1 **spec.targetNamespaces** リストでテナントの namespace のみを定義します。

   b. 以下のコマンドを実行して Operator グループを作成します。

```
$ oc create -f team1-operatorgroup.yaml
```

**次のステップ**

- テナント Operator namespace に Operator をインストールします。このタスクは、CLI ではなく Web コンソールでソフトウェアカタログを使用するとより簡単に実行できます。詳細な手順は、「Web コンソールを使用したソフトウェアカタログからのインストール」を参照してください。



**注記**

Operator のインストールが完了すると、Operator はテナントの Operator namespace に存在し、テナントの namespace を監視しますが、Operator の Pod もそのサービスアカウントも、テナントによって表示または使用されません。

**関連情報**

- マルチテナントクラスター内の Operators

### 4.1.5. カスタム namespace にグローバル Operator をインストールする

OpenShift Dedicated を使用して Operator をインストールする場合、デフォルトの動作では、**All namespaces** インストールモードをサポートする Operator がデフォルトの **openshift-operators** グローバル namespace にインストールされます。これにより、namespace 内のすべての Operator 間で共有インストールプランと更新ポリシーに関連する問題が発生する可能性があります。これらの制限の詳細は、「マルチテナント対応と Operator のコロケーション」を参照してください。

**dedicated-admin** ロールを持つ管理者は、カスタムのグローバル namespace を作成し、その namespace を使用して個別の Operators またはスコープ指定した Operators のセットとその依存関係をインストールすることで、このデフォルトの動作を手動で回避できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

**手順**

1. Operator をインストールする前に、目的の Operator をインストールするための namespace を作成します。これは、プロジェクトを作成することで実行できます。このプロジェクトの namespace は、カスタムのグローバル namespace になります。

```
$ oc new-project global-operators
```

2. すべての namespace を監視する Operator グループである、カスタム **global Operator group** を作成します。

   a. **OperatorGroup** リソースを定義し、**global-operatorgroup.yaml** などの YAML ファイルを保存します。**spec.selector** フィールドと **spec.targetNamespaces** フィールドの両方を省略して、すべての namespace を選択する **global Operator group** にします。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: global-operatorgroup
  namespace: global-operators
```

**注記**

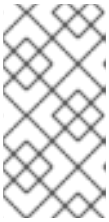
作成されたグローバル Operator グループの **status.namespaces** には、空の文字列 ("") が含まれています。これは、すべての namespace を監視する必要があることを消費する Operator に通知します。

   b. 以下のコマンドを実行して Operator グループを作成します。

```
$ oc create -f global-operatorgroup.yaml
```

**次のステップ**

- 必要な Operator をカスタムのグローバル namespace にインストールします。Web コンソールでは、Operator のインストール時に、カスタムのグローバル namespace が **Installed Namespace** メニューに追加されません。そのため、このインストールタスクを実行するには、OpenShift CLI (**oc**) を使用する必要があります。詳細なインストール手順は、「CLI を使用して OperatorHub からインストールする」を参照してください。



**注記**

Operator のインストールを開始すると、Operator に依存関係がある場合、その依存関係もカスタムグローバル namespace に自動的にインストールされます。その結果、依存関係 Operators が同じ更新ポリシーと共有インストールプランを持つことが有効になります。

**関連情報**

- マルチテナント対応と Operator のコロケーション

### 4.1.6. Operator ワークロードの Pod の配置

デフォルトで、Operator Lifecycle Manager (OLM) は、Operator のインストールまたはオペランドのワークロードのデプロイ時に Pod を任意のワーカーノードに配置します。管理者は、ノードセレクター、taint、および toleration の組み合わせを持つプロジェクトを使用して、Operators およびオペランドの特定のノードへの配置を制御できます。

Operator およびオペランドワークロードの Pod 配置の制御には以下の前提条件があります。

1. 要件に応じて Pod のターゲットとするノードまたはノードのセットを判別します。利用可能な場合は、単数または複数のノードを特定する **node-role.kubernetes.io/app** などの既存ラベルをメモします。それ以外の場合は、コンピュートマシンセットを使用するか、ノードを直接編集して、**myoperator** などのラベルを追加します。このラベルは、後のステップでプロジェクトのノードセレクターとして使用します。

2. 関連しないワークロードを他のノードに向けつつ、特定のラベルの付いた Pod のみがノードで実行されるようにする必要がある場合、コンピュートマシンセットを使用するか、ノードを直接編集して taint をノードに追加します。taint に一致しない新規 Pod がノードにスケジュールされないようにする effect を使用します。たとえば、**myoperator:NoSchedule** taint は、taint に一致しない新規 Pod がノードにスケジュールされないようにしますが、ノードの既存 Pod はそのまま残ります。

3. デフォルトのノードセレクターで設定され、taint を追加している場合に一致する toleration を持つプロジェクトを作成します。

この時点で、作成したプロジェクトでは、以下のシナリオの場合に指定されたノードに Pod を導くことができます。

**Operator Pod の場合**

管理者は、次のセクションで説明するように、プロジェクトに **Subscription** オブジェクトを作成できます。その結果、Operator Pod は指定されたノードに配置されます。

**オペランド Pod の場合**

インストールされた Operator を使用して、ユーザーはプロジェクトにアプリケーションを作成できます。これにより、Operator が所有するカスタムリソース (CR) がプロジェクトに置かれます。その結果、Operator が他の namespace にクラスター全体のオブジェクトまたはリソースをデプロイしない限り、オペランド Pod は指定されたノードに配置されます。この場合、このカスタマイズされた Pod の配置は適用されません。

**関連情報**

- プロジェクトスコープのノードセレクターの作成

### 4.1.7. Operator のインストール場所の制御

デフォルトでは、Operator をインストールすると、OpenShift Dedicated は Operator Pod をワーカーノードの 1 つにランダムにインストールします。ただし、特定のノードまたはノードのセットでその Pod をスケジュールする必要がある場合があります。

以下の例では、Operator Pod を特定のノードまたはノードのセットにスケジュールする状況を説明します。

- 同じホストまたは同じラックに配置されたホストでスケジュールされた一緒に動作する Operators が必要な場合
- ネットワークまたはハードウェアの問題によるダウンタイムを回避するために、Operators をインフラストラクチャー全体に分散させたい場合

Operator の **Subscription** オブジェクトにノードアフィニティー、Pod アフィニティー、または Pod アンチアフィニティー制約を追加することで、Operator Pod がインストールされる場所を制御できます。ノードアフィニティーは、Pod の配置場所を判別するためにスケジューラーによって使用されるルールのセットです。Pod アフィニティーを使用すると、関連する Pod が同じノードにスケジュールされていることを確認できます。Pod アンチアフィニティーを使用すると、ノードで Pod がスケジュールされないようにすることができます。

次の例は、ノードアフィニティーまたは Pod アンチアフィニティーを使用して、Custom Metrics Autoscaler Operator のインスタンスをクラスター内の特定のノードにインストールする方法を示しています。

**Operator Pod を特定のノードに配置するノードアフィニティーの例**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: (1)
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - ip-10-0-163-94.us-west-2.compute.internal
#...
```

(1) Operator の Pod を **ip-10-0-163-94.us-west-2.compute.internal** という名前のノードでスケジュールする必要があるノードアフィニティー。

### Operator Pod を特定のプラットフォームのノードに配置するノードアフィニティーの例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: (1)
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/arch
              operator: In
              values:
              - arm64
            - key: kubernetes.io/os
              operator: In
              values:
              - linux
#...
```

(1) Operator の Pod を **kubernetes.io/arch=arm64** および **kubernetes.io/os=linux** ラベルを持つノードでスケジュールする必要があるノードアフィニティー。

### Operator Pod を 1 つ以上の特定のノードに配置する Pod アフィニティーの例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAffinity: (1)
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - test
          topologyKey: kubernetes.io/hostname
#...
```

1 **app=test** ラベルを持つ Pod を持つノードに Operator の Pod を配置する Pod アフィニティー。

### Operator Pod が 1 つ以上の特定のノードからアクセスできないようにする Pod アンチアフィニティーの例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAntiAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: cpu
              operator: In
              values:
              - high
          topologyKey: kubernetes.io/hostname
#...
```

1 Operator の Pod が **cpu=high** ラベルの Pod を持つノードでスケジュールされないようにする Pod アンチアフィニティー。

### 手順

Operator Pod の配置を制御するには、次の手順を実行します。

1. 通常どおり Operator をインストールします。
2. 必要に応じて、ノードがアフィニティーに適切に応答するようにラベル付けされていることを確認してください。
3. Operator **Subscription** オブジェクトを編集してアフィニティーを追加します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity: 1
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
```

```
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
            - ip-10-0-185-229.ec2.internal
#...
```

1. **nodeAffinity**、**podAffinity**、または **podAntiAffinity** を追加します。アフィニティーの作成は、以下のその他のリソースセクションを参照してください。

**検証**

- Pod が特定のノードにデプロイされていることを確認するには、次のコマンドを実行します。

```
$ oc get pods -o wide
```

**出力例**

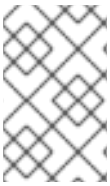
```
NAME                                    READY   STATUS    RESTARTS   AGE   IP
NODE                          NOMINATED NODE   READINESS GATES
custom-metrics-autoscaler-operator-5dcc45d656-bhshg   1/1     Running   0          50s
10.131.0.20   ip-10-0-185-229.ec2.internal   <none>          <none>
```

**関連情報**

- Pod のアフィニティーについて
- ノードアフィニティーについて

## 4.2. インストール済み OPERATOR の更新

**dedicated-admin** ロールを持つ管理者は、Operator Lifecycle Manager (OLM) を使用して OpenShift Dedicated クラスターに以前インストールした Operator を更新できます。



**注記**

OLM が同一 namespace に配置されたインストール済み Operator の更新を処理する方法や、カスタムグローバル Operator グループで Operator をインストールする別の方法は、マルチテナント対応と Operator のコロケーション を参照してください。

### 4.2.1. Operator 更新の準備

インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャネルを指定します。更新チャネルを変更して、新しいチャネルからの更新の追跡と受信を開始できます。

サブスクリプションの更新チャネルの名前は Operators 間で異なる可能性がありますが、命名スキーム通常、特定の Operators 内の共通の規則に従います。たとえば、チャネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に基づく可能性があります。

**注記**

インストールされた Operators は、現在のチャネルよりも古いチャネルに切り換えることはできません。

Red Hat Customer Portal Labs には、管理者が Operators の更新を準備するのに役立つ以下のアプリケーションが含まれています。

- Red Hat OpenShift Container Platform Operator Update Information Checker

このアプリケーションを使用すると、OpenShift Dedicated のさまざまなバージョンを対象に、Operator Lifecycle Manager ベースの Operator を検索し、更新チャネルごとに利用可能な Operator のバージョンを確認できます。Cluster Version Operator ベースの Operator は含まれません。

### 4.2.2. Operator の更新チャネルの変更

OpenShift Dedicated を使用して、Operator の更新チャネルを変更できます。

**ヒント**

サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合、アップグレードプロセスは、選択したチャネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが **Manual** に設定されている場合は、保留中のアップグレードを手動で承認する必要があります。

**前提条件**

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

**手順**

1. Web コンソールで、**Ecosystem → Installed Operators** に移動します。
2. 更新チャネルを変更する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。
4. **Update channel** の下にある更新チャネルの名前をクリックします。
5. 変更する新しい更新チャネルをクリックし、**Save** をクリックします。
6. **Automatic** 承認ストラテジーがあるサブスクリプションの場合、更新は自動的に開始します。**Ecosystem → Installed Operators** ページに戻って、更新の進行状況を監視します。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。
   **Manual** 承認ストラテジーのあるサブスクリプションの場合、**Subscription** タブから更新を手動で承認できます。

### 4.2.3. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

**前提条件**

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

**手順**

1. OpenShift Dedicated Web コンソールで、**Ecosystem → Installed Operators** に移動します。
2. 更新が保留中の Operators は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。承認が必要な更新は、**Upgrade status** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。
4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。
5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。
6. **Ecosystem → Installed Operators** ページに戻って、更新の進行状況を監視します。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

## 4.3. クラスターからの OPERATOR の削除

以下では、Operator Lifecycle Manager (OLM) を使用して OpenShift Dedicated クラスターに以前インストールした Operator を削除またはアンインストールする方法を説明します。

**重要**

同じ Operator の再インストールを試行する前に、Operator を正常かつ完全にアンインストールする必要があります。Operator を適切かつ完全にアンインストールできていない場合、プロジェクトや namespace などのリソースが "Terminating" ステータスでスタックし、Operator を再インストールしようとすると "error resolving resource" メッセージが表示される可能性があります。

### 4.3.1. Web コンソールの使用によるクラスターからの Operators の削除

クラスター管理者は Web コンソールを使用して、選択した namespace からインストールされた Operators を削除できます。

**前提条件**

- **dedicated-admin** パーミッションを持つアカウントを使用して OpenShift Dedicated クラスター Web コンソールにアクセスできる。

**手順**

1. **Ecosystem → Installed Operators** ページに移動します。
2. スクロールするか、キーワードを **Filter by name** フィールドに入力して、削除する Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** 一覧から **Uninstall Operator** を選択します。**Uninstall Operator?** ダイアログボックスが表示されます。
4. **Uninstall** を選択し、Operator、Operator デプロイメント、および Pod を削除します。このアクションの後には、Operator は実行を停止し、更新を受信しなくなります。

**注記**

このアクションは、カスタムリソース定義 (CRD) およびカスタムリソース (CR) など、Operator が管理するリソースは削除されません。Web コンソールおよび継続して実行されるクラスター外のリソースによって有効にされるダッシュボードおよびナビゲーションアイテムには、手動でのクリーンアップが必要になる場合があります。Operator のアンインストール後にこれらを削除するには、Operator CRD を手動で削除する必要があります。

### 4.3.2. CLI の使用によるクラスターからの Operators の削除

クラスター管理者は CLI を使用して、選択した namespace からインストールされた Operators を削除できます。

**前提条件**

- **dedicated-admin** パーミッションを持つアカウントを使用して OpenShift Dedicated クラスターにアクセスできる。
- OpenShift CLI (**oc**) がワークステーションにインストールされている。

**手順**

1. サブスクライブした Operator の最新バージョン (**serverless-operator** など) が、**currentCSV** フィールドで識別されていることを確認します。

```
$ oc get subscription.operators.coreos.com serverless-operator -n openshift-serverless -o yaml | grep currentCSV
```

**出力例**

```
  currentCSV: serverless-operator.v1.28.0
```

2. サブスクリプション (**serverless-operator** など) を削除します。

```
$ oc delete subscription.operators.coreos.com serverless-operator -n openshift-serverless
```

**出力例**

```
subscription.operators.coreos.com "serverless-operator" deleted
```

3. 直前の手順で **currentCSV** 値を使用し、ターゲット namespace の Operator の CSV を削除します。

```
$ oc delete clusterserviceversion serverless-operator.v1.28.0 -n openshift-serverless
```

**出力例**

```
clusterserviceversion.operators.coreos.com "serverless-operator.v1.28.0" deleted
```

### 4.3.3. 障害のあるサブスクリプションの更新

Operator Lifecycle Manager (OLM) で、ネットワークでアクセスできないイメージを参照する Operator をサブスクライブする場合、以下のエラーを出して失敗した **openshift-marketplace** namespace でジョブを見つけることができます。

**出力例**

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-
bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

**出力例**

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get
"https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

その結果、サブスクリプションはこの障害のある状態のままとなり、Operator はインストールまたはアップグレードを実行できません。

サブスクリプション、クラスターサービスバージョン (CSV) その他の関連オブジェクトを削除して、障害のあるサブスクリプションを更新できます。サブスクリプションを再作成した後に、OLM は Operator の正しいバージョンを再インストールします。

**前提条件**

- アクセス不可能なバンドルイメージをプルできない障害のあるサブスクリプションがある。
- 正しいバンドルイメージにアクセスできることを確認している。

**手順**

1. Operator がインストールされている namespace から **Subscription** および **ClusterServiceVersion** オブジェクトの名前を取得します。

   ```
   $ oc get sub,csv -n <namespace>
   ```

   **出力例**

   ```
   NAME                                         PACKAGE                  SOURCE             CHANNEL
   subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-
   operators   5.0

   NAME                                                  DISPLAY                            VERSION
   REPLACES   PHASE
   clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift
   Elasticsearch Operator   5.0.0-65              Succeeded
   ```

2. サブスクリプションを削除します。

   ```
   $ oc delete subscription <subscription_name> -n <namespace>
   ```

3. クラスターサービスバージョンを削除します。

   ```
   $ oc delete csv <csv_name> -n <namespace>
   ```

-

4. **openshift-marketplace** namespace の失敗したジョブおよび関連する config map の名前を取得します。

```
$ oc get job,configmap -n openshift-marketplace
```

**出力例**

```
NAME                                                                COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1
26s        9m30s

NAME                                                                DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3
9m30s
```

5. ジョブを削除します。

```
$ oc delete job <job_name> -n openshift-marketplace
```

これにより、アクセスできないイメージのプルを試行する Pod は再作成されなくなります。

6. 設定マップを削除します。

```
$ oc delete configmap <configmap_name> -n openshift-marketplace
```

7. Web コンソールのソフトウェアカタログを使用して Operator を再インストールします。

**検証**

- Operator が正常に再インストールされていることを確認します。

```
$ oc get sub,csv,installplan -n <namespace>
```

## 4.4. OPERATOR LIFECYCLE MANAGER でのプロキシーサポートの設定

グローバルプロキシーが OpenShift Dedicated クラスターに設定されている場合、Operator Lifecycle Manager (OLM) は、クラスター全体のプロキシーで管理する Operator を自動的に設定します。ただし、インストールされた Operators をグローバルプロキシーをオーバーライドするか、カスタム CA 証明書を注入するように設定することもできます。

**関連情報**

- クラスター全体のプロキシーの設定

### 4.4.1. Operator のプロキシー設定のオーバーライド

クラスター全体の egress プロキシーが設定されている場合、Operator Lifecycle Manager (OLM) を使用して実行する Operators は、デプロイメントでクラスター全体のプロキシー設定を継承します。**dedicated-admin** ロールを持つ管理者は、Operator のサブスクリプションを設定することで、これらのプロキシー設定をオーバーライドすることもできます。

**重要**

Operators は、マネージドオペランドの Pod でのプロキシー設定の環境変数の設定を処理する必要があります。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとして OpenShift Dedicated クラスターにアクセスできる。

**手順**

1. Web コンソールで、**Ecosystem → Software Catalog**ページに移動します。
2. Operator を選択し、**Install** をクリックします。
3. **Install Operator** ページで、**Subscription** オブジェクトを変更して以下の 1 つ以上の環境変数を **spec** セクションに組み込みます。
   - **HTTP_PROXY**
   - **HTTPS_PROXY**
   - **NO_PROXY**

   以下に例を示します。

   **プロキシー設定のオーバーライドのある Subscription オブジェクト**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: etcd-config-test
  namespace: openshift-operators
spec:
  config:
    env:
    - name: HTTP_PROXY
      value: test_http
    - name: HTTPS_PROXY
      value: test_https
    - name: NO_PROXY
      value: test
  channel: clusterwide-alpha
  installPlanApproval: Automatic
  name: etcd
  source: community-operators
  sourceNamespace: openshift-marketplace
  startingCSV: etcdoperator.v0.9.4-clusterwide
```

**注記**

これらの環境変数では、以前に設定されたクラスター全体またはカスタムプロキシーの設定を削除するために空の値を使用してそれらの設定を解除することもできます。

OLM はこれらの環境変数を単位として処理します。それらの環境変数が 1 つ以上設定されている場合、それらはすべてオーバーライドされているものと見なされ、クラスター全体のデフォルト値はサブスクライブされた Operator のデプロイメントには使用されません。

4. **Install** をクリックし、Operator を選択された namespace で利用可能にします。

5. Operator の CSV が関連する namespace に表示されると、カスタムプロキシーの環境変数がデプロイメントに設定されていることを確認できます。たとえば、CLI を使用します。

```
$ oc get deployment -n openshift-operators \
    etcd-operator -o yaml \
    | grep -i "PROXY" -A 2
```

**出力例**

```
        - name: HTTP_PROXY
          value: test_http
        - name: HTTPS_PROXY
          value: test_https
        - name: NO_PROXY
          value: test
        image: quay.io/coreos/etcd-
operator@sha256:66a37fd61a06a43969854ee6d3e21088a98b93838e284a6086b13917f96b0
d9c
...
```

### 4.4.2. カスタム CA 証明書の注入

**dedicated-admin** ロールを持つ管理者が config map を使用してカスタム CA 証明書をクラスターに追加すると、Cluster Network Operator がユーザー提供の証明書とシステム CA 証明書を 1 つのバンドルにマージします。このマージされたバンドルを Operator Lifecycle Manager (OLM) で実行されている Operator に注入することができます。これは、man-in-the-middle HTTPS プロキシーがある場合に役立ちます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとして OpenShift Dedicated クラスターにアクセスできる。
- 設定マップを使用してクラスターに追加されたカスタム CA 証明書。
- 必要な Operator が OLM にインストールされ、実行される。

**手順**

1. Operator のサブスクリプションがある namespace に空の設定マップを作成し、以下のラベルを組み込みます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: trusted-ca 1
  labels:
    config.openshift.io/inject-trusted-cabundle: "true" 2
```

1. 設定マップの名前。
2. Cluster Network Operator に対してマージされたバンドルを注入するように要求します。

この設定マップの作成後すぐに、設定マップにはマージされたバンドルの証明書の内容が設定されます。

2. **Subscription** オブジェクトを更新し、**trusted-ca** 設定マップをカスタム CA を必要とする Pod 内の各コンテナーにボリュームとしてマウントする **spec.config** セクションを追加します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-operator
spec:
  package: etcd
  channel: alpha
  config: 1
    selector:
      matchLabels:
        <labels_for_pods> 2
    volumes: 3
    - name: trusted-ca
      configMap:
        name: trusted-ca
        items:
          - key: ca-bundle.crt 4
            path: tls-ca-bundle.pem 5
    volumeMounts: 6
    - name: trusted-ca
      mountPath: /etc/pki/ca-trust/extracted/pem
      readOnly: true
```

1. **config** セクションがない場合に、これを追加します。
2. Operator が所有する Pod に一致するラベルを指定します。
3. **trusted-ca** ボリュームを作成します。
4. **ca-bundle.crt** は設定マップキーとして必要になります。
5. **tls-ca-bundle.pem** は設定マップパスとして必要になります。
6. **trusted-ca** ボリュームマウントを作成します。

**注記**

Operator のデプロイメントは認証局の検証に失敗し、**x509 certificate signed by unknown authority** エラーが表示される可能性があります。このエラーは、Operator のサブスクリプションの使用時にカスタム CA を注入した後でも発生する可能性があります。この場合、Operator のサブスクリプションを使用して、trusted-ca の **mountPath** を **/etc/ssl/certs** として設定できます。

# 4.5. OPERATOR ステータスの表示

Operator Lifecycle Manager (OLM) のシステムの状態を理解することは、インストールされた Operators に関する問題について意思決定を行い、デバッグを行う上で重要です。OLM は、サブスクリプションおよびそれに関連するカタログソースリソースの状態および実行されたアクションに関する知見を提供します。これは、それぞれの Operators の正常性を把握するのに役立ちます。

## 4.5.1. Operator サブスクリプションの状態のタイプ

サブスクリプションは状態に関する以下のタイプを報告します。

**表4.1 サブスクリプションの状態のタイプ**

| 状態 | 説明 |
| --- | --- |
| **CatalogSourcesUnhealthy** | 解決に使用される一部のまたはすべてのカタログソースは正常ではありません。 |
| **InstallPlanMissing** | サブスクリプションのインストール計画がありません。 |
| **InstallPlanPending** | サブスクリプションのインストール計画はインストールの保留中です。 |
| **InstallPlanFailed** | サブスクリプションのインストール計画が失敗しました。 |
| **ResolutionFailed** | サブスクリプションの依存関係の解決に失敗しました。 |

### 注記

デフォルトの OpenShift Dedicated クラスター Operator は、Cluster Version Operator (CVO) によって管理されます。この Operator には **Subscription** オブジェクトがありません。アプリケーション Operator は、Operator Lifecycle Manager (OLM) によって管理されます。この Operator には **Subscription** オブジェクトがあります。

**関連情報**

- 障害のあるサブスクリプションの更新

## 4.5.2. CLI を使用した Operator サブスクリプションステータスの表示

CLI を使用して Operator サブスクリプションステータスを表示できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. Operator サブスクリプションをリスト表示します。

```
$ oc get subs -n <operator_namespace>
```

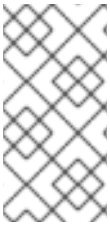
2. **oc describe** コマンドを使用して、**Subscription** リソースを検査します。

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. コマンド出力で、**Conditions** セクションで Operator サブスクリプションの状態タイプのステータスを確認します。以下の例では、利用可能なすべてのカタログソースが正常であるため、**CatalogSourcesUnhealthy** 状態タイプのステータスは **false** になります。

**出力例**

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```



**注記**

デフォルトの OpenShift Dedicated クラスター Operator は、Cluster Version Operator (CVO) によって管理されます。この Operator には **Subscription** オブジェクトがありません。アプリケーション Operator は、Operator Lifecycle Manager (OLM) によって管理されます。この Operator には **Subscription** オブジェクトがあります。

### 4.5.3. CLI を使用した Operator カタログソースのステータス表示

Operator カタログソースのステータスは、CLI を使用して確認できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. namespace のカタログソースをリスト表示します。たとえば、クラスター全体のカタログソースに使用されている **openshift-marketplace** namespace を確認することができます。

```
$ oc get catalogsources -n openshift-marketplace
```

**出力例**

```
NAME                  DISPLAY              TYPE   PUBLISHER   AGE
certified-operators   Certified Operators   grpc   Red Hat     55m
community-operators   Community Operators   grpc   Red Hat     55m
example-catalog       Example Catalog       grpc   Example Org 2m25s
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. カタログソースの詳細やステータスを確認するには、**oc describe** コマンドを使用します。

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

**出力例**

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
    Protocol:           grpc
    Service Name:       example-catalog
    Service Namespace:  openshift-marketplace
# ...
```

前述の出力例では、最後に観測された状態が **TRANSIENT_FAILURE** となっています。この状態は、カタログソースの接続確立に問題があることを示しています。

3. カタログソースが作成された namespace の Pod をリストアップします。

```
$ oc get pods -n openshift-marketplace
```

**出力例**

```
NAME                                  READY   STATUS             RESTARTS   AGE
certified-operators-cv9nn             1/1     Running            0          36m
community-operators-6v8lp             1/1     Running            0          36m
marketplace-operator-86bfc75f9b-jkgbc   1/1     Running          0          42m
example-catalog-bwt8z                 0/1     ImagePullBackOff   0          3m55s
redhat-operators-smxx8                1/1     Running            0          36m
```

namespace にカタログソースを作成すると、その namespace にカタログソース用の Pod が作成されます。前述の出力例では、**example-catalog-bwt8z** Pod のステータスが **ImagePullBackOff** になっています。このステータスは、カタログソースのインデックスイメージのプルに問題があることを示しています。

4. **oc describe** コマンドを使用して、より詳細な情報を得るために Pod を検査します。

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

**出力例**

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age              From               Message
  ----     ------          ----             ----               -------
  Normal   Scheduled       48s              default-scheduler  Successfully assigned openshift-marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s              multus             Add eth0 [10.131.0.40/23] from openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet          Back-off pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet          Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet          Pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet          Failed to pull image "quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet          Error: ErrImagePull
```

前述の出力例では、エラーメッセージは、カタログソースのインデックスイメージが承認問題のために正常にプルできないことを示しています。例えば、インデックスイメージがログイン認証情報を必要とするレジストリーに保存されている場合があります。

**関連情報**

- Operator Lifecycle Manager の概念およびリソース → カタログソース
- gRPC ドキュメント:接続性の状態

## 4.6. OPERATOR 条件の管理

**dedicated-admin** ロールを持つ管理者は、Operator Lifecycle Manager (OLM) を使用して Operator 条件を管理できます。

### 4.6.1. Operator 条件のオーバーライド

**dedicated-admin** ロールを持つ管理者は、Operator によって報告されるサポート対象の Operator 条件を無視することもできます。**Spec.Overrides** 配列に Operator 条件が存在する場合、この条件によって **Spec.Conditions** 配列の条件がオーバーライドされます。これを使用することで、**dedicated-admin** 管理者は、Operator が Operator Lifecycle Manager (OLM) に状態を誤って報告している状況に対処できます。

**注記**

デフォルトでは、**Spec.Overrides** 配列は、**dedicated-admin** ロールを持つ管理者が追加するまで、**OperatorCondition** オブジェクトに存在しません。**Spec.Conditions** 配列も、ユーザーが追加するか、カスタム Operator ロジックの結果として追加されるまで存在しません。

たとえば、アップグレードできないことを常に通信する Operator の既知のバージョンを考えてみましょう。この場合、Operator がアップグレードできないと通信していますが、Operator をアップグレードすることを推奨します。これは、条件の **type** および **status** を **OperatorCondition** オブジェクトの **Spec.Overrides** 配列に追加して Operator 条件をオーバーライドすることによって実行できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- **OperatorCondition** オブジェクトを持つ Operator が OLM を使用してインストールされている。

**手順**

1. Operator の **OperatorCondition** オブジェクトを編集します。

```
$ oc edit operatorcondition <name>
```

2. **Spec.Overrides** 配列をオブジェクトに追加します。

**Operator 条件のオーバーライドの例**

```
apiVersion: operators.coreos.com/v2
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
  overrides:
  - type: Upgradeable (1)
    status: "True"
    reason: "upgradeIsSafe"
    message: "This is a known issue with the Operator where it always reports that it cannot be upgraded."
  conditions:
  - type: Upgradeable
    status: "False"
    reason: "migration"
    message: "The operator is performing a migration."
    lastTransitionTime: "2020-08-24T23:15:55Z"
```

(1) このように編集すると、**dedicated-admin** ユーザーはアップグレードの準備状況を **True** に変更できます。

### 4.6.2. Operator 条件を使用するための Operator の更新

Operator Lifecycle Manager (OLM) は、調整する **ClusterServiceVersion** リソースごとに **OperatorCondition** リソースを自動的に作成します。CSV のすべてのサービスアカウントには、Operator が所有する **OperatorCondition** と対話するための RBAC が付与されます。

Operator の作成者は、Operator が OLM によってデプロイされた後に、独自の条件を設定できるように Operator を開発し、**operator-lib** ライブラリーを使用することができます。Operator 作成者として Operator 条件を設定する方法の詳細は、Operator 条件の有効化 ページを参照してください。

#### 4.6.2.1. デフォルトの設定

後方互換性を維持するために、OLM は **OperatorCondition** リソースがない状態を条件からのオプトアウトとして扱います。そのため、Operator 条件の使用にオプトインする Operator は、Pod の ready プローブが **true** に設定される前に、デフォルトの条件を設定する必要があります。これにより、Operator には、条件を正しい状態に更新するための猶予期間が与えられます。

### 4.6.3. 関連情報

- Operator 条件

## 4.7. カスタムカタログの管理

**dedicated-admin** ロールを持つ管理者と Operator カタログメンテナーは、OpenShift Dedicated の Operator Lifecycle Manager (OLM) で バンドル形式 を使用してパッケージ化したカスタムカタログを作成および管理できます。

**重要**

Kubernetes は定期的に特定の API を非推奨とし、後続のリリースで削除します。そのため、OpenShift Dedicated のバージョンで、API が削除された Kubernetes バージョンが採用されると、Operator がその API を使用できなくなります。

**関連情報**

- Red Hat が提供する Operator カタログ

### 4.7.1. 前提条件

- **opm** CLI がインストールされている。

### 4.7.2. ファイルベースのカタログ

**ファイルベースのカタログ** は、Operator Lifecycle Manager (OLM) の最新バージョンのカタログ形式です。この形式は、プレーンテキストベース (JSON または YAML) であり、以前の SQLite データベース形式の宣言的な設定の進化であり、完全な下位互換性があります。

**重要**

OpenShift Dedicated 4.11 以降、デフォルトの Red Hat 提供の Operator カタログはファイルベースのカタログ形式でリリースされます。OpenShift Dedicated 4.6 から 4.10 までの Red Hat が提供するデフォルトの Operator カタログは、非推奨の SQLite データベース形式でリリースされました。

**opm** サブコマンド、フラグ、および SQLite データベース形式に関連する機能も非推奨となり、今後のリリースで削除されます。機能は引き続きサポートされており、非推奨の SQLite データベース形式を使用するカタログに使用する必要があります。

**opm index prune** などの SQLite データベース形式を使用する **opm** サブコマンドおよびフラグの多くは、ファイルベースのカタログ形式では機能しません。ファイルベースのカタログを使用する方法の詳細は、Operator Framework パッケージ形式 を参照してください。

#### 4.7.2.1. ファイルベースのカタログイメージの作成

**opm** CLI を使用して、非推奨の SQLite データベース形式を置き換えるプレーンテキストの **ファイルベースのカタログ** 形式 (JSON または YAML) を使用するカタログイメージを作成できます。

**前提条件**

- **opm** CLI がインストールされている。
- **podman** バージョン 1.9.3 以降がある。
- バンドルイメージがビルドされ、Docker v2-2 をサポートするレジストリーにプッシュされている。

**手順**

1. カタログを初期化します。
   a. 次のコマンドを実行して、カタログ用のディレクトリーを作成します。

```
$ mkdir <catalog_dir>
```

   b. **opm generate dockerfile** コマンドを実行して、カタログイメージを構築できる Dockerfile を生成します。

```
$ opm generate dockerfile <catalog_dir> \
    -i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4 1
```

   1 **-i** フラグを使用して公式の Red Hat ベースイメージを指定します。それ以外の場合、Dockerfile はデフォルトのアップストリームイメージを使用します。

   Dockerfile は、直前の手順で作成したカタログディレクトリーと同じ親ディレクトリーに存在する必要があります。

   **ディレクトリー構造の例**

```
. 1
├── <catalog_dir> 2
└── <catalog_dir>.Dockerfile 3
```

1. 親ディレクトリー
2. カタログディレクトリー
3. **opm generate dockerfile** コマンドによって生成された Dockerfile

c. **opm init** コマンドを実行して、カタログに Operator のパッケージ定義を追加します。

```
$ opm init <operator_name> \ 1
    --default-channel=preview \ 2
    --description=./README.md \ 3
    --icon=./operator-icon.svg \ 4
    --output yaml \ 5
    > <catalog_dir>/index.yaml 6
```

1. Operator、またはパッケージ、名前。
2. 指定されていない場合にサブスクリプションがデフォルトで使用するチャネル
3. Operator の **README.md** またはその他のドキュメントへのパス。
4. Operator のアイコンへのパス。
5. 出力形式: JSON または YAML。
6. カタログ設定ファイルを作成するパス。

このコマンドは、指定されたカタログ設定ファイルに **olm.package** 宣言型設定 blob を生成します。

2. **opm render** コマンドを実行して、バンドルをカタログに追加します。

```
$ opm render <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
    --output=yaml \
    >> <catalog_dir>/index.yaml 2
```

1. バンドルイメージのプル仕様。
2. カタログ設定ファイルへのパス。

**注記**

チャネルには、1 つ以上のバンドルが含まれる必要があります。

3. バンドルのチャネルエントリーを追加します。たとえば、次の例を仕様に合わせて変更し、**<catalog_dir>/index.yaml** ファイルに追加します。

**チャネルエントリーの例**

```
---
schema: olm.channel
package: <operator_name>
name: preview
entries:
  - name: <operator_name>.v0.1.0 (1)
```

(1) **<operator_name>** の後、かつ、バージョンの **v** の前に、ピリオド (**.**) を追加するようにしてください。それ以外の場合、エントリーが **opm validate** コマンドに合格できません。

4. ファイルベースのカタログを検証します。
   a. カタログディレクトリーに対して **opm validate** コマンドを実行します。

   ```
   $ opm validate <catalog_dir>
   ```

   b. エラーコードが **0** であることを確認します。

   ```
   $ echo $?
   ```

   **出力例**

   ```
   0
   ```

5. **podman build** コマンドを実行して、カタログイメージをビルドします。

```
$ podman build . \
    -f <catalog_dir>.Dockerfile \
    -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. カタログイメージをレジストリーにプッシュします。
   a. 必要に応じて、**podman login** コマンドを実行してターゲットレジストリーで認証します。

   ```
   $ podman login <registry>
   ```

   b. **podman push** コマンドを実行して、カタログイメージをプッシュします。

   ```
   $ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
   ```

**関連情報**

- **opm** CLI リファレンス

#### 4.7.2.2. ファイルベースのカタログイメージの更新またはフィルタリング

**opm** CLI を使用して、ファイルベースのカタログ形式を使用するカタログイメージを更新またはフィルタリングできます。既存のカタログイメージのコンテンツを抽出すると、必要に応じてカタログを変更できます。たとえば、以下を実行できます。

- パッケージの追加
- パッケージの削除
- 既存のパッケージエントリーの更新
- パッケージ、チャネル、バンドルごとの非推奨メッセージの記載

その後、イメージをカタログの更新バージョンとして再構築できます。

**前提条件**

- ワークステーションに以下が含まれている。
  - **opm** CLI。
  - **podman** version 1.9.3+。
  - ファイルベースのカタログイメージ。
  - このカタログに関連するワークステーションで最近初期化されたカタログディレクトリー構造。
    初期化されたカタログディレクトリーがない場合は、ディレクトリーを作成し、Dockerfile を生成します。詳細は、「ファイルベースのカタログイメージの作成」手順の「カタログの初期化」手順を参照してください。

**手順**

1. カタログイメージのコンテンツを YAML 形式でカタログディレクトリーの **index.yaml** ファイルに展開します。

```
$ opm render <registry>/<namespace>/<catalog_image_name>:<tag> \
    -o yaml > <catalog_dir>/index.yaml
```

**注記**

または、**-o json** フラグを使用して JSON 形式で出力することもできます。

2. 作成された **index.yaml** ファイルの内容を仕様に合わせて変更します。

**重要**

バンドルがカタログに公開されたら、いずれかのユーザーがバンドルをインストールしていると想定します。カタログ内で以前に公開されたすべてのバンドルに、現在または新しいチャネルヘッドへの更新パスが設定されていることを確認し、そのバージョンがインストールされているユーザーが立ち往生するのを防ぎます。

- Operator を追加するには、「ファイルベースのカタログイメージの作成」手順のパッケージ、バンドル、およびチャネルエントリーを作成する手順に従います。
- Operator を削除するには、パッケージに関連する **olm.package**、**olm.channel**、および **olm.bundle** Blob のセットを削除します。次の例は、カタログから **example-operator** パッケージを削除するために削除する必要があるセットを示しています。

**例4.9 削除されたエントリーの例**

```
---
defaultChannel: release-2.7
icon:
  base64data: <base64_string>
  mediatype: image/svg+xml
name: example-operator
schema: olm.package
---
entries:
- name: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.0'
- name: example-operator.v2.7.1
  replaces: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.1'
- name: example-operator.v2.7.2
  replaces: example-operator.v2.7.1
  skipRange: '>=2.6.0 <2.7.2'
- name: example-operator.v2.7.3
  replaces: example-operator.v2.7.2
  skipRange: '>=2.6.0 <2.7.3'
- name: example-operator.v2.7.4
  replaces: example-operator.v2.7.3
  skipRange: '>=2.6.0 <2.7.4'
name: release-2.7
package: example-operator
schema: olm.channel
---
image: example.com/example-inc/example-operator-bundle@sha256:<digest>
name: example-operator.v2.7.0
package: example-operator
properties:
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyObject
    version: v1alpha1
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyOtherObject
    version: v1beta1
- type: olm.package
  value:
    packageName: example-operator
    version: 2.7.0
- type: olm.bundle.object
  value:
    data: <base64_string>
- type: olm.bundle.object
  value:
    data: <base64_string>
relatedImages:
- image: example.com/example-inc/example-related-image@sha256:<digest>
```

```
  name: example-related-image
schema: olm.bundle
---
```

- Operator の非推奨メッセージを追加または更新するには、パッケージの **index.yaml** ファイルと同じディレクトリーに **deprecations.yaml** ファイルがあることを確認してください。**deprecations.yaml** ファイル形式の詳細は、「olm.deprecations スキーマ」を参照してください。

3. 変更を保存します。
4. カタログを検証します。

```
$ opm validate <catalog_dir>
```

5. カタログを再構築します。

```
$ podman build . \
   -f <catalog_dir>.Dockerfile \
   -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. 更新されたカタログイメージをレジストリーにプッシュします。

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

**検証**

1. Web コンソールで、**Administration → Cluster Settings → Configuration** ページで OperatorHub 設定リソースに移動します。
2. カタログソースを追加するか、既存のカタログソースを更新して、更新されたカタログイメージのプル仕様を使用します。
   詳細は、このセクションの「関連情報」にある「クラスターへのカタログソースの追加」を参照してください。
3. カタログソースが **READY** 状態になったら、**Ecosystem → Software Catalog** ページに移動します。**Type** 見出しの下の **Operator** を選択し、行った変更が Operator のリストに反映されていることを確認します。

### 4.7.3. SQLite ベースのカタログ

**重要**

Operator カタログの SQLite データベース形式は非推奨の機能です。非推奨の機能は依然として OpenShift Dedicated に含まれており、引き続きサポートされますが、この製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Dedicated で非推奨化または削除された主な機能の最新のリストは、OpenShift Dedicated リリースノートの **非推奨および削除された機能** セクションを参照してください。

#### 4.7.3.1. SQLite ベースのインデックスイメージの作成

**opm** CLI を使用して、SQLite データベース形式に基づいてインデックスイメージを作成できます。

**前提条件**

- **opm** CLI がインストールされている。
- **podman** バージョン 1.9.3 以降がある。
- バンドルイメージがビルドされ、Docker v2-2 をサポートするレジストリーにプッシュされている。

**手順**

1. 新しいインデックスを開始します。

```
$ opm index add \
    --bundles <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
    --tag <registry>/<namespace>/<index_image_name>:<tag> \ 2
    [--binary-image <registry_base_image>] 3
```

1. インデックスに追加するバンドルイメージのコンマ区切りのリスト。
2. インデックスイメージで使用するイメージタグ。
3. オプション: カタログを提供するために使用する代替レジストリーベースイメージ。

2. インデックスイメージをレジストリーにプッシュします。
   a. 必要な場合は、ターゲットレジストリーで認証します。

```
$ podman login <registry>
```

   b. インデックスイメージをプッシュします。

```
$ podman push <registry>/<namespace>/<index_image_name>:<tag>
```

#### 4.7.3.2. SQLite ベースのインデックスイメージの更新

**dedicated-admin** ロールを持つ管理者は、カスタムインデックスイメージを参照するカタログソースを使用するようにソフトウェアカタログを設定した後、インデックスイメージにバンドルイメージを追加することで、クラスター上で利用可能な Operators を最新の状態に保つことができます。

**opm index add** コマンドを使用して既存インデックスイメージを更新できます。

**前提条件**

- **opm** CLI がインストールされている。
- **podman** バージョン 1.9.3 以降がある。
- インデックスイメージがビルドされ、レジストリーにプッシュされている。

- インデックスイメージを参照する既存のカタログソースがある。

**手順**

1. バンドルイメージを追加して、既存のインデックスを更新します。

```
$ opm index add \
    --bundles <registry>/<namespace>/<new_bundle_image>@sha256:<digest> \ 1
    --from-index <registry>/<namespace>/<existing_index_image>:<existing_tag> \ 2
    --tag <registry>/<namespace>/<existing_index_image>:<updated_tag> \ 3
    --pull-tool podman 4
```

1. **--bundles** フラグは、インデックスに追加する他のバンドルイメージのコンマ区切りリストを指定します。
2. **--from-index** フラグは、以前にプッシュされたインデックスを指定します。
3. **--tag** フラグは、更新されたインデックスイメージに適用するイメージタグを指定します。
4. **--pull-tool** フラグは、コンテナーイメージのプルに使用されるツールを指定します。

ここでは、以下のようになります。

**<registry>**

**quay.io** や **mirror.example.com** などのレジストリーのホスト名を指定します。

**<namespace>**

**ocs-dev** や **abc** など、レジストリーの namespace を指定します。

**<new_bundle_image>**

**ocs-operator** など、レジストリーに追加する新しいバンドルイメージを指定します。

**<digest>**

**c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a41** などのバンドルイメージの SHA イメージ ID またはダイジェストを指定します。

**<existing_index_image>**

**abc-redhat-operator-index** など、以前にプッシュされたイメージを指定します。

**<existing_tag>**

以前にプッシュしたイメージのタグ (**4** など) を指定します。

**<updated_tag>**

更新されたインデックスイメージに適用するイメージタグ (**4.1** など) を指定します。

**コマンドの例**

```
$ opm index add \
    --bundles quay.io/ocs-dev/ocs-
operator@sha256:c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a
41 \
    --from-index mirror.example.com/abc/abc-redhat-operator-index:4 \
    --tag mirror.example.com/abc/abc-redhat-operator-index:4.1 \
    --pull-tool podman
```

2. 更新されたインデックスイメージをプッシュします。

```
$ podman push <registry>/<namespace>/<existing_index_image>:<updated_tag>
```

3. Operator Lifecycle Manager (OLM) がカタログソースで参照されるインデックスイメージを一定間隔で自動的にポーリングした後に、新規パッケージが正常に追加されたことを確認します。

```
$ oc get packagemanifests -n openshift-marketplace
```

#### 4.7.3.3. SQLite ベースのインデックスイメージのフィルタリング

Operator Bundle Format に基づくインデックスイメージは、Operator カタログのコンテナー化されたスナップショットです。パッケージの指定された一覧以外のすべてのインデックスを **プルーニング** できます。これにより、必要な Operators のみが含まれるソースインデックスのコピーを作成できます。

**前提条件**

- **podman** バージョン 1.9.3 以降がある。
- **grpcurl** (サードパーティーのコマンドラインツール) がある。
- **opm** CLI がインストールされている。
- Docker v2-2 をサポートするレジストリーにアクセスできる。

**手順**

1. ターゲットレジストリーで認証します。

```
$ podman login <target_registry>
```

2. プルーニングされたインデックスに追加するパッケージのリストを判別します。

   a. コンテナーでプルーニングするソースインデックスイメージを実行します。以下に例を示します。

```
$ podman run -p50051:50051 \
    -it registry.redhat.io/redhat/redhat-operator-index:v4
```

   **出力例**

```
Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4...
Getting image source signatures
Copying blob ae8a0c23f5b1 done
...
INFO[0000] serving registry                 database=/database/index.db port=50051
```

   b. 別のターミナルセッションで、**grpcurl** コマンドを使用して、インデックスが提供するパッケージのリストを取得します。

```
$ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
```

c. **packages.out** ファイルを検査し、プルーニングされたインデックスに保持したいパッケージ名をこのリストから特定します。以下に例を示します。

**パッケージリストのスニペットの例**

```
...
{
  "name": "advanced-cluster-management"
}
...
{
  "name": "jaeger-product"
}
...
{
{
  "name": "quay-operator"
}
...
```

d. **podman run** コマンドを実行したターミナルセッションで、**Ctrl** と **C** を押してコンテナープロセスを停止します。

3. 以下のコマンドを実行して、指定したパッケージ以外のすべてのパッケージのソースインデックスをプルーニングします。

```
$ opm index prune \
   -f registry.redhat.io/redhat/redhat-operator-index:v4 \ 1
   -p advanced-cluster-management,jaeger-product,quay-operator \ 2
   [-i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4] \ 3
   -t <target_registry>:<port>/<namespace>/redhat-operator-index:v4 4
```

1 プルーニングするインデックス。

2 保持するパッケージのコンマ区切りリスト。

3 IBM Power® および IBM Z® イメージにのみ必要: OpenShift Dedicated クラスターのターゲットのメジャーバージョンとマイナーバージョンに一致するタグを持つ Operator Registry ベースイメージ。

4 ビルドされる新規インデックスイメージのカスタムタグ。

4. 以下のコマンドを実行して、新規インデックスイメージをターゲットレジストリーにプッシュします。

```
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4
```

ここで、**<namespace>** はレジストリー上の既存の namespace になります。

### 4.7.4. カタログソースと Pod セキュリティーアドミッション

Pod のセキュリティー標準を確保するために、**Pod セキュリティーアドミッション** が OpenShift

Dedicated 4.11 で導入されました。SQLite ベースのカタログ形式と、OpenShift Dedicated 4.11 より前にリリースされたバージョンの **opm** CLI ツールを使用してビルドされたカタログソースは、制限付き Pod セキュリティーの適用下では実行できません。

OpenShift Dedicated では、制限付き Pod セキュリティーが namespace にデフォルトで適用されず、カタログソースのデフォルトセキュリティーモードが **legacy** に設定されています。

すべての namespace に対するデフォルトの制限付き適用は、今後の OpenShift Dedicated リリースに組み込まれる予定です。制限付き適用が発生した場合、カタログソース Pod の Pod 仕様のセキュリティーコンテキストは、制限付き Pod のセキュリティー標準に一致する必要があります。カタログソースイメージで別の Pod セキュリティー標準が必要な場合は、namespace の Pod セキュリティーアドミッションラベルを明示的に設定する必要があります。

**注記**

SQLite ベースのカタログソース Pod を制限付きで実行する必要がない場合は、OpenShift Dedicated でカタログソースを更新する必要はありません。

ただし、制限付きの Pod セキュリティー適用下でカタログソースが確実に実行されるように、今すぐ対策を講じることを推奨します。制限付き Pod セキュリティー適用下でカタログソースが確実に実行されるように対策を講じないと、今後の OpenShift Dedicated リリースでカタログソースが動作しなくなる可能性があります。

カタログの作成者は、次のいずれかのアクションを実行することで、制限付き Pod セキュリティー適用との互換性を有効にできます。

- カタログをファイルベースのカタログ形式に移行します。
- OpenShift Dedicated 4.11 以降でリリースされた **opm** CLI ツールのバージョンを使用してカタログイメージを更新します。

**注記**

SQLite データベースカタログ形式は非推奨ですが、Red Hat では引き続きサポートされています。将来のリリースでは、SQLite データベース形式はサポートされなくなり、カタログはファイルベースのカタログ形式に移行する必要があります。OpenShift Dedicated 4.11 以降、デフォルトの Red Hat 提供の Operator カタログは、ファイルベースのカタログ形式でリリースされます。ファイルベースのカタログは、制限付き Pod セキュリティー適用と互換性があります。

SQLite データベースカタログイメージを更新したり、カタログをファイルベースのカタログ形式に移行したりしたくない場合は、昇格されたアクセス許可で実行するようにカタログを設定できます。

**関連情報**

- Pod セキュリティーアドミッションの理解と管理

#### 4.7.4.1. SQLite データベースカタログをファイルベースのカタログ形式に移行する

非推奨の SQLite データベース形式のカタログをファイルベースのカタログ形式に更新できます。

**前提条件**

- SQLite データベースカタログソースがある。

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Dedicated でリリースされた **opm** CLI ツールの最新バージョンが、ワークステーションにインストールされている。

**手順**

1. 次のコマンドを実行して、SQLite データベースカタログをファイルベースのカタログに移行します。

```
$ opm migrate <registry_image> <fbc_directory>
```

2. 次のコマンドを実行して、ファイルベースのカタログ用の Dockerfile を生成します。

```
$ opm generate dockerfile <fbc_directory> \
  --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4
```

**次のステップ**

- 生成された Dockerfile をビルドしてタグ付けし、レジストリーにプッシュできます。

**関連情報**

- クラスターへのカタログソースの追加

#### 4.7.4.2. SQLite データベースカタログイメージの再ビルド

OpenShift Dedicated のお使いのバージョンでリリースされた **opm** CLI ツールの最新バージョンを使用して、SQLite データベースカタログイメージを再ビルドできます。

**前提条件**

- SQLite データベースカタログソースがある。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Dedicated でリリースされた **opm** CLI ツールの最新バージョンが、ワークステーションにインストールされている。

**手順**

- 次のコマンドを実行して、最新バージョンの **opm** CLI ツールでカタログを再構築します。

```
$ opm index add --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4 \
  --from-index <your_registry_image> \
  --bundles "" -t \<your_registry_image>
```

#### 4.7.4.3. 昇格された権限で実行するためのカタログの設定

SQLite データベースカタログイメージを更新したり、カタログをファイルベースのカタログ形式に移行したりしたくない場合は、次のアクションを実行して、デフォルトの Pod セキュリティー適用が制限付きに変更されたときにカタログソースが確実に実行されるようにすることができます。

- カタログソース定義でカタログセキュリティーモードをレガシーに手動で設定します。このアクションにより、デフォルトのカタログセキュリティーモードが制限付きに変更された場合でも、カタログが従来のアクセス許可で実行されることが保証されます。
- ベースラインまたは特権付き Pod のセキュリティー適用のために、カタログソースの namespace にラベルを付けます。

**注記**

SQLite データベースカタログ形式は非推奨ですが、Red Hat では引き続きサポートされています。将来のリリースでは、SQLite データベース形式はサポートされなくなり、カタログはファイルベースのカタログ形式に移行する必要があります。ファイルベースのカタログは、制限付き Pod セキュリティー適用と互換性があります。

**前提条件**

- SQLite データベースカタログソースがある。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- Pod Security Admission 標準が **baseline** または **privileged** に昇格された実行中の Pod をサポートするターゲット namespace がある。

**手順**

1. 次の例に示すように、**spec.grpcPodConfig.securityContextConfig** ラベルを **legacy** に設定して、**CatalogSource** 定義を編集します。

   **CatalogSource 定義の例**

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: CatalogSource
   metadata:
     name: my-catsrc
     namespace: my-ns
   spec:
     sourceType: grpc
     grpcPodConfig:
       securityContextConfig: legacy
     image: my-image:latest
   ```

   **ヒント**

   OpenShift Dedicated では、**spec.grpcPodConfig.securityContextConfig** フィールドはデフォルトで **legacy** に設定されています。OpenShift Dedicated の今後のリリースでは、デフォルト設定が **restricted** に変更される予定です。カタログを制限付き適用で実行できない場合は、このフィールドを手動で **legacy** に設定することを推奨します。

2. 次の例に示すように、

2. 次の例に示すように、**<namespace>.yaml** ファイルを編集して、上位の Pod Security Admission 標準をカタログソース namespace に追加します。

**<namespace>.yaml ファイルの例**

```
apiVersion: v1
kind: Namespace
metadata:
...
  labels:
    security.openshift.io/scc.podSecurityLabelSync: "false" (1)
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: baseline (2)
  name: "<namespace_name>"
```

(1) **security.openshift.io/scc.podSecurityLabelSync=false** ラベルを namespace に追加して、Pod のセキュリティーラベルの同期をオフにします。

(2) Pod セキュリティーアドミッションの **pod-security.kubernetes.io/enforce** ラベルを適用します。ラベルを **baseline** または **privileged** に設定します。namespace 内の他のワークロードが **privileged** プロファイルを必要としないかぎり、**baseline** Pod セキュリティープロファイルを使用します。

### 4.7.5. クラスターへのカタログソースの追加

OpenShift Dedicated クラスターにカタログソースを追加すると、ユーザーが Operator を検出してインストールできるようになります。**dedicated-admin** ロールを持つ管理者は、インデックスイメージを参照する **CatalogSource** オブジェクトを作成できます。ソフトウェアカタログは、カタログソースを使用してユーザーインターフェイスの内容を表示します。

#### ヒント

または、Web コンソールを使用してカタログソースを管理できます。**Home → Search** ページからプロジェクトを選択し、**Resources** ドロップダウンをクリックして **CatalogSource** を検索します。個々のソースを作成、更新、削除、無効化、および有効化できます。

#### 前提条件

- インデックスイメージをビルドしてレジストリーにプッシュしている。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

#### 手順

1. インデックスイメージを参照する **CatalogSource** オブジェクトを作成します。
   a. 仕様を以下のように変更し、これを **catalogSource.yaml** ファイルとして保存します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace (1)
```

```
  annotations:
    olm.catalogImageTemplate: 2
      "<registry>/<namespace>/<index_image_name>:v{kube_major_version}.
{kube_minor_version}.{kube_patch_version}"
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 3
  image: <registry>/<namespace>/<index_image_name>:<tag> 4
  displayName: My Operator Catalog
  publisher: <publisher_name> 5
  updateStrategy:
    registryPoll: 6
      interval: 30m
```

1. カタログソースを全 namespace のユーザーがグローバルに利用できるようにする場合は、**openshift-marketplace** namespace を指定します。それ以外の場合は、そのカタログの別の namespace を対象とし、その namespace のみが利用できるように指定できます。

2. 任意: **olm.catalogImageTemplate** アノテーションをカタログイメージ名に設定し、イメージタグのテンプレートを作成する際に、1 つ以上の Kubernetes クラスターバージョン変数を使用します。

3. **legacy** または **restricted** の値を指定します。フィールドが設定されていない場合、デフォルト値は **legacy** です。今後の OpenShift Dedicated リリースでは、デフォルト値が **restricted** になる予定です。

   **注記**

   **restricted** 権限でカタログを実行できない場合は、このフィールドを手動で **legacy** に設定することを推奨します。

4. インデックスイメージを指定します。イメージ名の後にタグを指定すると (**:v4** など)、カタログソース Pod が **Always** イメージプルポリシーを使用します。つまり、Pod がコンテナーを起動する前に常にイメージをプルするようになります。**@sha256:<id>** などのダイジェストを指定した場合、イメージプルポリシーは **IfNotPresent** になります。これは、イメージがノード上にまだ存在しない場合にのみ、Pod がイメージをプルすることを意味します。

5. カタログを公開する名前または組織名を指定します。

6. カタログソースは新規バージョンの有無を自動的にチェックし、最新の状態を維持します。

b. このファイルを使用して **CatalogSource** オブジェクトを作成します。

```
$ oc apply -f catalogSource.yaml
```

2. 以下のリソースが正常に作成されていることを確認します。

a. Pod を確認します。

```
$ oc get pods -n openshift-marketplace
```

**出力例**

```
NAME                                READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6           1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. カタログソースを確認します。

```
$ oc get catalogsource -n openshift-marketplace
```

**出力例**

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc           5s
```

c. パッケージマニフェストを確認します。

```
$ oc get packagemanifest -n openshift-marketplace
```

**出力例**

```
NAME                     CATALOG               AGE
jaeger-product           My Operator Catalog   93s
```

OpenShift Dedicated Web コンソールの **Software Catalog** ページから Operator をインストールできるようになりました。

**関連情報**

- Operator Lifecycle Manager の概念およびリソース → カタログソース

### 4.7.6. カスタムカタログの削除

**dedicated-admin** ロールを持つ管理者は、関連するカタログソースを削除することで、以前にクラスターに追加したカスタム Operator カタログを削除できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

**手順**

1. Web コンソールの **Administrator** パースペクティブで、**Home → Search** に移動します。
2. **Project:** リストからプロジェクトを選択します。
3. **Resources** リストから **CatalogSource** を選択します。

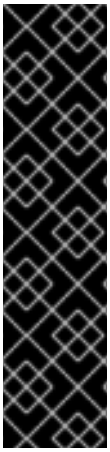
4. 削除するカタログの Options メニュー を選択し、**Delete CatalogSource** をクリックします。

## 4.8. カタログソース POD のスケジューリング

ソースタイプ **grpc** の Operator Lifecycle Manager (OLM) カタログソースが **spec.image** を定義すると、Catalog Operator は、定義されたイメージコンテンツを提供する Pod を作成します。デフォルトでは、この Pod は、その仕様で以下を定義します。

- **kubernetes.io/os=linux** ノードセレクターのみ
- デフォルトの優先クラス名: **system-cluster-critical**。
- toleration なし

管理者は、**CatalogSource** オブジェクトのオプションの **spec.grpcPodConfig** セクションのフィールドを変更すると、これらの値をオーバーライドできます。



**重要**

Marketplace Operator の **openshift-marketplace** は、デフォルトの **OperatorHub** カスタムリソース (CR) を管理します。この CR は **CatalogSource** オブジェクトを管理します。ユーザーが **CatalogSource** オブジェクトの **spec.grpcPodConfig** セクションのフィールドを変更しようとすると、Marketplace Operator によってその変更が自動的に元に戻されます。デフォルトでは、**CatalogSource** オブジェクトの **spec.grpcPodConfig** セクションのフィールドを変更すると、Marketplace Operator によってその変更が自動的に元に戻されます。

**CatalogSource** オブジェクトに永続的な変更を適用するには、まずデフォルトの **CatalogSource** オブジェクトを無効にする必要があります。

**関連情報**

- OLM concepts and resources → Catalog source

### 4.8.1. ローカルレベルでのデフォルト CatalogSource オブジェクトの無効化

デフォルトの **CatalogSource** オブジェクトを無効にすることで、カタログソース Pod などの永続的な変更をローカルレベルで **CatalogSource** オブジェクトに適用できます。デフォルトの **CatalogSource** オブジェクトの設定が組織のニーズを満たさない場合は、デフォルト設定を検討してください。デフォルトでは、**CatalogSource** オブジェクトの **spec.grpcPodConfig** セクションのフィールドを変更すると、Marketplace Operator によってその変更が自動的に元に戻されます。

Marketplace Operator の **openshift-marketplace** は、**OperatorHub** のデフォルトのカスタムリソース (CR) を管理します。**OperatorHub** は **CatalogSource** オブジェクトを管理します。

**CatalogSource** オブジェクトに永続的な変更を適用するには、まずデフォルトの **CatalogSource** オブジェクトを無効にする必要があります。

**手順**

- すべてのデフォルトの **CatalogSource** オブジェクトをローカルレベルで無効にするには、次のコマンドを入力します。

```
$ oc patch operatorhub cluster -p '{"spec": {"disableAllDefaultSources": true}}' --type=merge
```

**注記**

また、デフォルトの **OperatorHub** CR を設定して、すべての **CatalogSource** オブジェクトを無効にするか、または特定のオブジェクトを無効にすることもできます。

**関連情報**

- OperatorHub カスタムリソース

### 4.8.2. カタログソース Pod のノードセレクターのオーバーライド

**前提条件**

- **spec.image** を持つソースタイプ **grpc** の **CatalogSource** オブジェクトが定義されている。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

**手順**

- **CatalogSource** オブジェクトを編集し、**spec.grpcPodConfig** セクションを追加または変更して、以下を含めます。

```
grpcPodConfig:
  nodeSelector:
    custom_label: <label>
```

**<label>** は、カタログソース Pod がスケジュールに使用するノードセレクターのラベルです。

**関連情報**

- ノードセレクターの使用による特定ノードへの Pod の配置

### 4.8.3. カタログソース Pod の優先度クラス名のオーバーライド

**前提条件**

- **spec.image** を持つソースタイプ **grpc** の **CatalogSource** オブジェクトが定義されている。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

**手順**

- **CatalogSource** オブジェクトを編集し、**spec.grpcPodConfig** セクションを追加または変更して、以下を含めます。

```
grpcPodConfig:
  priorityClassName: <priority_class>
```

**<priority_class>** は次のいずれかです。

- Kubernetes によって提供されるデフォルトの優先度クラスの 1 つ: **system-cluster-critical** または **system-node-critical**
- デフォルトの優先度を割り当てる空のセット (**""**)
- 既存およびカスタム定義の優先度クラス

**注記**

以前は、オーバーライドできる唯一の Pod スケジューリングパラメーターは **priorityClassName** でした。これは、**operatorframework.io/priorityclass** アノテーションを **CatalogSource** オブジェクトに追加することによって行われました。以下に例を示します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    operatorframework.io/priorityclass: system-cluster-critical
```

**CatalogSource** オブジェクトがアノテーションと **spec.grpcPodConfig.priorityClassName** の両方を定義する場合、アノテーションは設定パラメーターよりも優先されます。

**関連情報**

- Pod の優先度クラス

### 4.8.4. カタログソース Pod の toleration のオーバーライド

**前提条件**

- **spec.image** を持つソースタイプ **grpc** の **CatalogSource** オブジェクトが定義されている。
- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

**手順**

- **CatalogSource** オブジェクトを編集し、**spec.grpcPodConfig** セクションを追加または変更して、以下を含めます。

```
grpcPodConfig:
  tolerations:
    - key: "<key_name>"
```

```
operator: "<operator_type>"
value: "<value>"
effect: "<effect>"
```

## 4.9. OPERATOR 関連の問題のトラブルシューティング

Operator に問題が発生した場合には、Operator Subscription のステータスを確認します。クラスター全体で Operator Pod の正常性を確認し、診断用に Operator ログを収集します。

### 4.9.1. Operator サブスクリプションの状態のタイプ

サブスクリプションは状態に関する以下のタイプを報告します。

**表4.2 サブスクリプションの状態のタイプ**

| 状態 | 説明 |
|---|---|
| **CatalogSourcesUnhealthy** | 解決に使用される一部のまたはすべてのカタログソースは正常ではありません。 |
| **InstallPlanMissing** | サブスクリプションのインストール計画がありません。 |
| **InstallPlanPending** | サブスクリプションのインストール計画はインストールの保留中です。 |
| **InstallPlanFailed** | サブスクリプションのインストール計画が失敗しました。 |
| **ResolutionFailed** | サブスクリプションの依存関係の解決に失敗しました。 |

**注記**

デフォルトの OpenShift Dedicated クラスター Operator は、Cluster Version Operator (CVO) によって管理されます。この Operator には **Subscription** オブジェクトがありません。アプリケーション Operator は、Operator Lifecycle Manager (OLM) によって管理されます。この Operator には **Subscription** オブジェクトがあります。

**関連情報**

- カタログの正常性要件

### 4.9.2. CLI を使用した Operator サブスクリプションステータスの表示

CLI を使用して Operator サブスクリプションステータスを表示できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. Operator サブスクリプションをリスト表示します。

```
$ oc get subs -n <operator_namespace>
```

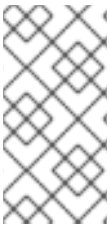
2. **oc describe** コマンドを使用して、**Subscription** リソースを検査します。

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. コマンド出力で、**Conditions** セクションで Operator サブスクリプションの状態タイプのステータスを確認します。以下の例では、利用可能なすべてのカタログソースが正常であるため、**CatalogSourcesUnhealthy** 状態タイプのステータスは **false** になります。

**出力例**

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```



**注記**

デフォルトの OpenShift Dedicated クラスター Operator は、Cluster Version Operator (CVO) によって管理されます。この Operator には **Subscription** オブジェクトがありません。アプリケーション Operator は、Operator Lifecycle Manager (OLM) によって管理されます。この Operator には **Subscription** オブジェクトがあります。

### 4.9.3. CLI を使用した Operator カタログソースのステータス表示

Operator カタログソースのステータスは、CLI を使用して確認できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. namespace のカタログソースをリスト表示します。たとえば、クラスター全体のカタログソースに使用されている **openshift-marketplace** namespace を確認することができます。

```
$ oc get catalogsources -n openshift-marketplace
```

**出力例**

```
NAME                  DISPLAY               TYPE   PUBLISHER   AGE
certified-operators   Certified Operators   grpc   Red Hat     55m
community-operators   Community Operators   grpc   Red Hat     55m
example-catalog       Example Catalog       grpc   Example Org 2m25s
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. カタログソースの詳細やステータスを確認するには、**oc describe** コマンドを使用します。

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

**出力例**

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
    Protocol:           grpc
    Service Name:       example-catalog
    Service Namespace:  openshift-marketplace
# ...
```

前述の出力例では、最後に観測された状態が **TRANSIENT_FAILURE** となっています。この状態は、カタログソースの接続確立に問題があることを示しています。

3. カタログソースが作成された namespace の Pod をリストアップします。

```
$ oc get pods -n openshift-marketplace
```

**出力例**

```
NAME                                  READY   STATUS             RESTARTS   AGE
certified-operators-cv9nn             1/1     Running            0          36m
community-operators-6v8lp             1/1     Running            0          36m
marketplace-operator-86bfc75f9b-jkgbc 1/1     Running            0          42m
example-catalog-bwt8z                 0/1     ImagePullBackOff   0          3m55s
redhat-operators-smxx8                1/1     Running            0          36m
```

namespace にカタログソースを作成すると、その namespace にカタログソース用の Pod が作成されます。前述の出力例では、**example-catalog-bwt8z** Pod のステータスが

**ImagePullBackOff** になっています。このステータスは、カタログソースのインデックスイメージのプルに問題があることを示しています。

4. **oc describe** コマンドを使用して、より詳細な情報を得るために Pod を検査します。

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

**出力例**

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age              From               Message
  ----     ------          ----             ----               -------
  Normal   Scheduled       48s              default-scheduler  Successfully assigned openshift-marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s              multus             Add eth0 [10.131.0.40/23] from openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet          Back-off pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet          Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet          Pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet          Failed to pull image "quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet          Error: ErrImagePull
```

前述の出力例では、エラーメッセージは、カタログソースのインデックスイメージが承認問題のために正常にプルできないことを示しています。例えば、インデックスイメージがログイン認証情報を必要とするレジストリーに保存されている場合があります。

**関連情報**

- gRPC ドキュメント:接続性の状態

### 4.9.4. Operator Pod ステータスのクエリー

クラスター内の Operator Pod およびそれらのステータスをリスト表示できます。詳細な Operator Pod の要約を収集することもできます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- API サービスが機能している。
- OpenShift CLI (**oc**) がインストールされている。

**手順**

1. クラスターで実行されている Operators をリスト表示します。出力には、Operator バージョン、可用性、およびアップタイムの情報が含まれます。

```
$ oc get clusteroperators
```

2. Operator の namespace で実行されている Operator Pod をリスト表示し、Pod のステータス、再起動、および経過時間をリスト表示します。

```
$ oc get pod -n <operator_namespace>
```

3. 詳細な Operator Pod の要約を出力します。

```
$ oc describe pod <operator_pod_name> -n <operator_namespace>
```

### 4.9.5. Operator ログの収集

Operator の問題が発生した場合、Operator Pod ログから詳細な診断情報を収集できます。

**前提条件**

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- API サービスが機能している。
- OpenShift CLI (**oc**) がインストールされている。
- コントロールプレーンまたはコントロールプレーンマシンの完全修飾ドメイン名がある。

**手順**

1. Operator の namespace で実行されている Operator Pod、Pod のステータス、再起動、および経過時間をリスト表示します。

```
$ oc get pods -n <operator_namespace>
```

2. Operator Pod のログを確認します。

```
$ oc logs pod/<pod_name> -n <operator_namespace>
```

Operator Pod に複数のコンテナーがある場合、前述のコマンドにより各コンテナーの名前が含まれるエラーが生成されます。個別のコンテナーに対して、ログのクエリーを実行します。

```
$ oc logs pod/<operator_pod_name> -c <container_name> -n <operator_namespace>
```

3. API が機能しない場合には、代わりに SSH を使用して各コントロールプレーンノードで Operator Pod およびコンテナーログを確認します。**<master-node>.<cluster_name>.<base_domain>** を適切な値に置き換えます。

   a. 各コントロールプレーンノードの Pod をリスト表示します。

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl pods
```

b. Operator Pod で **Ready** ステータスが表示されない場合は、Pod のステータスを詳細に検査します。**<operator_pod_id>** を直前のコマンドの出力にリスト表示されている Operator Pod の ID に置き換えます。

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspectp
<operator_pod_id>
```

c. Operator Pod に関連するコンテナーをリスト表示します。

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl ps --pod=
<operator_pod_id>
```

d. **Ready** ステータスが Operator コンテナーに表示されない場合は、コンテナーのステータスを詳細に検査します。**<container_id>** を前述のコマンドの出力に一覧表示されているコンテナー ID に置き換えます。

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspect
<container_id>
```

e. **Ready** ステータスが表示されない Operator コンテナーのログを確認します。**<container_id>** を前述のコマンドの出力に一覧表示されているコンテナー ID に置き換えます。

```
$ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl logs -f
<container_id>
```

**注記**

Red Hat Enterprise Linux CoreOS (RHCOS) を実行する OpenShift Dedicated 4 クラスターノードはイミュータブルです。クラスターの変更を適用するには、Operator を使用します。SSH を使用したクラスターノードへのアクセスは推奨されません。SSH 経由で診断データの収集を試行する前に、**oc adm must gather** およびその他の **oc** コマンドを実行して収集されるデータが十分であるかどうかを確認してください。ただし、OpenShift Dedicated API が使用できない場合、または kubelet がターゲットノード上で適切に機能していない場合は、**oc** 操作が影響を受けます。この場合は、代わりに **ssh core@<node>.<cluster_name>.<base_domain>** を使用してノードにアクセスできます。