OpenShift Container Platform 4.17

확장

OLM(Operator Lifecycle Manager) v1을 사용하여 OpenShift Container Platform의 확장 작업 OLM v1은 기술 프리뷰 기능 전용입니다.

Last Updated: 2024-12-16

OpenShift Container Platform 4.17 확장

OLM(Operator Lifecycle Manager) v1을 사용하여 OpenShift Container Platform의 확장 작업 OLM v1은 기술 프리뷰 기능 전용입니다.

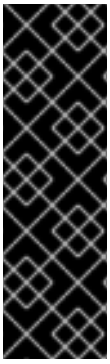
법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 확장 프로그램 및 Operator를 설치, 관리 및 구성하는 방법에 대해 설명합니다. OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다.

차례

1장. 확장 개요



중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

클러스터 관리자는 확장 기능을 통해 OpenShift Container Platform 클러스터에서 사용자의 기능을 확장할 수 있습니다.

OLM(Operator Lifecycle Manager)은 최초 릴리스 이후 OpenShift Container Platform 4에 포함되어 있습니다. OpenShift Container Platform 4.17에는 이 단계에서 *OLM v1* 로 알려진 일반 사용 가능(GA) 기능으로 OLM의 차세대 반복을 위한 구성 요소가 포함되어 있습니다. 이 업데이트된 프레임워크는 이전 버전의 OLM에 포함된 여러 개념을 개발하고 새로운 기능을 추가합니다.

1.1. 주요 사항

관리자는 다음과 같은 주요 사항을 확인할 수 있습니다.

GitOps 워크플로를 지원하는 완전히 선언적 모델

OLM v1은 다음 두 가지 주요 API를 통해 확장 관리를 간소화합니다.

- 새로운 **ClusterExtension** API는 사용자용 API를 단일 오브젝트로 통합하여 **registry+v1** 번들 형식을 통해 Operator를 포함하는 설치된 확장 기능을 간소화합니다. 이 API는 새 Operator Controller 구성 요소에서 **clusterextension.olm.operatorframework.io** 로 제공됩니다. 관리자와 SRE는 API를 사용하여 GitOps 원칙을 사용하여 프로세스를 자동화하고 원하는 상태를 정의할 수 있습니다.

  참고

  OLM v1의 이전 기술 프리뷰 단계에서는 새 **Operator** API가 도입되었습니다. 이 API는 OpenShift Container Platform 4.16에서 **ClusterExtension** 의 이름을 변경하여 다음과 같은 개선 사항을 해결합니다.

  - 클러스터 기능 확장의 단순화된 기능을 보다 정확하게 반영합니다.
  - 보다 유연한 패키징 형식을 더 잘 나타냅니다.
  - **클러스터** 접두사는 **ClusterExtension** 오브젝트가 클러스터 범위임을 명확하게 나타내며, 기존 OLM의 변경으로 Operator는 네임스페이스 범위 또는 클러스터 범위 중 하나일 수 있습니다.

- 새 카탈로그 구성 요소에서 제공하는 카탈로그 API는 OLM v1의 기반 역할을 하며 사용자가 Kubernetes 확장 및 Operator와 같은 설치 가능한 콘텐츠를 검색할 수 있도록 클러스터 내 클라이언트에 대한 카탈로그 압축을 풉니다. **이렇게 하면 세부 정보, 채널 및 업데이트 에지를 포함하여 사용 가능한 모든 Operator 번들 버전에 대한 가시성이 향상됩니다.**

중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

자세한 내용은 Operator Controller 및 Catalogd 를 참조하십시오.

확장 업데이트에 대한 제어 기능 개선

카탈로그 콘텐츠에 대한 통찰력이 개선되어 관리자는 설치 및 업데이트에 대한 대상 버전을 지정할 수 있습니다. 이를 통해 관리자는 대상 버전의 확장 업데이트를 더 많이 제어할 수 있습니다. 자세한 내용은 클러스터 확장 업데이트를 참조하십시오.

유연한 확장 패키지 형식

관리자는 기존 OLM 환경과 유사하게 파일 기반 카탈로그를 사용하여 OLM 기반 Operator와 같은 확장을 설치하고 관리할 수 있습니다.

또한 번들 크기는 더 이상 etcd 값 크기 제한으로 제한되지 않습니다. 자세한 내용은 확장 설치를 참조하십시오.

보안 카탈로그 통신

OLM v1은 카탈로그 서버 응답에 HTTPS 암호화를 사용합니다.

1.2. 목적

OLM(Operator Lifecycle Manager)의 미션은 Kubernetes 클러스터에서 클러스터 확장의 라이프사이클을 중앙 및 선언적으로 관리하는 것이었습니다. 그 목적은 항상 기본 클러스터의 라이프사이클 전반에 걸쳐 클러스터 및 platform-as-a-service (Platform-as-a-service) 관리자를 위해 쉽고 안전하며 재현할 수 있도록 기능 확장을 설치, 실행 및 업데이트하는 것이었습니다.

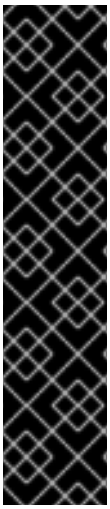
OpenShift Container Platform 4로 시작하고 기본적으로 포함되어 있는 OLM의 초기 버전은 Operator라는 특정 유형의 클러스터 확장에 대한 이러한 특정 요구 사항에 대한 고유한 지원을 제공하는 데 중점을 둡니다. Operator는 클러스터에 추가 기능을 제공하기 위해 하나 이상의 API 확장 기능과 함께 제공되는 하나 이상의 Kubernetes 컨트롤러로 CRD(CustomResourceDefinition) 오브젝트로 분류됩니다.

많은 릴리스의 프로덕션 클러스터에서 실행된 후 OLM의 차세대는 Operator뿐만 아니라 클러스터 확

장에 대한 라이프사이클을 포함하는 것을 목표로 합니다.

2장. 아키텍처

2.1. OLM V1 구성 요소 개요



중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

OLM(Operator Lifecycle Manager) v1은 다음과 같은 구성 요소 프로젝트로 구성됩니다.

Operator Controller

Operator 컨트롤러는 사용자가 **Operator** 및 확장의 라이프사이클을 설치하고 관리할 수 있는 **API**를 사용하여 **Kubernetes**를 확장하는 **OLM v1**의 핵심 구성 요소입니다. **catalogd**의 정보를 사용합니다.

Catalogd

Catalogd 는 클러스터 클라이언트의 사용을 위해 패키지화되어 컨테이너 이미지에 포함되어 있는 파일 기반 카탈로그(FBC) 콘텐츠의 압축을 풀는 **Kubernetes** 확장입니다. **OLM v1** 마이크로 서비스 아키텍처의 구성 요소로, 확장 작성자가 패키지한 **Kubernetes** 확장 기능에 대한 카탈로그 호스트 메타데이터를 호스트하므로 사용자가 설치 가능한 콘텐츠를 검색하는 데 도움이 됩니다.

2.2. OPERATOR CONTROLLER

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

Operator 컨트롤러는 OLM(Operator Lifecycle Manager) v1의 핵심 구성 요소이며 다른 OLM v1 구성 요소 catalogd를 사용합니다. 사용자가 Operator 및 확장을 설치할 수 있는 API를 사용하여 Kubernetes를 확장합니다.

2.2.1. ClusterExtension API

Operator 컨트롤러는 registry+v1 번들 형식을 통해 Operator를 포함하는 설치된 확장 인스턴스를 나타내는 단일 리소스인 새로운 ClusterExtension API 오브젝트를 제공합니다. 이 clusterextension.olm.operatorframework.io API는 사용자용 API를 단일 오브젝트로 통합하여 설치된 확장 확장의 관리를 간소화합니다.

중요

OLM v1에서 ClusterExtension 오브젝트는 클러스터 범위입니다. 이는 관련 Subscription 및 OperatorGroup 오브젝트의 구성에 따라 Operator가 네임스페이스 범위 또는 클러스터 범위 중 하나일 수 있는 기존 OLM과 다릅니다.

이전 동작에 대한 자세한 내용은 *Multitenancy 및 Operator colocation* 을 참조하십시오.

ClusterExtension 오브젝트의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: <operator_name>
spec:
  packageName: <package_name>
```

```
  installNamespace: <namespace_name>
  channel: <channel_name>
  version: <version_number>
```

추가 리소스

- OLM(Operator Lifecycle Manager) → 멀티 테넌시 및 Operator 공동 배치

2.2.1.1. 대상 버전을 지정하는 CR(사용자 정의 리소스)의 예

OLM(Operator Lifecycle Manager) v1에서 클러스터 관리자는 사용자 정의 리소스(CR)에서 Operator 또는 확장의 대상 버전을 선언적으로 설정할 수 있습니다.

다음 필드 중 하나를 지정하여 대상 버전을 정의할 수 있습니다.

- 채널
- 버전 번호
- 버전 범위

CR에 채널을 지정하면 OLM v1이 지정된 채널 내에서 해결할 수 있는 최신 버전의 Operator 또는 확장 버전을 설치합니다. 지정된 채널에 업데이트가 게시되면 OLM v1이 채널에서 확인할 수 있는 최신 릴리스로 자동으로 업데이트됩니다.

지정된 채널이 있는 CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
```

```
  serviceAccount:
    name: <service_account>
  channel: latest (1)
```

(1)

지정된 채널에서 확인할 수 있는 최신 릴리스를 설치합니다. 채널 업데이트가 자동으로 설치됩니다.

CR에서 Operator 또는 확장의 대상 버전을 지정하면 OLM v1이 지정된 버전을 설치합니다. 대상 버전이 CR에 지정되면 업데이트가 카탈로그에 게시될 때 OLM v1에서 대상 버전이 변경되지 않습니다.

클러스터에 설치된 Operator 버전을 업데이트하려면 Operator의 CR을 수동으로 편집해야 합니다. Operator의 대상 버전을 지정하면 Operator 버전이 지정된 릴리스에 고정됩니다.

대상 버전이 지정된 CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  version: "1.11.1" (1)
```

(1)

대상 버전을 지정합니다. 설치된 Operator 또는 확장 버전을 업데이트하려면 CR을 원하는 대상 버전으로 수동으로 업데이트해야 합니다.

Operator 또는 확장에 허용되는 다양한 버전을 정의하려면 비교 문자열을 사용하여 버전 범위를 지정할 수 있습니다. 버전 범위를 지정하면 OLM v1은 Operator 컨트롤러에서 해결할 수 있는 최신 버전의

Operator 또는 확장을 설치합니다.

버전 범위가 지정된 CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  version: ">1.11.1" (1)
```

(1) 원하는 버전 범위가 버전 1.11.1 보다 크도록 지정합니다. 자세한 내용은 "버전 범위 지원"을 참조하십시오.

CR을 생성하거나 업데이트한 후 다음 명령을 실행하여 구성 파일을 적용합니다.

명령 구문

```
$ oc apply -f <extension_name>.yaml
```

2.2.2. 클러스터 확장의 오브젝트 소유권

OLM(Operator Lifecycle Manager) v1에서 Kubernetes 오브젝트는 한 번에 단일 ClusterExtension 오브젝트에서만 소유할 수 있습니다. 이렇게 하면 OpenShift Container Platform 클러스터 내의 오브젝트를 일관되게 관리하고 동일한 오브젝트를 제어하려고 하는 여러 클러스터 확장 간의 충돌을 방지할 수 있습니다.

2.2.2.1. 단일 소유권

OLM v1에서 적용하는 코어 소유권 원칙은 각 오브젝트에 소유자로서 하나의 클러스터 확장만 있을 수 있다는 것입니다. 이렇게 하면 여러 클러스터 확장에 의해 중복되거나 충돌하는 관리가 방지되므로 각 오브젝트가 하나의 번들과 고유하게 연결되어 있습니다.

단일 소유권의 영향

- **CRD(CustomResourceDefinition)** 오브젝트를 제공하는 번들은 한 번만 설치할 수 있습니다.

  번들은 **ClusterExtension** 오브젝트의 일부인 **CRD**를 제공합니다. 즉, 클러스터에서 번들을 한 번만 설치할 수 있습니다. 각 사용자 정의 리소스에 소유자와 함께 하나의 클러스터 확장만 있을 수 있으므로 동일한 **CRD**를 제공하는 다른 번들을 설치하려고 하면 오류가 발생합니다.

- 클러스터 확장은 오브젝트를 공유할 수 없습니다.

  OLM v1의 단일 소유자 정책은 클러스터 확장이 오브젝트의 소유권을 공유할 수 없음을 의미합니다. 하나의 클러스터 확장에서 **Deployment,CustomResourceDefinition** 또는 **Service** 오브젝트와 같은 특정 오브젝트를 관리하는 경우 다른 클러스터 확장에서는 동일한 오브젝트의 소유권을 요청할 수 없습니다. 이렇게 하려는 모든 시도는 **OLM v1**에 의해 차단됩니다.

2.2.2.2. 오류 메시지

동일한 오브젝트를 관리하려고 하는 여러 클러스터 확장으로 인해 충돌이 발생하면 **Operator Controller**는 다음과 같은 소유권 충돌을 나타내는 오류 메시지를 반환합니다.

오류 메시지의 예

```
CustomResourceDefinition 'logfilemetricexporters.logging.kubernetes.io' already exists in namespace 'kubernetes-logging' and cannot be managed by operator-controller
```

이 오류 메시지는 개체가 이미 다른 클러스터 확장에 의해 관리되고 있으며 다시 할당하거나 공유할 수 없음을 나타냅니다.

2.2.2.3. 고려 사항

클러스터 또는 확장 관리자로서 다음 고려 사항을 검토하십시오.

번들의 고유성

동일한 CRD를 제공하는 Operator 번들이 두 번 이상 설치되지 않았는지 확인합니다. 이렇게 하면 소유권 충돌로 인해 잠재적인 설치 실패를 방지할 수 있습니다.

오브젝트 공유 방지

유사한 리소스와 상호 작용하기 위해 다른 클러스터 확장이 필요한 경우 별도의 오브젝트를 관리하고 있는지 확인하십시오. 클러스터 확장은 단일 소유자 적용으로 인해 동일한 오브젝트를 공동으로 관리할 수 없습니다.

2.3. CATALOGD

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

OLM(Operator Lifecycle Manager) v1은 카탈로그 구성 요소와 해당 리소스를 사용하여 Operator 및 확장 카탈로그를 관리합니다.

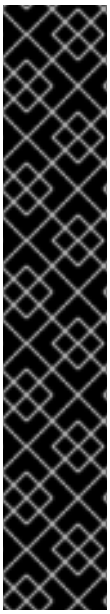
중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

2.3.1. OLM v1의 카탈로그 정보

카탈로그 구성 요소를 사용하여 Operator 및 컨트롤러와 같은 Kubernetes 확장 카탈로그를 쿼리하여 설치 가능한 콘텐츠를 검색할 수 있습니다. Catalogd는 클러스터 내 클라이언트의 카탈로그 콘텐츠의 압

축을 풀고 **OLM(Operator Lifecycle Manager) v1** 마이크로 서비스 제품군의 일부입니다. 현재 **catalogd** 는 컨테이너 이미지로 패키지 및 배포되는 카탈로그 콘텐츠의 압축을 풉니다.



중요

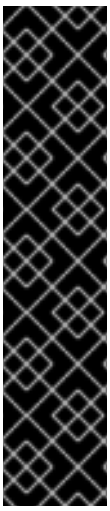
고유한 이름이 없는 **Operator** 또는 확장을 설치하려고 하면 설치에 실패하거나 예기치 않은 결과가 발생할 수 있습니다. 이는 다음과 같은 이유로 발생합니다.

- **mulitple** 카탈로그가 클러스터에 설치된 경우 **OLM(Operator Lifecycle Manager) v1**에는 **Operator** 또는 확장을 설치할 때 카탈로그를 지정하는 메커니즘이 포함되지 않습니다.
- **OLM v1**에서는 클러스터에 설치할 수 있는 모든 **Operator** 및 확장이 번들 및 패키지에 고유한 이름을 사용해야 합니다.

추가 리소스

- 파일 기반 카탈로그
- 클러스터에 카탈로그 추가
- **Red Hat** 제공 카탈로그

3장. OPERATOR 프레임워크 일반 용어집



중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

다음 용어는 **OLM(Operator Lifecycle Manager) v1**을 포함한 **Operator** 프레임워크와 관련이 있습니다.

3.1. 일반 OPERATOR 프레임워크 용어

3.1.1. 번들

번들 형식에서 *번들*은 **Operator CSV**, 매니페스트, 메타데이터로 이루어진 컬렉션입니다. 이러한 구성 요소가 모여 클러스터에 설치할 수 있는 고유한 버전의 **Operator**를 형성합니다.

3.1.2. 번들 이미지

번들 형식에서 *번들 이미지*는 **Operator** 매니페스트에서 빌드하고 하나의 번들을 포함하는 컨테이너 이미지입니다. 번들 이미지는 **Quay.io** 또는 **DockerHub**와 같은 **OCI(Open Container Initiative)** 사양 컨테이너 레지스트리에서 저장 및 배포합니다.

3.1.3. 카탈로그 소스

*카탈로그 소스*는 **OLM**에서 **Operator** 및 해당 종속 항목을 검색하고 설치하기 위해 쿼리할 수 있는 메타데이터 저장소를 나타냅니다.

3.1.4. 채널

*채널*은 **Operator**의 업데이트 스트림을 정의하고 구독자에게 업데이트를 배포하는 데 사용됩니다. 헤드는 해당 채널의 최신 버전을 가리킵니다. 예를 들어 **stable** 채널에는 **Operator**의 모든 안정적인 버전이

가장 오래된 것부터 최신 순으로 정렬되어 있습니다.

Operator에는 여러 개의 채널이 있을 수 있으며 특정 채널에 대한 서브스크립션 바인딩에서는 해당 채널의 업데이트만 찾습니다.

3.1.5. 채널 헤드

*채널 헤드*는 알려진 특정 채널의 최신 업데이트를 나타냅니다.

3.1.6. 클러스터 서비스 버전

*CSV(클러스터 서비스 버전)*는 Operator 메타데이터에서 생성하는 YAML 매니페스트로, OLM이 클러스터에서 Operator를 실행하는 것을 지원합니다. 로고, 설명, 버전과 같은 정보로 사용자 인터페이스를 채우는 데 사용되는 Operator 컨테이너 이미지와 함께 제공되는 메타데이터입니다.

또한 필요한 RBAC 규칙 및 관리하거나 사용하는 CR(사용자 정의 리소스)과 같이 Operator를 실행하는 데 필요한 기술 정보의 소스이기도 합니다.

3.1.7. 종속성

Operator는 클러스터에 있는 다른 Operator에 *종속*되어 있을 수 있습니다. 예를 들어 Vault Operator는 데이터 지속성 계층과 관련하여 etcd Operator에 종속됩니다.

OLM은 설치 단계 동안 지정된 모든 버전의 Operator 및 CRD가 클러스터에 설치되도록 하여 종속 항목을 해결합니다. 이러한 종속성은 필수 CRD API를 충족하고 패키지 또는 번들과 관련이 없는 카탈로그에서 Operator를 찾아 설치함으로써 해결할 수 있습니다.

3.1.8. 인덱스 이미지

번들 형식에서 *인덱스 이미지*는 모든 버전의 CSV 및 CRD를 포함하여 Operator 번들에 대한 정보를 포함하는 데이터베이스 이미지(데이터베이스 스냅샷)를 나타냅니다. 이 인덱스는 클러스터에서 Operator 기록을 호스팅하고 opm CLI 툴을 사용하여 Operator를 추가하거나 제거하는 방식으로 유지 관리할 수 있습니다.

3.1.9. 설치 계획

*설치 계획*은 CSV를 자동으로 설치하거나 업그레이드하기 위해 생성하는 계산된 리소스 목록입니다.

3.1.10. 멀티 테넌시

OpenShift Container Platform의 *테넌트*는 배포된 워크로드 세트(일반적으로 네임스페이스 또는 프로젝트로 표시되는)에 대한 공통 액세스 및 권한을 공유하는 사용자 또는 사용자 그룹입니다. 테넌트를 사용하여 다른 그룹 또는 팀 간에 격리 수준을 제공할 수 있습니다.

여러 사용자 또는 그룹에서 클러스터를 공유하는 *경우 다중 테넌트* 클러스터로 간주됩니다.

3.1.11. Operator group

*Operator group*은 동일한 네임스페이스에 배포된 모든 **Operator**를 **OperatorGroup** 오브젝트로 구성하여 네임스페이스 목록 또는 클러스터 수준에서 **CR**을 조사합니다.

3.1.12. 패키지

번들 형식에서 *패키지*는 각 버전과 함께 **Operator**의 모든 릴리스 내역을 포함하는 디렉터리입니다. 릴리스된 **Operator** 버전은 **CRD**와 함께 **CSV** 매니페스트에 설명되어 있습니다.

3.1.13. 레지스트리

*레지스트리*는 각각 모든 채널의 최신 버전 및 이전 버전이 모두 포함된 **Operator**의 번들 이미지를 저장하는 데이터베이스입니다.

3.1.14. Subscription

*서브스크립션*은 패키지의 채널을 추적하여 **CSV**를 최신 상태로 유지합니다.

3.1.15. 업데이트 그래프

*업데이트 그래프*는 패키지된 다른 소프트웨어의 업데이트 그래프와 유사하게 **CSV** 버전을 함께 연결합니다. **Operator**를 순서대로 설치하거나 특정 버전을 건너뛸 수 있습니다. 업데이트 그래프는 최신 버전이 추가됨에 따라 앞부분에서만 증가할 것으로 예상됩니다.

4장. 카탈로그

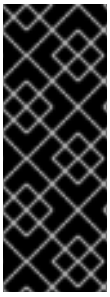
4.1. 파일 기반 카탈로그

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

OpenShift Container Platform의 **OLM(Operator Lifecycle Manager) v1**은 클러스터에서 **Operator**를 포함한 클러스터 확장 검색 및 소싱을 위해 *파일 기반 카탈로그*를 지원합니다.



중요

현재 **OLM(Operator Lifecycle Manager) v1**은 **Red Hat** 제공 **Operator** 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 **Red Hat Operator** 카탈로그를 설치하는 데 사용하는 **OLM v1** 절차가 작동하지 않습니다. (OCPBUGS-36364)

4.1.1. 주요 사항

*파일 기반 카탈로그*는 **OLM(Operator Lifecycle Manager)** 카탈로그 형식의 최신 버전입니다. 일반 텍스트 기반(**JSON** 또는 **YAML**)과 이전 **SQLite** 데이터베이스 형식의 선언적 구성 진화이며 완전히 이전 버전과 호환됩니다. 이 형식의 목표는 **Operator** 카탈로그 편집, 구성 가능성 및 확장성을 활성화하는 것입니다.

편집

파일 기반 카탈로그를 사용하면 카탈로그 내용과 상호 작용하는 사용자가 형식을 직접 변경하고 변경 사항이 유효한지 확인할 수 있습니다. 이 형식은 일반 텍스트 **JSON** 또는 **YAML**이므로 카탈로그 유지 관리자는 **jq CLI**와 같이 널리 알려진 지원되는 **JSON** 또는 **YAML** 툴을 사용하여 카탈로그 메타데이터를 쉽게 조작할 수 있습니다.

이 편집 기능을 사용하면 다음과 같은 기능 및 사용자 정의 확장 기능을 사용할 수 있습니다.

- 기존 번들을 새 채널로 승격
- 패키지의 기본 채널 변경
- 업그레이드 에지 추가, 업데이트 및 제거를 위한 사용자 정의 알고리즘

호환성

파일 기반 카탈로그는 카탈로그 구성이 가능한 임의의 디렉터리 계층 구조에 저장됩니다. 예를 들어 별도의 파일 기반 카탈로그 디렉터리인 **catalogA** 및 **catalogB**를 고려해 보십시오. 카탈로그 관리자는 새 디렉토리 **catalogC**를 만들고 **catalogA** 및 **catalogB**를 복사하여 새로 결합된 카탈로그를 만들 수 있습니다.

이 구성 가능성을 통해 분산된 카탈로그를 사용할 수 있습니다. 이 형식을 사용하면 **Operator** 작성자가 **Operator**별 카탈로그를 유지 관리할 수 있으며 유지 관리자가 개별 **Operator** 카탈로그로 구성된 카탈로그를 단순하게 빌드할 수 있습니다. 파일 기반 카탈로그는 하나의 카탈로그의 하위 집합을 추출하거나 두 개의 카탈로그를 조합하여 다른 여러 카탈로그를 결합하여 구성할 수 있습니다.

참고

패키지 내의 중복 패키지 및 중복 번들은 허용되지 않습니다. **opm validate** 명령은 중복이 발견되면 오류를 반환합니다.

Operator 작성자는 **Operator**, 해당 종속 항목 및 업그레이드 호환성에 가장 익숙하므로 자체 **Operator**별 카탈로그를 유지 관리하고 해당 콘텐츠를 직접 제어할 수 있습니다. **Operator** 작성자는 파일 기반 카탈로그를 사용하여 카탈로그에서 패키지를 빌드하고 유지 관리하는 작업을 소유합니다. 그러나 복합 카탈로그 유지 관리자만 카탈로그의 패키지를 큐레이션하고 카탈로그를 사용자에게 게시하는 작업만 소유합니다.

확장성

파일 기반 카탈로그 사양은 카탈로그의 하위 수준 형식입니다. 카탈로그 유지 관리자는 낮은 수준의 형식으로 직접 유지 관리할 수 있지만, 카탈로그 유지 관리자는 고유한 사용자 지정 툴링에서 원하는 수의 변경을 수행할 수 있는 확장 기능을 구축할 수 있습니다.

예를 들어 툴은 에지 업그레이드를 위해 높은 수준의 **API** (예:(mode=semver)를 낮은 수준의 파일 기반 카탈로그 형식으로 변환할 수 있습니다. 또는 카탈로그 유지 관리자는 특정 기준을 충족하는 번들에 새 속성을 추가하여 모든 번들 메타데이터를 사용자 지정해야 할 수 있습니다.

이러한 확장성을 통해 향후 **OpenShift Container Platform** 릴리스를 위해 하위 수준 **API**에서 추가 공식 툴을 개발할 수 있지만, 카탈로그 유지 관리자도 이러한 기능을 사용할 수 있다는 것이 가장 큰 장점입니다.

4.1.2. 디렉토리 구조

디렉토리 기반 파일 시스템에서 파일 기반 카탈로그를 저장하고 로드할 수 있습니다. **opm CLI**는 루트 디렉토리로 이동하고 하위 디렉토리로 재귀하여 카탈로그를 로드합니다. **CLI**는 발견한 모든 파일을 로드 시도하여 오류가 발생하면 실패합니다.

비카탈로그 파일은 **.gitignore** 파일과 패턴 및 우선 순위에 대해 동일한 규칙이 있는 **.indexignore** 파일을 사용하여 무시할 수 있습니다.

.indexignore 파일의 예

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

카탈로그 유지 관리자는 원하는 레이아웃을 선택할 수 있는 유연성을 가지지만 각 패키지의 파일 기반 카탈로그 **Blob**을 별도의 하위 디렉터리에 저장하는 것이 좋습니다. 각 개별 파일은 **JSON** 또는 **YAML**일 수 있습니다. 카탈로그의 모든 파일에서 동일한 형식을 사용할 필요는 없습니다.

기본 권장 구조

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

```
└── packageC
    └── index.json
    └── deprecations.yaml
```

이 권장 구조에는 디렉터리 계층 구조의 각 하위 디렉터리가 자체 포함된 카탈로그이므로 카탈로그 구성, 검색 및 탐색 간단한 파일 시스템 작업이 가능합니다. 카탈로그는 상위 카탈로그의 루트 디렉터리에 복사하여 상위 카탈로그에 포함할 수도 있습니다.

4.1.3. 스키마

파일 기반 카탈로그는 임의의 스키마로 확장할 수 있는 CUE 언어 사양을 기반으로 하는 형식을 사용합니다. 다음 _Meta CUE 스키마는 모든 파일 기반 카탈로그 Blob이 준수해야 하는 형식을 정의합니다.

_Meta 스키마

```
_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
}
```

참고

이 사양에 나열된 CUE 스키마는 완전한 것으로 간주되어서는 안 됩니다. opm validate 명령에는 CUE에서 간결하게 표현하기가 어렵거나 불가능한 추가 유효성 검사가 있습니다.

OLM(Operator Lifecycle Manager) 카탈로그에서는 현재 OLM의 기존 패키지 및 번들 개념에 해당하는 3개의 스키마 (olm.package, olm.channel, olm.bundle)를 사용합니다.

카탈로그의 각 Operator 패키지에는 정확히 하나의 olm.package blob, 하나 이상의 olm.channel blob, 하나 이상의 olm.bundle blob이 필요합니다.

참고

모든 olm.* 스키마는 OLM 정의 스키마용으로 예약됩니다. 사용자 지정 스키마는 소유한 도메인과 같이 고유한 접두사를 사용해야 합니다.

4.1.3.1. olm.package 스키마

olm.package 스키마는 Operator에 대한 패키지 수준 메타데이터를 정의합니다. 이에는 이름, 설명, 기본 채널 및 아이콘이 포함됩니다.

예 4.1. olm.package 스키마

```
#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
  }
}
```

4.1.3.2. olm.channel 스키마

olm.channel 스키마는 패키지 내의 채널, 채널의 멤버인 번들 항목 및 해당 번들의 업그레이드 에지를 정의합니다.

번들 항목이 여러 **olm.channel blob**의 에지를 나타내는 경우 채널당 한 번만 표시될 수 있습니다.

항목의 값이 이 카탈로그 또는 다른 카탈로그에서 찾을 수 없는 다른 번들 이름을 참조하는 것은 유효합니다. 그러나 여러 헤드가 없는 채널과 같이 다른 모든 채널 불변성은 **true**를 유지해야 합니다.

예 4.2. olm.channel 스키마

```
#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 & !=""
}
```

주의

skipRange 필드를 사용하는 경우 건너뛰는 Operator 버전이 업데이트 그래프에서 정리되며 **Subscription** 오브젝트의 **spec.startingCSV** 속성이 있는 사용자가 더 오래 설치할 수 있습니다.

skipRange 및 **replaces** 필드를 모두 사용하여 향후 설치를 위해 사용자가 이전에 설치한 버전을 사용할 수 있는 동안 Operator를 점진적으로 업데이트할 수 있습니다. **replaces** 필드가 해당 Operator 버전의 즉시 이전 버전을 가리키는지 확인합니다.

4.1.3.3. olm.bundle 스키마

예 4.3. olm.bundle 스키마

```
#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 & !=""
}
```

4.1.3.4. olm.deprecations 스키마

선택적 **olm.deprecations** 스키마는 카탈로그의 패키지, 번들 및 채널에 대한 사용 중단 정보를 정의

합니다. Operator 작성자는 이 스키마를 사용하여 카탈로그에서 해당 Operator를 실행하는 사용자에게 지원 상태 및 권장 업그레이드 경로와 같은 Operator에 대한 관련 메시지를 제공할 수 있습니다.

이 스키마가 정의되면 OpenShift Container Platform 웹 콘솔은 OperatorHub의 사전 및 설치 후 페이지에서 사용자 정의 사용 중단 메시지를 포함하여 Operator의 영향을 받는 요소에 대한 경고 배지를 표시합니다.

olm.deprecations 스키마 항목에는 사용 중단 범위를 나타내는 다음 참조 유형 중 하나 이상이 포함되어 있습니다. Operator가 설치되면 지정된 메시지를 관련 Subscription 오브젝트에서 상태 조건으로 볼 수 있습니다.

표 4.1. 사용 중단 참조 유형

유형	범위	상태 조건
olm.package	전체 패키지를 나타냅니다.	PackageDeprecated
olm.channel	하나의 채널을 나타냅니다.	ChannelDeprecated
olm.bundle	하나의 번들 버전을 나타냅니다.	BundleDeprecated

각 참조 유형에는 다음 예에 설명된 대로 자체 요구 사항이 있습니다.

예 4.4. 각 참조 유형이 있는 olm.deprecations 스키마의 예

```
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.
```

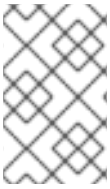
1. 각 사용 중단 스키마에는 패키지 값이 있어야 하며 해당 패키지 참조는 카탈로그 전체에서 고유해야 합니다. 연결된 이름 필드가 없어야 합니다.

2. **olm.package** 스키마는 스키마의 앞부분에서 정의한 **package** 필드에 의해 결정되므로 **name** 필드를 포함하지 않아야 합니다.

3. 모든 참조 유형에 대한 모든 메시지 필드는 **0**이 아닌 길이이고 불투명 텍스트 **Blob**으로 표현되어야 합니다.

4. **olm.channel** 스키마의 **name** 필드가 필요합니다.

5. **olm.bundle** 스키마의 **name** 필드가 필요합니다.



참고

사용 중단 기능은 중복 사용 중단(예: 패키지 대 채널 대 번들)을 고려하지 않습니다.

Operator 작성자는 **olm.deprecations** 스키마 항목을 패키지의 **index.yaml** 파일과 동일한 디렉터리에 **deprecations.yaml** 파일로 저장할 수 있습니다.

사용 중단이 있는 카탈로그의 디렉터리 구조 예

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

추가 리소스

- 파일 기반 카탈로그 이미지 업데이트 또는 필터링

4.1.4. 속성

속성은 파일 기반 카탈로그 스키마에 연결할 수 있는 임의 메타데이터입니다. **type** 필드는 **value** 필드의 의미 및 구문 의미를 효과적으로 지정하는 문자열입니다. 이 값은 임의의 **JSON** 또는 **YAML**일 수 있습니다.

OLM은 예약된 **olm.*** 접두사를 사용하여 몇 가지 속성 유형을 다시 정의합니다.

4.1.4.1. olm.package 속성

olm.package 속성은 패키지 이름과 버전을 정의합니다. 이는 번들의 필수 속성이며, 이러한 속성 중 정확히 하나가 있어야 합니다. **packageName** 필드는 번들의 최상위 **package** 필드와 일치해야 하며 **version** 필드는 유효한 의미 체계 버전이어야 합니다.

예 4.5. olm.package 속성

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

4.1.4.2. olm.gvk 속성

olm.gvk 속성은 이 번들에서 제공하는 **Kubernetes API**의 **GVK**(그룹/버전/종류)를 정의합니다. 이 속성은 **OLM**에서 이 속성이 포함된 번들을 필수 **API**와 동일한 **GVK**를 나열하는 다른 번들의 종속성으로 확인하는 데 사용됩니다. **GVK**는 **Kubernetes GVK** 검증을 준수해야 합니다.

예 4.6. olm.gvk 속성

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

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

4.1.4.3. olm.package.required

olm.package.required 속성은 이 번들에 필요한 다른 패키지의 패키지 이름 및 버전 범위를 정의합니다. 번들 목록이 필요한 모든 패키지 속성에 대해 **OLM**은 나열된 패키지 및 필수 버전 범위에 대한 클러스터에 **Operator**가 설치되어 있는지 확인합니다. **versionRange** 필드는 유효한 의미 버전(**semver**) 범위여야 합니다.

예 4.7. olm.package.required 속성

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

4.1.4.4. olm.gvk.required

olm.gvk.required 속성은 이 번들에 필요한 **Kubernetes API**의 **GVK**(그룹/버전/종류)를 정의합니다. 번들 목록이 필요한 모든 **GVK** 속성에 대해 **OLM**은 이를 제공하는 클러스터에 **Operator**가 설치되어 있는지 확인합니다. **GVK**는 **Kubernetes GVK** 검증을 준수해야 합니다.

예 4.8. olm.gvk.required 속성

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

4.1.5. 카탈로그의 예

파일 기반 카탈로그를 사용하면 카탈로그 유지 관리자가 **Operator** 큐레이션 및 호환성에 중점을 둘 수 있습니다. **Operator** 작성자는 **Operator**에 대한 **Operator**별 카탈로그를 이미 생성했기 때문에 카탈로그

유지 관리자는 각 **Operator** 카탈로그를 카탈로그의 루트 디렉터리의 하위 디렉터리에 렌더링하여 카탈로그를 빌드할 수 있습니다.

파일 기반 카탈로그를 구축하는 방법은 여러 가지가 있습니다. 다음 단계에서는 간단한 접근 방식을 간략하게 설명합니다.

1. 카탈로그의 각 **Operator**에 대한 이미지 참조가 포함된 카탈로그의 단일 구성 파일을 유지 관리합니다.

카탈로그 구성 파일 예

```
name: community-operators
repo: quay.io/community-operators/catalog
tag: latest
references:
- name: etcd-operator
  image: quay.io/etcd-
operator/index@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2
e846f6be03
- name: prometheus-operator
  image: quay.io/prometheus-
operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d
84101eb317
```

2. 구성 파일을 구문 분석하고 참조에서 새 카탈로그를 생성하는 스크립트를 실행합니다.

스크립트 예

```
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"
```

4.1.6. 지침

파일 기반 카탈로그를 유지 관리할 때 다음 지침을 고려하십시오.

4.1.6.1. 변경할 수 없는 번들

OLM(Operator Lifecycle Manager)의 일반적인 지침은 번들 이미지 및 해당 메타데이터를 변경할 수 없음으로 간주해야 한다는 것입니다.

손상된 번들이 카탈로그로 푸시된 경우 사용자 중 한 명이 해당 번들로 업그레이드되었다고 가정해야 합니다. 이러한 가정에 따라 손상된 번들이 설치된 사용자에게 업그레이드를 수신하려면 손상된 번들에서 업그레이드 엣지를 사용하여 다른 번들을 릴리스해야 합니다. 해당 번들의 콘텐츠가 카탈로그에서 업데이트되는 경우 **OLM**은 설치된 번들을 다시 설치하지 않습니다.

그러나 카탈로그 메타데이터의 변경이 권장되는 몇 가지 사례가 있습니다.

- 채널 승격: 번들을 이미 릴리스하고 나중에 다른 채널에 추가할 것을 결정한 경우, 다른 **olm.channel blob**에 번들에 대한 항목을 추가할 수 있습니다.
- 새로운 업그레이드 에지: 새 **1.2.z** 번들 버전(예: **1.2.4**)을 릴리스했지만 **1.3.0**이 이미 릴리스된 경우 **1.3.0**의 카탈로그 메타데이터를 업데이트하여 **1.2.4**를 건너뛸 수 있습니다.

4.1.6.2. 소스 제어

카탈로그 메타데이터는 소스 제어에 저장되고 정보 소스로 처리되어야 합니다. 카탈로그 이미지 업데이트에는 다음 단계가 포함되어야 합니다.

1. 소스 제어 카탈로그 디렉터리를 새 커밋으로 업데이트합니다.
2. 카탈로그 이미지를 빌드하고 내보냅니다. 사용자가 사용 가능하게 되면 카탈로그 업데이트

를 수신할 수 있도록 **:latest** 또는 **:<target_cluster_version>**과 같은 일관된 태그 지정 용어를 사용합니다.

4.1.7. CLI 사용

opm CLI를 사용하여 파일 기반 카탈로그를 생성하는 방법에 대한 자세한 내용은 사용자 정의 카탈로그 관리를 참조하십시오.

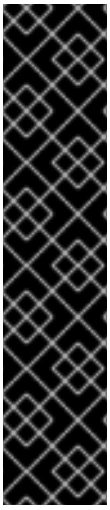
파일 기반 카탈로그 관리와 관련된 **opm CLI** 명령에 대한 참조 문서는 **CLI** 툴 을 참조하십시오.

4.1.8. 자동화

Operator 작성자 및 카탈로그 유지 관리자는 **CI/CD** 워크플로를 사용하여 카탈로그 유지 관리를 자동화하는 것이 좋습니다. 카탈로그 유지 관리자는 다음 작업을 수행하도록 **GitOps** 자동화를 빌드하여 이 작업을 추가로 개선할 수 있습니다.

- 예를 들어 패키지의 이미지 참조를 업데이트하여 해당 **PR(pull request)** 작성자가 요청된 변경을 수행할 수 있는지 확인합니다.
- 카탈로그 업데이트가 **opm validate** 명령을 전달하는지 확인합니다.
- 업데이트된 번들 또는 카탈로그 이미지 참조가 있는지, 카탈로그 이미지가 클러스터에서 성공적으로 실행되며 해당 패키지의 **Operator**가 성공적으로 설치될 수 있는지 확인합니다.
- 이전 검사를 통과하는 **PR**을 자동으로 병합합니다.
- 카탈로그 이미지를 자동으로 다시 빌드하고 다시 게시합니다.

4.2. RED HAT 제공 카탈로그

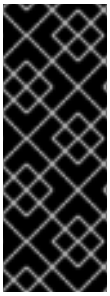


중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

Red Hat은 기본적으로 **OpenShift Container Platform**에 포함된 여러 **Operator** 카탈로그를 제공합니다.



중요

현재 **OLM(Operator Lifecycle Manager) v1**은 **Red Hat** 제공 **Operator** 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 **Red Hat Operator** 카탈로그를 설치하는 데 사용하는 **OLM v1** 절차가 작동하지 않습니다. (**OCPBUGS-36364**)

4.2.1. Red Hat 제공 Operator 카탈로그 정보

Red Hat 제공 카탈로그 소스는 **openshift-marketplace** 네임스페이스에 기본적으로 설치되므로 모든 네임스페이스에서 카탈로그를 클러스터 전체에서 사용할 수 있습니다.

다음은 **Red Hat**에서 제공하는 **Operator** 카탈로그입니다.

카탈로그	인덱스 이미지	설명
redhat-operators	**registry.redhat.io/redhat/redhat-operator-index:v4.17**	Red Hat에서 Red Hat 제품을 패키지 및 제공합니다. Red Hat에서 지원합니다.
certified-operators	**registry.redhat.io/redhat/certified-operator-index:v4.17**	선도적인 ISV(독립 소프트웨어 벤더)의 제품입니다. Red Hat은 패키지 및 제공을 위해 ISV와 협력합니다. ISV에서 지원합니다.

카탈로그	인덱스 이미지	설명
redhat-marketplace	**registry.redhat.io/redhat/redhat-marketplace-index:v4.17**	Red Hat Marketplace에서 구매할 수 있는 인증 소프트웨어입니다.
community-operators	**registry.redhat.io/redhat/community-operator-index:v4.17**	redhat-openshift-ecosystem/community-operators-prod/operators GitHub 리포지토리의 관련 담당자가 유지 관리하는 소프트웨어입니다. 공식적으로 지원되지 않습니다.

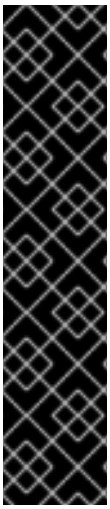
클러스터 업그레이드 중에 기본 **Red Hat** 제공 카탈로그 소스의 인덱스 이미지 태그는 **CVO(Cluster Version Operator)**에서 자동으로 업데이트하여 **OLM(Operator Lifecycle Manager)**이 업데이트된 버전의 카탈로그를 가져옵니다. 예를 들어 **OpenShift Container Platform 4.8**에서 **4.9**로 업그레이드하는 동안 **redhat-operators** 카탈로그의 **CatalogSource** 오브젝트의 **spec.image** 필드가 업데이트됩니다.

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

다음으로 변경합니다.

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

4.3. 카탈로그 관리



중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

클러스터 관리자는 *카탈로그* 또는 **Operator** 및 **Kubernetes** 확장의 컬렉션을 클러스터에 추가할 수 있습니다. **Operator** 작성자는 해당 카탈로그에 제품을 게시합니다. 클러스터에 카탈로그를 추가할 때 카탈

로그에 게시된 Operator 및 확장의 버전, 패치 및 무선 업데이트에 액세스할 수 있습니다.

CR(사용자 정의 리소스)을 사용하여 CLI에서 카탈로그 및 확장을 선언적으로 관리할 수 있습니다.

*파일 기반 카탈로그*는 OLM(Operator Lifecycle Manager) 카탈로그 형식의 최신 버전입니다. 일반 텍스트 기반(JSON 또는 YAML)과 이전 SQLite 데이터베이스 형식의 선언적 구성 진화이며 완전히 이전 버전과 호환됩니다.

중요

Kubernetes는 후속 릴리스에서 제거된 특정 API를 주기적으로 사용하지 않습니다. 결과적으로 Operator는 API를 제거한 Kubernetes 버전을 사용하는 OpenShift Container Platform 버전에서 시작하여 제거된 API를 사용할 수 없습니다.

클러스터가 사용자 정의 카탈로그를 사용하는 경우 Operator 작성자가 워크로드 문제를 방지하고 호환되지 않는 업그레이드를 방지하는 방법에 대한 자세한 내용은 OpenShift Container Platform 버전과 Operator 호환성 제어를 참조하십시오.

4.3.1. OLM v1의 카탈로그 정보

카탈로그 구성 요소를 사용하여 Operator 및 컨트롤러와 같은 Kubernetes 확장 카탈로그를 쿼리하여 설치 가능한 콘텐츠를 검색할 수 있습니다. Catalogd는 클러스터 내 클라이언트의 카탈로그 콘텐츠의 압축을 풀고 OLM(Operator Lifecycle Manager) v1 마이크로 서비스 제품군의 일부입니다. 현재 catalogd는 컨테이너 이미지로 패키지 및 배포되는 카탈로그 콘텐츠의 압축을 풉니다.

중요

고유한 이름이 없는 Operator 또는 확장을 설치하려고 하면 설치에 실패하거나 예기치 않은 결과가 발생할 수 있습니다. 이는 다음과 같은 이유로 발생합니다.

- mulitple 카탈로그가 클러스터에 설치된 경우 OLM(Operator Lifecycle Manager) v1에는 Operator 또는 확장을 설치할 때 카탈로그를 지정하는 메커니즘이 포함되지 않습니다.
- OLM v1에서는 클러스터에 설치할 수 있는 모든 Operator 및 확장이 번들 및 패키지에 고유한 이름을 사용해야 합니다.

추가 리소스

- 파일 기반 카탈로그

4.3.2. OLM v1의 Red Hat 제공 Operator 카탈로그

OLM(Operator Lifecycle Manager) v1에는 기본적으로 Red Hat 제공 Operator 카탈로그가 포함되어 있지 않습니다. Red Hat 제공 카탈로그를 클러스터에 추가하려면 카탈로그의 CR(사용자 정의 리소스)을 생성하여 클러스터에 적용합니다. 다음 CR(사용자 정의 리소스) 예제에서는 OLM v1에 대한 카탈로그 리소스를 생성하는 방법을 보여줍니다.

중요

- 현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

- registry.redhat.io 에서 Red Hat 제공 Operator 카탈로그와 같이 프라이빗 레지스트리에서 호스팅되는 카탈로그를 사용하려면 openshift-catalogd 네임스페이스에 풀 시크릿 범위가 지정되어야 합니다.

  자세한 내용은 "보안 레지스트리에서 호스팅되는 카탈로그의 풀 시크릿 생성"을 참조하십시오.

Red Hat Operator 카탈로그의 예

```
apiVersion: catalogd.operatorframework.io/v1alpha1
kind: ClusterCatalog
metadata:
  name: redhat-operators
spec:
  source:
    type: image
    image:
      ref: registry.redhat.io/redhat/redhat-operator-index:v4.17
      pullSecret: <pull_secret_name>
      pollInterval: <poll_interval_duration> 1
```

-

1

최신 이미지 다이제스트를 위해 원격 레지스트리를 폴링하는 간격을 지정합니다. 기본값은 **24h** 입니다. 유효한 단위에는 초(**s**), 분(**m**) 및 시간(**h**)이 포함됩니다. 폴링을 비활성화하려면 **0s** 와 같은 **0** 값을 설정합니다.

인증된 **Operator** 카탈로그의 예

```
apiVersion: catalogd.operatorframework.io/v1alpha1
kind: ClusterCatalog
metadata:
  name: certified-operators
spec:
  source:
    type: image
    image:
      ref: registry.redhat.io/redhat/certified-operator-index:v4.17
      pullSecret: <pull_secret_name>
      pollInterval: 24h
```

커뮤니티 **Operator** 카탈로그의 예

```
apiVersion: catalogd.operatorframework.io/v1alpha1
kind: ClusterCatalog
metadata:
  name: community-operators
spec:
  source:
    type: image
    image:
      ref: registry.redhat.io/redhat/community-operator-index:v4.17
      pullSecret: <pull_secret_name>
      pollInterval: 24h
```

다음 명령은 클러스터에 카탈로그를 추가합니다.

명령 구문

```
$ oc apply -f <catalog_name>.yaml 1
```

1

redhat-operators.yaml 과 같은 카탈로그 CR을 지정합니다.

4.3.3. 프라이빗 레지스트리에서 호스팅되는 카탈로그의 풀 시크릿 생성

registry.redhat.io 에서 Red Hat 제공 Operator 카탈로그와 같이 프라이빗 레지스트리에서 호스팅되는 카탈로그를 사용하려면 openshift-catalogd 네임스페이스에 풀 시크릿 범위가 지정되어야 합니다.

Catalogd는 OpenShift Container Platform 클러스터에서 글로벌 풀 시크릿을 읽을 수 없습니다. Catalogd는 배포된 네임스페이스에서만 보안에 대한 참조를 읽을 수 있습니다.

중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

사전 요구 사항

- 보안 레지스트리의 로그인 인증 정보
- 워크스테이션에 Docker 또는 Podman이 설치되어 있어야 합니다.

프로세스

-

보안 레지스트리에 대한 로그인 인증 정보가 있는 **.dockercfg** 파일이 이미 있는 경우 다음 명령을 실행하여 가져오기 보안을 생성합니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockercfg=<file_path>/.dockercfg \
    --type=kubernetes.io/dockercfg \
    --namespace=openshift-catalogd
```

예 4.9. 명령 예

```
$ oc create secret generic redhat-cred \
    --from-file=.dockercfg=/home/<username>/.dockercfg \
    --type=kubernetes.io/dockercfg \
    --namespace=openshift-catalogd
```

- 보안 레지스트리의 로그인 인증 정보가 있는 **$HOME/.docker/config.json** 파일이 이미 있는 경우 다음 명령을 실행하여 풀 시크릿을 생성합니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<file_path>/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson \
    --namespace=openshift-catalogd
```

예 4.10. 명령 예

```
$ oc create secret generic redhat-cred \
    --from-file=.dockerconfigjson=/home/<username>/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson \
    --namespace=openshift-catalogd
```

- 보안 레지스트리에 대한 로그인 인증 정보가 있는 **Docker** 구성 파일이 없는 경우 다음 명령을 실행하여 가져오기 보안을 생성합니다.

```
$ oc create secret docker-registry <pull_secret_name> \
    --docker-server=<registry_server> \
    --docker-username=<username> \
    --docker-password=<password> \
    --docker-email=<email> \
    --namespace=openshift-catalogd
```

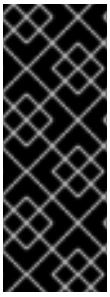
예 4.11. 명령 예

```
$ oc create secret docker-registry redhat-cred \
    --docker-server=registry.redhat.io \
    --docker-username=username \
```

```
    --docker-password=password \
    --docker-email=user@example.com \
    --namespace=openshift-catalogd
```

4.3.4. 클러스터에 카탈로그 추가

클러스터에 카탈로그를 추가하려면 카탈로그 CR(사용자 정의 리소스)을 생성하여 클러스터에 적용합니다.



중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

사전 요구 사항

- registry.redhat.io 에서 Red Hat 제공 Operator 카탈로그와 같이 프라이빗 레지스트리에서 호스팅되는 카탈로그를 사용하려면 openshift-catalogd 네임스페이스에 풀 시크릿 범위가 지정되어야 합니다.

Catalogd는 OpenShift Container Platform 클러스터에서 글로벌 풀 시크릿을 읽을 수 없습니다. Catalogd는 배포된 네임스페이스에서만 보안에 대한 참조를 읽을 수 있습니다.

프로세스

1. 다음 예와 유사한 카탈로그 CR(사용자 정의 리소스)을 생성합니다.

예: redhat-operators.yaml

```
apiVersion: catalogd.operatorframework.io/v1alpha1
kind: ClusterCatalog
metadata:
  name: redhat-operators
spec:
  source:
```

```
  type: image
  image:
    ref: registry.redhat.io/redhat/redhat-operator-index:v4.17 (1)
    pullSecret: <pull_secret_name> (2)
    pollInterval: <poll_interval_duration> (3)
```

(1) spec.source.image 필드에 카탈로그의 이미지를 지정합니다.

(2) 카탈로그가 registry.redhat.io 와 같은 보안 레지스트리에서 호스팅되는 경우 openshift-catalog 네임스페이스에 대한 풀 시크릿 범위를 생성해야 합니다.

(3) 최신 이미지 다이제스트를 위해 원격 레지스트리를 폴링하는 간격을 지정합니다. 기본값은 24h 입니다. 유효한 단위에는 초(s), 분(m) 및 시간(h)이 포함됩니다. 폴링을 비활성화하려면 0s 와 같은 0 값을 설정합니다.

2. 다음 명령을 실행하여 클러스터에 카탈로그를 추가합니다.

```
$ oc apply -f redhat-operators.yaml
```

출력 예

```
catalog.catalogd.operatorframework.io/redhat-operators created
```

검증

- 다음 명령을 실행하여 카탈로그 상태를 확인합니다.

  a. 다음 명령을 실행하여 카탈로그를 사용할 수 있는지 확인합니다.

  -

```
$ oc get clustercatalog
```

출력 예

```
NAME                AGE
redhat-operators    20s
```

b. 다음 명령을 실행하여 카탈로그의 상태를 확인합니다.

```
$ oc describe clustercatalog
```

출력 예

```
Name:         redhat-operators
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  catalogd.operatorframework.io/v1alpha1
Kind:         ClusterCatalog
Metadata:
  Creation Timestamp:  2024-06-10T17:34:53Z
  Finalizers:
    catalogd.operatorframework.io/delete-server-cache
  Generation:        1
  Resource Version:  46075
  UID:               83c0db3c-a553-41da-b279-9b3cddaa117d
Spec:
  Source:
    Image:
      Pull Secret:  redhat-cred
      Ref:          registry.redhat.io/redhat/redhat-operator-index:v4.17
    Type:           image
Status: (1)
  Conditions:
    Last Transition Time:  2024-06-10T17:35:15Z
    Message:
    Reason:                UnpackSuccessful (2)
    Status:                True
    Type:                  Unpacked
  Content URL:             https://catalogd-catalogserver.openshift-
catalogd.svc/catalogs/redhat-operators/all.json
  Observed Generation:     1
  Phase:                   Unpacked (3)
```

```
  Resolved Source:
    Image:
      Last Poll Attempt:  2024-06-10T17:35:10Z
      Ref:                registry.redhat.io/redhat/redhat-operator-index:v4.17
      Resolved Ref:       registry.redhat.io/redhat/redhat-operator-
index@sha256:f2ccc079b5e490a50db532d1dc38fd659322594dcf3e653d650ead0e86
2029d9 4
    Type:               image
Events:                 <none>
```

1 카탈로그 상태를 설명합니다.

2 카탈로그가 현재 상태에 있는 이유를 표시합니다.

3 설치 프로세스의 단계를 표시합니다.

4 카탈로그의 이미지 참조를 표시합니다.

4.3.5. 카탈로그 삭제

CR(사용자 정의 리소스)을 삭제하여 카탈로그를 삭제할 수 있습니다.

사전 요구 사항

- 카탈로그가 설치되어 있어야 합니다.

프로세스

- 다음 명령을 실행하여 카탈로그를 삭제합니다.

```
$ oc delete clustercatalog <catalog_name>
```

출력 예

```
catalog.catalogd.operatorframework.io "my-catalog" deleted
```

검증

- 다음 명령을 실행하여 카탈로그가 삭제되었는지 확인합니다.

```
$ oc get clustercatalog
```

4.4. 카탈로그 생성

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

카탈로그 유지 관리자는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager) v1과 함께 사용할 파일 기반 카탈로그 형식으로 새 카탈로그를 생성할 수 있습니다.

중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

4.4.1. 파일 기반 카탈로그 이미지 생성

opm CLI를 사용하여 더 이상 사용되지 않는 SQLite 데이터베이스 형식을 대체하는 일반 텍스트 *파일 기반 카탈로그* 형식(JSON 또는 YAML)을 사용하는 카탈로그 이미지를 생성할 수 있습니다.

사전 요구 사항

- opm CLI를 설치했습니다.
- podman 버전 1.9.3 이상이 있습니다.
- 번들 이미지가 빌드되어 Docker v2-2 를 지원하는 레지스트리로 푸시됩니다.

프로세스

1. 카탈로그를 초기화합니다.
   a. 다음 명령을 실행하여 카탈로그의 디렉터리를 생성합니다.

```
$ mkdir <catalog_dir>
```

   b. opm generate dockerfile 명령을 실행하여 카탈로그 이미지를 빌드할 수 있는 Dockerfile을 생성합니다.

```
$ opm generate dockerfile <catalog_dir> \
    -i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4.17 1
```

1 -i 플래그를 사용하여 공식 Red Hat 기본 이미지를 지정합니다. 그러지 않으면 Dockerfile에서 기본 업스트림 이미지를 사용합니다.

Dockerfile은 이전 단계에서 생성한 카탈로그 디렉터리와 동일한 상위 디렉터리에 있어야 합니다.

디렉터리 구조의 예

```
.
├── <catalog_dir>
└── <catalog_dir>.Dockerfile
```

1 상위 디렉터리

2 카탈로그 디렉터리

3 **opm generate dockerfile** 명령으로 생성된 Dockerfile

c. **opm init** 명령을 실행하여 카탈로그를 Operator의 패키지 정의로 채웁니다.

```
$ opm init <operator_name> \
  --default-channel=preview \
  --description=./README.md \
  --icon=./operator-icon.svg \
  --output yaml \
  > <catalog_dir>/index.yaml
```

1 Operator 또는 패키지, 이름

2 서브스크립션이 지정되지 않은 경우 기본값으로 설정된 채널

3 Operator의 **README.md** 또는 기타 문서 경로

4 Operator 아이콘 경로

5

출력 형식: JSON 또는 YAML

6

카탈로그 구성 파일을 생성하는 경로

이 명령은 지정된 카탈로그 구성 파일에 olm.package 선언적 구성 blob을 생성합니다.

2. opm render 명령을 실행하여 카탈로그에 번들을 추가합니다.

```
$ opm render <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
  --output=yaml \
  >> <catalog_dir>/index.yaml 2
```

1

번들 이미지의 가져오기 사양

2

카탈로그 구성 파일의 경로

참고

채널에는 하나 이상의 번들이 포함되어야 합니다.

3. 번들에 채널 항목을 추가합니다. 예를 들어 다음 예제를 사양에 맞게 수정하고 <catalog_dir>/index.yaml 파일에 추가합니다.

채널 항목 예

```
---
schema: olm.channel
package: <operator_name>
```

```
name: preview
entries:
 - name: <operator_name>.v0.1.0 1
```

1

<operator_name> 뒤의 마침표(.)를 버전 v 앞에 포함해야 합니다. 그렇지 않으면 항목이 opm validate 명령을 전달하지 못합니다.

4. 파일 기반 카탈로그를 확인합니다.

   a. 카탈로그 디렉터리에 대해 opm validate 명령을 실행합니다.

   ```
   $ opm validate <catalog_dir>
   ```

   b. 오류 코드가 0인지 확인합니다.

   ```
   $ echo $?
   ```

   출력 예

   ```
   0
   ```

5. podman build 명령을 실행하여 카탈로그 이미지를 빌드합니다.

```
$ podman build . \
   -f <catalog_dir>.Dockerfile \
   -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. 카탈로그 이미지를 레지스트리로 푸시합니다.

a. 필요한 경우 **podman login** 명령을 실행하여 대상 레지스트리로 인증합니다.

```
$ podman login <registry>
```

b. **podman push** 명령을 실행하여 카탈로그 이미지를 푸시합니다.

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

추가 리소스

- opm CLI 참조

4.4.2. 파일 기반 카탈로그 이미지 업데이트 또는 필터링

opm CLI를 사용하여 파일 기반 카탈로그 형식을 사용하는 카탈로그 이미지를 업데이트하거나 필터링할 수 있습니다. 기존 카탈로그 이미지의 콘텐츠를 추출하면 필요에 따라 카탈로그를 수정할 수 있습니다. 예를 들면 다음과 같습니다.

- 패키지 추가
- 패키지 제거
- 기존 패키지 항목 업데이트
- 패키지, 채널 및 번들당 사용 중단 메시지 세부 정보

그런 다음 업데이트된 카탈로그 버전으로 이미지를 다시 빌드할 수 있습니다.

참고

또는 미러 레지스트리에 카탈로그 이미지가 이미 있는 경우 **oc-mirror CLI** 플러그인을 사용하여 업데이트된 카탈로그 버전의 해당 카탈로그 이미지에서 제거된 이미지를 자동으로 정리하고 대상 레지스트리에 미러링할 수 있습니다.

oc-mirror 플러그인 및 이 사용 사례에 대한 자세한 내용은 "미러 미러 레지스트리 콘텐츠 업데이트" 섹션, 특히 "**oc-mirror** 플러그인을 사용하여 연결이 끊긴 설치를 위한 이미지 미러링" 섹션, 특히 "이미지 실행" 섹션을 참조하십시오.

사전 요구 사항

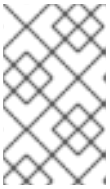
- 워크스테이션에 다음이 있습니다.
  - **opm CLI**입니다.
  - **podman** 버전 **1.9.3** 이상.
  - 파일 기반 카탈로그 이미지입니다.
  - 이 카탈로그와 관련된 워크스테이션에서 최근에 초기화된 카탈로그 디렉터리 구조입니다.

    초기화된 카탈로그 디렉터리가 없는 경우 디렉터리를 생성하고 **Dockerfile**을 생성합니다. 자세한 내용은 "파일 기반 카탈로그 이미지 생성" 절차의 "**catalog**" 단계를 참조하십시오.

프로세스

1. **YAML** 형식의 카탈로그 이미지의 콘텐츠를 카탈로그 디렉터리의 **index.yaml** 파일에 추출합니다.

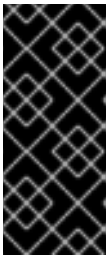
```
$ opm render <registry>/<namespace>/<catalog_image_name>:<tag> \
    -o yaml > <catalog_dir>/index.yaml
```



참고

또는 **-o json** 플래그를 사용하여 **JSON** 형식으로 출력할 수 있습니다.

2. 결과 **index.yaml** 파일의 내용을 사양으로 수정합니다.



중요

번들이 카탈로그에 게시되면 사용자 중 하나가 설치되었다고 가정합니다. 해당 버전이 설치된 사용자를 방지하려면 카탈로그의 이전에 게시된 모든 번들에 현재 또는 최신 채널 헤드에 대한 업데이트 경로가 있는지 확인합니다.

- **Operator**를 추가하려면 "파일 기반 카탈로그 이미지 생성" 프로세스에서 패키지, 번들 및 채널 항목을 생성하는 단계를 수행합니다.

- **Operator**를 제거하려면 패키지와 관련된 **olm.package,olm.channel, olm.bundle blobs** 세트를 삭제합니다. 다음 예제에서는 카탈로그에서 **example-operator** 패키지를 제거하려면 삭제해야 하는 세트를 보여줍니다.

예 4.12. 삭제된 항목의 예

```
---
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
---
```

- Operator에 대한 사용 중단 메시지를 추가하거나 업데이트하려면 패키지의 index.yaml 파일과 동일한 디렉터리에deprecations.yaml 파일이 있는지 확인합니다. deprecations.yaml 파일 형식에 대한 자세한 내용은 "olm.deprecations 스키마"를 참조하십시오.

3. 변경 사항을 저장하십시오.

4. 카탈로그를 확인합니다.

```
$ opm validate <catalog_dir>
```

5. 카탈로그를 다시 빌드합니다.

```
$ podman build . \
   -f <catalog_dir>.Dockerfile \
   -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. 업데이트된 카탈로그 이미지를 레지스트리로 푸시합니다.

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

검증

1. 웹 콘솔에서 관리 → 클러스터 설정 → 구성 페이지의 **OperatorHub** 구성 리소스로 이동합니다.

2. 업데이트된 카탈로그 이미지의 **pull** 사양을 사용하도록 카탈로그 소스를 추가하거나 기존 카탈로그 소스를 업데이트합니다.

   자세한 내용은 이 섹션의 "추가 리소스"의 "클러스터에 카탈로그 소스 추가"를 참조하십시오.

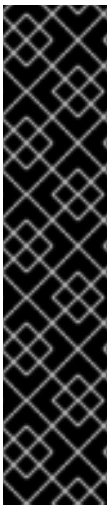
3. 카탈로그 소스가 **READY** 상태인 후 **Operator** → **OperatorHub** 페이지로 이동하여 수행된 변경 사항이 **Operator** 목록에 반영되었는지 확인합니다.

추가 리소스

- 패키징 형식 → 스키마 → **olm.deprecations** 스키마
- **oc-mirror** 플러그인을 사용하여 연결이 끊긴 설치의 이미지 미러링 → 미러 레지스트리 콘텐츠 유지
- 클러스터에 카탈로그 소스 추가

5장. 클러스터 확장

5.1. 클러스터 확장 관리



중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

카탈로그가 클러스터에 추가되면 카탈로그에 게시된 확장 및 **Operator**의 버전, 패치 및 무선 업데이트에 액세스할 수 있습니다.

CR(사용자 정의 리소스)을 사용하여 **CLI**에서 확장 기능을 선언적으로 관리할 수 있습니다.

중요

현재 **OLM(Operator Lifecycle Manager) v1**은 **Red Hat** 제공 **Operator** 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 **Red Hat Operator** 카탈로그를 설치하는 데 사용하는 **OLM v1** 절차가 작동하지 않습니다. (**OCPBUGS-36364**)

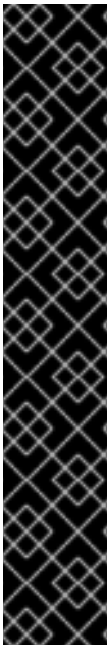
5.1.1. 지원되는 확장

현재 **OLM(Operator Lifecycle Manager) v1**에서는 다음 기준을 모두 충족하는 클러스터 확장 설치를 지원합니다.

- 확장 기능은 기존 **OLM**에 도입된 **registry+v1** 번들 형식을 사용해야 합니다.
- 확장 기능은 **AllNamespaces** 설치 모드를 통한 설치를 지원해야 합니다.

- 확장에서는 **Webhook**를 사용하지 않아야 합니다.
- 확장자는 다음 파일 기반 카탈로그 속성을 사용하여 종속성을 선언해서는 안 됩니다.
  - **olm.gvk.required**
  - **olm.package.required**
  - **olm.constraint**

OLM v1은 설치하려는 확장이 이러한 제약 조건을 충족하는지 확인합니다. 설치하려는 확장이 이러한 제약 조건을 충족하지 않으면 클러스터 확장 상태에 오류 메시지가 출력됩니다.



중요

OLM(Operator Lifecycle Manager) v1은 기존 **OLM**에 도입된 **OperatorConditions API**를 지원하지 않습니다.

확장 프로그램이 **OperatorConditions API**만 사용하여 업데이트를 관리하는 경우 확장이 올바르게 설치되지 않을 수 있습니다. 이 **API**에 의존하는 대부분의 확장은 시작 시 실패하지만 조정 중에 일부 확장이 실패할 수 있습니다.

이 문제를 해결하려면 확장 기능을 특정 버전에 고정할 수 있습니다. 확장을 업데이트하려는 경우 확장 기능을 참조하여 확장 기능을 새 버전에 고정하는 것이 안전한지 확인합니다.

추가 리소스

- Operator 상태

5.1.2. 카탈로그에서 설치할 Operator 찾기

클러스터에 카탈로그를 추가한 후 카탈로그를 쿼리하여 설치할 **Operator** 및 확장을 찾을 수 있습니다. 카탈로그를 쿼리하려면 먼저 카탈로그 서버 서비스를 전달하도록 포트해야 합니다.

사전 요구 사항

- 클러스터에 카탈로그를 추가했습니다.
- jq CLI 툴을 설치했습니다.

프로세스

1. 포트 다음 명령을 실행하여 openshift-catalogd 네임스페이스에서 카탈로그 서버 서비스를 전달합니다.

```
$ oc -n openshift-catalogd port-forward svc/catalogd-catalogserver 8080:443
```

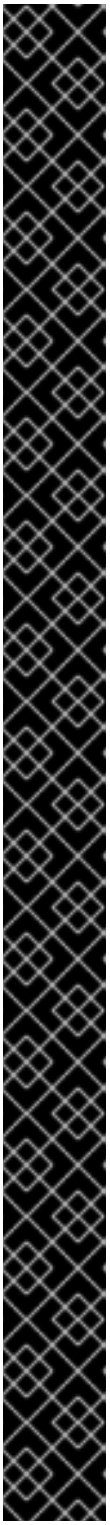
2. 새 터미널 창 또는 탭에서 다음 명령을 실행하여 카탈로그의 JSON 파일을 로컬로 다운로드합니다.

```
$ curl -L -k https://localhost:8080/catalogs/<catalog_name>/all.json \
  -C - -o /<path>/<catalog_name>.json
```

예 5.1. 명령 예

```
$ curl -L -k https://localhost:8080/catalogs/redhat-operators/all.json \
  -C - -o /home/username/catalogs/rhoc.json
```

3. 다음 명령 중 하나를 실행하여 카탈로그의 Operator 및 확장 목록을 반환합니다.



중요

현재 **OLM(Operator Lifecycle Manager) v1**에서는 다음 기준을 모두 충족하는 클러스터 확장 설치를 지원합니다.

- 확장 기능은 기존 **OLM**에 도입된 **registry+v1** 번들 형식을 사용해야 합니다.
- 확장 기능은 **AllNamespaces** 설치 모드를 통한 설치를 지원해야 합니다.
- 확장에서는 **Webhook**를 사용하지 않아야 합니다.
- 확장자는 다음 파일 기반 카탈로그 속성을 사용하여 종속성을 선언해서는 안 됩니다.
  - **olm.gvk.required**
  - **olm.package.required**
  - **olm.constraint**

OLM v1은 설치하려는 확장이 이러한 제약 조건을 충족하는지 확인합니다. 설치하려는 확장이 이러한 제약 조건을 충족하지 않으면 클러스터 확장 상태에 오류 메시지가 출력됩니다.

- 다음 명령을 실행하여 로컬 카탈로그 파일에서 모든 **Operator** 및 확장 프로그램 목록을 가져옵니다.

```
$ jq -s '.[] | select(.schema == "olm.package") | .name' \
  /<path>/<filename>.json
```

예 5.2. 명령 예

```
$ jq -s '.[] | select(.schema == "olm.package") | .name' \
  /home/username/catalogs/rhoc.json
```

```
-
```

예 5.3. 출력 예

```
NAME                                      AGE
"3scale-operator"
"advanced-cluster-management"
"amq-broker-rhel8"
"amq-online"
"amq-streams"
"amq7-interconnect-operator"
"ansible-automation-platform-operator"
"ansible-cloud-addons-operator"
"apicast-operator"
"aws-efs-csi-driver-operator"
"aws-load-balancer-operator"
"bamoe-businessautomation-operator"
"bamoe-kogito-operator"
"bare-metal-event-relay"
"businessautomation-operator"
...
```

- **AllNamespaces** 설치 모드를 지원하는 패키지 목록을 가져오고 다음 명령을 실행하여 로컬 카탈로그 파일의 **Webhook**를 사용하지 않습니다.

```
$ jq -c 'select(.schema == "olm.bundle") | \
  {"package":.package, "version":.properties[] | \
  select(.type == "olm.bundle.object").value.data | @base64d | fromjson | \
  select(.kind == "ClusterServiceVersion" and (.spec.installModes[] | \
  select(.type == "AllNamespaces" and .supported == true) != null) \
  and .spec.webhookdefinitions == null).spec.version}' \
  /<path>/<catalog_name>.json
```

예 5.4. 출력 예

```
{"package":"3scale-operator","version":"0.10.0-mas"}
{"package":"3scale-operator","version":"0.10.5"}
{"package":"3scale-operator","version":"0.11.0-mas"}
{"package":"3scale-operator","version":"0.11.1-mas"}
{"package":"3scale-operator","version":"0.11.2-mas"}
{"package":"3scale-operator","version":"0.11.3-mas"}
{"package":"3scale-operator","version":"0.11.5-mas"}
{"package":"3scale-operator","version":"0.11.6-mas"}
{"package":"3scale-operator","version":"0.11.7-mas"}
{"package":"3scale-operator","version":"0.11.8-mas"}
{"package":"amq-broker-rhel8","version":"7.10.0-opr-1"}
{"package":"amq-broker-rhel8","version":"7.10.0-opr-2"}
{"package":"amq-broker-rhel8","version":"7.10.0-opr-3"}
{"package":"amq-broker-rhel8","version":"7.10.0-opr-4"}
{"package":"amq-broker-rhel8","version":"7.10.1-opr-1"}
{"package":"amq-broker-rhel8","version":"7.10.1-opr-2"}
```

```
{"package":"amq-broker-rhel8","version":"7.10.2-opr-1"}
{"package":"amq-broker-rhel8","version":"7.10.2-opr-2"}
...
```

4. 다음 명령을 실행하여 Operator 또는 확장의 메타데이터 콘텐츠를 검사합니다.

```
$ jq -s '.[] | select( .schema == "olm.package") | \
  select( .name == "<package_name>")' /<path>/<catalog_name>.json
```

예 5.5. 명령 예

```
$ jq -s '.[] | select( .schema == "olm.package") | \
  select( .name == "openshift-pipelines-operator-rh")' \
  /home/username/rhoc.json
```

예 5.6. 출력 예

```
{
  "defaultChannel": "stable",
  "icon": {
    "base64data": "PHN2ZyB4bWxu..."
    "mediatype": "image/png"
  },
  "name": "openshift-pipelines-operator-rh",
  "schema": "olm.package"
}
```

5.1.2.1. 공통 카탈로그 쿼리

jq CLI 툴을 사용하여 카탈로그를 쿼리할 수 있습니다.

표 5.1. 일반적인 패키지 쿼리

쿼리	요청
카탈로그에서 사용 가능한 패키지	$ jq -s '.[] | select(.schema == "olm.package") | \ .name' <catalog_name>.json

쿼리	요청
AllNamespaces 설치 모드를 지원하고 Webhook를 사용하지 않는 패키지	$ jq -c 'select(.schema == "olm.bundle") | \ {"package":.package, "version":.properties[] | \ select(.type == "olm.bundle.object").value.data | \ @base64d | fromjson | \ select(.kind == "ClusterServiceVersion" and (.spec.installModes[] | \ select(.type == "AllNamespaces" and .supported == true) != null) \ and .spec.webhookdefinitions == null).spec.version}' \ <catalog_name>.json
패키지 메타데이터	$ jq -s '.[] | select(.schema == "olm.package") | \ select(.name == "<package_name>")' <catalog_name>.json
패키지의 카탈로그 Blob	$ jq -s '.[] | select(.package == "<package_name>")' \ <catalog_name>.json

표 5.2. 공통 채널 쿼리

쿼리	요청
패키지의 채널	$ jq -s '.[] | select(.schema == "olm.channel") | \ select(.package == "<package_name>") | .name' \ <catalog_name>.json
채널의 버전	$ jq -s '.[] | select(.package == "<package_name>") | \ select(.schema == "olm.channel") | \ select(.name == "<channel_name>") | \ .entries | .[] | .name' <catalog_name>.json
• 채널의 최신 버전 • 업그레이드 경로	$ jq -s '.[] | select(.schema == "olm.channel") | \ select (.name == "<channel>") | \ select(.package == "<package_name>")' \ <catalog_name>.json

표 5.3. 공통 번들 쿼리

쿼리	요청

쿼리	요청

패키지의 번들	`$ jq -s '.[] | select( .schema == "olm.bundle" ) | \`
	`select( .package == "<package_name>") | .name' \`
	`<catalog_name>.json`

• 번들 종속 항목
• 사용 가능한 API
	`$ jq -s '.[] | select( .schema == "olm.bundle" ) | \`
	`select ( .name == "<bundle_name>") | \`
	`select( .package == "<package_name>")' \`
	`<catalog_name>.json`

5.1.3. 클러스터 확장을 관리하는 서비스 계정 생성

기존 **OLM(Operator Lifecycle Manager)**과 달리 **OLM v1**에는 클러스터 확장 기능을 설치, 업데이트 및 관리할 수 있는 권한이 없습니다. 클러스터 관리자는 서비스 계정을 생성하고 클러스터 확장을 설치, 업데이트 및 관리하는 데 필요한 **RBAC(**역할 기반 액세스 제어**)**를 할당해야 합니다.

중요

OLM v1에는 알려진 문제가 있습니다. 확장의 서비스 계정에 올바른 역할 기반 액세스 제어**(RBAC)**를 할당하지 않으면 **OLM v1**이 중단되고 조정이 중지됩니다.

현재 **OLM v1**에는 확장 관리자가 서비스 계정에 대한 올바른 **RBAC**를 찾는 데 도움이 되는 도구가 없습니다.

OLM v1은 기술 프리뷰 기능이므로 프로덕션 클러스터에서 사용해서는 안되므로 문서에 포함된 보다 허용적인 **RBAC**를 사용하여 이 문제를 방지할 수 있습니다.

이 **RBAC**는 테스트 목적으로만 사용됩니다. 프로덕션 클러스터에서는 사용하지 마십시오.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.

프로세스

1. 다음 예와 유사한 서비스 계정을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <extension>-installer
  namespace: <namespace>
```

예 5.7. extension-service-account.yaml 파일의 예

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-installer
  namespace: pipelines
```

2. 다음 명령을 실행하여 서비스 계정을 적용합니다.

```
$ oc apply -f extension-service-account.yaml
```

3. 다음 예와 유사하게 클러스터 역할을 생성하고 RBAC를 할당합니다.

주의

다음 클러스터 역할은 최소 권한 원칙을 따르지 않습니다. 이 클러스터 역할은 테스트 목적으로만 사용됩니다. 프로덕션 클러스터에서는 사용하지 마십시오.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <extension>-installer-clusterrole
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

예 5.8. pipelines-cluster-role.yaml 파일의 예

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-installer-clusterrole
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

4. 다음 명령을 실행하여 클러스터에 클러스터 역할을 추가합니다.

```
$ oc apply -f pipelines-role.yaml
```

5. 다음 예와 유사하게 클러스터 역할 바인딩을 생성하여 클러스터 역할에 의해 부여된 권한을 서비스 계정에 바인딩합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: <extension>-installer-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: <extension>-installer-clusterrole
subjects:
- kind: ServiceAccount
  name: <extension>-installer
  namespace: <namespace>
```

예 5.9. pipelines-cluster-role-binding.yaml 파일의 예

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pipelines-installer-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-installer-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-installer
  namespace: pipelines
```

6. 다음 명령을 실행하여 클러스터 역할 바인딩을 적용합니다.

```
$ oc apply -f pipelines-cluster-role-binding.yaml
```

5.1.4. 카탈로그에서 클러스터 확장 설치

CR(사용자 정의 리소스)을 생성하고 클러스터에 적용하여 카탈로그에서 확장을 설치할 수 있습니다. OLM(Operator Lifecycle Manager) v1은 클러스터 범위인 registry+v1 번들 형식을 통해 기존 OLM Operator를 포함한 클러스터 확장 설치를 지원합니다. 자세한 내용은 *지원되는 확장 기능을* 참조하십시오.

중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

사전 요구 사항

- 클러스터에 카탈로그를 추가했습니다.
- 카탈로그 파일의 로컬 사본을 다운로드했습니다.
- jq CLI 툴을 설치했습니다.
- 서비스 계정을 생성하고 설치할 확장 기능을 설치, 업데이트 및 관리할 충분한 RBAC(역할 기반 액세스 제어)가 할당되었습니다. 자세한 내용은 *서비스 계정 생성을* 참조하십시오.

프로세스

1. 다음 단계를 완료하여 카탈로그 파일의 로컬 사본에서 채널 및 버전 정보에 대한 패키지를 검사합니다.
   a. 다음 명령을 실행하여 선택한 패키지에서 채널 목록을 가져옵니다.

```
$ jq -s '.[] | select( .schema == "olm.channel" ) | \
  select( .package == "<package_name>") | \
  .name' /<path>/<catalog_name>.json
```

예 5.10. 명령 예

```
$ jq -s '.[] | select( .schema == "olm.channel" ) | \
  select( .package == "openshift-pipelines-operator-rh") | \
  .name' /home/username/rhoc.json
```

예 5.11. 출력 예

```
"latest"
"pipelines-1.11"
"pipelines-1.12"
"pipelines-1.13"
"pipelines-1.14"
```

b.

다음 명령을 실행하여 채널에 게시된 버전 목록을 가져옵니다.

```
$ jq -s '.[] | select( .package == "<package_name>" ) | \
  select( .schema == "olm.channel" ) | \
  select( .name == "<channel_name>" ) | .entries | \
  .[] | .name' /<path>/<catalog_name>.json
```

예 5.12. 명령 예

```
$ jq -s '.[] | select( .package == "openshift-pipelines-operator-rh" ) | \
select( .schema == "olm.channel" ) | select( .name == "latest" ) | \
.entries | .[] | .name' /home/username/rhoc.json
```

예 5.13. 출력 예

```
"openshift-pipelines-operator-rh.v1.12.0"
"openshift-pipelines-operator-rh.v1.12.1"
"openshift-pipelines-operator-rh.v1.12.2"
"openshift-pipelines-operator-rh.v1.13.0"
"openshift-pipelines-operator-rh.v1.13.1"
"openshift-pipelines-operator-rh.v1.11.1"
"openshift-pipelines-operator-rh.v1.12.0"
"openshift-pipelines-operator-rh.v1.12.1"
"openshift-pipelines-operator-rh.v1.12.2"
"openshift-pipelines-operator-rh.v1.13.0"
"openshift-pipelines-operator-rh.v1.14.1"
"openshift-pipelines-operator-rh.v1.14.2"
"openshift-pipelines-operator-rh.v1.14.3"
"openshift-pipelines-operator-rh.v1.14.4"
```

2. 새 네임스페이스에 확장을 설치하려면 다음 명령을 실행합니다.

```
$ oc adm new-project <new_namespace>
```

3. 다음 예와 유사한 CR을 생성합니다.

pipelines-operator.yaml CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace>
  serviceAccount:
    name: <service_account>
  channel: <channel>
  version: "<version>"
```

다음과 같습니다.

<namespace>

파이프라인 또는 my-extension 와 같이 번들을 설치할 네임스페이스를 지정합니다. 확장 기능은 여전히 클러스터 범위이며 다른 네임스페이스에 설치된 리소스가 포함될 수 있습니다.

<service_account>

확장 기능을 설치, 업데이트 및 관리하기 위해 만든 서비스 계정의 이름을 지정합니다.

<channel>

선택 사항: 설치 또는 업데이트하려는 패키지에 대해 pipelines-1.11 또는 latest 와 같은 채널을 지정합니다.

<version>

선택 사항: 설치 또는 업데이트하려는 패키지의 1.11.1,1.12.x 또는 > =1.12.1 과 같은 버

전 범위를 지정합니다. 자세한 내용은 "대상 버전을 지정하는 CR(사용자 정의 리소스) 예" 및 "버전 범위에 대한 지원"을 참조하십시오.

중요

고유한 이름이 없는 Operator 또는 확장을 설치하려고 하면 설치에 실패하거나 예기치 않은 결과가 발생할 수 있습니다. 이는 다음과 같은 이유로 발생합니다.

- mulitple 카탈로그가 클러스터에 설치된 경우 OLM(Operator Lifecycle Manager) v1에는 Operator 또는 확장을 설치할 때 카탈로그를 지정하는 메커니즘이 포함되지 않습니다.
- OLM v1에서는 클러스터에 설치할 수 있는 모든 Operator 및 확장 이 번들 및 패키지에 고유한 이름을 사용해야 합니다.

4. 다음 명령을 실행하여 클러스터에 CR을 적용합니다.

```
$ oc apply -f pipeline-operator.yaml
```

출력 예

```
clusterextension.olm.operatorframework.io/pipelines-operator created
```

검증

1. 다음 명령을 실행하여 Operator 또는 확장의 CR을 YAML 형식으로 표시합니다.

```
$ oc get clusterextension pipelines-operator -o yaml
```

예 5.14. 출력 예

```
apiVersion: v1
items:
- apiVersion: olm.operatorframework.io/v1alpha1
```

```
  kind: ClusterExtension
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"olm.operatorframework.io/v1alpha1","kind":"ClusterExtension","me
tadata":{"annotations":{},"name":"pipelines-operator"},"spec":
{"channel":"latest","installNamespace":"pipelines","packageName":"openshift-
pipelines-operator-rh","serviceAccount":{"name":"pipelines-
installer"},"pollInterval":"30m"}}
    creationTimestamp: "2024-06-10T17:50:51Z"
    finalizers:
    - olm.operatorframework.io/cleanup-unpack-cache
    generation: 1
    name: pipelines-operator
    resourceVersion: "53324"
    uid: c54237be-cde4-46d4-9b31-d0ec6acc19bf
  spec:
    channel: latest
    installNamespace: pipelines
    packageName: openshift-pipelines-operator-rh
    serviceAccount:
      name: pipelines-installer
    upgradeConstraintPolicy: Enforce
  status:
    conditions:
    - lastTransitionTime: "2024-06-10T17:50:58Z"
      message: resolved to "registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:dd3d18367da2be42539e5dde8e484dac3df33ba3ce1d5bcf896838954f3864ec"
      observedGeneration: 1
      reason: Success
      status: "True"
      type: Resolved
    - lastTransitionTime: "2024-06-10T17:51:11Z"
      message: installed from "registry.redhat.io/openshift-pipelines/pipelines-
operator-
bundle@sha256:dd3d18367da2be42539e5dde8e484dac3df33ba3ce1d5bcf896838954f3864ec"
      observedGeneration: 1
      reason: Success
      status: "True"
      type: Installed
    - lastTransitionTime: "2024-06-10T17:50:58Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: Deprecated
    - lastTransitionTime: "2024-06-10T17:50:58Z"
      message: ""
      observedGeneration: 1
      reason: Deprecated
      status: "False"
      type: PackageDeprecated
    - lastTransitionTime: "2024-06-10T17:50:58Z"
```

```
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: ChannelDeprecated
  - lastTransitionTime: "2024-06-10T17:50:58Z"
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: BundleDeprecated
  - lastTransitionTime: "2024-06-10T17:50:58Z"
    message: 'unpack successful:
    observedGeneration: 1
    reason: UnpackSuccess
    status: "True"
    type: Unpacked
  installedBundle:
    name: openshift-pipelines-operator-rh.v1.14.4
    version: 1.14.4
  resolvedBundle:
    name: openshift-pipelines-operator-rh.v1.14.4
    version: 1.14.4
```

다음과 같습니다.

spec.channel

확장의 CR에 정의된 채널을 표시합니다.

spec.version

확장의 CR에 정의된 버전 또는 버전 범위를 표시합니다.

status.conditions

확장의 상태 및 상태에 대한 정보를 표시합니다.

유형: 더 이상 사용되지 않음

다음 중 하나 이상이 더 이상 사용되지 않는지 여부를 표시합니다.

type: PackageDeprecated

해결된 패키지가 더 이상 사용되지 않는지 여부를 표시합니다.

type: ChannelDeprecated

해결된 채널이 더 이상 사용되지 않는지 여부를 표시합니다.

유형: BundleDeprecated

해결된 번들이 더 이상 사용되지 않는지 여부를 표시합니다.

status 필드의 False 값은 reason: 더 이상 사용되지 않는 조건이 더 이상 사용되지 않음을 나타냅니다. status 필드의 True 값은 reason: 더 이상 사용되지 않는 조건이 더 이상 사용되지 않음을 나타냅니다.

installedBundle.name

설치된 번들의 이름을 표시합니다.

installedBundle.version

설치된 번들의 버전을 표시합니다.

resolvedBundle.name

확인된 번들의 이름을 표시합니다.

resolvedBundle.version

해결된 번들의 버전을 표시합니다.

추가 리소스

- 지원되는 확장
- 서비스 계정 생성
- 대상 버전을 지정하는 CR(사용자 정의 리소스)의 예
- 버전 범위 지원

5.1.5. 클러스터 확장 업데이트

CR(사용자 정의 리소스)을 수동으로 편집하고 변경 사항을 적용하여 클러스터 확장 또는 Operator를 업데이트할 수 있습니다.

사전 요구 사항

- 카탈로그가 설치되어 있어야 합니다.
- 카탈로그 파일의 로컬 사본을 다운로드했습니다.
- Operator 또는 확장이 설치되어 있어야 합니다.
- jq CLI 툴을 설치했습니다.

프로세스

1. 다음 단계를 완료하여 카탈로그 파일의 로컬 사본에서 채널 및 버전 정보에 대한 패키지를 검사합니다.

   a. 다음 명령을 실행하여 선택한 패키지에서 채널 목록을 가져옵니다.

```
$ jq -s '.[] | select( .schema == "olm.channel" ) | \
  select( .package == "<package_name>") | \
  .name' /<path>/<catalog_name>.json
```

   예 5.15. 명령 예

```
$ jq -s '.[] | select( .schema == "olm.channel" ) | \
  select( .package == "openshift-pipelines-operator-rh") | \
  .name' /home/username/rhoc.json
```

   예 5.16. 출력 예

```
"latest"
"pipelines-1.11"
"pipelines-1.12"
"pipelines-1.13"
"pipelines-1.14"
```

   b. 다음 명령을 실행하여 채널에 게시된 버전 목록을 가져옵니다.

```
$ jq -s '.[] | select( .package == "<package_name>" ) | \
  select( .schema == "olm.channel" ) | \
  select( .name == "<channel_name>" ) | .entries | \
  .[] | .name' /<path>/<catalog_name>.json
```

예 5.17. 명령 예

```
$ jq -s '.[] | select( .package == "openshift-pipelines-operator-rh" ) | \
select( .schema == "olm.channel" ) | select( .name == "latest" ) | \
.entries | .[] | .name' /home/username/rhoc.json
```

예 5.18. 출력 예

```
"openshift-pipelines-operator-rh.v1.11.1"
"openshift-pipelines-operator-rh.v1.12.0"
"openshift-pipelines-operator-rh.v1.12.1"
"openshift-pipelines-operator-rh.v1.12.2"
"openshift-pipelines-operator-rh.v1.13.0"
"openshift-pipelines-operator-rh.v1.14.1"
"openshift-pipelines-operator-rh.v1.14.2"
"openshift-pipelines-operator-rh.v1.14.3"
"openshift-pipelines-operator-rh.v1.14.4"
```

2. 다음 명령을 실행하여 **Operator** 또는 확장의 **CR**에 지정된 버전 또는 채널을 확인합니다.

```
$ oc get clusterextension <operator_name> -o yaml
```

명령 예

```
$ oc get clusterextension pipelines-operator -o yaml
```

예 5.19. 출력 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"olm.operatorframework.io/v1alpha1","kind":"ClusterExtension","me
tadata":{"annotations":{},"name":"pipelines-operator"},"spec":
{"channel":"latest","installNamespace":"openshift-
```

```
operators","packageName":"openshift-pipelines-operator-
rh","pollInterval":"30m","version":"\u003c1.12"}}
  creationTimestamp: "2024-06-11T15:55:37Z"
  generation: 1
  name: pipelines-operator
  resourceVersion: "69776"
  uid: 6a11dff3-bfa3-42b8-9e5f-d8babbd6486f
spec:
  channel: latest
  installNamespace: openshift-operators
  packageName: openshift-pipelines-operator-rh
  upgradeConstraintPolicy: Enforce
  version: <1.12
status:
  conditions:
  - lastTransitionTime: "2024-06-11T15:56:09Z"
    message: installed from "registry.redhat.io/openshift-pipelines/pipelines-
operator-
bundle@sha256:e09d37bb1e754db42324fd18c1cb3e7ce77e7b7fcbf4932d053539157
9938280"
    observedGeneration: 1
    reason: Success
    status: "True"
    type: Installed
  - lastTransitionTime: "2024-06-11T15:55:50Z"
    message: resolved to "registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:e09d37bb1e754db42324fd18c1cb3e7ce77e7b7fcbf4932d053539157
9938280"
    observedGeneration: 1
    reason: Success
    status: "True"
    type: Resolved
  - lastTransitionTime: "2024-06-11T15:55:50Z"
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: Deprecated
  - lastTransitionTime: "2024-06-11T15:55:50Z"
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: PackageDeprecated
  - lastTransitionTime: "2024-06-11T15:55:50Z"
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: ChannelDeprecated
  - lastTransitionTime: "2024-06-11T15:55:50Z"
    message: ""
    observedGeneration: 1
    reason: Deprecated
    status: "False"
    type: BundleDeprecated
```

```
  installedBundle:
    name: openshift-pipelines-operator-rh.v1.11.1
    version: 1.11.1
  resolvedBundle:
    name: openshift-pipelines-operator-rh.v1.11.1
    version: 1.11.1
```

3. 다음 방법 중 하나를 사용하여 **CR**을 편집합니다.

- **Operator** 또는 확장을 **1.12.1** 과 같은 특정 버전으로 고정하려면 다음 예와 유사한 **CR**을 편집합니다.

pipelines-operator.yaml CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace>
  version: "1.12.1" (1)
```

(1) 버전 **1.11.1** 에서 **1.12.1**로 업데이트

- 허용 가능한 업데이트 버전의 범위를 정의하려면 다음 예와 유사한 **CR**을 편집합니다.

버전 범위가 지정된 **CR**의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
```

```
packageName: openshift-pipelines-operator-rh
installNamespace: <namespace>
version: ">1.11.1, <1.13" 1
```

1

원하는 버전 범위가 버전 **1.11.1** 보다 크고 **1.13** 보다 작도록 지정합니다. 자세한 내용은 "버전 범위 지원" 및 "버전 비교 문자열"을 참조하십시오.

- 채널에서 확인할 수 있는 최신 버전으로 업데이트하려면 다음 예제와 유사한 **CR**을 편집합니다.

지정된 채널이 있는 **CR**의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace>
  channel: pipelines-1.13 1
```

1

지정된 채널에서 확인할 수 있는 최신 릴리스를 설치합니다. 채널 업데이트가 자동으로 설치됩니다.

- 채널 및 버전 범위를 지정하려면 다음 예와 유사한 **CR**을 편집합니다.

지정된 채널 및 버전 범위가 있는 **CR**의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
```

```
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace>
  channel: latest
  version: "<1.13"
```

자세한 내용은 "대상 버전을 지정하는 CR(사용자 정의 리소스) 예"를 참조하십시오.

4. 다음 명령을 실행하여 클러스터에 업데이트를 적용합니다.

```
$ oc apply -f pipelines-operator.yaml
```

출력 예

```
clusterextension.olm.operatorframework.io/pipelines-operator configured
```

작은 정보

다음 명령을 실행하여 CLI에서 CR을 패치하고 적용할 수 있습니다.

```
$ oc patch clusterextension/pipelines-operator -p \
  '{"spec":{"version":"<1.13"}}' \
  --type=merge
```

출력 예

```
clusterextension.olm.operatorframework.io/pipelines-operator patched
```

검증

- 다음 명령을 실행하여 채널 및 버전 업데이트가 적용되었는지 확인합니다.

```
$ oc get clusterextension pipelines-operator -o yaml
```

예 5.20. 출력 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"olm.operatorframework.io/v1alpha1","kind":"ClusterExtension","me
tadata":{"annotations":{},"name":"pipelines-operator"},"spec":
{"channel":"latest","installNamespace":"openshift-
operators","packageName":"openshift-pipelines-operator-
rh","pollInterval":"30m","version":"\u003c1.13"}}
  creationTimestamp: "2024-06-11T18:23:26Z"
  generation: 2
  name: pipelines-operator
  resourceVersion: "66310"
  uid: ce0416ba-13ea-4069-a6c8-e5efcbc47537
spec:
  channel: latest
  installNamespace: openshift-operators
  packageName: openshift-pipelines-operator-rh
  upgradeConstraintPolicy: Enforce
  version: <1.13
status:
  conditions:
  - lastTransitionTime: "2024-06-11T18:23:33Z"
    message: resolved to "registry.redhat.io/openshift-pipelines/pipelines-operator-
bundle@sha256:814742c8a7cc7e2662598e114c35c13993a7b423cfe92548124e43ea5d
469f82"
    observedGeneration: 2
    reason: Success
    status: "True"
    type: Resolved
  - lastTransitionTime: "2024-06-11T18:23:52Z"
    message: installed from "registry.redhat.io/openshift-pipelines/pipelines-
operator-
bundle@sha256:814742c8a7cc7e2662598e114c35c13993a7b423cfe92548124e43ea5d
469f82"
    observedGeneration: 2
    reason: Success
    status: "True"
    type: Installed
  - lastTransitionTime: "2024-06-11T18:23:33Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
```

```
    type: Deprecated
  - lastTransitionTime: "2024-06-11T18:23:33Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: PackageDeprecated
  - lastTransitionTime: "2024-06-11T18:23:33Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: ChannelDeprecated
  - lastTransitionTime: "2024-06-11T18:23:33Z"
    message: ""
    observedGeneration: 2
    reason: Deprecated
    status: "False"
    type: BundleDeprecated
  installedBundle:
    name: openshift-pipelines-operator-rh.v1.12.2
    version: 1.12.2
  resolvedBundle:
    name: openshift-pipelines-operator-rh.v1.12.2
    version: 1.12.2
```

문제 해결

- 더 이상 사용되지 않거나 존재하지 않는 대상 버전 또는 채널을 지정하는 경우 다음 명령을 실행하여 확장 상태를 확인할 수 있습니다.

```
$ oc get clusterextension <operator_name> -o yaml
```

예 5.21. 존재하지 않는 버전의 출력 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"olm.operatorframework.io/v1alpha1","kind":"ClusterExtension","me
tadata":{"annotations":{},"name":"pipelines-operator"},"spec":
{"channel":"latest","installNamespace":"openshift-
operators","packageName":"openshift-pipelines-operator-
rh","pollInterval":"30m","version":"3.0"}}
  creationTimestamp: "2024-06-11T18:23:26Z"
  generation: 3
  name: pipelines-operator
  resourceVersion: "71852"
  uid: ce0416ba-13ea-4069-a6c8-e5efcbc47537
spec:
```

```
  channel: latest
  installNamespace: openshift-operators
  packageName: openshift-pipelines-operator-rh
  upgradeConstraintPolicy: Enforce
  version: "3.0"
status:
  conditions:
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: 'error upgrading from currently installed version "1.12.2": no package
      "openshift-pipelines-operator-rh" matching version "3.0" found in channel
"latest"'
    observedGeneration: 3
    reason: ResolutionFailed
    status: "False"
    type: Resolved
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: installation has not been attempted as resolution failed
    observedGeneration: 3
    reason: InstallationStatusUnknown
    status: Unknown
    type: Installed
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: deprecation checks have not been attempted as resolution failed
    observedGeneration: 3
    reason: Deprecated
    status: Unknown
    type: Deprecated
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: deprecation checks have not been attempted as resolution failed
    observedGeneration: 3
    reason: Deprecated
    status: Unknown
    type: PackageDeprecated
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: deprecation checks have not been attempted as resolution failed
    observedGeneration: 3
    reason: Deprecated
    status: Unknown
    type: ChannelDeprecated
  - lastTransitionTime: "2024-06-11T18:29:02Z"
    message: deprecation checks have not been attempted as resolution failed
    observedGeneration: 3
    reason: Deprecated
    status: Unknown
    type: BundleDeprecated
```

추가 리소스

- 업그레이드 엣지

5.1.6. Operator 삭제

ClusterExtension CR(사용자 정의 리소스)을 삭제하여 Operator 및 해당 CRD(사용자 정의 리소스 정의)를 삭제할 수 있습니다.

사전 요구 사항

- 카탈로그가 설치되어 있어야 합니다.
- Operator가 설치되어 있어야 합니다.

프로세스

- 다음 명령을 실행하여 Operator 및 해당 CRD를 삭제합니다.

```
$ oc delete clusterextension <operator_name>
```

출력 예

```
clusterextension.olm.operatorframework.io "<operator_name>" deleted
```

검증

- 다음 명령을 실행하여 Operator 및 해당 리소스가 삭제되었는지 확인합니다.
  - 다음 명령을 실행하여 Operator가 삭제되었는지 확인합니다.

```
$ oc get clusterextensions
```

출력 예

```
No resources found
```

- 다음 명령을 실행하여 Operator의 시스템 네임스페이스가 삭제되었는지 확인합니다.

```
$ oc get ns <operator_name>-system
```

출력 예

```
Error from server (NotFound): namespaces "<operator_name>-system" not found
```

5.2. 업그레이드 엣지

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

설치된 클러스터 확장에 대한 업그레이드 경로 또는 업그레이드 제약 조건이라고도 하는 업그레이드 에지를 결정할 때 OLM(Operator Lifecycle Manager) v1은 OpenShift Container Platform 4.16에서 시작하는 기존 OLM 의미 체계를 지원합니다. 이 지원은 몇 가지 차이점이 있지만 대체,건너뛰기, skipRange 지시문을 포함하여 기존 OLM의 동작을 따릅니다.

OLM v1은 기존 OLM 의미 체계를 지원하여 카탈로그의 업그레이드 그래프를 정확하게 준수합니다.

중요

현재 OLM(Operator Lifecycle Manager) v1은 Red Hat 제공 Operator 카탈로그와 같은 프라이빗 레지스트리를 인증할 수 없습니다. 이것은 확인된 문제입니다. 결과적으로 Red Hat Operator 카탈로그를 설치하는 데 사용하는 OLM v1 절차가 작동하지 않습니다. (OCPBUGS-36364)

기존 기존 OLM 구현의 차이점

- 여러 성공자가 있는 경우 OLM v1 동작은 다음과 같은 방식으로 다릅니다.
  - 기존 OLM에서는 채널 헤드에 가장 가까운 후속 항목이 선택됩니다.
  - OLM v1에서 의미 체계(semver)가 가장 높은 후속 버전이 선택됩니다.
- 다음 파일 기반 카탈로그(FBC) 채널 항목을 고려하십시오.

```
# ...
- name: example.v3.0.0
  skips: ["example.v2.0.0"]
- name: example.v2.0.0
  skipRange: >=1.0.0 <2.0.0
```

1.0.0 이 설치된 경우 OLM v1 동작은 다음과 같은 방식으로 다릅니다.

  - v2.0.0 을 건너뛰고 대체 체인에 있지 않기 때문에 기존 OLM은 v2.0.0 으로의 업그레이드 에지를 감지하지 않습니다.
  - OLM v1에는 대체 체인의 개념이 없으므로 OLM v1은 업그레이드 에지를 감지합니다. OLM v1은 현재 설치된 버전을 처리하는 교체, 건너뛰기 또는skip Range 값이 있는 모든 항목을 찾습니다.

추가 리소스

- 기존 OLM 업그레이드 시맨틱

5.2.1. 버전 범위 지원

OLM(Operator Lifecycle Manager) v1에서는 Operator 또는 확장의 CR(사용자 정의 리소스)에서 비교 문자열을 사용하여 버전 범위를 지정할 수 있습니다. CR에 버전 범위를 지정하면 OLM v1이 버전 범위 내에서 해결할 수 있는 최신 버전의 Operator를 설치하거나 업데이트합니다.

해결된 버전 워크플로

- 해결된 버전은 Operator 및 환경의 제약 조건을 충족하는 최신 버전의 Operator입니다.
- 지정된 범위 내의 Operator 업데이트가 성공적으로 확인되면 자동으로 설치됩니다.
- 지정된 범위를 벗어나거나 성공적으로 해결할 수 없는 경우 업데이트가 설치되지 않습니다.

5.2.2. 버전 비교 문자열

Operator 또는 확장의 CR(사용자 정의 리소스)의 spec.version 필드에 비교 문자열을 추가하여 버전 범위를 정의할 수 있습니다. 비교 문자열은 공백 또는 쉼표로 구분된 값 목록과 큰따옴표(")로 묶인 하나 이상의 비교 연산자입니다. OR 또는 double vertical bar (||) 비교 연산자를 포함 하 여 다른 비교 문자열을 추가할 수 있습니다.You can add another comparison string by including an OR, or double vertical bar (||), comparison operator between the strings.

표 5.4. 기본 비교

비교 연산자	정의
=	동일
!=	같지 않음
>	보다 큼
<	보다 작음
>=	크거나 같음
<=	작거나 같음

다음 예와 유사한 범위 비교를 사용하여 Operator 또는 확장의 CR에서 버전 범위를 지정할 수 있습니다.

버전 범위 비교 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  version: ">=1.11, <1.13"
```

모든 유형의 비교 문자열에 와일드카드 문자를 사용할 수 있습니다. **OLM v1**에서는 **x,X** 및 별표(*)를 와일드카드 문자로 사용할 수 있습니다. 등호(=) 비교 연산자와 함께 와일드카드 문자를 사용하는 경우 패치 또는 마이너 버전 수준에서 비교를 정의합니다.

표 5.5. 비교 문자열에 있는 와일드카드 문자의 예

와일드카드 비교	일치하는 문자열
1.11.x	>=1.11.0, <1.12.0
>=1.12.X	>=1.12.0
<=2.x	<3
*	>=0.0.0

틸드(~) 비교 연산자를 사용하여 패치 릴리스 비교를 수행할 수 있습니다. 패치 릴리스 비교에서는 다음 주요 버전까지 마이너 버전을 지정합니다.

표 5.6. 패치 릴리스 비교의 예

패치 릴리스 비교	일치하는 문자열
~1.11.0	>=1.11.0, <1.12.0
~1	>=1, <2
~1.12	>=1.12, <1.13
~1.12.x	>=1.12.0, <1.13.0

패치 릴리스 비교	일치하는 문자열
~1.x	>=1, <2

caret(^) 비교 연산자를 사용하여 주요 릴리스를 비교할 수 있습니다. 첫 번째 안정적인 릴리스가 게시되기 전에 주요 릴리스 비교를 수행하는 경우 마이너 버전은 API의 안정성 수준을 정의합니다. 의미 체계 버전 관리 (semver) 사양에서 첫 번째 안정적인 릴리스는 1.0.0 버전으로 게시됩니다.

표 5.7. 주요 릴리스 비교의 예

주요 릴리스 비교	일치하는 문자열
^0	>=0.0.0, <1.0.0
^0.0	>=0.0.0, <0.1.0
^0.0.3	>=0.0.3, <0.0.4
^0.2	>=0.2.0, <0.3.0
^0.2.3	>=0.2.3, <0.3.0
^1.2.x	>= 1.2.0, < 2.0.0
^1.2.3	>= 1.2.3, < 2.0.0
^2.x	>= 2.0.0, < 3
^2.3	>= 2.3, < 3

5.2.3. 대상 버전을 지정하는 CR(사용자 정의 리소스)의 예

OLM(Operator Lifecycle Manager) v1에서 클러스터 관리자는 사용자 정의 리소스(CR)에서 Operator 또는 확장의 대상 버전을 선언적으로 설정할 수 있습니다.

다음 필드 중 하나를 지정하여 대상 버전을 정의할 수 있습니다.

- 채널

- 버전 번호
- 버전 범위

CR에 채널을 지정하면 OLM v1이 지정된 채널 내에서 해결할 수 있는 최신 버전의 Operator 또는 확장 버전을 설치합니다. 지정된 채널에 업데이트가 게시되면 OLM v1이 채널에서 확인할 수 있는 최신 릴리스로 자동으로 업데이트됩니다.

지정된 채널이 있는 CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  channel: latest (1)
```

(1) 지정된 채널에서 확인할 수 있는 최신 릴리스를 설치합니다. 채널 업데이트가 자동으로 설치됩니다.

CR에서 Operator 또는 확장의 대상 버전을 지정하면 OLM v1이 지정된 버전을 설치합니다. 대상 버전이 CR에 지정되면 업데이트가 카탈로그에 게시될 때 OLM v1에서 대상 버전이 변경되지 않습니다.

클러스터에 설치된 Operator 버전을 업데이트하려면 Operator의 CR을 수동으로 편집해야 합니다. Operator의 대상 버전을 지정하면 Operator 버전이 지정된 릴리스에 고정됩니다.

대상 버전이 지정된 CR의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
```

```
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  version: "1.11.1" 1
```

1

대상 버전을 지정합니다. 설치된 **Operator** 또는 확장 버전을 업데이트하려면 **CR**을 원하는 대상 버전으로 수동으로 업데이트해야 합니다.

Operator 또는 확장에 허용되는 다양한 버전을 정의하려면 비교 문자열을 사용하여 버전 범위를 지정할 수 있습니다. 버전 범위를 지정하면 **OLM v1**은 **Operator** 컨트롤러에서 해결할 수 있는 최신 버전의 **Operator** 또는 확장을 설치합니다.

버전 범위가 지정된 **CR**의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: pipelines-operator
spec:
  packageName: openshift-pipelines-operator-rh
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  version: ">1.11.1" 1
```

1

원하는 버전 범위가 버전 **1.11.1** 보다 크도록 지정합니다. 자세한 내용은 "버전 범위 지원"을 참조하십시오.

CR을 생성하거나 업데이트한 후 다음 명령을 실행하여 구성 파일을 적용합니다.

명령 구문

```
$ oc apply -f <extension_name>.yaml
```

5.2.4. 업데이트 또는 롤백 강제 적용

OLM v1은 다음 주요 버전 또는 롤백에 대한 자동 업데이트를 지원하지 않습니다. 주요 버전 업데이트 또는 롤백을 수행하려면 수동으로 업데이트를 확인하고 강제 적용해야 합니다.

주의

수동 업데이트 또는 롤백을 강제 적용하는 결과를 확인해야 합니다. 강제 업데이트 또는 롤백을 확인하지 않으면 데이터 손실과 같은 심각한 결과가 발생할 수 있습니다.

사전 요구 사항

- 카탈로그가 설치되어 있어야 합니다.
- Operator 또는 확장이 설치되어 있어야 합니다.
- 서비스 계정을 생성하고 설치할 확장 기능을 설치, 업데이트 및 관리할 충분한 RBAC(역할 기반 액세스 제어)가 할당되었습니다. 자세한 내용은 *서비스 계정 생성*을 참조하십시오.

프로세스

1. 다음 예와 같이 Operator 또는 확장의 CR(사용자 정의 리소스)을 편집합니다.

CR 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: Operator
metadata:
  name: <operator_name> 1
spec:
  packageName: <package_name> 2
  installNamespace: <namespace_name>
  serviceAccount:
    name: <service_account>
  version: <version> 3
  upgradeConstraintPolicy: Ignore 4
```

1 Operator 또는 확장의 이름(예: pipelines-operator)을 지정합니다.

2 openshift-pipelines-operator-rh 와 같은 패키지 이름을 지정합니다.

3 차단된 업데이트 또는 롤백 버전을 지정합니다.

4 선택 사항: 업그레이드 제약 조건 정책을 지정합니다. 업데이트 또는 롤백을 강제 적용하려면 필드를 Ignore 로 설정합니다. 지정되지 않은 경우 기본 설정은 Enforce 입니다.

2. 다음 명령을 실행하여 Operator 또는 extensions CR에 변경 사항을 적용합니다.

```
$ oc apply -f <extension_name>.yaml
```

추가 리소스

- 버전 범위 지원

5.3. CRD(사용자 정의 리소스 정의) 업그레이드 안전성

중요

OLM(Operator Lifecycle Manager) v1은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

클러스터 확장에서 제공하는 **CRD**(사용자 정의 리소스 정의)를 업데이트하면 **OLM(Operator Lifecycle Manager) v1**은 **CRD** 업그레이드 안전 우선 순위 검사를 실행하여 이전 버전의 **CRD**와 이전 버전과의 호환성을 보장합니다. **CRD** 업데이트에서는 클러스터에서 변경 사항을 진행하기 전에 검증 검사를 전달해야 합니다.

추가 리소스

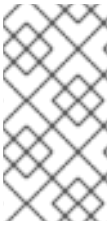
- 클러스터 확장 업데이트

5.3.1. CRD 업그레이드 금지

기존 **CRD**(사용자 정의 리소스 정의)에 대한 다음과 같은 변경 사항은 **CRD** 업그레이드 **safety preflight** 검사를 통해 발생하며 업그레이드를 방지합니다.

- 새 필수 필드가 기존 **CRD** 버전에 추가됩니다.
- 기존 필드가 기존 **CRD** 버전에서 제거됨
- 기존 필드 유형이 기존 **CRD** 버전에서 변경됨
- 이전에 기본값이 없는 필드에 새 기본값이 추가됩니다.

- 필드의 기본값이 변경됨
- 필드의 기존 기본값이 제거됨
- 이전에 **enum** 제한이 없는 기존 필드에 새로운 **enum** 제한 사항이 추가됨
- 기존 필드의 기존 **enum** 값이 제거됩니다.
- 기존 필드의 최소값이 기존 버전에서 증가했습니다.
- 기존 필드의 최대값이 기존 버전에서 감소합니다.
- 이전에 제약 조건이 없는 필드에 최소 또는 최대 필드 제약 조건이 추가됩니다.



참고

최소 및 최대값에 대한 변경 규칙은 최소 ,**minLength**,**minProperties**,**minItems**,최대,**maxLength**,**maxProperties** 및 **maxItems** 제약 조건에 적용됩니다.

기존 **CRD**에 대한 다음 변경 사항은 **CRD** 업그레이드 안전성 검사에서 보고하고 **Kubernetes API** 서버에서 작업을 기술적으로 처리하지만 업그레이드를 방지합니다.

- 범위가 클러스터에서 네임스페이스 로 또는 네임스페이스 에서 클러스터로 변경
- 저장된 **CRD**의 기존 버전이 제거됩니다.

CRD 업그레이드 **safety preflight** 검사에서 금지된 업그레이드 변경 중 하나가 발생하면 **CRD** 업그레이드에서 감지된 각 변경에 대한 오류를 기록합니다.

작은 정보

CRD에 대한 변경이 금지된 변경 카테고리 중 하나에 속하지 않지만 허용된 대로 적절하게 감지할 수 없는 경우 CRD 업그레이드 안전 전지 검사를 통해 업그레이드를 방지하고 "알 수 없는 변경"에 대한 오류를 기록합니다.

5.3.2. CRD 업그레이드 허용 변경

기존 CRD(사용자 정의 리소스 정의)에 대한 다음 변경 사항은 이전 버전과의 호환성을 위해 안전하지 않으며 CRD 업그레이드 안전 전선 검사로 인해 업그레이드가 중단되지 않습니다.

- 필드에 허용된 enum 값 목록에 새 enum 값 추가
- 기존 필수 필드가 기존 버전에서 선택 사항으로 변경되었습니다.
- 기존 버전의 기존 필드의 최소값이 감소합니다.
- 기존 필드의 최대값이 기존 버전에서 증가했습니다.
- 기존 버전에 대한 수정 없이 새 CRD 버전이 추가되었습니다.

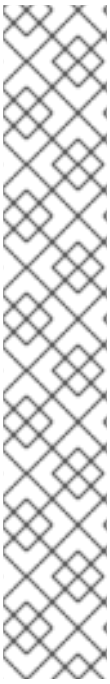
5.3.3. CRD 업그레이드 안전 전 검사 비활성화

CRD(사용자 정의 리소스 정의) 업그레이드 우선 순위 검사는 CRD를 제공하는 ClusterExtension 오브젝트에 값이 true 인 preflight.crdUpgradeSafety.disabled 필드를 추가하여 비활성화할 수 있습니다.

주의

CRD 업그레이드 안전 전지를 비활성화하면 저장된 CRD 버전과의 호환성이 중단되어 클러스터에 의도하지 않은 다른 결과가 발생할 수 있습니다.

개별 필드 검증기를 비활성화할 수 없습니다. CRD 업그레이드 safety preflight 검사를 비활성화하면 모든 필드 검증기가 비활성화됩니다.



참고

다음 검사는 Kubernetes API 서버에서 처리합니다.

- 범위가 클러스터에서 네임스페이스 로 또는 네임스페이스 에서 클러스터로 변경
- 저장된 CRD의 기존 버전이 제거됩니다.

OLM(Operator Lifecycle Manager) v1을 통해 CRD 업그레이드 안전 검사를 비활성화한 후에도 Kubernetes에서 이 두 작업을 계속 방지할 수 있습니다.

사전 요구 사항

- 클러스터 확장이 설치되어 있어야 합니다.

프로세스

1. CRD의 ClusterExtension 오브젝트를 편집합니다.

```
$ oc edit clusterextension <clusterextension_name>
```

2. preflight.crdUpgradeSafety.disabled 필드를 true 로 설정합니다.

예 5.22. ClusterExtension 오브젝트의 예

```
apiVersion: olm.operatorframework.io/v1alpha1
kind: ClusterExtension
metadata:
  name: clusterextension-sample
spec:
  installNamespace: default
  packageName: argocd-operator
  version: 0.6.0
  preflight:
    crdUpgradeSafety:
      disabled: true (1)
```

1

true 로 설정합니다.

5.3.4. 안전하지 않은 CRD 변경의 예

다음 예제에서는 CRD 업그레이드 안전 전 검사에 의해 catch되는 예제 CRD(사용자 정의 리소스 정의) 섹션의 특정 변경 사항을 보여줍니다.

다음 예제에서는 다음 시작 상태의 CRD 오브젝트를 고려하십시오.

예 5.23. CRD 오브젝트의 예

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    controller-gen.kubebuilder.io/version: v0.13.0
  name: example.test.example.com
spec:
  group: test.example.com
  names:
    kind: Sample
    listKind: SampleList
    plural: samples
    singular: sample
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            type: string
          kind:
            type: string
          metadata:
            type: object
          spec:
            type: object
          status:
            type: object
          pollInterval:
            type: string
        type: object
    served: true
```

```
    storage: true
    subresources:
      status: {}
```

5.3.4.1. 범위 변경

다음 CRD(사용자 정의 리소스 정의) 예에서 **scope** 필드가 **Namespaced** 에서 **Cluster** 로 변경됩니다.

예 5.24. CRD의 범위 변경 예

```
    spec:
      group: test.example.com
      names:
        kind: Sample
        listKind: SampleList
        plural: samples
        singular: sample
      scope: Cluster
      versions:
      - name: v1alpha1
```

예 5.25. 오류 출력 예

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoScopeChange" validation failed:
scope changed from "Namespaced" to "Cluster"
```

5.3.4.2. 저장된 버전 제거

다음 CRD(사용자 정의 리소스 정의) 예제에서는 기존 저장된 버전 **v1alpha1** 이 제거됩니다.

예 5.26. CRD에서 저장된 버전 제거 예

```
    versions:
    - name: v1alpha2
      schema:
        openAPIV3Schema:
          properties:
            apiVersion:
              type: string
            kind:
              type: string
            metadata:
              type: object
```

```
        spec:
          type: object
        status:
          type: object
        pollInterval:
          type: string
      type: object
```

예 5.27. 오류 출력 예

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoStoredVersionRemoved" validation
failed: stored version "v1alpha1" removed
```

5.3.4.3. 기존 필드 제거

다음 **CRD**(사용자 정의 리소스 정의) 예에서는 **pollInterval** 속성 필드가 **v1alpha1** 스키마에서 제거됩니다.

예 5.28. CRD의 기존 필드 제거 예

```
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            type: string
          kind:
            type: string
          metadata:
            type: object
          spec:
            type: object
          status:
            type: object
        type: object
```

예 5.29. 오류 출력 예

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "NoExistingFieldRemoved" validation
failed: crd/test.example.com version/v1alpha1 field/^.spec.pollInterval may not be removed
```

5.3.4.4. 필수 필드 추가

다음 CRD(사용자 정의 리소스 정의) 예에서는 pollInterval 속성이 필수 필드로 변경되었습니다.

예 5.30. CRD의 필수 필드 추가 예

```
versions:
- name: v1alpha2
  schema:
    openAPIV3Schema:
      properties:
        apiVersion:
          type: string
        kind:
          type: string
        metadata:
          type: object
        spec:
          type: object
        status:
          type: object
        pollInterval:
          type: string
      type: object
      required:
      - pollInterval
```

예 5.31. 오류 출력 예

```
validating upgrade for CRD "test.example.com" failed: CustomResourceDefinition
test.example.com failed upgrade safety validation. "ChangeValidator" validation failed:
version "v1alpha1", field "^": new required fields added: [pollInterval]
```