OpenShift Container Platform 4.11

Operator

OpenShift Container Platform의 Operator 작업

Last Updated: 2024-02-09

OpenShift Container Platform 4.11 Operator

OpenShift Container Platform의 Operator 작업

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 Operator를 사용하는 방법에 대해 설명합니다. 여기에는 클러스터 관리자를 위한 Operator 설치 및 관리 방법과 설치된 Operator에서 애플리케이션을 생성하는 방법에 대한 지침이 포함됩니다. 또한 Operator SDK를 사용하여 자체 Operator를 빌드하는 방법에 대한 지침이 포함되어 있습니다.

차례

1장. OPERATOR 개요

Operator는 OpenShift Container Platform의 가장 중요한 구성 요소 중 하나입니다. Operator는 컨트롤 플레인에서 서비스를 패키징, 배포 및 관리하는 기본 방법입니다. 또한 사용자가 실행하는 애플리케이션에 이점을 제공할 수 있습니다.

Operator는 **kubectl** 및 **oc** 명령과 같은 CLI 툴 및 Kubernetes API와 통합됩니다. 애플리케이션 모니터링, 상태 점검 수행, OTA(Over-The-Air) 업데이트 관리 및 애플리케이션이 지정된 상태로 유지되도록 하는 수단을 제공합니다.

둘 다 유사한 Operator 개념 및 목표를 따르지만 OpenShift Container Platform의 Operator는 목적에 따라 두 개의 다른 시스템에서 관리합니다.

- CVO(Cluster Version Operator)에서 관리하는 클러스터 Operator는 클러스터 기능을 수행하기 위해 기본적으로 설치됩니다.
- OLM(Operator Lifecycle Manager)에서 관리하는 선택적 애드온 Operator는 사용자가 애플리케이션에서 실행할 수 있도록 액세스할 수 있습니다.

Operator를 사용하면 클러스터에서 실행 중인 서비스를 모니터링할 애플리케이션을 생성할 수 있습니다. Operator는 애플리케이션용으로 특별히 설계되었습니다. Operator는 설치 및 구성과 같은 일반적인 Day 1 운영뿐만 아니라 자동 확장 및 축소 및 백업 생성과 같은 Day 2 작업을 구현하고 자동화합니다. 이러한 모든 활동은 클러스터 내에서 실행되는 소프트웨어에 포함되어 있습니다.

1.1. 개발자의 경우

개발자는 다음 Operator 작업을 수행할 수 있습니다.

- Operator SDK CLI를 설치합니다.
- Go 기반 Operator,Ansible 기반 Operator,Java 기반 Operator 및 Helm 기반 Operator 를 생성합니다.
- Operator SDK를 사용하여 Operator를 빌드, 테스트 및 배포합니다.
- Operator를 설치하고 네임스페이스에 서브스크립션합니다.
- 웹 콘솔을 통해 설치된 Operator에서 애플리케이션을 생성합니다.

추가 리소스

- Operator 개발자의 머신 삭제 라이프사이클 후크 예

1.2. 관리자의 경우

클러스터 관리자는 다음 Operator 작업을 수행할 수 있습니다.

- 사용자 정의 카탈로그 관리
- 비 클러스터 관리자가 Operator를 설치하도록 허용
- OperatorHub에서 Operator 설치
- Operator 상태를 표시합니다.

- Operator 조건 관리
- 설치된 Operator 업그레이드
- 설치된 Operator 삭제
- 프록시 지원 구성
- 제한된 네트워크에서 Operator Lifecycle Manager 사용

Red Hat에서 제공하는 클러스터 Operator에 대한 모든 내용을 알아보려면 Cluster Operators 참조 를 참조하십시오.

1.3. 다음 단계

Operator에 대한 자세한 내용은 Operator를 참조하십시오.
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/operators/#olm-what-operators-are

2장. OPERATOR 이해

2.1. OPERATOR란 무엇인가?

개념적으로 *Operator*는 사람의 운영 지식을 소비자와 더 쉽게 공유할 수 있는 소프트웨어로 인코딩합니다.

Operator는 다른 소프트웨어를 실행하는 데 따르는 운영의 복잡성을 완화해주는 소프트웨어입니다. 소프트웨어 벤더 엔지니어링 팀의 확장 기능처럼 작동하며 Kubernetes 환경(예: OpenShift Container Platform)을 모니터링하고 현재 상태를 사용하여 실시간으로 결정을 내립니다. 고급 Operator는 업그레이드를 원활하게 처리하고 오류에 자동으로 대응하며 시간을 절약하기 위해 소프트웨어 백업 프로세스를 생략하는 등의 바로 가기를 실행하지 않습니다.

Operator는 Kubernetes 애플리케이션을 패키징, 배포, 관리하는 메서드입니다.

Kubernetes 애플리케이션은 Kubernetes API 및 **kubectl** 또는 **oc** 툴링을 사용하여 Kubernetes API에 배포하고 관리하는 앱입니다. Kubernetes를 최대한 활용하기 위해서는 Kubernetes에서 실행되는 앱을 제공하고 관리하기 위해 확장할 응집력 있는 일련의 API가 필요합니다. Operator는 Kubernetes에서 이러한 유형의 앱을 관리하는 런타임으로 생각할 수 있습니다.

2.1.1. Operator를 사용하는 이유는 무엇입니까?

Operator는 다음과 같은 기능을 제공합니다.

- 반복된 설치 및 업그레이드.
- 모든 시스템 구성 요소에 대한 지속적인 상태 점검.
- OpenShift 구성 요소 및 ISV 콘텐츠에 대한 OTA(Over-The-Air) 업데이트
- 필드 엔지니어의 지식을 캡슐화하여 한두 명이 아닌 모든 사용자에게 전파.

Kubernetes에 배포하는 이유는 무엇입니까?

Kubernetes(및 확장으로)에는 온프레미스 및 클라우드 공급자 전체에서 작동하는 복잡한 분산 시스템을 빌드하는 데 필요한 모든 기본 기능(비밀 처리, 로드 밸런싱, 서비스 검색, 자동 스케일링)이 포함되어 있습니다.

Kubernetes API 및 kubectl 툴링으로 앱을 관리하는 이유는 무엇입니까?

이러한 API는 기능이 다양하고 모든 플랫폼에 대한 클라이언트가 있으며 클러스터의 액세스 제어/감사에 연결됩니다. Operator는 Kubernetes 확장 메커니즘인 CRD(사용자 정의 리소스 정의)를 사용하므로 사용자 정의 오브젝트(예: **MongoDB**)가 기본 제공되는 네이티브 Kubernetes 오브젝트처럼 보이고 작동합니다.

Operator는 서비스 브로커와 어떻게 다릅니까?

서비스 브로커는 앱의 프로그래밍 방식 검색 및 배포를 위한 단계입니다. 그러나 오래 실행되는 프로세스가 아니므로 업그레이드, 장애 조치 또는 스케일링과 같은 2일 차 작업을 실행할 수 없습니다. 튜닝할 수 있는 항목에 대한 사용자 정의 및 매개변수화는 설치 시 제공되지만 Operator는 클러스터의 현재 상태를 지속적으로 관찰합니다. 클러스터 외부 서비스는 서비스 브로커에 적합하지만 해당 서비스를 위한 Operator도 있습니다.

2.1.2. Operator 프레임워크

Operator 프레임워크는 위에서 설명한 고객 경험을 제공하는 툴 및 기능 제품군입니다. 코드를 작성하는 데 그치지 않고 Operator를 테스트, 제공, 업데이트하는 것이 중요합니다. Operator 프레임워크 구성 요소는 이러한 문제를 해결하는 오픈 소스 툴로 구성됩니다.

Operator SDK

Operator SDK는 Operator 작성자가 Kubernetes API 복잡성에 대한 지식이 없어도 전문 지식을 기반으로 자체 Operator를 부트스트랩, 빌드, 테스트, 패키지할 수 있도록 지원합니다.

Operator Lifecycle Manager

OLM(Operator Lifecycle Manager)은 클러스터에서 Operator의 설치, 업그레이드, RBAC(역할 기반 액세스 제어)를 제어합니다. OpenShift Container Platform 4.11에 기본적으로 배포됩니다.

Operator 레지스트리

Operator 레지스트리는 CSV(클러스터 서비스 버전) 및 CRD(사용자 정의 리소스 정의)를 클러스터에 생성하기 위해 저장하고 패키지 및 채널에 대한 Operator 메타데이터를 저장합니다. 이 Operator 카탈로그 데이터를 OLM에 제공하기 위해 Kubernetes 또는 OpenShift 클러스터에서 실행됩니다.

OperatorHub

OperatorHub는 클러스터 관리자가 클러스터에 설치할 Operator를 검색하고 선택할 수 있는 웹 콘솔입니다. OpenShift Container Platform에 기본적으로 배포됩니다.

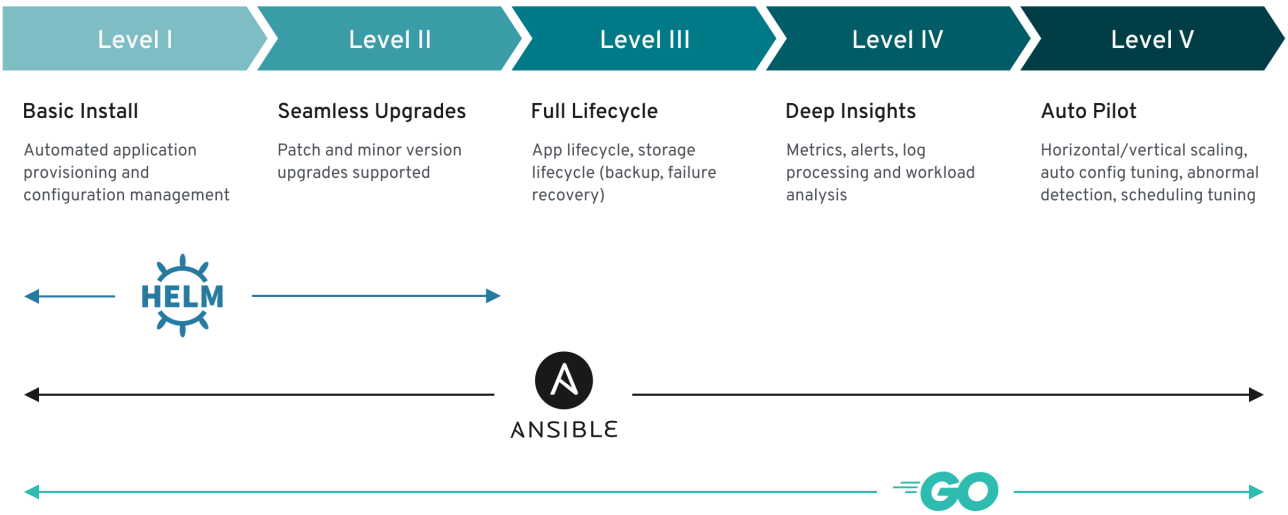
이러한 툴은 구성 가능하도록 설계되어 있어 유용한 툴을 모두 사용할 수 있습니다.

2.1.3. Operator 완성 모델

Operator 내에 캡슐화된 관리 논리의 세분화 수준은 다를 수 있습니다. 이 논리는 일반적으로 Operator에서 표시하는 서비스 유형에 따라 크게 달라집니다.

그러나 대부분의 Operator에 포함될 수 있는 특정 기능 세트의 경우 캡슐화된 Operator 작업의 완성 정도를 일반화할 수 있습니다. 이를 위해 다음 Operator 완성 모델에서는 Operator의 일반적인 2일 차 작업에 대해 5단계 완성도를 정의합니다.

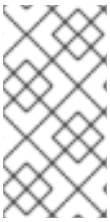
그림 2.1. Operator 완성 모델



또한 위 모델은 Operator SDK의 Helm, Go, Ansible 기능을 통해 이러한 기능을 가장 잘 개발할 수 있는 방법을 보여줍니다.

2.2. OPERATOR 프레임워크 패키지 형식

이 가이드에서는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)에서 지원하는 Operator의 패키지 형식에 대해 간단히 설명합니다.



참고

Operator에 대한 레거시 *패키지 매니페스트 형식* 지원은 OpenShift Container Platform 4.8 이상에서 제거됩니다. 패키지 매니페스트 형식의 기존 Operator 프로젝트는 Operator SDK **pkgman-to-bundle** 명령을 사용하여 번들 형식으로 마이그레이션할 수 있습니다. 자세한 내용은 번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션을 참조하십시오.

2.2.1. 번들 형식

Operator의 *번들 형식*은 Operator 프레임워크에서 도입한 패키지 형식입니다. 번들 형식 사양에서는 확장성을 개선하고 자체 카탈로그를 호스팅하는 업스트림 사용자를 더 효과적으로 지원하기 위해 Operator 메타데이터의 배포를 단순화합니다.

Operator 번들은 단일 버전의 Operator를 나타냅니다. 디스크상의 *번들 매니페스트*는 컨테이너화되어 *번들 이미지*(Kubernetes 매니페스트와 Operator 메타데이터를 저장하는 실행 불가능한 컨테이너 이미지)로 제공됩니다. 그런 다음 **podman** 및 **docker**와 같은 기존 컨테이너 툴과 Quay와 같은 컨테이너 레지스트리를 사용하여 번들 이미지의 저장 및 배포를 관리합니다.

Operator 메타데이터에는 다음이 포함될 수 있습니다.

- Operator 확인 정보(예: 이름 및 버전)
- UI를 구동하는 추가 정보(예: 아이콘 및 일부 예제 CR(사용자 정의 리소스))
- 필요한 API 및 제공된 API.
- 관련 이미지.

Operator 레지스트리 데이터베이스에 매니페스트를 로드할 때 다음 요구 사항이 검증됩니다.

- 번들의 주석에 하나 이상의 채널이 정의되어 있어야 합니다.
- 모든 번들에 정확히 하나의 CSV(클러스터 서비스 버전)가 있습니다.
- CSV에 CRD(사용자 정의 리소스 정의)가 포함된 경우 해당 CRD가 번들에 있어야 합니다.

2.2.1.1. 매니페스트

번들 매니페스트는 Operator의 배포 및 RBAC 모델을 정의하는 Kubernetes 매니페스트 세트를 나타냅니다.

번들에는 디렉터리당 하나의 CSV와 일반적으로 **/manifests** 디렉터리에서 CSV의 고유 API를 정의하는 CRD가 포함됩니다.

Bundle 형식 레이아웃의 예

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

추가 지원 오브젝트
다음과 같은 오브젝트 유형도 번들의 **/manifests** 디렉터리에 선택적으로 포함될 수 있습니다.

지원되는 선택적 오브젝트 유형

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

이러한 선택적 오브젝트가 번들에 포함된 경우 OLM(Operator Lifecycle Manager)은 번들에서 해당 오브젝트를 생성하고 CSV와 함께 해당 라이프사이클을 관리할 수 있습니다.

선택적 오브젝트의 라이프사이클

- CSV가 삭제되면 OLM은 선택적 오브젝트를 삭제합니다.
- CSV가 업그레이드되면 다음을 수행합니다.
    - 선택적 오브젝트의 이름이 동일하면 OLM에서 해당 오브젝트를 대신 업데이트합니다.
    - 버전 간 선택적 오브젝트의 이름이 변경된 경우 OLM은 해당 오브젝트를 삭제하고 다시 생성합니다.

2.2.1.2. 주석

번들의 **/metadata** 디렉터리에는 **annotations.yaml** 파일도 포함되어 있습니다. 이 파일에서는 번들을 번들 인덱스에 추가하는 방법에 대한 형식 및 패키지 정보를 설명하는 데 도움이 되는 고급 집계 데이터를 정의합니다.

예제 annotations.yaml

```
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 번들의 미디어 유형 또는 형식입니다. **registry+v1** 형식은 CSV 및 관련 Kubernetes 오브젝트가 포함됨을 나타냅니다.

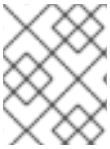
(2) Operator 매니페스트가 포함된 디렉터리의 이미지 경로입니다. 이 라벨은 나중에 사용할 수 있도록 예약되어 있으며 현재 기본값은 **manifests/**입니다. 값 **manifests.v1**은 번들에 Operator 매니페스트가 포함되어 있음을 나타냅니다.

(3) 번들에 대한 메타데이터 파일이 포함된 디렉터리의 이미지의 경로입니다. 이 라벨은 나중에 사용할 수 있도록 예약되어 있으며 현재 기본값은 **metadata/**입니다. 값 **metadata.v1**은 이 번들에 Operator 메타데이터가 있음을 나타냅니다.

(4) 번들의 패키지 이름입니다.

(5) Operator 레지스트리에 추가될 때 번들이 서브스크립션되는 채널 목록입니다.

(6) 레지스트리에서 설치할 때 Operator를 서브스크립션해야 하는 기본 채널입니다.



참고

불일치하는 경우 이러한 주석을 사용하는 클러스터상의 Operator 레지스트리만 이 파일에 액세스할 수 있기 때문에 **annotations.yaml** 파일을 신뢰할 수 있습니다.

2.2.1.3. 종속 항목

Operator의 종속 항목은 번들의 **metadata/** 폴더에 있는 **dependencies.yaml** 파일에 나열되어 있습니다. 이 파일은 선택 사항이며 현재는 명시적인 Operator 버전 종속 항목을 지정하는 데만 사용됩니다.

종속성 목록에는 종속성의 유형을 지정하기 위해 각 항목에 대한 **type** 필드가 포함되어 있습니다. 다음 유형의 Operator 종속성이 지원됩니다.

olm.package

이 유형은 특정 Operator 버전에 대한 종속성을 나타냅니다. 종속 정보에는 패키지 이름과 패키지 버전이 semver 형식으로 포함되어야 합니다. 예를 들어 **0.5.2**와 같은 정확한 버전이나 **>0.5.1**과 같은 버전 범위를 지정할 수 있습니다.

olm.gvk

이 유형을 사용하면 작성자는 CSV의 기존 CRD 및 API 기반 사용과 유사하게 GVK(그룹/버전/종류) 정보를 사용하여 종속성을 지정할 수 있습니다. 이 경로를 통해 Operator 작성자는 모든 종속 항목, API 또는 명시적 버전을 동일한 위치에 통합할 수 있습니다.

olm.constraint

이 유형은 임의의 Operator 속성에 대한 일반 제약 조건을 선언합니다.

다음 예제에서는 Prometheus Operator 및 etcd CRD에 대한 종속 항목을 지정합니다.

dependencies.yaml 파일의 예

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

추가 리소스

- Operator Lifecycle Manager 종속성 확인

2.2.1.4. opm CLI 정보

opm CLI 툴은 Operator 번들 형식과 함께 사용할 수 있도록 Operator 프레임워크에서 제공합니다. 이 툴을 사용하면 소프트웨어 리포지토리와 유사한 Operator 번들 목록에서 Operator 카탈로그를 생성하고 유지 관리할 수 있습니다. 결과적으로 컨테이너 레지스트리에 저장한 다음 클러스터에 설치할 수 있는 컨테이너 이미지가 생성됩니다.

카탈로그에는 컨테이너 이미지가 실행될 때 제공되는 포함된 API를 통해 쿼리할 수 있는 Operator 매니페스트 콘텐츠에 대한 포인터 데이터베이스가 포함되어 있습니다. OpenShift Container Platform에서 OLM(Operator Lifecycle Manager)은 **CatalogSource** 오브젝트에서 정의한 카탈로그 소스의 이미지를 참조할 수 있으며 주기적으로 이미지를 폴링하여 클러스터에 설치된 Operator를 자주 업데이트할 수 있습니다.

- **opm** CLI 설치 단계는 CLI 툴 을 참조하십시오.

2.2.2. 파일 기반 카탈로그

*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 YAML이므로 카탈로그 유지 관리자는 **jq** CLI와 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 Operator별 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 Operator별 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 API에서 추가 공식 툴을 개발할 수 있지만, 카탈로그 유지 관리자도 이러한 기능을 사용할 수 있다는 것이 가장 큰 장점입니다.

중요

OpenShift Container Platform 4.11부터 기본 Red Hat 제공 Operator 카탈로그는 파일 기반 카탈로그 형식으로 제공됩니다. 더 이상 사용되지 않는 SQLite 데이터베이스 형식으로 릴리스된 4.10을 통한 OpenShift Container Platform 4.6의 기본 Red Hat 제공 Operator 카탈로그입니다.

SQLite 데이터베이스 형식과 관련된 **opm** 하위 명령, 플래그 및 기능은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 기능은 계속 지원되며 더 이상 사용되지 않는 SQLite 데이터베이스 형식을 사용하는 카탈로그에 사용해야 합니다.

opm index prune 와 같은 SQLite 데이터베이스 형식을 사용하기 위한 많은 opm 하위 명령과 플래그는 파일 기반 카탈로그 형식으로 작동하지 않습니다. 파일 기반 카탈로그 사용에 대한 자세한 내용은 oc-mirror 플러그인을 사용하여 사용자 정의 카탈로그 관리 및 연결이 끊긴 설치의 이미지 미러링 을 참조하십시오.

2.2.2.1. 디렉토리 구조

디렉토리 기반 파일 시스템에서 파일 기반 카탈로그를 저장하고 로드할 수 있습니다. **opm** CLI는 루트 디렉토리로 이동하고 하위 디렉토리로 재귀하여 카탈로그를 로드합니다. CLI는 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 Blob을 별도의 하위 디렉터리에 저장하는 것이 좋습니다. 각 개별 파일은 JSON 또는 YAML일 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
```

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 카탈로그에도 포함할 수 있습니다.

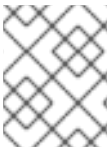
2.2.2.2. 스키마

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
}
```

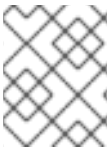


참고

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.

2.2.2.2.1. olm.package 스키마

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

예 2.1. 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
  }
}
```

2.2.2.2.2. olm.channel 스키마

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

번들은 여러 **olm.channel** Blob에 항목으로 포함될 수 있지만 채널당 하나의 항목만 있을 수 있습니다.

항목의 값이 이 카탈로그나 다른 카탈로그에서 찾을 수 없는 다른 번들 이름을 참조하는 것은 유효합니다. 그러나 여러 헤드가 없는 채널과 같이 다른 모든 채널 불변성은 true를 유지해야 합니다.

예 2.2. 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 & !=""
}
```

2.2.2.2.3. olm.bundle 스키마

예 2.3. 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 & !=""
}
```

2.2.2.3. 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 YAML일 수 있습니다.

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

2.2.2.3.1. 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 2.4. olm.package 속성

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

2.2.2.3.2. olm.gvk 속성

olm.gvk 속성은 이 번들에서 제공하는 Kubernetes API의 GVK(그룹/버전/종류)를 정의합니다. 이 속성은 OLM에서 이 속성이 포함된 번들을 필수 API와 동일한 GVK를 나열하는 다른 번들의 종속성으로 확인하는 데 사용됩니다. GVK는 Kubernetes GVK 검증을 준수해야 합니다.

예 2.5. olm.gvk 속성

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

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

2.2.2.3.3. 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 OLM은 나열된 패키지 및 필수 버전 범위에 대한 클러스터에 Operator가 설치되어 있는지 확인합니다. **versionRange** 필드는 유효한 의미 버전(semver) 범위여야 합니다.

예 2.6. olm.package.required 속성

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

2.2.2.3.4. olm.gvk.required

olm.gvk.required 속성은 이 번들에 필요한 Kubernetes API의 GVK(그룹/버전/종류)를 정의합니다. 번들 목록이 필요한 모든 GVK 속성에 대해 OLM은 이를 제공하는 클러스터에 Operator가 설치되어 있는지 확인합니다. GVK는 Kubernetes GVK 검증을 준수해야 합니다.

예 2.7. olm.gvk.required 속성

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

2.2.2.4. 카탈로그의 예

파일 기반 카탈로그를 사용하면 카탈로그 유지 관리자가 Operator 큐레이션 및 호환성에 중점을 둘 수 있습니다. Operator 작성자는 Operator에 대한 Operator별 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 Operator에 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@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f
6be03
- name: prometheus-operator
  image: quay.io/prometheus-
operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d84101
eb317
```

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" > "$file"
done
opm alpha generate dockerfile "$name"
indexImage=$(yq eval '.repo + ":" + .tag' catalog.yaml)
docker build -t "$indexImage" -f "$name.Dockerfile" .
docker push "$indexImage"
```

2.2.2.5. 지침

파일 기반 카탈로그를 유지 관리할 때 다음 지침을 고려하십시오.

2.2.2.5.1. 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 OLM은 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** blob에 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.

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

2.2.2.6. CLI 사용

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

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

2.2.2.7. 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 Operator가 성공적으로 설치될 수 있는지 확인합니다.
- 이전 검사를 통과하는 PR을 자동으로 병합합니다.
- 카탈로그 이미지를 자동으로 다시 빌드하고 다시 게시합니다.

2.3. OPERATOR 프레임워크 일반 용어집

이 주제에서는 OLM(Operator Lifecycle Manager) 및 Operator SDK를 포함하여 Operator 프레임워크와 관련된 일반 용어집을 제공합니다.

2.3.1. 일반 Operator 프레임워크 용어

2.3.1.1. 번들

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

2.3.1.2. 번들 이미지

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

2.3.1.3. 카탈로그 소스

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

2.3.1.4. 채널

*채널*은 Operator의 업데이트 스트림을 정의하고 구독자에게 업데이트를 배포하는 데 사용됩니다. 헤드는 해당 채널의 최신 버전을 가리킵니다. 예를 들어 **stable** 채널에는 Operator의 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.

2.3.1.5. 채널 헤드

*채널 헤드*는 알려진 특정 채널의 최신 업데이트를 나타냅니다.

2.3.1.6. 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.

2.3.1.7. 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.

2.3.1.8. 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.

2.3.1.9. 설치 계획

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

2.3.1.10. 멀티 테넌시

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

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

2.3.1.11. Operator group

*Operator group*은 동일한 네임스페이스에 배포된 모든 Operator를 **OperatorGroup** 오브젝트로 구성하여 네임스페이스 목록 또는 클러스터 수준에서 CR을 조사합니다.

2.3.1.12. 패키지

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

2.3.1.13. 레지스트리

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

2.3.1.14. Subscription

*서브스크립션*은 패키지의 채널을 추적하여 CSV를 최신 상태로 유지합니다.

2.3.1.15. 업데이트 그래프

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

2.4. OLM(OPERATOR LIFECYCLE MANAGER)

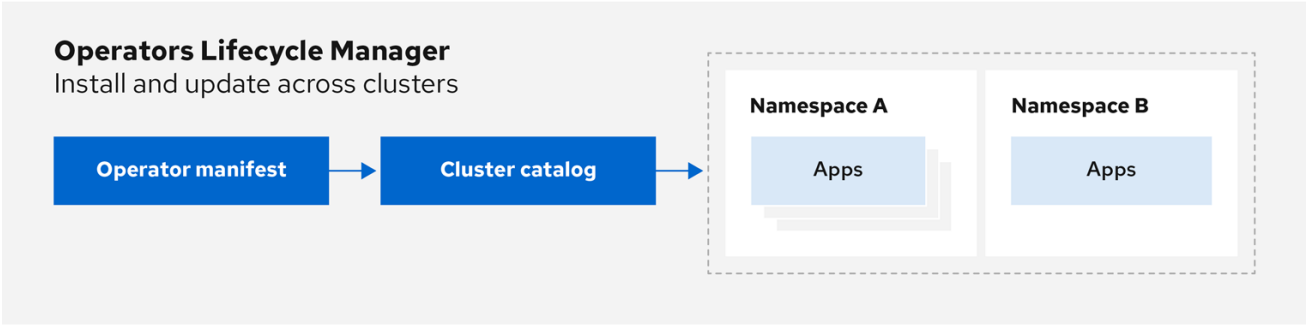
2.4.1. Operator Lifecycle Manager 개념 및 리소스

이 가이드에서는 OpenShift Container Platform에서 OLM(Operator Lifecycle Manager)을 구동하는 개념에 대한 개요를 제공합니다.

2.4.1.1. Operator Lifecycle Manager란?

OLM(*Operator Lifecycle Manager*)은 OpenShift Container Platform 클러스터에서 실행되는 Kubernetes 네이티브 애플리케이션(Operator) 및 관련 서비스의 라이프사이클을 설치, 업데이트, 관리하는 데 도움이 됩니다. Operator 프레임워크의 일부로, 효과적이고 자동화되었으며 확장 가능한 방식으로 Operator를 관리하도록 설계된 오픈 소스 툴킷입니다.

그림 2.2. Operator Lifecycle Manager 워크플로



OpenShift_43_1019

OLM은 OpenShift Container Platform 4.11에서 기본적으로 실행되므로 클러스터 관리자가 클러스터에서 실행되는 Operator를 설치, 업그레이드 및 액세스할 수 있도록 지원합니다. OpenShift Container Platform 웹 콘솔은 클러스터 관리자가 Operator를 설치할 수 있는 관리 화면을 제공하고, 클러스터에 제공되는 Operator 카탈로그를 사용할 수 있는 액세스 권한을 특정 프로젝트에 부여합니다.

개발자의 경우 분야별 전문가가 아니어도 셀프서비스 경험을 통해 데이터베이스, 모니터링, 빅 데이터 서비스의 인스턴스를 프로비저닝하고 구성할 수 있습니다. Operator에서 해당 지식을 제공하기 때문입니다.

2.4.1.2. OLM 리소스

다음 CRD(사용자 정의 리소스 정의)는 OLM(Operator Lifecycle Manager)에서 정의하고 관리합니다.

표 2.1. OLM 및 Catalog Operator에서 관리하는 CRD

리소스	짧은 이름	설명
ClusterServiceVersion(CSV)	**csv**	애플리케이션 메타데이터입니다. 예를 들면 이름, 버전, 아이콘, 필수 리소스입니다.
CatalogSource	**catsrc**	애플리케이션을 정의하는 CSV, CRD, 패키지의 리포지토리입니다.
서브스크립션	**sub**	패키지의 채널을 추적하여 CSV를 최신 상태로 유지합니다.
InstallPlan	**ip**	CSV를 자동으로 설치하거나 업그레이드하기 위해 생성하는 계산된 리소스 목록입니다.
OperatorGroup	**og**	동일한 네임스페이스에 배포된 모든 Operator를 **OperatorGroup** 오브젝트로 구성하여 네임스페이스 목록 또는 클러스터 수준에서 CR(사용자 정의 리소스)을 조사합니다.
OperatorConditions	-	OLM과 OLM에서 관리하는 Operator 간 통신 채널을 생성합니다. Operator는 복잡한 상태를 OLM에 보고하기 위해 **Status.Conditions** 어레이에 작성할 수 있습니다.

2.4.1.2.1. 클러스터 서비스 버전

CSV(*클러스터 서비스 버전*)는 OpenShift Container Platform 클러스터에서 실행 중인 특정 버전의 Operator를 나타냅니다. 클러스터에서 Operator를 실행할 때 OLM(Operator Lifecycle Manager)을 지원하는 Operator 메타데이터에서 생성한 YAML 매니페스트입니다.

이러한 Operator 관련 메타데이터는 OLM이 클러스터에서 Operator가 계속 안전하게 실행되도록 유지하고 새 버전의 Operator가 게시되면 업데이트 적용 방법에 대한 정보를 제공하는 데 필요합니다. 이는 기존 운영 체제의 패키징 소프트웨어와 유사합니다. OLM 패키징 단계를 **rpm**, **deb** 또는 **apk** 번들을 생성하는 단계로 고려해 보십시오.

CSV에는 Operator 컨테이너 이미지와 함께 제공되는 메타데이터가 포함되며 이러한 데이터는 이름, 버전, 설명, 라벨, 리포지토리 링크, 로고와 같은 정보로 사용자 인터페이스를 채우는 데 사용됩니다.

CSV는 Operator를 실행하는 데 필요한 기술 정보의 소스이기도 합니다(예: RBAC 규칙, 클러스터 요구 사항, 설치 전략을 관리하고 사용하는 CR(사용자 정의 리소스)). 이 정보는 OLM에 필요한 리소스를 생성하고 Operator를 배포로 설정하는 방법을 지정합니다.

2.4.1.2.2. 카탈로그 소스

*카탈로그 소스*는 일반적으로 컨테이너 레지스트리에 저장된 *인덱스 이미지*를 참조하여 메타데이터 저장소를 나타냅니다. OLM(Operator Lifecycle Manager)은 카탈로그 소스를 쿼리하여 Operator 및 해당 종속성을 검색하고 설치합니다. OpenShift Container Platform 웹 콘솔의 OperatorHub에는 카탈로그 소스에서 제공하는 Operator도 표시됩니다.

작은 정보

클러스터 관리자는 웹 콘솔의 **관리 → 클러스터 설정 → 구성 → OperatorHub** 페이지를 사용하여 클러스터에서 활성화된 카탈로그 소스에서 제공하는 전체 Operator 목록을 볼 수 있습니다.

CatalogSource 오브젝트의 **spec**은 Pod를 구성하는 방법 또는 Operator Registry gRPC API를 제공하는 서비스와 통신하는 방법을 나타냅니다.

예 2.8. CatalogSource 오브젝트의 예

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

```
  port: 50051
  protocol: grpc
  serviceName: example-catalog
  serviceNamespace: openshift-marketplace
```

1. **CatalogSource** 오브젝트의 이름입니다. 이 값은 요청된 네임스페이스에 생성된 관련 Pod의 이름으로도 사용됩니다.

2. 카탈로그를 생성할 네임스페이스입니다. 카탈로그를 모든 네임스페이스에서 클러스터 전체로 사용하려면 이 값을 **openshift-marketplace**로 설정합니다. 기본 Red Hat 제공 카탈로그 소스에서도 **openshift-marketplace** 네임스페이스를 사용합니다. 그러지 않으면 해당 네임스페이스에서만 Operator를 사용할 수 있도록 값을 특정 네임스페이스로 설정합니다.

3. 선택 사항: 클러스터 업그레이드를 통해 Operator 설치가 지원되지 않거나 지속적인 업데이트 경로가 없는 경우 클러스터 업그레이드의 일부로 Operator 카탈로그의 인덱스 이미지 버전을 자동으로 변경할 수 있습니다.

   olm.catalogImageTemplate 주석을 인덱스 이미지 이름으로 설정하고 이미지 태그의 템플릿을 구성할 때 표시된 대로 하나 이상의 Kubernetes 클러스터 버전 변수를 사용합니다. 이 주석은 런타임 시 **spec.image** 필드를 덮어씁니다. 자세한 내용은 "사용자 지정 카탈로그 소스의 이미지 템플릿" 섹션을 참조하십시오.

4. 웹 콘솔 및 CLI에 있는 카탈로그의 표시 이름입니다.

5. 카탈로그의 인덱스 이미지입니다. 선택적으로 런타임 시 pull 사양을 설정하는 **olm.catalogImageTemplate** 주석을 사용할 때 생략할 수 있습니다.

6. 카탈로그 소스의 가중치입니다. OLM은 종속성 확인 중에 가중치를 사용하여 우선순위를 지정합니다. 가중치가 높을수록 가중치가 낮은 카탈로그보다 카탈로그가 선호됨을 나타냅니다.

7. 소스 유형에는 다음이 포함됩니다.

   - **image** 참조가 있는 **grpc**: OLM이 이미지를 가져온 후 Pod를 실행합니다. Pod는 규격 API를 제공할 것으로 예상됩니다.
   - **address** 필드가 있는 **grpc**: OLM이 지정된 주소에서 gRPC API에 연결을 시도합니다. 대부분의 경우 사용해서는 안 됩니다.
   - **ConfigMap**: OLM은 구성 맵 데이터를 구문 분석하고 이에 대해 gRPC API를 제공할 수 있는 Pod를 실행합니다.

8. 선택 사항: **grpc** 유형 카탈로그 소스의 경우, 정의된 경우 **spec.image** 의 콘텐츠를 제공하는 Pod의 기본 노드 선택기를 덮어씁니다.

9. 선택 사항: **grpc** 유형 카탈로그 소스의 경우, 정의된 경우 **spec.image** 의 콘텐츠를 제공하는 Pod의 기본 우선 순위 클래스 이름을 덮어씁니다. Kubernetes는 기본적으로 **system-cluster-critical** 및 **system-node-critical** 우선순위 클래스를 제공합니다. 필드를 빈("")으로 설정하면 Pod에 기본 우선순위가 할당됩니다. 기타 우선순위 클래스는 수동으로 정의할 수 있습니다.

10. 선택 사항: **grpc** 유형 카탈로그 소스의 경우 정의된 경우 **spec.image** 의 콘텐츠를 제공하는 Pod의 기본 허용 오차를 덮어씁니다.

11. 지정된 간격으로 새 버전을 자동으로 확인하여 최신 상태를 유지합니다.

12. 카탈로그 연결의 마지막으로 관찰된 상태입니다. 예를 들어 다음과 같습니다.

    - **READY**: 성공적으로 연결되었습니다.

- **CONNECTING**: 계속 연결을 시도합니다.
- **TRANSIENT_FAILURE**: 연결을 시도하는 동안 시간 초과와 같은 일시적인 문제가 발생했습니다. 상태는 결국 **CONNECTING**으로 다시 전환되고 다시 연결 시도합니다.

자세한 내용은 gRPC 문서의 연결 상태를 참조하십시오.

13 카탈로그 이미지를 저장하는 컨테이너 레지스트리가 폴링되어 이미지가 최신 상태인지 확인할 수 있는 마지막 시간입니다.

14 카탈로그의 Operator 레지스트리 서비스의 상태 정보입니다.

서브스크립션의 **CatalogSource** 오브젝트 **name**을 참조하면 요청된 Operator를 찾기 위해 검색할 위치를 OLM에 지시합니다.

예 2.9. 카탈로그 소스를 참조하는 Subscription 오브젝트의 예

```
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
```

추가 리소스

- OperatorHub 이해
- Red Hat 제공 Operator 카탈로그
- 클러스터에 카탈로그 소스 추가
- 카탈로그 우선순위
- CLI를 사용하여 Operator 카탈로그 소스 상태 보기
- 카탈로그 소스 Pod 예약

2.4.1.2.2.1. 사용자 정의 카탈로그 소스의 이미지 템플릿

기본 클러스터와의 Operator 호환성은 다양한 방법으로 카탈로그 소스로 표시할 수 있습니다. 기본 Red Hat 제공 카탈로그 소스에 사용되는 한 가지 방법은 특정 플랫폼 릴리스용으로 특별히 생성된 인덱스 이미지의 이미지 태그를 확인하는 것입니다(예: OpenShift Container Platform 4.11).

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 4.10에서 4.11로 업그레이드하는 동안

redhat-operators 카탈로그의 **CatalogSource** 오브젝트의 **spec.image** 필드가 다음과 같이 업데이트됩니다.

```
registry.redhat.io/redhat/redhat-operator-index:v4.10
```

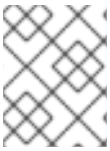
다음으로 변경합니다.

```
registry.redhat.io/redhat/redhat-operator-index:v4.11
```

그러나 CVO는 사용자 정의 카탈로그의 이미지 태그를 자동으로 업데이트하지 않습니다. 클러스터 업그레이드 후 사용자가 호환 가능하고 지원되는 Operator 설치를 유지하려면 업데이트된 인덱스 이미지를 참조하도록 사용자 정의 카탈로그도 업데이트해야 합니다.

OpenShift Container Platform 4.9부터 클러스터 관리자는 사용자 정의 카탈로그의 **CatalogSource** 오브젝트에 **olm.catalogImageTemplate** 주석을 템플릿이 포함된 이미지 참조에 추가할 수 있습니다. 템플릿에서 사용할 수 있도록 지원되는 Kubernetes 버전 변수는 다음과 같습니다.

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



참고

현재 템플릿에 사용할 수 없으므로 OpenShift Container Platform 클러스터 버전이 아닌 Kubernetes 클러스터 버전을 지정해야 합니다.

업데이트된 Kubernetes 버전을 지정하는 태그로 인덱스 이미지를 생성하고 푸시한 경우 이 주석을 설정하면 사용자 정의 카탈로그의 인덱스 이미지 버전이 클러스터 업그레이드 후 자동으로 변경될 수 있습니다. 주석 값은 **CatalogSource** 오브젝트의 **spec.image** 필드에서 이미지 참조를 설정하거나 업데이트하는 데 사용됩니다. 이로 인해 Operator가 지원되지 않는 상태이거나 지속적인 업데이트 경로가 없는 클러스터 업그레이드가 방지됩니다.

중요

업데이트된 태그가 있는 인덱스 이미지에 저장되어 있는 레지스트리가 클러스터 업그레이드 시 클러스터에서 액세스할 수 있는지 확인해야 합니다.

예 2.10. 이미지 템플릿이 있는 카탈로그 소스의 예

```
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:v1.24
priority: -400
publisher: Example Org
```

참고

spec.image 필드와 **olm.catalogImageTemplate** 주석이 둘 다 설정된 경우 주석의 확인된 값으로 **spec.image** 필드를 덮어씁니다. 주석이 사용 가능한 풀 사양으로 확인되지 않으면 카탈로그 소스는 설정된 **spec.image** 값으로 대체됩니다.

spec.image 필드가 설정되지 않고 주석이 사용 가능한 풀 사양으로 확인되지 않으면 OLM에서 카탈로그 소스의 조정을 중지하고 사람이 읽을 수 있는 오류 조건으로 설정합니다.

이전 예제의 **olm.catalogImageTemplate** 주석은 Kubernetes 1.24를 사용하는 OpenShift Container Platform 4.11 클러스터의 경우 다음 이미지 참조로 확인됩니다.

```
quay.io/example-org/example-catalog:v1.24
```

향후 OpenShift Container Platform 릴리스에서는 이후 OpenShift Container Platform 버전에서 사용하는 이후 Kubernetes 버전을 대상으로 하는 사용자 정의 카탈로그의 업데이트된 인덱스 이미지를 생성할 수 있습니다. 업그레이드 전에 **olm.catalogImageTemplate** 주석을 설정하여 클러스터를 이후 OpenShift Container Platform 버전으로 업그레이드하면 카탈로그의 인덱스 이미지도 자동으로 업데이트됩니다.

2.4.1.2.2.2. 카탈로그 상태 요구 사항

클러스터의 Operator 카탈로그는 설치 확인 관점에서 서로 바꿔 사용할 수 있습니다. **서브스크립션** 오브젝트는 특정 카탈로그를 참조할 수 있지만 클러스터의 모든 카탈로그를 사용하여 종속성을 해결합니다.

예를 들어 카탈로그 A가 비정상이면 카탈로그 A를 참조하는 서브스크립션은 일반적으로 A보다 카탈로그 우선 순위가 낮기 때문에 클러스터 관리자가 예상하지 못할 수 있는 카탈로그 B의 종속성을 확인할 수 있습니다.

결과적으로 OLM에서는 지정된 글로벌 네임스페이스(예: 기본 **openshift-marketplace** 네임스페이스 또는 사용자 정의 글로벌 네임스페이스)가 있는 모든 카탈로그가 정상이어야 합니다. 카탈로그가 비정상이면 공유 글로벌 네임스페이스 내의 모든 Operator 설치 또는 업데이트 작업이 **CatalogSourcesUnhealthy** 조건으로 실패합니다. 비정상적인 상태에서 이러한 작업이 허용된 경우 OLM에서 클러스터 관리자에게 예기치 않은 확인 및 설치 결정을 내릴 수 있습니다.

클러스터 관리자는 비정상 카탈로그를 관찰하고 카탈로그를 유효하지 않은 것으로 간주하고 Operator 설치를 재개하려는 경우 비정상 카탈로그 제거에 대한 자세한 내용은 "사용자 정의 카탈로그 제거" 또는 "기본 OperatorHub 카탈로그 소스 비활성화" 섹션을 참조하십시오.

추가 리소스

- 사용자 정의 카탈로그 제거
- 기본 OperatorHub 카탈로그 소스 비활성화

2.4.1.2.3. 서브스크립션

Subscription 오브젝트에서 정의하는 *서브스크립션*은 Operator를 설치하려는 의도를 나타냅니다. Operator와 카탈로그 소스를 연결하는 사용자 정의 리소스입니다.

서브스크립션은 Operator 패키지에서 구독할 채널과 업데이트를 자동 또는 수동으로 수행할지를 나타냅니다. 자동으로 설정된 경우 OLM(Operator Lifecycle Manager)은 서브스크립션을 통해 클러스터에서 항상 최신 버전의 Operator가 실행되도록 Operator를 관리하고 업그레이드합니다.

Subscription 개체 예

```
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
```

이 **Subscription** 오브젝트는 Operator의 이름 및 네임스페이스, Operator 데이터를 확인할 수 있는 카탈로그를 정의합니다. **alpha**, **beta** 또는 **stable**과 같은 채널은 카탈로그 소스에서 설치해야 하는 Operator 스트림을 결정하는 데 도움이 됩니다.

서브스크립션에서 채널 이름은 Operator마다 다를 수 있지만 이름 지정 스키마는 지정된 Operator 내의 공통 규칙을 따라야 합니다. 예를 들어 채널 이름은 Operator(**1.2**, **1.3**) 또는 릴리스 빈도(**stable**, **fast**)에서 제공하는 애플리케이션의 마이너 릴리스 업데이트 스트림을 따를 수 있습니다.

OpenShift Container Platform 웹 콘솔에서 쉽게 확인할 수 있을 뿐만 아니라 관련 서브스크립션의 상태를 검사하여 사용 가능한 최신 버전의 Operator가 있는 경우 이를 확인할 수 있습니다. **currentCSV** 필드와 연결된 값은 OLM에 알려진 최신 버전이고 **installedCSV**는 클러스터에 설치된 버전입니다.

추가 리소스

- 멀티 테넌시 및 Operator 공동 배치
- CLI를 사용하여 Operator 서브스크립션 상태 보기

2.4.1.2.4. 설치 계획

InstallPlan 오브젝트에서 정의하는 *설치 계획*은 OLM(Operator Lifecycle Manager)에서 특정 버전의 Operator로 설치 또는 업그레이드하기 위해 생성하는 리소스 세트를 설명합니다. 버전은 CSV(클러스터 서비스 버전)에서 정의합니다.

Operator, 클러스터 관리자 또는 Operator 설치 권한이 부여된 사용자를 설치하려면 먼저 **Subscription** 오브젝트를 생성해야 합니다. 서브스크립션은 카탈로그 소스에서 사용 가능한 Operator 버전의 스트림을 구독하려는 의도를 나타냅니다. 그런 다음 서브스크립션을 통해 Operator의 리소스를 쉽게 설치할 수 있도록 **InstallPlan** 오브젝트가 생성됩니다.

그런 다음 다음 승인 전략 중 하나에 따라 설치 계획을 승인해야 합니다.

- 서브스크립션의 **spec.installPlanApproval** 필드가 **Automatic**로 설정된 경우 설치 계획이 자동으로 승인됩니다.
- 서브스크립션의 **spec.installPlanApproval** 필드가 **Manual**로 설정된 경우 클러스터 관리자 또는 적절한 권한이 있는 사용자가 설치 계획을 수동으로 승인해야 합니다.

설치 계획이 승인되면 OLM에서 지정된 리소스를 생성하고 서브스크립션에서 지정한 네임스페이스에 Operator를 설치합니다.

예 2.11. InstallPlan 오브젝트의 예

```
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
  ...
```

추가 리소스

- 멀티 테넌시 및 Operator 공동 배치
- 비 클러스터 관리자가 Operator를 설치하도록 허용

2.4.1.2.5. Operator groups

OperatorGroup 리소스에서 정의하는 *Operator group*에서는 OLM에서 설치한 Operator에 다중 테넌트 구성을 제공합니다. Operator group은 멤버 Operator에 필요한 RBAC 액세스 권한을 생성할 대상 네임스페이스를 선택합니다.

대상 네임스페이스 세트는 쉼표로 구분된 문자열 형식으로 제공되며 CSV(클러스터 서비스 버전)의 **olm.targetNamespaces** 주석에 저장되어 있습니다. 이 주석은 멤버 Operator의 CSV 인스턴스에 적용되며 해당 배포에 프로젝션됩니다.

추가 리소스

- Operator groups

2.4.1.2.6. Operator 상태

OLM(Operator Lifecycle Manager)에서는 Operator의 라이프사이클을 관리하는 역할의 일부로, Operator를 정의하는 Kubernetes 리소스의 상태에서 Operator의 상태를 유추합니다. 이 접근 방식에서는 Operator가 지정된 상태에 있도록 어느 정도는 보장하지만 Operator에서 다른 방법으로는 유추할 수 없는 정보를 OLM에 보고해야 하는 경우가 많습니다. 그러면 OLM에서 이러한 정보를 사용하여 Operator의 라이프사이클을 더 효과적으로 관리할 수 있습니다.

OLM에서는 Operator에서 OLM에 조건을 보고할 수 있는 **OperatorCondition**이라는 CRD(사용자 정의 리소스 정의)를 제공합니다. **OperatorCondition** 리소스의 **Spec.Conditions** 어레이에 있는 경우 OLM의 Operator 관리에 영향을 줄 수 있는 일련의 조건이 지원됩니다.

참고

기본적으로 spec **.Conditions** 배열은 사용자가 추가하거나 사용자 정의 Operator 논리의 결과로 **OperatorCondition** 오브젝트에 존재하지 않습니다.

추가 리소스

- Operator 상태

2.4.2. Operator Lifecycle Manager 아키텍처

이 가이드에서는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager) 구성 요소 아키텍처를 간략하게 설명합니다.

2.4.2.1. 구성 요소의 역할

OLM(Operator Lifecycle Manager)은 OLM Operator와 Catalog Operator의 두 Operator로 구성됩니다.

각 Operator는 OLM 프레임워크의 기반이 되는 CRD(사용자 정의 리소스 정의)를 관리합니다.

표 2.2. OLM 및 Catalog Operator에서 관리하는 CRD

리소스	짧은 이름	소유자	Description
ClusterServiceVersion(CSV)	**csv**	OLM	애플리케이션 메타데이터: 이름, 버전, 아이콘, 필수 리소스, 설치 등입니다.
InstallPlan	**ip**	카탈로그	CSV를 자동으로 설치하거나 업그레이드하기 위해 생성하는 계산된 리소스 목록입니다.
CatalogSource	**catsrc**	카탈로그	애플리케이션을 정의하는 CSV, CRD, 패키지의 리포지토리입니다.
서브스크립션	**sub**	카탈로그	패키지의 채널을 추적하여 CSV를 최신 상태로 유지하는 데 사용됩니다.
OperatorGroup	**og**	OLM	동일한 네임스페이스에 배포된 모든 Operator를 **OperatorGroup** 오브젝트로 구성하여 네임스페이스 목록 또는 클러스터 수준에서 CR(사용자 정의 리소스)을 조사합니다.

또한 각 Operator는 다음 리소스를 생성합니다.

표 2.3. OLM 및 Catalog Operator에서 생성하는 리소스

리소스	소유자
Deployments	OLM
ServiceAccounts	
(Cluster)Roles	
(Cluster)RoleBindings	
CRD(**CustomResourceDefinitions**)	카탈로그
ClusterServiceVersions	

2.4.2.2. OLM Operator

CSV에 지정된 필수 리소스가 클러스터에 제공되면 OLM Operator는 CSV 리소스에서 정의하는 애플리케이션을 배포합니다.

OLM Operator는 필수 리소스 생성과는 관련이 없습니다. CLI 또는 Catalog Operator를 사용하여 이러한 리소스를 수동으로 생성하도록 선택할 수 있습니다. 이와 같은 분리를 통해 사용자는 애플리케이션에 활용하기 위해 선택하는 OLM 프레임워크의 양을 점차 늘리며 구매할 수 있습니다.

OLM Operator에서는 다음 워크플로를 사용합니다.

1. 네임스페이스에서 CSV(클러스터 서비스 버전)를 조사하고 해당 요구 사항이 충족되는지 확인합니다.
2. 요구사항이 충족되면 CSV에 대한 설치 전략을 실행합니다.

참고

설치 전략을 실행하기 위해서는 CSV가 Operator group의 활성 멤버여야 합니다.

2.4.2.3. Catalog Operator

Catalog Operator는 CSV(클러스터 서비스 버전) 및 CSV에서 지정하는 필수 리소스를 확인하고 설치합니다. 또한 채널에서 패키지 업데이트에 대한 카탈로그 소스를 조사하고 원하는 경우 사용 가능한 최신 버전으로 자동으로 업그레이드합니다.

채널에서 패키지를 추적하려면 원하는 패키지를 구성하는 **Subscription** 오브젝트, 채널, 업데이트를 가져오는 데 사용할 **CatalogSource** 오브젝트를 생성하면 됩니다. 업데이트가 확인되면 사용자를 대신하여 네임스페이스에 적절한 **InstallPlan** 오브젝트를 기록합니다.

Catalog Operator에서는 다음 워크플로를 사용합니다.

1. 클러스터의 각 카탈로그 소스에 연결합니다.
2. 사용자가 생성한 설치 계획 중 확인되지 않은 계획이 있는지 조사하고 있는 경우 다음을 수행합니다.
    a. 요청한 이름과 일치하는 CSV를 찾아 확인된 리소스로 추가합니다.

b. 각 관리 또는 필수 CRD의 경우 CRD를 확인된 리소스로 추가합니다.

c. 각 필수 CRD에 대해 이를 관리하는 CSV를 확인합니다.

3. 확인된 설치 계획을 조사하고 사용자의 승인에 따라 또는 자동으로 해당 계획에 대해 검색된 리소스를 모두 생성합니다.

4. 카탈로그 소스 및 서브스크립션을 조사하고 이에 따라 설치 계획을 생성합니다.

2.4.2.4. 카탈로그 레지스트리

Catalog 레지스트리는 클러스터에서 생성할 CSV 및 CRD를 저장하고 패키지 및 채널에 대한 메타데이터를 저장합니다.

*패키지 매니페스트*는 패키지 ID를 CSV 세트와 연결하는 카탈로그 레지스트리의 항목입니다. 패키지 내에서 채널은 특정 CSV를 가리킵니다. CSV는 교체하는 CSV를 명시적으로 참조하므로 패키지 매니페스트는 Catalog Operator에 각 중간 버전을 거쳐 CSV를 최신 버전으로 업데이트하는 데 필요한 모든 정보를 제공합니다.

2.4.3. Operator Lifecycle Manager 워크플로

이 가이드에서는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)의 워크플로를 간략하게 설명합니다.

2.4.3.1. OLM의 Operator 설치 및 업그레이드 워크플로

OLM(Operator Lifecycle Manager) 에코시스템에서 다음 리소스를 사용하여 Operator 설치 및 업그레이드를 확인합니다.

- **ClusterServiceVersion**(CSV)
- **CatalogSource**
- **서브스크립션**

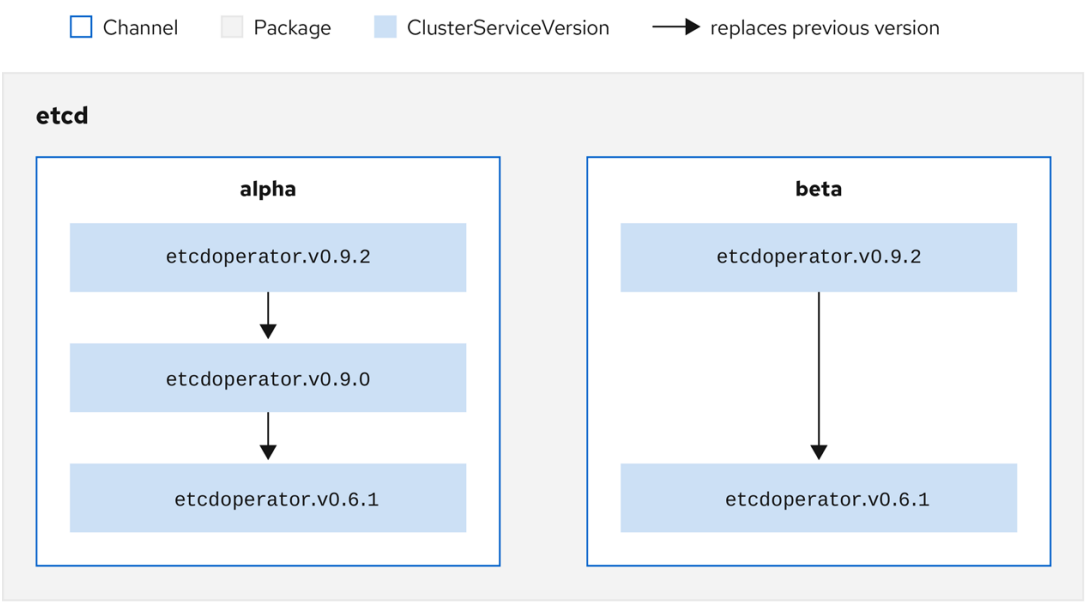
CSV에 정의된 Operator 메타데이터는 카탈로그 소스라는 컬렉션에 저장할 수 있습니다. OLM은 Operator Registry API를 사용하는 카탈로그 소스를 통해 사용 가능한 Operator와 설치된 Operator의 업그레이드를 쿼리합니다.

그림 2.3. 카탈로그 소스 개요

OpenShift_43_1019

Operator는 카탈로그 소스 내에서 *패키지*와 *채널*이라는 업데이트 스트림으로 구성되는데, 채널은 웹 브라우저와 같이 연속 릴리스 주기에서 OpenShift Container Platform 또는 기타 소프트웨어에 친숙한 업데이트 패턴이어야 합니다.

그림 2.4. 카탈로그 소스의 패키지 및 채널



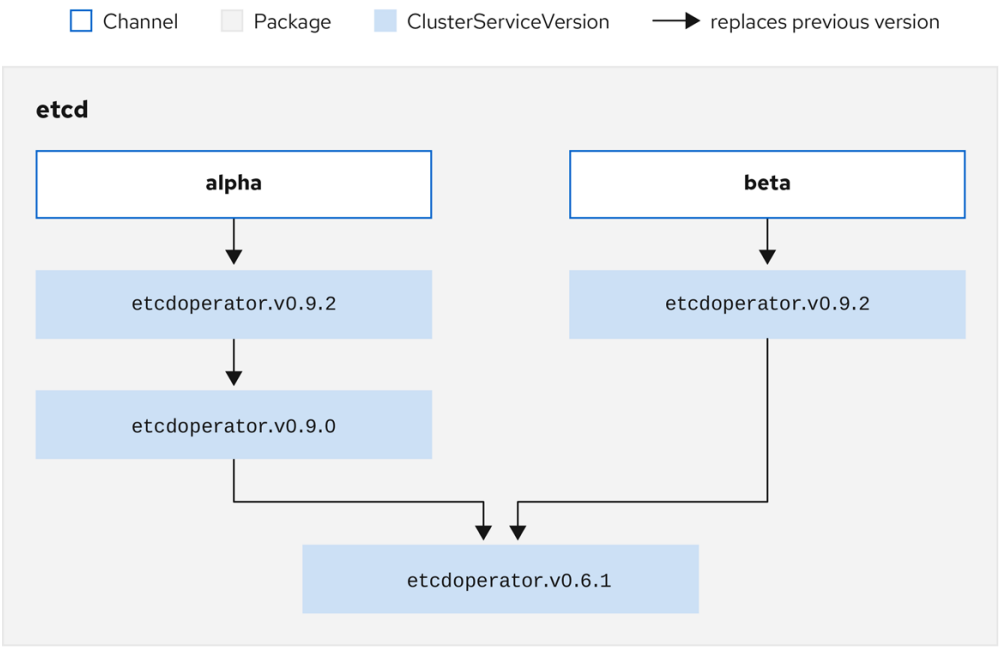
사용자는 *서브스크립션*의 특정 카탈로그 소스에서 특정 패키지 및 채널(예: **etcd** 패키지 및 해당 **alpha** 채널)을 나타냅니다. 네임스페이스에 아직 설치되지 않은 패키지에 서브스크립션이 생성되면 해당 패키지의 최신 Operator가 설치됩니다.

참고

OLM에서는 의도적으로 버전을 비교하지 않으므로 지정된 *카탈로그* → *채널* → *패키지* 경로에서 사용 가능한 "최신" Operator의 버전 번호가 가장 높은 버전 번호일 필요는 없습니다. Git 리포지토리와 유사하게 채널의 *헤드* 참조로 간주해야 합니다.

각 CSV에는 교체 대상 Operator를 나타내는 **replaces** 매개변수가 있습니다. 이 매개변수를 통해 OLM에서 쿼리할 수 있는 CSV 그래프가 빌드되고 업데이트를 채널 간에 공유할 수 있습니다. 채널은 업데이트 그래프의 진입점으로 간주할 수 있습니다.

그림 2.5. 사용 가능한 채널 업데이트의 OLM 그래프



패키지에 포함된 채널의 예

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

OLM에서 카탈로그 소스, 패키지, 채널, CSV와 관련된 업데이트를 쿼리하려면 카탈로그에서 입력 CSV를 **replaces**하는 단일 CSV를 모호하지 않게 결정적으로 반환할 수 있어야 합니다.

2.4.3.1.1. 업그레이드 경로의 예

업그레이드 시나리오 예제에서는 CSV 버전 **0.1.1**에 해당하는 Operator가 설치되어 있는 것으로 간주합니다. OLM은 카탈로그 소스를 쿼리하고 구독 채널에서 이전 버전이지만 설치되지 않은 CSV 버전 **0.1.2**를 교체하는(결국 설치된 이전 CSV 버전 **0.1.1**을 교체함) 새 CSV 버전 **0.1.3**이 포함된 업그레이드를 탐지합니다.

OLM은 CSV에 지정된 **replaces** 필드를 통해 채널 헤드에서 이전 버전으로 돌아가 업그레이드 경로 **0.1.3 → 0.1.2 → 0.1.1**을 결정합니다. 화살표 방향은 전자가 후자를 대체함을 나타냅니다. OLM은 채널 헤드에 도달할 때까지 Operator 버전을 한 번에 하나씩 업그레이드합니다.

지정된 이 시나리오의 경우 OLM은 Operator 버전 **0.1.2**를 설치하여 기존 Operator 버전 **0.1.1**을 교체합니다. 그런 다음 Operator 버전 **0.1.3**을 설치하여 이전에 설치한 Operator 버전 **0.1.2**를 대체합니다. 이 시점에 설치한 Operator 버전 **0.1.3**이 채널 헤드와 일치하며 업그레이드가 완료됩니다.

2.4.3.1.2. 업그레이드 건너뛰기

OLM의 기본 업그레이드 경로는 다음과 같습니다.

- 카탈로그 소스는 Operator에 대한 하나 이상의 업데이트로 업데이트됩니다.

- OLM은 카탈로그 소스에 포함된 최신 버전에 도달할 때까지 Operator의 모든 버전을 트래버스합니다.

그러나 경우에 따라 이 작업을 수행하는 것이 안전하지 않을 수 있습니다. 게시된 버전의 Operator가 아직 설치되지 않은 경우 클러스터에 설치해서는 안 되는 경우가 있습니다. 예를 들면 버전에 심각한 취약성이 있기 때문입니다.

이러한 경우 OLM에서는 두 가지 클러스터 상태를 고려하여 다음을 둘 다 지원하는 업데이트 그래프를 제공해야 합니다.

- "잘못"된 중간 Operator가 클러스터에 표시되고 설치되었습니다.
- "잘못된" 중간 Operator가 클러스터에 아직 설치되지 않았습니다.

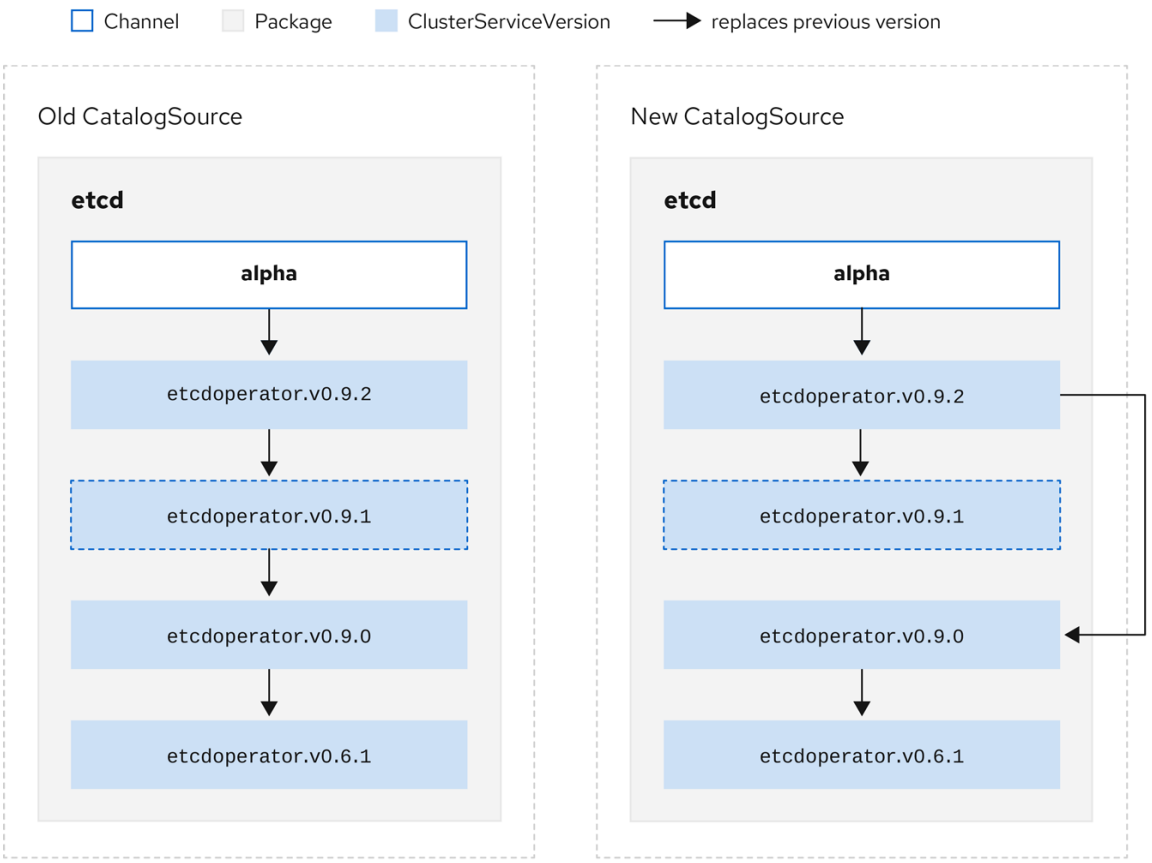
새 카탈로그를 제공하고 *건너뛰기* 릴리스를 추가하면 클러스터 상태 및 잘못된 업데이트가 있는지와 관계없이 OLM에서 항상 고유한 단일 업데이트를 가져올 수 있습니다.

릴리스를 건너뛰는 CSV의 예

```
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
```

기존 CatalogSource 및 **새 CatalogSource**의 다음 예제를 고려하십시오.

그림 2.6. 업데이트 건너뛰기



이 그래프에는 다음이 유지됩니다.

- **기존 CatalogSource**에 있는 모든 Operator에는 **새 CatalogSource**에 단일 대체 항목이 있습니다.
- **새 CatalogSource**에 있는 모든 Operator에는 **새 CatalogSource**에 단일 대체 항목이 있습니다.
- 잘못된 업데이트가 아직 설치되지 않은 경우 설치되지 않습니다.

2.4.3.1.3. 여러 Operator 교체

설명된 **새 CatalogSource**를 생성하려면 하나의 Operator를 **replace**하지만 여러 Operator를 **건너뛸** 수 있는 CSV를 게시해야 합니다. 이 작업은 **skipRange** 주석을 사용하여 수행할 수 있습니다.

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

여기서 **<semver_range>**에는 semver 라이브러리에서 지원하는 버전 범위 형식이 있습니다.

카탈로그에서 업데이트를 검색할 때 채널 헤드에 **skipRange** 주석이 있고 현재 설치된 Operator에 범위에 해당하는 버전 필드가 있는 경우 OLM이 채널의 최신 항목으로 업데이트됩니다.

우선순위 순서는 다음과 같습니다.

1. 기타 건너뛰기 기준이 충족되는 경우 서브스크립션의 **sourceName**에 지정된 소스의 채널 헤드
2. **sourceName**에 지정된 소스의 현재 Operator를 대체하는 다음 Operator

3. 기타 건너뛰기 조건이 충족되는 경우 서브스크립션에 표시되는 다른 소스의 채널 헤드.

4. 서브스크립션에 표시되는 모든 소스의 현재 Operator를 대체하는 다음 Operator.

skipRange가 있는 CSV의 예

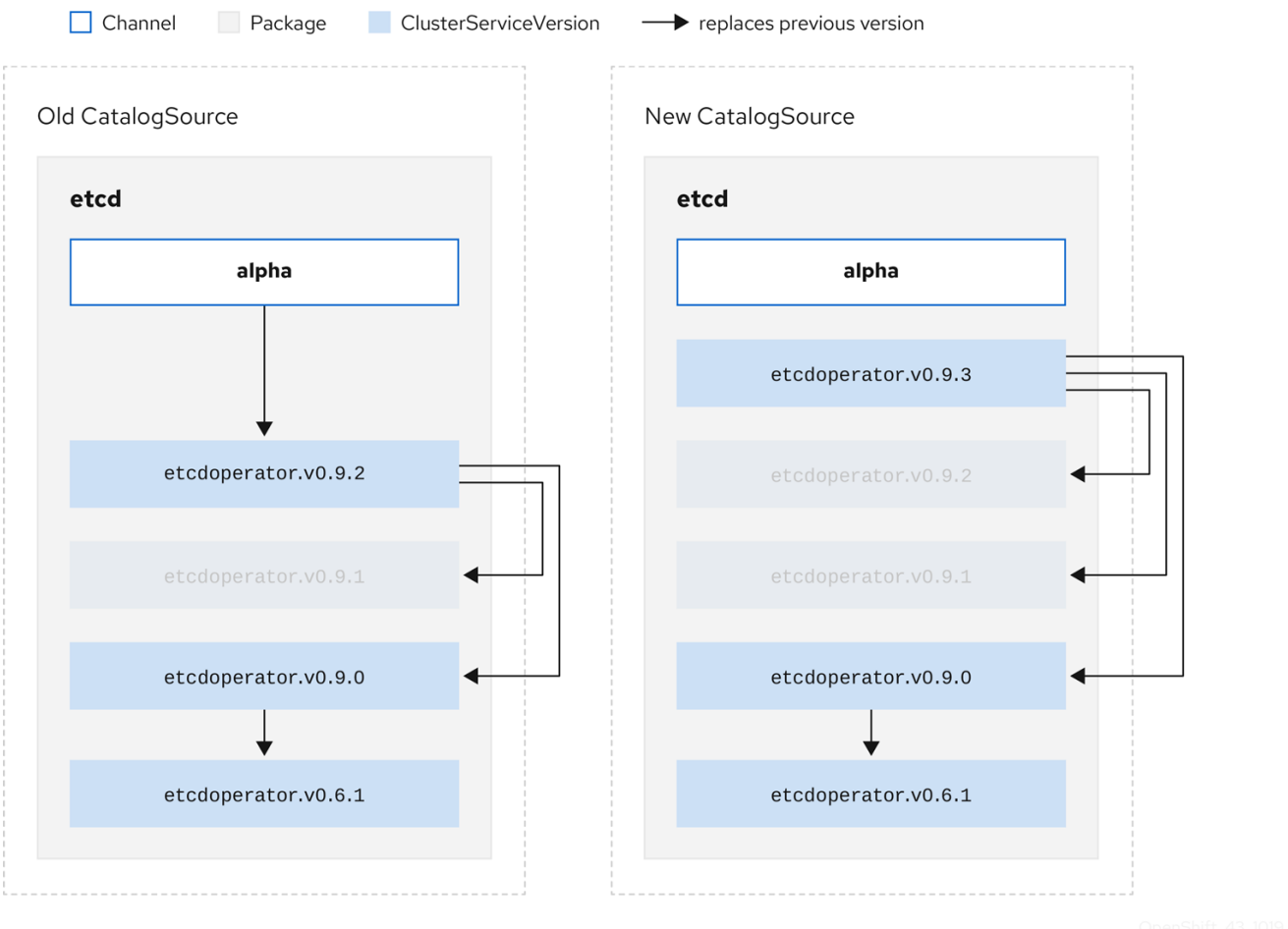
```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: elasticsearch-operator.v4.1.2
  namespace: <namespace>
  annotations:
    olm.skipRange: '>=4.1.0 <4.1.2'
```

2.4.3.1.4. z-stream 지원

마이너 버전이 동일한 경우 *z-stream* 또는 패치 릴리스로 이전 z-stream 릴리스를 모두 교체해야 합니다. OLM은 메이저, 마이너 또는 패치 버전을 구분하지 않으므로 카탈로그에 올바른 그래프만 빌드해야 합니다.

즉 OLM은 **이전 CatalogSource**에서와 같이 그래프를 가져올 수 있어야 하고 이전과 유사하게 **새 CatalogSource**에서와 같이 그래프를 생성할 수 있어야 합니다.

그림 2.7. 여러 Operator 교체



이 그래프에는 다음이 유지됩니다.

- **기존 CatalogSource**에 있는 모든 Operator에는 **새 CatalogSource**에 단일 대체 항목이 있습니다.
- **새 CatalogSource**에 있는 모든 Operator에는 **새 CatalogSource**에 단일 대체 항목이 있습니다.
- **이전 CatalogSource**의 모든 z-stream 릴리스가 **새 CatalogSource**의 최신 z-stream 릴리스로 업데이트됩니다.
- 사용할 수 없는 릴리스는 "가상" 그래프 노드로 간주할 수 있습니다. 해당 콘텐츠가 존재할 필요는 없으며 그래프가 이와 같은 경우 레지스트리에서 응답하기만 하면 됩니다.

2.4.4. Operator Lifecycle Manager 종속성 확인

이 가이드에서는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)을 사용한 종속성 해결 및 CRD(사용자 정의 리소스 정의) 업그레이드 라이프사이클에 대해 간단히 설명합니다.

2.4.4.1. 종속성 확인 정보

OLM(Operator Lifecycle Manager)은 실행 중인 Operator의 종속성 확인 및 업그레이드 라이프사이클을 관리합니다. OLM에서 발생하는 문제는 여러 가지 면에서 **yum** 및 **rpm** 과 같은 다른 시스템 또는 언어 패키지 관리자와 유사합니다.

그러나 OLM에는 유사한 시스템에는 일반적으로 해당하지 않는 한 가지 제약 조건이 있습니다. 즉 Operator가 항상 실행되고 있으므로 OLM에서 서로 함께 작동하지 않는 Operator 세트를 제공하지 않도록 합니다.

결과적으로 OLM은 다음 시나리오를 생성하지 않아야 합니다.

- 제공할 수 없는 API가 필요한 Operator 세트를 설치합니다.
- Operator에 종속된 다른 Operator를 중단하는 방식으로 Operator 업데이트

이 작업은 다음 두 가지 유형의 데이터에서 수행할 수 있습니다.

속성	종속성 확인자에서 공용 인터페이스를 구성하는 Operator에 대한 입력한 메타데이터입니다. 예를 들어 Operator에서 제공하는 API의 GVK(그룹/버전/종류)와 Operator의 시맨틱 버전(semver)이 있습니다.
제약 조건 또는 종속 항목	대상 클러스터에 이미 설치되어 있거나 설치되지 않았을 수 있는 다른 Operator에서 충족해야 하는 Operator의 요구 사항입니다. 이러한 작업은 사용 가능한 모든 Operator에 대한 쿼리 또는 필터 역할을 하며 종속성 확인 및 설치 중에 선택을 제한합니다. 예를 들어 클러스터에서 특정 API를 사용할 수 있거나 특정 버전이 있는 특정 Operator를 설치해야 하는 경우가 있습니다.

OLM은 이러한 속성과 제약 조건을 부울 공식 시스템으로 변환하고 이를 SAT solver(SAT solver)로 전달합니다. 이 프로그램은 Operator를 설치해야 하는 부울 사격성을 설정하는 프로그램입니다.

2.4.4.2. Operator 속성

카탈로그의 모든 Operator에는 다음과 같은 속성이 있습니다.

olm.package

패키지 이름 및 Operator 버전 포함

olm.gvk

CSV(클러스터 서비스 버전)에서 제공된 각 API의 단일 속성

Operator 번들의 **metadata/** 디렉터리에 **properties.yaml** 파일을 포함하여 Operator 작성자가 직접 추가 속성을 선언할 수도 있습니다.

임의의 속성 예

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

2.4.4.2.1. 임의의 속성

Operator 작성자는 Operator 번들의 **metadata/** 디렉터리에 **properties.yaml** 파일에 임의의 속성을 선언할 수 있습니다. 이러한 속성은 런타임 시 OLM(Operator Lifecycle Manager) 리졸버에 대한 입력으로 사용되는 맵 데이터 구조로 변환됩니다.

이러한 속성은 속성을 이해할 수 없으므로 확인자에서 불투명하지만 해당 속성에 대한 일반 제약 조건을 평가하여 제약 조건이 속성 목록을 제공할 수 있는지 여부를 결정할 수 있습니다.

임의의 속성 예

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

이 구조는 일반 제약 조건에 대한 CEL(Common Expression Language) 식을 구성하는 데 사용할 수 있습니다.

추가 리소스

- CEL(Common Expression Language) 제약 조건

2.4.4.3. Operator 종속 항목

Operator의 종속 항목은 번들의 **metadata/** 폴더에 있는 **dependencies.yaml** 파일에 나열되어 있습니다. 이 파일은 선택 사항이며 현재는 명시적인 Operator 버전 종속 항목을 지정하는 데만 사용됩니다.

종속성 목록에는 종속성의 유형을 지정하기 위해 각 항목에 대한 **type** 필드가 포함되어 있습니다. 다음 유형의 Operator 종속성이 지원됩니다.

olm.package

이 유형은 특정 Operator 버전에 대한 종속성을 나타냅니다. 종속 정보에는 패키지 이름과 패키지 버전이 semver 형식으로 포함되어야 합니다. 예를 들어 **0.5.2**와 같은 정확한 버전이나 **>0.5.1**과 같은 버전 범위를 지정할 수 있습니다.

olm.gvk

이 유형을 사용하면 작성자는 CSV의 기존 CRD 및 API 기반 사용과 유사하게 GVK(그룹/버전/종류) 정보를 사용하여 종속성을 지정할 수 있습니다. 이 경로를 통해 Operator 작성자는 모든 종속 항목, API 또는 명시적 버전을 동일한 위치에 통합할 수 있습니다.

olm.constraint

이 유형은 임의의 Operator 속성에 대한 일반 제약 조건을 선언합니다.

다음 예제에서는 Prometheus Operator 및 etcd CRD에 대한 종속 항목을 지정합니다.

dependencies.yaml 파일의 예

```
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. 일반 제약 조건

olm.constraint 속성은constraint가 아닌 속성을 구분하여 특정 유형의 종속성 제약 조건을 선언합니다. 해당 **value** 필드는 제약 조건 메시지를 나타내는 **failureMessage** 필드가 포함된 오브젝트입니다. 이 메시지는 런타임에 제약 조건이 만족되지 않은 경우 사용자에게 유용한 설명으로 표시됩니다.

다음 키는 사용 가능한 제약 조건 유형을 나타냅니다.

gvk

값 및 해석이 **olm.gvk** 유형과 동일한 유형을 입력합니다.

패키지

값 및 해석이 **olm.package** 유형과 동일한 유형을 입력합니다.

cel

런타임 시 임의의 번들 속성 및 클러스터 정보에 대해 OLM(Operator Lifecycle Manager)에서 평가한 CEL(Common Expression Language) 표현식

모든,어떤,아니

gvk 또는 중첩된 복합 제약 조건과 같은 하나 이상의 구체적인 제약 조건을 포함하는 결합, 분리 및 부정 제약 조건

2.4.4.4.1. CEL(Common Expression Language) 제약 조건

Cel **제약** 조건 유형은 표현식 언어로 CEL(Common Expression Language) 을 지원합니다. **cel** struct에는 런타임 시 Operator 속성에 대해 평가되는 CEL 표현식 문자열이 포함된 **rule** 필드가 있어 Operator가 제약 조건을 충족하는지 확인합니다.

cel 제약 조건의 예

-

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

CEL 구문은 **AND** 및 **또는** 와 같은 광범위한 논리 연산자를 지원합니다. 결과적으로 단일 CEL 식에는 이러한 논리 연산자가 함께 연결된 여러 조건에 대해 여러 규칙이 있을 수 있습니다. 이러한 규칙은 번들 또는 지정된 소스의 여러 속성 데이터 집합에 대해 평가되며 출력은 단일 제약 조건 내에서 해당 규칙을 모두 충족하는 단일 번들 또는 Operator로 해결됩니다.

여러 규칙이 있는 컬 제약 조건의 예

```
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. 혼합 제약 조건 (모두, any, not)

복합 제약 조건 유형은 논리 정의에 따라 평가됩니다.

다음은 두 패키지와 하나의 GVK의 개념적 제약 조건(**모든**)의 예입니다. 즉, 설치 번들에 의해 모두 충족되어야 합니다.

모든 제약 조건의 예

```
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
```

다음은 동일한 GVK의 세 가지 버전의 분산 제약 조건의 예입니다. **즉, 설치된 번들로 최소 1개 이상 충족되어야 합니다.**

제약 조건의 예

-

```
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
```

다음은 하나의 GVK 버전에 대한 부정 제약 조건의 예입니다. 즉, 결과 집합의 번들에서 이 GVK를 제공할 수 없습니다.

제약 조건 없음 예

```
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
```

부정 의미는 제약 조건 컨텍스트에서 명확하지 않은 것처럼 보일 수 있습니다. 명확히하기 위해 부정은 확인자에 특정 GVK, 버전 패키지를 포함하는 가능한 모든 솔루션을 제거하도록 지시하거나 결과 집합에서 일부 하위 복합 제약 조건을 충족하도록 지시합니다.

강제적으로는 가능한 종속 항목 집합을 선택하지 않고 부정할 수 있으므로 모든 또는 제약 조건 내에서만 사용할 수 있어야 합니다.

2.4.4.4.3. 중첩된 복합 제약 조건

하나 이상의 간단한 제약 조건과 함께 하나 이상의 자식 복합 제약 조건을 포함하는 중첩된 복합 제약 조건은 이전에 설명한 각 제약 조건 유형에 대한 프로시저에 따라 아래쪽에서 평가됩니다.A nested compound constraint, one that contains at least one child compound constraint along with zero or more simple constraints, is evaluated from the bottom up following the procedures for each previously described constraint type.

다음은 결합의 접합의 일례입니다. 여기서 하나, 다른 하나 또는 둘 다 제약 조건을 충족할 수 있습니다.

중첩된 복합 제약 조건의 예

```
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
```

참고

olm.constraint 유형의 최대 원시 크기는 리소스 소모 공격을 제한하는 **64KB**입니다.

2.4.4.5. 종속 기본 설정

Operator의 종속성을 동등하게 충족하는 옵션이 여러 개가 있을 수 있습니다. **OLM(Operator Lifecycle Manager)**의 종속성 확인자는 요청된 **Operator**의 요구 사항에 가장 적합한 옵션을 결정합니다. **Operator** 작성자 또는 사용자에게는 명확한 종속성 확인을 위해 이러한 선택이 어떻게 이루어지는지 이해하는 것이 중요할 수 있습니다.

2.4.4.5.1. 카탈로그 우선순위

OpenShift Container Platform 클러스터에서 **OLM**은 카탈로그 소스를 읽고 설치에 사용할 수 있는 **Operator**를 확인합니다.

CatalogSource 오브젝트의 예

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

CatalogSource 오브젝트에는 **priority** 필드가 있으며 확인자는 이 필드를 통해 종속성에 대한 옵션의 우선순위를 부여하는 방법을 확인합니다.

카탈로그 기본 설정을 관리하는 규칙에는 다음 두 가지가 있습니다.

- 우선순위가 높은 카탈로그의 옵션이 우선순위가 낮은 카탈로그의 옵션보다 우선합니다.
- 종속 항목과 동일한 카탈로그의 옵션이 다른 카탈로그보다 우선합니다.

2.4.4.5.2. 채널 순서

카탈로그의 **Operator** 패키지는 사용자가 **OpenShift Container Platform** 클러스터에서 구독할 수 있는 업데이트 채널 컬렉션입니다. 채널은 마이너 릴리스(**1.2, 1.3**) 또는 릴리스 빈도(**stable, fast**)에 대해 특정 업데이트 스트림을 제공하는 데 사용할 수 있습니다.

동일한 패키지에 있지만 채널이 다른 **Operator**로 종속성을 충족할 수 있습니다. 예를 들어 버전 **1.2**의 **Operator**는 **stable** 및 **fast** 채널 모두에 존재할 수 있습니다.

각 패키지에는 기본이 채널이 있으며 항상 기본이 아닌 채널에 우선합니다. 기본 채널의 옵션으로 종속성을 충족할 수 없는 경우 남아 있는 채널의 옵션을 채널 이름의 사전 순으로 고려합니다.

2.4.4.5.3. 채널 내 순서

대부분의 경우 단일 채널 내에는 종속성을 충족하는 옵션이 여러 개 있습니다. 예를 들어 하나의 패키지 및 채널에 있는 **Operator**에서는 동일한 **API** 세트를 제공합니다.

이는 사용자가 서브스크립션을 생성할 때 업데이트를 수신하는 채널을 나타냅니다. 이를 통해 검색 범위가 이 하나의 채널로 즉시 줄어듭니다. 하지만 채널 내에서 다수의 **Operator**가 종속성을 충족할 수 있습니다.

채널 내에서는 업데이트 그래프에서 더 높이 있는 최신 **Operator**가 우선합니다. 채널 헤드에서 종속성을 충족하면 먼저 시도됩니다.

2.4.4.5.4. 기타 제약 조건

패키지 종속 항목에서 제공하는 제약 조건 외에도 **OLM**에는 필요한 사용자 상태를 나타내고 확인 불변성을 적용하는 추가 제약 조건이 포함됩니다.

2.4.4.5.4.1. 서브스크립션 제약 조건

서브스크립션 제약 조건은 서브스크립션을 충족할 수 있는 Operator 세트를 필터링합니다. 서브스크립션은 종속성 확인자에 대한 사용자 제공 제약 조건입니다. Operator가 클러스터에 없는 경우 새 Operator를 설치하거나 기존 Operator를 계속 업데이트할지를 선언합니다.

2.4.4.5.4.2. 패키지 제약 조건

하나의 네임스페이스 내에 동일한 패키지의 두 Operator가 제공되지 않을 수 있습니다.

2.4.4.5.5. 추가 리소스

- 카탈로그 상태 요구 사항

2.4.4.6. CRD 업그레이드

OLM은 CRD(사용자 정의 리소스 정의)가 단수형 CSV(클러스터 서비스 버전)에 속하는 경우 CRD를 즉시 업그레이드합니다. CRD가 여러 CSV에 속하는 경우에는 다음과 같은 하위 호환 조건을 모두 충족할 때 CRD가 업그레이드됩니다.

- 현재 CRD의 기존 서비스 버전은 모두 새 CRD에 있습니다.
- CRD 제공 버전과 연결된 기존의 모든 인스턴스 또는 사용자 정의 리소스는 새 CRD의 검증 스키마에 대해 검증할 때 유효합니다.

추가 리소스

- 새 CRD 버전 추가
- CRD 버전 사용 중단 또는 제거

2.4.4.7. 종속성 모범 사례

종속 항목을 지정할 때는 모범 사례를 고려해야 합니다.

API 또는 특정 버전의 Operator 범위에 따라

Operator는 언제든지 API를 추가하거나 제거할 수 있습니다. 항상 Operator에서 요구하는 API에 olm.gvk 종속성을 지정합니다. 이에 대한 예외는 대신 olm.package 제약 조건을 지정하는 경우입니다.

최소 버전 설정

API 변경에 대한 Kubernetes 설명서에서는 Kubernetes 스타일 Operator에 허용되는 변경 사항을 설명합니다. 이러한 버전 관리 규칙을 사용하면 API가 이전 버전과 호환되는 경우 Operator에서 API 버전 충돌 없이 API를 업데이트할 수 있습니다.

Operator 종속 항목의 경우 이는 API 버전의 종속성을 확인하는 것으로는 종속 Operator가 의도한 대로 작동하는지 확인하는 데 충분하지 않을 수 있을 의미합니다.

예를 들면 다음과 같습니다.

- TestOperator v1.0.0에서는 v1alpha1 API 버전의 MyObject 리소스를 제공합니다.
- TestOperator v1.0.1에서는 새 필드 spec.newfield를 MyObject에 추가하지만 여전히 v1alpha1입니다.

Operator에 spec.newfield를 MyObject 리소스에 쓰는 기능이 필요할 수 있습니다. olm.gvk 제약 조건만으로는 OLM에서 TestOperator v1.0.0이 아닌 TestOperator v1.0.1이 필요한 것으로 판단하는 데 충분하지 않습니다.

가능한 경우 API를 제공하는 특정 Operator를 미리 알고 있는 경우 추가 olm.package 제약 조건을 지정하여 최솟값을 설정합니다.

최대 버전 생략 또는 광범위한 범위 허용

Operator는 API 서비스 및 CRD와 같은 클러스터 범위의 리소스를 제공하기 때문에 짧은 종속성 기간을 지정하는 Operator는 해당 종속성의 다른 소비자에 대한 업데이트를 불필요하게 제한할 수 있습니다.

가능한 경우 최대 버전을 설정하지 마십시오. 또는 다른 Operator와 충돌하지 않도록 매우 광범위한 의미 범위를 설정하십시오. 예를 들면 >1.0.0 <2.0.0과 같습니다.

기존 패키지 관리자와 달리 Operator 작성자는 OLM의 채널을 통해 업데이트가 안전함을 명시적으로 인코딩합니다. 기존 서브스크립션에 대한 업데이트가 제공되면 Operator 작성자가 이전 버전에

서 업데이트할 수 있음을 나타내는 것으로 간주합니다. 종속성에 최대 버전을 설정하면 특정 상한에서 불필요하게 잘라 작성자의 업데이트 스트림을 덮어씁니다.

참고

클러스터 관리자는 Operator 작성자가 설정한 종속 항목을 덮어쓸 수 없습니다.

그러나 피해야 하는 알려진 비호환성이 있는 경우 최대 버전을 설정할 수 있으며 설정해야 합니다. 버전 범위 구문을 사용하여 특정 버전을 생략할 수 있습니다(예: > 1.0.0 !1.2.1).

추가 리소스

- Kubernetes 설명서: API 변경

2.4.4.8. 종속성 경고

종속성을 지정할 때 고려해야 할 경고 사항이 있습니다.

혼합 제약 조건(AND) 없음

현재 제약 조건 간 AND 관계를 지정할 수 있는 방법은 없습니다. 즉 하나의 Operator가 지정된 API를 제공하면서 버전이 >1.1.0인 다른 Operator에 종속되도록 지정할 수 없습니다.

즉 다음과 같은 종속성을 지정할 때를 나타냅니다.

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

OLM은 EtcdCluster를 제공하는 Operator와 버전이 >3.1.0인 Operator를 사용하여 이러한 조건을 충족할 수 있습니다. 이러한 상황이 발생하는지 또는 두 제약 조건을 모두 충족하는 Operator가 선택되었는지는 잠재적 옵션을 방문하는 순서에 따라 다릅니다. 종속성 기본 설정 및 순서 지정 옵션은

잘 정의되어 있으며 추론할 수 있지만 주의를 기울이기 위해 **Operator**는 둘 중 하나의 메커니즘을 유지해야 합니다.

네임스페이스 간 호환성

OLM은 네임스페이스 범위에서 종속성 확인을 수행합니다. 한 네임스페이스의 **Operator**를 업데이트하면 다른 네임스페이스의 **Operator**에 문제가 되고 반대의 경우도 마찬가지인 경우 업데이트 교착 상태에 빠질 수 있습니다.

2.4.4.9. 종속성 확인 시나리오 예제

다음 예제에서 *공급자*는 **CRD** 또는 **API** 서비스를 "보유"한 **Operator**입니다.

예: 종속 **API** 사용 중단

A 및 **B**는 다음과 같은 **API**입니다(**CRD**).

- **A** 공급자는 **B**에 의존합니다.
- **B** 공급자에는 서브스크립션이 있습니다.
- **C**를 제공하도록 **B** 공급자를 업데이트하지만 **B**를 더 이상 사용하지 않습니다.

결과는 다음과 같습니다.

- **B**에는 더 이상 공급자가 없습니다.
- **A**가 더 이상 작동하지 않습니다.

이는 **OLM**에서 업그레이드 전략으로 방지하는 사례입니다.

예: 버전 교착 상태

A 및 **B**는 다음과 같은 **API**입니다.

- A 공급자에는 B가 필요합니다.
- B 공급자에는 A가 필요합니다.
- A 공급자를 업데이트(A2 제공, B2 요청)하고 A를 더 이상 사용하지 않습니다.
- B 공급자를 업데이트(A2 제공, B2 요청)하고 B를 더 이상 사용하지 않습니다.

OLM에서 B를 동시에 업데이트하지 않고 A를 업데이트하거나 반대 방향으로 시도하는 경우 새 호환 가능 세트가 있는 경우에도 새 버전의 Operator로 진행할 수 없습니다.

이는 OLM에서 업그레이드 전략으로 방지하는 또 다른 사례입니다.

2.4.5. Operator groups

이 가이드에서는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)에서 Operator groups을 사용하는 방법을 간략하게 설명합니다.

2.4.5.1. Operator groups 정의

OperatorGroup 리소스에서 정의하는 *Operator group*에서는 OLM에서 설치한 Operator에 다중 테넌트 구성을 제공합니다. Operator group은 멤버 Operator에 필요한 RBAC 액세스 권한을 생성할 대상 네임스페이스를 선택합니다.

대상 네임스페이스 세트는 쉼표로 구분된 문자열 형식으로 제공되며 CSV(클러스터 서비스 버전)의 olm.targetNamespaces 주석에 저장되어 있습니다. 이 주석은 멤버 Operator의 CSV 인스턴스에 적용되며 해당 배포에 프로젝션됩니다.

2.4.5.2. Operator group 멤버십

다음 조건이 충족되면 Operator가 Operator group의 *멤버*로 간주됩니다.

- Operator의 CSV는 Operator group과 동일한 네임스페이스에 있습니다.

- **Operator의 CSV 설치 모드에서는 Operator group이 대상으로 하는 네임스페이스 세트를 지원합니다.**

CSV의 설치 모드는 InstallModeType 필드 및 부울 Supported 필드로 구성됩니다. CSV 사양에는 다음 네 가지 InstallModeTypes로 구성된 설치 모드 세트가 포함될 수 있습니다.

표 2.4. 설치 모드 및 지원되는 Operator groups

InstallModeType	설명
OwnNamespace	Operator가 자체 네임스페이스를 선택하는 Operator group의 멤버일 수 있습니다.
SingleNamespace	Operator가 하나의 네임스페이스를 선택하는 Operator group의 멤버일 수 있습니다.
MultiNamespace	Operator가 네임스페이스를 두 개 이상 선택하는 Operator group의 멤버일 수 있습니다.
AllNamespaces	Operator가 네임스페이스를 모두 선택하는 Operator group의 멤버일 수 있습니다(대상 네임스페이스 세트는 빈 문자열("")임).

참고

CSV 사양에서 InstallModeType 항목이 생략되면 암시적으로 지원하는 기존 항목에서 지원을 유추할 수 있는 경우를 제외하고 해당 유형을 지원하지 않는 것으로 간주합니다.

2.4.5.3. 대상 네임스페이스 선택

spec.targetNamespaces 매개변수를 사용하여 Operator group의 대상 네임스페이스의 이름을 명시적으로 지정할 수 있습니다.

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

주의

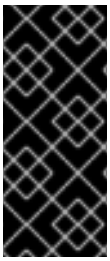
OLM(Operator Lifecycle Manager)은 각 **Operator** 그룹에 대해 다음 클러스터 역할을 생성합니다.

- **<operatorgroup_name>-admin**
- **<operatorgroup_name>-edit**
- **<operatorgroup_name>-view**

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 **Operator** 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

또는 라벨 선택기를 **spec.selector** 매개변수와 함께 사용하여 네임스페이스를 지정할 수도 있습니다.

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



중요

spec.targetNamespaces를 통해 여러 네임스페이스를 나열하거나 **spec.selector**를 통해 라벨 선택기를 사용하는 것은 바람직하지 않습니다. **Operator group**의 대상 네임스페이스 두 개 이상에 대한 지원이 향후 릴리스에서 제거될 수 있습니다.

spec.targetNamespaces 및 **spec.selector**를 둘 다 정의하면 **spec.selector**가 무시됩니다. 또는 모든 네임스페이스를 선택하는 *global* **Operator group**을 지정하려면 **spec.selector** 및 **spec.targetNamespaces**를 둘 다 생략하면 됩니다.

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

선택된 네임스페이스의 확인된 세트는 Opeator group의 status.namespaces 매개변수에 표시됩니다. 글로벌 Operator group의 status.namespace에는 사용 중인 Operator에 모든 네임스페이스를 조사해야 함을 알리는 빈 문자열("")이 포함됩니다.

2.4.5.4. Operator group CSV 주석

Operator group의 멤버 CSV에는 다음과 같은 주석이 있습니다.

주석	설명
olm.operatorGroup=<group_name>	Operator group의 이름을 포함합니다.
olm.operatorNamespace=<group_namespace>	Operator group의 네임스페이스를 포함합니다.
olm.targetNamespaces=<target_namespaces>	Operator group의 대상 네임스페이스 선택 사항을 나열하는 쉼표로 구분된 문자열을 포함합니다.

참고

olm.targetNamespaces를 제외한 모든 주석은 CSV 복사본에 포함됩니다. CSV 복제본에서 olm.targetNamespaces 주석을 생략하면 테넌트 간에 대상 네임스페이스를 복제할 수 없습니다.

2.4.5.5. 제공된 API 주석

*GVK(그룹/버전/종류)*는 Kubernetes API의 고유 ID입니다. Operator group에서 제공하는 GVK에 대한 정보는 olm.providedAPIs 주석에 표시됩니다. 주석 값은 <kind>.<version>.<group>으로 구성된 문자열로, 쉼표로 구분됩니다. Operator group의 모든 활성 멤버 CSV에서 제공하는 CRD 및 API 서비스의 GVK가 포함됩니다.

PackageManifest 리소스를 제공하는 하나의 활성 멤버 CSV에서 OperatorGroup 오브젝트의 다음 예제를 검토합니다.

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

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

2.4.5.6. 역할 기반 액세스 제어

Operator v이 생성되면 세 개의 클러스터 역할이 생성됩니다. 각 역할에는 다음과 같이 라벨과 일치하도록 클러스터 역할 선택기가 설정된 단일 집계 규칙이 포함됩니다.

클러스터 역할	일치해야 하는 라벨
<operatorgroup_name>-admin	olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>
<operatorgroup_name>-edit	olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>
<operatorgroup_name>-view	olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>

주의

OLM(Operator Lifecycle Manager)은 각 Operator 그룹에 대해 다음 클러스터 역할을 생성합니다.

- <operatorgroup_name>-admin
- <operatorgroup_name>-edit
- <operatorgroup_name>-view

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 Operator 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

다음 RBAC 리소스는 CSV가 AllNamespaces 설치 모드로 모든 네임스페이스를 조사하고 이유가 InterOperatorGroupOwnerConflict인 실패 상태가 아닌 한 CSV가 Operator group의 활성 멤버가 될 때 생성됩니다.

- CRD의 각 API 리소스에 대한 클러스터 역할
- API 서비스의 각 API 리소스에 대한 클러스터 역할
- 추가 역할 및 역할 바인딩

표 2.5. CRD에서 각 API 리소스에 대해 생성된 클러스터 역할

클러스터 역할	설정

클러스터 역할	설정
<kind>.<group>-<version>-admin	**<kind>**의 동사: ● ***** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-admin: true** ● **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>**
<kind>.<group>-<version>-edit	**<kind>**의 동사: ● **create** ● **update** ● **patch** ● **delete** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-edit: true** ● **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>**
<kind>.<group>-<version>-view	**<kind>**의 동사: ● **get** ● **list** ● **watch** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-view: true** ● **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

클러스터 역할	설정
<kind>.<group>-<version>-view-crdview	**apiextensions.k8s.io customresourcedefinitions <crd-name>**의 동사: ● **get** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-view: true** ● **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

표 2.6. API 서비스에서 각 API 리소스에 대해 생성한 클러스터 역할

클러스터 역할	설정
<kind>.<group>-<version>-admin	**<kind>**의 동사: ● *** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-admin: true** ● **olm.opgroup.permissions/aggregate-to-admin: <operatorgroup_name>**
<kind>.<group>-<version>-edit	**<kind>**의 동사: ● **create** ● **update** ● **patch** ● **delete** 집계 라벨: ● **rbac.authorization.k8s.io/aggregate-to-edit: true** ● **olm.opgroup.permissions/aggregate-to-edit: <operatorgroup_name>**

클러스터 역할	설정
<kind>.<group>-<version>-view	**<kind>**의 동사: • **get** • **list** • **watch** 집계 라벨: • **rbac.authorization.k8s.io/aggregate-to-view: true** • **olm.opgroup.permissions/aggregate-to-view: <operatorgroup_name>**

추가 역할 및 역할 바인딩

- **CSV**에서 *를 포함하는 정확히 하나의 대상 네임스페이스를 정의하는 경우 **CSV**의 **permissions** 필드에 정의된 각 권한에 대해 클러스터 역할 및 해당 클러스터 역할 바인딩이 생성됩니다. 생성된 모든 리소스에는 **olm.owner: <csv_name>** 및 **olm.owner.namespace: <csv_namespace>** 라벨이 지정됩니다.

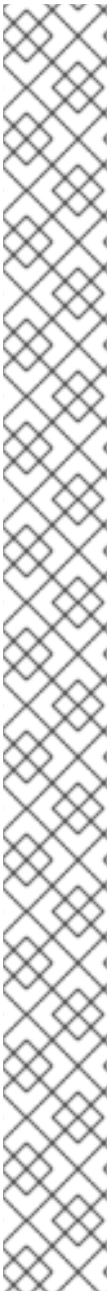
- **CSV**에서 *를 포함하는 정확히 하나의 대상 네임스페이스를 정의하지 *않는* 경우에는 **olm.owner: <csv_name>** 및 **olm.owner.namespace: <csv_namespace>** 라벨이 있는 **Operator** 네임스페이스의 모든 역할 및 역할 바인딩이 대상 네임스페이스에 복사됩니다.

2.4.5.7. CSV 복사본

OLM은 해당 **Operator group**의 각 대상 네임스페이스에서 **Operator group**의 모든 활성 멤버에 대한 **CSV** 복사본을 생성합니다. **CSV** 복사본의 용도는 대상 네임스페이스의 사용자에게 특정 **Operator**가 그곳에서 생성된 리소스를 조사하도록 구성됨을 알리는 것입니다.

CSV 복사본은 상태 이유가 **Copied**이고 해당 소스 **CSV**의 상태와 일치하도록 업데이트됩니다. **olm.targetNamespaces** 주석은 해당 주석이 클러스터에서 생성되기 전에 **CSV** 복사본에서 제거됩니다. 대상 네임스페이스 선택 단계를 생략하면 테넌트 간 대상 네임스페이스가 중복되지 않습니다.

CSV 복사본은 복사본의 소스 **CSV**가 더 이상 존재하지 않거나 소스 **CSV**가 속한 **Operator group**이 더 이상 **CSV** 복사본의 네임스페이스를 대상으로 하지 않는 경우 삭제됩니다.



참고

기본적으로 **disableCopiedCSVs** 필드는 비활성화되어 있습니다. **disableCopiedCSVs** 필드를 활성화하면 **OLM**이 클러스터에서 기존 **CSV**를 삭제합니다. **disableCopiedCSVs** 필드가 비활성화되면 **OLM**에서 **CSV** 복사본을 다시 추가합니다.

- **disableCopiedCSVs** 필드를 비활성화합니다.

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

- **disableCopiedCSVs** 필드를 활성화합니다.

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

2.4.5.8. 정적 Operator groups

spec.staticProvidedAPIs 필드가 **true**로 설정된 경우 **Operator group**은 ***static***입니다. 결과적으로 **OLM**은 **Operator group**의 **olm.providedAPIs** 주석을 수정하지 않으므로 사전에 설정할 수 있습니다. 이는 사용자가 **Operator group**을 사용하여 일련의 네임스페이스에서 리소스 경합을 방지하려고 하지만 해당 리소스에 대한 **API**를 제공하는 활성 멤버 **CSV**가 없는 경우 유용합니다.

다음은 **something.cool.io/cluster-monitoring: "true"** 주석을 사용하여 모든 네임스페이스에서 **Prometheus** 리소스를 보호하는 **Operator group**의 예입니다.

```
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,PrometheusR
ule.v1.monitoring.coreos.com,ServiceMonitor.v1.monitoring.coreos.com
spec:
  staticProvidedAPIs: true
  selector:
    matchLabels:
      something.cool.io/cluster-monitoring: "true"
```

주의

OLM(Operator Lifecycle Manager)은 각 Operator 그룹에 대해 다음 클러스터 역할을 생성합니다.

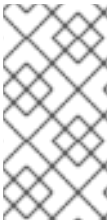
- <operatorgroup_name>-admin
- <operatorgroup_name>-edit
- <operatorgroup_name>-view

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 Operator 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

2.4.5.9. Operator group 교집합

대상 네임스페이스 세트의 교집합이 빈 세트가 아니고 olm.providedAPIs 주석으로 정의되어 제공된 API 세트의 교집합이 빈 세트가 아닌 경우 두 Operator groups을 *교차 제공 API*가 있다고 합니다.

잠재적인 문제는 교차 제공 API가 있는 Operator groups이 교차 네임스페이스 세트에서 동일한 리소스에 대해 경쟁할 수 있다는 점입니다.



참고

교집합 규칙을 확인할 때는 **Operator group** 네임스페이스가 항상 선택한 대상 네임스페이스의 일부로 포함됩니다.

교집합 규칙

활성 멤버 **CSV**가 동기화될 때마다 **OLM**은 **CSV**의 **Operator group** 및 기타 모든 그룹 간에 교차 제공되는 **API** 세트에 대한 클러스터를 쿼리합니다. 그런 다음 **OLM**은 해당 세트가 빈 세트인지 확인합니다.

- **true** 및 **CSV** 제공 **API**가 **Operator group**의 서브 세트인 경우:
  - 계속 전환합니다.
- **true** 및 **CSV** 제공 **API**가 **Operator group**의 서브 세트가 *아닌* 경우:
  - **Operator group**이 정적이면 다음을 수행합니다.
    - **CSV**에 속하는 배포를 정리합니다.
    - 상태 이유 **CannotModifyStaticOperatorGroupProvidedAPIs**와 함께 **CSV**를 실패 상태로 전환합니다.
  - **Operator group**이 정적이 *아니면* 다음을 수행합니다.
    - **Operator group**의 **olm.providedAPIs** 주석을 주석 자체와 **CSV** 제공 **API**를 결합한 내용으로 교체합니다.
- **false** 및 **CSV** 제공 **API**가 **Operator group**의 서브 세트가 *아닌* 경우:
  - **CSV**에 속하는 배포를 정리합니다.
  - 상태 이유 **InterOperatorGroupOwnerConflict**와 함께 **CSV**를 실패 상태로 전환합니

다.

- false 및 CSV 제공 API가 Operator group의 서브 세트인 경우:
  - Operator group이 정적이면 다음을 수행합니다.
    - CSV에 속하는 배포를 정리합니다.
    - 상태 이유 CannotModifyStaticOperatorGroupProvidedAPIs와 함께 CSV를 실패 상태로 전환합니다.
  - Operator group이 정적이 *아니면* 다음을 수행합니다.
    - Operator group의 olm.providedAPIs 주석을 주석 자체와 CSV 제공 API 간의 차이로 교체합니다.

참고

Operator group으로 인한 실패 상태는 터미널이 아닙니다.

Operator group이 동기화될 때마다 다음 작업이 수행됩니다.

- 활성 멤버 CSV에서 제공되는 API 세트는 클러스터에서 계산됩니다. CSV 복사본은 무시됩니다.
- 클러스터 세트는 olm.providedAPIs와 비교되며 olm.providedAPIs에 추가 API가 포함된 경우 해당 API가 정리됩니다.
- 모든 네임스페이스에서 동일한 API를 제공하는 모든 CSV가 다시 큐에 추가됩니다. 그러면 충돌하는 CSV의 크기 조정 또는 삭제를 통해 충돌이 해결되었을 수 있음을 교차 group의 충돌하는 CSV에 알립니다.

2.4.5.10. 다중 테넌트 Operator 관리에 대한 제한 사항

OpenShift Container Platform은 동일한 클러스터에 다른 버전의 Operator를 동시에 설치할 수 있도록 제한된 지원을 제공합니다. OLM(Operator Lifecycle Manager)은 다른 네임스페이스에 Operator를 여러 번 설치합니다. 이에 대한 한 가지 제한 사항은 Operator의 API 버전이 동일해야 한다는 것입니다.

Operator는 Kubernetes의 글로벌 리소스인 CRD(CustomResourceDefinition 오브젝트)를 사용하므로 컨트롤 플레인 확장입니다. Operator의 다른 주요 버전에는 호환되지 않는 CRD가 있는 경우가 많습니다. 이로 인해 클러스터의 다른 네임스페이스에 동시에 설치할 수 없습니다.

모든 테넌트 또는 네임스페이스는 클러스터의 동일한 컨트롤 플레인을 공유합니다. 따라서 다중 테넌트 클러스터의 테넌트도 글로벌 CRD를 공유하므로 동일한 Operator의 다른 인스턴스를 동일한 클러스터에서 병렬로 사용할 수 있는 시나리오를 제한합니다.

지원되는 시나리오에는 다음이 포함됩니다.

- 정확히 동일한 CRD 정의를 제공하는 다른 버전의 Operator (버전이 지정된 CRD의 경우 정확히 동일한 버전 세트)
- CRD를 제공하지 않고 OperatorHub의 별도의 번들에서 CRD를 사용할 수 있는 다른 버전의 Operator

동일한 클러스터에서 조정할 다른 Operator 버전의 여러 가지 CRD가 있는 경우 클러스터 데이터의 무결성을 보장할 수 없기 때문에 다른 모든 시나리오가 지원되지 않습니다.

추가 리소스

- OLM(Operator Lifecycle Manager) → 멀티 테넌시 및 Operator 공동 배치
- 다중 테넌트 클러스터의 Operator
- 비 클러스터 관리자가 Operator를 설치하도록 허용

2.4.5.11. Operator groups 문제 해결

멤버십

- 설치 계획의 네임스페이스에는 하나의 **Operator** 그룹만 포함되어야 합니다. 네임스페이스에서 **CSV(**클러스터 서비스 버전**)**를 생성하려고 할 때 설치 계획은 다음 시나리오에서 **Operator** 그룹이 유효하지 않은 것으로 간주합니다.
  - 설치 계획의 네임스페이스에 **Operator** 그룹이 없습니다.
  - 설치 계획의 네임스페이스에 여러 **Operator** 그룹이 있습니다.
  - **Operator** 그룹에 잘못된 서비스 계정 이름이 지정되어 있습니다.

  설치 계획이 유효하지 않은 **Operator** 그룹이 발생하면 **CSV**가 생성되지 않고 **InstallPlan** 리소스가 관련 메시지와 함께 계속 설치됩니다. 예를 들어 동일한 네임스페이스에 둘 이상의 **Operator** 그룹이 있는 경우 다음 메시지가 제공됩니다.

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

  여기서 **count=** 은 네임스페이스에서 **Operator** 그룹 수를 지정합니다.

- **CSV**의 설치 모드가 해당 네임스페이스에서 **Operator group**의 대상 네임스페이스 선택을 지원하지 않는 경우 **CSV**는 **UnsupportedOperatorGroup** 이유와 함께 실패 상태로 전환됩니다. 이러한 이유로 실패 상태에 있는 **CSV**는 **Operator group**의 대상 네임스페이스 선택이 지원되는 구성으로 변경된 후 보류 중으로 전환되거나 대상 네임스페이스 선택을 지원하도록 **CSV**의 설치 모드가 수정됩니다.

2.4.6. 멀티 테넌시 및 Operator 공동 배치

이 가이드에서는 **OLM(Operator Lifecycle Manager)**의 멀티 테넌시 및 **Operator** 공동 배치에 대해 간단히 설명합니다.

2.4.6.1. 네임스페이스에 Operator 공동 배치

OLM(Operator Lifecycle Manager)은 동일한 네임스페이스에 설치된 **OLM** 관리 **Operator**를 처리합니다. 즉 서브스크립션 리소스는 관련 **Operator**와 동일한 네임스페이스에 배치됩니다. **OLM**은 실제로 관련이 없는 경우에도 버전 및 업데이트 정책과 같은 상태를 업데이트할 때 해당 상태를 고려합니다.

이 기본 동작 매니페스트는 다음 두 가지 방법으로 수행됩니다.

- 보류 중인 업데이트의 **InstallPlan** 리소스에는 동일한 네임스페이스에 있는 다른 모든 **Operator**의 **CSV(ClusterServiceVersion)** 리소스가 포함됩니다.

- 동일한 네임스페이스에 있는 모든 **Operator**는 동일한 업데이트 정책을 공유합니다. 예를 들어 한 **Operator**가 수동 업데이트로 설정된 경우 기타 모든 **Operator**의 업데이트 정책도 **manual**로 설정됩니다.

이러한 시나리오에서는 다음과 같은 문제가 발생할 수 있습니다.

- 업데이트된 **Operator**보다 더 많은 리소스가 정의되어 있기 때문에 **Operator** 업데이트의 설치 계획에 대해 추론하기가 어렵습니다.

- 다른 **Operator**가 수동으로 업데이트되는 동안 네임스페이스의 일부 **Operator**를 자동으로 설정할 수 없습니다. 이는 클러스터 관리자에게 일반적인 요구 사항입니다.

이러한 문제는 **OpenShift Container Platform** 웹 콘솔을 사용하여 **Operator**를 설치할 때 모든 네임스페이스 설치 모드를 지원하는 **Operator**를 기본 **openshift-operators** 글로벌 네임스페이스에 설치하므로 일반적으로 표시됩니다.

클러스터 관리자는 다음 워크플로우를 사용하여 이 기본 동작을 수동으로 바이패스할 수 있습니다.

1. **Operator** 설치를 위한 네임스페이스를 생성합니다.

2. 모든 네임스페이스를 *감시하는* ***Operator group****인 사용자 정의 글로벌* **Operator group**을 생성합니다. 이 **Operator group**을 방금 생성한 네임스페이스와 연결하여 설치 네임스페이스를 글로벌 네임스페이스로 만들어 모든 네임스페이스에서 **Operator**를 사용할 수 있습니다.

3. 설치 네임스페이스에 원하는 **Operator**를 설치합니다.

Operator에 종속 항목이 있는 경우 사전 생성된 네임스페이스에 종속 항목이 자동으로 설치됩니다. 결과적으로 종속성 **Operator**에 동일한 업데이트 정책 및 공유 설치 계획이 있는 것이 유효합니다. 자세한

절차는 "사용자 정의 네임스페이스에 글로벌 Operator 설치"를 참조하십시오.

추가 리소스

- 사용자 지정 네임스페이스에 글로벌 Operator 설치
- 다중 테넌트 클러스터의 Operator

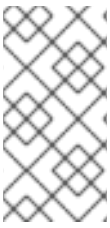
2.4.7. Operator 상태

이 가이드에서는 OLM(Operator Lifecycle Manager)에서 Operator 조건을 사용하는 방법을 간단히 설명합니다.

2.4.7.1. Operator 조건 정보

OLM(Operator Lifecycle Manager)에서는 Operator의 라이프사이클을 관리하는 역할의 일부로, Operator를 정의하는 Kubernetes 리소스의 상태에서 Operator의 상태를 유추합니다. 이 접근 방식에서는 Operator가 지정된 상태에 있도록 어느 정도는 보장하지만 Operator에서 다른 방법으로는 유추할 수 없는 정보를 OLM에 보고해야 하는 경우가 많습니다. 그러면 OLM에서 이러한 정보를 사용하여 Operator의 라이프사이클을 더 효과적으로 관리할 수 있습니다.

OLM에서는 Operator에서 OLM에 조건을 보고할 수 있는 OperatorCondition이라는 CRD(사용자 정의 리소스 정의)를 제공합니다. OperatorCondition 리소스의 Spec.Conditions 어레이에 있는 경우 OLM의 Operator 관리에 영향을 줄 수 있는 일련의 조건이 지원됩니다.



참고

기본적으로 spec .Conditions 배열은 사용자가 추가하거나 사용자 정의 Operator 논리의 결과로 OperatorCondition 오브젝트에 존재하지 않습니다.

2.4.7.2. 지원되는 조건

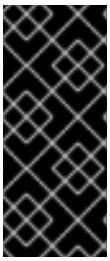
OLM(Operator Lifecycle Manager)에서는 다음과 같은 Operator 조건을 지원합니다.

2.4.7.2.1. 업그레이드 가능한 조건

Upgradeable Operator 조건을 사용하면 기존 CSV(클러스터 서비스 버전)가 최신 버전의 CSV로 교체되지 않습니다. 이 조건은 다음과 같은 경우 유용합니다.

- Operator에서 중요한 프로세스를 시작할 예정이며 프로세스가 완료될 때까지 업그레이드해서는 안 됩니다.

- Operator에서 Operator를 업그레이드하기 위해 준비하기 전에 완료해야 하는 CR(사용자 정의 리소스) 마이그레이션을 수행하고 있습니다.



중요

Upgradeable Operator 조건을 False 값으로 설정하면 Pod가 중단되지 않습니다. Pod가 중단되지 않도록 해야 하는 경우 "Pod 중단 예산을 사용하여 가동해야 하는 Pod 수" 및 "추가 리소스" 섹션의 "Graceful termination"를 참조하십시오.

Upgradeable Operator 조건의 예

```
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) 조건의 이름입니다.

(2) False 값은 Operator를 업그레이드할 준비가 되지 않았음을 나타냅니다. OLM에서는 Operator의 기존 CSV를 대체하는 CSV가 Pending 상태가 되지 않도록 합니다. False 값은 클러스터 업그레이드를 차단하지 않습니다.

2.4.7.3. 추가 리소스

- Operator 조건 관리
- Operator 조건 활성화
- Pod 중단 예산을 사용하여 가동해야 하는 Pod 수 지정
- 정상적인 종료

2.4.8. Operator Lifecycle Manager 지표

2.4.8.1. 표시되는 지표

OLM(Operator Lifecycle Manager)에서는 Prometheus 기반 OpenShift Container Platform 클러스터 모니터링 스택에서 사용할 특정 OLM 관련 리소스를 표시합니다.

표 2.7. OLM에서 표시하는 지표

이름	설명
catalog_source_count	카탈로그 소스 수입니다.
catalogsource_ready	카탈로그 소스의 상태입니다. 값 **1** 은 카탈로그 소스가 **READY** 상태에 있음을 나타냅니다. 값 **0** 은 카탈로그 소스가 **READY** 상태가 아님을 나타냅니다.
csv_abnormal	CSV(클러스터 서비스 버전)를 조정할 때 CSV 버전이 **Succeeded** 이외의 상태가 될 때마다(예: 설치되지 않은 경우) 표시됩니다. **name**, **namespace**, **phase**, **reason**, **version** 라벨이 포함됩니다. 이 지표가 표시되면 Prometheus 경고가 생성됩니다.
csv_count	성공적으로 등록된 CSV 수입니다.
csv_succeeded	CSV를 재조정할 때 CSV 버전이 **Succeeded** 상태(값 **1**)인지 아닌지(값 **0**)를 나타냅니다. **name**, **namespace**, **version** 라벨이 포함됩니다.
csv_upgrade_count	CSV 업그레이드의 단조 수입니다.
install_plan_count	설치 계획 수입니다.

이름	설명
installplan_warnings_total	더 이상 사용되지 않는 리소스와 같이 설치 계획에 포함된 리소스에서 생성한 경고의 단조 수입니다.
olm_resolution_duration_seconds	종속성 확인 시도의 기간입니다.
subscription_count	서브스크립션 수입니다.
subscription_sync_total	서브스크립션 동기화의 단조 수입니다. **channel**, **installed** CSV, 서브스크립션 **name** 라벨이 포함됩니다.

2.4.9. Operator Lifecycle Manager의 Webhook 관리

Operator 작성자는 Webhook를 통해 리소스를 오브젝트 저장소에 저장하고 Operator 컨트롤러에서 이를 처리하기 전에 리소스를 가로채기, 수정, 수락 또는 거부할 수 있습니다. Operator와 함께 webhook가 제공 될 때 OLM (Operator Lifecycle Manager)은 이러한 Webhook의 라이프 사이클을 관리할 수 있습니다.

Operator 개발자가 Operator에 대한 Webhook 를 정의하는 방법과 OLM에서 실행할 때의 고려 사항에 대한 자세한 내용은 CSV(클러스터 서비스 버전) 정의를 참조하십시오.

2.4.9.1. 추가 리소스

- 웹 후크 승인 플러그인의 유형
- **Kubernetes 설명서:**
  - 승인 Webhook 검증
  - 변경 승인 Webhook
  - 변환 Webhook

2.5. OPERATORHUB 이해

2.5.1. OperatorHub 정보

OperatorHub는 클러스터 관리자가 **Operator**를 검색하고 설치하는 데 사용하는 **OpenShift Container Platform**의 웹 콘솔 인터페이스입니다. 한 번의 클릭으로 **Operator**를 클러스터 외부 소스에서 가져와서 클러스터에 설치 및 구독하고 엔지니어링 팀에서 **OLM(Operator Lifecycle Manager)**을 사용하여 배포 환경에서 제품을 셀프서비스로 관리할 수 있습니다.

클러스터 관리자는 다음 카테고리로 그룹화된 카탈로그에서 선택할 수 있습니다.

카테고리	설명
Red Hat Operator	Red Hat에서 Red Hat 제품을 패키지 및 제공합니다. Red Hat에서 지원합니다.
인증된 Operator	선도적인 ISV(독립 소프트웨어 벤더)의 제품입니다. Red Hat은 패키지 및 제공을 위해 ISV와 협력합니다. ISV에서 지원합니다.
Red Hat Marketplace	Red Hat Marketplace에서 구매할 수 있는 인증 소프트웨어입니다.
커뮤니티 Operator	redhat-openshift-ecosystem/community-operators-prod/operators GitHub 리포지토리의 관련 담당자가 유지 관리하는 선택적 보기 소프트웨어입니다. 공식적으로 지원되지 않습니다.
사용자 정의 Operator	클러스터에 직접 추가하는 Operator입니다. 사용자 정의 Operator를 추가하지 않은 경우 OperatorHub의 웹 콘솔에 **사용자 정의** 카테고리가 표시되지 않습니다.

OperatorHub의 **Operator**는 **OLM**에서 실행되도록 패키지됩니다. 여기에는 **Operator**를 설치하고 안전하게 실행하는 데 필요한 모든 **CRD**, **RBAC** 규칙, 배포, 컨테이너 이미지가 포함된 **CSV**(클러스터 서비스 버전)라는 **YAML** 파일이 포함됩니다. 또한 해당 기능 및 지원되는 **Kubernetes** 버전에 대한 설명과 같이 사용자가 볼 수 있는 정보가 포함됩니다.

Operator SDK는 **OLM** 및 **OperatorHub**에서 사용하도록 **Operator**를 패키지하는 개발자를 지원하는 데 사용할 수 있습니다. 고객이 액세스할 수 있도록 설정할 상용 애플리케이션이 있는 경우 **Red Hat Partner Connect** 포털(**connect.redhat.com**)에 제공된 인증 워크플로를 사용하여 포함합니다.

2.5.2. OperatorHub 아키텍처

OperatorHub UI 구성 요소는 기본적으로 **openshift-marketplace** 네임스페이스의 **OpenShift Container Platform**에서 **Marketplace Operator**에 의해 구동됩니다.

2.5.2.1. OperatorHub 사용자 정의 리소스

Marketplace Operator는 OperatorHub와 함께 제공되는 기본 CatalogSource 오브젝트를 관리하는 cluster라는 OperatorHub CR(사용자 정의 리소스)을 관리합니다. 이 리소스를 수정하여 기본 카탈로그를 활성화하거나 비활성화할 수 있어 제한된 네트워크 환경에서 OpenShift Container Platform을 구성할 때 유용합니다.

OperatorHub 사용자 정의 리소스의 예

```
apiVersion: config.openshift.io/v1
kind: OperatorHub
metadata:
  name: cluster
spec:
  disableAllDefaultSources: true (1)
  sources: [ (2)
    {
      name: "community-operators",
      disabled: false
    }
  ]
```

(1) disableAllDefaultSources는 OpenShift Container Platform 설치 중 기본적으로 구성되는 모든 기본 카탈로그의 가용성을 제어하는 덮어쓰기입니다.

(2) 소스에 따라 disabled 매개변수 값을 변경하여 기본 카탈로그를 개별적으로 비활성화합니다.

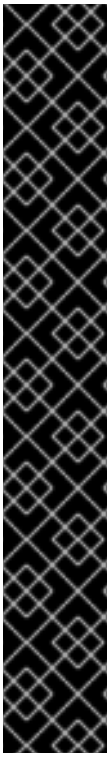
2.5.3. 추가 리소스

- 카탈로그 소스
- Operator SDK 정보
- CSV(클러스터 서비스 버전) 정의

- OLM의 Operator 설치 및 업그레이드 워크플로
- Red Hat Partner Connect
- Red Hat Marketplace

2.6. RED HAT 제공 OPERATOR 카탈로그

Red Hat은 기본적으로 OpenShift Container Platform에 포함된 여러 Operator 카탈로그를 제공합니다.



중요

OpenShift Container Platform 4.11부터 기본 Red Hat 제공 Operator 카탈로그는 파일 기반 카탈로그 형식으로 제공됩니다. 더 이상 사용되지 않는 SQLite 데이터베이스 형식으로 릴리스된 4.10을 통한 OpenShift Container Platform 4.6의 기본 Red Hat 제공 Operator 카탈로그입니다.

SQLite 데이터베이스 형식과 관련된 opm 하위 명령, 플래그 및 기능은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 기능은 계속 지원되며 더 이상 사용되지 않는 SQLite 데이터베이스 형식을 사용하는 카탈로그에 사용해야 합니다.

opm index prune 와 같은 SQLite 데이터베이스 형식을 사용하기 위한 많은 opm 하위 명령과 플래그는 파일 기반 카탈로그 형식으로 작동하지 않습니다. 파일 기반 카탈로그 사용에 대한 자세한 내용은 사용자 정의 카탈로그 관리,Operator Framework 패키징 형식, oc-mirror 플러그인을 사용하여 연결이 끊긴 설치용 이미지 미러링을 참조하십시오.

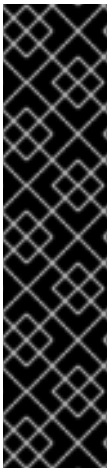
2.6.1. Operator 카탈로그 정보

Operator 카탈로그는 OLM(Operator Lifecycle Manager)에서 쿼리하여 Operator 및 해당 종속성을 검색하고 설치할 수 있는 메타데이터 리포지토리입니다. OLM은 항상 최신 버전의 카탈로그에 있는 Operator를 설치합니다.

Operator 번들 형식을 기반으로 하는 인덱스 이미지는 컨테이너화된 카탈로그 스냅샷입니다. 일련의 Operator 매니페스트 콘텐츠에 대한 포인터의 데이터베이스를 포함하는 변경 불가능한 아티팩트입니다. 카탈로그는 인덱스 이미지를 참조하여 클러스터에서 OLM에 대한 콘텐츠를 소싱할 수 있습니다.

카탈로그가 업데이트되면 최신 버전의 **Operator**가 변경되고 이전 버전은 제거되거나 변경될 수 있습니다. 또한 **OLM**이 네트워크가 제한된 환경의 **OpenShift Container Platform** 클러스터에서 실행되면 최신 콘텐츠를 가져오기 위해 인터넷에서 카탈로그에 직접 액세스할 수 없습니다.

클러스터 관리자는 **Red Hat**에서 제공하는 카탈로그를 기반으로 또는 처음부터 자체 사용자 정의 인덱스 이미지를 생성할 수 있습니다. 이 이미지는 클러스터에서 카탈로그 콘텐츠를 소싱하는 데 사용할 수 있습니다. 자체 인덱스 이미지를 생성하고 업데이트하면 클러스터에서 사용 가능한 **Operator** 세트를 사용자 정의할 수 있을 뿐만 아니라 앞서 언급한 제한된 네트워크 환경 문제도 방지할 수 있습니다.



중요

Kubernetes는 후속 릴리스에서 제거된 특정 **API**를 주기적으로 사용하지 않습니다. 결과적으로 **Operator**는 **API**를 제거한 **Kubernetes** 버전을 사용하는 **OpenShift Container Platform** 버전에서 시작하여 제거된 **API**를 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** 버전과의 **Operator** 호환성 제어를 참조하십시오.

참고

OpenShift Container Platform 4.8 이상에서는 레거시 형식을 사용하는 사용자 지정 카탈로그를 포함하여 **Operators**에 대한 레거시 *패키지 매니페스트 형식*에 대한 지원이 제거되었습니다.

사용자 정의 카탈로그 이미지를 생성할 때 이전 버전의 **OpenShift Container Platform 4**에서는 **oc adm catalog build** 명령을 사용해야 했습니다. 이 명령은 여러 릴리스에서 더 이상 사용되지 않으며 지금은 제거되었습니다. **OpenShift Container Platform 4.6**부터 **Red Hat** 제공 인덱스 이미지를 사용할 수 있으므로 카탈로그 빌더는 **opm index** 명령을 사용하여 인덱스 이미지를 관리해야 합니다.

추가 리소스

- 사용자 정의 카탈로그 관리
- 패키징 형식

- 제한된 네트워크에서 Operator Lifecycle Manager 사용

2.6.2. Red Hat 제공 Operator 카탈로그 정보

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

다음은 Red Hat에서 제공하는 Operator 카탈로그입니다.

카탈로그	인덱스 이미지	설명
redhat-operators	**registry.redhat.io/redhat/redhat-operator-index:v4.11**	Red Hat에서 Red Hat 제품을 패키지 및 제공합니다. Red Hat에서 지원합니다.
certified-operators	**registry.redhat.io/redhat/certified-operator-index:v4.11**	선도적인 ISV(독립 소프트웨어 벤더)의 제품입니다. Red Hat은 패키지 및 제공을 위해 ISV와 협력합니다. ISV에서 지원합니다.
redhat-marketplace	**registry.redhat.io/redhat/redhat-marketplace-index:v4.11**	Red Hat Marketplace에서 구매할 수 있는 인증 소프트웨어입니다.
community-operators	**registry.redhat.io/redhat/community-operator-index:v4.11**	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 4.8에서 4.9로 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
```

2.7. 다중 테넌트 클러스터의 OPERATOR

OLM(Operator Lifecycle Manager)의 기본 동작은 Operator 설치 중에 단순화를 제공하는 것입니다. 그러나 이러한 동작으로 인해 특히 다중 테넌트 클러스터에서 유연성이 부족할 수 있습니다. OpenShift Container Platform 클러스터의 여러 테넌트가 Operator를 사용하려면 OLM의 기본 동작에서 관리자가 최소 권한 원칙을 위반하는 것으로 간주되는 모든 네임스페이스 모드로 Operator를 설치해야 합니다.

환경 및 요구 사항에 가장 적합한 Operator 설치 워크플로우를 확인하려면 다음 시나리오를 고려하십시오.

추가 리소스

- 일반 용어: 다중 테넌트
- 다중 테넌트 Operator 관리에 대한 제한 사항

2.7.1. 기본 Operator 설치 모드 및 동작

웹 콘솔을 사용하여 Operator를 관리자로 설치하는 경우 일반적으로 Operator의 기능에 따라 설치 모드에 대한 두 가지 선택 사항이 있습니다.

단일 네임스페이스

선택한 단일 네임스페이스에 Operator를 설치하고 해당 네임스페이스에서 Operator를 사용할 수 있는 모든 권한을 만듭니다.

모든 네임스페이스

클러스터의 모든 네임스페이스를 감시하고 사용할 수 있도록 기본 openshift-operators 네임스페이스에 Operator를 설치합니다. Operator에서 요청하는 모든 권한을 모든 네임스페이스에서 사용할 수 있도록 합니다. Operator 작성자는 메타데이터를 정의하여 해당 Operator의 제안된 네임스페이스에 대한 두 번째 옵션을 사용자에게 제공할 수 있습니다.

또한 영향을 받는 네임스페이스의 사용자는 네임스페이스의 역할에 따라 사용자 정의 리소스(CR)를 활용할 수 있는 Operator API에 액세스할 수 있습니다.

- **namespace-admin** 및 **namespace-edit** 역할은 **Operator API**를 읽고 쓸 수 있으므로 사용할 수 있습니다.

- **namespace-view** 역할은 해당 **Operator**의 **CR** 오브젝트를 읽을 수 있습니다.

단일 네임스페이스 모드의 경우 **Operator**가 선택한 네임스페이스에 설치되므로 해당 **Pod** 및 서비스 계정도 있습니다. 모든 네임스페이스 모드의 경우 **Operator**의 권한은 모두 클러스터 역할로 자동으로 향상됩니다. 즉 **Operator**는 모든 네임스페이스에 해당 권한이 있습니다.

추가 리소스

- 클러스터에 Operator 추가

- 설치 모드 유형

- 제안된 네임스페이스 설정

2.7.2. 다중 테넌트 클러스터에 권장되는 솔루션

Multinamespace 설치 모드가 존재하지만 **Operator**는 매우 적은 수의 **Operator**에서 지원합니다. 표준 모든 네임스페이스와 단일 네임스페이스 설치 모드 간의 중간 솔루션이므로 다음 워크플로우를 사용하여 각 테넌트에 대해 동일한 **Operator**의 여러 인스턴스를 설치할 수 있습니다.

1. 테넌트의 네임스페이스와 별도로 테넌트 **Operator**의 네임스페이스를 생성합니다.

2. 테넌트의 네임스페이스에 대해서만 테넌트 **Operator** 범위에 대한 **Operator group**을 생성합니다.

3. 테넌트 **Operator** 네임스페이스에 **Operator**를 설치합니다.

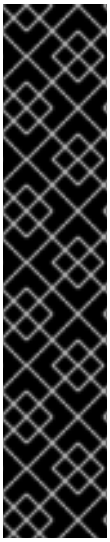
결과적으로 **Operator**는 테넌트 **Operator** 네임스페이스에 상주하고 테넌트 네임스페이스를 감시하지만 테넌트에서 **Operator**의 **Pod**와 서비스 계정을 보거나 사용할 수 없습니다.

이 솔루션은 테넌트 분리, 리소스 사용 비용의 권한 이상의 원칙, 제약 조건 충족을 위한 추가 오케스트레이션을 제공합니다. 자세한 절차는 " 다중 테넌트 클러스터를 위한 Operator의 여러 인스턴스 준비"에서 참조하십시오.

제한 사항 및 고려 사항

이 솔루션은 다음 제약 조건을 충족하는 경우에만 작동합니다.

- 동일한 Operator의 모든 인스턴스는 동일한 버전이어야 합니다.
- Operator는 다른 Operator에 종속될 수 없습니다.
- Operator는 CRD 변환 Webhook를 제공할 수 없습니다.



중요

동일한 클러스터에서 다른 버전의 동일한 Operator를 사용할 수 없습니다. 결국 다음 조건을 충족하는 경우 Operator의 다른 인스턴스 설치가 차단되었습니다.

- 인스턴스는 최신 버전의 Operator가 아닙니다.
- 인스턴스는 클러스터에서 이미 사용 중인 최신 버전에 있는 정보 또는 버전이 없는 이전 버전의 CRD를 제공합니다.

주의

관리자는 "클러스터 이외의 관리자가 Operator를 설치할 수 있도록 허용"에 설명된 대로 클러스터 이외의 관리자가 Operator를 직접 설치할 수 있도록 허용할 때 주의해야 합니다. 이러한 테넌트는 종속 항목이 없는 것으로 알려진 Operator의 선별된 카탈로그에만 액세스할 수 있어야 합니다. CRD가 변경되지 않도록 이러한 테넌트도 Operator의 동일한 버전 라인을 사용해야 합니다. 이를 위해서는 네임스페이스 범위 카탈로그를 사용해야 하며 글로벌 기본 카탈로그가 비활성화될 수 있습니다.

추가 리소스

- 다중 테넌트 클러스터에 대한 Operator의 여러 인스턴스 준비
- 비 클러스터 관리자가 Operator를 설치하도록 허용
- 기본 OperatorHub 카탈로그 소스 비활성화

2.7.3. Operator colocation 및 Operator groups

OLM(Operator Lifecycle Manager)은 동일한 네임스페이스에 설치된 OLM 관리 Operator를 처리합니다. 즉 서브스크립션 리소스는 관련 Operator와 동일한 네임스페이스에 배치됩니다. OLM은 실제로 관련이 없는 경우에도 버전 및 업데이트 정책과 같은 상태를 업데이트할 때 해당 상태를 고려합니다.

Operator 공동 배치 및 Operator 그룹을 효과적으로 사용하는 방법에 대한 자세한 내용은 OLM(Operator Lifecycle Manager) → 멀티 테넌시 및 Operator 공동 배치를 참조하십시오.

2.8. CRD

2.8.1. 사용자 정의 리소스 정의를 사용하여 Kubernetes API 확장

Operator는 Kubernetes 확장 메커니즘인 CRD(사용자 정의 리소스 정의)를 사용하므로 Operator에서 관리하는 사용자 정의 오브젝트가 네이티브 Kubernetes 오브젝트처럼 보이고 작동합니다. 이 가이드에서는 클러스터 관리자가 CRD를 생성하고 관리하여 OpenShift Container Platform 클러스터를 확장할 수 있는 방법을 설명합니다.

2.8.1.1. 사용자 정의 리소스 정의

Kubernetes API에서 *리소스*는 특정 종류의 API 오브젝트 컬렉션을 저장하는 끝점입니다. 예를 들어 기본 제공 Pod 리소스에는 Pod 오브젝트의 컬렉션이 포함됩니다.

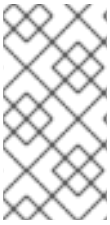
CRD(*사용자 정의 리소스 정의*) 오브젝트는 클러스터에서 *종류*라는 새로운 고유한 오브젝트 유형을 정의하고 Kubernetes API 서버에서 전체 라이프사이클을 처리하도록 합니다.

CR(*사용자 정의 리소스*) 오브젝트는 클러스터 관리자가 클러스터에 추가한 CRD에서 생성하므로 모든 클러스터 사용자가 새 리소스 유형을 프로젝트에 추가할 수 있습니다.

클러스터 관리자가 새 CRD를 클러스터에 추가하면 Kubernetes API 서버는 전체 클러스터 또는 단일 프로젝트(네임스페이스)에서 액세스할 수 있는 새 RESTful 리소스 경로를 생성하여 반응하고 지정된 CR을 제공하기 시작합니다.

클러스터 관리자가 다른 사용자에게 CRD에 대한 액세스 권한을 부여하려면 클러스터 역할 집계를 사용하여 admin, edit 또는 view 기본 클러스터 역할이 있는 사용자에게 액세스 권한을 부여할 수 있습니다. 클러스터 역할 집계를 사용하면 이러한 클러스터 역할에 사용자 정의 정책 규칙을 삽입할 수 있습니다. 이 동작은 새 리소스를 기본 제공 리소스인 것처럼 클러스터의 RBAC 정책에 통합합니다.

특히 운영자는 CRD를 필수 RBAC 정책 및 기타 소프트웨어별 논리와 함께 패키지로 제공하는 방식으로 CRD를 사용합니다. 또한 클러스터 관리자는 Operator의 라이프사이클 외부에서 클러스터에 CRD를 수동으로 추가하여 모든 사용자에게 제공할 수 있습니다.



참고

클러스터 관리자만 CRD를 생성할 수 있지만 기존 CRD에 대한 읽기 및 쓰기 권한이 있는 개발자의 경우 기존 CRD에서 CR을 생성할 수 있습니다.

2.8.1.2. 사용자 정의 리소스 정의 생성

사용자 정의 리소스(CR) 오브젝트를 생성하려면 클러스터 관리자가 먼저 CRD(사용자 정의 리소스 정의)를 생성해야 합니다.

사전 요구 사항

- cluster-admin 사용자 권한을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

프로세스

CRD를 생성하려면 다음을 수행합니다.

1. 다음 필드 유형을 포함하는 YAML 파일을 생성합니다.

CRD에 대한 YAML 파일의 예

```
apiVersion: apiextensions.k8s.io/v1 1
kind: CustomResourceDefinition
```

```
metadata:
  name: crontabs.stable.example.com (2)
spec:
  group: stable.example.com (3)
  versions:
    name: v1 (4)
  scope: Namespaced (5)
  names:
    plural: crontabs (6)
    singular: crontab (7)
    kind: CronTab (8)
    shortNames:
    - ct (9)
```

(1) apiextensions.k8s.io/v1 API를 사용합니다.

(2) 정의의 이름을 지정합니다. group 및 plural 필드의 값을 사용하는 <plural-name>.<group> 형식이어야 합니다.

(3) API의 그룹 이름을 지정합니다. API 그룹은 논리적으로 관련된 오브젝트의 컬렉션입니다. 예를 들어 Job 또는 ScheduledJob과 같은 배치 오브젝트는 모두 배치 API 그룹(예: batch.api.example.com)에 있을 수 있습니다. 조직의 FQDN(정규화된 도메인 이름)을 사용하는 것이 좋습니다.

(4) URL에 사용할 버전 이름을 지정합니다. 각 API 그룹은 여러 버전(예: v1alpha, v1beta, v1)에 있을 수 있습니다.

(5) 특정 프로젝트(Namespaced) 또는 클러스터의 모든 프로젝트(Cluster)에서 사용자 정의 오브젝트를 사용할 수 있는지 지정합니다.

(6) URL에서 사용할 복수형 이름을 지정합니다. plural 필드는 API URL의 리소스와 동일합니다.

7

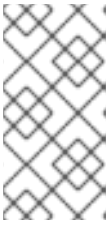
CLI 및 디스플레이에서 별칭으로 사용할 단수형 이름을 지정합니다.

8

생성할 수 있는 오브젝트의 종류를 지정합니다. 유형은 CamelCase에 있을 수 있습니다.

9

CLI의 리소스와 일치하는 짧은 문자열을 지정합니다.



참고

기본적으로 CRD는 클러스터 범위이며 모든 프로젝트에서 사용할 수 있습니다.

2. CRD 오브젝트를 생성합니다.

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

다음에 새 RESTful API 끝점이 생성됩니다.

```
/apis/<spec:group>/<spec:version>/<scope>/*/<names-plural>/...
```

예를 들어 예제 파일을 사용하면 다음 끝점이 생성됩니다.

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

이 끝점 URL을 사용하여 CR을 생성하고 관리할 수 있습니다. 오브젝트 종류는 생성한 CRD 오브젝트의 spec.kind 필드를 기반으로 합니다.

2.8.1.3. 사용자 정의 리소스 정의에 대한 클러스터 역할 생성

클러스터 관리자는 기존 클러스터 범위의 CRD(사용자 정의 리소스 정의)에 권한을 부여할 수 있습니다. admin, edit, view 기본 클러스터 역할을 사용하는 경우 해당 규칙에 클러스터 역할 집계를 활용할 수 있습니다.

중요

이러한 각 역할에 대한 권한을 명시적으로 할당해야 합니다. 권한이 더 많은 역할은 권한이 더 적은 역할의 규칙을 상속하지 않습니다. 역할에 규칙을 할당하는 경우 권한이 더 많은 역할에 해당 동사를 할당해야 합니다. 예를 들어 보기 역할에 **get crontabs** 권한을 부여하는 경우 **edit** 및 **admin** 역할에도 부여해야 합니다. **admin** 또는 **edit** 역할은 일반적으로 프로젝트 템플릿을 통해 프로젝트를 생성한 사용자에게 할당됩니다.

사전 요구 사항

- **CRD**를 생성합니다.

프로세스

1. **CRD**의 클러스터 역할 정의 파일을 생성합니다. 클러스터 역할 정의는 각 클러스터 역할에 적용되는 규칙이 포함된 **YAML** 파일입니다. **OpenShift Container Platform** 컨트롤러는 사용자가 지정하는 규칙을 기본 클러스터 역할에 추가합니다.

클러스터 역할 정의에 대한 **YAML** 파일의 예

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1 (1)
metadata:
  name: aggregate-cron-tabs-admin-edit (2)
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin: "true" (3)
    rbac.authorization.k8s.io/aggregate-to-edit: "true" (4)
rules:
- apiGroups: ["stable.example.com"] (5)
  resources: ["crontabs"] (6)
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete", "deletecollection"]
(7)
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: aggregate-cron-tabs-view (8)
  labels:
    # Add these permissions to the "view" default role.
    rbac.authorization.k8s.io/aggregate-to-view: "true" (9)
    rbac.authorization.k8s.io/aggregate-to-cluster-reader: "true" (10)
rules:
- apiGroups: ["stable.example.com"] (11)
  resources: ["crontabs"] (12)
  verbs: ["get", "list", "watch"] (13)
```

1

rbac.authorization.k8s.io/v1 API를 사용합니다.

2 8

정의의 이름을 지정합니다.

3

관리 기본 역할에 권한을 부여하려면 이 라벨을 지정합니다.

4

편집 기본 역할에 권한을 부여하려면 이 라벨을 지정합니다.

5 11

CRD의 그룹 이름을 지정합니다.

6 12

이러한 규칙이 적용되는 **CRD**의 복수형 이름을 지정합니다.

7 13

역할에 부여된 권한을 나타내는 동사를 지정합니다. 예를 들어 **admin** 및 **edit** 역할에는 읽기 및 쓰기 권한을 적용하고 **view** 역할에는 읽기 권한만 적용합니다.

9

view 기본 역할에 권한을 부여하려면 이 라벨을 지정합니다.

10

cluster-reader 기본 역할에 권한을 부여하려면 이 라벨을 지정합니다.

2. 클러스터 역할을 생성합니다.

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

2.8.1.4. 파일에서 사용자 정의 리소스 생성

CRD(사용자 정의 리소스 정의)가 클러스터에 추가되면 CR 사양을 사용하여 파일에서 CLI를 통해 CR(사용자 정의 리소스)을 생성할 수 있습니다.

사전 요구 사항

- 클러스터 관리자가 클러스터에 CRD를 추가했습니다.

프로세스

1. CR에 대한 YAML 파일을 생성합니다. 다음 예제 정의에서 cronSpec 및 image 사용자 정의 필드는 Kind: CronTab의 CR에 설정됩니다. Kind는 CRD 오브젝트의 spec.kind 필드에서 제공합니다.

CR에 대한 YAML 파일의 예

```
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. CRD에서 그룹 이름 및 API 버전(이름/버전)을 지정합니다.
2. CRD에 유형을 지정합니다.
3. 오브젝트의 이름을 지정합니다.

4

해당하는 경우 오브젝트의 종료자를 지정합니다. 종료자를 사용하면 컨트롤러에서 오브젝트를 삭제하기 전에 완료해야 하는 조건을 구현할 수 있습니다.

5

오브젝트 유형별 조건을 지정합니다.

2. 파일을 생성한 후 오브젝트를 생성합니다.

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

2.8.1.5. 사용자 정의 리소스 검사

CLI를 사용하여 클러스터에 존재하는 CR(사용자 정의 리소스) 오브젝트를 검사할 수 있습니다.

사전 요구 사항

- CR 오브젝트는 액세스할 수 있는 네임스페이스에 있습니다.

프로세스

1. 특정 종류의 CR에 대한 정보를 얻으려면 다음을 실행합니다.

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

예를 들면 다음과 같습니다.

```
$ oc get crontab
```

출력 예

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

리소스 이름은 대소문자를 구분하지 않으며 CRD에 정의된 단수형 또는 복수형 양식이나 짧은 이름을 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. CR의 원시 YAML 데이터를 볼 수도 있습니다.

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

예를 들면 다음과 같습니다.

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

출력 예

```
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

오브젝트를 생성하는 데 사용한 **YAML**의 사용자 정의 데이터가 표시됩니다.

2.8.2. 사용자 정의 리소스 정의에서 리소스 관리

이 가이드에서는 개발자가 **CRD**(사용자 정의 리소스 정의)에서 제공하는 **CR**(사용자 정의 리소스)을 관리하는 방법을 설명합니다.

2.8.2.1. 사용자 정의 리소스 정의

Kubernetes API에서 *리소스*는 특정 종류의 **API** 오브젝트 컬렉션을 저장하는 끝점입니다. 예를 들어 기본 제공 **Pod** 리소스에는 **Pod** 오브젝트의 컬렉션이 포함됩니다.

CRD(*사용자 정의 리소스 정의*) 오브젝트는 클러스터에서 *종류*라는 새로운 고유한 오브젝트 유형을 정의하고 **Kubernetes API** 서버에서 전체 라이프사이클을 처리하도록 합니다.

CR(*사용자 정의 리소스*) 오브젝트는 클러스터 관리자가 클러스터에 추가한 **CRD**에서 생성하므로 모든 클러스터 사용자가 새 리소스 유형을 프로젝트에 추가할 수 있습니다.

특히 운영자는 **CRD**를 필수 **RBAC** 정책 및 기타 소프트웨어별 논리와 함께 패키지로 제공하는 방식으로 **CRD**를 사용합니다. 또한 클러스터 관리자는 **Operator**의 라이프사이클 외부에서 클러스터에 **CRD**를 수동으로 추가하여 모든 사용자에게 제공할 수 있습니다.

참고

클러스터 관리자만 **CRD**를 생성할 수 있지만 기존 **CRD**에 대한 읽기 및 쓰기 권한이 있는 개발자의 경우 기존 **CRD**에서 **CR**을 생성할 수 있습니다.

2.8.2.2. 파일에서 사용자 정의 리소스 생성

CRD(사용자 정의 리소스 정의)가 클러스터에 추가되면 **CR** 사양을 사용하여 파일에서 **CLI**를 통해 **CR**(사용자 정의 리소스)을 생성할 수 있습니다.

사전 요구 사항

- 클러스터 관리자가 클러스터에 CRD를 추가했습니다.

프로세스

1. CR에 대한 YAML 파일을 생성합니다. 다음 예제 정의에서 cronSpec 및 image 사용자 정의 필드는 Kind: CronTab의 CR에 설정됩니다. Kind는 CRD 오브젝트의 spec.kind 필드에서 제공합니다.

CR에 대한 YAML 파일의 예

```
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 CRD에서 그룹 이름 및 API 버전(이름/버전)을 지정합니다.

2 CRD에 유형을 지정합니다.

3 오브젝트의 이름을 지정합니다.

4 해당하는 경우 오브젝트의 종료자를 지정합니다. 종료자를 사용하면 컨트롤러에서 오브젝트를 삭제하기 전에 완료해야 하는 조건을 구현할 수 있습니다.

5

오브젝트 유형별 조건을 지정합니다.

2. 파일을 생성한 후 오브젝트를 생성합니다.

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

2.8.2.3. 사용자 정의 리소스 검사

CLI를 사용하여 클러스터에 존재하는 CR(사용자 정의 리소스) 오브젝트를 검사할 수 있습니다.

사전 요구 사항

- CR 오브젝트는 액세스할 수 있는 네임스페이스에 있습니다.

프로세스

1. 특정 종류의 CR에 대한 정보를 얻으려면 다음을 실행합니다.

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

예를 들면 다음과 같습니다.

```
$ oc get crontab
```

출력 예

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

리소스 이름은 대소문자를 구분하지 않으며 CRD에 정의된 단수형 또는 복수형 양식이나 짧은 이름을 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. CR의 원시 YAML 데이터를 볼 수도 있습니다.

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

예를 들면 다음과 같습니다.

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

출력 예

```
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 오브젝트를 생성하는 데 사용한 YAML의 사용자 정의 데이터가 표시됩니다.

3장. 사용자 작업

3.1. 설치된 OPERATOR에서 애플리케이션 생성

이 가이드에서는 개발자에게 OpenShift Container Platform 웹 콘솔을 사용하여 설치한 Operator에서 애플리케이션을 생성하는 예제를 안내합니다.

3.1.1. Operator를 사용하여 etcd 클러스터 생성

이 절차에서는 OLM(Operator Lifecycle Manager)에서 관리하는 etcd Operator를 사용하여 새 etcd 클러스터를 생성하는 과정을 안내합니다.

사전 요구 사항

- OpenShift Container Platform 4.11 클러스터에 액세스할 수 있습니다.
- 관리자가 클러스터 수준에 etcd Operator를 이미 설치했습니다.

프로세스

1. 이 절차를 위해 OpenShift Container Platform 웹 콘솔에 새 프로젝트를 생성합니다. 이 예제에서는 my-etcd라는 프로젝트를 사용합니다.
2. Operator → 설치된 Operator 페이지로 이동합니다. 이 페이지에는 클러스터 관리자가 클러스터에 설치하여 사용할 수 있는 Operator가 CSV(클러스터 서비스 버전) 목록으로 표시됩니다. CSV는 Operator에서 제공하는 소프트웨어를 시작하고 관리하는 데 사용됩니다.

   작은 정보

   다음을 사용하여 CLI에서 이 목록을 가져올 수 있습니다.

   ```
   $ oc get csv
   ```

3. 자세한 내용과 사용 가능한 작업을 확인하려면 설치된 Operator 페이지에서 etcd Operator를 클릭합니다.

이 Operator에서는 제공된 API 아래에 표시된 것과 같이 etcd 클러스터(EtcdCluster 리소스)용 하나를 포함하여 새로운 리소스 유형 세 가지를 사용할 수 있습니다. 이러한 오브젝트는 내장된 네이티브 Kubernetes 오브젝트(예: Deployment 또는 ReplicaSet)와 비슷하게 작동하지만 etcd 관리와 관련된 논리가 포함됩니다.

4. 새 etcd 클러스터를 생성합니다.

   a. etcd 클러스터 API 상자에서 인스턴스 생성을 클릭합니다.

   b. 다음 화면을 사용하면 클러스터 크기와 같은 EtcdCluster 오브젝트의 최소 시작 템플릿을 수정할 수 있습니다. 지금은 생성을 클릭하여 종료하십시오. 그러면 Operator에서 새 etcd 클러스터의 Pod, 서비스 및 기타 구성 요소를 가동합니다.

5. 예제 etcd 클러스터를 클릭한 다음 리소스 탭을 클릭하여 Operator에서 자동으로 생성 및 구성한 리소스 수가 프로젝트에 포함되는지 확인합니다.

   프로젝트의 다른 Pod에서 데이터베이스에 액세스할 수 있도록 Kubernetes 서비스가 생성되었는지 확인합니다.

6. 지정된 프로젝트에서 edit 역할을 가진 모든 사용자는 클라우드 서비스와 마찬가지로 셀프서비스 방식으로 프로젝트에 이미 생성된 Operator에서 관리하는 애플리케이션 인스턴스(이 예제의 etcd 클러스터)를 생성, 관리, 삭제할 수 있습니다. 이 기능을 사용하여 추가 사용자를 활성화하려면 프로젝트 관리자가 다음 명령을 사용하여 역할을 추가하면 됩니다.

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

이제 Pod가 비정상적인 상태가 되거나 클러스터의 다른 노드로 마이그레이션되면 오류에 반응하고 데이터를 재조정할 etcd 클러스터가 생성되었습니다. 가장 중요한 점은 적절한 액세스 권한이 있는 클러스터 관리자 또는 개발자가 애플리케이션과 함께 데이터베이스를 쉽게 사용할 수 있다는 점입니다.

3.2. 네임스페이스에 OPERATOR 설치

클러스터 관리자가 계정에 Operator 설치 권한을 위임한 경우 셀프서비스 방식으로 Operator를 설치하고 네임스페이스에 등록할 수 있습니다.

3.2.1. 사전 요구 사항

- 클러스터 관리자는 셀프서비스 Operator를 네임스페이스에 설치할 수 있도록 OpenShift

Container Platform 사용자 계정에 특정 권한을 추가해야 합니다. 자세한 내용은 비 클러스터 관리자가 Operator를 설치하도록 허용 을 참조하십시오.

3.2.2. OperatorHub를 통한 Operator 설치 정보

OperatorHub는 Operator를 검색하는 사용자 인터페이스입니다. 이는 클러스터에 Operator를 설치하고 관리하는 OLM(Operator Lifecycle Manager)과 함께 작동합니다.

적절한 권한이 있는 클러스터 관리자는 OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 OperatorHub에서 Operator를 설치할 수 있습니다.

설치하는 동안 Operator의 다음 초기 설정을 결정해야합니다.

설치 모드

Operator를 설치할 특정 네임스페이스를 선택합니다.

업데이트 채널

여러 채널을 통해 Operator를 사용할 수있는 경우 구독할 채널을 선택할 수 있습니다. 예를 들어, stable 채널에서 배치하려면 (사용 가능한 경우) 목록에서 해당 채널을 선택합니다.

승인 전략

자동 또는 수동 업데이트를 선택할 수 있습니다.

설치된 Operator에 대해 자동 업데이트를 선택하는 경우 선택한 채널에 해당 Operator의 새 버전이 제공되면 OLM(Operator Lifecycle Manager)에서 Operator의 실행 중인 인스턴스를 개입 없이 자동으로 업그레이드합니다.

수동 업데이트를 선택하면 최신 버전의 Operator가 사용 가능할 때 OLM이 업데이트 요청을 작성합니다. 클러스터 관리자는 Operator를 새 버전으로 업데이트하려면 OLM 업데이트 요청을 수동으로 승인해야 합니다.

- OperatorHub 이해

3.2.3. 웹 콘솔을 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하여 OperatorHub에서 Operator를 설치하고 구독할

수 있습니다.

사전 요구 사항

- Operator 설치 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

프로세스

1. 웹 콘솔에서 Operators → OperatorHub 페이지로 이동합니다.

2. 원하는 Operator를 찾으려면 키워드를 Filter by keyword 상자에 입력하거나 스크롤합니다. 예를 들어 Kubernetes Operator의 고급 클러스터 관리 기능을 찾으려면 advanced를 입력합니다.

   인프라 기능에서 옵션을 필터링할 수 있습니다. 예를 들어, 연결이 끊긴 환경 (제한된 네트워크 환경이라고도 함)에서 작업하는 Operator를 표시하려면 Disconnected를 선택합니다.

3. Operator를 선택하여 추가 정보를 표시합니다.

참고

커뮤니티 Operator를 선택하면 Red Hat이 커뮤니티 Operator를 인증하지 않는다고 경고합니다. 계속하기 전에 경고를 확인해야합니다.

4. Operator에 대한 정보를 확인하고 Install을 클릭합니다.

5. Operator 설치 페이지에서 다음을 수행합니다.

   a. Operator를 설치할 특정 단일 네임스페이스를 선택합니다. Operator는 이 단일 네임 스페이스에서만 모니터링 및 사용할 수 있게 됩니다.

   b. Update Channe을 선택합니다 (하나 이상이 사용 가능한 경우).

c.

앞에서 설명한 대로 자동 또는 수동 승인 전략을 선택합니다.

6.

이 **OpenShift Container Platform** 클러스터에서 선택한 네임스페이스에서 **Operator**를 사용할 수 있도록 하려면 설치를 클릭합니다.

a.

수동 승인 전략을 선택한 경우 설치 계획을 검토하고 승인할 때까지 서브스크립션의 업그레이드 상태가 업그레이드 중으로 유지됩니다.

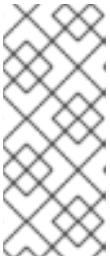
Install Plan 페이지에서 승인 한 후 **subscription** 업그레이드 상태가 **Up to date**로 이동합니다.

b.

자동 승인 전략을 선택한 경우 업그레이드 상태가 개입 없이 최신 상태로 확인되어야 합니다.

7.

서브스크립션의 업그레이드 상태가 최신이면 **Operator → 설치된 Operator**를 선택하여 설치된 **Operator**의 **CSV**(클러스터 서비스 버전)가 최종적으로 표시되는지 확인합니다. 상태는 최종적으로 관련 네임스페이스에서 **InstallSucceeded**로 확인되어야 합니다.



참고

모든 네임스페이스... 설치 모드의 경우, **openshift-operators** 네임스페이스에서 상태가 **InstallSucceeded**로 확인되지만 다른 네임스페이스에서 확인하면 상태가 복사됨입니다.

그렇지 않은 경우 다음을 수행합니다.

a.

워크로드 → **Pod** 페이지의 **openshift-operators** 프로젝트(또는 특정 네임스페이스...설치 모드가 선택된 경우 기타 관련 네임스페이스)에서 문제를 보고하는 모든 **Pod**의 로그를 확인하여 문제를 추가로 해결합니다.

3.2.4. CLI를 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하는 대신 **CLI**를 사용하여 **OperatorHub**에서 **Operator**를 설치할 수 있습니다. **oc** 명령을 사용하여 **Subscription** 개체를 만들거나 업데이트합니다.

사전 요구 사항

- Operator 설치 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

- 로컬 시스템에 oc 명령을 설치합니다.

프로세스

1. OperatorHub에서 클러스터에 사용 가능한 Operator의 목록을 표시합니다.

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

출력 예

```
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
...
```

필요한 Operator의 카탈로그를 기록해 둡니다.

2. 필요한 Operator를 검사하여 지원되는 설치 모드 및 사용 가능한 채널을 확인합니다.

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

3. OperatorGroup 오브젝트로 정의되는 Operator group에서 Operator group과 동일한 네임스페이스에 있는 모든 Operator에 대해 필요한 RBAC 액세스 권한을 생성할 대상 네임스페이스를 선택합니다.

Operator를 서브스크립션하는 네임스페이스에는 Operator의 설치 모드, 즉 AllNamespaces 또는 SingleNamespace 모드와 일치하는 Operator group이 있어야 합니다. 설치하려는 Operator에서 AllNamespaces를 사용하는 경우 openshift-operators 네임스페이스에 적절한 Operator group이 이미 있습니다.

그러나 Operator에서 SingleNamespace 모드를 사용하고 적절한 Operator group이 없는 경우 이를 생성해야 합니다.

참고

이 프로세스의 웹 콘솔 버전에서는 SingleNamespace 모드를 선택할 때 자동으로 OperatorGroup 및 Subscription 개체 생성을 자동으로 처리합니다.

a. OperatorGroup 개체 YAML 파일을 만듭니다 (예: operatorgroup.yaml).

OperatorGroup 오브젝트의 예

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

주의

OLM(Operator Lifecycle Manager)은 각 Operator 그룹에 대해 다음 클러스터 역할을 생성합니다.

- <operatorgroup_name>-admin
- <operatorgroup_name>-edit
- <operatorgroup_name>-view

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 Operator 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

b. OperatorGroup 개체를 생성합니다.

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

4. Subscription 개체 YAML 파일을 생성하여 OpenShift Pipelines Operator에 네임스페이스를 등록합니다(예: sub.yaml).

Subscription 개체 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace 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

기본 **AllNamespaces** 설치 모드 사용량의 경우 **openshift-operators** 네임스페이스를 지정합니다. 또는 사용자 지정 글로벌 네임스페이스를 생성한 경우 지정할 수 있습니다. 그 외에는 **SingleNamespace** 설치 모드를 사용하도록 관련 단일 네임스페이스를 지정합니다.

2

등록할 채널의 이름입니다.

3

등록할 **Operator**의 이름입니다.

4

Operator를 제공하는 카탈로그 소스의 이름입니다.

5

6

env 매개 변수는 OLM에서 생성한 포드의 모든 컨테이너에 있어야 하는 환경 변수 목록을 정의합니다.

7

envFrom 매개 변수는 컨테이너에서 환경 변수를 채울 소스 목록을 정의합니다.

8

volumes 매개변수는 OLM에서 생성한 Pod에 있어야 하는 볼륨 목록을 정의합니다.

9

volumeMounts 매개변수는 OLM에서 생성한 Pod의 모든 컨테이너에 있어야 하는 VolumeMount 목록을 정의합니다. volumeMount 가 존재하지 않는 볼륨을 참조하는 경우 OLM에서 Operator를 배포하지 못합니다.

10

tolerations 매개변수는 OLM에서 생성한 Pod의 허용 목록을 정의합니다.

11

resources 매개변수는 OLM에서 생성한 Pod의 모든 컨테이너에 대한 리소스 제약 조건을 정의합니다.

12

nodeSelector 매개변수는 OLM에서 생성한 Pod에 대한 NodeSelector 를 정의합니다.

5. Subscription 오브젝트를 생성합니다.

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

이 시점에서 OLM은 이제 선택한 Operator를 인식합니다. Operator의 CSV(클러스터 서비스 버전)가 대상 네임스페이스에 표시되고 Operator에서 제공하는 API를 생성에 사용할 수 있어야 합니다.

추가 리소스

- Operator groups
- 채널 이름

3.2.5. 특정 버전의 Operator 설치

Subscription 오브젝트에 **CSV**(클러스터 서비스 버전)를 설정하면 특정 버전의 **Operator**를 설치할 수 있습니다.

사전 요구 사항

- **Operator** 설치 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- **OpenShift CLI(oc)**가 설치됨

프로세스

1. **startingCSV** 필드를 설정하여 특정 버전의 **Operator**에 네임스페이스를 서브스크립션하는 **Subscription** 오브젝트 **YAML** 파일을 생성합니다. 카탈로그에 이후 버전이 있는 경우 **Operator**가 자동으로 업그레이드되지 않도록 **installPlanApproval** 필드를 **Manual**로 설정합니다.

   예를 들어 다음 **sub.yaml** 파일을 사용하여 특별히 **3.4.0** 버전에 **Red Hat Quay Operator**를 설치할 수 있습니다.

   특정 시작 **Operator** 버전이 있는 서브스크립션

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: quay-operator
  namespace: quay
spec:
  channel: quay-v3.4
  installPlanApproval: Manual (1)
  name: quay-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: quay-operator.v3.4.0 (2)
```

1 지정된 버전이 카탈로그의 이후 버전으로 대체될 경우 승인 전략을 **Manual**로 설정합니다. 이 계획에서는 이후 버전으로 자동 업그레이드할 수 없으므로 시작 **CSV**에서 설치를 완료하려면 수동 승인이 필요합니다.

2 **Operator CSV**의 특정 버전을 설정합니다.

2. **Subscription** 오브젝트를 생성합니다.

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

3. 보류 중인 설치 계획을 수동으로 승인하여 **Operator** 설치를 완료합니다.

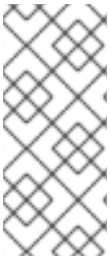
추가 리소스

- 보류 중인 **Operator** 업데이트 수동 승인

4장. 관리자 작업

4.1. 클러스터에 OPERATOR 추가

클러스터 관리자는 OperatorHub가 있는 네임스페이스에 Operator를 등록하여 OpenShift Container Platform 클러스터에 Operator를 설치할 수 있습니다.



참고

OLM에서 동일한 네임스페이스에 배치된 설치된 Operator에 대한 업데이트를 처리하는 방법과 사용자 정의 글로벌 Operator 그룹을 사용하여 Operator를 설치하는 대체 방법은 Multitenancy 및 Operator colocation 을 참조하십시오.

4.1.1. OperatorHub를 통한 Operator 설치 정보

OperatorHub는 Operator를 검색하는 사용자 인터페이스입니다. 이는 클러스터에 Operator를 설치하고 관리하는 OLM(Operator Lifecycle Manager)과 함께 작동합니다.

적절한 권한이 있는 클러스터 관리자는 OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 OperatorHub에서 Operator를 설치할 수 있습니다.

설치하는 동안 Operator의 다음 초기 설정을 결정해야합니다.

설치 모드

Operator를 설치할 특정 네임스페이스를 선택합니다.

업데이트 채널

여러 채널을 통해 Operator를 사용할 수있는 경우 구독할 채널을 선택할 수 있습니다. 예를 들어, stable 채널에서 배치하려면 (사용 가능한 경우) 목록에서 해당 채널을 선택합니다.

승인 전략

자동 또는 수동 업데이트를 선택할 수 있습니다.

설치된 Operator에 대해 자동 업데이트를 선택하는 경우 선택한 채널에 해당 Operator의 새 버전이 제공되면 OLM(Operator Lifecycle Manager)에서 Operator의 실행 중인 인스턴스를 개입 없이 자동으로 업그레이드합니다.

수동 업데이트를 선택하면 최신 버전의 Operator가 사용 가능할 때 OLM이 업데이트 요청을 작성합니다. 클러스터 관리자는 Operator를 새 버전으로 업데이트하려면 OLM 업데이트 요청을 수동으로 승인해야 합니다.

추가 리소스

- OperatorHub 이해

4.1.2. 웹 콘솔을 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하여 OperatorHub에서 Operator를 설치하고 구독할 수 있습니다.

사전 요구 사항

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

프로세스

1. 웹 콘솔에서 Operators → OperatorHub 페이지로 이동합니다.
2. 원하는 Operator를 찾으려면 키워드를 Filter by keyword 상자에 입력하거나 스크롤합니다. 예를 들어 Kubernetes Operator의 고급 클러스터 관리 기능을 찾으려면 advanced를 입력합니다.

   인프라 기능에서 옵션을 필터링할 수 있습니다. 예를 들어, 연결이 끊긴 환경 (제한된 네트워크 환경이라고도 함)에서 작업하는 Operator를 표시하려면 Disconnected를 선택합니다.
3. Operator를 선택하여 추가 정보를 표시합니다.

참고

커뮤니티 **Operator**를 선택하면 **Red Hat**이 커뮤니티 **Operator**를 인증하지 않는다고 경고합니다. 계속하기 전에 경고를 확인해야합니다.

4. **Operator**에 대한 정보를 확인하고 **Install**을 클릭합니다.

5. **Operator** 설치 페이지에서 다음을 수행합니다.

   a. 다음 명령 중 하나를 선택합니다.

   - **All namespaces on the cluster (default)**에서는 기본 **openshift-operators** 네임스페이스에 **Operator**가 설치되므로 **Operator**가 클러스터의 모든 네임스페이스를 모니터링하고 사용할 수 있습니다. 이 옵션을 항상 사용할 수있는 것은 아닙니다.
   - **A specific namespace on the cluster**를 사용하면 **Operator**를 설치할 특정 단일 네임 스페이스를 선택할 수 있습니다. **Operator**는 이 단일 네임 스페이스에서만 모니터링 및 사용할 수 있게 됩니다.

   b. **Operator**를 설치할 특정 단일 네임스페이스를 선택합니다. **Operator**는 이 단일 네임 스페이스에서만 모니터링 및 사용할 수 있게 됩니다.

   c. **Update Channe**을 선택합니다 (하나 이상이 사용 가능한 경우).

   d. 앞에서 설명한 대로 자동 또는 수동 승인 전략을 선택합니다.

6. 이 **OpenShift Container Platform** 클러스터에서 선택한 네임스페이스에서 **Operator**를 사용할 수 있도록 하려면 설치를 클릭합니다.

   a. 수동 승인 전략을 선택한 경우 설치 계획을 검토하고 승인할 때까지 서브스크립션의 업그레이드 상태가 업그레이드 중으로 유지됩니다.

   Install Plan 페이지에서 승인 한 후 **subscription** 업그레이드 상태가 **Up to date**로 이동합니다.

b.
자동 승인 전략을 선택한 경우 업그레이드 상태가 개입 없이 최신 상태로 확인되어야 합니다.

7.
서브스크립션의 업그레이드 상태가 최신이면 **Operator** → 설치된 **Operator**를 선택하여 설치된 **Operator**의 **CSV**(클러스터 서비스 버전)가 최종적으로 표시되는지 확인합니다. 상태는 최종적으로 관련 네임스페이스에서 **InstallSucceeded**로 확인되어야 합니다.

참고

모든 네임스페이스... 설치 모드의 경우, **openshift-operators** 네임스페이스에서 상태가 **InstallSucceeded**로 확인되지만 다른 네임스페이스에서 확인하면 상태가 복사됨입니다.

그렇지 않은 경우 다음을 수행합니다.

a.
워크로드 → **Pod** 페이지의 **openshift-operators** 프로젝트(또는 특정 네임스페이스...설치 모드가 선택된 경우 기타 관련 네임스페이스)에서 문제를 보고하는 모든 **Pod**의 로그를 확인하여 문제를 추가로 해결합니다.

4.1.3. CLI를 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하는 대신 **CLI**를 사용하여 **OperatorHub**에서 **Operator**를 설치할 수 있습니다. **oc** 명령을 사용하여 **Subscription** 개체를 만들거나 업데이트합니다.

사전 요구 사항

- **Operator** 설치 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- 로컬 시스템에 **oc** 명령을 설치합니다.

절차

1.
OperatorHub에서 클러스터에 사용 가능한 **Operator**의 목록을 표시합니다.

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

출력 예

```
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
...
```

필요한 Operator의 카탈로그를 기록해 둡니다.

2. 필요한 Operator를 검사하여 지원되는 설치 모드 및 사용 가능한 채널을 확인합니다.

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

3. OperatorGroup 오브젝트로 정의되는 Operator group에서 Operator group과 동일한 네임스페이스에 있는 모든 Operator에 대해 필요한 RBAC 액세스 권한을 생성할 대상 네임스페이스를 선택합니다.

Operator를 서브스크립션하는 네임스페이스에는 Operator의 설치 모드, 즉 AllNamespaces 또는 SingleNamespace 모드와 일치하는 Operator group이 있어야 합니다. 설치하려는 Operator에서 AllNamespaces를 사용하는 경우 openshift-operators 네임스페이스에 적절한 Operator group이 이미 있습니다.

그러나 Operator에서 SingleNamespace 모드를 사용하고 적절한 Operator group이 없는 경우 이를 생성해야 합니다.

참고

이 프로세스의 웹 콘솔 버전에서는 **SingleNamespace** 모드를 선택할 때 자동으로 **OperatorGroup** 및 **Subscription** 개체 생성을 자동으로 처리합니다.

a. **OperatorGroup** 개체 **YAML** 파일을 만듭니다 (예: **operatorgroup.yaml**).

OperatorGroup 오브젝트의 예

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

주의

OLM(Operator Lifecycle Manager)은 각 Operator 그룹에 대해 다음 클러스터 역할을 생성합니다.

- <operatorgroup_name>-admin
- <operatorgroup_name>-edit
- <operatorgroup_name>-view

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 Operator 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

b. OperatorGroup 개체를 생성합니다.

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

4. Subscription 개체 YAML 파일을 생성하여 OpenShift Pipelines Operator에 네임스페이스를 등록합니다(예: sub.yaml).

Subscription 개체 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace 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

기본 **AllNamespaces** 설치 모드 사용량의 경우 **openshift-operators** 네임스페이스를 지정합니다. 또는 사용자 지정 글로벌 네임스페이스를 생성한 경우 지정할 수 있습니다. 그 외에는 **SingleNamespace** 설치 모드를 사용하도록 관련 단일 네임스페이스를 지정합니다.

2

등록할 채널의 이름입니다.

3

등록할 **Operator**의 이름입니다.

4

Operator를 제공하는 카탈로그 소스의 이름입니다.

5

6

env 매개 변수는 **OLM**에서 생성한 포드의 모든 컨테이너에 있어야 하는 환경 변수 목록을 정의합니다.

7

envFrom 매개 변수는 컨테이너에서 환경 변수를 채울 소스 목록을 정의합니다.

8

volumes 매개변수는 **OLM**에서 생성한 **Pod**에 있어야 하는 볼륨 목록을 정의합니다.

9

volumeMounts 매개변수는 **OLM**에서 생성한 **Pod**의 모든 컨테이너에 있어야 하는 **VolumeMount** 목록을 정의합니다. **volumeMount** 가 존재하지 않는 볼륨을 참조하는 경우 **OLM**에서 **Operator**를 배포하지 못합니다.

10

tolerations 매개변수는 **OLM**에서 생성한 **Pod**의 허용 목록을 정의합니다.

11

resources 매개변수는 **OLM**에서 생성한 **Pod**의 모든 컨테이너에 대한 리소스 제약 조건을 정의합니다.

12

nodeSelector 매개변수는 **OLM**에서 생성한 **Pod**에 대한 **NodeSelector** 를 정의합니다.

5. **Subscription** 오브젝트를 생성합니다.

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

이 시점에서 **OLM**은 이제 선택한 **Operator**를 인식합니다. **Operator**의 **CSV**(클러스터 서비스 버전)가 대상 네임스페이스에 표시되고 **Operator**에서 제공하는 **API**를 생성에 사용할 수 있어야 합니다.

추가 리소스

- Operator groups 정의

4.1.4. 특정 버전의 Operator 설치

Subscription 오브젝트에 CSV(클러스터 서비스 버전)를 설정하면 특정 버전의 Operator를 설치할 수 있습니다.

사전 요구 사항

- Operator 설치 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)가 설치됨

절차

1. startingCSV 필드를 설정하여 특정 버전의 Operator에 네임스페이스를 서브스크립션하는 Subscription 오브젝트 YAML 파일을 생성합니다. 카탈로그에 이후 버전이 있는 경우 Operator가 자동으로 업그레이드되지 않도록 installPlanApproval 필드를 Manual로 설정합니다.

   예를 들어 다음 sub.yaml 파일을 사용하여 특별히 3.4.0 버전에 Red Hat Quay Operator를 설치할 수 있습니다.

   특정 시작 Operator 버전이 있는 서브스크립션

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: quay-operator
  namespace: quay
spec:
  channel: quay-v3.4
  installPlanApproval: Manual (1)
  name: quay-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: quay-operator.v3.4.0 (2)
```

1

지정된 버전이 카탈로그의 이후 버전으로 대체될 경우 승인 전략을 **Manual**로 설정합니다. 이 계획에서는 이후 버전으로 자동 업그레이드할 수 없으므로 시작 **CSV**에서 설치를 완료하려면 수동 승인이 필요합니다.

2

Operator CSV의 특정 버전을 설정합니다.

2. **Subscription** 오브젝트를 생성합니다.

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

3. 보류 중인 설치 계획을 수동으로 승인하여 **Operator** 설치를 완료합니다.

추가 리소스

- 보류 중인 **Operator** 업데이트 수동 승인

4.1.5. 다중 테넌트 클러스터에 대한 Operator의 여러 인스턴스 준비

클러스터 관리자는 다중 테넌트 클러스터에서 사용할 **Operator**의 여러 인스턴스를 추가할 수 있습니다. 이는 표준 모든 네임스페이스 설치 모드를 사용하는 대체 솔루션으로, 최소 권한 원칙을 위반하거나 널리 채택되지 않는 **Multinamespace** 모드를 위반하는 것으로 간주될 수 있습니다. 자세한 내용은 " 다중 테넌트 클러스터의 **Operator**"를 참조하십시오.

다음 절차에서는 배포된 워크로드 집합에 대한 공통 액세스 및 권한을 공유하는 사용자 또는 사용자 그룹입니다. *테넌트 **Operator** 는 해당 테넌트에서만 사용하도록 설계된 **Operator**의 인스턴스입니다.*

사전 요구 사항

- *설치하려는 **Operator**의 모든 인스턴스는 지정된 클러스터에서 버전이 동일해야 합니다.*

중요

*이 제한 사항 및 기타 제한 사항에 대한 자세한 내용은 " 다중 테넌트 클러스터의 **Operator**"를 참조하십시오.*

프로세스

1. Operator를 설치하기 전에 테넌트의 네임스페이스와 별도의 테넌트 Operator의 네임스페이스를 생성합니다. 예를 들어 테넌트 네임스페이스가 team1 인 경우 team1-operator 네임스페이스를 생성할 수 있습니다.

   a. Namespace 리소스를 정의하고 YAML 파일을 저장합니다(예: team1-operator.yaml).

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: team1-operator
   ```

   b. 다음 명령을 실행하여 네임스페이스를 생성합니다.

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

2. spec.targetNamespaces 목록에 있는 하나의 namespace 항목만 사용하여 테넌트 Operator 범위에 대한 Operator group을 생성합니다.

   a. OperatorGroup 리소스를 정의하고 YAML 파일을 저장합니다(예: team1-operatorgroup.yaml).

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

   1 spec.targetNamespaces 목록에서 테넌트의 네임스페이스만 정의합니다.

   b. 다음 명령을 실행하여 Operator 그룹을 생성합니다.

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

다음 단계

- 테넌트 Operator 네임스페이스에 Operator를 설치합니다. 이 작업은 CLI 대신 웹 콘솔에서 OperatorHub를 사용하여 더 쉽게 수행할 수 있습니다. 자세한 절차는 웹 콘솔을 사용하여 OperatorHub에서 설치를 참조하십시오.

참고

Operator 설치를 완료한 후 Operator는 테넌트 Operator 네임스페이스에 상주하고 테넌트 네임스페이스를 감시하지만, 테넌트에서 Operator와 해당 서비스 계정을 보거나 사용할 수 없습니다.

추가 리소스

- 다중 테넌트 클러스터의 Operator

4.1.6. 사용자 지정 네임스페이스에 글로벌 Operator 설치

OpenShift Container Platform 웹 콘솔을 사용하여 Operator 를 설치할 때 기본 동작은 모든 네임스페이스 설치 모드를 지원하는 Operator를 기본 openshift-operators 글로벌 네임스페이스에 설치합니다. 이로 인해 공유 설치 계획 및 네임스페이스의 모든 Operator 간의 정책 업데이트와 관련된 문제가 발생할 수 있습니다. 이러한 제한 사항에 대한 자세한 내용은 "Multitenancy and Operator colocation"을 참조하십시오.

클러스터 관리자는 사용자 정의 글로벌 네임스페이스를 생성하고 해당 네임스페이스를 사용하여 Operator 및 해당 종속 항목을 개별적으로 설치하거나 범위가 지정된 종속 항목을 설치하여 이 기본 동작을 수동으로 바이패스할 수 있습니다.

프로세스

1. Operator를 설치하기 전에 원하는 Operator를 설치할 네임스페이스를 생성합니다. 이 설치 네임스페이스는 사용자 지정 글로벌 네임스페이스가 됩니다.

   a. 네임스페이스 리소스를 정의하고 YAML 파일을 저장합니다(예: global-operators.yaml).

```
apiVersion: v1
kind: Namespace
metadata:
  name: global-operators
```

b.

다음 명령을 실행하여 네임스페이스를 생성합니다.

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

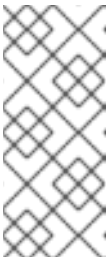
2.

모든 네임스페이스를 감시하는 Operator group인 사용자 정의 글로벌 Operator group을 생성합니다.

a.

OperatorGroup 리소스를 정의하고 YAML 파일(예: global-operatorgroup.yaml)을 저장합니다. spec.selector 및 spec.targetNamespaces 필드를 모두 생략하여 모든 네임스페이스를 선택하는 글로벌 Operator group 으로 설정합니다.

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



참고

생성된 글로벌 Operator group의 status.namespaces 에는 사용 중인 Operator에 모든 네임스페이스를 조사해야 함을 알리는 빈 문자열("")이 포함되어 있습니다.

b.

다음 명령을 실행하여 Operator 그룹을 생성합니다.

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

다음 단계

- 사용자 정의 글로벌 네임스페이스에 원하는 Operator를 설치합니다. Operator 설치 중에 웹 콘솔은 사용자 정의 글로벌 네임스페이스로 설치된 네임스페이스 메뉴를 채우지 않으므로 이 작업은 OpenShift CLI(oc)로만 수행할 수 있습니다. 자세한 절차는 CLI를 사용하여 OperatorHub에서 설치를 참조하십시오.

참고

Operator 설치를 시작할 때 Operator에 종속 항목이 있으면 종속 항목도 사용자 지정 글로벌 네임스페이스에 자동으로 설치됩니다. 결과적으로 종속성 Operator에 동일한 업데이트 정책 및 공유 설치 계획이 있는 것이 유효합니다.

추가 리소스

- 멀티 테넌시 및 Operator 공동 배치

4.1.7. Operator 워크로드의 Pod 배치

기본적으로 OLM(Operator Lifecycle Manager)은 Operator를 설치하거나 Operand 워크로드를 배포할 때 임의의 작업자 노드에 Pod를 배치합니다. 관리자는 노드 선택기, 테인트 및 허용 오차가 결합된 프로젝트를 사용하여 특정 노드에 Operator 및 Operand 배치를 제어할 수 있습니다.

Operator 및 Operand 워크로드의 Pod 배치 제어에는 다음과 같은 사전 요구 사항이 있습니다.

1. 요구 사항에 따라 Pod를 대상으로 할 노드 또는 노드 집합을 결정합니다. 사용 가능한 경우 노드 또는 노드를 식별하는 node-role.kubernetes.io/app과 같은 기존 레이블을 확인합니다. 그렇지 않으면 머신 세트를 사용하거나 노드를 직접 편집하여 myoperator와 같은 레이블을 추가합니다. 이후 단계에서 이 레이블을 프로젝트의 노드 선택기로 사용합니다.

2. 특정 레이블이 있는 Pod만 노드에서 실행되도록 허용하지만 관련이 없는 워크로드를 다른 노드에 추가하려면 머신 세트를 사용하거나 노드를 직접 편집하여 노드 또는 노드에 테인트를 추가합니다. 테인트와 일치하지 않는 새 Pod를 노드에서 예약할 수 없도록 하는 효과를 사용합니다. 예를 들어 myoperator:NoSchedule 테인트를 사용하면 테인트와 일치하지 않는 새 Pod가 해당 노드에 예약되지 않지만 노드의 기존 Pod는 그대로 유지됩니다.

3. 기본 노드 선택기와 테인트를 추가한 경우 일치하는 허용 오차로 구성된 프로젝트를 만듭니다.

이 시점에서 생성한 프로젝트를 사용하여 다음 시나리오에서 지정된 노드로 Pod를 이동할 수 있습니다.

Operator Pod의 경우

관리자는 프로젝트에 Subscription 오브젝트를 생성할 수 있습니다. 결과적으로 Operator Pod가 지정된 노드에 배치됩니다.

Operand Pod의 경우

설치된 Operator를 사용하여 프로젝트에 애플리케이션을 생성할 수 있습니다. 그러면 Operator가 프로젝트에 소유한 CR(사용자 정의 리소스)이 배치됩니다. 결과적으로 Operator가 다른 네임스페이스에 클러스터 수준 오브젝트 또는 리소스를 배포하지 않는 한 피연산자 Pod가 지정된 노드에 배치됩니다. 이 경우 사용자 정의 Pod 배치가 적용되지 않습니다.

추가 리소스

- 노드에 수동으로 또는 머신 세트를사용하여 테인트 및 허용 오차 추가
- 프로젝트 수준 노드 선택기 생성
- 노드 선택기 및 허용 오차를 사용하여 프로젝트 생성

4.2. 설치된 OPERATOR 업데이트

클러스터 관리자는 OpenShift Container Platform 클러스터에서 OLM(Operator Lifecycle Manager)을 사용하여 이전에 설치한 Operator를 업데이트할 수 있습니다.

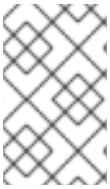
참고

OLM에서 동일한 네임스페이스에 배치된 설치된 Operator에 대한 업데이트를 처리하는 방법과 사용자 정의 글로벌 Operator 그룹을 사용하여 Operator를 설치하는 대체 방법은 Multitenancy 및 Operator colocation 을 참조하십시오.

4.2.1. Operator 업데이트 준비

설치된 Operator의 서브스크립션은 Operator의 업데이트를 추적하고 수신하는 업데이트 채널을 지정합니다. 업데이트 채널을 변경하여 추적을 시작하고 최신 채널에서 업데이트를 수신할 수 있습니다.

서브스크립션의 업데이트 채널 이름은 Operator마다 다를 수 있지만 이름 지정 스키마는 일반적으로 지정된 Operator 내의 공통 규칙을 따릅니다. 예를 들어 채널 이름은 Operator(1.2, 1.3) 또는 릴리스 빈도(stable, fast)에서 제공하는 애플리케이션의 마이너 릴리스 업데이트 스트림을 따를 수 있습니다.



참고

설치된 Operator는 현재 채널보다 오래된 채널로 변경할 수 없습니다.

Red Hat Customer Portal 랩에는 관리자가 Operator 업데이트를 준비하는 데 도움이 되는 다음 애플리케이션이 포함되어 있습니다.

- *Red Hat OpenShift Container Platform Operator 업데이트 정보 검사기*

애플리케이션을 사용하여 **Operator Lifecycle Manager** 기반 **Operator**를 검색하고 다양한 **OpenShift Container Platform** 버전에서 업데이트 채널당 사용 가능한 **Operator** 버전을 확인할 수 있습니다. **Cluster Version Operator** 기반 **Operator**는 포함되어 있지 않습니다.

4.2.2. Operator의 업데이트 채널 변경

OpenShift Container Platform 웹 콘솔을 사용하여 **Operator**의 업데이트 채널을 변경할 수 있습니다.

작은 정보

서브스크립션의 승인 전략이 자동으로 설정되어 있으면 선택한 채널에서 새 **Operator** 버전을 사용할 수 있게 되는 즉시 업데이트 프로세스가 시작됩니다. 승인 전략이 수동으로 설정된 경우 보류 중인 업데이트를 수동으로 승인해야 합니다.

사전 요구 사항

- **OLM(Operator Lifecycle Manager)**을 사용하여 이전에 설치한 **Operator**입니다.

프로세스

1. 웹 콘솔의 **Administrator** 모드에서 **Operator → Installed Operators**로 이동합니다.
2. 업데이트 채널을 변경할 **Operator** 이름을 클릭합니다.
3. 서브스크립션 탭을 클릭합니다.
4. 채널에서 업데이트 채널의 이름을 클릭합니다.
5. 변경할 최신 업데이트 채널을 클릭한 다음 저장을 클릭합니다.
6. 자동 승인 전략이 있는 서브스크립션의 경우 업데이트가 자동으로 시작됩니다. **Operator →**

설치된 Operator 페이지로 다시 이동하여 업데이트 진행 상황을 모니터링합니다. 완료되면 상태가 성공 및 최신으로 변경됩니다.

수동 승인 전략이 있는 서브스크립션의 경우 서브스크립션 탭에서 업데이트를 수동으로 승인할 수 있습니다.

4.2.3. 보류 중인 Operator 업데이트 수동 승인

설치된 Operator의 서브스크립션에 있는 승인 전략이 수동으로 설정된 경우 새 업데이트가 현재 업데이트 채널에 릴리스될 때 업데이트를 수동으로 승인해야 설치가 시작됩니다.

사전 요구 사항

- OLM(Operator Lifecycle Manager)을 사용하여 이전에 설치한 Operator입니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 관리자 관점에서 Operator → 설치된 Operator로 이동합니다.
2. 보류 중인 업데이트가 있는 Operator에 업그레이드 사용 가능 상태가 표시됩니다. 업데이트할 Operator 이름을 클릭합니다.
3. 서브스크립션 탭을 클릭합니다. 승인이 필요한 업데이트는 업그레이드 상태 옆에 표시됩니다. 예를 들어 1 승인 필요가 표시될 수 있습니다.
4. 1 승인 필요를 클릭한 다음 설치 계획 프리뷰를 클릭합니다.
5. 업데이트에 사용할 수 있는 것으로 나열된 리소스를 검토합니다. 문제가 없는 경우 승인을 클릭합니다.
6. Operator → 설치된 Operator 페이지로 다시 이동하여 업데이트 진행 상황을 모니터링합니다. 완료되면 상태가 성공 및 최신으로 변경됩니다.

4.3. 클러스터에서 OPERATOR 삭제

다음은 OpenShift Container Platform 클러스터에서 OLM(Operator Lifecycle Manager)을 사용하여 이전에 설치한 Operator를 삭제하거나 제거하는 방법을 설명합니다.

중요

동일한 Operator를 다시 설치하기 전에 Operator를 성공적으로 제거하고 완전히 제거해야 합니다. Operator를 완전히 설치 해제하지 않으면 프로젝트 또는 네임스페이스와 같은 리소스를 "Terminating" 상태가 되고 Operator를 다시 설치하려고 할 때 "error resolving resource" 메시지가 확인될 수 있습니다. 자세한 내용은 제거 실패 후 Operator 다시 설치를 참조하십시오.

4.3.1. 웹 콘솔을 사용하여 클러스터에서 Operator 삭제

클러스터 관리자는 웹 콘솔을 사용하여 선택한 네임스페이스에서 설치된 Operator를 삭제할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터 웹 콘솔에 액세스할 수 있습니다.

프로세스

1. Operator → 설치된 Operator 페이지로 이동합니다.
2. 제거하려는 Operator를 찾으려면 이름으로 필터링 필드에 키워드를 스크롤하거나 입력합니다. 그런 다음 해당 Operator를 클릭합니다.
3. Operator 세부 정보 페이지 오른쪽에 있는 작업 목록에서 Operator 제거를 선택합니다.

   Operator를 설치 제거하시겠습니까? 대화 상자가 표시됩니다.
4. 설치 제거를 선택하여 Operator, Operator 배포 및 Pod를 제거합니다. 이 작업 후에 Operator는 실행을 중지하고 더 이상 업데이트가 수신되지 않습니다.

참고

이 작업은 CRD(사용자 정의 리소스 정의) 및 CR(사용자 정의 리소스)을 포함하여 Operator에서 관리하는 리소스를 제거하지 않습니다. 웹 콘솔에서 활성화된 대시보드 및 탐색 항목과 계속 실행되는 클러스터 외부 리소스는 수동 정리가 필요할 수 있습니다. Operator를 설치 제거한 후 해당 항목을 제거하려면 Operator CRD를 수동으로 삭제해야 할 수 있습니다.

4.3.2. CLI를 사용하여 클러스터에서 Operator 삭제

클러스터 관리자는 CLI를 사용하여 선택한 네임스페이스에서 설치된 Operator를 삭제할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- oc 명령이 워크스테이션에 설치되어 있습니다.

프로세스

1. 구독된 Operator의 최신 버전(예: 서버리스-operator)이 currentCSV 필드에서 식별되는지 확인합니다.

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

출력 예

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

2. 서브스크립션을 삭제합니다(예: 서버리스-operator).

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

출력 예

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

3. 이전 단계의 currentCSV 값을 사용하여 대상 네임스페이스에서 Operator의 CSV를 삭제합니다.

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

출력 예

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

4.3.3. 실패한 서브스크립션 새로 고침

OLM(Operator Lifecycle Manager)에서는 네트워크상에서 액세스할 수 없는 이미지를 참조하는 Operator를 구독하는 경우 openshift-marketplace 네임스페이스에 다음 오류로 인해 실패하는 작업을 확인할 수 있습니다.

출력 예

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

출력 예

```
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
```

결과적으로 서브스크립션이 이러한 장애 상태에 고착되어 Operator를 설치하거나 업그레이드할 수 없습니다.

서브스크립션, CSV(클러스터 서비스 버전) 및 기타 관련 오브젝트를 삭제하여 실패한 서브스크립션을 새로 고칠 수 있습니다. 서브스크립션을 다시 생성하면 OLM에서 올바른 버전의 Operator를 다시 설치합니다.

사전 요구 사항

- 액세스할 수 없는 번들 이미지를 가져올 수 없는 실패한 서브스크립션이 있습니다.
- 올바른 번들 이미지에 액세스할 수 있는지 확인했습니다.

절차

1. Operator가 설치된 네임스페이스에서 Subscription 및 ClusterServiceVersion 오브젝트의 이름을 가져옵니다.

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

출력 예

```
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. *서브스크립션을 삭제합니다.*

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

3. *클러스터 서비스 버전을 삭제합니다.*

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

4. **openshift-marketplace** *네임스페이스에서 실패한 모든 작업 및 관련 구성 맵의 이름을 가져옵니다.*

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

출력 예

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

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

5. *작업을 삭제합니다.*

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

이렇게 하면 액세스할 수 없는 이미지를 가져오려는 **Pod**가 *다시 생성되지 않습니다.*

6. *구성 맵을 삭제합니다.*

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

–

7. 웹 콘솔에서 OperatorHub를 사용하여 Operator를 다시 설치합니다.

검증

- Operator가 제대로 다시 설치되었는지 확인합니다.

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

4.4. OPERATOR LIFECYCLE MANAGER 기능 구성

OLM(Operator Lifecycle Manager) 컨트롤러는 cluster 라는 OLMConfig CR(사용자 정의 리소스)에 의해 구성됩니다. 클러스터 관리자는 특정 기능을 활성화하거나 비활성화하도록 이 리소스를 수정할 수 있습니다.

이 문서에서는 OLMConfig 리소스에서 구성하는 OLM에서 현재 지원하는 기능에 대해 간단히 설명합니다.

4.4.1. CSV 복사본 비활성화

OLM(Operator Lifecycle Manager)에서 Operator를 설치하면 Operator가 조사하도록 구성된 모든 네임스페이스에서 CSV(클러스터 서비스 버전)의 단순화된 사본이 생성됩니다. 이러한 CSV는 CSV 복사본 이라고 하며 지정된 네임스페이스에서 현재 리소스 이벤트를 조정 중인 컨트롤러와 사용자에게 통신합니다.

Operator가 AllNamespaces 설치 모드를 사용하도록 구성된 경우, 단일 네임스페이스 또는 지정된 네임스페이스 세트를 대상으로 하는 경우 클러스터의 모든 네임스페이스에 복사된 CSV가 생성됩니다. 특히 대규모 클러스터에서 네임스페이스 및 Operator가 수백 또는 수천에 설치되었을 가능성이 있는 CSV는 OLM의 메모리 사용량, 클러스터 etcd 제한, 네트워킹과 같은 비활성화되지 않은 양의 리소스를 사용합니다.

이러한 대규모 클러스터를 지원하기 위해 클러스터 관리자는 AllNamespaces 모드로 설치된 Operator의 CSV 복사본을 비활성화할 수 있습니다.

주의

CSV 복사본을 비활성화하면 OperatorHub 및 CLI에서 Operator를 검색하는 사용자가 사용자 네임스페이스에 직접 설치된 Operator로 제한됩니다.

Operator가 사용자 네임스페이스의 이벤트를 조정하지만 다른 네임스페이스에 설치되도록 구성된 경우 사용자는 OperatorHub 또는 CLI에서 Operator를 볼 수 없습니다. 이 제한에 영향을 받는 Operator는 계속 사용할 수 있으며 사용자 네임스페이스에서 이벤트를 계속 조정합니다.

이 동작은 다음과 같은 이유로 발생합니다.

- CSV 복사본은 지정된 네임스페이스에 사용할 수 있는 Operator를 식별합니다.
- RBAC(역할 기반 액세스 제어)는 OperatorHub 및 CLI에서 Operator를 보고 검색할 수 있는 사용자의 기능을 지정합니다.

절차

- cluster 라는 OLMConfig 오브젝트를 편집하고 spec.features.disableCopiedCSVs 필드를 true 로 설정합니다.

```
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: true (1)
EOF
```

(1) AllNamespaces 설치 모드 Operator에 대해 복사된 CSV가 비활성화됨

검증

- CSV 복사본이 비활성화된 경우 OLM은 Operator의 네임 스페이스의 이벤트에 이 정보를 캡처합니다.

```
$ oc get events
```

출력 예

```
LAST SEEN   TYPE      REASON              OBJECT                                MESSAGE
85s         Warning   DisabledCopiedCSVs  clusterserviceversion/my-csv.v1.0.0   CSV copying disabled for operators/my-csv.v1.0.0
```

spec.features.disableCopiedCSVs 필드가 누락되거나 false 로 설정되면 OLM은 AllNamespaces 모드로 설치된 모든 Operator에 대해 복사된 CSV를 다시 만들고 이전에 언급한 이벤트를 삭제합니다.

추가 리소스

- 설치 모드

4.5. OPERATOR LIFECYCLE MANAGER에서 프록시 지원 구성

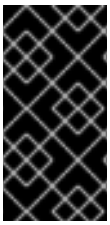
OpenShift Container Platform 클러스터에 글로벌 프록시가 구성된 경우 OLM(Operator Lifecycle Manager)은 클러스터 수준 프록시를 사용하여 관리하는 Operator를 자동으로 구성합니다. 그러나 설치된 Operator를 글로벌 프록시를 덮어쓰거나 사용자 정의 CA 인증서를 삽입하도록 구성할 수도 있습니다.

추가 리소스

- 클러스터 전체 프록시 구성
- 사용자 정의 PKI 구성 (사용자 정의 CA 인증서)
- Go,Ansible 및 Helm의 프록시 설정을 지원하는 Operator 개발

4.5.1. Operator의 프록시 설정 덮어쓰기

클러스터 수준 송신 프록시가 구성된 경우 **OLM(Operator Lifecycle Manager)**에서 실행되는 **Operator**는 배포 시 클러스터 수준 프록시 설정을 상속합니다. 클러스터 관리자는 **Operator** 서브스크립션을 구성하여 이러한 프록시 설정을 덮어쓸 수도 있습니다.



중요

Operator에서는 관리형 **Operand**에 대해 **Pod**의 프록시 설정을 위한 환경 변수 설정을 처리해야 합니다.

사전 요구 사항

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

절차

1. 웹 콘솔에서 **Operators → OperatorHub** 페이지로 이동합니다.
2. **Operator**를 선택하고 설치를 클릭합니다.
3. **Operator** 설치 페이지에서 **spec** 섹션에 다음 환경 변수를 하나 이상 포함하도록 **Subscription** 오브젝트를 수정합니다.
   - **HTTP_PROXY**
   - **HTTPS_PROXY**
   - **NO_PROXY**

   예를 들면 다음과 같습니다.

프록시 설정 덮어쓰기가 포함된 **Subscription** 오브젝트

```
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
```

참고

이러한 환경 변수는 이전에 설정한 클러스터 수준 또는 사용자 정의 프록시 설정을 제거하기 위해 빈 값을 사용하여 설정을 해제할 수도 있습니다.

OLM에서는 이러한 환경 변수를 단위로 처리합니다. 환경 변수가 한 개 이상 설정되어 있으면 세 개 모두 덮어쓰는 것으로 간주하며 구독한 Operator의 배포에 클러스터 수준 기본값이 사용되지 않습니다.

4. 선택한 네임스페이스에서 Operator를 사용할 수 있도록 설치를 클릭합니다.

5. Operator의 CSV가 관련 네임스페이스에 표시되면 사용자 정의 프록시 환경 변수가 배포에 설정되어 있는지 확인할 수 있습니다. 예를 들면 CLI를 사용합니다.

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

출력 예

```
      - name: HTTP_PROXY
        value: test_http
      - name: HTTPS_PROXY
        value: test_https
      - name: NO_PROXY
        value: test
      image: quay.io/coreos/etcd-
operator@sha256:66a37fd61a06a43969854ee6d3e21088a98b93838e284a6086b13917f96
b0d9c
...
```

4.5.2. 사용자 정의 CA 인증서 삽입

클러스터 관리자가 구성 맵을 사용하여 클러스터에 사용자 정의 **CA** 인증서를 추가하면 **Cluster Network Operator**는 사용자 제공 인증서와 시스템 **CA** 인증서를 단일 번들로 병합합니다. 이 병합된 번들은 **OLM(Operator Lifecycle Manager)**에서 실행 중인 **Operator**에 삽입할 수 있는데 이러한 작업은 중간자 **HTTPS** 프록시를 사용하는 경우 유용합니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- 구성 맵을 사용하여 사용자 정의 **CA** 인증서를 클러스터에 추가했습니다.
- 필요한 **Operator**가 **OLM**에 설치되어 실행되고 있습니다.

프로세스

1. **Operator**의 서브스크립션이 존재하고 다음 라벨을 포함하는 네임스페이스에 빈 구성 맵을 생성합니다.

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

-

1

구성 맵의 이름입니다.

2

***CNO(Cluster Network Operator)**에 병합된 번들을 삽입하도록 요청합니다.*

이 구성 맵이 생성되면 병합된 번들의 인증서 콘텐츠로 즉시 채워집니다.

2. ***Subscription** 오브젝트를 업데이트하여 사용자 정의 **CA**가 필요한 **Pod** 내의 각 컨테이너에 **trusted-ca** 구성 맵을 볼륨으로 마운트하는 **spec.config** 섹션을 포함합니다.*

```
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 섹션이 없는 경우 추가합니다.

2

*Operator에서 보유한 **Pod**와 일치하도록 라벨을 지정합니다.*

3

trusted-ca 볼륨을 생성합니다.

4

구성 맵 키로 ca-bundle.crt가 필요합니다.

5

구성 맵 경로로 tls-ca-bundle.pem이 필요합니다.

6

trusted-ca 볼륨 마운트를 생성합니다.

참고

Operator 배포는 기관을 검증하지 못하고 알 수 없는 기관 오류로 서명된 x509 인증서를 표시할 수 있습니다. 이 오류는 Operator 서브스크립션을 사용할 때 사용자 정의 CA를 삽입한 후에도 발생할 수 있습니다. 이 경우 Operator 서브스크립션을 사용하여 mountPath 를 trusted-ca의 /etc/ssl/certs 로 설정할 수 있습니다.

4.6. OPERATOR 상태 보기

OLM(Operator Lifecycle Manager)의 시스템 상태를 이해하는 것은 설치된 Operator와 관련된 결정을 내리고 문제를 디버깅하는 데 중요합니다. OLM에서는 서브스크립션 및 관련 카탈로그 소스의 상태 및 수행된 작업과 관련된 통찰력을 제공합니다. 이를 통해 사용자는 Operator의 상태를 더 잘 이해할 수 있습니다.

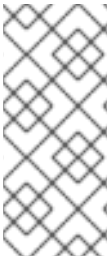
4.6.1. Operator 서브스크립션 상태 유형

서브스크립션은 다음 상태 유형을 보고할 수 있습니다.

표 4.1. 서브스크립션 상태 유형

상태	설명
CatalogSourcesUnhealthy	해결에 사용되는 일부 또는 모든 카탈로그 소스가 정상 상태가 아닙니다.
InstallPlanMissing	서브스크립션 설치 계획이 없습니다.

상태	설명
InstallPlanPending	서브스크립션 설치 계획이 설치 대기 중입니다.
InstallPlanFailed	서브스크립션 설치 계획이 실패했습니다.
ResolutionFailed	서브스크립션의 종속성 확인이 실패했습니다.



참고

기본 OpenShift Container Platform 클러스터 Operator는 CVO(Cluster Version Operator)에서 관리하며 Subscription 오브젝트가 없습니다. 애플리케이션 Operator는 OLM(Operator Lifecycle Manager)에서 관리하며 Subscription 오브젝트가 있습니다.

추가 리소스

- *실패한 서브스크립션 새로 고침*

4.6.2. CLI를 사용하여 Operator 서브스크립션 상태 보기

CLI를 사용하여 Operator 서브스크립션 상태를 볼 수 있습니다.

사전 요구 사항

- *cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.*
- *OpenShift CLI(oc)가 설치되어 있습니다.*

프로세스

1. *Operator 서브스크립션을 나열합니다.*

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

2. *oc describe 명령을 사용하여 Subscription 리소스를 검사합니다.*

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

3. 명령 출력에서 **Operator** 서브스크립션 조건 유형의 상태에 대한 **Conditions** 섹션을 확인합니다. 다음 예에서 사용 가능한 모든 카탈로그 소스가 정상이므로 **CatalogSourcesUnhealthy** 조건 유형의 상태가 **false**입니다.

출력 예

```
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
# ...
```

참고

기본 **OpenShift Container Platform** 클러스터 **Operator**는 **CVO(Cluster Version Operator)**에서 관리하며 **Subscription** 오브젝트가 없습니다. 애플리케이션 **Operator**는 **OLM(Operator Lifecycle Manager)**에서 관리하며 **Subscription** 오브젝트가 있습니다.

4.6.3. CLI를 사용하여 Operator 카탈로그 소스 상태 보기

CLI를 사용하여 **Operator** 카탈로그 소스의 상태를 볼 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **OpenShift CLI(oc)**가 설치되어 있습니다.

프로세스

1. 네임스페이스의 카탈로그 소스를 나열합니다. 예를 들어 클러스터 전체 카탈로그 소스에 사용되는 **openshift-marketplace** 네임스페이스를 확인할 수 있습니다.

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

출력 예

```
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-marketplace    Red Hat Marketplace   grpc   Red Hat     55m
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. **oc describe** 명령을 사용하여 카탈로그 소스에 대한 자세한 내용 및 상태를 가져옵니다.

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

출력 예

```
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
# ...
```

*앞의 예제 출력에서 마지막으로 관찰된 상태는 **TRANSIENT_FAILURE**입니다. 이 상태는 카탈로그 소스에 대한 연결을 설정하는 데 문제가 있음을 나타냅니다.*

3. *카탈로그 소스가 생성된 네임스페이스의 **Pod**를 나열합니다.*

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

출력 예

```
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-marketplace-57p8c            1/1     Running            0          36m
redhat-operators-smxx8              1/1     Running            0          36m
```

*카탈로그 소스가 네임스페이스에 생성되면 해당 네임스페이스에 카탈로그 소스의 **Pod**가 생성됩니다. 위 예제 출력에서 **example-catalog-bwt8z pod**의 상태는 **ImagePullBackOff**입니다. 이 상태는 카탈로그 소스의 인덱스 이미지를 가져오는 데 문제가 있음을 나타냅니다.*

4. *자세한 정보는 **oc describe** 명령을 사용하여 **Pod**를 검사합니다.*

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

출력 예

```
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
```

앞의 예제 출력에서 오류 메시지는 권한 부여 문제로 인해 카탈로그 소스의 인덱스 이미지를 성공적으로 가져오지 못한 것으로 표시됩니다. 예를 들어 인덱스 이미지는 로그인 인증 정보가 필요한 레지스트리에 저장할 수 있습니다.

추가 리소스

- Operator Lifecycle Manager 개념 및 리소스 → 카탈로그 소스
- gRPC 문서: 연결 상태
- 프라이빗 레지스트리에서 Operator용 이미지에 액세스

4.7. OPERATOR 조건 관리

클러스터 관리자는 OLM(Operator Lifecycle Manager)을 사용하여 Operator 상태를 관리할 수 있습니다.

4.7.1. Operator 상태 덮어쓰기

클러스터 관리자는 Operator에서 보고한 지원되는 Operator 상태를 무시해야 할 수 있습니다. 이러한 상태가 있는 경우 Spec.Overrides 어레이의 Operator 상태가 Spec.Conditions 어레이의 상태를 덮어씁

니다. 그러면 클러스터 관리자가 Operator에서 OLM(Operator Lifecycle Manager)에 상태를 잘못 보고하는 상황을 처리할 수 있습니다.

참고

기본적으로 클러스터 관리자가 추가할 때까지 spec .Overrides 배열이 OperatorCondition 오브젝트에 존재하지 않습니다. 사용자가 추가하거나 사용자 정의 Operator 논리의 결과로 Spec.Conditions 어레이도 존재하지 않습니다.

예를 들어 항상 업그레이드할 수 없다고 보고하는 알려진 버전의 Operator를 떠올려 보십시오. 이 경우 Operator에서 업그레이드할 수 없다고 보고하더라도 Operator를 업그레이드해야 할 수 있습니다. 이 작업은 OperatorCondition 오브젝트의 Spec.Overrides 어레이에 조건 유형 및 상태를 추가하여 Operator 조건을 덮어쓰는 방식으로 수행할 수 있습니다.

사전 요구 사항

- OLM을 사용하여 설치된 OperatorCondition 오브젝트가 있는 Operator입니다.

절차

1. Operator의 OperatorCondition 오브젝트를 편집합니다.

```
$ oc edit operatorcondition <name>
```

2. 오브젝트에 Spec.Overrides 어레이를 추가합니다.

Operator 조건 덮어쓰기 예제

```
apiVersion: operators.coreos.com/v1
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

클러스터 관리자는 업그레이드 준비 상태를 True로 변경할 수 있습니다.

4.7.2. Operator 조건을 사용하도록 Operator 업데이트

OLM(Operator Lifecycle Manager)은 OLM에서 조정하는 각 ClusterServiceVersion 리소스에 대해 OperatorCondition 리소스를 자동으로 생성합니다. CSV의 모든 서비스 계정에는 Operator에 속하는 OperatorCondition과 상호 작용할 수 있도록 RBAC가 부여됩니다.

Operator 작성자는 Operator가 OLM에 의해 배포된 후 operator-lib 라이브러리를 사용하여 자체 조건을 설정할 수 있도록 Operator를 개발할 수 있습니다. Operator 작성자로 Operator 조건을 설정하는 방법에 대한 자세한 내용은 Operator 조건 활성화 페이지를 참조하십시오.

4.7.2.1. 기본값 설정

OLM은 이전 버전과의 호환성을 유지하기 위해 OperatorCondition 리소스의 부재를 조건을 옵트아웃하는 것으로 처리합니다. 따라서 Operator 조건 사용에 옵트인하는 Operator는 Pod의 준비 상태 프로브를 true로 설정하기 전에 기본 조건을 설정해야 합니다. 그러면 Operator에 조건을 올바른 상태로 업데이트할 수 있는 유예 기간이 제공됩니다.

4.7.3. 추가 리소스

- Operator 상태

4.8. 비 클러스터 관리자가 OPERATOR를 설치하도록 허용

클러스터 관리자는 Operator group 을 사용하여 일반 사용자가 Operator를 설치할 수 있습니다.

추가 리소스

- *Operator groups*

4.8.1. Operator 설치 정책 이해

Operator를 실행하는 데 광범위한 권한이 필요할 수 있으며 필요한 권한이 버전에 따라 다를 수 있습니다. OLM(Operator Lifecycle Manager)은 cluster-admin 권한으로 실행됩니다. 기본적으로 Operator 작성자는 CSV(클러스터 서비스 버전)의 권한 세트를 지정할 수 있으며 OLM은 결과적으로 Operator에 권한을 부여합니다.

Operator가 클러스터 범위 권한을 얻을 수 없고 사용자가 OLM을 사용하여 권한을 상승시킬 수 없도록 클러스터 관리자가 Operator를 클러스터에 추가하기 전에 수동으로 감사할 수 있습니다. 클러스터 관리자에게는 서비스 계정을 사용하여 Operator를 설치 또는 업그레이드하는 동안 수행할 수 있는 작업을 결정하고 제한하는 툴도 제공됩니다.

클러스터 관리자는 Operator group을 일련의 권한이 부여된 서비스 계정과 연결할 수 있습니다. 서비스 계정은 역할 기반 액세스 제어(RBAC) 규칙을 사용하여 사전 정의된 범위 내에서만 실행되도록 Operator에 정책을 설정합니다. 결과적으로 Operator는 해당 규칙에서 명시적으로 허용하지 않는 작업을 수행할 수 없습니다.

Operator 그룹을 사용하면 충분한 권한이 있는 사용자는 범위가 제한된 Operator를 설치할 수 있습니다. 결과적으로 더 많은 사용자가 더 많은 Operator 프레임워크 툴을 안전하게 사용할 수 있으므로 Operator를 사용하여 애플리케이션을 빌드하는 보다 풍부한 환경을 제공할 수 있습니다.

참고

Subscription 오브젝트에 대한 RBAC(역할 기반 액세스 제어)는 네임스페이스에서 edit 또는 admin 역할이 있는 모든 사용자에게 자동으로 부여됩니다. 그러나 RBAC는 OperatorGroup 오브젝트에 존재하지 않습니다. 이는 일반 사용자가 Operator를 설치하지 못하도록 하는 것입니다. Operator group을 사전 설치하는 것은 효과적으로 설치 권한을 부여하는 것입니다.

Operator group을 서비스 계정과 연결할 때 다음 사항을 고려하십시오.

- *APIService 및 CustomResourceDefinition 리소스는 항상 cluster-admin 역할을 사용하여 OLM에 의해 생성됩니다. Operator group과 연결된 서비스 계정에는 이러한 리소스를 작성할 수 있는 권한을 부여해서는 안 됩니다.*

- *이제 Operator group에 연결된 모든 Operator의 권한이 지정된 서비스 계정에 부여된 권한*

으로 제한됩니다. Operator에서 서비스 계정 범위를 벗어나는 권한을 요청하는 경우 클러스터 관리자가 문제를 해결하고 해결할 수 있도록 설치가 실패하고 적절한 오류가 발생합니다.

4.8.1.1. 설치 시나리오

Operator를 클러스터에서 설치하거나 업그레이드할 수 있는지 결정하는 경우 OLM(Operator Lifecycle Manager)은 다음 시나리오를 고려합니다.

- 클러스터 관리자가 새 Operator group을 생성하고 서비스 계정을 지정합니다. 이 Operator group과 연결된 모든 Operator가 설치되고 서비스 계정에 부여된 권한에 따라 실행됩니다.
- 클러스터 관리자가 새 Operator group을 생성하고 서비스 계정을 지정하지 않습니다. OpenShift Container Platform은 이전 버전과의 호환성을 유지하므로 기본 동작은 그대로 유지되면서 Operator 설치 및 업그레이드가 허용됩니다.
- 서비스 계정을 지정하지 않는 기존 Operator group의 경우 기본 동작은 그대로 유지되면서 Operator 설치 및 업그레이드가 허용됩니다.
- 클러스터 관리자가 기존 Operator group을 업데이트하고 서비스 계정을 지정합니다. OLM을 사용하면 현재 권한을 사용하여 기존 Operator를 계속 실행될 수 있습니다. 이러한 기존 Operator에서 업그레이드를 수행하면 기존 Operator가 새 Operator와 같이 서비스 계정에 부여된 권한에 따라 다시 설치되어 실행됩니다.
- 권한을 추가하거나 제거함으로써 Operator group에서 지정하는 서비스 계정이 변경되거나 기존 서비스 계정을 새 서비스 계정과 교체합니다. 기존 Operator에서 업그레이드를 수행하면 기존 Operator가 새 Operator와 같이 업데이트된 서비스 계정에 부여된 권한에 따라 다시 설치되어 실행됩니다.
- 클러스터 관리자는 Operator group에서 서비스 계정을 제거합니다. 기본 동작은 유지되고 Operator 설치 및 업그레이드는 허용됩니다.

4.8.1.2. 설치 워크플로

Operator group이 서비스 계정에 연결되고 Operator가 설치 또는 업그레이드되면 OLM(Operator Lifecycle Manager)에서 다음과 같은 워크플로를 사용합니다.

1. OLM에서 지정된 Subscription 오브젝트를 선택합니다.

2. *OLM에서 이 서브스크립션에 연결된 Operator group을 가져옵니다.*

3. *OLM에서 Operator group에 서비스 계정이 지정되었는지 확인합니다.*

4. *OLM에서 서비스 계정에 대한 클라이언트 범위를 생성하고 범위가 지정된 클라이언트를 사용하여 Operator를 설치합니다. 이렇게 하면 Operator에서 요청한 모든 권한이 항상 Operator group 서비스 계정의 권한으로 제한됩니다.*

5. *OLM은 CSV에 지정된 권한 세트를 사용하여 새 서비스 계정을 생성하고 Operator에 할당합니다. Operator는 할당된 서비스 계정으로 실행됩니다.*

4.8.2. Operator 설치 범위 지정

OLM(Operator Lifecycle Manager)의 Operator 설치 및 업그레이드에 대한 범위 지정 규칙을 제공하려면 서비스 계정을 Operator group에 연결합니다.

클러스터 관리자는 다음 예제를 통해 일련의 Operator를 지정된 네임스페이스로 제한할 수 있습니다.

프로세스

1. *새 네임스페이스를 생성합니다.*

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: scoped
EOF
```

2. *Operator를 제한할 권한을 할당합니다. 이를 위해서는 새 서비스 계정, 관련 역할, 역할 바인딩을 생성해야 합니다.*

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: scoped
  namespace: scoped
EOF
```

다음 예제에서는 간소화를 위해 지정된 네임스페이스에서 모든 작업을 수행할 수 있는 서비스 계정 권한을 부여합니다. 프로덕션 환경에서는 더 세분화된 권한 세트를 생성해야 합니다.

```
$ cat <<EOF | oc create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: scoped
  namespace: scoped
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: scoped-bindings
  namespace: scoped
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: scoped
subjects:
- kind: ServiceAccount
  name: scoped
  namespace: scoped
EOF
```

3.

지정된 네임스페이스에 **OperatorGroup** 오브젝트를 생성합니다. 이 **Operator group**은 지정된 네임스페이스를 대상으로 하여 테넌시가 제한되도록 합니다.

또한 **Operator group**에서는 사용자가 서비스 계정을 지정할 수 있습니다. 이전 단계에서 생성한 서비스 계정을 지정합니다.

```
$ cat <<EOF | oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: scoped
  namespace: scoped
spec:
  serviceAccountName: scoped
  targetNamespaces:
  - scoped
EOF
```

지정된 네임스페이스에 설치된 **Operator**는 모두 이 **Operator group** 및 지정된 서비스 계정에 연결됩니다.

주의

OLM(Operator Lifecycle Manager)은 각 Operator 그룹에 대해 다음 클러스터 역할을 생성합니다.

- <operatorgroup_name>-admin
- <operatorgroup_name>-edit
- <operatorgroup_name>-view

Operator group을 수동으로 생성할 때 클러스터의 기존 클러스터 역할 또는 기타 Operator 그룹과 충돌하지 않는 고유한 이름을 지정해야 합니다.

4. 지정된 네임스페이스에 Subscription 오브젝트를 생성하여 Operator를 설치합니다.

```
$ cat <<EOF | oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: etcd
  namespace: scoped
spec:
  channel: singlenamespace-alpha
  name: etcd
  source: <catalog_source_name> 1
  sourceNamespace: <catalog_source_namespace> 2
EOF
```

1 지정된 네임스페이스에 이미 존재하는 카탈로그 소스 또는 글로벌 카탈로그 네임스페이스에 있는 카탈로그 소스를 지정합니다.

2

카탈로그 소스가 생성된 네임스페이스를 지정합니다.

Operator group에 연결된 모든 Operator의 권한이 지정된 서비스 계정에 부여된 권한으로 제한됩니다. Operator에서 서비스 계정 외부에 있는 권한을 요청하는 경우 설치가 실패하고 관련 오류가 표시됩니다.

4.8.2.1. 세분화된 권한

OLM(Operator Lifecycle Manager)은 Operator group에 지정된 서비스 계정을 사용하여 설치 중인 Operator와 관련하여 다음 리소스를 생성하거나 업데이트합니다.

- ClusterServiceVersion
- 서브스크립션
- Secret
- ServiceAccount
- Service
- ClusterRole 및 ClusterRoleBinding
- Role 및 RoleBinding

Operator를 지정된 네임스페이스로 제한하려면 클러스터 관리자가 서비스 계정에 다음 권한을 부여하여 시작하면 됩니다.

참고

다음 역할은 일반적인 예이며 특정 **Operator**를 기반으로 추가 규칙이 필요할 수 있습니다.

```
kind: Role
rules:
- apiGroups: ["operators.coreos.com"]
  resources: ["subscriptions", "clusterserviceversions"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: [""]
  resources: ["services", "serviceaccounts"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["get", "create", "update", "patch"]
- apiGroups: ["apps"] 1
  resources: ["deployments"]
  verbs: ["list", "watch", "get", "create", "update", "patch", "delete"]
- apiGroups: [""] 2
  resources: ["pods"]
  verbs: ["list", "watch", "get", "create", "update", "patch", "delete"]
```

1 2

여기에 표시된 배포 및 **Pod**와 같은 기타 리소스를 생성하는 권한을 추가합니다.

또한 **Operator**에서 가져오기 보안을 지정하는 경우 다음 권한도 추가해야 합니다.

```
kind: ClusterRole 1
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
---
kind: Role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "update", "patch"]
```

1

OLM 네임스페이스에서 보안을 가져오는 데 필요합니다.

4.8.3. Operator 카탈로그 액세스 제어

글로벌 카탈로그 네임스페이스 **openshift-marketplace** 에서 **Operator** 카탈로그가 생성되면 모든 네임스페이스에서 카탈로그의 **Operator**를 클러스터 전체에서 사용할 수 있습니다. 다른 네임 스페이스에서 생성된 카탈로그는 해당 **Operator**를 카탈로그의 동일한 네임스페이스에서만 사용할 수 있도록 합니다.

비 클러스터 관리자 사용자가 **Operator** 설치 권한을 위임한 클러스터에서 클러스터 관리자는 해당 사용자가 설치할 수 있는 **Operator** 세트를 추가로 제어하거나 제한할 수 있습니다. 이 작업은 다음 작업을 통해 수행할 수 있습니다.

1. 모든 기본 글로벌 카탈로그를 비활성화합니다.

2. 관련 **Operator** 그룹이 사전 설치된 동일한 네임스페이스에서 사용자 정의 큐레이션 카탈로그를 활성화합니다.

추가 리소스

- 기본 **OperatorHub** 소스 비활성화
- 클러스터에 카탈로그 소스 추가

4.8.4. 권한 장애 문제 해결

권한 부족으로 인해 **Operator** 설치가 실패하는 경우 다음 절차를 사용하여 오류를 확인합니다.

프로세스

1. **Subscription** 오브젝트를 검토합니다. 해당 상태에는 **Operator**에 필요한 **[Cluster]Role[Binding]** 오브젝트를 생성하는 **InstallPlan** 오브젝트를 가리키는 오브젝트 참조 **installPlanRef**가 있습니다.

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: etcd
  namespace: scoped
status:
  installPlanRef:
    apiVersion: operators.coreos.com/v1
    kind: InstallPlan
    name: install-4plp8
```

```
  namespace: scoped
  resourceVersion: "117359"
  uid: 2c1df80e-afea-11e9-bce3-5254009c9c23
```

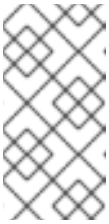
2. *InstallPlan 오브젝트의 상태에 오류가 있는지 확인합니다.*

```
apiVersion: operators.coreos.com/v1
kind: InstallPlan
status:
 conditions:
 - lastTransitionTime: "2019-07-26T21:13:10Z"
   lastUpdateTime: "2019-07-26T21:13:10Z"
   message: 'error creating clusterrole etcdoperator.v0.9.4-clusterwide-dsfx4:
clusterroles.rbac.authorization.k8s.io
     is forbidden: User "system:serviceaccount:scoped:scoped" cannot create resource
     "clusterroles" in API group "rbac.authorization.k8s.io" at the cluster scope'
   reason: InstallComponentFailed
   status: "False"
   type: Installed
 phase: Failed
```

오류 메시지에는 다음이 표시됩니다.

- *리소스의 **API** 그룹을 포함하여 생성할 수 없는 리소스 유형. 이 경우 **rbac.authorization.k8s.io** 그룹의 **clusterroles**입니다.*
- *리소스의 이름.*
- *오류 유형 **is forbidden**은 사용자에게 작업을 수행할 수 있는 권한이 충분하지 않음을 나타냅니다.*
- *리소스를 생성하거나 업데이트하려고 시도한 사용자의 이름. 이 경우 **Operator group**에 지정된 서비스 계정을 나타냅니다.*
- *작업 범위: **cluster scope** 여부*

사용자는 서비스 계정에 누락된 권한을 추가한 다음 다시 수행할 수 있습니다.

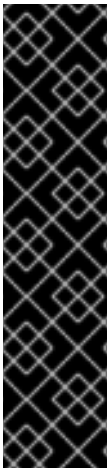


참고

OLM(Operator Lifecycle Manager)은 현재 첫 번째 시도에서 전체 오류 목록을 제공하지 않습니다.

4.9. 사용자 정의 카탈로그 관리

클러스터 관리자 및 Operator 카탈로그 관리자는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)에서 번들 형식을 사용하여 패키지된 사용자 정의 카탈로그를 생성하고 관리할 수 있습니다.



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 버전과의 Operator 호환성 제어를 참조하십시오.

추가 리소스

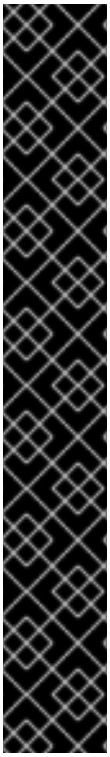
- Red Hat 제공 Operator 카탈로그

4.9.1. 사전 요구 사항

- opm CLI 를 설치합니다.

4.9.2. 파일 기반 카탈로그

파일 기반 카탈로그는 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

OpenShift Container Platform 4.11부터 기본 **Red Hat** 제공 **Operator** 카탈로그는 파일 기반 카탈로그 형식으로 제공됩니다. 더 이상 사용되지 않는 **SQLite** 데이터베이스 형식으로 릴리스된 **4.10**을 통한 **OpenShift Container Platform 4.6**의 기본 **Red Hat** 제공 **Operator** 카탈로그입니다.

SQLite 데이터베이스 형식과 관련된 **opm** 하위 명령, 플래그 및 기능은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 기능은 계속 지원되며 더 이상 사용되지 않는 **SQLite** 데이터베이스 형식을 사용하는 카탈로그에 사용해야 합니다.

opm index prune 와 같은 **SQLite** 데이터베이스 형식을 사용하기 위한 많은 **opm** 하위 명령과 플래그는 파일 기반 카탈로그 형식으로 작동하지 않습니다. 파일 기반 카탈로그 사용에 대한 자세한 내용은 **oc-mirror** 플러그인을 사용하여 **Operator Framework** 패키징 형식 및 연결이 끊긴 설치의 이미지 미러링 을 참조하십시오.

4.9.2.1. 파일 기반 카탈로그 이미지 생성

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

사전 요구 사항

- **opm**
- **podman** 버전 **1.9.3+**
- 번들 이미지를 빌드하여 **Docker v2-2**를 지원하는 레지스트리로 내보냄

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:v4.11 (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

```
. (1)
├── <catalog_dir> (2)
└── <catalog_dir>.Dockerfile (3)
```

(1) *상위 디렉터리*

(2) *카탈로그 디렉터리*

(3) *opm generate dockerfile 명령으로 생성된 Dockerfile*

c.

opm init 명령을 실행하여 Operator의 패키지 정의로 카탈로그를 채웁니다.

```
$ 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 또는 패키지, 이름*

2 *서브스크립션이 지정되지 않은 경우 기본값으로 설정된 채널*

3 *Operator의 README.md 또는 기타 문서의 경로입니다.*

4 *Operator 아이콘 경로*

5 *출력 형식: JSON 또는 YAML*

6 *카탈로그 구성 파일 생성 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 pull spec*

2

카탈로그 구성 파일의 경로입니다.

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.

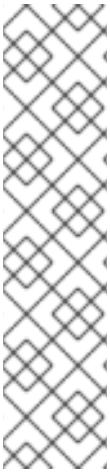
```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

추가 리소스

- opm CLI 참조

4.9.2.2. 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 추출 및 수정하여 카탈로그에서 하나 이상의 Operator 패키지 항목을 업데이트, 추가 또는 제거할 수 있습니다. 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

- **opm CLI.**
- **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. 하나 이상의 **Operator** 패키지 항목을 업데이트, 추가 또는 제거하여 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.

*예를 들어 **Operator** 패키지를 제거하려면 다음 예제에 **olm.package,olm.channel, olm.bundle blobs** 세트가 나열되어 있으며 카탈로그에서 패키지를 제거하려면 삭제해야 합니다.*

예 4.1. 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
  ---
```

3. *index.yaml 파일에 변경 사항을 저장합니다.*

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

- **oc-mirror** 플러그인을 사용하여 연결이 끊긴 설치의 이미지 미러링 → 미러 레지스트리 콘텐츠 유지
- 클러스터에 카탈로그 소스 추가

4.9.3. SQLite 기반 카탈로그

중요

Operator 카탈로그의 **SQLite** 데이터베이스 형식은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 **OpenShift Container Platform**에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 **OpenShift Container Platform** 릴리스 노트에서 더 이상 사용되지 않고 삭제된 기능 섹션을 참조하십시오.

4.9.3.1. SQLite 기반 인덱스 이미지 생성

opm CLI를 사용하여 **SQLite** 데이터베이스 형식을 기반으로 인덱스 이미지를 생성할 수 있습니다.

사전 요구 사항

- **opm**

- podman 버전 1.9.3+
- 번들 이미지를 빌드하여 Docker v2-2를 지원하는 레지스트리로 내보냄

프로세스

1. 새 인덱스를 시작합니다.

```
$ 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. 인덱스에 추가할 번들 이미지를 쉼표로 구분한 목록입니다.
2. 인덱스 이미지에 포함할 이미지 태그입니다.
3. 선택 사항: 카탈로그 제공에 사용할 대체 레지스트리 기본 이미지입니다.

2. 인덱스 이미지를 레지스트리로 내보냅니다.

   a. 필요한 경우 대상 레지스트리로 인증합니다.

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

   b. 인덱스 이미지를 내보냅니다.

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

4.9.3.2. SQLite 기반 인덱스 이미지 업데이트

사용자 정의 인덱스 이미지를 참조하는 카탈로그 소스를 사용하도록 OperatorHub를 구성하면 클러

스터 관리자가 인덱스 이미지에 번들 이미지를 추가하여 클러스터에 사용 가능한 **Operator**를 최신 상태로 유지할 수 있습니다.

opm index add 명령을 사용하여 기존 인덱스 이미지를 업데이트할 수 있습니다.

사전 요구 사항

- **opm**
- **podman 버전 1.9.3+**
- 인덱스 이미지를 빌드하여 레지스트리로 내보냈습니다.
- 기존 카탈로그 소스에서 인덱스 이미지를 참조합니다.

절차

1. 번들 이미지를 추가하여 기존 인덱스를 업데이트합니다.

```
$ 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** 플래그는 인덱스에 추가할 추가 번들 이미지의 쉼표로 구분된 목록을 지정합니다.
2. **--from-index** 플래그는 이전에 내보낸 인덱스를 지정합니다.
3. **--tag** 플래그는 업데이트된 인덱스 이미지에 적용할 이미지 태그를 지정합니다.
4.

다음과 같습니다.

<registry>

레지스트리의 호스트 이름(예: **quay.io** 또는 **mirror.example.com**)을 지정합니다.

<namespace>

레지스트리의 네임스페이스(예: **ocs-dev** 또는 **abc**)를 지정합니다.

<new_bundle_image>

레지스트리에 추가할 새 번들 이미지를 지정합니다(예: **ocs-operator**).

<digest>

번들 이미지의 **SHA** 이미지 **ID** 또는 다이제스트를 지정합니다(예: **c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a41**).

<existing_index_image>

이전에 내보낸 이미지를 **abc-redhat-operator-index** 와 같이 지정합니다.

<existing_tag>

이전에 내보낸 이미지 태그(예: **4.11**)를 지정합니다.

<updated_tag>

업데이트된 인덱스 이미지에 적용할 이미지 태그를 지정합니다 (예: **4.11.1**).

명령 예

```
$ opm index add \
   --bundles quay.io/ocs-dev/ocs-
operator@sha256:c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef6
83a41 \
   --from-index mirror.example.com/abc/abc-redhat-operator-index:4.11 \
   --tag mirror.example.com/abc/abc-redhat-operator-index:4.11.1 \
   --pull-tool podman
```

2. 업데이트된 인덱스 이미지를 내보냅니다.

```
$ podman push <registry>/<namespace>/<existing_index_image>:<updated_tag>
```

3. OLM(Operator Lifecycle Manager)이 정기적으로 카탈로그 소스에서 참조하는 인덱스 이미지를 자동으로 폴링하면 새 패키지가 성공적으로 추가되었는지 확인합니다.

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

4.9.3.3. SQLite 기반 인덱스 이미지 필터링

Operator 번들 포맷을 기반으로 하는 인덱스 이미지는 Operator 카탈로그의 컨테이너화된 스냅샷입니다. 원하는 Operator만 포함하는 소스 인덱스 복사본을 생성하는 패키지 목록을 제외한 모든 인덱스의 인덱스를 필터링하거나 정리 할 수 있습니다.

사전 요구 사항

- podman 버전 1.9.3+
- grpcurl (제3자 명령줄 도구)
- opm
- Docker v2-2를 지원하는 레지스트리에 액세스

절차

1. 대상 레지스트리로 인증합니다.

```
$ podman login <target_registry>
```

2. 정리된 인덱스에 포함하려는 패키지 목록을 결정합니다.

a. 컨테이너에서 정리하려는 소스 인덱스 이미지를 실행합니다. 예를 들면 다음과 같습니다.

```
$ podman run -p50051:50051 \
    -it registry.redhat.io/redhat/redhat-operator-index:v4.11
```

출력 예

```
Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4.11...
Getting image source signatures
Copying blob ae8a0c23f5b1 done
...
INFO[0000] serving registry                 database=/database/index.db
port=50051
```

b. 별도의 터미널 세션에서 **grpcurl** 명령을 사용하여 인덱스에서 제공하는 패키지 목록을 가져옵니다.

```
$ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
```

c. **packages.out** 파일을 검사하고 정리된 인덱스에 보관할 이 목록에 있는 패키지 이름을 확인합니다. 예를 들면 다음과 같습니다.

패키지 목록 조각의 예

```
...
{
  "name": "advanced-cluster-management"
}
...
{
  "name": "jaeger-product"
}
...
{
{
  "name": "quay-operator"
}
...
```

d. *podman run 명령을 실행한 터미널 세션에서 Ctrl 및 C를 눌러 컨테이너 프로세스를 중지합니다.*

3. *다음 명령을 실행하여 지정된 패키지를 제외한 소스 인덱스를 모두 정리합니다.*

```
$ opm index prune \
  -f registry.redhat.io/redhat/redhat-operator-index:v4.11 \ 1
  -p advanced-cluster-management,jaeger-product,quay-operator \ 2
  [-i registry.redhat.io/openshift4/ose-operator-registry:v4.9] \ 3
  -t <target_registry>:<port>/<namespace>/redhat-operator-index:v4.11 4
```

1. *정리할 인덱스입니다.*

2. *쉼표로 구분된 보관할 패키지 목록입니다.*

3. *IBM Power 및 IBM Z 이미지에만 필요합니다. 대상 OpenShift Container Platform 클러스터 주 버전 및 부 버전과 일치하는 태그를 사용하여 Operator 레지스트리 기본 이미지입니다.*

4. *빌드 중인 새 인덱스 이미지에 대한 사용자 정의 태그입니다.*

4. *다음 명령을 실행하여 새 인덱스 이미지를 대상 레지스트리로 내보냅니다.*

```
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4.11
```

<namespace>는 레지스트리의 기존 네임스페이스입니다.

4.9.4. 클러스터에 카탈로그 소스 추가

OpenShift Container Platform 클러스터에 카탈로그 소스를 추가하면 사용자를 위한 Operator를 검색하고 설치할 수 있습니다. 클러스터 관리자는 인덱스 이미지를 참조하는 CatalogSource 오브젝트를 생성할 수 있습니다. OperatorHub는 카탈로그 소스를 사용하여 사용자 인터페이스를 채웁니다.

작은 정보

또는 웹 콘솔을 사용하여 카탈로그 소스를 관리할 수 있습니다. 관리 → 클러스터 설정 → 구성 → OperatorHub 페이지에서 개별 소스 를 생성, 업데이트, 삭제, 비활성화 및 활성화할 수 있는 소스 탭을 클릭합니다.

사전 요구 사항

- 인덱스 이미지를 빌드하여 레지스트리로 내보냈습니다.

절차

1. 인덱스 이미지를 참조하는 CatalogSource 오브젝트를 생성합니다.

   a. 다음을 사양에 맞게 수정하고 catalogsource.yaml 파일로 저장합니다.

```
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
  image: <registry>/<namespace>/<index_image_name>:<tag> (3)
  displayName: My Operator Catalog
  publisher: <publisher_name> (4)
  updateStrategy:
    registryPoll: (5)
      interval: 30m
```

(1) 카탈로그 소스를 모든 네임스페이스의 사용자가 전역적으로 사용할 수 있도록 하려면 openshift-marketplace 네임스페이스를 지정합니다. 그러지 않으면 카탈로그의 범위가 지정되고 해당 네임스페이스에 대해서만 사용할 수 있도록 다른 네임스페이스를 지정할 수 있습니다.

2

*선택 사항: **olm.catalogImageTemplate** 주석을 인덱스 이미지 이름으로 설정하고 이미지 태그의 템플릿을 구성할 때 표시된 대로 하나 이상의 **Kubernetes** 클러스터 버전 변수를 사용합니다.*

3

*인덱스 이미지를 지정합니다. 이미지 이름 뒤에 태그를 지정하는 경우 (예: **:v4.11**) 카탈로그 소스 **Pod**는 **Always** 라는 이미지 가져오기 정책을 사용합니다. 즉, **Pod**는 컨테이너를 시작하기 전에 항상 이미지를 가져옵니다. 다이제스트(예: **@sha256:<id >**)를 지정하는 경우 이미지 가져오기 정책은 **IfNotPresent** 입니다. 즉, 노드에 아직 존재하지 않는 경우에만 **Pod**에서 이미지를 가져옵니다.*

4

카탈로그를 게시하는 이름 또는 조직 이름을 지정합니다.

5

카탈로그 소스는 새 버전을 자동으로 확인하여 최신 상태를 유지할 수 있습니다.

b. *파일을 사용하여 **CatalogSource** 오브젝트를 생성합니다.*

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

2. *다음 리소스가 성공적으로 생성되었는지 확인합니다.*

a. ***Pod**를 확인합니다.*

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

출력 예

```
NAME                                READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6           1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. 카탈로그 소스를 확인합니다.

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

출력 예

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc           5s
```

c. 패키지 매니페스트 확인합니다.

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

출력 예

```
NAME                  CATALOG               AGE
jaeger-product        My Operator Catalog   93s
```

이제 OpenShift Container Platform 웹 콘솔의 OperatorHub 페이지에서 Operator를 설치할 수 있습니다.

추가 리소스

- Operator Lifecycle Manager 개념 및 리소스 → 카탈로그 소스
- 프라이빗 레지스트리에서 Operator용 이미지에 액세스
- 이미지 가져오기 정책

4.9.5. 프라이빗 레지스트리에서 Operator용 이미지에 액세스

OLM(Operator Lifecycle Manager)에서 관리하는 Operator와 관련된 특정 이미지가 개인 레지스트리라고도 하는 인증된 컨테이너 이미지 레지스트리에서 호스팅되는 경우 OLM 및 OperatorHub는 기본적으로 이미지를 가져올 수 없습니다. 액세스를 활성화하려면 레지스트리의 인증 정보가 포함된 풀 시크릿을 생성할 수 있습니다. OLM은 카탈로그 소스에서 하나 이상의 가져오기 보안을 참조함으로써 Operator 및 카탈로그 네임스페이스에 보안을 배치하여 설치를 허용할 수 있습니다.

Operator 또는 해당 Operand에 필요한 기타 이미지에서도 프라이빗 레지스트리에 액세스해야 할 수 있습니다. 이 시나리오에서 OLM은 대상 테넌트 네임스페이스에 보안을 배치하지 않지만 필요한 액세스 권한을 활성화하기 위해 글로벌 클러스터 가져오기 보안 또는 개별 네임스페이스 서비스 계정에 인증 자격 증명을 추가할 수 있습니다.

OLM에서 관리하는 Operator에 적절한 가져오기 액세스 권한이 있는지 확인할 때는 다음 유형의 이미지를 고려해야 합니다.

인덱스 이미지

CatalogSource 오브젝트는 Operator 번들 형식을 사용하고 이미지 레지스트리에서 호스팅되는 컨테이너 이미지로 패키지된 카탈로그 소스에 해당하는 인덱스 이미지를 참조할 수 있습니다. 인덱스 이미지가 프라이빗 레지스트리에서 호스팅되는 경우 시크릿을 사용하여 가져오기 액세스 권한을 활성화할 수 있습니다.

번들 이미지

Operator 번들 이미지는 컨테이너 이미지로 패키지되어 Operator의 고유 버전을 나타내는 메타데이터와 매니페스트입니다. 카탈로그 소스에서 참조한 번들 이미지가 하나 이상의 프라이빗 레지스트리에서 호스팅되는 경우 보안을 사용하여 가져오기 액세스 권한을 활성화할 수 있습니다.

Operator 및 Operand 이미지

카탈로그 소스에서 설치한 Operator에서 Operator 이미지 자체 또는 조사하는 Operand 이미지 중 하나로 프라이빗 이미지를 사용하는 경우 Operator가 설치되지 않습니다. 배포에 필수 레지스트리 인증에 대한 액세스 권한이 없기 때문입니다. 카탈로그 소스의 보안을 참조해도 OLM에서 Operand가 설치된 대상 테넌트 네임스페이스에 보안을 배치할 수 없습니다.

대신 클러스터의 모든 네임스페이스에 대한 액세스 권한을 제공하는 openshift-config 네임스페이스의 글로벌 클러스터 가져오기 보안에 인증 세부 정보를 추가하면 됩니다. 또는 전체 클러스터에 대한 액세스 권한을 제공할 수 없는 경우 대상 테넌트 네임스페이스의 default 서비스 계정에 가져오기 보안을 추가할 수 있습니다.

사전 요구 사항

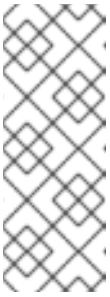
- 다음 중 한 개 이상이 프라이빗 레지스트리에서 호스팅됩니다.
  - 인덱스 이미지 또는 카탈로그 이미지.
  - **Operator** 번들 이미지.
  - **Operator** 또는 **Operand** 이미지.

프로세스

1. 필요한 각 프라이빗 레지스트리에 대해 보안을 생성합니다.

   a. 프라이빗 레지스트리에 로그인하여 레지스트리 자격 증명 파일을 생성하거나 업데이트합니다.

   ```
   $ podman login <registry>:<port>
   ```

   참고

   레지스트리 자격 증명의 파일 경로는 레지스트리에 로그인하는 데 사용하는 컨테이너 툴에 따라 다를 수 있습니다. **podman CLI**의 경우 기본 위치는 **${XDG_RUNTIME_DIR}/containers/auth.json**입니다. **docker CLI**의 경우 기본 위치는 **/root/.docker/config.json**입니다.

   b. 보안당 하나의 레지스트리에 대한 자격 증명만 포함하고 여러 레지스트리의 자격 증명은 별도의 보안에서 관리하는 것이 좋습니다. 이후 단계에서 **CatalogSource** 오브젝트에 여러 보안을 포함할 수 있으며 **OpenShift Container Platform**은 이미지를 가져오는 동안 사용할 수 있도록 보안을 단일 가상 자격 증명 파일에 병합합니다.

   기본적으로 레지스트리 자격 증명 파일은 둘 이상의 레지스트리 또는 하나의 레지스트리에 있는 여러 리포지토리의 세부 정보를 저장할 수 있습니다. 파일의 현재 콘텐츠를 확인합니다. 예를 들면 다음과 같습니다.

   여러 레지스트리에 대한 자격 증명을 저장하는 파일

```
{
  "auths": {
    "registry.redhat.io": {
      "auth": "FrNHNydQXdzclNqdg=="
    },
    "quay.io": {
      "auth": "fegdsRib21iMQ=="
    },
    "https://quay.io/my-namespace/my-user/my-image": {
      "auth": "eWfjwsDdfsa221=="
    },
    "https://quay.io/my-namespace/my-user": {
      "auth": "feFweDdscw34rR=="
    },
    "https://quay.io/my-namespace": {
      "auth": "frwEews4fescyq=="
    }
  }
}
```

이 파일은 이후 단계에서 보안을 생성하는 데 사용되므로 파일마다 하나의 레지스트리에만 세부 정보를 저장해야 합니다. 이 작업은 다음 방법 중 하나를 사용하여 수행할 수 있습니다.

- **podman logout <registry>** 명령을 사용하여 원하는 레지스트리 하나만 남을 때까지 추가 레지스트리의 자격 증명을 제거합니다.

- 레지스트리 자격 증명 파일을 편집하고 레지스트리 세부 정보를 분리하여 여러 파일에 저장합니다. 예를 들면 다음과 같습니다.

하나의 레지스트리에 대한 자격 증명을 저장하는 파일

```
{
    "auths": {
        "registry.redhat.io": {
            "auth": "FrNHNydQXdzclNqdg=="
        }
    }
}
```

또 다른 레지스트리에 대한 자격 증명을 저장하는 파일

```
{
        "auths": {
                "quay.io": {
                        "auth": "Xd2lhdsbnRib21iMQ=="
                }
        }
}
```

c.

*프라이빗 레지스트리의 인증 자격 증명 정보가 포함된 **openshift-marketplace** 네임스페이스에 보안을 생성합니다.*

```
$ oc create secret generic <secret_name> \
    -n openshift-marketplace \
    --from-file=.dockerconfigjson=<path/to/registry/credentials> \
    --type=kubernetes.io/dockerconfigjson
```

*이 단계를 반복하여 다른 필수 프라이빗 레지스트리에 대한 추가 보안을 생성하고 **--from-file** 플래그를 업데이트하여 다른 레지스트리 자격 증명 파일 경로를 지정합니다.*

2.

*기존 **CatalogSource** 오브젝트를 생성하거나 업데이트하여 하나 이상의 보안을 참조합니다.*

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  secrets: 1
  - "<secret_name_1>"
  - "<secret_name_2>"
  image: <registry>:<port>/<namespace>/<image>:<tag>
  displayName: My Operator Catalog
  publisher: <publisher_name>
  updateStrategy:
    registryPoll:
      interval: 30m
```

1

spec.secrets 섹션을 추가하고 필요한 보안을 지정합니다.

3. 구독한 **Operator**에서 참조하는 **Operator** 또는 **Operand** 이미지에 프라이빗 레지스트리에 대한 액세스 권한이 필요한 경우 클러스터의 모든 네임스페이스 또는 개별 대상 테넌트 네임스페이스에 액세스 권한을 제공하면 됩니다.

- 클러스터의 모든 네임스페이스에 액세스 권한을 제공하려면 **openshift-config** 네임스페이스의 글로벌 클러스터 가져오기 보안에 인증 세부 정보를 추가합니다.

> 주의
>
> 클러스터 리소스는 클러스터의 사용성을 일시적으로 제한할 수 있는 새 글로벌 가져오기 보안에 맞게 조정해야 합니다.

a. 글로벌 가져오기 보안에서 **.dockerconfigjson** 파일을 추출합니다.

```
$ oc extract secret/pull-secret -n openshift-config --confirm
```

b. 필요한 프라이빗 레지스트리 또는 레지스트리에 대한 인증 자격 증명을 사용하여 **.dockerconfigjson** 파일을 업데이트한 후 새 파일로 저장합니다.

```
$ cat .dockerconfigjson | \
    jq --compact-output '.auths["<registry>:<port>/<namespace>/"] |= . +
{"auth":"<token>"}' \ 1
    > new_dockerconfigjson
```

1 **<registry>:<port>/<namespace>**를 프라이빗 레지스트리 세부 정보로 교체하고 **<token>**을 인증 자격 증명으로 교체합니다.

c. 새 파일을 사용하여 글로벌 가져오기 보안을 업데이트합니다.

```
$ oc set data secret/pull-secret -n openshift-config \
  --from-file=.dockerconfigjson=new_dockerconfigjson
```

- 개별 네임스페이스를 업데이트하려면 대상 테넌트 네임스페이스에 액세스해야 하는 Operator의 서비스 계정에 가져오기 보안을 추가합니다.

a. 테넌트 네임스페이스에서 openshift-marketplace용으로 생성한 보안을 다시 생성합니다.

```
$ oc create secret generic <secret_name> \
  -n <tenant_namespace> \
  --from-file=.dockerconfigjson=<path/to/registry/credentials> \
  --type=kubernetes.io/dockerconfigjson
```

b. 테넌트 네임스페이스를 검색하여 Operator의 서비스 계정 이름을 확인합니다.

```
$ oc get sa -n <tenant_namespace> 1
```

1 Operator가 개별 네임스페이스에 설치된 경우 해당 네임스페이스를 검색합니다. Operator가 모든 네임스페이스에 설치된 경우 openshift-operators 네임스페이스를 검색합니다.

출력 예

```
NAME            SECRETS   AGE
builder         2         6m1s
default         2         6m1s
deployer        2         6m1s
etcd-operator   2         5m18s 1
```

1 설치된 etcd Operator의 서비스 계정입니다.

c. Operator의 서비스 계정에 보안을 연결합니다.

```
$ oc secrets link <operator_sa> \
   -n <tenant_namespace> \
    <secret_name> \
   --for=pull
```

추가 리소스

- 레지스트리 자격 증명에 사용되는 보안 유형을 포함하여 보안 유형에 대한 자세한 내용은 보안이란? 을 참조하십시오.
- 이 시크릿 변경이 미치는 영향에 대한 자세한 내용은 글로벌 클러스터 풀 시크릿 업데이트를 참조하십시오.
- 가져오기 보안을 네임스페이스별 서비스 계정에 연결하는 방법에 대한 자세한 내용은 Pod에서 다른 보안 레지스트리의 이미지를 참조하도록 허용 을 참조하십시오.

4.9.6. 기본 OperatorHub 소스 비활성화

Red Hat 및 커뮤니티 프로젝트에서 제공하는 콘텐츠를 소싱하는 Operator 카탈로그는 OpenShift Container Platform을 설치하는 동안 기본적으로 OperatorHub용으로 구성됩니다. 클러스터 관리자는 기본 카탈로그 세트를 비활성화할 수 있습니다.

프로세스

- OperatorHub 오브젝트에 disableAllDefaultSources: true를 추가하여 기본 카탈로그의 소스를 비활성화합니다.

```
$ oc patch OperatorHub cluster --type json \
   -p '[{"op": "add", "path": "/spec/disableAllDefaultSources", "value": true}]'
```

작은 정보

또는 웹 콘솔을 사용하여 카탈로그 소스를 관리할 수 있습니다. 관리 → 클러스터 설정 → 구성 → OperatorHub 페이지에서 개별 소스 를 생성, 업데이트, 삭제, 비활성화 및 활성화할 수 있는 소스 탭을 클릭합니다.

4.9.7. 사용자 정의 카탈로그 제거

클러스터 관리자는 관련 카탈로그 소스를 삭제하여 이전에 클러스터에 추가된 사용자 정의 **Operator** 카탈로그를 제거할 수 있습니다.

프로세스

1. 웹 콘솔의 관리자 화면에서 관리자 → 클러스터 설정으로 이동합니다.
2. 구성 탭을 클릭한 다음 **OperatorHub** 를 클릭합니다.
3. 소스 탭을 클릭합니다.
4. 제거할 카탈로그의 옵션 메뉴

를 선택한 다음 카탈로그 소스 삭제를 클릭합니다.

4.10. 제한된 네트워크에서 OPERATOR LIFECYCLE MANAGER 사용

제한된 네트워크(연결이 끊긴 클러스터)에 설치된 **OpenShift Container Platform** 클러스터의 경우 **OLM(Operator Lifecycle Manager)**은 기본적으로 원격 레지스트리에서 호스팅되는 **Red Hat** 제공 **OperatorHub** 소스에 액세스할 수 없습니다. 이러한 원격 소스에는 완전한 인터넷 연결이 필요하기 때문입니다.

그러나 클러스터 관리자는 워크스테이션에 완전한 인터넷 액세스 권한이 있는 경우 제한된 네트워크에서 **OLM**을 사용하도록 클러스터를 활성화할 수 있습니다. 원격 **OperatorHub** 콘텐츠를 가져오는 데 완전한 인터넷 액세스 권한이 필요한 워크스테이션은 원격 소스의 로컬 미러를 준비하고 콘텐츠를 미러 레지스트리로 내보내는 데 사용됩니다.

미러 레지스트리는 워크스테이션 및 연결이 끊긴 클러스터 모두에 연결해야 하는 베스천 호스트에 있거나 미러링된 콘텐츠를 연결이 끊긴 환경에 물리적으로 이동하기 위해 이동식 미디어가 필요한 완전히 연결이 끊긴 호스트 또는 에어갭(**Airgap**) 호스트에 있을 수 있습니다.

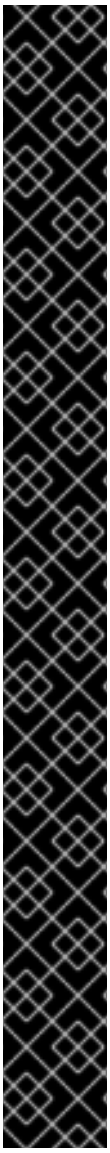
이 가이드에서는 제한된 네트워크에서 **OLM**을 활성화하는 데 필요한 다음 프로세스를 설명합니다.

- **OLM**의 기본 원격 **OperatorHub** 소스를 비활성화합니다.

- 완전한 인터넷 액세스가 가능한 워크스테이션을 사용하여 **OperatorHub** 콘텐츠의 로컬 미러를 생성하고 미러 레지스트리로 내보냅니다.

- 기본 원격 소스가 아닌 미러 레지스트리의 로컬 소스에서 **Operator**를 설치하고 관리하도록 **OLM**을 구성합니다.

제한된 네트워크에서 **OLM**을 활성화한 후에는 제한되지 않은 워크스테이션을 계속 사용하여 최신 버전의 **Operator**가 출시될 때 로컬 **OperatorHub** 소스를 업데이트할 수 있습니다.



중요

OLM은 로컬 소스에서 **Operator**를 관리할 수 있지만 지정된 **Operator**를 제한된 네트워크에서 성공적으로 실행할 수 있는 기능은 다음 기준을 충족하는 **Operator** 자체에 따라 다릅니다.

- **Operator**에서 기능을 수행하는 데 필요할 수 있는 관련 이미지 또는 기타 컨테이너 이미지를 **CSV(ClusterServiceVersion)** 오브젝트의 **relatedImages** 매개변수에 나열합니다.

- 태그가 아닌 다이제스트(**SHA**)를 통해 지정된 모든 이미지를 참조합니다.

다음 선택 사항으로 필터링하여 **Red Hat Ecosystem Catalog** 에서 소프트웨어를 검색하여 연결이 끊긴 모드에서 실행을 지원하는 **Red Hat Operator** 목록을 검색할 수 있습니다.

유형	컨테이너화된 애플리케이션
배포 방법	Operator
인프라 기능	disconnected

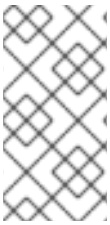
추가 리소스

- **Red Hat 제공 Operator 카탈로그**

- *제한된 네트워크 환경에 대한 Operator 활성화*

4.10.1. 사전 요구 사항

- cluster-admin 권한이 있는 사용자로 OpenShift Container Platform 클러스터에 로그인합니다.



참고

IBM Z의 제한된 네트워크에서 OLM을 사용하는 경우 레지스트리를 배치하는 디렉터리에 12GB 이상을 할당해야 합니다.

4.10.2. 기본 OperatorHub 소스 비활성화

Red Hat 및 커뮤니티 프로젝트에서 제공하는 콘텐츠를 소싱하는 Operator 카탈로그는 OpenShift Container Platform을 설치하는 동안 기본적으로 OperatorHub용으로 구성됩니다. 제한된 네트워크 환경에서는 클러스터 관리자로서 기본 카탈로그를 비활성화해야 합니다. 그런 다음 로컬 카탈로그 소스를 사용하도록 OperatorHub를 구성할 수 있습니다.

프로세스

- OperatorHub 오브젝트에 disableAllDefaultSources: true를 추가하여 기본 카탈로그의 소스를 비활성화합니다.

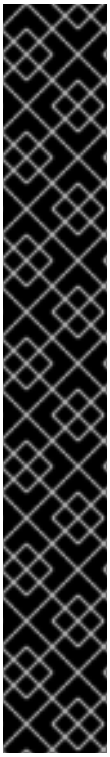
```
$ oc patch OperatorHub cluster --type json \
    -p '[{"op": "add", "path": "/spec/disableAllDefaultSources", "value": true}]'
```

작은 정보

또는 웹 콘솔을 사용하여 카탈로그 소스를 관리할 수 있습니다. 관리 → 클러스터 설정 → 구성 → OperatorHub 페이지에서 개별 소스 를 생성, 업데이트, 삭제, 비활성화 및 활성화할 수 있는 소스 탭을 클릭합니다.

4.10.3. Operator 카탈로그 미러링

연결이 끊긴 클러스터와 함께 사용할 수 있도록 Operator 카탈로그를 미러링하는 방법에 대한 자세한 내용은 연결이 *끊긴 설치를 위한 이미지 설치 → 미러링을* 참조하십시오.



중요

OpenShift Container Platform 4.11부터 기본 Red Hat 제공 Operator 카탈로그는 파일 기반 카탈로그 형식으로 제공됩니다. 더 이상 사용되지 않는 SQLite 데이터베이스 형식으로 릴리스된 4.10을 통한 OpenShift Container Platform 4.6의 기본 Red Hat 제공 Operator 카탈로그입니다.

SQLite 데이터베이스 형식과 관련된 opm 하위 명령, 플래그 및 기능은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 기능은 계속 지원되며 더 이상 사용되지 않는 SQLite 데이터베이스 형식을 사용하는 카탈로그에 사용해야 합니다.

opm index prune 와 같은 SQLite 데이터베이스 형식을 사용하기 위한 많은 opm 하위 명령과 플래그는 파일 기반 카탈로그 형식으로 작동하지 않습니다. 파일 기반 카탈로그 사용에 대한 자세한 내용은 Operator Framework 패키징 형식,사용자 정의 카탈로그 관리, oc-mirror 플러그인을 사용하여 연결이 끊긴 설치용 이미지 미러링을 참조하십시오.

4.10.4. 클러스터에 카탈로그 소스 추가

OpenShift Container Platform 클러스터에 카탈로그 소스를 추가하면 사용자를 위한 Operator를 검색하고 설치할 수 있습니다. 클러스터 관리자는 인덱스 이미지를 참조하는 CatalogSource 오브젝트를 생성할 수 있습니다. OperatorHub는 카탈로그 소스를 사용하여 사용자 인터페이스를 채웁니다.

작은 정보

또는 웹 콘솔을 사용하여 카탈로그 소스를 관리할 수 있습니다. 관리 → 클러스터 설정 → 구성 → OperatorHub 페이지에서 개별 소스 를 생성, 업데이트, 삭제, 비활성화 및 활성화할 수 있는 소스 탭을 클릭합니다.

사전 요구 사항

- 인덱스 이미지를 빌드하여 레지스트리로 내보냈습니다.

프로세스

1. 인덱스 이미지를 참조하는 CatalogSource 오브젝트를 생성합니다. oc adm catalog mirror 명령을 사용하여 카탈로그를 대상 레지스트리에 미러링한 경우 매니페스트 디렉터리에서 생성된 catalogSource.yaml 파일을 시작점으로 사용할 수 있습니다.

   a. 다음을 사양에 맞게 수정하고 catalogsource.yaml 파일로 저장합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog (1)
  namespace: openshift-marketplace (2)
spec:
  sourceType: grpc
  image: <registry>/<namespace>/redhat-operator-index:v4.11 (3)
  displayName: My Operator Catalog
  publisher: <publisher_name> (4)
  updateStrategy:
    registryPoll: (5)
      interval: 30m
```

(1) 레지스트리에 업로드하기 전에 콘텐츠를 로컬 파일에 미러링한 경우 오브젝트를 생성할 때 "잘못된 리소스 이름" 오류가 발생하지 않도록 **metadata.name** 필드에서 백슬래시(/) 문자를 제거합니다.

(2) 카탈로그 소스를 모든 네임스페이스의 사용자가 전역적으로 사용할 수 있도록 하려면 **openshift-marketplace** 네임스페이스를 지정합니다. 그러지 않으면 카탈로그의 범위가 지정되고 해당 네임스페이스에 대해서만 사용할 수 있도록 다른 네임스페이스를 지정할 수 있습니다.

(3) 인덱스 이미지를 지정합니다. 이미지 이름 뒤에 태그를 지정하는 경우 (예: **:v4.11**) 카탈로그 소스 **Pod**는 **Always** 라는 이미지 가져오기 정책을 사용합니다. 즉, **Pod**는 컨테이너를 시작하기 전에 항상 이미지를 가져옵니다. 다이제스트(예: **@sha256:<id >**)를 지정하는 경우 이미지 가져오기 정책은 **IfNotPresent** 입니다. 즉, 노드에 아직 존재하지 않는 경우에만 **Pod**에서 이미지를 가져옵니다.

(4) 카탈로그를 게시하는 이름 또는 조직 이름을 지정합니다.

(5) 카탈로그 소스는 새 버전을 자동으로 확인하여 최신 상태를 유지할 수 있습니다.

b. 파일을 사용하여 **CatalogSource** 오브젝트를 생성합니다.

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

2. *다음 리소스가 성공적으로 생성되었는지 확인합니다.*

a. ***Pod**를 확인합니다.*

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

출력 예

```
NAME                                READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6           1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. *카탈로그 소스를 확인합니다.*

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

출력 예

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc           5s
```

c. *패키지 매니페스트 확인합니다.*

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

출력 예

```
NAME                  CATALOG               AGE
jaeger-product                My Operator Catalog   93s
```

이제 OpenShift Container Platform 웹 콘솔의 OperatorHub 페이지에서 Operator를 설치할 수 있습니다.

추가 리소스

- 프라이빗 레지스트리에서 Operator용 이미지에 액세스
- 사용자 정의 카탈로그 소스의 이미지 템플릿
- 이미지 가져오기 정책

4.11. 카탈로그 소스 POD 예약

소스 유형의 OLM(Operator Lifecycle Manager) 카탈로그 소스 grpc 가 spec.image 를 정의하면 Catalog Operator는 정의된 이미지 콘텐츠를 제공하는 Pod를 생성합니다. 기본적으로 이 Pod는 사양에 다음을 정의합니다.

- kubernetes.io/os=linux 노드 선택기만
- 우선순위 클래스 이름 없음
- 허용 오차 없음

관리자는 CatalogSource 오브젝트의 선택적 spec.grpcPodConfig 섹션의 필드를 수정하여 이러한 값을 덮어쓸 수 있습니다.

추가 리소스

- OLM 개념 및 리소스 → 카탈로그 소스

4.11.1. 카탈로그 소스 Pod의 노드 선택기 덮어쓰기

전제 조건

- *spec.image 가 정의된 source type grpc 의 CatalogSource 오브젝트*

프로세스

- *CatalogSource 오브젝트를 편집하고 다음을 포함하도록 spec.grpcPodConfig 섹션을 추가하거나 수정합니다.*

```
grpcPodConfig:
  nodeSelector:
    custom_label: <label>
```

여기서 < label>은 카탈로그 소스 Pod가 예약에 사용할 노드 선택기의 레이블입니다.

추가 리소스

- *노드 선택기를 사용하여 특정 노드에 Pod 배치*

4.11.2. 카탈로그 소스 Pod의 우선순위 클래스 이름 덮어쓰기

전제 조건

- *spec.image 가 정의된 source type grpc 의 CatalogSource 오브젝트*

프로세스

- *CatalogSource 오브젝트를 편집하고 다음을 포함하도록 spec.grpcPodConfig 섹션을 추가하거나 수정합니다.*

```
grpcPodConfig:
  priorityClassName: <priority_class>
```

여기서 <priority_class >는 다음 중 하나입니다.

  - *Kubernetes에서 제공하는 기본 우선순위 클래스 중 하나: system-cluster-critical 또는 system-node-critical*

- 기본 우선 순위를 할당하는 빈 세트("")
- 기존 및 사용자 지정 우선순위 클래스

참고

이전에는 재정의될 수 있는 유일한 **Pod** 예약 매개변수는 **priorityClassName** 이었습니다. 이 작업은 **CatalogSource** 오브젝트에 **operatorframework.io/priorityclass** 주석을 추가하여 수행됩니다. 예를 들면 다음과 같습니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    operatorframework.io/priorityclass: system-cluster-critical
```

CatalogSource 오브젝트에서 주석과 **spec.grpcPodConfig.priorityClassName** 을 모두 정의하는 경우 주석은 구성 매개변수보다 우선합니다.

추가 리소스

- **Pod** 우선순위 클래스

4.11.3. 카탈로그 소스 Pod에 대한 허용 오차 덮어쓰기

전제 조건

- **spec.image** 가 정의된 **source type grpc** 의 **CatalogSource** 오브젝트

프로세스

- **CatalogSource** 오브젝트를 편집하고 다음을 포함하도록 **spec.grpcPodConfig** 섹션을 추가하거나 수정합니다.

```
grpcPodConfig:
  tolerations:
    - key: "<key_name>"
```

```
      operator: "<operator_type>"
      value: "<value>"
      effect: "<effect>"
```

추가 리소스

- *테인트(Taints) 및 톨러레이션(Tolerations)의 이해*

5장. OPERATOR 개발

5.1. OPERATOR SDK 정보

Operator 프레임워크는 Operator라는 Kubernetes 네이티브 애플리케이션을 효율적이고 확장 가능하며 자동화된 방식으로 관리하는 오픈 소스 툴킷입니다. Operator는 Kubernetes 확장성을 활용하여 클라우드 서비스(예: 프로비저닝, 스케일링, 백업, 복원) 자동화의 이점을 제공하는 동시에 Kubernetes를 실행할 수 있는 모든 위치에서 실행할 수 있습니다.

Operator를 사용하면 Kubernetes 상의 복잡한 상태 저장 애플리케이션을 쉽게 관리할 수 있습니다. 그러나 현재는 하위 API 사용, 상용구 작성, 중복으로 이어지는 모듈성 부족과 같은 문제로 인해 Operator를 작성하기 어려울 수 있습니다.

Operator 프레임워크의 구성 요소인 Operator SDK는 Operator 개발자가 Operator를 빌드, 테스트, 배포하는 데 사용할 수 있는 CLI(명령줄 인터페이스) 툴을 제공합니다.

Operator SDK를 사용하는 이유는 무엇입니까?

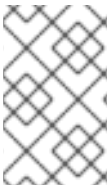
Operator SDK는 Kubernetes 네이티브 애플리케이션 구축 프로세스를 간소화하는데 이를 위해서는 애플리케이션별 운영 지식이 필요할 수 있습니다. Operator SDK를 사용하면 이러한 문제가 완화될 뿐만 아니라 미터링 또는 모니터링과 같이 다수의 공통 관리 기능에 필요한 상용구 코드의 양을 줄이는 데 도움이 됩니다.

Operator SDK는 controller-runtime 라이브러리를 사용하여 다음과 같은 기능을 제공함으로써 Operator를 더 쉽게 작성할 수 있는 프레임워크입니다.

- 운영 논리를 더 직관적으로 작성할 수 있는 고급 API 및 추상화
- 새 프로젝트를 신속하게 부트스트랩하기 위한 스캐폴드 및 코드 생성 툴
- OLM(Operator Lifecycle Manager) 통합을 통해 클러스터에서 수행되는 Operator 패키지, 설치, 실행 작업 간소화
- 일반적인 Operator 사용 사례를 포함하는 확장 기능

- Prometheus Operator가 배포된 클러스터에서 사용할 수 있도록 생성되는 모든 Go 기반 Operator에 자동으로 설정된 지표

Kubernetes 기반 클러스터(예: OpenShift Container Platform)에 대한 클러스터 관리자 액세스 권한이 있는 Operator 작성자는 Operator SDK CLI를 사용하여 Go, Ansible 또는 Helm을 기반으로 자체 Operator를 개발할 수 있습니다. Kubebuilder는 Go 기반 Operator의 스캐폴드 솔루션으로 Operator SDK에 포함되어 있습니다. 즉 기존 Kubebuilder 프로젝트를 그대로 Operator SDK와 함께 사용할 수 있으며 계속 작업할 수 있습니다.



참고

OpenShift Container Platform 4.11은 Operator SDK v1.22.2를 지원합니다.

5.1.1. Operator란 무엇인가?

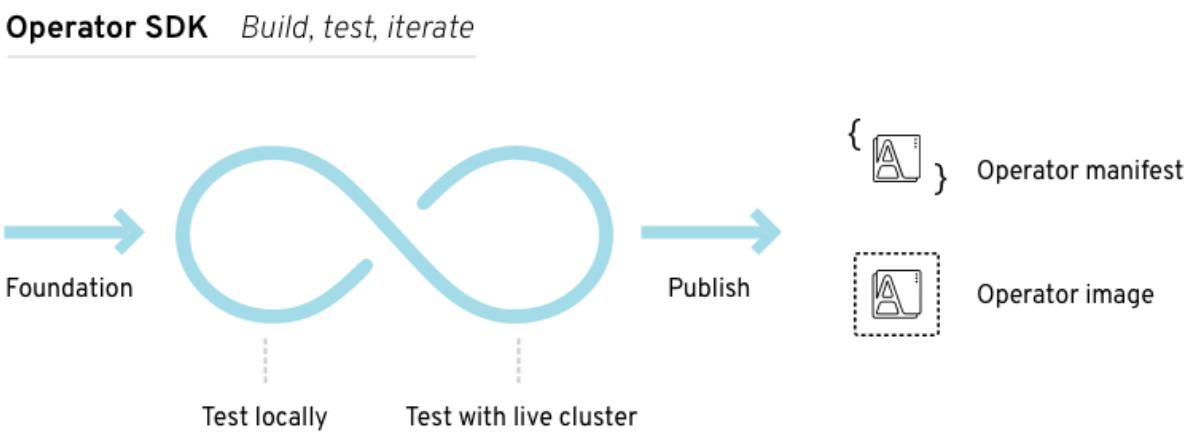
기본 Operator 개념 및 용어에 대한 개요는 Operator 이해를 참조하십시오.

5.1.2. 개발 워크플로

Operator SDK는 다음 워크플로를 제공하여 새 Operator를 개발합니다.

1. Operator SDK CLI(명령줄 인터페이스)를 사용하여 Operator 프로젝트를 생성합니다.
2. CRD(사용자 정의 리소스 정의)를 추가하여 새 리소스 API를 정의합니다.
3. Operator SDK API를 사용하여 조사할 리소스를 지정합니다.
4. 지정된 핸들러에서 Operator 조정 논리를 정의하고 Operator SDK API를 사용하여 리소스와 상호 작용합니다.
5. Operator SDK CLI를 사용하여 Operator 배포 매니페스트를 빌드하고 생성합니다.

그림 5.1. Operator SDK 워크플로



Operator SDK를 사용하는 Operator는 Operator 작성자가 정의한 핸들러에서 조사하는 리소스에 대한 이벤트를 처리하고 애플리케이션 상태를 조정하는 작업을 수행합니다.

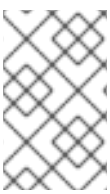
5.1.3. 추가 리소스

- 인증된 Operator 빌드 가이드

5.2. OPERATOR SDK CLI 설치

Operator SDK는 Operator 개발자가 Operator를 빌드, 테스트, 배포하는 데 사용할 수 있는 CLI(명령줄 인터페이스) 툴을 제공합니다. 워크스테이션에 Operator SDK CLI를 설치하여 자체 Operator를 작성할 준비를 할 수 있습니다.

OpenShift Container Platform과 같은 Kubernetes 기반 클러스터에 대한 클러스터 관리자 액세스 권한이 있는 Operator 작성자는 Operator SDK CLI를 사용하여 Go, Ansible 또는 Helm을 기반으로 자체 Operator를 개발할 수 있습니다. Kubebuilder는 Go 기반 Operator의 스캐폴드 솔루션으로 Operator SDK에 포함되어 있습니다. 즉 기존 Kubebuilder 프로젝트를 그대로 Operator SDK와 함께 사용할 수 있으며 계속 작업할 수 있습니다.



참고

OpenShift Container Platform 4.11은 Operator SDK v1.22.2를 지원합니다.

5.2.1. Operator SDK CLI 설치

Linux에 OpenShift SDK CLI 툴을 설치할 수 있습니다.

사전 요구 사항

- Go v1.18+
- docker v17.03 이상, podman v1.9.3 이상 또는 buildah v1.7 이상

프로세스

1. OpenShift 미러 사이트로 이동합니다.
2. 최신 4.11 디렉터리에서 최신 버전의 Linux용 tarball을 다운로드합니다.
3. 아카이브의 압축을 풉니다.

```
$ tar xvf operator-sdk-v1.22.2-ocp-linux-x86_64.tar.gz
```

4. 파일을 실행 가능으로 설정합니다.

```
$ chmod +x operator-sdk
```

5. 추출된 operator-sdk 바이너리를 PATH에 있는 디렉터리로 이동합니다.

작은 정보

PATH를 확인하려면 다음을 실행합니다.

```
$ echo $PATH
```

```
$ sudo mv ./operator-sdk /usr/local/bin/operator-sdk
```

검증

- Operator SDK CLI를 설치한 후 사용할 수 있는지 확인합니다.

```
$ operator-sdk version
```

출력 예

```
operator-sdk version: "v1.22.2-ocp", ...
```

5.3. GO 기반 OPERATOR

5.3.1. Go 기반 Operator를 위한 Operator SDK 시작하기

Operator 개발자는 Operator SDK에서 제공하는 툴 및 라이브러리를 사용하여 Go 기반 Operator를 설정 및 실행하는 기본 동작을 설명하기 위해 분산형 키-값 저장소인 Memcached에 대한 Go 기반 Operator 예제를 빌드하고 클러스터에 배포할 수 있습니다.

5.3.1.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Go v1.18+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- *Operator SDK CLI 설치*
- *OpenShift CLI 시작하기*

5.3.1.2. Go 기반 Operator 생성 및 배포

Operator SDK를 사용하여 Memcached에 대한 간단한 Go 기반 Operator를 빌드하고 배포할 수 있습니다.

절차

1. *프로젝트를 생성합니다.*
   a. *프로젝트 디렉토리를 생성합니다.*

```
$ mkdir memcached-operator
```

   b. *프로젝트 디렉터리로 변경합니다.*

```
$ cd memcached-operator
```

   c. *operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.*

```
$ operator-sdk init \
    --domain=example.com \
    --repo=github.com/example-inc/memcached-operator
```

   명령은 기본적으로 Go 플러그인을 사용합니다.

2. *API를 생성합니다.*

   간단한 Memcached API를 생성합니다.

```
$ operator-sdk create api \
    --resource=true \
    --controller=true \
```

```
--group cache \
--version v1 \
--kind Memcached
```

3. Operator 이미지를 빌드하여 내보냅니다.

기본 Makefile 대상을 사용하여 Operator를 빌드하고 내보냅니다. 내보낼 수 있는 레지스트리를 사용하는 이미지의 가져오기 사양에 IMG를 설정합니다.

```
$ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

4. Operator를 실행합니다.

a. CRD를 설치합니다.

```
$ make install
```

b. 클러스터에 프로젝트를 배포합니다. 내보낸 이미지에 IMG를 설정합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

5. 샘플 CR(사용자 정의 리소스)을 생성합니다.

a. 샘플 CR을 생성합니다.

```
$ oc apply -f config/samples/cache_v1_memcached.yaml \
    -n memcached-operator-system
```

b. CR에서 Operator를 조정하는지 확인합니다.

```
$ oc logs deployment.apps/memcached-operator-controller-manager \
    -c manager \
    -n memcached-operator-system
```

6. CR 삭제

다음 명령을 실행하여 CR을 삭제합니다.

```
$ oc delete -f config/samples/cache_v1_memcached -n memcached-operator-system
```

7. 정리합니다.

다음 명령을 실행하여 이 절차의 일부로 생성된 리소스를 정리합니다.

```
$ make undeploy
```

5.3.1.3. 다음 단계

- Go 기반 Operator를 빌드하는 방법에 대한 자세한 내용은 Go 기반 Operator를 위한 Operator SDK 튜토리얼 을 참조하십시오.

5.3.2. Go 기반 Operator를 위한 Operator SDK 튜토리얼

Operator 개발자는 Operator SDK의 Go 프로그래밍 언어 지원을 활용하여 분산형 키-값 저장소인 Memcached에 대한 Go 기반 Operator 예제를 빌드하고 라이프사이클을 관리할 수 있습니다.

이 프로세스는 Operator 프레임워크의 두 가지 주요 요소를 사용하여 수행됩니다.

Operator SDK

operator-sdk CLI 툴 및 controller-runtime 라이브러리 API

OLM(Operator Lifecycle Manager)

클러스터에 대한 Operator의 설치, 업그레이드, RBAC(역할 기반 액세스 제어)

참고

이 튜토리얼에는 Go 기반 Operator용 Operator SDK 시작하기보다 자세히 설명되어 있습니다.

5.3.2.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Go v1.18+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치
- OpenShift CLI 시작하기

5.3.2.2. 프로젝트 생성

Operator SDK CLI를 사용하여 memcached-operator라는 프로젝트를 생성합니다.

프로세스

1. 프로젝트에 사용할 디렉터리를 생성합니다.

```
$ mkdir -p $HOME/projects/memcached-operator
```

2. 디렉터리로 변경합니다.

```
$ cd $HOME/projects/memcached-operator
```

3. Go 모듈에 대한 지원을 활성화합니다.

```
$ export GO111MODULE=on
```

4. operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.

```
$ operator-sdk init \
  --domain=example.com \
  --repo=github.com/example-inc/memcached-operator
```

참고

operator-sdk init 명령은 기본적으로 Go 플러그인을 사용합니다.

operator-sdk init 명령은 Go 모듈과 함께 사용할 go.mod 파일을 생성합니다. $GOPATH/src/ 외부에서 프로젝트를 생성할 때는 생성된 파일에 유효한 모듈 경로가 필요하기 때문에 --repo 플래그가 있어야 합니다.

5.3.2.2.1. PROJECT 파일

operator-sdk init 명령으로 생성된 파일 중에는 Kubebuilder PROJECT 파일이 있습니다. 이어서 프로젝트 루트에서 실행되는 operator-sdk 명령과 help 출력에서는 이 파일을 읽고 프로젝트 유형이 Go 임을 확인합니다. 예를 들면 다음과 같습니다.

```
domain: example.com
layout:
- go.kubebuilder.io/v3
projectName: memcached-operator
repo: github.com/example-inc/memcached-operator
version: "3"
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
```

5.3.2.2.2. Manager 정보

Operator의 기본 프로그램은 Manager를 초기화하고 실행하는 main.go 파일입니다. Manager는 모든 CR(사용자 정의 리소스) API 정의에 대한 스키마를 자동으로 등록하고 컨트롤러 및 Webhook를 설정 및 실행합니다.

Manager는 모든 컨트롤러에서 리소스를 조사하는 네임스페이스를 제한할 수 있습니다.

```
mgr, err := ctrl.NewManager(cfg, manager.Options{Namespace: namespace})
```

기본적으로 Manager는 Operator가 실행되는 네임스페이스를 조사합니다. 모든 네임스페이스를 조사하려면 namespace 옵션을 비워두면 됩니다.

```
mgr, err := ctrl.NewManager(cfg, manager.Options{Namespace: ""})
```

MultiNamespacedCacheBuilder 기능을 사용하여 특정 네임스페이스 세트를 조사할 수도 있습니다.

```
var namespaces []string (1)
mgr, err := ctrl.NewManager(cfg, manager.Options{ (2)
  NewCache: cache.MultiNamespacedCacheBuilder(namespaces),
})
```

(1) 네임스페이스 목록입니다.

(2) Cmd 구조를 생성하여 공유 종속 항목을 제공하고 구성 요소를 시작합니다.

5.3.2.2.3. 다중 그룹 API 정보

API 및 컨트롤러를 생성하기 전에 Operator에 여러 API 그룹이 필요한지 확인하도록 합니다. 이 튜토리얼에서는 단일 그룹 API의 기본 사례를 다루지만 프로젝트의 레이아웃을 변경하여 다중 그룹 API를 지원하려면 다음 명령을 실행하면 됩니다.

```
$ operator-sdk edit --multigroup=true
```

이 명령은 다음 예와 유사한 PROJECT 파일을 업데이트합니다.

```
domain: example.com
layout: go.kubebuilder.io/v3
multigroup: true
...
```

다중 그룹 프로젝트의 경우 API Go 유형 파일은 apis/<group>/<version>/ 디렉터리에 생성되고 컨트롤러는 controllers/<group>/ 디렉터리에 생성됩니다. 그런 다음 Dockerfile이 적절하게 업데이트됩니다.

추가 리소스

- 다중 그룹 프로젝트로 마이그레이션하는 방법에 대한 자세한 내용은 Kubebuilder 설명서를 참조하십시오.

5.3.2.3. API 및 컨트롤러 생성

Operator SDK CLI를 사용하여 CRD(사용자 정의 리소스 정의) API 및 컨트롤러를 생성합니다.

절차

1. 다음 명령을 실행하여 그룹이 cache이고 버전이 v1, 종류가 Memcached인 API를 생성합니다.

```
$ operator-sdk create api \
  --group=cache \
  --version=v1 \
  --kind=Memcached
```

2. 메시지가 표시되면 리소스 및 컨트롤러 모두 생성하도록 y를 입력합니다.

```
Create Resource [y/n]
y
Create Controller [y/n]
y
```

출력 예

```
Writing scaffold for you to edit...
api/v1/memcached_types.go
controllers/memcached_controller.go
...
```

이 프로세스는 api/v1/memcached_types.go에 Memcached 리소스 API를 생성하고 controllers/memcached_controller.go에 컨트롤러를 생성합니다.

5.3.2.3.1. API 정의

Memcached CR(사용자 정의 리소스)의 API를 정의합니다.

절차

1. spec 및 status가 다음과 같도록 api/v1/memcached_types.go에서 Go 유형 정의를 수정합니다.

```
// MemcachedSpec defines the desired state of Memcached
type MemcachedSpec struct {
 // +kubebuilder:validation:Minimum=0
 // Size is the size of the memcached deployment
 Size int32 `json:"size"`
}

// MemcachedStatus defines the observed state of Memcached
type MemcachedStatus struct {
 // Nodes are the names of the memcached pods
 Nodes []string `json:"nodes"`
}
```

2. 리소스 유형에 대해 생성된 코드를 업데이트합니다.

```
$ make generate
```

작은 정보

*_types.go 파일을 수정한 후에는 make generate 명령을 실행하여 해당 리소스 유형에 대해 생성된 코드를 업데이트해야 합니다.

위의 Makefile 대상은 controller-gen 유틸리티를 호출하여 api/v1/zz_generated.deepcopy.go 파일을 업데이트합니다. 이렇게 하면 API Go 유형 정의에서 모든 종류의 유형에서 구현해야 하는 runtime.Object 인터페이스를 구현할 수 있습니다.

5.3.2.3.2. CRD 매니페스트 생성

spec 및 status 필드 그리고 CRD(사용자 정의 리소스 정의) 검증 마커를 사용하여 API를 정의한 후에는 CRD 매니페스트를 생성할 수 있습니다.

절차

- 다음 명령을 실행하여 CRD 매니페스트를 생성하고 업데이트합니다.

```
$ make manifests
```

이 Makefile 대상은 controller-gen 유틸리티를 호출하여 config/crd/bases/cache.example.com_memcacheds.yaml 파일에서 CRD 매니페스트를 생성합니다.

5.3.2.3.2.1. OpenAPI 검증 정보

매니페스트가 생성될 때 spec.validation 블록의 CRD 매니페스트에 OpenAPIv3 스키마가 추가됩니다. 이 검증 블록을 사용하면 Memcached CR(사용자 정의 리소스)을 생성하거나 업데이트할 때 Kubernetes에서 해당 속성을 검증할 수 있습니다.

마커 또는 주석을 사용하여 API에 대한 검증을 구성할 수 있습니다. 이러한 마커에는 항상 +kubebuilder:validation 접두사가 있습니다.

추가 리소스

- API 코드의 마커 사용에 대한 자세한 내용은 다음 Kubebuilder 설명서를 참조하십시오.
  - CRD 생성
  - 마커
  - OpenAPIv3 검증 마커 목록
- CRD의 OpenAPIv3 검증 스키마에 대한 자세한 내용은 Kubernetes 설명서를 참조하십시오.

5.3.2.4. 컨트롤러 구현

새 API 및 컨트롤러를 생성하면 컨트롤러 논리를 구현할 수 있습니다.

절차

- 이 예제에서는 생성된 컨트롤러 파일 **controllers/memcached_controller.go**를 다음 예제 구현으로 교체합니다.

예 5.1. memcached_controller.go의 예

```
/*
Copyright 2020.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package controllers

import (
	appsv1 "k8s.io/api/apps/v1"
	corev1 "k8s.io/api/core/v1"
	"k8s.io/apimachinery/pkg/api/errors"
	metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
	"k8s.io/apimachinery/pkg/types"
	"reflect"

	"context"

	"github.com/go-logr/logr"
	"k8s.io/apimachinery/pkg/runtime"
	ctrl "sigs.k8s.io/controller-runtime"
	"sigs.k8s.io/controller-runtime/pkg/client"
	ctrllog "sigs.k8s.io/controller-runtime/pkg/log"

	cachev1 "github.com/example-inc/memcached-operator/api/v1"
)

// MemcachedReconciler reconciles a Memcached object
type MemcachedReconciler struct {
	client.Client
	Log    logr.Logger
	Scheme *runtime.Scheme
}

//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds,verbs=get;
list;watch;create;update;patch;delete
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/status,ver
bs=get;update;patch
```

```
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/finalizers,
verbs=update
//
+kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;crea
te;update;patch;delete
// +kubebuilder:rbac:groups=core,resources=pods,verbs=get;list;

// Reconcile is part of the main kubernetes reconciliation loop which aims to
// move the current state of the cluster closer to the desired state.
// TODO(user): Modify the Reconcile function to compare the state specified by
// the Memcached object against the actual cluster state, and then
// perform operations to make the cluster state reflect the state specified by
// the user.
//
// For more details, check Reconcile and its Result here:
// - https://pkg.go.dev/sigs.k8s.io/controller-runtime@v0.7.0/pkg/reconcile
func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error) {
	//log := r.Log.WithValues("memcached", req.NamespacedName)
	log := ctrllog.FromContext(ctx)
	// Fetch the Memcached instance
	memcached := &cachev1.Memcached{}
	err := r.Get(ctx, req.NamespacedName, memcached)
	if err != nil {
		if errors.IsNotFound(err) {
			// Request object not found, could have been deleted after reconcile
request.
			// Owned objects are automatically garbage collected. For additional
cleanup logic use finalizers.
			// Return and don't requeue
			log.Info("Memcached resource not found. Ignoring since object must
be deleted")
			return ctrl.Result{}, nil
		}
		// Error reading the object - requeue the request.
		log.Error(err, "Failed to get Memcached")
		return ctrl.Result{}, err
	}

	// Check if the deployment already exists, if not create a new one
	found := &appsv1.Deployment{}
	err = r.Get(ctx, types.NamespacedName{Name: memcached.Name,
Namespace: memcached.Namespace}, found)
	if err != nil && errors.IsNotFound(err) {
		// Define a new deployment
		dep := r.deploymentForMemcached(memcached)
		log.Info("Creating a new Deployment", "Deployment.Namespace",
dep.Namespace, "Deployment.Name", dep.Name)
		err = r.Create(ctx, dep)
		if err != nil {
			log.Error(err, "Failed to create new Deployment",
"Deployment.Namespace", dep.Namespace, "Deployment.Name", dep.Name)
			return ctrl.Result{}, err
		}
		// Deployment created successfully - return and requeue
```

```
            return ctrl.Result{Requeue: true}, nil
        } else if err != nil {
            log.Error(err, "Failed to get Deployment")
            return ctrl.Result{}, err
        }

        // Ensure the deployment size is the same as the spec
        size := memcached.Spec.Size
        if *found.Spec.Replicas != size {
            found.Spec.Replicas = &size
            err = r.Update(ctx, found)
            if err != nil {
                log.Error(err, "Failed to update Deployment",
"Deployment.Namespace", found.Namespace, "Deployment.Name", found.Name)
                return ctrl.Result{}, err
            }
            // Spec updated - return and requeue
            return ctrl.Result{Requeue: true}, nil
        }

        // Update the Memcached status with the pod names
        // List the pods for this memcached's deployment
        podList := &corev1.PodList{}
        listOpts := []client.ListOption{
            client.InNamespace(memcached.Namespace),
            client.MatchingLabels(labelsForMemcached(memcached.Name)),
        }
        if err = r.List(ctx, podList, listOpts...); err != nil {
            log.Error(err, "Failed to list pods", "Memcached.Namespace",
memcached.Namespace, "Memcached.Name", memcached.Name)
            return ctrl.Result{}, err
        }
        podNames := getPodNames(podList.Items)

        // Update status.Nodes if needed
        if !reflect.DeepEqual(podNames, memcached.Status.Nodes) {
            memcached.Status.Nodes = podNames
            err := r.Status().Update(ctx, memcached)
            if err != nil {
                log.Error(err, "Failed to update Memcached status")
                return ctrl.Result{}, err
            }
        }

        return ctrl.Result{}, nil
}

// deploymentForMemcached returns a memcached Deployment object
func (r *MemcachedReconciler) deploymentForMemcached(m
*cachev1.Memcached) *appsv1.Deployment {
        ls := labelsForMemcached(m.Name)
        replicas := m.Spec.Size

        dep := &appsv1.Deployment{
            ObjectMeta: metav1.ObjectMeta{
                Name:      m.Name,
```

```
                Namespace: m.Namespace,
            },
            Spec: appsv1.DeploymentSpec{
                Replicas: &replicas,
                Selector: &metav1.LabelSelector{
                    MatchLabels: ls,
                },
                Template: corev1.PodTemplateSpec{
                    ObjectMeta: metav1.ObjectMeta{
                        Labels: ls,
                    },
                    Spec: corev1.PodSpec{
                        Containers: []corev1.Container{{
                            Image:   "memcached:1.4.36-alpine",
                            Name:    "memcached",
                            Command: []string{"memcached", "-m=64", "-o",
"modern", "-v"},
                            Ports: []corev1.ContainerPort{{
                                ContainerPort: 11211,
                                Name:          "memcached",
                            }},
                        }},
                    },
                },
            },
        }
        // Set Memcached instance as the owner and controller
        ctrl.SetControllerReference(m, dep, r.Scheme)
        return dep
}

// labelsForMemcached returns the labels for selecting the resources
// belonging to the given memcached CR name.
func labelsForMemcached(name string) map[string]string {
        return map[string]string{"app": "memcached", "memcached_cr": name}
}

// getPodNames returns the pod names of the array of pods passed in
func getPodNames(pods []corev1.Pod) []string {
        var podNames []string
        for _, pod := range pods {
                podNames = append(podNames, pod.Name)
        }
        return podNames
}

// SetupWithManager sets up the controller with the Manager.
func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
        return ctrl.NewControllerManagedBy(mgr).
                For(&cachev1.Memcached{}).
                Owns(&appsv1.Deployment{}).
                Complete(r)
}
```

예제 컨트롤러는 각 Memcached CR(사용자 정의 리소스)에 대해 다음 조정 논리를 실행합니다.

- Memcached 배포가 없는 경우 생성합니다.
- 배포 크기가 Memcached CR 사양에 지정된 것과 같은지 확인합니다.
- Memcached CR 상태를 memcached Pod의 이름으로 업데이트합니다.

다음 하위 섹션에서는 구현 예제의 컨트롤러에서 리소스를 조사하는 방법과 조정 반복문을 트리거하는 방법을 설명합니다. 이러한 하위 섹션을 건너뛰고 Operator 실행 으로 직접 이동할 수 있습니다.

5.3.2.4.1. 컨트롤러에서 조사하는 리소스

controllers/memcached_controller.go의 SetupWithManager() 함수는 해당 컨트롤러에서 보유하고 관리하는 CR 및 기타 리소스를 조사하기 위해 컨트롤러를 빌드하는 방법을 지정합니다.

```
import (
 ...
 appsv1 "k8s.io/api/apps/v1"
 ...
)

func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
 return ctrl.NewControllerManagedBy(mgr).
  For(&cachev1.Memcached{}).
  Owns(&appsv1.Deployment{}).
  Complete(r)
}
```

NewControllerManagedBy()에서는 다양한 컨트롤러 구성을 허용하는 컨트롤러 빌더를 제공합니다.

For(&cachev1.Memcached{})는 조사할 기본 리소스로 Memcached 유형을 지정합니다. Memcached 유형에 대한 각 추가, 업데이트 또는 삭제 이벤트의 경우 조정 반복문은 해당 Memcached 오브젝트의 조정 Request 인수(네임스페이스 및 이름 키로 구성됨)로 전송됩니다.

Owns(&appsv1.Deployment{})는 조사할 보조 리소스로 Deployment 유형을 지정합니다. 이벤트 핸들러는 Deployment 유형, 즉 추가, 업데이트 또는 삭제 이벤트가 발생할 때마다 각 이벤트를 배포 소유자의 조정 요청에 매핑합니다. 이 경우 소유자는 배포가 생성된 Memcached 오브젝트입니다.

5.3.2.4.2. 컨트롤러 구성

기타 여러 유용한 구성을 사용하여 컨트롤러를 초기화할 수 있습니다. 예를 들면 다음과 같습니다.

- **MaxConcurrentReconciles** 옵션을 사용하여 컨트롤러에 대한 최대 동시 조정 수를 설정합니다. 기본값은 **1**입니다.

```
func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&cachev1.Memcached{}).
        Owns(&appsv1.Deployment{}).
        WithOptions(controller.Options{
            MaxConcurrentReconciles: 2,
        }).
        Complete(r)
}
```

- 서술자를 사용하여 조사 이벤트를 필터링합니다.

- 조정 반복문에 대한 요청을 조정하기 위해 조사 이벤트를 변환하는 방법을 변경하려면 *EventHandler* 유형을 선택합니다. 기본 및 보조 리소스보다 복잡한 **Operator** 관계의 경우 **EnqueueRequestsFromMapFunc** 핸들러를 사용하여 조사 이벤트를 임의의 조정 요청 세트로 변환할 수 있습니다.

이러한 구성 및 기타 구성에 대한 자세한 내용은 업스트림 *빌더* 및 *컨트롤러* **GoDocs**를 참조하십시오.

5.3.2.4.3. 조정 반복문

모든 컨트롤러에는 조정 반복문을 구현하는 **Reconcile()** 메서드가 포함된 조정기 오브젝트가 있습니다. 조정 반복문에는 캐시에서 기본 리소스 오브젝트인 **Memcached**를 찾는 데 사용되는 네임스페이스 및 이름 키인 **Request** 인수가 전달됩니다.

```
import (
 ctrl "sigs.k8s.io/controller-runtime"

 cachev1 "github.com/example-inc/memcached-operator/api/v1"
 ...
)

func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result,
error) {
  // Lookup the Memcached instance for this reconcile request
  memcached := &cachev1.Memcached{}
```

```
  err := r.Get(ctx, req.NamespacedName, memcached)
  ...
}
```

반환 값, 결과, 오류에 따라 요청이 다시 큐에 추가되고 조정 루프가 다시 트리거될 수 있습니다.

```
// Reconcile successful - don't requeue
return ctrl.Result{}, nil
// Reconcile failed due to error - requeue
return ctrl.Result{}, err
// Requeue for any reason other than an error
return ctrl.Result{Requeue: true}, nil
```

Result.RequeueAfter를 설정하여 유예 기간 후 요청을 다시 큐에 추가할 수 있습니다.

```
import "time"

// Reconcile for any reason other than an error after 5 seconds
return ctrl.Result{RequeueAfter: time.Second*5}, nil
```

참고

주기적으로 **CR**을 조정하도록 **RequeueAfter**를 설정하여 **Result**를 반환할 수 있습니다.

조정기, 클라이언트, 리소스 이벤트와의 상호 작용에 대한 자세한 내용은 Controller Runtime Client API 설명서를 참조하십시오.

5.3.2.4.4. 권한 및 RBAC 매니페스트

컨트롤러에서 관리하는 리소스와 상호 작용하려면 특정 **RBAC** 권한이 필요합니다. 이러한 권한은 다음과 같은 **RBAC** 마커를 사용하여 지정합니다.

```
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds,verbs=get;list;watch;
create;update;patch;delete
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/status,verbs=get;upd
ate;patch
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/finalizers,verbs=upd
ate
//
+kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;
```

```
patch;delete
// +kubebuilder:rbac:groups=core,resources=pods,verbs=get;list;

func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result,
error) {
  ...
}
```

config/rbac/role.yaml의 ClusterRole 오브젝트 매니페스트는 make manifests 명령이 실행될 때마다 controller-gen 유틸리티를 사용하여 이전 마커에서 생성됩니다.

5.3.2.5. 프록시 지원 활성화

Operator 작성자는 네트워크 프록시를 지원하는 Operator를 개발할 수 있습니다. 클러스터 관리자는 OLM(Operator Lifecycle Manager)에서 처리하는 환경 변수에 대한 프록시 지원을 구성합니다. 프록시된 클러스터를 지원하려면 Operator에서 다음 표준 프록시 변수에 대한 환경을 검사하고 해당 값을 Operands에 전달해야 합니다.

- HTTP_PROXY
- HTTPS_PROXY
- NO_PROXY

참고

이 튜토리얼에서는 예제 환경 변수로 HTTP_PROXY를 사용합니다.

사전 요구 사항

- 클러스터 전체 egress 프록시가 활성화된 클러스터입니다.

프로세스

1. 다음을 포함하도록 controllers/memcached_controller.go 파일을 편집합니다.

   a. operator-lib 라이브러리에서 프록시 패키지를 가져옵니다.

```
import (
  ...
   "github.com/operator-framework/operator-lib/proxy"
)
```

b. 조정 루프에 **proxy.ReadProxyVarsFromEnv** 도우미 함수를 추가하고 **Operand** 환경에 결과를 추가합니다.

```
for i, container := range dep.Spec.Template.Spec.Containers {
  dep.Spec.Template.Spec.Containers[i].Env = append(container.Env,
proxy.ReadProxyVarsFromEnv()...)
}
...
```

2. **config/manager/manager.yaml** 파일에 다음을 추가하여 **Operator** 배포에서 환경 변수를 설정합니다.

```
containers:
 - args:
   - --leader-elect
   - --leader-election-id=ansible-proxy-demo
   image: controller:latest
   name: manager
   env:
     - name: "HTTP_PROXY"
       value: "http_proxy_test"
```

5.3.2.6. Operator 실행

다음 세 가지 방법으로 **Operator SDK CLI**를 사용하여 **Operator**를 빌드하고 실행할 수 있습니다.

- **Go** 프로그램으로 클러스터 외부에서 로컬로 실행합니다.
- 클러스터에서 배포로 실행합니다.
- **Operator**를 번들로 제공하고 **OLM(Operator Lifecycle Manager)**을 사용하여 클러스터에 배포합니다.

참고

Go 기반 Operator를 OpenShift Container Platform의 배포 또는 OLM을 사용하는 번들로 실행하려면 지원되는 이미지를 사용하도록 프로젝트를 업데이트해야 합니다.

5.3.2.6.1. 클러스터 외부에서 로컬로 실행

Operator 프로젝트를 클러스터 외부의 Go 프로그램으로 실행할 수 있습니다. 이는 배포 및 테스트 속도를 높이기 위한 개발 목적에 유용합니다.

프로세스

- 다음 명령을 실행하여 ~/.kube/config 파일에 구성된 클러스터에 CRD(사용자 정의 리소스 정의)를 설치하고 Operator를 로컬로 실행합니다.

```
$ make install run
```

출력 예

```
...
2021-01-10T21:09:29.016-0700 INFO controller-runtime.metrics metrics server is
starting to listen {"addr": ":8080"}
2021-01-10T21:09:29.017-0700 INFO setup starting manager
2021-01-10T21:09:29.017-0700 INFO controller-runtime.manager starting metrics server
{"path": "/metrics"}
2021-01-10T21:09:29.018-0700 INFO controller-runtime.manager.controller.memcached
Starting EventSource {"reconciler group": "cache.example.com", "reconciler kind":
"Memcached", "source": "kind source: /, Kind="}
2021-01-10T21:09:29.218-0700 INFO controller-runtime.manager.controller.memcached
Starting Controller {"reconciler group": "cache.example.com", "reconciler kind":
"Memcached"}
2021-01-10T21:09:29.218-0700 INFO controller-runtime.manager.controller.memcached
Starting workers {"reconciler group": "cache.example.com", "reconciler kind":
"Memcached", "worker count": 1}
```

5.3.2.6.2. 클러스터에서 배포로 실행

Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

사전 요구 사항

- 지원되는 이미지로 프로젝트를 업데이트하여 OpenShift Container Platform에서 Go 기반 Operator를 실행할 준비가 됨

프로세스

1. 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

   a. 이미지를 빌드합니다.

   ```
   $ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. 이미지를 리포지토리로 내보냅니다.

   ```
   $ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   두 명령 모두 이미지의 이름과 태그(예: IMG=<registry>/<user>/<image_name>:<tag>)를 Makefile에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 IMG ?= controller:latest 값을 수정합니다.

2. 다음 명령을 실행하여 Operator를 배포합니다.

   ```
   $ make deploy IMG=<registry>/<user>/<image_name>:<tag>
   ```

   기본적으로 이 명령은 <project_name>-system 형식으로 된 Operator 프로젝트 이름을 사

용하여 네임스페이스를 생성하고 배포에 사용됩니다. 이 명령은 또한 config/rbac에서 RBAC 매니페스트를 설치합니다.

3. *다음 명령을 실행하여 Operator가 실행 중인지 확인합니다.*

```
$ oc get deployment -n <project_name>-system
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
<project_name>-controller-manager       1/1     1            1           8m
```

5.3.2.6.3. Operator 번들링 및 Operator Lifecycle Manager를 통한 배포

5.3.2.6.3.1. Operator 번들

Operator 번들 형식은 Operator SDK 및 Operator Lifecycle Manager (OLM)의 기본 패키지 메서드입니다. Operator SDK를 사용하여 Operator 프로젝트를 번들 이미지로 빌드하고 푸시하여 OLM에서 Operator를 사용할 수 있습니다.

사전 요구 사항

- *개발 워크스테이션에 Operator SDK CLI가 설치됨*
- *OpenShift CLI(oc) v4.11 이상이 설치됨*
- *Operator SDK를 사용하여 Operator 프로젝트를 초기화함*
- *Operator가 Go 기반인 경우 OpenShift Container Platform에서 실행하기 위해 지원되는 이미지를 사용하도록 프로젝트를 업데이트해야 함*

프로세스

1. *Operator 프로젝트 디렉터리에서 다음 make 명령을 실행하여 Operator 이미지를 빌드하*

고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 **IMG** 인수를 수정합니다. **Quay.io**와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

a. 이미지를 빌드합니다.

```
$ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
```

참고

Operator에 대해 **SDK**에서 생성한 **Dockerfile**은 **Go** 빌드 를 위해 **GOARCH=amd64** 를 명시적으로 참조합니다. 이는**AMD64** 이외의 아키텍처의 경우 **GOARCH=$CACHEGETARCH** 에 수정될 수 있습니다. **Docker**는 환경 변수를 **-platform** 에서 지정한 값으로 자동 설정합니다. **Buildah**를 사용하면 **-build-arg** 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

b. 이미지를 리포지토리로 내보냅니다.

```
$ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
```

2. **Operator SDK generate bundle** 및 **bundle validate** 명령을 비롯한 다양한 명령을 호출하는 **make bundle** 명령을 실행하여 **Operator** 번들 매니페스트를 생성합니다.

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Operator의 번들 매니페스트는 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. **make bundle** 명령은 **Operator** 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

- **ClusterServiceVersion** 오브젝트를 포함하는 **bundle/manifests**라는 번들 매니페스트 디렉터리
- **bundle/metadata**라는 번들 메타데이터 디렉터리
- **config/crd** 디렉터리의 모든 **CRD**(사용자 정의 리소스 정의)
- **Dockerfile bundle.Dockerfile**

그런 다음 **operator-sdk bundle validate**를 사용하여 이러한 파일을 자동으로 검증하고 디스크상의 번들 표현이 올바른지 확인합니다.

3. 다음 명령을 실행하여 번들 이미지를 빌드하고 내보냅니다. **OLM**에서는 하나 이상의 번들 이미지를 참조하는 인덱스 이미지를 통해 **Operator** 번들을 사용합니다.

   a. 번들 이미지를 빌드합니다. 이미지를 내보낼 레지스트리, 사용자 네임스페이스, 이미지 태그에 대한 세부 정보를 사용하여 **BUNDLE_IMG**를 설정합니다.

   ```
   $ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
   ```

   b. 번들 이미지를 내보냅니다.

   ```
   $ docker push <registry>/<user>/<bundle_image_name>:<tag>
   ```

5.3.2.6.3.2. Operator Lifecycle Manager를 사용하여 Operator 배포

OLM(Operator Lifecycle Manager)은 **Kubernetes** 클러스터에서 **Operator** 및 관련 서비스를 설치, 업데이트하고 라이프사이클을 관리하는 데 도움이 됩니다. **OLM**은 기본적으로 **OpenShift Container Platform**에 설치되고 **Kubernetes** 확장으로 실행되므로 추가 툴 없이 모든 **Operator** 라이프사이클 관리 기능에 웹 콘솔과 **OpenShift CLI(oc)**를 사용할 수 있습니다.

Operator 번들 형식은 **Operator SDK** 및 **OLM**의 기본 패키지 메서드입니다. **Operator SDK**를 사용하여 **OLM**에서 번들 이미지를 신속하게 실행하여 올바르게 실행되는지 확인할 수 있습니다.

사전 요구 사항

- 개발 워크스테이션에 **Operator SDK CLI**가 설치됨
- **Operator** 번들 이미지를 빌드하여 레지스트리로 내보냄
- **Kubernetes** 기반 클러스터에 **OLM**이 설치되어 있음(**apiextensions.k8s.io/v1 CRD**(예: **OpenShift Container Platform 4.11**)를 사용하는 경우 **v1.16.0** 이상)

- cluster-admin 권한이 있는 계정을 사용하여 oc로 클러스터에 로그인됨

- Operator가 Go 기반인 경우 OpenShift Container Platform에서 실행하기 위해 지원되는 이미지를 사용하도록 프로젝트를 업데이트해야 함

프로세스

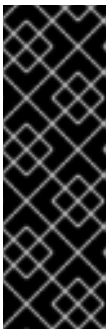
1. 다음 명령을 입력하여 클러스터에서 Operator를 실행합니다.

```
$ operator-sdk run bundle \ 1
  -n <namespace> \ 2
  <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. run bundle 명령은 유효한 파일 기반 카탈로그를 생성하고 OLM을 사용하여 클러스터에 Operator 번들을 설치합니다.

2. 선택 사항: 기본적으로 이 명령은 현재 활성 프로젝트에 ~/.kube/config 파일에 Operator를 설치합니다. -n 플래그를 추가하면 설치에 다른 네임스페이스 범위를 설정할 수 있습니다.

3. 이미지를 지정하지 않으면 명령에서 quay.io/operator-framework/opm:latest 를 기본 인덱스 이미지로 사용합니다. 이미지를 지정하면 명령에서 번들 이미지 자체를 인덱스 이미지로 사용합니다.



중요

OpenShift Container Platform 4.11부터 run bundle 명령은 기본적으로 Operator 카탈로그의 파일 기반 카탈로그 형식을 지원합니다. Operator 카탈로그의 더 이상 사용되지 않는 SQLite 데이터베이스 형식은 계속 지원되지만 향후 릴리스에서 제거됩니다. Operator 작성자는 해당 워크플로우를 파일 기반 카탈로그 형식으로 마이그레이션하는 것이 좋습니다.

이 명령은 다음 작업을 수행합니다.

-

번들 이미지를 참조하는 인덱스 이미지를 생성합니다. 인덱스 이미지는 불투명하고 일시적이지만 프로덕션에서 카탈로그에 번들을 추가하는 방법을 정확하게 반영합니다.

- OperatorHub에서 Operator를 검색할 수 있도록 새 인덱스 이미지를 가리키는 카탈로그 소스를 생성합니다.

- OperatorGroup,Subscription,InstallPlan 및 RBAC를 포함한 기타 모든 필수 리소스를 생성하여 Operator를 클러스터에 배포합니다.

5.3.2.7. 사용자 정의 리소스 생성

Operator가 설치되면 Operator에서 현재 클러스터에 제공하는 CR(사용자 정의 리소스)을 생성하여 Operator를 테스트할 수 있습니다.

사전 요구 사항

- Memcached CR을 제공하는 Memcached Operator의 예가 클러스터에 설치됨

프로세스

1. Operator가 설치된 네임스페이스로 변경합니다. 예를 들어 make deploy 명령을 사용하여 Operator를 배포한 경우 다음을 실행합니다.

```
$ oc project memcached-operator-system
```

2. 다음 사양을 포함하도록 config/samples/cache_v1_memcached.yaml의 샘플 Memcached CR 매니페스트를 편집합니다.

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
...
spec:
...
  size: 3
```

3. CR을 생성합니다.

```
$ oc apply -f config/samples/cache_v1_memcached.yaml
```

4. Memcached Operator에서 샘플 CR에 대한 배포를 올바른 크기로 생성하는지 확인합니다.

```
$ oc get deployments
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           8m
memcached-sample                        3/3     3            3           1m
```

5. Pod 및 CR 상태를 확인하여 상태가 Memcached Pod 이름으로 업데이트되었는지 확인합니다.

a. Pod를 확인합니다.

```
$ oc get pods
```

출력 예

```
NAME                                  READY     STATUS    RESTARTS   AGE
memcached-sample-6fd7c98d8-7dqdr      1/1       Running   0          1m
memcached-sample-6fd7c98d8-g5k7v      1/1       Running   0          1m
memcached-sample-6fd7c98d8-m7vn7      1/1       Running   0          1m
```

b. CR 상태 확인:

```
$ oc get memcached/memcached-sample -o yaml
```

출력 예

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
...
  name: memcached-sample
...
spec:
  size: 3
status:
  nodes:
  - memcached-sample-6fd7c98d8-7dqdr
  - memcached-sample-6fd7c98d8-g5k7v
  - memcached-sample-6fd7c98d8-m7vn7
```

6. *배포 크기를 업데이트합니다.*

a. *config/samples/cache_v1_memcached.yaml 파일을 업데이트하여 Memcached CR의 spec.size 필드를 3에서 5로 변경합니다.*

```
$ oc patch memcached memcached-sample \
    -p '{"spec":{"size": 5}}' \
    --type=merge
```

b. *Operator에서 배포 크기를 변경하는지 확인합니다.*

```
$ oc get deployments
```

출력 예

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           10m
memcached-sample                      5/5     5            5           3m
```

7. 다음 명령을 실행하여 CR을 삭제합니다.

```
$ oc delete -f config/samples/cache_v1_memcached.yaml
```

8. 이 튜토리얼의 일부로 생성된 리소스를 정리합니다.

- make deploy 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ make undeploy
```

- operator-sdk run bundle 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ operator-sdk cleanup <project_name>
```

5.3.2.8. 추가 리소스

- Operator SDK에서 생성한 디렉터리 구조에 대한 자세한 내용은 Go 기반 Operator의 프로젝트 레이아웃 을 참조하십시오.

- 클러스터 전체 송신 프록시가 구성된 경우 클러스터 관리자는 프록시 설정을 덮어쓰거나 OLM(Operator Lifecycle Manager)에서 실행되는 특정 Operator에 대한 사용자 정의 CA 인증서를 삽입 할 수 있습니다.

5.3.3. Go 기반 Operator의 프로젝트 레이아웃

operator-sdk CLI에서는 각 Operator 프로젝트에 대해 다양한 패키지 및 파일을 생성하거나 스캐폴드를 지정할 수 있습니다.

5.3.3.1. Go 기반 프로젝트 레이아웃

operator-sdk init 명령을 사용하여 생성된 기본 유형의 Go 기반 Operator 프로젝트에는 다음 파일과 디렉터리가 포함됩니다.

파일 또는 디렉터리	목적

파일 또는 디렉터리	목적
main.go	Operator의 기본 프로그램으로, 모든 CRD(사용자 정의 리소스 정의)를 **apis/** 디렉터리에 등록하고 **controllers/** 디렉터리의 모든 컨트롤러를 시작하는 새 관리자를 인스턴스화합니다.
apis/	CRD의 API를 정의하는 디렉터리 트리입니다. **apis/<version>/<kind>_types.go** 파일을 편집하여 각 리소스 유형에 대한 API를 정의하고 컨트롤러에서 이러한 패키지를 가져와서 해당 리소스 유형이 있는지 조사해야 합니다.
controllers/	컨트롤러 구현입니다. **controller/<kind>_controller.go** 파일을 편집하여 지정된 유형의 리소스 유형을 처리하도록 컨트롤러의 조정 논리를 정의합니다.
config/	CRD, RBAC, 인증서를 포함하여 클러스터에 컨트롤러를 배포하는 데 사용하는 Kubernetes 매니페스트입니다.
Makefile	컨트롤러를 빌드하고 배포하는 데 사용하는 대상입니다.
Dockerfile	컨테이너 엔진에서 Operator를 빌드하는 데 사용하는 지침입니다.
manifests/	CRD 등록, RBAC 설정, Operator를 배포로 배포하는 Kubernetes 매니페스트입니다.

5.3.4. 최신 Operator SDK 버전에 대한 Go 기반 Operator 프로젝트 업데이트

OpenShift Container Platform 4.11은 Operator SDK 1.22.2를 지원합니다. 워크스테이션에 1.16.0 CLI가 이미 설치되어 있는 경우 최신 버전을 설치하여 CLI를 1.22.2로 업데이트할 수 있습니다.

그러나 기존 Operator 프로젝트에서 Operator SDK 1.22.2와의 호환성을 유지하려면 1.16.0 이후의 중단된 변경 사항에 대한 업데이트 단계가 필요합니다. 1.16.0으로 이전에 생성되거나 유지 관리되는 Operator 프로젝트에서 업데이트 단계를 수동으로 수행해야 합니다.

5.3.4.1. Operator SDK 1.22.2의 Go 기반 Operator 프로젝트 업데이트

다음 절차에서는 1.22.2와의 호환성을 위해 기존 Go 기반 Operator 프로젝트를 업데이트합니다.

사전 요구 사항

- ***Operator SDK 1.22.2가 설치되어 있어야 합니다.***

- Operator SDK 1.16.0을 사용하여 Operator 프로젝트를 생성 또는 유지 관리합니다.

프로세스

1. config/default/manager_auth_proxy_patch.yaml 파일을 다음과 같이 변경합니다.

```
...
spec:
  template:
    spec:
      containers:
      - name: kube-rbac-proxy
        image: registry.redhat.io/openshift4/ose-kube-rbac-proxy:v4.11 (1)
        args:
        - "--secure-listen-address=0.0.0.0:8443"
        - "--upstream=http://127.0.0.1:8080/"
        - "--logtostderr=true"
        - "--v=0" (2)
...
resources:
  limits:
    cpu: 500m
    memory: 128Mi
  requests:
    cpu: 5m
    memory: 64Mi (3)
```

(1) 태그 버전을 v4.10 에서 v4.11 로 업데이트합니다.

(2) 디버깅 로그 수준을 --v=10 에서 --v=0 로 줄입니다.

(3) 리소스 요청 및 제한을 추가합니다.

2. Makefile 을 다음과 같이 변경합니다.

   a. Makefile 에 다음 환경 변수를 추가하여 이미지 다이제스트에 대한 지원을 활성화합니다.

이전 Makefile

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)
...
```

새 Makefile

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)

# BUNDLE_GEN_FLAGS are the flags passed to the operator-sdk generate bundle
command
BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

# USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
# You can enable this value if you would like to use SHA Based Digests
# To enable set flag to true
USE_IMAGE_DIGESTS ?= false
ifeq ($(USE_IMAGE_DIGESTS), true)
 BUNDLE_GEN_FLAGS += --use-image-digests
endif
```

b. *Makefile 을 편집하여 번들 대상을 BUNDLE_GEN_FLAGS 환경 변수로 교체합니다.*

이전 Makefile

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --
overwrite --version $(VERSION) $(BUNDLE_METADATA_OPTS)
```

새 Makefile

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS)
```

c. *Makefile 을 편집하여 opm 을 버전 1.23.0으로 업데이트합니다.*

```
.PHONY: opm
OPM = ./bin/opm
opm: ## Download opm locally if necessary.
ifeq (,$(wildcard $(OPM)))
ifeq (,$(shell which opm 2>/dev/null))
 @{ \
 set -e ;\
 mkdir -p $(dir $(OPM)) ;\
 OS=$(shell go env GOOS) && ARCH=$(shell go env GOARCH) && \
 curl -sSLo $(OPM) https://github.com/operator-framework/operator-
registry/releases/download/v1.23.0/$${OS}-$${ARCH}-opm ;\ 1
 chmod +x $(OPM) ;\
 }
else
OPM = $(shell which opm)
endif
endif
```

1 *v1.19.1 을 v1.23.0 으로 바꿉니다.*

d. *Makefile 을 편집하여 go get 대상을 go install 대상으로 바꿉니다.*

이전 Makefile

```
CONTROLLER_GEN = $(shell pwd)/bin/controller-gen
.PHONY: controller-gen
controller-gen: ## Download controller-gen locally if necessary.
 $(call go-get-tool,$(CONTROLLER_GEN),sigs.k8s.io/controller-
tools/cmd/controller-gen@v0.8.0)

KUSTOMIZE = $(shell pwd)/bin/kustomize
.PHONY: kustomize
kustomize: ## Download kustomize locally if necessary.
 $(call go-get-tool,$(KUSTOMIZE),sigs.k8s.io/kustomize/kustomize/v3@v3.8.7)

ENVTEST = $(shell pwd)/bin/setup-envtest
```

```
.PHONY: envtest
envtest: ## Download envtest-setup locally if necessary.
 $(call go-get-tool,$(ENVTEST),sigs.k8s.io/controller-runtime/tools/setup-
envtest@latest)

# go-get-tool will 'go get' any package $2 and install it to $1.
PROJECT_DIR := $(shell dirname $(abspath $(lastword $(MAKEFILE_LIST))))
define go-get-tool
@[ -f $(1) ] || { \
set -e ;\
TMP_DIR=$$(mktemp -d) ;\
cd $$TMP_DIR ;\
go mod init tmp ;\
echo "Downloading $(2)" ;\
GOBIN=$(PROJECT_DIR)/bin go get $(2) ;\
rm -rf $$TMP_DIR ;\
}
endef
```

ㅋ Makefile

```
##@ Build Dependencies

## Location to install dependencies to
LOCALBIN ?= $(shell pwd)/bin
$(LOCALBIN):
 mkdir -p $(LOCALBIN)

## Tool Binaries
KUSTOMIZE ?= $(LOCALBIN)/kustomize
CONTROLLER_GEN ?= $(LOCALBIN)/controller-gen
ENVTEST ?= $(LOCALBIN)/setup-envtest

## Tool Versions
KUSTOMIZE_VERSION ?= v3.8.7
CONTROLLER_TOOLS_VERSION ?= v0.8.0

KUSTOMIZE_INSTALL_SCRIPT ?=
"https://raw.githubusercontent.com/kubernetes-
sigs/kustomize/master/hack/install_kustomize.sh"
.PHONY: kustomize
kustomize: $(KUSTOMIZE) ## Download kustomize locally if necessary.
$(KUSTOMIZE): $(LOCALBIN)
 curl -s $(KUSTOMIZE_INSTALL_SCRIPT) | bash -s -- $(subst
v,,$(KUSTOMIZE_VERSION)) $(LOCALBIN)

.PHONY: controller-gen
controller-gen: $(CONTROLLER_GEN) ## Download controller-gen locally if
necessary.
```

```
$(CONTROLLER_GEN): $(LOCALBIN)
 GOBIN=$(LOCALBIN) go install sigs.k8s.io/controller-tools/cmd/controller-gen@$(CONTROLLER_TOOLS_VERSION)

.PHONY: envtest
envtest: $(ENVTEST) ## Download envtest-setup locally if necessary.
$(ENVTEST): $(LOCALBIN)
 GOBIN=$(LOCALBIN) go install sigs.k8s.io/controller-runtime/tools/setup-envtest@latest
```

e. *Makefile 의 ENVTEST_K8S_VERSION 및 controller-gen 필드를 업데이트하여 Kubernetes 1.24를 지원합니다.*

```
...
ENVTEST_K8S_VERSION = 1.24 (1)
...
sigs.k8s.io/controller-tools/cmd/controller-gen@v0.9.0 (2)
```

(1) *1.22 버전을 1.24 로 업데이트합니다.*

(2) *버전 0.7.0 을 0.9.0 으로 업데이트합니다.*

f. *Makefile 에 변경 사항을 적용하고 다음 명령을 입력하여 Operator를 다시 빌드합니다.*

```
$ make
```

3. *go.mod 파일을 다음과 같이 변경하여 Go 및 해당 종속 항목을 업데이트합니다.*

```
go 1.18 (1)

require (
 github.com/onsi/ginkgo v1.16.5 (2)
 github.com/onsi/gomega v1.18.1 (3)
 k8s.io/api v0.24.0 (4)
 k8s.io/apimachinery v0.24.0 (5)
```

```
    k8s.io/client-go v0.24.0 6
    sigs.k8s.io/controller-runtime v0.12.1 7
)
```

1 버전 1.16 을 1.18 로 업데이트합니다.

2 버전 v1.16.4 를 v1.16.5 로 업데이트합니다.

3 버전 v1.15.0 을 v1.18.1 로 업데이트합니다.

4 5 6 버전 v0.22.1 을 v0.24.0 으로 업데이트합니다.

7 버전 v0.10.0 을 v0.12.1 로 업데이트합니다.

4. 다음 명령을 입력하여 종속 항목을 다운로드하여 정리합니다.

```
$ go mod tidy
```

5. api/webhook_suitetest.go 및 controllers/suite_test.go 제품군 테스트 파일을 사용하는 경우 다음과 같이 변경합니다.

이전 제품군 테스트 파일

```
cfg, err := testEnv.Start()
```

새로운 제품군 테스트 파일

```
var err error
// cfg is defined in this file globally.
cfg, err = testEnv.Start()
```

6. *Kubernetes 선언 플러그인을 사용하는 경우 다음과 같은 변경 사항으로 Dockerfile을 업데이트합니다.*

   a. *COPY 컨트롤러/컨트롤러/:를 시작하는 행 아래에 다음 변경 사항을 추가합니다.*

```
# https://github.com/kubernetes-sigs/kubebuilder-declarative-pattern/blob/master/docs/addon/walkthrough/README.md#adding-a-manifest
# Stage channels and make readable
COPY channels/ /channels/
RUN chmod -R a+rx /channels/
```

   b. *COPY --from=builder /workspace/manager를 시작하는 행 아래에 다음 변경 사항을 추가합니다.:*

```
# copy channels
COPY --from=builder /channels /channels
```

5.3.4.2. 추가 리소스

- *번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션*
- *Operator SDK 1.16.0 프로젝트 업그레이드*
- *Operator SDK v1.10.1의 프로젝트 업그레이드*
- *Operator SDK v1.8.0의 프로젝트 업그레이드*

5.4. ANSIBLE 기반 OPERATOR

5.4.1. Ansible 기반 Operator를 위한 Operator SDK 시작하기

Operator SDK에는 Go 코드를 작성하지 않고도 기존 Ansible 플레이북 및 모듈을 활용하여 Kubernetes 리소스를 통합 애플리케이션으로 배포하는 Operator 프로젝트를 생성하는 옵션이 포함되어 있습니다.

Operator 개발자는 Operator SDK에서 제공하는 툴 및 라이브러리를 사용하여 Ansible 기반 Operator를 설정 및 실행하는 기본 동작을 설명하기 위해 분산형 키-값 저장소인 Memcached에 대한 Go 기반 Operator 예제를 빌드하고 클러스터에 배포할 수 있습니다.

5.4.1.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Ansible v2.9.0
- Ansible Runner v2.0.2 이상
- Ansible Runner HTTP Event Emitter 플러그인 v1.0.0 이상
- Python 3.8.6+
- OpenShift Python client v0.12.0+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- *Operator SDK CLI 설치*
- *OpenShift CLI 시작하기*

5.4.1.2. Ansible 기반 Operator 생성 및 배포

Operator SDK를 사용하여 Memcached에 대한 간단한 Ansible 기반 Operator를 빌드하고 배포할 수 있습니다.

프로세스

1. *프로젝트를 생성합니다.*

   a. *프로젝트 디렉토리를 생성합니다.*

   ```
   $ mkdir memcached-operator
   ```

   b. *프로젝트 디렉터리로 변경합니다.*

   ```
   $ cd memcached-operator
   ```

   c. *ansible 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.*

   ```
   $ operator-sdk init \
     --plugins=ansible \
     --domain=example.com
   ```

2. *API를 생성합니다.*

   간단한 Memcached API를 생성합니다.

   ```
   $ operator-sdk create api \
     --group cache \
     --version v1 \
     --kind Memcached \
     --generate-role 1
   ```

1

API에 대한 Ansible 역할을 생성합니다.

3. *Operator 이미지를 빌드하여 내보냅니다.*

기본 Makefile 대상을 사용하여 Operator를 빌드하고 내보냅니다. 내보낼 수 있는 레지스트리를 사용하는 이미지의 가져오기 사양에 IMG를 설정합니다.

```
$ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

4. *Operator를 실행합니다.*

a. *CRD를 설치합니다.*

```
$ make install
```

b. *클러스터에 프로젝트를 배포합니다. 내보낸 이미지에 IMG를 설정합니다.*

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

5. *샘플 CR(사용자 정의 리소스)을 생성합니다.*

a. *샘플 CR을 생성합니다.*

```
$ oc apply -f config/samples/cache_v1_memcached.yaml \
    -n memcached-operator-system
```

b. *CR에서 Operator를 조정하는지 확인합니다.*

```
$ oc logs deployment.apps/memcached-operator-controller-manager \
    -c manager \
    -n memcached-operator-system
```

출력 예

```
...
I0205 17:48:45.881666       7 leaderelection.go:253] successfully acquired lease
memcached-operator-system/memcached-operator
{"level":"info","ts":1612547325.8819902,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612547325.98242,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612547325.9824686,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
workers","worker count":4}
{"level":"info","ts":1612547348.8311093,"logger":"runner","msg":"Ansible-runner
exited successfully","job":"4037200794235010051","name":"memcached-
sample","namespace":"memcached-operator-system"}
```

6. *CR 삭제*

   다음 명령을 실행하여 CR을 삭제합니다.

   ```
   $ oc delete -f config/samples/cache_v1_memcached -n memcached-operator-system
   ```

7. *정리합니다.*

   다음 명령을 실행하여 이 절차의 일부로 생성된 리소스를 정리합니다.

   ```
   $ make undeploy
   ```

5.4.1.3. 다음 단계

- *Ansible 기반 Operator를 빌드하는 방법에 대한 자세한 내용은 Ansible 기반 Operator용 Operator SDK 튜토리얼 을 참조하십시오.*

5.4.2. Ansible 기반 Operator를 위한 Operator SDK 튜토리얼

Operator 개발자는 Operator SDK의 Ansible 지원을 활용하여 분산형 키-값 저장소인 Memcached에 대한 Ansible 기반 Operator 예제를 빌드하고 라이프사이클을 관리할 수 있습니다. 이 튜토리얼에서는 다음 프로세스를 안내합니다.

- Memcached 배포 생성
- 배포 크기가 Memcached CR(사용자 정의 리소스) 사양에 지정된 것과 같은지 확인합니다.
- memcached Pod의 이름으로 상태 작성기를 사용하여 Memcached CR 상태를 업데이트합니다.

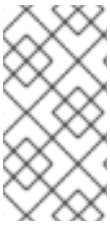
이 프로세스는 Operator 프레임워크의 두 가지 주요 요소를 사용하여 수행됩니다.

Operator SDK

operator-sdk CLI 툴 및 controller-runtime 라이브러리 API

OLM(Operator Lifecycle Manager)

클러스터에 대한 Operator의 설치, 업그레이드, RBAC(역할 기반 액세스 제어)



참고

이 튜토리얼에는 Ansible 기반 Operator용 Operator SDK 시작하기보다 자세히 설명되어 있습니다.

5.4.2.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Ansible v2.9.0
- Ansible Runner v2.0.2 이상
- Ansible Runner HTTP Event Emitter 플러그인 v1.0.0 이상

- Python 3.8.6+
- OpenShift Python client v0.12.0+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치
- OpenShift CLI 시작하기

5.4.2.2. 프로젝트 생성

Operator SDK CLI를 사용하여 memcached-operator라는 프로젝트를 생성합니다.

프로세스

1. 프로젝트에 사용할 디렉터리를 생성합니다.

```
$ mkdir -p $HOME/projects/memcached-operator
```

2. 디렉터리로 변경합니다.

```
$ cd $HOME/projects/memcached-operator
```

3. ansible 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.

```
$ operator-sdk init \
  --plugins=ansible \
  --domain=example.com
```

-

5.4.2.2.1. PROJECT 파일

operator-sdk init 명령으로 생성된 파일 중에는 Kubebuilder PROJECT 파일이 있습니다. 이어서 프로젝트 루트에서 실행되는 operator-sdk 명령과 help 출력에서는 이 파일을 읽고 프로젝트 유형이 Ansible임을 확인합니다. 예를 들면 다음과 같습니다.

```
domain: example.com
layout:
- ansible.sdk.operatorframework.io/v1
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
projectName: memcached-operator
version: "3"
```

5.4.2.3. API 생성

Operator SDK CLI를 사용하여 Memcached API를 생성합니다.

프로세스

- 다음 명령을 실행하여 그룹이 cache이고 버전이 v1, 종류가 Memcached인 API를 생성합니다.

```
$ operator-sdk create api \
    --group cache \
    --version v1 \
    --kind Memcached \
    --generate-role 1
```

1

API에 대한 Ansible 역할을 생성합니다.

API가 생성되면 Operator 프로젝트에서 다음 구조를 사용하여 업데이트합니다.

Memcached CRD

샘플 Memcached 리소스 포함

관리자

다음을 사용하여 클러스터 상태를 원하는 상태로 조정하는 프로그램입니다.

- 조정기(**Ansible** 역할 또는 플레이북 중 하나)
- **watches.yaml** 파일(**Memcached** 리소스를 **memcached Ansible** 역할에 연결)

5.4.2.4. 관리자 수정

Memcached 리소스가 생성, 업데이트 또는 삭제될 때마다 실행되는 **Ansible** 역할의 형태로 조정 논리를 제공하도록 **Operator** 프로젝트를 업데이트합니다.

프로세스

1. 다음 구조를 사용하여 **roles/memcached/tasks/main.yml** 파일을 업데이트합니다.

```
---
- name: start memcached
  k8s:
    definition:
      kind: Deployment
      apiVersion: apps/v1
      metadata:
        name: '{{ ansible_operator_meta.name }}-memcached'
        namespace: '{{ ansible_operator_meta.namespace }}'
      spec:
        replicas: "{{size}}"
        selector:
          matchLabels:
            app: memcached
        template:
          metadata:
            labels:
              app: memcached
          spec:
            containers:
            - name: memcached
              command:
              - memcached
              - -m=64
              - -o
              - modern
              - -v
              image: "docker.io/memcached:1.4.36-alpine"
              ports:
                - containerPort: 11211
```

-

이 **memcached** 역할은 **memcached** 배포가 있는지 확인하고 배포 크기를 설정합니다.

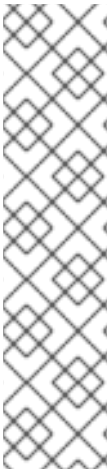
2. **roles/memcached/defaults/main.yml** 파일을 편집하여 **Ansible** 역할에 사용되는 변수의 기본값을 설정합니다.

```
---
# defaults file for Memcached
size: 1
```

3. 다음 구조를 사용하여 **config/samples/cache_v1_memcached.yaml** 파일에서 **Memcached** 샘플 리소스를 업데이트합니다.

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  labels:
    app.kubernetes.io/name: memcached
    app.kubernetes.io/instance: memcached-sample
    app.kubernetes.io/part-of: memcached-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: memcached-operator
  name: memcached-sample
spec:
  size: 3
```

CR(사용자 정의 리소스) 사양의 키-값 쌍은 **Ansible**에 추가 변수로 전달됩니다.



참고

spec 필드에 있는 모든 변수의 이름은 **Ansible**을 실행하기 전에 **Operator**에서 스네이크 케이스(밑줄이 포함된 소문자)로 변환합니다. 예를 들어 사양의 **serviceAccount**는 **Ansible**에서 **service_account**로 변환됩니다.

watches.yaml 파일에서 **snakeCaseParameters** 옵션을 **false**로 설정하여 이 대소문자 변환을 비활성화할 수 있습니다. 애플리케이션에 예상대로 입력되고 있는지 확인하려면 변수에 대해 **Ansible**에서 일부 유형 검증을 수행하는 것이 좋습니다.

5.4.2.5. 프록시 지원 활성화

Operator 작성자는 네트워크 프록시를 지원하는 **Operator**를 개발할 수 있습니다. 클러스터 관리자는 **OLM(Operator Lifecycle Manager)**에서 처리하는 환경 변수에 대한 프록시 지원을 구성합니다. 프록시

*된 클러스터를 지원하려면 **Operator**에서 다음 표준 프록시 변수에 대한 환경을 검사하고 해당 값을 **Operands**에 전달해야 합니다.*

- ***HTTP_PROXY***
- ***HTTPS_PROXY***
- ***NO_PROXY***

참고

*이 튜토리얼에서는 예제 환경 변수로 **HTTP_PROXY**를 사용합니다.*

사전 요구 사항

- *클러스터 전체 **egress** 프록시가 활성화된 클러스터입니다.*

프로세스

1. ***roles/memcached/tasks/main.yml** 파일을 다음으로 업데이트하여 배포에 환경 변수를 추가합니다.*

```
...
env:
  - name: HTTP_PROXY
    value: '{{ lookup("env", "HTTP_PROXY") | default("", True) }}'
  - name: http_proxy
    value: '{{ lookup("env", "HTTP_PROXY") | default("", True) }}'
...
```

2. ***config/manager/manager.yaml** 파일에 다음을 추가하여 **Operator** 배포에서 환경 변수를 설정합니다.*

```
containers:
 - args:
   - --leader-elect
   - --leader-election-id=ansible-proxy-demo
   image: controller:latest
   name: manager
```

```
env:
  - name: "HTTP_PROXY"
    value: "http_proxy_test"
```

5.4.2.6. Operator 실행

다음 세 가지 방법으로 **Operator SDK CLI**를 사용하여 **Operator**를 빌드하고 실행할 수 있습니다.

- **Go** 프로그램으로 클러스터 외부에서 로컬로 실행합니다.
- 클러스터에서 배포로 실행합니다.
- **Operator**를 번들로 제공하고 **OLM(Operator Lifecycle Manager)**을 사용하여 클러스터에 배포합니다.

5.4.2.6.1. 클러스터 외부에서 로컬로 실행

Operator 프로젝트를 클러스터 외부의 **Go** 프로그램으로 실행할 수 있습니다. 이는 배포 및 테스트 속도를 높이기 위한 개발 목적에 유용합니다.

프로세스

- 다음 명령을 실행하여 **~/.kube/config** 파일에 구성된 클러스터에 **CRD(사용자 정의 리소스 정의)**를 설치하고 **Operator**를 로컬로 실행합니다.

```
$ make install run
```

출력 예

```
...
{"level":"info","ts":1612589622.7888272,"logger":"ansible-
controller","msg":"Watching
resource","Options.Group":"cache.example.com","Options.Version":"v1","Options.Ki
nd":"Memcached"}
{"level":"info","ts":1612589622.7897573,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
{"level":"info","ts":1612589622.789971,"logger":"controller-
runtime.manager","msg":"starting metrics server","path":"/metrics"}
{"level":"info","ts":1612589622.7899997,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
```

```
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612589622.8904517,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612589622.8905244,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting workers","worker
count":8}
```

5.4.2.6.2. 클러스터에서 배포로 실행

Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

프로세스

1. 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

   a. 이미지를 빌드합니다.

   ```
   $ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
   ```

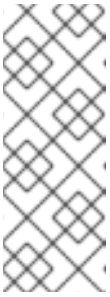
   

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. 이미지를 리포지토리로 내보냅니다.

   ```
   $ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```



참고

두 명령 모두 이미지의 이름과 태그(예: **IMG=<registry>/<user>/<image_name>:<tag>**)를 **Makefile**에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 **IMG ?= controller:latest** 값을 수정합니다.

2. 다음 명령을 실행하여 **Operator**를 배포합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

기본적으로 이 명령은 **<project_name>-system** 형식으로 된 **Operator** 프로젝트 이름을 사용하여 네임스페이스를 생성하고 배포에 사용됩니다. 이 명령은 또한 **config/rbac**에서 **RBAC** 매니페스트를 설치합니다.

3. 다음 명령을 실행하여 **Operator**가 실행 중인지 확인합니다.

```
$ oc get deployment -n <project_name>-system
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
<project_name>-controller-manager       1/1     1            1           8m
```

5.4.2.6.3. Operator 번들링 및 Operator Lifecycle Manager를 통한 배포

5.4.2.6.3.1. Operator 번들

Operator 번들 형식은 **Operator SDK** 및 **Operator Lifecycle Manager (OLM)**의 기본 패키지 메서드입니다. **Operator SDK**를 사용하여 **Operator** 프로젝트를 번들 이미지로 빌드하고 푸시하여 **OLM**에서 **Operator**를 사용할 수 있습니다.

사전 요구 사항

- 개발 워크스테이션에 **Operator SDK CLI**가 설치됨

- OpenShift CLI(oc) v4.11 이상이 설치됨
- Operator SDK를 사용하여 Operator 프로젝트를 초기화함

프로세스

1. Operator 프로젝트 디렉터리에서 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

   a. 이미지를 빌드합니다.

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. 이미지를 리포지토리로 내보냅니다.

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

2. Operator SDK generate bundle 및 bundle validate 명령을 비롯한 다양한 명령을 호출하는 make bundle 명령을 실행하여 Operator 번들 매니페스트를 생성합니다.

   ```
   $ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   Operator의 번들 매니페스트는 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. make bundle 명령은 Operator 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

   - ClusterServiceVersion 오브젝트를 포함하는 bundle/manifests라는 번들 매니페스트 디렉터리

- *bundle/metadata라는 번들 메타데이터 디렉터리*
- *config/crd 디렉터리의 모든 CRD(사용자 정의 리소스 정의)*
- *Dockerfile bundle.Dockerfile*

그런 다음 operator-sdk bundle validate를 사용하여 이러한 파일을 자동으로 검증하고 디스크상의 번들 표현이 올바른지 확인합니다.

3. *다음 명령을 실행하여 번들 이미지를 빌드하고 내보냅니다. OLM에서는 하나 이상의 번들 이미지를 참조하는 인덱스 이미지를 통해 Operator 번들을 사용합니다.*

   a. *번들 이미지를 빌드합니다. 이미지를 내보낼 레지스트리, 사용자 네임스페이스, 이미지 태그에 대한 세부 정보를 사용하여 BUNDLE_IMG를 설정합니다.*

   ```
   $ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
   ```

   b. *번들 이미지를 내보냅니다.*

   ```
   $ docker push <registry>/<user>/<bundle_image_name>:<tag>
   ```

5.4.2.6.3.2. Operator Lifecycle Manager를 사용하여 Operator 배포

OLM(Operator Lifecycle Manager)은 Kubernetes 클러스터에서 Operator 및 관련 서비스를 설치, 업데이트하고 라이프사이클을 관리하는 데 도움이 됩니다. OLM은 기본적으로 OpenShift Container Platform에 설치되고 Kubernetes 확장으로 실행되므로 추가 툴 없이 모든 Operator 라이프사이클 관리 기능에 웹 콘솔과 OpenShift CLI(oc)를 사용할 수 있습니다.

Operator 번들 형식은 Operator SDK 및 OLM의 기본 패키지 메서드입니다. Operator SDK를 사용하여 OLM에서 번들 이미지를 신속하게 실행하여 올바르게 실행되는지 확인할 수 있습니다.

사전 요구 사항

-

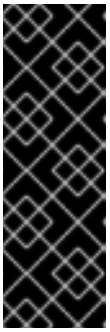
- 개발 워크스테이션에 Operator SDK CLI가 설치됨
- Operator 번들 이미지를 빌드하여 레지스트리로 내보냄
- Kubernetes 기반 클러스터에 OLM이 설치되어 있음(apiextensions.k8s.io/v1 CRD(예: OpenShift Container Platform 4.11)를 사용하는 경우 v1.16.0 이상)
- cluster-admin 권한이 있는 계정을 사용하여 oc로 클러스터에 로그인됨

프로세스

1. 다음 명령을 입력하여 클러스터에서 Operator를 실행합니다.

```
$ operator-sdk run bundle \ 1
  -n <namespace> \ 2
  <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. run bundle 명령은 유효한 파일 기반 카탈로그를 생성하고 OLM을 사용하여 클러스터에 Operator 번들을 설치합니다.
2. 선택 사항: 기본적으로 이 명령은 현재 활성 프로젝트에 ~/.kube/config 파일에 Operator를 설치합니다. -n 플래그를 추가하면 설치에 다른 네임스페이스 범위를 설정할 수 있습니다.
3. 이미지를 지정하지 않으면 명령에서 quay.io/operator-framework/opm:latest 를 기본 인덱스 이미지로 사용합니다. 이미지를 지정하면 명령에서 번들 이미지 자체를 인덱스 이미지로 사용합니다.



중요

OpenShift Container Platform 4.11부터 run bundle 명령은 기본적으로 Operator 카탈로그의 파일 기반 카탈로그 형식을 지원합니다. Operator 카탈로그의 더 이상 사용되지 않는 SQLite 데이터베이스 형식은 계속 지원되지만 향후 릴리스에서 제거됩니다. Operator 작성자는 해당 워크플로우를 파일 기반 카탈로그 형식으로 마이그레이션하는 것이 좋습니다.

이 명령은 다음 작업을 수행합니다.

- 번들 이미지를 참조하는 인덱스 이미지를 생성합니다. 인덱스 이미지는 불투명하고 일시적이지만 프로덕션에서 카탈로그에 번들을 추가하는 방법을 정확하게 반영합니다.
- OperatorHub에서 Operator를 검색할 수 있도록 새 인덱스 이미지를 가리키는 카탈로그 소스를 생성합니다.
- OperatorGroup,Subscription,InstallPlan 및 RBAC를 포함한 기타 모든 필수 리소스를 생성하여 Operator를 클러스터에 배포합니다.

5.4.2.7. 사용자 정의 리소스 생성

Operator가 설치되면 Operator에서 현재 클러스터에 제공하는 CR(사용자 정의 리소스)을 생성하여 Operator를 테스트할 수 있습니다.

사전 요구 사항

- Memcached CR을 제공하는 Memcached Operator의 예가 클러스터에 설치됨

프로세스

1. Operator가 설치된 네임스페이스로 변경합니다. 예를 들어 make deploy 명령을 사용하여 Operator를 배포한 경우 다음을 실행합니다.

```
$ oc project memcached-operator-system
```

2. 다음 사양을 포함하도록 config/samples/cache_v1_memcached.yaml의 샘플 Memcached CR 매니페스트를 편집합니다.

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
...
spec:
...
  size: 3
```

3. *CR을 생성합니다.*

```
$ oc apply -f config/samples/cache_v1_memcached.yaml
```

4. *Memcached Operator에서 샘플 CR에 대한 배포를 올바른 크기로 생성하는지 확인합니다.*

```
$ oc get deployments
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           8m
memcached-sample                        3/3     3            3           1m
```

5. *Pod 및 CR 상태를 확인하여 상태가 Memcached Pod 이름으로 업데이트되었는지 확인합니다.*

   a. *Pod를 확인합니다.*

```
$ oc get pods
```

출력 예

```
NAME                                  READY     STATUS    RESTARTS   AGE
memcached-sample-6fd7c98d8-7dqdr      1/1       Running   0          1m
memcached-sample-6fd7c98d8-g5k7v      1/1       Running   0          1m
memcached-sample-6fd7c98d8-m7vn7      1/1       Running   0          1m
```

b. *CR 상태 확인:*

```
$ oc get memcached/memcached-sample -o yaml
```

출력 예

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
...
  name: memcached-sample
...
spec:
  size: 3
status:
  nodes:
  - memcached-sample-6fd7c98d8-7dqdr
  - memcached-sample-6fd7c98d8-g5k7v
  - memcached-sample-6fd7c98d8-m7vn7
```

6. *배포 크기를 업데이트합니다.*

a. *config/samples/cache_v1_memcached.yaml 파일을 업데이트하여 Memcached CR의 spec.size 필드를 3에서 5로 변경합니다.*

```
$ oc patch memcached memcached-sample \
    -p '{"spec":{"size": 5}}' \
    --type=merge
```

b. *Operator에서 배포 크기를 변경하는지 확인합니다.*

```
$ oc get deployments
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           10m
memcached-sample                        5/5     5            5           3m
```

7. 다음 명령을 실행하여 CR을 삭제합니다.

```
$ oc delete -f config/samples/cache_v1_memcached.yaml
```

8. 이 튜토리얼의 일부로 생성된 리소스를 정리합니다.

- make deploy 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ make undeploy
```

- operator-sdk run bundle 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ operator-sdk cleanup <project_name>
```

5.4.2.8. 추가 리소스

- Operator SDK에서 생성한 디렉터리 구조에 대한 자세한 내용은 Ansible 기반 Operator의 프로젝트 레이아웃 을 참조하십시오.

- 클러스터 전체 송신 프록시가 구성된 경우 클러스터 관리자는 프록시 설정을 덮어쓰거나 OLM(Operator Lifecycle Manager)에서 실행되는 특정 Operator에 대한 사용자 정의 CA 인증서를 삽입 할 수 있습니다.

5.4.3. Ansible 기반 Operator의 프로젝트 레이아웃

operator-sdk CLI에서는 각 Operator 프로젝트에 대해 다양한 패키지 및 파일을 생성하거나 스캐폴드를 지정할 수 있습니다.

5.4.3.1. Ansible 기반 프로젝트 레이아웃

***operator-sdk init --plugins ansible** 명령을 사용하여 생성된 **Ansible** 기반 **Operator** 프로젝트에는 다음 디렉터리 및 파일이 포함됩니다.*

파일 또는 디렉터리	목적
Dockerfile	Operator의 컨테이너 이미지를 빌드하는 Dockerfile입니다.
Makefile	Operator 바이너리를 래핑하는 컨테이너 이미지를 빌드, 게시, 배포할 대상 및 CRD(사용자 정의 리소스 정의)를 설치 및 설치 제거할 대상입니다.
PROJECT	Operator의 메타데이터 정보가 포함된 YAML 파일입니다.
config/crd	기본 CRD 파일 및 **kustomization.yaml** 파일 설정입니다.
config/default	배포를 위해 모든 Operator 매니페스트를 수집합니다. **make deploy** 명령에서 사용합니다.
config/manager	컨트롤러 관리자 배포입니다.
config/prometheus	Operator 모니터링을 위한 **ServiceMonitor** 리소스입니다.
config/rbac	리더 선택 방식 및 인증 프록시 관련 역할 및 역할 바인딩입니다.
config/samples	CRD에 대해 생성된 샘플 리소스입니다.
config/testing	테스트를 위한 샘플 구성입니다.
playbooks/	플레이북을 실행할 하위 디렉터리입니다.
roles/	역할 트리를 실행할 하위 디렉터리입니다.
watches.yaml	조사할 리소스의 GVK(그룹/버전/종류) 및 Ansible 호출 메서드입니다. **create api** 명령을 사용하여 새 항목이 추가되었습니다.
requirements.yml	빌드 중 설치할 Ansible 컬렉션 및 역할 종속 항목이 포함된 YAML 파일입니다.
molecule/	역할 및 Operator의 끝점 테스트에 대한 개별 시나리오입니다.

5.4.4. 최신 Operator SDK 버전의 프로젝트 업데이트

OpenShift Container Platform 4.11은 Operator SDK 1.22.2를 지원합니다. 워크스테이션에 1.16.0 CLI가 이미 설치되어 있는 경우 최신 버전을 설치하여 CLI를 1.22.2로 업데이트할 수 있습니다.

그러나 기존 Operator 프로젝트에서 Operator SDK 1.22.2와의 호환성을 유지하려면 1.16.0 이후의 중단된 변경 사항에 대한 업데이트 단계가 필요합니다. 1.16.0으로 이전에 생성되거나 유지 관리되는 Operator 프로젝트에서 업데이트 단계를 수동으로 수행해야 합니다.

5.4.4.1. Operator SDK 1.22.2용 Ansible 기반 Operator 프로젝트 업데이트

다음 절차에서는 1.22.2와의 호환성을 위해 기존 Ansible 기반 Operator 프로젝트를 업데이트합니다.

사전 요구 사항

- Operator SDK 1.22.2가 설치되어 있어야 합니다.
- Operator SDK 1.16.0을 사용하여 Operator 프로젝트를 생성 또는 유지 관리합니다.

프로세스

1. config/default/manager_auth_proxy_patch.yaml 파일을 다음과 같이 변경합니다.

```
...
spec:
  template:
    spec:
      containers:
      - name: kube-rbac-proxy
        image: registry.redhat.io/openshift4/ose-kube-rbac-proxy:v4.11 1
        args:
        - "--secure-listen-address=0.0.0.0:8443"
        - "--upstream=http://127.0.0.1:8080/"
        - "--logtostderr=true"
        - "--v=0" 2
...
resources:
  limits:
    cpu: 500m
    memory: 128Mi
  requests:
    cpu: 5m
    memory: 64Mi 3
```

1. 태그 버전을 v4.10 에서 v4.11 로 업데이트합니다.

2

*디버깅 로그 수준을 **--v=10** 에서 **--v=0** 로 줄입니다.*

3

리소스 요청 및 제한을 추가합니다.

2. ***Makefile** 을 다음과 같이 변경합니다.*

   a. ***Makefile** 에 다음 환경 변수를 추가하여 이미지 다이제스트에 대한 지원을 활성화합니다.*

   *이전 **Makefile***

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)
...
```

   *새 **Makefile***

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)

# BUNDLE_GEN_FLAGS are the flags passed to the operator-sdk generate bundle
command
BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

# USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
# You can enable this value if you would like to use SHA Based Digests
# To enable set flag to true
USE_IMAGE_DIGESTS ?= false
ifeq ($(USE_IMAGE_DIGESTS), true)
 BUNDLE_GEN_FLAGS += --use-image-digests
endif
```

b.

Makefile 을 편집하여 번들 대상을 BUNDLE_GEN_FLAGS 환경 변수로 교체합니다.

이전 Makefile

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --
overwrite --version $(VERSION) $(BUNDLE_METADATA_OPTS)
```

새 Makefile

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS)
```

c.

Makefile 을 편집하여 opm 을 버전 1.23.0으로 업데이트합니다.

```
.PHONY: opm
OPM = ./bin/opm
opm: ## Download opm locally if necessary.
ifeq (,$(wildcard $(OPM)))
ifeq (,$(shell which opm 2>/dev/null))
 @{ \
 set -e ;\
 mkdir -p $(dir $(OPM)) ;\
 OS=$(shell go env GOOS) && ARCH=$(shell go env GOARCH) && \
 curl -sSLo $(OPM) https://github.com/operator-framework/operator-
registry/releases/download/v1.23.0/$${OS}-$${ARCH}-opm ;\ 1
 chmod +x $(OPM) ;\
 }
else
OPM = $(shell which opm)
endif
endif
```

1

v1.19.1 을 v1.23.0 으로 바꿉니다.

d. *Makefile 에 변경 사항을 적용하고 다음 명령을 입력하여 Operator를 다시 빌드합니다.*

```
$ make
```

3. *다음 예와 같이 Operator의 Dockerfile에서 이미지 태그를 업데이트합니다.*

Dockerfile 예

```
FROM registry.redhat.io/openshift4/ose-ansible-operator:v4.11 (1)
```

(1) *version 태그를 v4.11 로 업데이트합니다.*

4. *다음 예와 같이 requirements.yml 파일을 업데이트합니다.*

```
collections:
 - name: community.kubernetes
   version: "2.0.1" (1)
 - name: operator_sdk.util
   version: "0.4.0" (2)
 - name: kubernetes.core
   version: "2.3.1" (3)
 - name: cloud.common (4)
   version: "2.1.1"
```

(1) *버전 1.2.1 을 2.0.1 로 업데이트합니다.*

(2) *버전 0.3.1 을 0.4.0 으로 업데이트합니다.*

(3) *버전 2.2.0 을 2.3.1 로 업데이트합니다.*

4

cloud.common 컬렉션을 추가하여 Operator Ansible SDK에 대한 지원을 추가합니다.

중요

버전 2.0.0부터 community.kubernetes 컬렉션이 kubernetes.core 로 변경되었습니다. community.kubernetes 수집은 더 이상 사용되지 않는 리디렉션으로 kubernetes.core 로 교체되었습니다. community.kubernetes.kubernetes로 시작하는 정규화된 컬렉션 이름(FQCNs)을 사용하는 경우 kubernetes.core 를 사용하도록 FQCNs를 업데이트해야 합니다.

5.4.4.2. 추가 리소스

- *번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션*
- *Operator SDK v1.16.0 프로젝트 업그레이드*
- *Operator SDK v1.10.1의 프로젝트 업그레이드*
- *Operator SDK v1.8.0의 프로젝트 업그레이드*

5.4.5. Operator SDK의 Ansible 지원

5.4.5.1. 사용자 정의 리소스 파일

Operator는 Kubernetes 확장 메커니즘인 CRD(사용자 정의 리소스 정의)를 사용하므로 CR(사용자 정의 리소스)이 기본 제공되는 네이티브 Kubernetes 오브젝트처럼 보이고 작동합니다.

CR 파일 형식은 Kubernetes 리소스 파일입니다. 오브젝트에는 필수 및 선택적 필드가 있습니다.

표 5.1. 사용자 정의 리소스 필드

필드	설명
apiVersion	생성할 CR의 버전입니다.

필드	설명
kind	생성할 CR의 종류입니다.
metadata	생성할 Kubernetes별 메타데이터입니다.
spec(선택 사항)	Ansible에 전달되는 변수의 키-값 목록입니다. 이 필드는 기본적으로 비어 있습니다.
status	오브젝트의 현재 상태를 요약합니다. Ansible 기반 Operator의 경우 CRD에 대해 **status** 하위 리소스가 활성화되고, 기본적으로 CR **status**에 **condition** 정보를 포함하는 **operator_sdk.util.k8s_status** Ansible 모듈에서 관리합니다.
annotations	CR에 추가할 Kubernetes별 주석입니다.

다음 CR 주석 목록은 Operator의 동작을 수정합니다.

표 5.2. Ansible 기반 Operator 주석

주석	설명
ansible.operator-sdk/reconcile-period	CR 조정 간격을 지정합니다. 이 값은 표준 Golang 패키지 **time**을 사용하여 구문 분석합니다. 특히 **ParseDuration**은 기본 접미사인 **s**를 적용하여 초 단위로 값을 지정하는 데 사용됩니다.

Ansible 기반 Operator 주석의 예

```
apiVersion: "test1.example.com/v1alpha1"
kind: "Test1"
metadata:
  name: "example"
annotations:
  ansible.operator-sdk/reconcile-period: "30s"
```

5.4.5.2. watches.yaml 파일

GVK(그룹/버전/종류)는 Kubernetes API의 고유 ID입니다. watches.yaml 파일에는 GVK로 확인하는 CR(사용자 정의 리소스)에서 Ansible 역할 또는 플레이북으로의 매핑 목록이 포함됩니다. Operator는 이 매핑 파일이 미리 정의된 위치(/opt/ansible/watches.yaml)에 있을 것으로 예상합니다.

표 5.3. watches.yaml 파일 매핑

필드	설명
group	조사할 CR 그룹입니다.
version	조사할 CR 버전입니다.
kind	조사할 CR의 종류입니다.
role(기본값)	컨테이너에 추가된 Ansible 역할의 경로입니다. 예를 들어 **roles** 디렉터리가 **/opt/ansible/roles/**에 있고 역할 이름이 **busybox**인 경우 이 값은 **/opt/ansible/roles/busybox**입니다. 이 필드는 **playbook** 필드와 함께 사용할 수 없습니다.
playbook	컨테이너에 추가된 Ansible 플레이북의 경로입니다. 이 플레이북은 역할을 호출하는 방법이 될 것으로 예상됩니다. 이 필드는 **role** 필드와 함께 사용할 수 없습니다.
reconcilePeriod(선택 사항)	지정된 CR에 대한 조정 간격, 즉 역할 또는 플레이북이 실행되는 간격입니다.
manageStatus(선택 사항)	**true**(기본값)로 설정된 경우 Operator는 일반적으로 CR의 상태를 관리합니다. **false**로 설정하면 지정된 역할이나 플레이북 또는 별도의 컨트롤러에서 CR의 상태를 관리합니다.

watches.yaml 파일의 예

```
- version: v1alpha1 (1)
  group: test1.example.com
  kind: Test1
  role: /opt/ansible/roles/Test1

- version: v1alpha1 (2)
  group: test2.example.com
  kind: Test2
  playbook: /opt/ansible/playbook.yml

- version: v1alpha1 (3)
  group: test3.example.com
  kind: Test3
  playbook: /opt/ansible/test3.yml
  reconcilePeriod: 0
  manageStatus: false
```

(1)

2

Test2를 플레이북에 매핑하는 간단한 예입니다.

3

더 복잡한 Test3 종류 예제입니다. 플레이북에서 CR 상태를 다시 큐에 넣거나 관리하지 않습니다.

5.4.5.2.1. 고급 옵션

GVK별 watches.yaml 파일에 고급 기능을 추가하여 사용할 수 있습니다. 해당 기능은 group, version, kind, playbook 또는 role 필드로 이동할 수 있습니다.

일부 기능은 해당 CR의 주석을 사용하여 리소스별로 덮어쓸 수 있습니다. 덮어쓸 수 있는 옵션에는 아래에 지정된 주석이 있습니다.

표 5.4. 고급 watches.yaml 파일 옵션

기능	YAML 키	설명	덮어쓸 주석	기본값
기간 조정	**reconcilePeriod**	특정 CR에 대한 조정 실행 간격입니다.	**ansible.operator-sdk/reconcile-period**	**1m**
상태 관리	**manageStatus**	Operator에서 각 CR **status** 섹션의 **conditions** 섹션을 관리할 수 있습니다.		**true**
종속 리소스 조사	**watchDependentResources**	Operator에서 Ansible을 통해 생성한 리소스를 동적으로 조사할 수 있습니다.		**true**
클러스터 범위 리소스 조사	**watchClusterScopedResources**	Operator에서 Ansible을 통해 생성한 클러스터 범위 리소스를 조사할 수 있습니다.		**false**
최대 실행기 아티팩트 수	**maxRunnerArtifacts**	Ansible Runner에서 각 개별 리소스에 대해 Operator 컨테이너에 보관하는 아티팩트 디렉터리의 수를 관리합니다.	**ansible.operator-sdk/max-runner-artifacts**	**20**

고급 옵션이 있는 watches.yml 파일의 예

```
- version: v1alpha1
  group: app.example.com
  kind: AppService
  playbook: /opt/ansible/playbook.yml
  maxRunnerArtifacts: 30
  reconcilePeriod: 5s
  manageStatus: False
  watchDependentResources: False
```

5.4.5.3. Ansible로 전송된 추가 변수

추가 변수는 Ansible로 보낸 다음 Operator에서 관리할 수 있습니다. CR(사용자 정의 리소스)의 spec 섹션은 키-값 쌍을 추가 변수로 전달합니다. 이는 ansible-playbook 명령에 전달된 추가 변수와 동일합니다.

또한 Operator는 CR 이름 및 CR 네임스페이스에 대한 meta 필드에 추가 변수를 전달합니다.

다음 CR 예제의 경우

```
apiVersion: "app.example.com/v1alpha1"
kind: "Database"
metadata:
  name: "example"
spec:
  message: "Hello world 2"
  newParameter: "newParam"
```

Ansible에 추가 변수로 전달되는 구조는 다음과 같습니다.

```
{ "meta": {
        "name": "<cr_name>",
        "namespace": "<cr_namespace>",
  },
  "message": "Hello world 2",
  "new_parameter": "newParam",
  "_app_example_com_database": {
```

```
    <full_crd>
  },
}
```

message 및 newParameter 필드는 최상위 레벨에서 추가 변수로 설정되며, meta는 Operator에 정의된 CR에 대한 관련 메타데이터를 제공합니다. meta 필드는 Ansible의 점 표기법을 사용하여 액세스할 수 있습니다. 예를 들면 다음과 같습니다.

```
---
- debug:
    msg: "name: {{ ansible_operator_meta.name }}, {{ ansible_operator_meta.namespace }}"
```

5.4.5.4. Ansible Runner 디렉터리

Ansible Runner는 컨테이너에서 실행되는 Ansible에 대한 정보를 보관합니다. 해당 정보는 /tmp/ansible-operator/runner/<group>/<version>/<kind>/<namespace>/<name>에 있습니다.

추가 리소스

- runner 디렉터리에 대한 자세한 내용은 Ansible Runner 설명서를 참조하십시오.

5.4.6. Ansible용 Kubernetes 컬렉션

Ansible을 사용하여 Kubernetes에서 애플리케이션 라이프사이클을 관리하려면 Ansible용 Kubernetes 컬렉션을 사용하면 됩니다. 개발자는 이 Ansible 모듈 컬렉션을 통해 YAML로 작성된 기존 Kubernetes 리소스 파일을 활용하거나 네이티브 Ansible에서 라이프사이클 관리를 표시할 수 있습니다.

기존 Kubernetes 리소스 파일과 함께 Ansible을 사용할 때의 가장 큰 이점 중 하나는 Jinja 템플릿을 사용할 수 있다는 점입니다. 그러면 Ansible의 몇 가지 간단한 변수를 사용하여 리소스를 사용자 정의할 수 있습니다.

이 섹션에서는 Kubernetes 컬렉션 사용에 대해 자세히 설명합니다. 시작하려면 로컬 워크스테이션에 컬렉션을 설치하고 플레이북을 사용하여 테스트한 후 Operator 내에서 사용하십시오.

5.4.6.1. Ansible용 Kubernetes 컬렉션 설치

로컬 워크스테이션에 Ansible용 Kubernetes 컬렉션을 설치할 수 있습니다.

프로세스

1. Ansible 2.9 이상을 설치합니다.

```
$ sudo dnf install ansible
```

2. OpenShift python 클라이언트 패키지를 설치합니다.

```
$ pip3 install openshift
```

3. 다음 메서드 중 하나를 사용하여 Kubernetes 컬렉션을 설치합니다.

- Ansible Galaxy에서 컬렉션을 직접 설치할 수 있습니다.

```
$ ansible-galaxy collection install community.kubernetes
```

- Operator를 이미 초기화한 경우 프로젝트의 최상위 수준에 requirements.yml 파일이 있을 수 있습니다. 이 파일은 Operator 작동을 위해 설치해야 하는 Ansible 종속 항목을 지정합니다. 기본적으로 이 파일은 community.kubernetes 컬렉션과 operator_sdk.util 컬렉션을 설치합니다. 이 컬렉션은 Operator별 기능에 대한 모듈과 플러그인을 제공합니다.

  requirements.yml 파일에서 종속 모듈을 설치하려면 다음을 수행합니다.

```
$ ansible-galaxy collection install -r requirements.yml
```

5.4.6.2. 로컬에서 Kubernetes 컬렉션 테스트

Operator 개발자는 매번 Operator를 실행하고 다시 빌드하는 대신 로컬 머신에서 Ansible 코드를 실행할 수 있습니다.

사전 요구 사항

- Ansible 기반 Operator 프로젝트를 초기화하고 Operator SDK를 사용하여 생성한 Ansible 역할이 있는 API 생성
- Ansible용 Kubernetes 컬렉션 설치

프로세스

1. Ansible 기반 Operator 프로젝트 디렉터리에서 원하는 Ansible 논리로 roles/<kind>/tasks/main.yml 파일을 수정합니다. roles/<kind>/ 디렉터리는 API를 생성하는 동안 --generate-role 플래그를 사용할 때 생성됩니다. 교체 가능한 <kind>는 API에 지정한 종류와 일치합니다.

   다음 예제에서는 state라는 변수 값에 따라 구성 맵을 생성 및 삭제합니다.

```
---
- name: set ConfigMap example-config to {{ state }}
  community.kubernetes.k8s:
    api_version: v1
    kind: ConfigMap
    name: example-config
    namespace: default (1)
    state: "{{ state }}"
  ignore_errors: true (2)
```

(1) 구성 맵을 default가 아닌 네임스페이스에 생성하려면 이 값을 변경하십시오.

(2) ignore_errors: true를 설정하면 존재하지 않는 구성 맵을 삭제해도 실패하지 않습니다.

2. roles/<kind>/defaults/main.yml 파일을 수정하여 state를 기본적으로 present로 설정합니다.

```
---
state: present
```

3. 프로젝트의 최상위 디렉터리에 playbook.yml 파일을 생성하여 Ansible 플레이북을 생성하고 <kind> 역할을 포함합니다.

```
---
- hosts: localhost
  roles:
    - <kind>
```

4. Playbook을 실행합니다.

```
$ ansible-playbook playbook.yml
```

-

출력 예

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'

PLAY [localhost] ***************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [memcached : set ConfigMap example-config to present]
*********************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

5. *구성 맵이 생성되었는지 확인합니다.*

```
$ oc get configmaps
```

출력 예

```
NAME             DATA   AGE
example-config    0     2m1s
```

6. ***state*를 *absent*로 설정하여 플레이북을 재실행합니다.**

```
$ ansible-playbook playbook.yml --extra-vars state=absent
```

출력 예

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'

PLAY [localhost] ***************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [memcached : set ConfigMap example-config to absent]
*********************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

7. 구성 맵이 삭제되었는지 확인합니다.

```
$ oc get configmaps
```

5.4.6.3. 다음 단계

- CR(사용자 정의 리소스)이 변경될 때 Operator 내부에서 사용자 정의 Ansible 논리를 트리거하는 방법에 대한 자세한 내용은 Operator 내에서 Ansible 사용을 참조하십시오.

5.4.7. Operator 내에서 Ansible 사용

로컬에서 Ansible용 Kubernetes 컬렉션 사용에 익숙해지면 CR(사용자 정의 리소스)이 변경될 때 Operator 내부에서 동일한 Ansible 논리를 트리거할 수 있습니다. 이 예제에서는 Operator에서 조사하는 특정 Kubernetes 리소스에 Ansible 역할을 매핑합니다. 이 매핑은 watches.yaml 파일에서 수행됩니다.

5.4.7.1. 사용자 정의 리소스 파일

Operator는 Kubernetes 확장 메커니즘인 CRD(사용자 정의 리소스 정의)를 사용하므로 CR(사용자 정의 리소스)이 기본 제공되는 네이티브 Kubernetes 오브젝트처럼 보이고 작동합니다.

CR 파일 형식은 Kubernetes 리소스 파일입니다. 오브젝트에는 필수 및 선택적 필드가 있습니다.

표 5.5. 사용자 정의 리소스 필드

필드	설명
apiVersion	생성할 CR의 버전입니다.
kind	생성할 CR의 종류입니다.
metadata	생성할 Kubernetes별 메타데이터입니다.
spec(선택 사항)	Ansible에 전달되는 변수의 키-값 목록입니다. 이 필드는 기본적으로 비어 있습니다.
status	오브젝트의 현재 상태를 요약합니다. Ansible 기반 Operator의 경우 CRD에 대해 **status** 하위 리소스가 활성화되고, 기본적으로 CR **status**에 **condition** 정보를 포함하는 **operator_sdk.util.k8s_status** Ansible 모듈에서 관리합니다.
annotations	CR에 추가할 Kubernetes별 주석입니다.

다음 CR 주석 목록은 Operator의 동작을 수정합니다.

표 5.6. Ansible 기반 Operator 주석

주석	설명
ansible.operator-sdk/reconcile-period	CR 조정 간격을 지정합니다. 이 값은 표준 Golang 패키지 **time**을 사용하여 구문 분석합니다. 특히 **ParseDuration**은 기본 접미사인 **s**를 적용하여 초 단위로 값을 지정하는 데 사용됩니다.

Ansible 기반 Operator 주석의 예

```
apiVersion: "test1.example.com/v1alpha1"
kind: "Test1"
metadata:
  name: "example"
annotations:
  ansible.operator-sdk/reconcile-period: "30s"
```

5.4.7.2. Ansible 기반 Operator를 로컬에서 테스트

*Operator 프로젝트의 최상위 디렉터리에서 **make run** 명령을 사용하여 로컬에서 실행되는 **Ansible***

기반 Operator 내부의 논리를 테스트할 수 있습니다. make run Makefile 대상은 watches.yaml 파일에서 읽고 ~/.kube/config 파일을 사용하여 k8s 모듈과 같은 방식으로 Kubernetes 클러스터와 통신하는 ansible-operator 바이너리를 로컬로 실행합니다.

참고

환경 변수 ANSIBLE_ROLES_PATH를 설정하거나 ansible-roles-path 플래그를 사용하여 역할 경로를 사용자 정의할 수 있습니다. ANSIBLE_ROLES_PATH 값에 역할이 없는 경우 Operator는 {{current directory}}/roles에서 역할을 찾습니다.

사전 요구 사항

- *Ansible Runner v2.0.2 이상*
- *Ansible Runner HTTP Event Emitter 플러그인 v1.0.0 이상*
- *Kubernetes 컬렉션을 로컬에서 테스트하는 데 필요한 이전 단계 수행*

프로세스

1. *CR(사용자 정의 리소스)에 대한 CRD(사용자 정의 리소스 정의) 및 적절한 RBAC(역할 기반 액세스 제어) 정의를 설치합니다.*

```
$ make install
```

출력 예

```
/usr/bin/kustomize build config/crd | kubectl apply -f -
customresourcedefinition.apiextensions.k8s.io/memcacheds.cache.example.com
created
```

2. *make run 명령을 실행합니다.*

```
$ make run
```

출력 예

```
/home/user/memcached-operator/bin/ansible-operator run
{"level":"info","ts":1612739145.2871568,"logger":"cmd","msg":"Version","Go
Version":"go1.15.5","GOOS":"linux","GOARCH":"amd64","ansible-
operator":"v1.10.1","commit":"1abf57985b43bf6a59dcd18147b3c574fa57d3f6"}
...
{"level":"info","ts":1612739148.347306,"logger":"controller-
runtime.metrics","msg":"metrics server is starting to listen","addr":":8080"}
{"level":"info","ts":1612739148.3488882,"logger":"watches","msg":"Environment
variable not set; using default
value","envVar":"ANSIBLE_VERBOSITY_MEMCACHED_CACHE_EXAMPLE_COM","def
ault":2}
{"level":"info","ts":1612739148.3490262,"logger":"cmd","msg":"Environment variable
not set; using default
value","Namespace":"","envVar":"ANSIBLE_DEBUG_LOGS","ANSIBLE_DEBUG_LOG
S":false}
{"level":"info","ts":1612739148.3490646,"logger":"ansible-
controller","msg":"Watching
resource","Options.Group":"cache.example.com","Options.Version":"v1","Options.Ki
nd":"Memcached"}
{"level":"info","ts":1612739148.350217,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
{"level":"info","ts":1612739148.3506632,"logger":"controller-
runtime.manager","msg":"starting metrics server","path":"/metrics"}
{"level":"info","ts":1612739148.350784,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612739148.5511978,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612739148.5512562,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting workers","worker
count":8}
```

*이제 **Operator**에서 이벤트의 **CR**을 조사하므로 **CR**을 생성하면 **Ansible** 역할이 실행됩니다.*

참고

config/samples/<gvk>.yaml CR 매니페스트 예제를 살펴보십시오.

```
apiVersion: <group>.example.com/v1alpha1
kind: <kind>
metadata:
  name: "<kind>-sample"
```

spec 필드가 설정되지 않았기 때문에 추가 변수 없이 Ansible이 호출됩니다. CR에서 Ansible로 추가 변수를 전달하는 방법은 다른 섹션에서 설명합니다. Operator에 적절한 기본값을 설정하는 것이 중요합니다.

3. 기본 변수 state를 present로 설정하여 CR 인스턴스를 생성합니다.

```
$ oc apply -f config/samples/<gvk>.yaml
```

4. example-config 구성 맵이 생성되었는지 확인합니다.

```
$ oc get configmaps
```

출력 예

```
NAME                 STATUS    AGE
example-config       Active    3s
```

5. config/samples/<gvk>.yaml 파일을 수정하여 state 필드를 absent로 설정합니다. 예를 들면 다음과 같습니다.

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
spec:
  state: absent
```

6.

6.

변경 사항을 적용합니다.

```
$ oc apply -f config/samples/<gvk>.yaml
```

7.

구성 맵이 삭제되었는지 확인합니다.

```
$ oc get configmap
```

5.4.7.3. 클러스터에서 Ansible 기반 Operator 테스트

Operator 내부에서 로컬로 사용자 정의 Ansible 논리를 테스트한 후 OpenShift Container Platform 클러스터의 Pod 내부에서 Operator를 테스트할 수 있습니다.

Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

프로세스

1.

다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

a.

이미지를 빌드합니다.

```
$ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
```

참고

Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

b.

이미지를 리포지토리로 내보냅니다.

```
$ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

참고

두 명령 모두 이미지의 이름과 태그(예: IMG=<registry>/<user>/<image_name>:<tag>)를 Makefile에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 IMG ?= controller:latest 값을 수정합니다.

2. 다음 명령을 실행하여 Operator를 배포합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

기본적으로 이 명령은 <project_name>-system 형식으로 된 Operator 프로젝트 이름을 사용하여 네임스페이스를 생성하고 배포에 사용됩니다. 이 명령은 또한 config/rbac에서 RBAC 매니페스트를 설치합니다.

3. 다음 명령을 실행하여 Operator가 실행 중인지 확인합니다.

```
$ oc get deployment -n <project_name>-system
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
<project_name>-controller-manager       1/1     1            1           8m
```

5.4.7.4. Ansible 로그

Ansible 기반 Operator는 Ansible 실행에 대한 로그를 제공합니다. 이 로그는 Ansible 작업을 디버깅하는 데 유용할 수 있습니다. 로그에는 Operator의 내부 및 Kubernetes와의 상호 작용에 대한 세부 정보가 포함될 수 있습니다.

5.4.7.4.1. Ansible 로그 보기

사전 요구 사항

- 클러스터에서 배포로 실행되는 Ansible 기반 Operator

프로세스

- Ansible 기반 Operator에서 로그를 보려면 다음 명령을 실행합니다.

```
$ oc logs deployment/<project_name>-controller-manager \
  -c manager \ 1
  -n <namespace> 2
```

1 manager 컨테이너의 로그를 확인합니다.

2 make deploy 명령을 사용하여 Operator를 배포로 실행한 경우 <project_name>-system 네임스페이스를 사용합니다.

출력 예

```
{"level":"info","ts":1612732105.0579333,"logger":"cmd","msg":"Version","Go
Version":"go1.15.5","GOOS":"linux","GOARCH":"amd64","ansible-
operator":"v1.10.1","commit":"1abf57985b43bf6a59dcd18147b3c574fa57d3f6"}
{"level":"info","ts":1612732105.0587437,"logger":"cmd","msg":"WATCH_NAMESPACE
environment variable not set. Watching all namespaces.","Namespace":""}
I0207 21:08:26.110949       7 request.go:645] Throttling request took 1.035521578s,
request: GET:https://172.30.0.1:443/apis/flowcontrol.apiserver.k8s.io/v1alpha1?
timeout=32s
{"level":"info","ts":1612732107.768025,"logger":"controller-
runtime.metrics","msg":"metrics server is starting to listen","addr":"127.0.0.1:8080"}
{"level":"info","ts":1612732107.768796,"logger":"watches","msg":"Environment
variable not set; using default
value","envVar":"ANSIBLE_VERBOSITY_MEMCACHED_CACHE_EXAMPLE_COM","def
ault":2}
{"level":"info","ts":1612732107.7688773,"logger":"cmd","msg":"Environment variable
not set; using default
value","Namespace":"","envVar":"ANSIBLE_DEBUG_LOGS","ANSIBLE_DEBUG_LOG
S":false}
{"level":"info","ts":1612732107.7688901,"logger":"ansible-
controller","msg":"Watching
resource","Options.Group":"cache.example.com","Options.Version":"v1","Options.Ki
nd":"Memcached"}
{"level":"info","ts":1612732107.770032,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
I0207 21:08:27.770185       7 leaderelection.go:243] attempting to acquire leader lease
memcached-operator-system/memcached-operator...
{"level":"info","ts":1612732107.770202,"logger":"controller-
runtime.manager","msg":"starting metrics server","path":"/metrics"}
I0207 21:08:27.784854       7 leaderelection.go:253] successfully acquired lease
```

```
memcached-operator-system/memcached-operator
{"level":"info","ts":1612732107.7850506,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612732107.8853772,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612732107.8854098,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting workers","worker
count":4}
```

5.4.7.4.2. 로그에서 전체 Ansible 결과 활성화

환경 변수 **ANSIBLE_DEBUG_LOGS**를 **True**로 설정하여 로그에서 전체 **Ansible** 결과를 확인할 수 있습니다. 이 방법은 디버깅 시 유용할 수 있습니다.

프로세스

- 다음 구성을 포함하도록 **config/manager/manager.yaml** 및 **config/default/manager_auth_proxy_patch.yaml** 파일을 편집합니다.

```
containers:
- name: manager
  env:
  - name: ANSIBLE_DEBUG_LOGS
    value: "True"
```

5.4.7.4.3. 로그에서 verbose 디버깅 활성화

Ansible 기반 **Operator**를 개발하는 동안 로그에서 추가 디버깅을 활성화하는 것이 도움이 될 수 있습니다.

프로세스

- **ansible.sdk.operatorframework.io/verbosity** 주석을 사용자 정의 리소스에 추가하여 원하는 세부 정보 표시 수준을 활성화합니다. 예를 들면 다음과 같습니다.

```
apiVersion: "cache.example.com/v1alpha1"
kind: "Memcached"
metadata:
  name: "example-memcached"
  annotations:
```

```
    "ansible.sdk.operatorframework.io/verbosity": "4"
spec:
  size: 4
```

5.4.8. 사용자 정의 리소스 상태 관리

5.4.8.1. Ansible 기반 Operator의 사용자 정의 리소스 상태 정보

Ansible 기반 Operator는 이전 Ansible 실행에 대한 일반적인 정보를 사용하여 CR(사용자 정의 리소스) status 하위 리소스를 자동으로 업데이트합니다. 여기에는 다음과 같이 성공 및 실패한 작업의 수와 관련 오류 메시지가 포함됩니다.

```
status:
  conditions:
  - ansibleResult:
      changed: 3
      completion: 2018-12-03T13:45:57.13329
      failures: 1
      ok: 6
      skipped: 0
    lastTransitionTime: 2018-12-03T13:45:57Z
    message: 'Status code was -1 and not [200]: Request failed: <urlopen error [Errno
      113] No route to host>'
    reason: Failed
    status: "True"
    type: Failure
  - lastTransitionTime: 2018-12-03T13:46:13Z
    message: Running reconciliation
    reason: Running
    status: "True"
    type: Running
```

Ansible 기반 Operator를 사용하면 Operator 작성자가 operator_sdk.util 컬렉션에 포함된 k8s_status Ansible 모듈로 사용자 정의 상태 값을 제공할 수 있습니다. 그러면 작성자는 원하는 키-값 쌍을 사용하여 Ansible 내에서 status를 업데이트할 수 있습니다.

기본적으로 Ansible 기반 Operator에는 항상 위에 표시된 것처럼 일반 Ansible 실행 출력이 포함됩니다. 애플리케이션에서 Ansible 출력을 통해 상태를 업데이트하지 않도록 하려면 애플리케이션에서 수동으로 상태를 추적하면 됩니다.

5.4.8.2. 수동으로 사용자 정의 리소스 상태 추적

operator_sdk.util 컬렉션을 사용하면 Ansible 기반 Operator를 수정하여 애플리케이션에서 CR(사용자 정의 리소스) 상태를 수동으로 추적할 수 있습니다.

사전 요구 사항

- Operator SDK를 사용하여 Ansible 기반 Operator 프로젝트 생성

프로세스

1. manageStatus 필드를 false로 설정하여 watches.yaml 파일을 업데이트합니다.

```
- version: v1
  group: api.example.com
  kind: <kind>
  role: <role>
  manageStatus: false
```

2. operator_sdk.util.k8s_status Ansible 모듈을 사용하여 하위 리소스를 업데이트합니다. 예를 들어 키 test 및 값 data를 사용하여 업데이트하려면 operator_sdk.util을 다음과 같이 사용하면 됩니다.

```
- operator_sdk.util.k8s_status:
    api_version: app.example.com/v1
    kind: <kind>
    name: "{{ ansible_operator_meta.name }}"
    namespace: "{{ ansible_operator_meta.namespace }}"
    status:
      test: data
```

3. 역할의 meta/main.yml 파일에 컬렉션을 선언할 수 있습니다. 이 파일은 스캐폴드된 Ansible 기반 Operator에 포함됩니다.

```
collections:
  - operator_sdk.util
```

4. 역할 메타에 컬렉션을 선언한 후에는 k8s_status 모듈을 직접 호출할 수 있습니다.

```
k8s_status:
  ...
  status:
    key1: value1
```

5.5. HELM 기반 OPERATOR

5.5.1. Helm 기반 Operator를 위한 Operator SDK 시작하기

Operator SDK에는 Go 코드를 작성하지 않고도 기존 Helm 차트를 활용하여 Kubernetes 리소스를 통

합 애플리케이션으로 배포하는 Operator 프로젝트를 생성하는 옵션이 포함되어 있습니다.

Operator 개발자는 Operator SDK에서 제공하는 툴 및 라이브러리를 사용하여 Helm 기반 Operator를 설정 및 실행하는 기본 동작을 설명하기 위해 Nginx에 대한 Helm 기반 Operator 예제를 빌드하고 클러스터에 배포할 수 있습니다.

5.5.1.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터가 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치
- OpenShift CLI 시작하기

5.5.1.2. Helm 기반 Operator 생성 및 배포

Operator SDK를 사용하여 Nginx에 대한 간단한 Helm 기반 Operator를 빌드하고 배포할 수 있습니다.

절차

1. 프로젝트를 생성합니다.
   a. 프로젝트 디렉토리를 생성합니다.
      -

```
$ mkdir nginx-operator
```

b. 프로젝트 디렉터리로 변경합니다.

```
$ cd nginx-operator
```

c. helm 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.

```
$ operator-sdk init \
  --plugins=helm
```

2. API를 생성합니다.

간단한 Nginx API를 생성합니다.

```
$ operator-sdk create api \
  --group demo \
  --version v1 \
  --kind Nginx
```

이 API는 helm create 명령의 기본 제공 Helm 차트 상용구를 사용합니다.

3. Operator 이미지를 빌드하여 내보냅니다.

기본 Makefile 대상을 사용하여 Operator를 빌드하고 내보냅니다. 내보낼 수 있는 레지스트리를 사용하는 이미지의 가져오기 사양에 IMG를 설정합니다.

```
$ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

4. Operator를 실행합니다.

a. CRD를 설치합니다.

```
$ make install
```

b. 클러스터에 프로젝트를 배포합니다. 내보낸 이미지에 IMG를 설정합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

5. SCC(보안 컨텍스트 제약 조건)를 추가합니다.

Nginx 서비스 계정에는 OpenShift Container Platform에서 실행할 수 있는 권한이 필요합니다. nginx-sample Pod의 서비스 계정에 다음 SCC를 추가합니다.

```
$ oc adm policy add-scc-to-user \
    anyuid system:serviceaccount:nginx-operator-system:nginx-sample
```

6. 샘플 CR(사용자 정의 리소스)을 생성합니다.

a. 샘플 CR을 생성합니다.

```
$ oc apply -f config/samples/demo_v1_nginx.yaml \
    -n nginx-operator-system
```

b. CR에서 Operator를 조정하는지 확인합니다.

```
$ oc logs deployment.apps/nginx-operator-controller-manager \
    -c manager \
    -n nginx-operator-system
```

7.

```
$ oc delete -f config/samples/demo_v1_nginx -n nginx-operator-system
```

8. 정리합니다.

다음 명령을 실행하여 이 절차의 일부로 생성된 리소스를 정리합니다.

```
$ make undeploy
```

5.5.1.3. 다음 단계

-

Helm 기반 Operator를 빌드하는 방법에 대한 자세한 내용은 Helm 기반 Operator를 위한 Operator SDK 튜토리얼을 참조하십시오.

5.5.2. helm 기반 Operator를 위한 Operator SDK 튜토리얼

Operator 개발자는 Operator SDK에서 Helm 지원을 활용하여 Nginx에 대한 Helm 기반 Operator 예제를 빌드하고 라이프사이클을 관리할 수 있습니다. 이 튜토리얼에서는 다음 프로세스를 안내합니다.

- Nginx 배포 생성
- 배포 크기가 Nginx CR(사용자 정의 리소스) 사양에 지정된 것과 같은지 확인합니다.
- 상태 작성기를 사용하여 Nginx CR 상태를 nginx Pod의 이름으로 업데이트합니다.

이 프로세스는 Operator 프레임워크의 두 가지 주요 요소를 사용하여 수행됩니다.

Operator SDK

operator-sdk CLI 툴 및 controller-runtime 라이브러리 API

OLM(Operator Lifecycle Manager)

클러스터에 대한 Operator의 설치, 업그레이드, RBAC(역할 기반 액세스 제어)

참고

이 튜토리얼에는 Helm 기반 Operator용 Operator SDK 시작하기보다 자세히 설명되어 있습니다.

5.5.2.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
-

cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함

- 클러스터가 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치
- OpenShift CLI 시작하기

5.5.2.2. 프로젝트 생성

Operator SDK CLI를 사용하여 nginx-operator라는 프로젝트를 생성합니다.

프로세스

1. 프로젝트에 사용할 디렉터리를 생성합니다.

```
$ mkdir -p $HOME/projects/nginx-operator
```

2. 디렉터리로 변경합니다.

```
$ cd $HOME/projects/nginx-operator
```

3. helm 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.

```
$ operator-sdk init \
  --plugins=helm \
  --domain=example.com \
  --group=demo \
  --version=v1 \
  --kind=Nginx
```

참고

*기본적으로 **helm** 플러그인은 상용구 **Helm** 차트를 사용하여 프로젝트를 초기화합니다. 기존 **Helm** 차트를 사용하는 프로젝트를 초기화하려면 **--helm-chart** 플래그와 같은 추가 플래그를 사용하면 됩니다.*

*init 명령은 **API** 버전이 **example.com/v1**이고 종류가 **Nginx**인 리소스를 조사하기 위해 특별히 **nginx-operator** 프로젝트를 생성합니다.*

4. *Helm 기반 프로젝트의 경우 **init** 명령은 차트의 기본 매니페스트에 의해 배포되는 리소스를 기반으로 **config/rbac/role.yaml** 파일에 **RBAC** 규칙을 생성합니다. 이 파일에 생성된 규칙이 **Operator**의 권한 요구 사항을 충족하는지 확인합니다.*

5.5.2.2.1. 기존 Helm 차트

*상용구 **Helm** 차트로 프로젝트를 생성하는 대신 다음 플래그를 사용하여 로컬 파일 시스템 또는 원격 차트 리포지토리에서 기존 차트를 사용할 수 있습니다.*

- ***--helm-chart***
- ***--helm-chart-repo***
- ***--helm-chart-version***

***--helm-chart** 플래그가 지정되면 **--group, --version, --kind** 플래그가 선택 사항이 됩니다. 설정되지 않으면 다음 기본값이 사용됩니다.*

플래그	값
--domain	**my.domain**
--group	**charts**
--version	**v1**
--kind	지정된 차트에서 추론됨

--helm-chart 플래그가 로컬 차트 아카이브(예: example-chart-1.2.0.tgz 또는 디렉터리)를 지정하는 경우 차트를 검증하고 압축을 풀거나 프로젝트에 복사합니다. 그러지 않으면 Operator SDK가 원격 리포지토리에서 차트를 가져옵니다.

--helm-chart-repo 플래그로 사용자 정의 리포지토리 URL이 지정되지 않는 경우 다음 차트 참조 형식이 지원됩니다.

형식	설명
<repo_name>/<chart_name>	**$HELM_HOME/repositories/repositories.yaml** 파일에 지정된 대로 helm 차트 리포지토리 **<repo_name>**에서 Helm 차트 **<chart_name>**을 가져옵니다. **helm repo add** 명령을 사용하여 이 파일을 구성합니다.
<url>	지정된 URL에서 Helm 차트 아카이브를 가져옵니다.

사용자 정의 리포지토리 URL이 --helm-chart-repo로 지정된 경우 다음 차트 참조 형식이 지원됩니다.

형식	설명
<chart_name>	**--helm-chart-repo** URL 값으로 지정된 Helm 차트 리포지토리에서 **<chart_name>**이라는 Helm 차트를 가져옵니다.

--helm-chart-version 플래그를 설정하지 않으면 Operator SDK에서 사용 가능한 최신 버전의 Helm 차트를 가져옵니다. 그러지 않으면 지정된 버전을 가져옵니다. --helm-chart 플래그로 지정한 차트에서 특정 버전을 참조하는 경우(예: 로컬 경로 또는 URL) 선택적 --helm-chart-version 플래그는 사용되지 않습니다.

자세한 내용 및 예를 보려면 다음을 실행합니다.

```
$ operator-sdk init --plugins helm --help
```

5.5.2.2.2. PROJECT 파일

operator-sdk init 명령으로 생성된 파일 중에는 Kubebuilder PROJECT 파일이 있습니다. 이어서 프로젝트 루트에서 실행되는 operator-sdk 명령과 help 출력에서는 이 파일을 읽고 프로젝트 유형이 Helm임을 확인합니다. 예를 들면 다음과 같습니다.

```
domain: example.com
```

```
layout:
- helm.sdk.operatorframework.io/v1
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
projectName: nginx-operator
resources:
- api:
    crdVersion: v1
    namespaced: true
  domain: example.com
  group: demo
  kind: Nginx
  version: v1
version: "3"
```

5.5.2.3. Operator 논리 이해

이 예제에서 **nginx-operator** 프로젝트는 각 **Nginx CR**(사용자 정의 리소스)에 대해 다음과 같은 조정 논리를 실행합니다.

- **Nginx** 배포가 없는 경우 해당 배포를 생성합니다.
- **Nginx** 서비스가 없는 경우 해당 서비스를 생성합니다.
- **Nginx** 수신이 활성화되어 있지만 없는 경우 해당 수신을 생성합니다.
- 배포, 서비스, 선택적 수신이 **Nginx CR**에 지정된 대로 원하는 구성(예 : 복제본 수, 이미지, 서비스 유형)과 일치하는지 확인합니다.

기본적으로 **nginx-operator** 프로젝트는 **watches.yaml** 파일에 표시된 **Nginx** 리소스 이벤트를 조사하고 지정된 차트를 사용하여 **Helm** 릴리스를 실행합니다.

```
# Use the 'create api' subcommand to add watches to this file.
- group: demo
  version: v1
  kind: Nginx
  chart: helm-charts/nginx
# +kubebuilder:scaffold:watch
```

5.5.2.3.1. 샘플 Helm 차트

Helm Operator 프로젝트가 생성되면 Operator SDK는 간단한 Nginx 릴리스에 대한 일련의 템플릿이 포함된 샘플 Helm 차트를 생성합니다.

이 예제에서는 Helm 차트 개발자가 릴리스에 대한 유용한 정보를 전달하는 데 사용하는 NOTES.txt 템플릿과 함께 배포, 서비스, 수신 리소스에 대해 템플릿을 사용할 수 있습니다.

Helm 차트에 대해 잘 모르는 경우 Helm 개발자 설명서를 검토하십시오.

5.5.2.3.2. 사용자 정의 리소스 사양 수정

Helm은 값이라는 개념을 사용하여 values.yaml 파일에 정의된 Helm 차트 기본값에 대한 사용자 정의 기능을 제공합니다.

CR(사용자 정의 리소스) 사양에 원하는 값을 설정하여 이러한 기본값을 덮어쓸 수 있습니다. 예를 들어 복제본 수를 사용할 수 있습니다.

프로세스

1. helm-charts/nginx/values.yaml 파일에는 replicaCount라는 값이 있으며 기본적으로 1로 설정되어 있습니다. 배포에 Nginx 인스턴스를 두 개 포함하려면 CR 사양에 replicaCount: 2를 포함해야 합니다.

   config/samples/demo_v1_nginx.yaml 파일을 편집하여 replicaCount: 2를 설정합니다.

   ```
   apiVersion: demo.example.com/v1
   kind: Nginx
   metadata:
     name: nginx-sample
   ...
   spec:
   ...
     replicaCount: 2
   ```

2. 마찬가지로 기본 서비스 포트는 80으로 설정됩니다. 8080을 사용하려면 config/samples/demo_v1_nginx.yaml 파일을 편집하여 서비스 포트 덮어쓰기를 추가하는 spec.port: 8080을 설정합니다.

   ```
   apiVersion: demo.example.com/v1
   kind: Nginx
   metadata:
   ```

```
  name: nginx-sample
spec:
  replicaCount: 2
  service:
    port: 8080
```

Helm Operator는 helm install -f ./overrides.yaml 명령과 마찬가지로 값 파일의 콘텐츠인 것처럼 전체 사양을 적용합니다.

5.5.2.4. 프록시 지원 활성화

Operator 작성자는 네트워크 프록시를 지원하는 Operator를 개발할 수 있습니다. 클러스터 관리자는 OLM(Operator Lifecycle Manager)에서 처리하는 환경 변수에 대한 프록시 지원을 구성합니다. 프록시된 클러스터를 지원하려면 Operator에서 다음 표준 프록시 변수에 대한 환경을 검사하고 해당 값을 Operands에 전달해야 합니다.

- HTTP_PROXY
- HTTPS_PROXY
- NO_PROXY

참고

이 튜토리얼에서는 예제 환경 변수로 HTTP_PROXY를 사용합니다.

사전 요구 사항

- 클러스터 전체 egress 프록시가 활성화된 클러스터입니다.

절차

1. overrideValues 필드를 추가하여 환경 변수에 따라 덮어쓰기를 포함하도록 watches.yaml 파일을 편집합니다.

```
...
- group: demo.example.com
  version: v1alpha1
  kind: Nginx
```

```
  chart: helm-charts/nginx
  overrideValues:
    proxy.http: $HTTP_PROXY
...
```

2.

```
...
proxy:
  http: ""
  https: ""
  no_proxy: ""
```

3. 차트 템플릿이 변수 사용을 지원하는지 확인하려면 다음을 포함하도록 **helm-charts/nginx/templates/deployment.yaml** 파일에서 차트 템플릿을 편집합니다.

```
containers:
  - name: {{ .Chart.Name }}
    securityContext:
      - toYaml {{ .Values.securityContext | nindent 12 }}
    image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default
.Chart.AppVersion }}"
    imagePullPolicy: {{ .Values.image.pullPolicy }}
    env:
      - name: http_proxy
        value: "{{ .Values.proxy.http }}"
```

4. **config/manager/manager.yaml** 파일에 다음을 추가하여 **Operator** 배포에서 환경 변수를 설정합니다.

```
containers:
 - args:
   - --leader-elect
   - --leader-election-id=ansible-proxy-demo
   image: controller:latest
   name: manager
   env:
     - name: "HTTP_PROXY"
       value: "http_proxy_test"
```

5.5.2.5. Operator 실행

다음 세 가지 방법으로 **Operator SDK CLI**를 사용하여 **Operator**를 빌드하고 실행할 수 있습니다.

- *Go 프로그램으로 클러스터 외부에서 로컬로 실행합니다.*

- *클러스터에서 배포로 실행합니다.*

- *Operator를 번들로 제공하고 OLM(Operator Lifecycle Manager)을 사용하여 클러스터에 배포합니다.*

5.5.2.5.1. 클러스터 외부에서 로컬로 실행

Operator 프로젝트를 클러스터 외부의 Go 프로그램으로 실행할 수 있습니다. 이는 배포 및 테스트 속도를 높이기 위한 개발 목적에 유용합니다.

절차

- *다음 명령을 실행하여 ~/.kube/config 파일에 구성된 클러스터에 CRD(사용자 정의 리소스 정의)를 설치하고 Operator를 로컬로 실행합니다.*

```
$ make install run
```

출력 예

```
...
{"level":"info","ts":1612652419.9289865,"logger":"controller-
runtime.metrics","msg":"metrics server is starting to listen","addr":":8080"}
{"level":"info","ts":1612652419.9296563,"logger":"helm.controller","msg":"Watching
resource","apiVersion":"demo.example.com/v1","kind":"Nginx","namespace":"","reco
ncilePeriod":"1m0s"}
{"level":"info","ts":1612652419.929983,"logger":"controller-
runtime.manager","msg":"starting metrics server","path":"/metrics"}
{"level":"info","ts":1612652419.930015,"logger":"controller-
runtime.manager.controller.nginx-controller","msg":"Starting
EventSource","source":"kind source: demo.example.com/v1, Kind=Nginx"}
{"level":"info","ts":1612652420.2307851,"logger":"controller-
runtime.manager.controller.nginx-controller","msg":"Starting Controller"}
{"level":"info","ts":1612652420.2309358,"logger":"controller-
runtime.manager.controller.nginx-controller","msg":"Starting workers","worker
count":8}
```

5.5.2.5.2. 클러스터에서 배포로 실행

Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

절차

1. 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

   a. 이미지를 빌드합니다.

   ```
   $ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. 이미지를 리포지토리로 내보냅니다.

   ```
   $ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   두 명령 모두 이미지의 이름과 태그(예: IMG=<registry>/<user>/<image_name>:<tag>)를 Makefile에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 IMG ?= controller:latest 값을 수정합니다.

2. 다음 명령을 실행하여 Operator를 배포합니다.

   ```
   $ make deploy IMG=<registry>/<user>/<image_name>:<tag>
   ```

기본적으로 이 명령은 **<project_name>-system** 형식으로 된 **Operator** 프로젝트 이름을 사용하여 네임스페이스를 생성하고 배포에 사용됩니다. 이 명령은 또한 **config/rbac**에서 **RBAC** 매니페스트를 설치합니다.

3. 다음 명령을 실행하여 **Operator**가 실행 중인지 확인합니다.

```
$ oc get deployment -n <project_name>-system
```

출력 예

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
<project_name>-controller-manager   1/1     1            1           8m
```

5.5.2.5.3. Operator 번들링 및 Operator Lifecycle Manager를 통한 배포

5.5.2.5.3.1. Operator 번들

Operator 번들 형식은 **Operator SDK** 및 **Operator Lifecycle Manager (OLM)**의 기본 패키지 메서드입니다. **Operator SDK**를 사용하여 **Operator** 프로젝트를 번들 이미지로 빌드하고 푸시하여 **OLM**에서 **Operator**를 사용할 수 있습니다.

사전 요구 사항

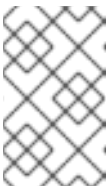
- 개발 워크스테이션에 **Operator SDK CLI**가 설치됨
- **OpenShift CLI(oc) v4.11** 이상이 설치됨
- **Operator SDK**를 사용하여 **Operator** 프로젝트를 초기화함

절차

1. **Operator** 프로젝트 디렉터리에서 다음 **make** 명령을 실행하여 **Operator** 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 **IMG** 인수를 수정합니다. **Quay.io**와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

a. 이미지를 빌드합니다.

```
$ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
```



참고

b. 이미지를 리포지토리로 내보냅니다.

```
$ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
```

2. Operator SDK generate bundle 및 bundle validate 명령을 비롯한 다양한 명령을 호출하는 make bundle 명령을 실행하여 Operator 번들 매니페스트를 생성합니다.

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Operator의 번들 매니페스트는 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. make bundle 명령은 Operator 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

- ClusterServiceVersion 오브젝트를 포함하는 bundle/manifests라는 번들 매니페스트 디렉터리
- bundle/metadata라는 번들 메타데이터 디렉터리
- config/crd 디렉터리의 모든 CRD(사용자 정의 리소스 정의)
- Dockerfile bundle.Dockerfile

그런 다음 operator-sdk bundle validate를 사용하여 이러한 파일을 자동으로 검증하고 디스크상의 번들 표현이 올바른지 확인합니다.

3. 다음 명령을 실행하여 번들 이미지를 빌드하고 내보냅니다. OLM에서는 하나 이상의 번들 이미지를 참조하는 인덱스 이미지를 통해 Operator 번들을 사용합니다.

a. 번들 이미지를 빌드합니다. 이미지를 내보낼 레지스트리, 사용자 네임스페이스, 이미지 태그에 대한 세부 정보를 사용하여 **BUNDLE_IMG**를 설정합니다.

```
$ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
```

b. 번들 이미지를 내보냅니다.

```
$ docker push <registry>/<user>/<bundle_image_name>:<tag>
```

5.5.2.5.3.2. Operator Lifecycle Manager를 사용하여 Operator 배포

OLM(Operator Lifecycle Manager)은 **Kubernetes** 클러스터에서 **Operator** 및 관련 서비스를 설치, 업데이트하고 라이프사이클을 관리하는 데 도움이 됩니다. **OLM**은 기본적으로 **OpenShift Container Platform**에 설치되고 **Kubernetes** 확장으로 실행되므로 추가 툴 없이 모든 **Operator** 라이프사이클 관리 기능에 웹 콘솔과 **OpenShift CLI(oc)**를 사용할 수 있습니다.

Operator 번들 형식은 **Operator SDK** 및 **OLM**의 기본 패키지 메서드입니다. **Operator SDK**를 사용하여 **OLM**에서 번들 이미지를 신속하게 실행하여 올바르게 실행되는지 확인할 수 있습니다.

사전 요구 사항

- 개발 워크스테이션에 **Operator SDK CLI**가 설치됨
- **Operator** 번들 이미지를 빌드하여 레지스트리로 내보냄
- **Kubernetes** 기반 클러스터에 **OLM**이 설치되어 있음(**apiextensions.k8s.io/v1 CRD**(예: **OpenShift Container Platform 4.11**)를 사용하는 경우 **v1.16.0** 이상
- **cluster-admin** 권한이 있는 계정을 사용하여 **oc**로 클러스터에 로그인됨

프로세스

1.

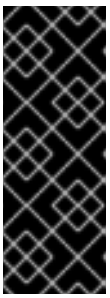
```
$ operator-sdk run bundle \ 1
  -n <namespace> \ 2
  <registry>/<user>/<bundle_image_name>:<tag> 3
```

1

2

-n 플래그를 추가하면 설치에 다른 네임스페이스 범위를 설정할 수 있습니다.

3



중요

OpenShift Container Platform 4.11부터 run bundle 명령은 기본적으로 Operator 카탈로그의 파일 기반 카탈로그 형식을 지원합니다. Operator 카탈로그의 더 이상 사용되지 않는 SQLite 데이터베이스 형식은 계속 지원되지만 향후 릴리스에서 제거됩니다.

이 명령은 다음 작업을 수행합니다.

- 번들 이미지를 참조하는 인덱스 이미지를 생성합니다. 인덱스 이미지는 불투명하고 일시적이지만 프로덕션에서 카탈로그에 번들을 추가하는 방법을 정확하게 반영합니다.
- OperatorHub에서 Operator를 검색할 수 있도록 새 인덱스 이미지를 가리키는 카탈로그 소스를 생성합니다.
-

5.5.2.6. 사용자 정의 리소스 생성

Operator가 설치되면 Operator에서 현재 클러스터에 제공하는 CR(사용자 정의 리소스)을 생성하여 Operator를 테스트할 수 있습니다.

사전 요구 사항

- *Nginx CR을 제공하는 Nginx Operator 예제가 클러스터에 설치되어 있음*

절차

1. *Operator가 설치된 네임스페이스로 변경합니다. 예를 들어 make deploy 명령을 사용하여 Operator를 배포한 경우 다음을 실행합니다.*

```
$ oc project nginx-operator-system
```

2. *다음 사양을 포함하도록 config/samples/demo_v1_nginx.yaml에서 샘플 Nginx CR 매니페스트를 편집합니다.*

```
apiVersion: demo.example.com/v1
kind: Nginx
metadata:
  name: nginx-sample
...
spec:
...
  replicaCount: 3
```

3. *Nginx 서비스 계정에는 OpenShift Container Platform에서 실행할 수 있는 권한이 필요합니다. nginx-sample Pod의 서비스 계정에 다음 SCC(보안 컨텍스트 제약 조건)를 추가합니다.*

```
$ oc adm policy add-scc-to-user \
    anyuid system:serviceaccount:nginx-operator-system:nginx-sample
```

4. *CR을 생성합니다.*

```
$ oc apply -f config/samples/demo_v1_nginx.yaml
```

5. *Nginx Operator에서 샘플 CR에 대한 배포를 올바른 크기로 생성하는지 확인합니다.*

```
$ oc get deployments
```

출력 예

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
nginx-operator-controller-manager     1/1     1            1           8m
nginx-sample                          3/3     3            3           1m
```

6. *Pod 및 CR 상태를 확인하여 상태가 Nginx Pod 이름으로 업데이트되었는지 확인합니다.*

a. *Pod를 확인합니다.*

```
$ oc get pods
```

출력 예

```
NAME                              READY    STATUS    RESTARTS   AGE
nginx-sample-6fd7c98d8-7dqdr      1/1      Running   0          1m
nginx-sample-6fd7c98d8-g5k7v      1/1      Running   0          1m
nginx-sample-6fd7c98d8-m7vn7      1/1      Running   0          1m
```

b. *CR 상태 확인:*

```
$ oc get nginx/nginx-sample -o yaml
```

출력 예

```
apiVersion: demo.example.com/v1
kind: Nginx
metadata:
...
  name: nginx-sample
...
spec:
  replicaCount: 3
status:
  nodes:
```

```
  - nginx-sample-6fd7c98d8-7dqdr
  - nginx-sample-6fd7c98d8-g5k7v
  - nginx-sample-6fd7c98d8-m7vn7
```

7. 배포 크기를 업데이트합니다.

a. config/samples/demo_v1_nginx.yaml 파일을 업데이트하여 Nginx CR의 spec.size 필드를 3에서 5로 변경합니다.

```
$ oc patch nginx nginx-sample \
  -p '{"spec":{"replicaCount": 5}}' \
  --type=merge
```

b. Operator에서 배포 크기를 변경하는지 확인합니다.

```
$ oc get deployments
```

출력 예

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-operator-controller-manager   1/1     1            1           10m
nginx-sample                        5/5     5            5           3m
```

8.

```
$ oc delete -f config/samples/demo_v1_nginx.yaml
```

9. 이 튜토리얼의 일부로 생성된 리소스를 정리합니다.

- make deploy 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ make undeploy
```

- *operator-sdk run bundle 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.*

```
$ operator-sdk cleanup <project_name>
```

5.5.2.7. 추가 리소스

- *Operator SDK에서 생성한 디렉터리 구조에 대한 자세한 내용은 Helm 기반 Operator의 프로젝트 레이아웃 을 참조하십시오.*

- *클러스터 전체 송신 프록시가 구성된 경우 클러스터 관리자는 프록시 설정을 재정의하거나 OLM(Operator Lifecycle Manager)에서 실행되는 특정 Operator에 대한 사용자 정의 CA 인증서를 삽입 할 수 있습니다.*

5.5.3. Helm 기반 Operator의 프로젝트 레이아웃

operator-sdk CLI에서는 각 Operator 프로젝트에 대해 다양한 패키지 및 파일을 생성하거나 스캐폴드를 지정할 수 있습니다.

5.5.3.1. Helm 기반 프로젝트 레이아웃

operator-sdk init --plugins helm 명령을 사용하여 생성된 Helm 기반 Operator 프로젝트에는 다음 디렉터리 및 파일이 포함됩니다.

파일/폴더	목적
config/	Kubernetes 클러스터에 Operator를 배포하는 Kustomize 매니페스트입니다.
helm-charts/	**operator-sdk create api** 명령을 사용하여 초기화한 Helm 차트입니다.
Dockerfile	**make docker-build** 명령을 사용하여 Operator 이미지를 빌드하는 데 사용합니다.
watches.yaml	GVK(그룹/버전/종류) 및 Helm 차트 위치입니다.
Makefile	프로젝트 관리에 사용되는 대상입니다.
PROJECT	Operator의 메타데이터 정보가 포함된 YAML 파일입니다.

5.5.4.

5.5.4.1.

사전 요구 사항

-
-

프로세스

1.

```
...
spec:
  template:
    spec:
      containers:
      - name: kube-rbac-proxy
        image: registry.redhat.io/openshift4/ose-kube-rbac-proxy:v4.11 1
        args:
        - "--secure-listen-address=0.0.0.0:8443"
        - "--upstream=http://127.0.0.1:8080/"
        - "--logtostderr=true"
        - "--v=0" 2
...
resources:
  limits:
    cpu: 500m
    memory: 128Mi
  requests:
    cpu: 5m
    memory: 64Mi 3
```

1

2

3

2.

a.

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)
...
```

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)

# BUNDLE_GEN_FLAGS are the flags passed to the operator-sdk generate bundle
command
BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

# USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
# You can enable this value if you would like to use SHA Based Digests
# To enable set flag to true
USE_IMAGE_DIGESTS ?= false
ifeq ($(USE_IMAGE_DIGESTS), true)
 BUNDLE_GEN_FLAGS += --use-image-digests
endif
```

b.

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --
overwrite --version $(VERSION) $(BUNDLE_METADATA_OPTS)
```

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS)
```

c.

```
.PHONY: opm
OPM = ./bin/opm
opm: ## Download opm locally if necessary.
ifeq (,$(wildcard $(OPM)))
ifeq (,$(shell which opm 2>/dev/null))
 @{ \
 set -e ;\
 mkdir -p $(dir $(OPM)) ;\
 OS=$(shell go env GOOS) && ARCH=$(shell go env GOARCH) && \
 curl -sSLo $(OPM) https://github.com/operator-framework/operator-
registry/releases/download/v1.23.0/$${OS}-$${ARCH}-opm ;\ 1
 chmod +x $(OPM) ;\
 }
else
OPM = $(shell which opm)
endif
endif
```

1

d.

```
$ make
```

2

3.

Dockerfile 에

```
FROM registry.redhat.io/openshift4/ose-helm-operator:v4.11 1
```

1

5.5.4.2. 추가 리소스

- 번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션
-
- Operator SDK v1.10.1의 프로젝트 업그레이드
- Operator SDK v1.8.0의 프로젝트 업그레이드

5.5.5. Operator SDK의 Helm 지원

5.5.5.1. Helm 차트

Operator 프로젝트를 생성하기 위한 Operator SDK 옵션 중 하나는 Go 코드를 작성하지 않고 Kubernetes 리소스를 통합 애플리케이션으로 배포하기 위해 기존 Helm 차트를 활용하는 것입니다. 이러한 Helm 기반 Operator는 차트의 일부로 생성되는 Kubernetes 오브젝트에 변경 사항을 적용해야 하기 때문에 롤아웃 시 논리가 거의 필요하지 않은 상태 비저장 애플리케이션에서 잘 작동하도록 설계되었습니다. 이는 제한적으로 들릴 수 있지만 Kubernetes 커뮤니티에서 빌드한 Helm 차트의 확산으로 알 수 있듯이 대용량 사용 사례에도 충분할 수 있습니다.

Operator의 주요 기능은 애플리케이션 인스턴스를 표시하는 사용자 정의 오브젝트에서 읽고 원하는 상태가 실행 중인 것과 일치하도록 하는 것입니다. Helm 기반 Operator의 경우 오브젝트의 spec 필드는

일반적으로 Helm values.yaml 파일에 설명된 구성 옵션의 목록입니다. Helm CLI(예: helm install -f values.yaml)를 사용하여 이러한 값을 플래그로 설정하는 대신 CR(사용자 정의 리소스) 내에 표시할 수 있습니다. 그러면 네이티브 Kubernetes 개체로서 여기에 적용된 RBAC 및 감사 추적의 이점을 활용할 수 있습니다.

Tomcat이라는 간단한 CR 예제를 살펴보겠습니다.

```
apiVersion: apache.org/v1alpha1
kind: Tomcat
metadata:
  name: example-app
spec:
  replicaCount: 2
```

이 경우 replicaCount 값인 2가 다음이 사용되는 차트의 템플릿으로 전파됩니다.

```
{{ .Values.replicaCount }}
```

Operator를 빌드하고 배포한 후에는 CR 인스턴스를 새로 생성하여 앱의 새 인스턴스를 배포하거나 oc 명령을 사용하여 모든 환경에서 실행 중인 다른 인스턴스를 나열할 수 있습니다.

```
$ oc get Tomcats --all-namespaces
```

Helm CLI를 사용하거나 Tiller를 설치할 필요가 없습니다. Helm 기반 Operator는 Helm 프로젝트에서 코드를 가져옵니다. Operator의 인스턴스를 실행하고 CRD(사용자 정의 리소스 정의)를 사용하여 CR을 등록하기만 하면 됩니다. RBAC를 준수하기 때문에 프로덕션이 변경되는 것을 더 쉽게 방지할 수 있습니다.

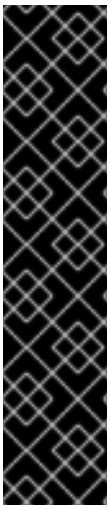
5.5.6. 하이브리드 Helm Operator를 위한 Operator SDK 튜토리얼

Operator SDK의 표준 Helm 기반 Operator 지원은 Operator 완성 모델의 Auto Pilot 기능(level V)에 도달한 Go 기반 및 Ansible 기반 Operator 지원에 비해 기능이 제한되어 있습니다.

하이브리드 Helm Operator는 Go API를 통해 기존 Helm 기반 지원 기능을 향상시킵니다. 이 Helm 및 Go의 하이브리드 접근 방식을 통해 Operator SDK를 사용하면 Operator 작성자가 다음 프로세스를 사용할 수 있습니다.

- Helm과 동일한 프로젝트에서 Go API에 대한 기본 구조 또는 스캐폴드 를 생성합니다.
-

하이브리드 **Helm Operator**에서 제공하는 라이브러리를 통해 프로젝트의 **main.go** 파일에서 **Helm** 조정 프로그램을 구성합니다.



중요

하이브리드 **Helm Operator**는 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.

-
-
-

5.5.6.1. 사전 요구 사항

- **Operator SDK CLI**가 설치됨
- **OpenShift CLI(oc) v4.11** 이상이 설치됨
-
-

추가 리소스

- *Operator SDK CLI 설치*
- *OpenShift CLI 시작하기*

5.5.6.2. 프로젝트 생성

Operator SDK CLI를 사용하여 memcached-operator라는 프로젝트를 생성합니다.

절차

1. *프로젝트에 사용할 디렉터리를 생성합니다.*

```
$ mkdir -p $HOME/github.com/example/memcached-operator
```

2. *디렉터리로 변경합니다.*

```
$ cd $HOME/github.com/example/memcached-operator
```

3.

```
$ operator-sdk init \
  --plugins=hybrid.helm.sdk.operatorframework.io \
  --project-version="3" \
  --domain example.com \
  --repo=github.com/example/memcached-operator
```

추가 리소스

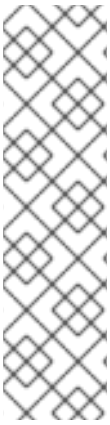
-

5.5.6.3.

절차

-

```
$ operator-sdk create api \
    --plugins helm.sdk.operatorframework.io/v1 \
    --group cache \
    --version v1 \
    --kind Memcached
```



참고

```
$ operator-sdk create api --plugins helm.sdk.operatorframework.io/v1 --help
```

추가 리소스

- *기존 Helm 차트*

5.5.6.3.1. Helm API의 Operator 논리

기본적으로 스캐폴드된 Operator 프로젝트는 watches.yaml 파일에 표시된 Memcached 리소스 이벤트를 감시하고 지정된 차트를 사용하여 Helm 릴리스를 실행합니다.

예 5.2. watches.yaml 파일의 예

```
# Use the 'create api' subcommand to add watches to this file.
- group: cache.my.domain
  version: v1
  kind: Memcached
  chart: helm-charts/memcached
#+kubebuilder:scaffold:watch
```

추가 리소스

- *차트를 통해 Helm Operator 논리를 사용자 정의하는 방법에 대한 자세한 내용은 Operator 논리 이해를 참조하십시오.*

5.5.6.3.2. 제공된 라이브러리 API를 사용하여 사용자 정의 Helm 조정기 구성

기존 Helm 기반 Operator의 단점은 사용자가 추상화되기 때문에 Helm 조정기를 구성할 수 없다는 것입니다. Helm 기반 Operator가 기존 Helm 차트를 재사용하는 Seamless Upgrades 기능(레벨 II 이상)에 도달할 수 있도록, Go와 Helm Operator 유형 간의 하이브리드에는 값이 추가됩니다.

helm-operator-plugins 라이브러리에 제공된 API를 사용하여 Operator 작성자는 다음과 같은 구성을 만들 수 있습니다.

- 클러스터 상태에 따라 값 매핑 사용자 정의
- 조정자의 이벤트 레코드를 구성하여 특정 이벤트에서 코드 실행
- 조정자의 로거 사용자 지정
- 조정자가 조사한 사용자 정의 리소스에 있는 주석을 기반으로 Helm의 작업을 구성할 수 있도록 Install,Upgrade, Uninstall annotations
- 사전 및 사후 후크로 실행되도록 조정기 구성

조정기에 대한 위의 구성은 main.go 파일에서 수행할 수 있습니다.

main.go 파일 예

```
// Operator's main.go
// With the help of helpers provided in the library, the reconciler can be
// configured here before starting the controller with this reconciler.
reconciler := reconciler.New(
 reconciler.WithChart(*chart),
 reconciler.WithGroupVersionKind(gvk),
)

if err := reconciler.SetupWithManager(mgr); err != nil {
 panic(fmt.Sprintf("unable to create reconciler: %s", err))
}
```

5.5.6.4. Go API 생성

Operator SDK CLI를 사용하여 Go API를 생성합니다.

절차

1. 다음 명령을 실행하여 그룹 캐시 가 있는 Go API, 버전 v1 및 kind MemcachedBackup 을 생성합니다.

```
$ operator-sdk create api \
  --group=cache \
  --version v1 \
  --kind MemcachedBackup \
  --resource \
  --controller \
  --plugins=go/v3
```

2. 메시지가 표시되면 리소스 및 컨트롤러를 모두 생성하도록 y 를 입력합니다.

```
$ Create Resource [y/n]
y
Create Controller [y/n]
y
```

이 절차에서는 api/v1/memcachedbackup_types.go 에 MemcachedBackup 리소스 API를 생성하고 controllers/memcachedbackup_controller.go 에 컨트롤러를 생성합니다.

5.5.6.4.1. API 정의

MemcachedBackup CR(사용자 정의 리소스)의 API를 정의합니다.

MemcachedBackup 유형을 정의하여 MemcachedBackupSpec.Size 필드를 지정하여 배포할 MemcachedBackupSpec.Size 필드가 있고 CR의 Pod 이름을 저장할 MemcachedBackupStatus.Nodes 필드를 나타냅니다.

참고

Node 필드는 상태 필드의 예를 설명하는 데 사용됩니다.

절차

1. api/v1/memcachedbackup_types.go 파일에서 다음 spec 및 status 를 갖도록 Go 유형 정의를 수정하여 MemcachedBackup CR의 API를 정의합니다.

예 5.3. api/v1/memcachedbackup_types.go 파일의 예

```
// MemcachedBackupSpec defines the desired state of MemcachedBackup
type MemcachedBackupSpec struct {
 // INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
 // Important: Run "make" to regenerate code after modifying this file

 //+kubebuilder:validation:Minimum=0
 // Size is the size of the memcached deployment
 Size int32 `json:"size"`
}

// MemcachedBackupStatus defines the observed state of MemcachedBackup
type MemcachedBackupStatus struct {
 // INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
 // Important: Run "make" to regenerate code after modifying this file
 // Nodes are the names of the memcached pods
 Nodes []string `json:"nodes"`
}
```

2. 리소스 유형에 대해 생성된 코드를 업데이트합니다.

```
$ make generate
```

작은 정보

*_types.go 파일을 수정한 후에는 make generate 명령을 실행하여 해당 리소스 유형에 대해 생성된 코드를 업데이트해야 합니다.

3. spec 및 status 필드 및 CRD 검증 마커를 사용하여 API를 정의한 후 CRD 매니페스트를 생성하고 업데이트합니다.

```
$ make manifests
```

이 Makefile 대상은 controller-gen 유틸리티를 호출하여 config/crd/bases/cache.my.domain_memcachedbackups.yaml 파일에서 CRD 매니페스트를 생성합니다.

5.5.6.4.2. 컨트롤러 구현

이 튜토리얼의 컨트롤러는 다음 작업을 수행합니다.

- Memcached 배포가 없는 경우 생성합니다.
- 배포 크기가 Memcached CR 사양에 지정된 것과 같은지 확인합니다.
- Memcached CR 상태를 memcached Pod의 이름으로 업데이트합니다.

위에 언급된 작업을 수행하도록 컨트롤러를 구성하는 방법에 대한 자세한 내용은 표준 Go 기반 Operator에 대한 Operator SDK 튜토리얼의 컨트롤러 구현 을 참조하십시오.

5.5.6.4.3. main.go의 차이점

표준 Go 기반 Operator 및 하이브리드 Helm Operator의 경우 main.go 파일은 Go API에 대한 Manager 프로그램의 초기화 및 실행을 스캐폴딩합니다. 그러나 하이브리드 Helm Operator의 경우 main.go 파일은 watches.yaml 파일을 로드하고 Helm 조정기를 구성하는 논리도 노출합니다.

예 5.4. main.go 파일 예

```
...
 for _, w := range ws {
  // Register controller with the factory
  reconcilePeriod := defaultReconcilePeriod
  if w.ReconcilePeriod != nil {
   reconcilePeriod = w.ReconcilePeriod.Duration
  }

  maxConcurrentReconciles := defaultMaxConcurrentReconciles
  if w.MaxConcurrentReconciles != nil {
   maxConcurrentReconciles = *w.MaxConcurrentReconciles
  }

  r, err := reconciler.New(
   reconciler.WithChart(*w.Chart),
   reconciler.WithGroupVersionKind(w.GroupVersionKind),
   reconciler.WithOverrideValues(w.OverrideValues),
```

```
    reconciler.SkipDependentWatches(w.WatchDependentResources != nil &&
!*w.WatchDependentResources),
    reconciler.WithMaxConcurrentReconciles(maxConcurrentReconciles),
    reconciler.WithReconcilePeriod(reconcilePeriod),
    reconciler.WithInstallAnnotations(annotation.DefaultInstallAnnotations...),
    reconciler.WithUpgradeAnnotations(annotation.DefaultUpgradeAnnotations...),
    reconciler.WithUninstallAnnotations(annotation.DefaultUninstallAnnotations...),
   )
...
```

관리자는 **Helm** 및 **Go** 조정기를 모두 사용하여 초기화됩니다.

예 5.5. Helm 및 **Go** 조정기의 예

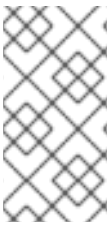
```
...
// Setup manager with Go API
    if err = (&controllers.MemcachedBackupReconciler{
   Client: mgr.GetClient(),
   Scheme: mgr.GetScheme(),
  }).SetupWithManager(mgr); err != nil {
   setupLog.Error(err, "unable to create controller", "controller", "MemcachedBackup")
   os.Exit(1)
  }

    ...
// Setup manager with Helm API
  for _, w := range ws {

      ...
   if err := r.SetupWithManager(mgr); err != nil {
    setupLog.Error(err, "unable to create controller", "controller", "Helm")
    os.Exit(1)
   }
   setupLog.Info("configured watch", "gvk", w.GroupVersionKind, "chartPath", w.ChartPath,
"maxConcurrentReconciles", maxConcurrentReconciles, "reconcilePeriod",
reconcilePeriod)
  }

// Start the manager
    if err := mgr.Start(ctrl.SetupSignalHandler()); err != nil {
   setupLog.Error(err, "problem running manager")
   os.Exit(1)
  }
```

5.5.6.4.4. 권한 및 RBAC 매니페스트

컨트롤러에서 관리하는 리소스와 상호 작용하려면 특정 **RBAC**(역할 기반 액세스 제어) 권한이 필요합니다. **Go API**의 경우 표준 **Go** 기반 **Operator**를 위해 **Operator SDK** 튜토리얼에 표시된 대로 **RBAC** 마

커로 지정됩니다.

Helm API의 경우 **roles.yaml** 에서 권한이 기본적으로 스캐폴드됩니다. 그러나 현재 **Go API**가 스캐폴드되면 알려진 문제로 인해 **Helm API**에 대한 권한을 덮어씁니다. 이 문제로 인해 **roles.yaml** 에 정의된 권한이 사용자의 요구 사항과 일치하는지 확인합니다.



참고

이 알려진 문제는 https://github.com/operator-framework/helm-operator-plugins/issues/142 에서 추적 중입니다.

다음은 **Memcached Operator**의 **role.yaml** 의 예입니다.

예 5.6. Helm 및 Go 조정기의 예

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: manager-role
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
- apiGroups:
  - apps
  resources:
  - deployments
  - daemonsets
  - replicasets
  - statefulsets
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - cache.my.domain
  resources:
  - memcachedbackups
  verbs:
  - create
```

```
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - cache.my.domain
  resources:
  - memcachedbackups/finalizers
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - ""
  resources:
  - pods
  - services
  - services/finalizers
  - endpoints
  - persistentvolumeclaims
  - events
  - configmaps
  - secrets
  - serviceaccounts
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - cache.my.domain
  resources:
  - memcachedbackups/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - policy
  resources:
  - events
  - poddisruptionbudgets
  verbs:
  - create
  - delete
  - get
  - list
```

```
  - patch
  - update
  - watch
- apiGroups:
  - cache.my.domain
  resources:
  - memcacheds
  - memcacheds/status
  - memcacheds/finalizers
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
```

추가 리소스

- Go 기반 Operator의 RBAC 마커

5.5.6.5. 클러스터 외부에서 로컬로 실행

Operator 프로젝트를 클러스터 외부의 Go 프로그램으로 실행할 수 있습니다. 이는 배포 및 테스트 속도를 높이기 위한 개발 목적에 유용합니다.

절차

- 다음 명령을 실행하여 ~/.kube/config 파일에 구성된 클러스터에 CRD(사용자 정의 리소스 정의)를 설치하고 Operator를 로컬로 실행합니다.

```
$ make install run
```

5.5.6.6. 클러스터에서 배포로 실행

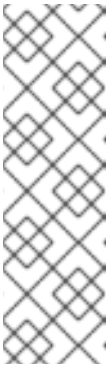
Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

절차

1. 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

a. 이미지를 빌드합니다.

```
$ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
```

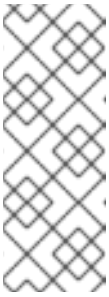


참고

Operator용 SDK에서 생성한 Dockerfile은 Go 빌드에 대해 GOARCH=amd64 를 명시적으로 참조합니다. AMD64 이외의 아키텍처의 경우 GOARCH=$TARGETARCH 로 수정할 수 있습니다. Docker는 자동으로 -platform 에서 지정한 값으로 환경 변수를 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 다중 아키텍처를 참조하십시오.

b. 이미지를 리포지토리로 내보냅니다.

```
$ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
```



참고

두 명령 모두 이미지의 이름과 태그(예: IMG=<registry>/<user>/<image_name>:<tag>)를 Makefile에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 IMG ?= controller:latest 값을 수정합니다.

2. 다음 명령을 실행하여 Operator를 배포합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

기본적으로 이 명령은 <project_name>-system 형식으로 된 Operator 프로젝트 이름을 사용하여 네임스페이스를 생성하고 배포에 사용됩니다. 이 명령은 또한 config/rbac에서 RBAC 매니페스트를 설치합니다.

3. 다음 명령을 실행하여 Operator가 실행 중인지 확인합니다.

```
$ oc get deployment -n <project_name>-system
```

출력 예

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
<project_name>-controller-manager       1/1     1            1           8m
```

5.5.6.7. 사용자 정의 리소스 생성

Operator가 설치되면 Operator가 이제 클러스터에 제공하는 CR(사용자 정의 리소스)을 생성하여 Operator를 테스트할 수 있습니다.

절차

1. Operator가 설치된 네임스페이스로 변경합니다.

```
$ oc project <project_name>-system
```

2. replicaCount 필드를 3 으로 업데이트하여 config/samples/cache_v1_memcached.yaml 파일에서 샘플 Memcached CR 매니페스트를 업데이트합니다.

예 5.7. config/samples/cache_v1_memcached.yaml 파일의 예

```
apiVersion: cache.my.domain/v1
kind: Memcached
metadata:
  name: memcached-sample
spec:
  # Default values copied from <project_dir>/helm-charts/memcached/values.yaml
  affinity: {}
  autoscaling:
    enabled: false
    maxReplicas: 100
    minReplicas: 1
    targetCPUUtilizationPercentage: 80
  fullnameOverride: ""
  image:
    pullPolicy: IfNotPresent
    repository: nginx
    tag: ""
  imagePullSecrets: []
  ingress:
    annotations: {}
    className: ""
    enabled: false
    hosts:
    - host: chart-example.local
```

```
    paths:
    - path: /
      pathType: ImplementationSpecific
  tls: []
nameOverride: ""
nodeSelector: {}
podAnnotations: {}
podSecurityContext: {}
replicaCount: 3
resources: {}
securityContext: {}
service:
  port: 80
  type: ClusterIP
serviceAccount:
  annotations: {}
  create: true
  name: ""
tolerations: []
```

3. *Memcached CR을 생성합니다.*

```
$ oc apply -f config/samples/cache_v1_memcached.yaml
```

4. *Memcached Operator가 샘플 CR에 대한 배포를 올바른 크기로 생성하는지 확인합니다.*

```
$ oc get pods
```

출력 예

```
NAME                                READY   STATUS    RESTARTS   AGE
memcached-sample-6fd7c98d8-7dqdr    1/1     Running   0          18m
memcached-sample-6fd7c98d8-g5k7v    1/1     Running   0          18m
memcached-sample-6fd7c98d8-m7vn7    1/1     Running   0          18m
```

5. *크기를 2 로 업데이트하여 config/samples/cache_v1_memcachedbackup.yaml 파일에서 샘플 MemcachedBackup CR 매니페스트를 업데이트합니다.*

예 5.8. config/samples/cache_v1_memcachedbackup.yaml 파일의 예

```
apiVersion: cache.my.domain/v1
```

```
kind: MemcachedBackup
metadata:
  name: memcachedbackup-sample
spec:
  size: 2
```

6. *MemcachedBackup CR을 생성합니다.*

```
$ oc apply -f config/samples/cache_v1_memcachedbackup.yaml
```

7. *memcachedbackup Pod 수가 CR에 지정된 것과 같은지 확인합니다.*

```
$ oc get pods
```

출력 예

```
NAME                                  READY   STATUS    RESTARTS   AGE
memcachedbackup-sample-8649699989-4bbzg   1/1     Running   0          22m
memcachedbackup-sample-8649699989-mq6mx   1/1     Running   0          22m
```

8. *위의 각 CR에서 사양 을 업데이트한 다음 다시 적용할 수 있습니다. 컨트롤러는 다시 조정하고 Pod의 크기가 각 CR의 사양에 지정된 대로 되도록 합니다.*

9. *이 튜토리얼의 일부로 생성된 리소스를 정리합니다.*

   a. *Memcached 리소스를 삭제합니다.*

   ```
   $ oc delete -f config/samples/cache_v1_memcached.yaml
   ```

   b. *MemcachedBackup 리소스를 삭제합니다.*

   ```
   $ oc delete -f config/samples/cache_v1_memcachedbackup.yaml
   ```

c.

make deploy 명령을 사용하여 Operator를 테스트한 경우 다음 명령을 실행합니다.

```
$ make undeploy
```

5.5.6.8. 프로젝트 레이아웃

하이브리드 Helm Operator 스캐폴딩은 Helm 및 Go API와 호환되도록 사용자 지정할 수 있습니다.

파일/폴더	목적
Dockerfile	컨테이너 엔진에서 **make docker-build** 명령을 사용하여 Operator 이미지를 빌드하는 데 사용하는 지침입니다.
Makefile	프로젝트 작업에 도움이 되는 도우미 대상을 사용하여 파일을 빌드합니다.Build file with helper targets to help you work with your project.
PROJECT	Operator의 메타데이터 정보가 포함된 YAML 파일입니다. 프로젝트의 구성을 나타내며 CLI 및 플러그인에 대한 유용한 정보를 추적하는 데 사용됩니다.
bin/	프로젝트를 로컬로 실행하는 데 사용되는 **관리자와** 프로젝트 구성에 사용되는 **kustomize** 유틸리티와 같은 유용한 바이너리를 포함합니다.
config/	클러스터에서 Operator 프로젝트를 시작하는 모든 Kustomize 매니페스트를 포함한 구성 파일을 포함합니다. Plugins는 이를 사용하여 기능을 제공할 수 있습니다. 예를 들어 Operator SDK에서 Operator 번들을 생성하는 데 도움이 되는 CLI는 이 디렉터리에 스캐폴드된 CRD 및 CR을 찾습니다. **config/crd/** CRD(사용자 정의 리소스 정의)를 포함합니다. **config/default/** 표준 구성으로 컨트롤러를 시작하기 위한 Kustomize 베이스가 포함되어 있습니다. **config/manager/** 클러스터에서 Operator 프로젝트를 Pod로 시작하는 매니페스트를 포함합니다. **config/manifests/** **bundle/** 디렉터리에 OLM 매니페스트를 생성하는 기반이 포함되어 있습니다. **config/prometheus/** 프로젝트에서 **ServiceMonitor** 리소스와 같은 Prometheus에 대한 메트릭을 제공하는 데 필요한 매니페스트를 포함합니다. **config/scorecard/** 스코어 카드 툴을 사용하여 프로젝트를 테스트할 수 있도록 하는 데 필요한 매니페스트를 포함합니다. **config/rbac/** 프로젝트를 실행하는 데 필요한 RBAC 권한을 포함합니다. **config/samples/** 사용자 정의 리소스에 대한 샘플이 포함되어 있습니다.

파일/폴더	목적
api/	Go API 정의를 포함합니다.
controllers/	Go API용 컨트롤러를 포함합니다.
Hack/	프로젝트 파일에 대한 라이센스 헤더를 스캐폴드하는 데 사용되는 파일과 같은 유틸리티 파일이 포함되어 있습니다.
main.go	Operator의 기본 프로그램으로, **apis/** 디렉터리에 모든 CRD(사용자 정의 리소스 정의)를 등록하고 **controllers/** 디렉터리의 모든 컨트롤러를 시작하는 새 관리자를 인스턴스화합니다.
helm-charts/	Helm 플러그인으로 **create api** 명령을 사용하여 지정할 수 있는 Helm 차트가 포함되어 있습니다.
watches.yaml	GVK(그룹/버전/종류) 및 Helm 차트 위치가 포함되어 있습니다. Helm 감시를 구성하는 데 사용됩니다.

5.5.7. 최신 Operator SDK 버전의 하이브리드 Helm 기반 프로젝트 업데이트

OpenShift Container Platform 4.11은 Operator SDK 1.22.2를 지원합니다. 워크스테이션에 1.16.0 CLI가 이미 설치되어 있는 경우 최신 버전을 설치하여 CLI를 1.22.2로 업데이트할 수 있습니다.

그러나 기존 Operator 프로젝트에서 Operator SDK 1.22.2와의 호환성을 유지하려면 1.16.0 이후의 중단된 변경 사항에 대한 업데이트 단계가 필요합니다. 1.16.0을 사용하여 이전에 생성되거나 유지 관리되는 Operator 프로젝트에서 업데이트 단계를 수동으로 수행해야 합니다.

5.5.7.1. Operator SDK 1.22.2의 하이브리드 Helm 기반 Operator 프로젝트 업데이트

다음 절차에서는 1.22.2와의 호환성을 위해 기존 하이브리드 Helm 기반 Operator 프로젝트를 업데이트합니다.

사전 요구 사항

- ***Operator SDK 1.22.2가 설치되어 있어야 합니다.***
- ***Operator SDK 1.16.0을 사용하여 Operator 프로젝트를 생성하거나 유지 관리합니다.***

절차

1. *config/default/manager_auth_proxy_patch.yaml* 파일을 다음과 같이 변경합니다.

```
...
spec:
  template:
    spec:
      containers:
      - name: kube-rbac-proxy
        image: registry.redhat.io/openshift4/ose-kube-rbac-proxy:v4.11 (1)
        args:
        - "--secure-listen-address=0.0.0.0:8443"
        - "--upstream=http://127.0.0.1:8080/"
        - "--logtostderr=true"
        - "--v=0" (2)
...
resources:
  limits:
    cpu: 500m
    memory: 128Mi
  requests:
    cpu: 5m
    memory: 64Mi (3)
```

(1) 태그 버전을 **v4.10** 에서 **v4.11** 로 업데이트합니다.

(2) 디버깅 로그 수준을 **--v=10** 에서 **--v=0** 으로 줄입니다.

(3) 리소스 요청 및 제한을 추가합니다.

2. **Makefile** 을 다음과 같이 변경합니다.

   a. **Makefile** 에 다음 환경 변수를 추가하여 이미지 다이제스트 지원을 활성화합니다.

이전 Makefile

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)
...
```

*새로운 **Makefile***

```
BUNDLE_IMG ?= $(IMAGE_TAG_BASE)-bundle:v$(VERSION)

# BUNDLE_GEN_FLAGS are the flags passed to the operator-sdk generate bundle
command
BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

# USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
# You can enable this value if you would like to use SHA Based Digests
# To enable set flag to true
USE_IMAGE_DIGESTS ?= false
ifeq ($(USE_IMAGE_DIGESTS), true)
 BUNDLE_GEN_FLAGS += --use-image-digests
endif
```

b. *Makefile 을 편집하여 번들 대상을 **BUNDLE_GEN_FLAGS** 환경 변수로 교체합니다.*

*이전 **Makefile***

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --
overwrite --version $(VERSION) $(BUNDLE_METADATA_OPTS)
```

*새로운 **Makefile***

```
$(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS)
```

c.

Makefile 을 편집하여 opm 을 버전 1.23.0으로 업데이트합니다.

```
.PHONY: opm
OPM = ./bin/opm
opm: ## Download opm locally if necessary.
ifeq (,$(wildcard $(OPM)))
ifeq (,$(shell which opm 2>/dev/null))
 @{ \
 set -e ;\
 mkdir -p $(dir $(OPM)) ;\
 OS=$(shell go env GOOS) && ARCH=$(shell go env GOARCH) && \
 curl -sSLo $(OPM) https://github.com/operator-framework/operator-
registry/releases/download/v1.23.0/$${OS}-$${ARCH}-opm ;\ 1
 chmod +x $(OPM) ;\
 }
else
OPM = $(shell which opm)
endif
endif
```

1

v1.19.1 을 v1.23.0 으로 교체합니다.

d.

Kubernetes 1.24를 지원하도록 Makefile 의 ENVTEST_K8S_VERSION 및 controller-gen 필드를 업데이트합니다.

```
...
ENVTEST_K8S_VERSION = 1.24 1
...
sigs.k8s.io/controller-tools/cmd/controller-gen@v0.9.0 2
```

1

버전 1.22 를 1.24 로 업데이트합니다.

2

버전 0.7.0 을 0.9.0 으로 업데이트합니다.

e.

다음 명령을 입력하여 Makefile 에 변경 사항을 적용하고 Operator를 다시 빌드합니다.

```
$ make
```

3. Go .mod 파일을 다음과 같이 변경하여 Go 및 해당 종속 항목을 업데이트합니다.

```
go 1.18 (1)

require (
  github.com/onsi/ginkgo v1.16.5 (2)
  github.com/onsi/gomega v1.18.1 (3)
  k8s.io/api v0.24.0 (4)
  k8s.io/apimachinery v0.24.0 (5)
  k8s.io/client-go v0.24.0 (6)
  sigs.k8s.io/controller-runtime v0.12.1 (7)
)
```

(1) 버전 1.16 을 1.18 로 업데이트합니다.

(2) 버전 v1.16.4 를 v1.16.5 로 업데이트합니다.

(3) 버전 v1.15.0 을 v1.18.1 로 업데이트합니다.

(4) (5) (6) 버전 v0.22.1 을 v0.24.0 으로 업데이트합니다.

(7) v0.10.0 버전을 v0.12.1 로 업데이트합니다.

4. go.mod 파일을 편집하여 Helm Operator 플러그인을 업데이트합니다.

```
github.com/operator-framework/helm-operator-plugins v0.0.11 (1)
```

(1) 버전 v0.0.8 을 v0.0.11 로 업데이트합니다.

5.

*Dockerfile*을 다음과 같이 변경하여 버전 *1.18*로 이동합니다.

이전 dockerfile.go 파일

```
const dockerfileTemplate = `# Build the manager binary
FROM golang:1.17 as builder
```

새로운 dockerfile.go 파일

```
const dockerfileTemplate = `# Build the manager binary
FROM golang:1.18 as builder
```

6. 다음 명령을 입력하여 종속 항목을 다운로드하여 정리합니다.

```
$ go mod tidy
```

5.5.7.2. 추가 리소스

- *번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션*
- *Operator SDK 1.16.0 프로젝트 업그레이드*
- *Operator SDK v1.10.1의 프로젝트 업그레이드*
- *Operator SDK v1.8.0의 프로젝트 업그레이드*

5.6. JAVA 기반 OPERATOR

5.6.1. Java 기반 Operator를 위한 Operator SDK 시작하기

중요

Java 기반 Operator SDK는 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 개발자는 Operator SDK에서 제공하는 툴 및 라이브러리를 사용하여 Java 기반 Operator를 설정 및 실행하는 기본 동작을 설명하기 위해 분산형 키-값 저장소인 Memcached에 대한 Java 기반 Operator 예제를 빌드하고 클러스터에 배포할 수 있습니다.

5.6.1.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Java v11+
- Maven v3.6.3+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치

- *OpenShift CLI 시작하기*

5.6.1.2. Java 기반 Operator 생성 및 배포

Operator SDK를 사용하여 Memcached에 대한 간단한 Java 기반 Operator를 빌드하고 배포할 수 있습니다.

프로세스

1. *프로젝트를 생성합니다.*

   a. *프로젝트 디렉토리를 생성합니다.*

   ```
   $ mkdir memcached-operator
   ```

   b. *프로젝트 디렉터리로 변경합니다.*

   ```
   $ cd memcached-operator
   ```

   c. *quarkus 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.*

   ```
   $ operator-sdk init \
     --plugins=quarkus \
     --domain=example.com \
     --project-name=memcached-operator
   ```

2. *API를 생성합니다.*

   간단한 Memcached API를 생성합니다.

   ```
   $ operator-sdk create api \
     --plugins quarkus \
     --group cache \
     --version v1 \
     --kind Memcached
   ```

3. *Operator 이미지를 빌드하여 내보냅니다.*

   기본 Makefile 대상을 사용하여 Operator를 빌드하고 내보냅니다. 내보낼 수 있는 레지스트리를 사용하는 이미지의 가져오기 사양에 IMG를 설정합니다.

```
$ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

4. *Operator를 실행합니다.*

   a. *CRD를 설치합니다.*

```
$ make install
```

   b. *클러스터에 프로젝트를 배포합니다. 내보낸 이미지에 IMG를 설정합니다.*

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

5. *샘플 CR(사용자 정의 리소스)을 생성합니다.*

   a. *샘플 CR을 생성합니다.*

```
$ oc apply -f config/samples/cache_v1_memcached.yaml \
    -n memcached-operator-system
```

   b. *CR에서 Operator를 조정하는지 확인합니다.*

```
$ oc logs deployment.apps/memcached-operator-controller-manager \
    -c manager \
    -n memcached-operator-system
```

6. *CR 삭제*

   다음 명령을 실행하여 CR을 삭제합니다.

```
$ oc delete -f config/samples/cache_v1_memcached -n memcached-operator-system
```

7. 정리합니다.

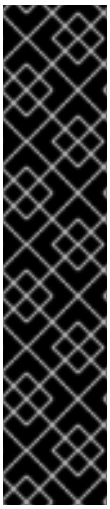
다음 명령을 실행하여 이 절차의 일부로 생성된 리소스를 정리합니다.

```
$ make undeploy
```

5.6.1.3. 다음 단계

- Java 기반 Operator를 빌드하는 방법에 대한 자세한 내용은 Java 기반 Operator를 위한 Operator SDK 튜토리얼을 참조하십시오.

5.6.2. Java 기반 Operator를 위한 Operator SDK 튜토리얼



중요

Java 기반 Operator SDK는 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 개발자는 Operator SDK의 Java 프로그래밍 언어 지원을 활용하여 분산형 키-값 저장소인 Memcached에 대한 Java 기반 Operator 예제를 빌드하고 라이프사이클을 관리할 수 있습니다.

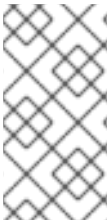
이 프로세스는 Operator 프레임워크의 두 가지 주요 요소를 사용하여 수행됩니다.

Operator SDK

operator-sdk CLI 툴 및 java-operator-sdk 라이브러리 API

OLM(Operator Lifecycle Manager)

클러스터에 대한 Operator의 설치, 업그레이드, RBAC(역할 기반 액세스 제어)



참고

이 튜토리얼에는 Java 기반 Operator용 Operator SDK 시작하기보다 자세히 설명되어 있습니다.

5.6.2.1. 사전 요구 사항

- Operator SDK CLI가 설치됨
- OpenShift CLI(oc) v4.11 이상이 설치됨
- Java v11+
- Maven v3.6.3+
- cluster-admin 권한이 있는 계정으로 oc 를 사용하여 OpenShift Container Platform 4.11 클러스터에 로그인함
- 클러스터에서 이미지를 가져올 수 있도록 하려면 이미지를 내보내는 리포지토리를 공개로 설정하거나 이미지 가져오기 보안을 구성해야 합니다.

추가 리소스

- Operator SDK CLI 설치
- OpenShift CLI 시작하기

5.6.2.2. 프로젝트 생성

Operator SDK CLI를 사용하여 memcached-operator라는 프로젝트를 생성합니다.

프로세스

1. 프로젝트에 사용할 디렉터리를 생성합니다.

```
$ mkdir -p $HOME/projects/memcached-operator
```

2. 디렉터리로 변경합니다.

```
$ cd $HOME/projects/memcached-operator
```

3. quarkus 플러그인과 함께 operator-sdk init 명령을 실행하여 프로젝트를 초기화합니다.

```
$ operator-sdk init \
  --plugins=quarkus \
  --domain=example.com \
  --project-name=memcached-operator
```

5.6.2.2.1. PROJECT 파일

operator-sdk init 명령으로 생성된 파일 중에는 Kubebuilder PROJECT 파일이 있습니다. 이어서 프로젝트 루트에서 실행되는 operator-sdk 명령과 help 출력에서는 이 파일을 읽고 프로젝트 유형이 Java임을 확인합니다. 예를 들면 다음과 같습니다.

```
domain: example.com
layout:
- quarkus.javaoperatorsdk.io/v1-alpha
projectName: memcached-operator
version: "3"
```

5.6.2.3. API 및 컨트롤러 생성

Operator SDK CLI를 사용하여 CRD(사용자 정의 리소스 정의) API 및 컨트롤러를 생성합니다.

프로세스

1. 다음 명령을 실행하여 API를 생성합니다.

```
$ operator-sdk create api \
  --plugins=quarkus \ 1
  --group=cache \ 2
  --version=v1 \ 3
  --kind=Memcached 4
```

1. 플러그인 플래그를 **quarkus** 로 설정합니다.

2. 그룹 플래그를 **cache** 로 설정합니다.

3. **version** 플래그를 **v1** 로 설정합니다.

4. **kind** 플래그를 **Memcached** 로 설정합니다.

검증

1. **tree** 명령을 실행하여 파일 구조를 확인합니다.

```
$ tree
```

출력 예

```
.
├── Makefile
├── PROJECT
├── pom.xml
└── src
    └── main
        ├── java
        │   └── com
        │       └── example
        │           ├── Memcached.java
        │           ├── MemcachedReconciler.java
        │           ├── MemcachedSpec.java
        │           └── MemcachedStatus.java
        └── resources
            └── application.properties

6 directories, 8 files
```

5.6.2.3.1. API 정의

Memcached CR(사용자 정의 리소스)의 API를 정의합니다.

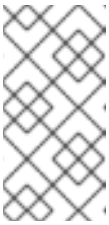
프로세스

- **create api 프로세스의 일부로 생성된 다음 파일을 편집합니다.**

a. **MemcachedSpec.java 파일에서 다음 속성을 업데이트하여 Memcached CR의 원하는 상태를 정의합니다.**

```
public class MemcachedSpec {

   private Integer size;

   public Integer getSize() {
      return size;
   }

   public void setSize(Integer size) {
      this.size = size;
   }
}
```

b. **MemcachedStatus.java 파일에서 다음 속성을 업데이트하여 Memcached CR의 관찰 상태를 정의합니다.**



참고

아래 예제에서는 **Node status** 필드를 보여줍니다. 실제로 일반적인 상태 속성을 사용하는 것이 좋습니다.

```
import java.util.ArrayList;
import java.util.List;

public class MemcachedStatus {

   // Add Status information here
   // Nodes are the names of the memcached pods
   private List<String> nodes;

   public List<String> getNodes() {
      if (nodes == null) {
         nodes = new ArrayList<>();
      }
```

```
        return nodes;
    }

    public void setNodes(List<String> nodes) {
        this.nodes = nodes;
    }
}
```

c. Memcached.java 파일을 업데이트하여 MemcachedSpec.java 및 MemcachedStatus.java 파일로 확장되는 Memcached API의 Schema를 정의합니다.

```
@Version("v1")
@Group("cache.example.com")
public class Memcached extends CustomResource<MemcachedSpec,
MemcachedStatus> implements Namespaced {}
```

5.6.2.3.2. CRD 매니페스트 생성

API가 MemcachedSpec 및 MemcachedStatus 파일을 사용하여 정의한 후 CRD 매니페스트를 생성할 수 있습니다.

프로세스

- memcached-operator 디렉터리에서 다음 명령을 실행하여 CRD를 생성합니다.

```
$ mvn clean install
```

검증

- 다음 예와 같이 target/kubernetes/memcacheds.cache.example.com-v1.yml 파일에서 CRD의 콘텐츠를 확인합니다.

```
$ cat target/kubernetes/memcacheds.cache.example.com-v1.yaml
```

출력 예

```
# Generated by Fabric8 CRDGenerator, manual edits might get overwritten!
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: memcacheds.cache.example.com
spec:
  group: cache.example.com
```

```
  names:
    kind: Memcached
    plural: memcacheds
    singular: memcached
  scope: Namespaced
  versions:
  - name: v1
    schema:
      openAPIV3Schema:
        properties:
          spec:
            properties:
              size:
                type: integer
            type: object
          status:
            properties:
              nodes:
                items:
                  type: string
                type: array
            type: object
        type: object
    served: true
    storage: true
    subresources:
      status: {}
```

5.6.2.3.3. 사용자 정의 리소스 생성

CRD 매니페스트를 생성한 후 CR(사용자 정의 리소스)을 생성할 수 있습니다.

프로세스

- memcached-sample.yaml 이라는 Memcached CR을 생성합니다.

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
spec:
  # Add spec fields here
  size: 1
```

5.6.2.4. 컨트롤러 구현

새 API 및 컨트롤러를 생성하면 컨트롤러 논리를 구현할 수 있습니다.

프로세스

1. *pom.xml 파일에 다음 종속성을 추가합니다.*

```
<dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.2.2</version>
</dependency>
```

2. *이 예제에서는 생성된 컨트롤러 파일 MemcachedReconciler.java 를 다음 예제 구현으로 교체합니다.*

예 5.9. Example MemcachedReconciler.java

```
package com.example;

import io.fabric8.kubernetes.client.KubernetesClient;
import io.javaoperatorsdk.operator.api.reconciler.Context;
import io.javaoperatorsdk.operator.api.reconciler.Reconciler;
import io.javaoperatorsdk.operator.api.reconciler.UpdateControl;
import io.fabric8.kubernetes.api.model.ContainerBuilder;
import io.fabric8.kubernetes.api.model.ContainerPortBuilder;
import io.fabric8.kubernetes.api.model.LabelSelectorBuilder;
import io.fabric8.kubernetes.api.model.ObjectMetaBuilder;
import io.fabric8.kubernetes.api.model.OwnerReferenceBuilder;
import io.fabric8.kubernetes.api.model.Pod;
import io.fabric8.kubernetes.api.model.PodSpecBuilder;
import io.fabric8.kubernetes.api.model.PodTemplateSpecBuilder;
import io.fabric8.kubernetes.api.model.apps.Deployment;
import io.fabric8.kubernetes.api.model.apps.DeploymentBuilder;
import io.fabric8.kubernetes.api.model.apps.DeploymentSpecBuilder;
import org.apache.commons.collections.CollectionUtils;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class MemcachedReconciler implements Reconciler<Memcached> {
  private final KubernetesClient client;

  public MemcachedReconciler(KubernetesClient client) {
    this.client = client;
  }

  // TODO Fill in the rest of the reconciler

  @Override
```

```
  public UpdateControl<Memcached> reconcile(
      Memcached resource, Context context) {
      // TODO: fill in logic
      Deployment deployment = client.apps()
              .deployments()
              .inNamespace(resource.getMetadata().getNamespace())
              .withName(resource.getMetadata().getName())
              .get();

      if (deployment == null) {
          Deployment newDeployment = createMemcachedDeployment(resource);
          client.apps().deployments().create(newDeployment);
          return UpdateControl.noUpdate();
      }

      int currentReplicas = deployment.getSpec().getReplicas();
      int requiredReplicas = resource.getSpec().getSize();

      if (currentReplicas != requiredReplicas) {
          deployment.getSpec().setReplicas(requiredReplicas);
          client.apps().deployments().createOrReplace(deployment);
          return UpdateControl.noUpdate();
      }

      List<Pod> pods = client.pods()
          .inNamespace(resource.getMetadata().getNamespace())
          .withLabels(labelsForMemcached(resource))
          .list()
          .getItems();

      List<String> podNames =
          pods.stream().map(p ->
p.getMetadata().getName()).collect(Collectors.toList());


      if (resource.getStatus() == null
              || !CollectionUtils.isEqualCollection(podNames,
resource.getStatus().getNodes())) {
          if (resource.getStatus() == null) resource.setStatus(new
MemcachedStatus());
          resource.getStatus().setNodes(podNames);
          return UpdateControl.updateResource(resource);
      }

      return UpdateControl.noUpdate();
  }

  private Map<String, String> labelsForMemcached(Memcached m) {
    Map<String, String> labels = new HashMap<>();
    labels.put("app", "memcached");
    labels.put("memcached_cr", m.getMetadata().getName());
    return labels;
  }

  private Deployment createMemcachedDeployment(Memcached m) {
      Deployment deployment = new DeploymentBuilder()
```

```
        .withMetadata(
            new ObjectMetaBuilder()
                .withName(m.getMetadata().getName())
                .withNamespace(m.getMetadata().getNamespace())
                .build())
        .withSpec(
            new DeploymentSpecBuilder()
                .withReplicas(m.getSpec().getSize())
                .withSelector(
                    new
LabelSelectorBuilder().withMatchLabels(labelsForMemcached(m)).build())
                .withTemplate(
                    new PodTemplateSpecBuilder()
                        .withMetadata(
                            new
ObjectMetaBuilder().withLabels(labelsForMemcached(m)).build())
                        .withSpec(
                            new PodSpecBuilder()
                                .withContainers(
                                    new ContainerBuilder()
                                        .withImage("memcached:1.4.36-alpine")
                                        .withName("memcached")
                                        .withCommand("memcached", "-m=64", "-o", "modern", "-
v")
                                        .withPorts(
                                            new ContainerPortBuilder()
                                                .withContainerPort(11211)
                                                .withName("memcached")
                                                .build())
                                        .build())
                                .build())
                        .build())
                .build())
        .build();
    deployment.addOwnerReference(m);
    return deployment;
  }
}
```

예제 컨트롤러는 각 **Memcached CR**(사용자 정의 리소스)에 대해 다음 조정 논리를 실행합니다.

- **Memcached** 배포가 없는 경우 생성합니다.
- 배포 크기가 **Memcached CR** 사양에 지정된 크기와 일치하는지 확인합니다.
- **Memcached CR** 상태를 **memcached Pod**의 이름으로 업데이트합니다.

다음 하위 섹션에서는 구현 예제의 컨트롤러에서 리소스를 조사하는 방법과 조정 반복문을 트리거하는 방법을 설명합니다. 이러한 하위 섹션을 건너뛰어 Operator 실행으로 바로 이동할 수 있습니다.

5.6.2.4.1. 조정 반복문

1. 모든 컨트롤러에는 조정 반복문을 구현하는 Reconcile() 메서드가 포함된 조정기 오브젝트가 있습니다. 조정 반복문은 다음 예와 같이 Deployment 인수를 전달합니다.

```
Deployment deployment = client.apps()
        .deployments()
        .inNamespace(resource.getMetadata().getNamespace())
        .withName(resource.getMetadata().getName())
        .get();
```

2. 다음 예와 같이 Deployment 가 null 인 경우 배포를 생성해야 합니다. 배포를 생성한 후 조정이 필요한지 확인할 수 있습니다. 조정이 필요하지 않은 경우 UpdateControl.noUpdate() 의 값을 반환하고, 그렇지 않으면 'UpdateControl.updateStatus(resource) 값을 반환합니다.

```
if (deployment == null) {
    Deployment newDeployment = createMemcachedDeployment(resource);
    client.apps().deployments().create(newDeployment);
    return UpdateControl.noUpdate();
}
```

3. 배포를 가져온 후 다음 예와 같이 현재 및 필요한 복제본을 가져옵니다.

```
int currentReplicas = deployment.getSpec().getReplicas();
int requiredReplicas = resource.getSpec().getSize();
```

4. currentReplicas 가 requiredReplicas 와 일치하지 않는 경우 다음 예 와 같이 배포를 업데이트해야 합니다.

```
if (currentReplicas != requiredReplicas) {
    deployment.getSpec().setReplicas(requiredReplicas);
    client.apps().deployments().createOrReplace(deployment);
    return UpdateControl.noUpdate();
}
```

5. 다음 예제에서는 Pod 및 해당 이름 목록을 가져오는 방법을 보여줍니다.

```
List<Pod> pods = client.pods()
    .inNamespace(resource.getMetadata().getNamespace())
```

```
            .withLabels(labelsForMemcached(resource))
            .list()
            .getItems();

        List<String> podNames =
            pods.stream().map(p -> p.getMetadata().getName()).collect(Collectors.toList());
```

6. 리소스가 생성되었는지 확인하고 Memcached 리소스를 사용하여 podnames를 확인합니다. 이러한 조건 중 하나에 불일치가 존재하는 경우 다음 예와 같이 조정을 수행합니다.

```
        if (resource.getStatus() == null
                || !CollectionUtils.isEqualCollection(podNames,
    resource.getStatus().getNodes())) {
            if (resource.getStatus() == null) resource.setStatus(new MemcachedStatus());
            resource.getStatus().setNodes(podNames);
            return UpdateControl.updateResource(resource);
        }
```

5.6.2.4.2. labelsForMemcached정의

labelsForMemcached 는 리소스에 연결할 라벨 맵을 반환하는 유틸리티입니다.

```
  private Map<String, String> labelsForMemcached(Memcached m) {
    Map<String, String> labels = new HashMap<>();
    labels.put("app", "memcached");
    labels.put("memcached_cr", m.getMetadata().getName());
    return labels;
  }
```

5.6.2.4.3. createMemcachedDeployment정의

createMemcachedDeployment 메서드에서는 fabric8 DeploymentBuilder 클래스를 사용합니다.

```
  private Deployment createMemcachedDeployment(Memcached m) {
    Deployment deployment = new DeploymentBuilder()
      .withMetadata(
        new ObjectMetaBuilder()
          .withName(m.getMetadata().getName())
          .withNamespace(m.getMetadata().getNamespace())
          .build())
      .withSpec(
        new DeploymentSpecBuilder()
          .withReplicas(m.getSpec().getSize())
          .withSelector(
            new LabelSelectorBuilder().withMatchLabels(labelsForMemcached(m)).build())
          .withTemplate(
            new PodTemplateSpecBuilder()
              .withMetadata(
```

```
                new ObjectMetaBuilder().withLabels(labelsForMemcached(m)).build())
            .withSpec(
                new PodSpecBuilder()
                    .withContainers(
                        new ContainerBuilder()
                            .withImage("memcached:1.4.36-alpine")
                            .withName("memcached")
                            .withCommand("memcached", "-m=64", "-o", "modern", "-v")
                            .withPorts(
                                new ContainerPortBuilder()
                                    .withContainerPort(11211)
                                    .withName("memcached")
                                    .build())
                            .build())
                    .build())
            .build())
        .build();
    deployment.addOwnerReference(m);
    return deployment;
  }
```

5.6.2.5. Operator 실행

다음 세 가지 방법으로 **Operator SDK CLI**를 사용하여 **Operator**를 빌드하고 실행할 수 있습니다.

- **Go** 프로그램으로 클러스터 외부에서 로컬로 실행합니다.
- 클러스터에서 배포로 실행합니다.
- **Operator**를 번들로 제공하고 **OLM(Operator Lifecycle Manager)**을 사용하여 클러스터에 배포합니다.

5.6.2.5.1. 클러스터 외부에서 로컬로 실행

Operator 프로젝트를 클러스터 외부의 **Go** 프로그램으로 실행할 수 있습니다. 이는 배포 및 테스트 속도를 높이기 위한 개발 목적에 유용합니다.

프로세스

1. 다음 명령을 실행하여 **Operator**를 컴파일합니다.

```
$ mvn clean install
```

출력 예

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.193 s
[INFO] Finished at: 2021-05-26T12:16:54-04:00
[INFO] ------------------------------------------------------------------------
```

2. *다음 명령을 실행하여 **CRD**를 기본 네임스페이스에 설치합니다.*

```
$ oc apply -f target/kubernetes/memcacheds.cache.example.com-v1.yml
```

출력 예

```
customresourcedefinition.apiextensions.k8s.io/memcacheds.cache.example.com
created
```

3. *다음 예와 같이 **rbac.yaml** 이라는 파일을 생성합니다.*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: memcached-operator-admin
subjects:
- kind: ServiceAccount
  name: memcached-quarkus-operator-operator
  namespace: default
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: ""
```

4. *다음 명령을 실행하여 **rbac.yaml** 파일을 적용하여 **memcached-quarkus-operator-operator** 에 **cluster-admin** 권한을 부여합니다.*

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

5. 다음 명령을 입력하여 **Operator**를 실행합니다.

```
$ java -jar target/quarkus-app/quarkus-run.jar
```

참고

java 명령은 프로세스를 종료할 때까지 **Operator**를 실행하고 계속 실행됩니다. 이러한 명령의 나머지 부분을 완료하려면 다른 터미널이 필요합니다.

6. 다음 명령을 사용하여 **memcached-sample.yaml** 파일을 적용합니다.

```
$ kubectl apply -f memcached-sample.yaml
```

출력 예

```
memcached.cache.example.com/memcached-sample created
```

검증

- 다음 명령을 실행하여 **Pod**가 시작되었는지 확인합니다.

```
$ oc get all
```

출력 예

```
NAME                                     READY   STATUS    RESTARTS   AGE
pod/memcached-sample-6c765df685-mfqnz    1/1     Running   0          18s
```

5.6.2.5.2. 클러스터에서 배포로 실행

Operator 프로젝트를 클러스터에서 배포로 실행할 수 있습니다.

프로세스

1. 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

   a. 이미지를 빌드합니다.

   ```
   $ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. 이미지를 리포지토리로 내보냅니다.

   ```
   $ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```

   참고

   두 명령 모두 이미지의 이름과 태그(예: IMG=<registry>/<user>/<image_name>:<tag>)를 Makefile에 설정할 수 있습니다. 기본 이미지 이름을 설정하려면 IMG ?= controller:latest 값을 수정합니다.

2. 다음 명령을 실행하여 CRD를 기본 네임스페이스에 설치합니다.

   ```
   $ oc apply -f target/kubernetes/memcacheds.cache.example.com-v1.yml
   ```

   출력 예

```
customresourcedefinition.apiextensions.k8s.io/memcacheds.cache.example.com
created
```

3. 다음 예와 같이 **rbac.yaml** 이라는 파일을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: memcached-operator-admin
subjects:
- kind: ServiceAccount
  name: memcached-quarkus-operator-operator
  namespace: default
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: ""
```

중요

rbac.yaml 파일은 이후 단계에서 적용됩니다.

4. 다음 명령을 실행하여 **Operator**를 배포합니다.

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

5. 다음 명령을 실행하여 이전 단계에서 생성한 **rbac.yaml** 파일을 적용하여 **memcached-quarkus-operator-operator -operator**에 **cluster-admin** 권한을 부여합니다.

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

6. 다음 명령을 실행하여 **Operator**가 실행 중인지 확인합니다.

```
$ oc get all -n default
```

출력 예

```
NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
pod/memcached-quarkus-operator-operator-7db86ccf58-k4mlm   0/1     Running   0
18s
```

7. 다음 명령을 실행하여 **memcached-sample.yaml** 을 적용하고 **memcached-sample Pod** 를 생성합니다.

```
$ oc apply -f memcached-sample.yaml
```

출력 예

```
memcached.cache.example.com/memcached-sample created
```

검증

- 다음 명령을 실행하여 **Pod**가 시작되었는지 확인합니다.

```
$ oc get all
```

출력 예

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/memcached-quarkus-operator-operator-7b766f4896-kxnzt   1/1     Running   1
79s
pod/memcached-sample-6c765df685-mfqnz                       1/1     Running   0          18s
```

5.6.2.5.3. Operator 번들링 및 Operator Lifecycle Manager를 통한 배포

5.6.2.5.3.1. Operator 번들

Operator 번들 형식은 Operator SDK 및 Operator Lifecycle Manager (OLM)의 기본 패키지 메서드입니다. Operator SDK를 사용하여 Operator 프로젝트를 번들 이미지로 빌드하고 푸시하여 OLM에서 Operator를 사용할 수 있습니다.

사전 요구 사항

- *개발 워크스테이션에 Operator SDK CLI가 설치됨*
- *OpenShift CLI(oc) v4.11 이상이 설치됨*
- *Operator SDK를 사용하여 Operator 프로젝트를 초기화함*

프로세스

1. *Operator 프로젝트 디렉터리에서 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.*

   a. *이미지를 빌드합니다.*

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. *이미지를 리포지토리로 내보냅니다.*

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

2. *Operator SDK generate bundle 및 bundle validate 명령을 비롯한 다양한 명령을 호출하는 make bundle 명령을 실행하여 Operator 번들 매니페스트를 생성합니다.*

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Operator의 번들 매니페스트는 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. make bundle 명령은 Operator 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

- ClusterServiceVersion 오브젝트를 포함하는 bundle/manifests라는 번들 매니페스트 디렉터리
- bundle/metadata라는 번들 메타데이터 디렉터리
- config/crd 디렉터리의 모든 CRD(사용자 정의 리소스 정의)
- Dockerfile bundle.Dockerfile

그런 다음 operator-sdk bundle validate를 사용하여 이러한 파일을 자동으로 검증하고 디스크상의 번들 표현이 올바른지 확인합니다.

3. 다음 명령을 실행하여 번들 이미지를 빌드하고 내보냅니다. OLM에서는 하나 이상의 번들 이미지를 참조하는 인덱스 이미지를 통해 Operator 번들을 사용합니다.

   a. 번들 이미지를 빌드합니다. 이미지를 내보낼 레지스트리, 사용자 네임스페이스, 이미지 태그에 대한 세부 정보를 사용하여 BUNDLE_IMG를 설정합니다.

   ```
   $ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:
   <tag>
   ```

   b. 번들 이미지를 내보냅니다.

   ```
   $ docker push <registry>/<user>/<bundle_image_name>:<tag>
   ```

5.6.2.5.3.2. Operator Lifecycle Manager를 사용하여 Operator 배포

OLM(Operator Lifecycle Manager)은 Kubernetes 클러스터에서 Operator 및 관련 서비스를 설치, 업데이트하고 라이프사이클을 관리하는 데 도움이 됩니다. OLM은 기본적으로 OpenShift Container

Platform에 설치되고 Kubernetes 확장으로 실행되므로 추가 툴 없이 모든 Operator 라이프사이클 관리 기능에 웹 콘솔과 OpenShift CLI(oc)를 사용할 수 있습니다.

Operator 번들 형식은 Operator SDK 및 OLM의 기본 패키지 메서드입니다. Operator SDK를 사용하여 OLM에서 번들 이미지를 신속하게 실행하여 올바르게 실행되는지 확인할 수 있습니다.

사전 요구 사항

- *개발 워크스테이션에 Operator SDK CLI가 설치됨*
- *Operator 번들 이미지를 빌드하여 레지스트리로 내보냄*
- *Kubernetes 기반 클러스터에 OLM이 설치되어 있음(apiextensions.k8s.io/v1 CRD(예: OpenShift Container Platform 4.11)를 사용하는 경우 v1.16.0 이상)*
- *cluster-admin 권한이 있는 계정을 사용하여 oc로 클러스터에 로그인됨*

절차

1. *다음 명령을 입력하여 클러스터에서 Operator를 실행합니다.*

```
$ operator-sdk run bundle \ 1
  -n <namespace> \ 2
  <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. *run bundle 명령은 유효한 파일 기반 카탈로그를 생성하고 OLM을 사용하여 클러스터에 Operator 번들을 설치합니다.*

2. *선택 사항: 기본적으로 이 명령은 현재 활성 프로젝트에 ~/.kube/config 파일에 Operator를 설치합니다. -n 플래그를 추가하면 설치에 다른 네임스페이스 범위를 설정할 수 있습니다.*

3. *이미지를 지정하지 않으면 명령에서 quay.io/operator-framework/opm:latest 를 기본 인덱스 이미지로 사용합니다. 이미지를 지정하면 명령에서 번들 이미지 자체를 인덱스*

이미지로 사용합니다.

중요

OpenShift Container Platform 4.11부터 run bundle 명령은 기본적으로 Operator 카탈로그의 파일 기반 카탈로그 형식을 지원합니다. Operator 카탈로그의 더 이상 사용되지 않는 SQLite 데이터베이스 형식은 계속 지원되지만 향후 릴리스에서 제거됩니다. Operator 작성자는 해당 워크플로우를 파일 기반 카탈로그 형식으로 마이그레이션하는 것이 좋습니다.

이 명령은 다음 작업을 수행합니다.

- 번들 이미지를 참조하는 인덱스 이미지를 생성합니다. 인덱스 이미지는 불투명하고 일시적이지만 프로덕션에서 카탈로그에 번들을 추가하는 방법을 정확하게 반영합니다.
- OperatorHub에서 Operator를 검색할 수 있도록 새 인덱스 이미지를 가리키는 카탈로그 소스를 생성합니다.
- OperatorGroup,Subscription,InstallPlan 및 RBAC를 포함한 기타 모든 필수 리소스를 생성하여 Operator를 클러스터에 배포합니다.

5.6.2.6. 추가 리소스

- Operator SDK에서 생성한 디렉터리 구조에 대한 자세한 내용은 Java 기반 Operator의 프로젝트 레이아웃 을 참조하십시오.
- 클러스터 전체 송신 프록시가 구성된 경우 클러스터 관리자는 프록시 설정을 재정의하거나 OLM(Operator Lifecycle Manager)에서 실행되는 특정 Operator에 대한 사용자 정의 CA 인증서를 삽입 할 수 있습니다.

5.6.3. Java 기반 Operator의 프로젝트 레이아웃

중요

***Java 기반 Operator SDK**는 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-sdk CLI**에서는 각 **Operator** 프로젝트에 대해 다양한 패키지 및 파일을 생성하거나 스캐폴드를 지정할 수 있습니다.*

5.6.3.1. Java 기반 프로젝트 레이아웃

***operator-sdk init** 명령으로 생성된 **Java 기반 Operator** 프로젝트에는 다음 파일 및 디렉터리가 포함됩니다.*

파일 또는 디렉터리	목적
pom.xml	Operator를 실행하는 데 필요한 종속 항목이 포함된 파일입니다.
<domain>/	API를 나타내는 파일이 들어 있는 디렉터리입니다. 도메인이 **example.com** 인 경우 이 폴더는 **example/** 라고 합니다.
MemcachedReconciler.java	컨트롤러 구현을 정의하는 Java 파일입니다.
MemcachedSpec.java	Memcached CR의 원하는 상태를 정의하는 Java 파일입니다.
MemcachedStatus.java	Memcached CR의 관찰된 상태를 정의하는 Java 파일입니다.
Memcached.java	Memcached API의 스키마를 정의하는 Java 파일입니다.
target/kubernetes/	CRD yaml 파일이 포함된 디렉터리입니다.

5.7. CSV(클러스터 서비스 버전) 정의

ClusterServiceVersion 오브젝트에서 정의하는 CSV(클러스터 서비스 버전)는 클러스터에서 Operator를 실행할 때 OLM(Operator Lifecycle Manager)을 지원하는 Operator 메타데이터에서 생성되는 YAML 매니페스트입니다. 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.

Operator SDK에는 CSV 생성기가 포함되어 YAML 매니페스트 및 Operator 소스 파일에 포함된 정보를 사용하여 사용자 정의한 현재 Operator 프로젝트에 대해 CSV를 생성합니다.

CSV 생성 명령을 사용하면 Operator에서 OLM과 상호 작용하거나 메타데이터를 카탈로그 레지스트리에 게시하는 데 전문적인 OLM 지식을 보유한 Operator 작성자가 필요하지 않습니다. 또한 새로운 Kubernetes 및 OLM 기능이 구현되면서 시간이 지남에 따라 CSV 사양이 변경될 수 있으므로 Operator SDK는 앞으로의 새로운 CSV 기능을 처리할 수 있도록 업데이트 시스템을 쉽게 확장할 수 있습니다.

5.7.1. CSV 생성 작동 방식

CSV(클러스터 서비스 버전)를 포함하는 Operator 번들 매니페스트에서는 OLM(Operator Lifecycle Manager)을 사용하여 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. generate bundle 하위 명령으로 호출되는 Operator SDK의 CSV 생성기는 Operator를 카탈로그에 게시하고 OLM과 함께 배포하기 위한 첫 번째 단계입니다. 하위 명령에는 CSV 매니페스트를 구성하는 데 특정 입력 매니페스트가 필요합니다. 명령을 호출하면 CSV 베이스와 함께 모든 입력을 읽어 CSV를 멱등하게 생성하거나 재생성합니다.

일반적으로 generate bundle 하위 명령에서 사용하는 입력 Kustomize 베이스를 생성하기 위해 generate kustomize manifests 하위 명령을 먼저 실행합니다. 그러나 Operator SDK에서는 다음 하위 명령을 순서대로 실행하는 것을 포함하여 여러 작업을 자동화하는 make bundle 명령을 제공합니다.

1. generate kustomize manifests
2. generate bundle
3. bundle validate

추가 리소스

- 번들 및 CSV 생성을 비롯한 전체 프로시저는 Operator 번들링을 참조하십시오.

5.7.1.1. 생성된 파일 및 리소스

make bundle 명령은 **Operator** 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

- **ClusterServiceVersion(CSV)** 오브젝트를 포함하는 **bundle/manifests**라는 번들 매니페스트 디렉터리
- **bundle/metadata**라는 번들 메타데이터 디렉터리
- **config/crd** 디렉터리의 모든 **CRD(**사용자 정의 리소스 정의**)**
- **Dockerfile bundle.Dockerfile**

일반적으로 **CSV**에는 다음 리소스가 포함됩니다.

역할

네임스페이스 내에서 **Operator** 권한을 정의합니다.

ClusterRole

클러스터 수준 **Operator** 권한을 정의합니다.

Deployment

Pod에서 **Operator**의 **Operand**를 실행하는 방법을 정의합니다.

CRD(CustomResourceDefinition)

Operator에서 조정하는 사용자 정의 리소스를 정의합니다.

사용자 정의 리소스 예제

특정 **CRD**의 사양을 준수하는 리소스의 예입니다.

5.7.1.2. 버전 관리

generate bundle 하위 명령의 **--version** 플래그는 처음으로 번들을 생성하고 기존 번들을 업그레이드할 때 번들에 대한 의미 체계 버전을 제공합니다.

Makefile에서 VERSION 변수를 설정하면 make bundle 명령에 의해 generate bundle 하위 명령이 실행될 때 해당 값을 사용하여 --version 플래그가 자동으로 호출됩니다. CSV 버전은 Operator 버전과 동일하며 Operator 버전을 업그레이드하면 새 CSV가 생성됩니다.

5.7.2. 수동으로 정의한 CSV 필드

대부분의 CSV 필드는 Operator SDK와 관련 없이 생성된 일반 매니페스트를 사용하여 채울 수 없습니다. 이러한 필드는 대부분 Operator 및 다양한 CRD(사용자 정의 리소스 정의)에 대해 사람이 작성한 메타데이터입니다.

Operator 작성자는 CSV(클러스터 서비스 버전) YAML 파일을 직접 수정하여 다음과 같은 필수 필드에 개인화된 데이터를 추가해야 합니다. Operator SDK는 필수 필드에서 데이터 부족이 탐지되는 경우 CSV 생성 중 경고를 표시합니다.

다음 테이블에는 필수 또는 선택적인 수동 정의 CSV 필드가 자세히 설명되어 있습니다.

표 5.7. 필수 항목

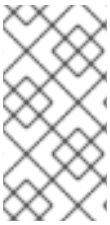
필드	설명
metadata.name	이 CSV의 고유 이름입니다. 고유성을 유지하도록 이름에 Operator 버전이 포함되어야 합니다(예: **app-operator.v0.1.1**).
metadata.capabilities	Operator 완성 모델에 따른 기능 수준입니다. 옵션에는 **Basic Install**, **Seamless Upgrades**, **Full Lifecycle**, **Deep Insights**, **Auto Pilot**이 있습니다.
spec.displayName	Operator를 확인하는 공용 이름입니다.
spec.description	Operator 기능에 대한 간단한 설명입니다.
spec.keywords	Operator를 설명하는 키워드입니다.
spec.maintainers	**name** 및 **email**을 사용하여 Operator를 유지 관리하는 사람 또는 조직 엔티티입니다.
spec.provider	**name**이 있는 Operator의 공급자입니다(일반적으로 조직).
spec.labels	Operator 내부에서 사용할 키-값 쌍입니다.
spec.version	Operator의 의미 체계 버전입니다(예: **0.1.1**).

필드	설명
spec.customresourcedefinitions	Operator에서 사용하는 모든 CRD입니다. CRD YAML 파일이 **deploy/**에 있는 경우 이 필드는 Operator SDK에 의해 자동으로 채워집니다. 그러나 CRD 매니페스트 사양에 없는 몇몇 필드에는 사용자 입력이 필요합니다. • **description**: CRD 설명입니다. • **resources**: CRD에서 활용하는 모든 Kubernetes 리소스입니다(예:**Pod** 및 **StatefulSet** 오브젝트). • **specDescriptors**: Operator의 입력 및 출력에 대한 UI 힌트입니다.

표 5.8. 선택 사항

필드	설명
spec.replaces	CSV 이름이 이 CSV로 교체됩니다.
spec.links	Operator 또는 애플리케이션과 관련된 URL(예: 웹 사이트 및 문서)을 각각 **name** 및 **url**을 사용하여 관리합니다.
spec.selector	Operator에서 클러스터의 리소스와 연결할 수 있는 선택기입니다.
spec.icon	Operator 고유의 base64로 인코딩된 아이콘으로, **mediatype**을 사용하여 **base64data** 필드에 설정됩니다.
spec.maturity	이 버전의 소프트웨어에서 달성한 완성 수준입니다. 옵션에는 **planning**, **pre-alpha**, **alpha**, **beta**, **stable**, **mature**, **inactive**, **deprecated**가 포함됩니다.

위의 각 필드에 보관해야 하는 데이터에 대한 자세한 내용은 CSV 사양에서 확인할 수 있습니다.



참고

현재 사용자 개입이 필요한 여러 YAML 필드를 Operator 코드에서 구문 분석할 수 있습니다.

추가 리소스

- *Operator 완성 모델*

5.7.2.1. Operator 메타데이터 주석

Operator 개발자는 CSV(클러스터 서비스 버전) 메타데이터에 특정 주석을 수동으로 정의하여 OperatorHub와 같은 UI의 기능을 활성화하거나 성능을 강조할 수 있습니다.

다음 테이블에는 metadata.annotations 필드를 사용하여 수동으로 정의할 수 있는 Operator 메타데이터 주석이 나열되어 있습니다.

표 5.9. 주석

필드	Description
alm-examples	CRD(사용자 정의 리소스 정의) 템플릿에 최소 구성 세트를 제공합니다. 사용자가 추가로 사용자 정의할 수 있도록 호환되는 UI가 이 템플릿에 미리 채워집니다.
operatorframework.io/initialization-resource	Operator 설치 중 **operatorframework.io/initialization-resource** 주석을 CSV(클러스터 서비스 버전)에 추가하여 필요한 단일 사용자 정의 리소스를 지정합니다. 그러면 사용자에게 CSV에 제공된 템플릿을 통해 사용자 정의 리소스를 생성하라는 메시지가 표시됩니다. 전체 YAML 정의를 포함하는 템플릿을 포함해야 합니다.
operatorframework.io/suggested-namespace	Operator를 배포해야 하는 제안된 네임스페이스를 설정합니다.

필드	Description
operators.openshift.io/infrastructure-features	Operator에서 지원하는 인프라 기능입니다. 사용자는 웹 콘솔에서 OperatorHub를 통해 Operator를 검색할 때 이러한 기능을 확인하고 필터링할 수 있습니다. 유효한 대소문자를 구분하는 값은 다음과 같습니다. ● **disconnected**: Operator는 모든 종속성을 포함하여 연결이 끊긴 카탈로그로 미러링되며 인터넷 액세스가 필요하지 않습니다. Operator에서 미러링에 필요한 모든 관련 이미지를 나열합니다. ● **CNF**: Operator는 CNF(클라우드 네이티브 네트워크 기능) Kubernetes 플러그인을 제공합니다. ● **CNI**: Operator는 CNI(Container Network Interface) Kubernetes 플러그인을 제공합니다. ● **CSI**: Operator는 CSI(Container Storage Interface) Kubernetes 플러그인을 제공합니다. ● **FIPS**: Operator는 기본 플랫폼의 FIPS 모드를 허용하고 FIPS 모드로 부팅되는 노드에서 작동합니다.  **중요** FIPS 검증 또는 모듈 In Process 암호화 라이브러리 사용은 **x86_64** 아키텍처의 OpenShift Container Platform 배포에서만 지원됩니다. ● **proxy-aware**: Operator는 프록시 뒤의 클러스터에서 실행을 지원합니다. Operator는 클러스터가 프록시를 사용하도록 구성된 경우 OLM(Operator Lifecycle Manager)에서 Operator에 자동으로 제공하는 표준 프록시 환경 변수 **HTTP_PROXY** 및 **HTTPS_PROXY**를 허용합니다. 필수 환경 변수는 관리되는 워크로드의 Operand로 전달됩니다.
operators.openshift.io/valid-subscription	Operator를 사용하는 데 필요한 특정 서브스크립션을 나열하기 위한 자유 양식 어레이입니다. 예를 들면 **'["3Scale Commercial License", "Red Hat Managed Integration"]'**입니다.
operators.operatorframework.io/internal-objects	UI에서 사용자 조작용이 아닌 CRD를 숨깁니다.

사용 사례 예

Operator에서 연결이 끊긴 및 프록시 인식 지원

```
operators.openshift.io/infrastructure-features: '["disconnected", "proxy-aware"]'
```

Operator에는 OpenShift Container Platform 라이센스가 필요합니다.

```
operators.openshift.io/valid-subscription: '["OpenShift Container Platform"]'
```

Operator에는 3scale 라이센스가 필요합니다.

```
operators.openshift.io/valid-subscription: '["3Scale Commercial License", "Red Hat Managed Integration"]'
```

Operator는 연결이 끊긴 프록시 인식을 지원하며 OpenShift Container Platform 라이센스가 필요합니다.

```
operators.openshift.io/infrastructure-features: '["disconnected", "proxy-aware"]'
operators.openshift.io/valid-subscription: '["OpenShift Container Platform"]'
```

추가 리소스

- *CRD 템플릿*

- 필수 사용자 정의 리소스 초기화
- 제안된 네임스페이스 설정
- 제한된 네트워크 환경에 대해 Operator 활성화 (연결이 끊긴 모드)
- 내부 오브젝트 숨기기
- FIPS 암호화 지원

5.7.3. 제한된 네트워크 환경에 대한 Operator 활성화

사용 중인 Operator는 Operator 작성자로서 제한된 네트워크 또는 연결이 끊긴 환경에서 제대로 실행하려면 추가 요구 사항을 충족해야 합니다.

연결이 끊긴 모드를 지원하는 Operator 요구 사항

- 하드 코딩된 이미지 참조를 환경 변수로 교체합니다.
- Operator의 CSV(클러스터 서비스 버전)에서 다음을 수행합니다.
  - Operator에서 기능을 수행하는 데 필요할 수 있는 관련 이미지 또는 기타 컨테이너 이미지를 나열합니다.
  - 태그가 아닌 다이제스트(SHA)를 통해 지정된 모든 이미지를 참조합니다.
- Operator의 모든 종속 항목은 연결이 끊긴 모드에서 실행할 수 있어야 합니다.
- Operator에 클러스터 외부 리소스가 필요하지 않아야 합니다.

사전 요구 사항

- Operator 프로젝트에 CSV가 포함되어 있습니다. 다음 절차에서는 Go-, Ansible- 및 Helm 기반 프로젝트의 예로 Memcached Operator를 사용합니다.

프로세스

1. config/manager/manager.yaml 파일에서 Operator에서 사용하는 추가 이미지 참조에 대한 환경 변수를 설정합니다.

예 5.10. config/manager/manager.yaml 파일 예

```
...
spec:
 ...
   spec:
     ...
     containers:
     - command:
       - /manager
       ...
       env:
       - name: <related_image_environment_variable> 1
         value: "<related_image_reference_with_tag>" 2
```

1 RELATED_IMAGE_MEMCACHED 와 같은 환경 변수를 정의합니다.

2 관련 이미지 참조 및 태그를 설정합니다(예: docker.io/memcached:1.4.36-alpine).

2. 하드 코딩된 이미지 참조를 Operator 프로젝트 유형의 관련 파일의 환경 변수로 교체합니다.

- Go 기반 Operator 프로젝트의 경우 다음 예와 같이 controllers/memcached_controller.go 파일에 환경 변수를 추가합니다.

예 5.11. controllers/memcached_controller.go 파일의 예

```
// deploymentForMemcached returns a memcached Deployment object

...

Spec: corev1.PodSpec{
```

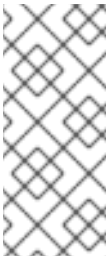
```
        Containers: []corev1.Container{{
-   Image:   "memcached:1.4.36-alpine",
+   Image:   os.Getenv("<related_image_environment_variable>"),
    Name:    "memcached",
    Command: []string{"memcached", "-m=64", "-o", "modern", "-v"},
    Ports: []corev1.ContainerPort{{

...
```

1

이미지 참조 및 태그를 삭제합니다.

2

os.Getenv 함수를 사용하여 < related_image_environment_variable>을 호출합니다.



참고

변수가 설정되지 않은 경우 os.Getenv 함수는 빈 문자열을 반환합니다. 파일을 변경하기 전에 <related_image_environment_variable >을 설정합니다.

- Ansible 기반 Operator 프로젝트의 경우 다음 예와 같이 roles/memcached/tasks/main.yml 파일에 환경 변수를 추가합니다.

예 5.12. roles/memcached/tasks/main.yml 파일의 예

```
spec:
  containers:
  - name: memcached
    command:
    - memcached
    - -m=64
    - -o
    - modern
    - -v
-   image: "docker.io/memcached:1.4.36-alpine"
+   image: "{{ lookup('env', '<related_image_environment_variable>') }}"
    ports:
      - containerPort: 11211

...
```

1

이미지 참조 및 태그를 삭제합니다.

2

lookup 함수를 사용하여 < related_image_environment_variable>.

- Helm 기반 Operator 프로젝트의 경우 다음 예와 같이 watches.yaml 파일에 overrideValues 필드를 추가합니다.

예 5.13. watches.yaml 파일의 예

```
...
- group: demo.example.com
  version: v1alpha1
  kind: Memcached
  chart: helm-charts/memcached
  overrideValues: 1
    relatedImage: ${<related_image_environment_variable>} 2
```

1

overrideValues 필드를 추가합니다.

2

RELATED_IMAGE_MEMCACHED 와 같이 < related_image_environment_variable >을 사용하여 overrideValues 필드를 정의합니다.

a. 다음 예와 같이 helm-charts/memchached/values.yaml 파일에 overrideValues 필드 값을 추가합니다.

helm-charts/memchached/values.yaml 파일의 예

```
...
relatedImage: ""
```

b.

다음 예와 같이 helm-charts/memcached/templates/deployment.yaml 파일에서 차트 템플릿을 편집합니다.

예 5.14. helm-charts/memcached/templates/deployment.yaml 파일의 예

```
containers:
 - name: {{ .Chart.Name }}
   securityContext:
    - toYaml {{ .Values.securityContext | nindent 12 }}
   image: "{{ .Values.image.pullPolicy }}
   env: (1)
    - name: related_image (2)
      value: "{{ .Values.relatedImage }}" (3)
```

(1) *env 필드를 추가합니다.*

(2) *환경 변수 이름을 지정합니다.*

(3) *환경 변수의 값을 정의합니다.*

3.

다음과 같은 변경 사항이 있는 Makefile 에 BUNDLE_GEN_FLAGS 변수 정의를 추가합니다.

Makefile예

```
  BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

  # USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
  # You can enable this value if you would like to use SHA Based Digests
  # To enable set flag to true
  USE_IMAGE_DIGESTS ?= false
  ifeq ($(USE_IMAGE_DIGESTS), true)
      BUNDLE_GEN_FLAGS += --use-image-digests
  endif

...

-  $(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --overwrite -
```

```
-version $(VERSION) $(BUNDLE_METADATA_OPTS) (1)
+  $(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS) (2)

...
```

(1) Makefile 에서 이 행을 삭제합니다.

(2) 위의 행을 이 행으로 바꿉니다.

4. 태그가 아닌 다이제스트(SHA)를 사용하도록 Operator 이미지를 업데이트하려면 make bundle 명령을 실행하고 USE_IMAGE_DIGESTS 를 true 로 설정합니다.

```
$ make bundle USE_IMAGE_DIGESTS=true
```

5. 연결이 끊긴 환경에서도 Operator가 작동함을 나타내는 Disconnected 주석을 추가합니다.

```
metadata:
  annotations:
    operators.openshift.io/infrastructure-features: '["disconnected"]'
```

이 인프라 기능을 통해 OperatorHub에서 Operator를 필터링할 수 있습니다.

5.7.4. 여러 아키텍처 및 운영 체제의 Operator 활성화

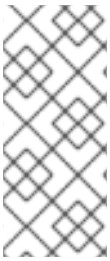
OLM(Operator Lifecycle Manager)은 모든 Operator가 Linux 호스트에서 실행되는 것으로 가정합니다. 그러나 Operator 작성자는 OpenShift Container Platform 클러스터에서 작업자 노드를 사용할 수 있는 경우 Operator가 다른 아키텍처에서 워크로드 관리를 지원하는지를 지정할 수 있습니다.

Operator가 AMD64 및 Linux 이외의 변형을 지원하는 경우 Operator에 지원되는 변형 목록을 제공하는 라벨을 CSV(클러스터 서비스 버전)에 추가할 수 있습니다. 지원되는 아키텍처 및 운영 체제를 나타내는 라벨은 다음과 같이 정의합니다.

```
labels:
  operatorframework.io/arch.<arch>: supported (1)
  operatorframework.io/os.<os>: supported (2)
```

(1) *<arch>를 지원되는 문자열로 설정합니다.*

(2) *<os>를 지원되는 문자열로 설정합니다.*



참고

페키지 매니페스트를 라벨별로 필터링하는 경우 기본 채널의 채널 헤드에 있는 라벨만 사용됩니다. 예를 들어 기본이 아닌 채널에서 Operator에 추가 아키텍처를 제공할 수는 있지만 이러한 아키텍처는 PackageManifest API에서 필터링하는 데는 사용할 수 없습니다.

CSV에 os 라벨이 포함되지 않은 경우 기본적으로 다음 Linux 지원 라벨이 있는 것처럼 처리됩니다.

```
labels:
  operatorframework.io/os.linux: supported
```

CSV에 arch 라벨이 포함되지 않은 경우 기본적으로 다음 AMD64 지원 라벨이 있는 것처럼 처리됩니다.

```
labels:
  operatorframework.io/arch.amd64: supported
```

Operator에서 여러 개의 노드 아키텍처 또는 운영 체제를 지원하는 경우 라벨을 여러 개 추가할 수 있습니다.

사전 요구 사항

- *Operator 프로젝트에 CSV가 포함되어 있습니다.*
- *여러 아키텍처 및 운영 체제를 나열하기 위해서는 CSV에서 참조하는 Operator 이미지가 매니페스트 목록 이미지여야 합니다.*

- *Operator가 제한된 네트워크 또는 연결이 끊긴 환경에서 제대로 작동하려면 참조하는 이미지도 태그가 아닌 다이제스트(SHA)를 사용하여 지정해야 합니다.*

프로세스

- *Operator에서 지원하는 각 지원 아키텍처 및 운영 체제에 대해 CSV의 metadata.labels에 라벨을 추가합니다.*

```
labels:
  operatorframework.io/arch.s390x: supported
  operatorframework.io/os.zos: supported
  operatorframework.io/os.linux: supported 1
  operatorframework.io/arch.amd64: supported 2
```

1 2 *새 아키텍처 또는 운영 체제를 추가한 후에는 기본 os.linux 및 arch.amd64 변형도 명시적으로 포함해야 합니다.*

추가 리소스

- *매니페스트 목록에 대한 자세한 내용은 Image Manifest V 2, Schema 2 사양을 참조하십시오.*

5.7.4.1. Operator에 대한 아키텍처 및 운영 체제 지원

여러 아키텍처 및 운영 체제를 지원하는 Operator에 라벨을 지정하거나 해당 Operator를 필터링할 때는 OpenShift Container Platform의 OLM(Operator Lifecycle Manager)에서 다음 문자열이 지원됩니다.

표 5.10. OpenShift Container Platform에서 지원되는 아키텍처

아키텍처	문자열
AMD64	**amd64**
64비트 PowerPC little-endian	**ppc64le**
IBM Z	**s390x**

표 5.11. OpenShift Container Platform에서 지원되는 운영 체제

운영 체제	문자열
Linux	**linux**
z/OS	**zos**

참고

*다른 버전의 **OpenShift Container Platform** 및 기타 **Kubernetes** 기반 배포에서는 다른 아키텍처 및 운영 체제를 지원할 수 있습니다.*

5.7.5. 제안된 네임스페이스 설정

*일부 **Operator**는 특정 네임스페이스에 배포하거나 특정 네임스페이스에 보조 리소스가 있어야 제대로 작동합니다. 서브스크립션에서 확인된 경우 **OLM(Operator Lifecycle Manager)**은 기본적으로 **Operator**의 네임스페이스 리소스를 해당 서브스크립션의 네임스페이스로 설정합니다.*

*Operator 작성자는 원하는 대상 네임스페이스를 **CSV(클러스터 서비스 버전)**의 일부로 표시하여 **Operator**용으로 설치된 리소스의 최종 네임스페이스를 제어할 수 있습니다. **OperatorHub**를 사용하여 클러스터에 **Operator**를 추가하면 설치 프로세스 중 웹 콘솔에서 클러스터 관리자에게 제안된 네임스페이스를 자동으로 채울 수 있습니다.*

프로세스

- *CSV에서 **operatorframework.io/suggested-namespace** 주석을 제안된 네임스페이스로 설정합니다.*

```
metadata:
  annotations:
    operatorframework.io/suggested-namespace: <namespace> (1)
```

(1) *제안된 네임스페이스를 설정합니다.*

5.7.6. Operator 조건 활성화

OLM(Operator Lifecycle Manager)에서는 Operator에 Operator를 관리하는 동안 OLM 동작에 영향을 미치는 복잡한 상태를 보고하는 채널을 제공합니다. 기본적으로 OLM은 Operator를 설치할 때

OperatorCondition CRD(사용자 정의 리소스 정의)를 생성합니다. OperatorCondition CR(사용자 정의)에 설정된 조건에 따라 OLM의 동작이 적절하게 변경됩니다.

Operator 조건을 지원하려면 Operator가 OLM에서 생성한 OperatorCondition CR을 읽고 다음 작업을 완료할 수 있어야 합니다.

- 특정 조건을 가져옵니다.
- 특정 조건의 상태를 설정합니다.

이 작업은 operator-lib 라이브러리를 사용하여 수행할 수 있습니다. Operator 작성자는 라이브러리에서 클러스터의 Operator 보유 OperatorCondition CR에 액세스할 수 있도록 Operator에 controller-runtime 클라이언트를 제공할 수 있습니다.

라이브러리에서는 일반 Conditions 인터페이스를 제공합니다. 이 인터페이스는 OperatorCondition CR에서 다음과 같은 방법으로 conditionType을 Get 및 Set합니다.

Get

라이브러리는 특정 조건을 가져오기 위해 controller-runtime의 client.Get 함수를 사용합니다. 이 함수에는 conditionAccessor에 있는 types.NamespacedName 유형의 ObjectKey가 필요합니다.

Set

특정 조건의 상태를 업데이트하기 위해 라이브러리는 controller-runtime의 client.Update 함수를 사용합니다. conditionType이 CRD에 없으면 오류가 발생합니다.

Operator는 CR의 status 하위 리소스만 수정할 수 있습니다. Operator는 조건을 포함하도록 status.conditions 어레이를 삭제하거나 업데이트할 수 있습니다. 조건에 있는 필드의 형식 및 설명에 대한 자세한 내용은 업스트림 조건 GoDocs를 참조하십시오.

참고

Operator SDK v1.10.1은 operator-lib v0.3.0을 지원합니다.

사전 요구 사항

- Operator SDK를 사용하여 생성한 Operator 프로젝트입니다.

프로세스

Operator 프로젝트에서 Operator 조건을 활성화하려면 다음을 수행합니다.

1. Operator 프로젝트의 go.mod 파일에 operator-framework/operator-lib을 필수 라이브러리로 추가합니다.

```
module github.com/example-inc/memcached-operator

go 1.15

require (
  k8s.io/apimachinery v0.19.2
  k8s.io/client-go v0.19.2
  sigs.k8s.io/controller-runtime v0.7.0
  operator-framework/operator-lib v0.3.0
)
```

2. Operator 논리에 다음과 같은 결과가 발생하는 자체 생성자를 작성합니다.

   - controller-runtime 클라이언트를 허용합니다.
   - conditionType을 허용합니다.
   - Condition 인터페이스를 반환하여 조건을 업데이트하거나 추가합니다.

   OLM은 현재 Upgradeable 조건을 지원하므로 Upgradeable 조건에 액세스할 수 있는 인터페이스를 생성할 수 있습니다. 예를 들면 다음과 같습니다.

```
import (
  ...
  apiv1 "github.com/operator-framework/api/pkg/operators/v1"
)

func NewUpgradeable(cl client.Client) (Condition, error) {
  return NewCondition(cl, "apiv1.OperatorUpgradeable")
```

```
}

cond, err := NewUpgradeable(cl);
```

이 예제에서는 유형 Condition의 변수 cond를 생성하는 데 NewUpgradeable 생성자가 추가로 사용됩니다. 결국 cond 변수에는 OLM Upgradeable 조건을 처리하는 데 사용할 수 있는 Get 및 Set 방법이 포함됩니다.

추가 리소스

- Operator 상태

5.7.7. Webhook 정의

Operator 작성자는 Webhook를 통해 리소스를 오브젝트 저장소에 저장하고 Operator 컨트롤러에서 이를 처리하기 전에 리소스를 가로채기, 수정, 수락 또는 거부할 수 있습니다. Operator와 함께 webhook가 제공 될 때 OLM (Operator Lifecycle Manager)은 이러한 Webhook의 라이프 사이클을 관리할 수 있습니다.

Operator의 CSV(클러스터 서비스 버전) 리소스에는 다음 유형의 Webhook를 정의하는 webhookdefinitions 섹션을 포함할 수 있습니다.

- 승인 Webhook(검증 및 변경)
- 변환 Webhook

프로세스

- Operator CSV의 spec 섹션에 webhookdefinitions 섹션을 추가하고 ValidatingAdmissionWebhook, MutatingAdmissionWebhook 또는 ConversionWebhook type을 사용하여 Webhook 정의를 포함합니다. 다음 예제에는 세 가지 유형의 Webhook가 모두 포함되어 있습니다.

Webhook를 포함하는 CSV

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
```

```
  name: webhook-operator.v0.0.1
spec:
  customresourcedefinitions:
    owned:
    - kind: WebhookTest
      name: webhooktests.webhook.operators.coreos.io 1
      version: v1
  install:
    spec:
      deployments:
      - name: webhook-operator-webhook
        ...
        ...
        ...
    strategy: deployment
  installModes:
  - supported: false
    type: OwnNamespace
  - supported: false
    type: SingleNamespace
  - supported: false
    type: MultiNamespace
  - supported: true
    type: AllNamespaces
  webhookdefinitions:
  - type: ValidatingAdmissionWebhook 2
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    failurePolicy: Fail
    generateName: vwebhooktest.kb.io
    rules:
    - apiGroups:
      - webhook.operators.coreos.io
      apiVersions:
      - v1
      operations:
      - CREATE
      - UPDATE
      resources:
      - webhooktests
    sideEffects: None
    webhookPath: /validate-webhook-operators-coreos-io-v1-webhooktest
  - type: MutatingAdmissionWebhook 3
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    failurePolicy: Fail
    generateName: mwebhooktest.kb.io
    rules:
```

```
    - apiGroups:
      - webhook.operators.coreos.io
      apiVersions:
      - v1
      operations:
      - CREATE
      - UPDATE
      resources:
      - webhooktests
    sideEffects: None
    webhookPath: /mutate-webhook-operators-coreos-io-v1-webhooktest
  - type: ConversionWebhook (4)
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    generateName: cwebhooktest.kb.io
    sideEffects: None
    webhookPath: /convert
    conversionCRDs:
    - webhooktests.webhook.operators.coreos.io (5)
...
```

(1)

변환 Webhook에서 대상으로 하는 CRD는 여기에 있어야 합니다.

(2)

검증 승인 Webhook입니다.

(3)

변경 승인 Webhook.

(4)

변환 Webhook입니다.

(5)

각 CRD의 spec.PreserveUnknownFields 속성을 false 또는 nil로 설정해야 합니다.

추가 리소스

- 웹 후크 승인 플러그인의 유형
- Kubernetes 설명서:
  - 승인 Webhook 검증
  - 변경 승인 Webhook
  - 변환 Webhook

5.7.7.1. OLM의 Webhook 고려 사항

OLM(Operator Lifecycle Manager)을 사용하여 Operator를 Webhook와 함께 배포할 때는 다음을 정의해야 합니다.

- type 필드는 ValidatingAdmissionWebhook, MutatingAdmissionWebhook 또는 ConversionWebhook 중 하나로 설정해야 합니다. 그러지 않으면 CSV가 실패한 단계에 배치됩니다.
- CSV에는 webhookdefinition의 deploymentName 필드에 제공된 값과 이름이 같은 배포가 포함되어야 합니다.

Webhook가 생성되면 OLM은 Operator가 배포된 Operator group과 일치하는 네임스페이스에서만 Webhook가 작동하는지 확인합니다.

인증 기관 제약 조건

OLM은 각 배포에 하나의 CA(인증 기관)를 제공하도록 구성되어 있습니다. 배포에 CA를 생성하고 마운트하는 논리는 원래 API 서비스 라이프사이클 논리에 사용되었습니다. 결과는 다음과 같습니다.

- TLS 인증서 파일이 배포에 마운트됩니다 (/apiserver.local.config/certificates/apiserver.crt).
- TLS 키 파일이 배포에 마운트됩니다(/apiserver.local.config/certificates/apiserver.key).

승인 Webhook 규칙 제약 조건

Operator에서 클러스터를 복구할 수 없는 상태로 구성하는 것을 방지하기 위해 OLM은 승인 Webhook에 정의된 규칙에서 다음 요청을 가로채는 경우 CSV를 실패한 단계에 배치합니다.

- 모든 그룹을 대상으로 하는 요청
- operators.coreos.com 그룹을 대상으로 하는 요청
- ValidatingWebhookConfigurations 또는 MutatingWebhookConfigurations 리소스를 대상으로 하는 요청

변환 Webhook 제약 조건

OLM은 변환 Webhook 정의가 다음 제약 조건을 준수하지 않는 경우 CSV를 실패한 단계에 배치합니다.

- 변환 Webhook가 있는 CSV는 AllNamespaces 설치 모드만 지원할 수 있습니다.
- 변환 Webhook에서 대상으로 하는 CRD는 spec.preserveUnknownFields 필드가 false 또는 nil로 설정되어 있어야 합니다.
- CSV에 정의된 변환 Webhook는 고유한 CRD를 대상으로 해야 합니다.
- 지정된 CRD의 전체 클러스터에는 하나의 변환 Webhook만 있을 수 있습니다.

5.7.8. CRD(사용자 정의 리소스 정의) 이해

Operator에서 사용할 수 있는 CRD(사용자 정의 리소스 정의)에는 두 가지 유형이 있습니다. 하나는 보유 CRD이고 다른 하나는 의존하는 필수 CRD입니다.

5.7.8.1. 보유 CRD

Operator가 보유한 CRD(사용자 정의 리소스 정의)는 CSV에서 가장 중요한 부분입니다. 이를 통해 Operator와 필수 RBAC 규칙, 종속성 관리 및 기타 Kubernetes 개념 간 관계가 설정됩니다.

Operator는 여러 CRD를 사용하여 개념을 함께 연결하는 것이 일반적입니다(한 오브젝트의 최상위 데이터베이스 구성 및 다른 오브젝트의 복제본 세트 표현 등). 각각은 CSV 파일에 나열되어야 합니다.

표 5.12. 보유 CRD 필드

필드	Description	필수/선택 사항
Name	CRD의 전체 이름입니다.	필수 항목
Version	해당 오브젝트 API의 버전입니다.	필수 항목
Kind	머신에서 읽을 수 있는 CRD 이름입니다.	필수 항목
DisplayName	사람이 읽을 수 있는 버전의 CRD 이름입니다(예: **MongoDB Standalone**).	필수 항목
Description	Operator에서 이 CRD를 사용하는 방법에 대한 간단한 설명 또는 CRD에서 제공하는 기능에 대한 설명입니다.	필수 항목
Group	이 CRD가 속하는 API 그룹입니다(예: **database.example.com**).	선택 사항
Resources	CRD에는 하나 이상의 Kubernetes 오브젝트 유형이 있습니다. 이러한 항목은 **resources** 섹션에 나열되어 문제 해결에 필요할 수 있는 오브젝트 또는 애플리케이션에 연결하는 방법을 사용자에게 알립니다(예: 데이터베이스를 표시하는 서비스 또는 수신 규칙). 오케스트레이션하는 모든 항목의 전체 목록이 아닌, 사용자에게 중요한 오브젝트만 나열하는 것이 좋습니다. 예를 들어 사용자가 수정할 수 없는 내부 상태를 저장하는 구성 맵은 나열하지 않도록 합니다.	선택 사항

필드	Description	필수/선택 사항
SpecDescriptors, **StatusDescriptors**, **ActionDescriptors**	이러한 설명자는 Operator의 특정 입력 또는 출력에서 최종 사용자에게 가장 중요한 UI를 나타내는 방법입니다. CRD에 사용자가 제공해야 하는 보안 또는 구성 맵의 이름이 있는 경우 여기에서 지정할 수 있습니다. 이러한 항목은 호환되는 UI에서 연결되고 강조됩니다. 설명자에는 세 가지 유형이 있습니다. ● **SpecDescriptors**: 오브젝트의 **spec** 블록에 있는 필드에 대한 참조입니다. ● **StatusDescriptors**: 오브젝트의 **status** 블록에 있는 필드에 대한 참조입니다. ● **ActionDescriptors**: 오브젝트에서 수행할 수 있는 작업에 대한 참조입니다. 모든 설명자에는 다음 필드를 사용할 수 있습니다. ● **DisplayName**: 사람이 읽을 수 있는 **Spec**, **Status** 또는 **Action**의 이름입니다. ● **Description**: **Spec**, **Status** 또는 **Action** 및 Operator에서 사용하는 방법에 대한 간단한 설명입니다. ● **Path**: 이 설명자에서 설명하는 오브젝트상의 점으로 구분된 필드 경로입니다. ● **X-Descriptors**: 이 설명자의 "기능"과 사용할 UI 구성 요소를 결정하는 데 사용합니다. OpenShift Container Platform의 표준 React UI X-Descriptor 목록은 **openshift/console** 프로젝트를 참조하십시오. 일반적으로 설명자에 대한 자세한 내용은 **openshift/console** 프로젝트를 참조하십시오.	선택 사항

다음 예제에서는 보안 및 구성 맵의 형태로 일부 사용자 입력이 필요하고 서비스, 상태 저장 세트, Pod, 구성 맵을 오케스트레이션하는 MongoDB Standalone CRD를 보여줍니다.

보유 CRD의 예

```
- displayName: MongoDB Standalone
  group: mongodb.com
  kind: MongoDbStandalone
  name: mongodbstandalones.mongodb.com
  resources:
    - kind: Service
      name: ''
```

```
      version: v1
    - kind: StatefulSet
      name: ''
      version: v1beta2
    - kind: Pod
      name: ''
      version: v1
    - kind: ConfigMap
      name: ''
      version: v1
  specDescriptors:
    - description: Credentials for Ops Manager or Cloud Manager.
      displayName: Credentials
      path: credentials
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:selector:core:v1:Secret'
    - description: Project this deployment belongs to.
      displayName: Project
      path: project
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:selector:core:v1:ConfigMap'
    - description: MongoDB version to be installed.
      displayName: Version
      path: version
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:label'
  statusDescriptors:
    - description: The status of each of the pods for the MongoDB cluster.
      displayName: Pod Status
      path: pods
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:podStatuses'
  version: v1
  description: >-
    MongoDB Deployment consisting of only one host. No replication of
    data.
```

5.7.8.2. 필수 CRD

기타 필수 CRD에 의존하는 것은 전적으로 선택 사항이며 개별 Operator의 범위를 줄이고 여러 Operator를 함께 구성하여 종단 간 사용 사례를 해결하는 방법을 제공하기 위해서만 존재합니다.

이에 대한 예로는 애플리케이션을 설정하고 (etcd Operator에서) 분산형 잠금에 사용할 etcd 클러스터를 설치하고, 데이터 저장을 위해 (Postgres Operator에서) Postgres 데이터베이스를 설치할 수 있는 Operator가 있습니다.

OLM(Operator Lifecycle Manager)은 이러한 요구 사항을 충족하기 위해 클러스터에서 사용 가능한

CRD 및 Operator를 점검합니다. 적합한 버전이 있는 경우 Operator는 원하는 네임스페이스 내에서 시작되고 각 Operator에서 필요한 Kubernetes 리소스를 생성, 조사, 수정할 수 있도록 서비스 계정이 생성됩니다.

표 5.13. 필수 CRD 필드

필드	Description	필수/선택 사항
Name	필요한 CRD의 전체 이름입니다.	필수 항목
Version	해당 오브젝트 API의 버전입니다.	필수 항목
Kind	Kubernetes 오브젝트 종류입니다.	필수 항목
DisplayName	사람이 읽을 수 있는 CRD 버전입니다.	필수 항목
Description	구성 요소가 더 큰 아키텍처에 어떻게 적용되는지 요약합니다.	필수 항목

필수 CRD 예제

```
required:
- name: etcdclusters.etcd.database.coreos.com
  version: v1beta2
  kind: EtcdCluster
  displayName: etcd Cluster
  description: Represents a cluster of etcd nodes.
```

5.7.8.3. CRD 업그레이드

OLM은 CRD(사용자 정의 리소스 정의)가 단수형 CSV(클러스터 서비스 버전)에 속하는 경우 CRD를 즉시 업그레이드합니다. CRD가 여러 CSV에 속하는 경우에는 다음과 같은 하위 호환 조건을 모두 충족할 때 CRD가 업그레이드됩니다.

- *현재 CRD의 기존 서비스 버전은 모두 새 CRD에 있습니다.*
- *CRD 제공 버전과 연결된 기존의 모든 인스턴스 또는 사용자 정의 리소스는 새 CRD의 검증 스키마에 대해 검증할 때 유효합니다.*

5.7.8.3.1. 새 CRD 버전 추가

절차

Operator에 새 버전의 CRD를 추가하려면 다음을 수행합니다.

1. CSV의 versions 섹션에서 CRD 리소스에 새 항목을 추가합니다.

   예를 들어 현재 CRD에 버전 v1alpha1이 있고, 새 버전 v1beta1을 추가한 후 새 스토리지 버전으로 표시하려면 v1beta1에 새 항목을 추가합니다.

```
versions:
 - name: v1alpha1
   served: true
   storage: false
 - name: v1beta1 1
   served: true
   storage: true
```

1 새 항목입니다.

2. CSV에서 새 버전을 사용하려고 하는 경우 CSV의 owned 섹션에 있는 CRD의 참조 버전이 업데이트되었는지 확인합니다.

```
customresourcedefinitions:
  owned:
  - name: cluster.example.com
    version: v1beta1 1
    kind: cluster
    displayName: Cluster
```

1 version을 업데이트합니다.

3. 업데이트된 CRD 및 CSV를 번들로 내보냅니다.

5.7.8.3.2. CRD 버전 사용 중단 또는 제거

OLM(Operator Lifecycle Manager)에서는 CRD(사용자 정의 리소스 정의)의 제공 버전을 즉시 제거

하는 것을 허용하지 않습니다. 대신 CRD의 served 필드를 false로 설정하여 더 이상 사용되지 않는 CRD 버전을 먼저 비활성화해야 합니다. 그러면 후속 CRD 업그레이드에서 제공되지 않는 버전을 제거할 수 있습니다.

절차

특정 버전의 CRD를 사용 중단하고 제거하려면 다음을 수행합니다.

1. 이 버전이 더 이상 사용되지 않으며 후속 업그레이드에서 제거될 수 있음을 나타내려면 더 이상 사용되지 않는 버전을 제공되지 않음으로 표시합니다. 예를 들면 다음과 같습니다.

```
versions:
 - name: v1alpha1
   served: false (1)
   storage: true
```

(1) false로 설정합니다.

2. 사용을 중단할 버전이 현재 storage 버전인 경우 storage 버전을 제공 버전으로 전환합니다. 예를 들면 다음과 같습니다.

```
versions:
 - name: v1alpha1
   served: false
   storage: false (1)
 - name: v1beta1
   served: true
   storage: true (2)
```

(1) (2) storage 필드를 적절하게 업데이트합니다.

참고

CRD에서 storage 버전이거나 이 버전이었던 특정 버전을 제거하려면 CRD 상태의 storedVersion에서 해당 버전을 제거해야 합니다. OLM은 저장된 버전이 새 CRD에 더 이상 존재하지 않는 것으로 탐지하면 이 작업을 수행합니다.

3.

3. 위 변경 사항으로 CRD를 업그레이드합니다.

4. 이어지는 업그레이드 주기에서는 서비스되지 않는 버전을 CRD에서 완전히 제거할 수 있습니다. 예를 들면 다음과 같습니다.

```
versions:
  - name: v1beta1
    served: true
    storage: true
```

5. CSV의 owned 섹션에서 참조하는 CRD 버전이 CRD에서 제거된 경우 적절하게 업데이트되었는지 확인합니다.

5.7.8.4. CRD 템플릿

Operator 사용자는 필수 옵션과 선택 옵션을 알고 있어야 합니다. 각 CRD(사용자 정의 리소스 정의)에 대한 템플릿에 alm-examples라는 주석으로 최소 구성 세트를 제공할 수 있습니다. 사용자가 추가로 사용자 정의할 수 있도록 호환되는 UI가 이 템플릿에 미리 채워집니다.

주석은 종류 목록으로 구성되며, 예를 들면 CRD 이름 및 Kubernetes 오브젝트의 해당 metadata 및 spec입니다.

다음 전체 예제에서는 EtcdCluster, EtcdBackup, EtcdRestore에 대한 템플릿을 제공합니다.

```
metadata:
  annotations:
    alm-examples: >-
      [{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdCluster","metadata":
{"name":"example","namespace":"default"},"spec":{"size":3,"version":"3.2.13"}},
{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdRestore","metadata":
{"name":"example-etcd-cluster"},"spec":{"etcdCluster":{"name":"example-etcd-
cluster"},"backupStorageType":"S3","s3":{"path":"<full-s3-path>","awsSecret":"<aws-
secret>"}}},
{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdBackup","metadata":
{"name":"example-etcd-cluster-backup"},"spec":{"etcdEndpoints":["<etcd-cluster-
endpoints>"],"storageType":"S3","s3":{"path":"<full-s3-path>","awsSecret":"<aws-
secret>"}}}]
```

5.7.8.5. 내부 오브젝트 숨기기

Operator는 작업을 수행하기 위해 내부적으로 CRD(사용자 정의 리소스 정의)를 사용하는 것이 일반적입니다. 이러한 오브젝트는 사용자 조작용이 아니며 Operator 사용자에게 혼동을 줄 수 있습니다. 예를

들어 데이터베이스 Operator에는 사용자가 replication: true를 사용하여 데이터베이스 오브젝트를 생성할 때마다 생성되는 Replication CRD가 있을 수 있습니다.

Operator 작성자는 operators.operatorframework.io/internal-objects 주석을 Operator의 CSV(클러스터 서비스 버전)에 추가하여 사용자 조작용이 아닌 사용자 인터페이스에서 CRD를 숨길 수 있습니다.

절차

1. CRD 중 하나를 내부로 표시하기 전에 애플리케이션을 관리하는 데 필요할 수 있는 디버깅 정보 또는 구성이 CR의 상태 또는 spec 블록에 반영되는지 확인합니다(Operator에 적용 가능한 경우).

2. operators.operatorframework.io/internal-objects 주석을 Operator의 CSV에 추가하여 사용자 인터페이스에서 숨길 내부 오브젝트를 지정합니다.

내부 오브젝트 주석

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: my-operator-v1.2.3
  annotations:
    operators.operatorframework.io/internal-objects:
'["my.internal.crd1.io","my.internal.crd2.io"]' 1
...
```

1 내부 CRD를 문자열 배열로 설정합니다.

5.7.8.6. 필수 사용자 정의 리소스 초기화

Operator가 완전히 작동하기 위해서는 사용자가 사용자 정의 리소스를 인스턴스화해야 할 수 있습니다. 그러나 사용자가 필요한 내용과 리소스를 정의하는 방법을 결정하는 것이 어려울 수 있습니다.

Operator 개발자는 Operator 설치 중에 operatorframework.io/initialization-resource 를 CSV(클러스터 서비스 버전)에 추가하여 필요한 단일 사용자 정의 리소스를 지정할 수 있습니다. 그런 다음 CSV에

제공된 템플릿을 통해 사용자 정의 리소스를 생성하라는 메시지가 표시됩니다. 주석에는 설치 중 리소스를 초기화하는 데 필요한 전체 YAML 정의가 포함된 템플릿이 포함되어야 합니다.

OpenShift Container Platform 웹 콘솔에서 Operator를 설치한 후 이 주석을 정의한 경우 사용자에게 CSV에 제공된 템플릿을 사용하여 리소스를 생성하라는 메시지가 표시됩니다.

절차

- *Operator의 CSV에 operatorframework.io/initialization-resource 주석을 추가하여 필요한 사용자 정의 리소스를 지정합니다. 예를 들어 다음 주석은 StorageCluster 리소스 생성이 필요하고 전체 YAML 정의를 제공합니다.*

  초기화 리소스 주석

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
 name: my-operator-v1.2.3
 annotations:
  operatorframework.io/initialization-resource: |-
    {
       "apiVersion": "ocs.openshift.io/v1",
       "kind": "StorageCluster",
       "metadata": {
          "name": "example-storagecluster"
       },
       "spec": {
          "manageNodes": false,
          "monPVCTemplate": {
             "spec": {
                "accessModes": [
                   "ReadWriteOnce"
                ],
                "resources": {
                   "requests": {
                      "storage": "10Gi"
                   }
                },
                "storageClassName": "gp2"
             }
          },
          "storageDeviceSets": [
             {
                "count": 3,
                "dataPVCTemplate": {
                   "spec": {
                      "accessModes": [
                         "ReadWriteOnce"
```

```
                    ],
                    "resources": {
                       "requests": {
                          "storage": "1Ti"
                       }
                    },
                    "storageClassName": "gp2",
                    "volumeMode": "Block"
                 }
              },
              "name": "example-deviceset",
              "placement": {},
              "portable": true,
              "resources": {}
           }
        ]
     }
  }
...
```

5.7.9. API 서비스 이해

CRD와 마찬가지로 Operator에서 사용할 수 있는 API 서비스에는 두 가지 유형, 즉 보유 및 필수가 있습니다.

5.7.9.1. 보유 API 서비스

API 서비스가 CSV에 속하는 경우 CSV는 해당 서비스를 지원하는 확장 api-server 및 해당 서비스에서 제공하는 GVK(그룹/버전/종류)의 배포에 대해 설명해야 합니다.

API 서비스는 제공하는 그룹/버전별로 고유하게 식별되며 제공할 것으로 예상되는 다양한 종류를 나타내기 위해 여러 번 나열될 수 있습니다.

표 5.14. 보유 API 서비스 필드

필드	Description	필수/선택 사항
Group	API 서비스에서 제공하는 그룹입니다(예: **database.example.com**).	필수 항목
Version	API 서비스의 버전입니다(예: **v1alpha1**).	필수 항목
Kind	API 서비스에서 제공해야 하는 종류입니다.	필수 항목

필드	Description	필수/선택 사항
Name	제공되는 API 서비스의 복수형 이름입니다.	필수 항목
DeploymentName	CSV에서 정의한 배포 이름으로, API 서비스에 해당합니다(보유 API 서비스의 경우 필수). OLM Operator는 CSV 보류 단계 동안 CSV의 **InstallStrategy**에서 이름이 일치하는 **Deployment** 사양을 검색한 후 찾을 수 없는 경우 CSV를 "설치 준비" 단계로 전환하지 않습니다.	필수 항목
DisplayName	사람이 읽을 수 있는 버전의 API 서비스 이름입니다(예: **MongoDB Standalone**).	필수 항목
Description	Operator에서 이 API 서비스를 사용하는 방법에 대한 간단한 설명 또는 API 서비스에서 제공하는 기능에 대한 설명입니다.	필수 항목
Resources	사용 중인 API 서비스에서 Kubernetes 오브젝트 유형을 한 개 이상 보유하고 있습니다. 이러한 항목은 resources 섹션에 나열되어 문제 해결에 필요할 수 있는 오브젝트 또는 애플리케이션에 연결하는 방법을 사용자에게 알립니다(예: 데이터베이스를 표시하는 서비스 또는 수신 규칙). 오케스트레이션하는 모든 항목의 전체 목록이 아닌, 사용자에게 중요한 오브젝트만 나열하는 것이 좋습니다. 예를 들어 사용자가 수정할 수 없는 내부 상태를 저장하는 구성 맵은 나열하지 않도록 합니다.	선택 사항
SpecDescriptors, **StatusDescriptors**, **ActionDescriptors**	본질적으로 보유 CRD와 동일합니다.	선택 사항

5.7.9.1.1. API 서비스 리소스 생성

OLM(Operator Lifecycle Manager)은 각각의 고유한 보유 API 서비스에 대해 서비스 및 API 서비스 리소스를 생성하거나 교체합니다.

- ***서비스 Pod 선택기는 API 서비스 설명의 DeploymentName 필드와 일치하는 CSV 배포에서 복사합니다.***
- ***각 설치에 대해 새 CA 키/인증서 쌍이 생성되고 base64로 인코딩된 CA 번들이 각 API 서비스 리소스에 포함됩니다.***

5.7.9.1.2. API 서비스 제공 인증서

OLM은 보유 API 서비스가 설치될 때마다 제공 키/인증서 쌍을 생성합니다. 제공 인증서에는 생성된 Service 리소스의 호스트 이름을 포함하는 CN(일반 이름)이 있으며 해당 API 서비스 리소스에 포함된 CA 번들의 개인 키로 서명합니다.

인증서는 배포 네임스페이스에 유형 kubernetes.io/tls의 보안으로 저장되고 API 서비스 설명의 DeploymentName 필드와 일치하는 CSV의 배포 볼륨 섹션에 apiservice-cert라는 볼륨이 자동으로 추가됩니다.

아직 존재하지 않는 경우 이름이 일치하는 볼륨 마운트도 해당 배포의 모든 컨테이너에 추가됩니다. 그러면 사용자가 필요한 이름으로 볼륨 마운트를 정의하여 모든 사용자 정의 경로 요구 사항을 충족할 수 있습니다. 생성된 볼륨 마운트의 경로는 기본적으로 /apiserver.local.config/certificates이며 동일한 경로의 기존 볼륨 마운트는 교체됩니다.

5.7.9.2. 필수 API 서비스

OLM은 설치를 시도하기 전에 모든 필수 CSV에 사용 가능한 API 서비스가 있고 필요한 GVK를 모두 검색할 수 있는지 확인합니다. 이를 통해 CSV는 보유하지 않는 API 서비스에서 제공하는 특정 종류에 의존할 수 있습니다.

표 5.15. 필수 API 서비스 필드

필드	Description	필수/선택 사항
Group	API 서비스에서 제공하는 그룹입니다(예: **database.example.com**).	필수 항목
Version	API 서비스의 버전입니다(예: **v1alpha1**).	필수 항목
Kind	API 서비스에서 제공해야 하는 종류입니다.	필수 항목
DisplayName	사람이 읽을 수 있는 버전의 API 서비스 이름입니다(예: **MongoDB Standalone**).	필수 항목
Description	Operator에서 이 API 서비스를 사용하는 방법에 대한 간단한 설명 또는 API 서비스에서 제공하는 기능에 대한 설명입니다.	필수 항목

5.8. 번들 이미지 작업

Operator SDK를 사용하여 OLM(Operator Lifecycle Manager)에서 Operator를 번들 형식으로 패키지, 배포, 업그레이드할 수 있습니다.

5.8.1. Operator 번들

Operator 번들 형식은 Operator SDK 및 Operator Lifecycle Manager (OLM)의 기본 패키지 메서드입니다. Operator SDK를 사용하여 Operator 프로젝트를 번들 이미지로 빌드하고 푸시하여 OLM에서 Operator를 사용할 수 있습니다.

사전 요구 사항

- *개발 워크스테이션에 Operator SDK CLI가 설치됨*
- *OpenShift CLI(oc) v4.11 이상이 설치됨*
- *Operator SDK를 사용하여 Operator 프로젝트를 초기화함*
- *Operator가 Go 기반인 경우 OpenShift Container Platform에서 실행하기 위해 지원되는 이미지를 사용하도록 프로젝트를 업데이트해야 함*

절차

1. *Operator 프로젝트 디렉터리에서 다음 make 명령을 실행하여 Operator 이미지를 빌드하고 내보냅니다. 액세스할 수 있는 리포지토리를 참조하려면 다음 단계에서 IMG 인수를 수정합니다. Quay.io와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.*

   a. *이미지를 빌드합니다.*

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   참고

   Operator에 대해 SDK에서 생성한 Dockerfile은 Go 빌드 를 위해 GOARCH=amd64 를 명시적으로 참조합니다. 이는AMD64 이외의 아키텍처의 경우 GOARCH=$CACHEGETARCH 에 수정될 수 있습니다. Docker는 환경 변수를 -platform 에서 지정한 값으로 자동 설정합니다. Buildah를 사용하면 -build-arg 를 목적으로 사용해야 합니다. 자세한 내용은 여러 아키텍처를 참조하십시오.

   b. *이미지를 리포지토리로 내보냅니다.*

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

-

2. Operator SDK generate bundle 및 bundle validate 명령을 비롯한 다양한 명령을 호출하는 make bundle 명령을 실행하여 Operator 번들 매니페스트를 생성합니다.

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Operator의 번들 매니페스트는 애플리케이션을 표시, 생성, 관리하는 방법을 설명합니다. make bundle 명령은 Operator 프로젝트에서 다음 파일 및 디렉터리를 생성합니다.

- ClusterServiceVersion 오브젝트를 포함하는 bundle/manifests라는 번들 매니페스트 디렉터리
- bundle/metadata라는 번들 메타데이터 디렉터리
- config/crd 디렉터리의 모든 CRD(사용자 정의 리소스 정의)
- Dockerfile bundle.Dockerfile

그런 다음 operator-sdk bundle validate를 사용하여 이러한 파일을 자동으로 검증하고 디스크상의 번들 표현이 올바른지 확인합니다.

3. 다음 명령을 실행하여 번들 이미지를 빌드하고 내보냅니다. OLM에서는 하나 이상의 번들 이미지를 참조하는 인덱스 이미지를 통해 Operator 번들을 사용합니다.

a. 번들 이미지를 빌드합니다. 이미지를 내보낼 레지스트리, 사용자 네임스페이스, 이미지 태그에 대한 세부 정보를 사용하여 BUNDLE_IMG를 설정합니다.

```
$ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:
<tag>
```

b. 번들 이미지를 내보냅니다.

```
$ docker push <registry>/<user>/<bundle_image_name>:<tag>
```

5.8.2. Operator Lifecycle Manager를 사용하여 Operator 배포

OLM(Operator Lifecycle Manager)은 Kubernetes 클러스터에서 Operator 및 관련 서비스를 설치, 업데이트하고 라이프사이클을 관리하는 데 도움이 됩니다. OLM은 기본적으로 OpenShift Container Platform에 설치되고 Kubernetes 확장으로 실행되므로 추가 툴 없이 모든 Operator 라이프사이클 관리 기능에 웹 콘솔과 OpenShift CLI(oc)를 사용할 수 있습니다.

Operator 번들 형식은 Operator SDK 및 OLM의 기본 패키지 메서드입니다. Operator SDK를 사용하여 OLM에서 번들 이미지를 신속하게 실행하여 올바르게 실행되는지 확인할 수 있습니다.

사전 요구 사항

- *개발 워크스테이션에 Operator SDK CLI가 설치됨*
- *Operator 번들 이미지를 빌드하여 레지스트리로 내보냄*
- *Kubernetes 기반 클러스터에 OLM이 설치되어 있음(apiextensions.k8s.io/v1 CRD(예: OpenShift Container Platform 4.11)를 사용하는 경우 v1.16.0 이상)*
- *cluster-admin 권한이 있는 계정을 사용하여 oc로 클러스터에 로그인됨*
- *Operator가 Go 기반인 경우 OpenShift Container Platform에서 실행하기 위해 지원되는 이미지를 사용하도록 프로젝트를 업데이트해야 함*

절차

1. *다음 명령을 입력하여 클러스터에서 Operator를 실행합니다.*

```
$ operator-sdk run bundle \ 1
  -n <namespace> \ 2
  <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. *run bundle 명령은 유효한 파일 기반 카탈로그를 생성하고 OLM을 사용하여 클러스터에 Operator 번들을 설치합니다.*

2.

3

이미지를 지정하지 않으면 명령에서 **quay.io/operator-framework/opm:latest** 를 기본 인덱스 이미지로 사용합니다. 이미지를 지정하면 명령에서 번들 이미지 자체를 인덱스 이미지로 사용합니다.

중요

OpenShift Container Platform 4.11부터 **run bundle** 명령은 기본적으로 **Operator** 카탈로그의 파일 기반 카탈로그 형식을 지원합니다. **Operator** 카탈로그의 더 이상 사용되지 않는 **SQLite** 데이터베이스 형식은 계속 지원되지만 향후 릴리스에서 제거됩니다. **Operator** 작성자는 해당 워크플로우를 파일 기반 카탈로그 형식으로 마이그레이션하는 것이 좋습니다.

이 명령은 다음 작업을 수행합니다.

- 번들 이미지를 참조하는 인덱스 이미지를 생성합니다. 인덱스 이미지는 불투명하고 일시적이지만 프로덕션에서 카탈로그에 번들을 추가하는 방법을 정확하게 반영합니다.
- **OperatorHub**에서 **Operator**를 검색할 수 있도록 새 인덱스 이미지를 가리키는 카탈로그 소스를 생성합니다.
- **OperatorGroup**,**Subscription**,**InstallPlan** 및 **RBAC**를 포함한 기타 모든 필수 리소스를 생성하여 **Operator**를 클러스터에 배포합니다.

추가 리소스

- **Operator** 프레임워크 패키지 형식의 파일 기반 카탈로그
- 사용자 정의 카탈로그 관리의 파일 기반 카탈로그
- 번들 형식

5.8.3. 번들 Operator가 포함된 카탈로그 게시

Operator를 설치하고 관리하려면 **OLM(Operator Lifecycle Manager)**이 클러스터의 카탈로그에서 참

조하는 인덱스 이미지에 **Operator** 번들을 나열해야 합니다. **Operator** 작성자는 **Operator SDK**를 사용하여 **Operator** 및 모든 종속 항목에 대한 번들이 포함된 인덱스를 생성할 수 있습니다. 이 기능은 원격 클러스터에서 테스트하고 컨테이너 레지스트리에 게시하는 데 유용합니다.

참고

Operator SDK는 **opm CLI**를 사용하여 인덱스 이미지 생성을 용이하게 합니다. **opm** 명령에 대한 경험이 필요하지 않습니다. 고급 사용 사례의 경우 **Operator SDK** 대신 **opm** 명령을 직접 사용할 수 있습니다.

사전 요구 사항

- 개발 워크스테이션에 **Operator SDK CLI**가 설치됨
- **Operator** 번들 이미지를 빌드하여 레지스트리로 내보냄
- **Kubernetes** 기반 클러스터에 **OLM**이 설치되어 있음(**apiextensions.k8s.io/v1 CRD**(예: **OpenShift Container Platform 4.11**)를 사용하는 경우 **v1.16.0** 이상)
- **cluster-admin** 권한이 있는 계정을 사용하여 **oc**로 클러스터에 로그인됨

프로세스

1. **Operator** 프로젝트 디렉터리에서 다음 **make** 명령을 실행하여 **Operator** 번들이 포함된 인덱스 이미지를 빌드합니다.

```
$ make catalog-build CATALOG_IMG=<registry>/<user>/<index_image_name>:<tag>
```

여기서 **CATALOG_IMG** 인수는 액세스할 수 있는 리포지토리를 참조합니다. **Quay.io**와 같은 리포지토리 사이트에 컨테이너를 저장하기 위해 계정을 받을 수 있습니다.

2. 빌드 인덱스 이미지를 리포지토리로 푸시합니다.

```
$ make catalog-push CATALOG_IMG=<registry>/<user>/<index_image_name>:<tag>
```

작은 정보

한 번에 여러 작업을 순서대로 수행하는 경우 **Operator SDK make** 명령을 함께 사용할 수 있습니다. 예를 들어 **Operator** 프로젝트에 대한 번들 이미지를 아직 빌드하지 않은 경우 다음 구문을 사용하여 번들 이미지와 인덱스 이미지를 빌드하고 푸시할 수 있습니다.

```
$ make bundle-build bundle-push catalog-build catalog-push \
    BUNDLE_IMG=<bundle_image_pull_spec> \
    CATALOG_IMG=<index_image_pull_spec>
```

또는 **Makefile**의 **IMAGE_TAG_BASE** 필드를 기존 리포지토리로 설정할 수도 있습니다.

```
IMAGE_TAG_BASE=quay.io/example/my-operator
```

다음 구문을 사용하여 자동으로 생성된 이름으로 이미지를 빌드하고 푸시할 수 있습니다 (예: 번들 이미지의 경우 **quay.io/example/my-operator-bundle:v0.0.1** 및 인덱스 이미지의 경우 **quay.io/example/my-operator-catalog:v0.0.1.**)

```
$ make bundle-build bundle-push catalog-build catalog-push
```

3. 방금 생성한 인덱스 이미지를 참조하는 **CatalogSource** 오브젝트를 정의한 다음 **oc apply** 명령 또는 웹 콘솔을 사용하여 오브젝트를 생성합니다.

CatalogSource YAML의 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: cs-memcached
  namespace: default
spec:
  displayName: My Test
  publisher: Company
  sourceType: grpc
  image: quay.io/example/memcached-catalog:v0.0.1 (1)
  updateStrategy:
    registryPoll:
      interval: 10m
```

1

CATALOG_IMG 인수와 함께 이전에 사용한 이미지 가져오기 사양으로 image를 설정합니다.

4. *카탈로그 소스를 확인합니다.*

```
$ oc get catalogsource
```

출력 예

```
NAME          DISPLAY    TYPE   PUBLISHER   AGE
cs-memcached  My Test    grpc   Company     4h31m
```

검증

1. *카탈로그를 사용하여 Operator를 설치합니다.*

   a. *OperatorGroup 오브젝트를 정의하고 oc apply 명령 또는 웹 콘솔을 사용하여 생성합니다.*

   OperatorGroup YAML의 예

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

   b. *Subscription 오브젝트를 정의하고 oc apply 명령 또는 웹 콘솔을 사용하여 생성합니*

다.

Subscription YAML의 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: catalogtest
  namespace: default
spec:
  channel: "alpha"
  installPlanApproval: Manual
  name: catalog
  source: cs-memcached
  sourceNamespace: default
  startingCSV: memcached-operator.v0.0.1
```

2. 설치된 Operator가 실행 중인지 확인합니다.

a. Operator 그룹을 확인합니다.

```
$ oc get og
```

출력 예

```
NAME         AGE
my-test      4h40m
```

b. CSV(클러스터 서비스 버전)를 확인합니다.

```
$ oc get csv
```

출력 예

```
NAME                    DISPLAY   VERSION   REPLACES   PHASE
memcached-operator.v0.0.1   Test      0.0.1             Succeeded
```

c. Operator의 Pod를 확인합니다.

```
$ oc get pods
```

출력 예

```
NAME                                                READY   STATUS      RESTARTS   AGE
9098d908802769fbde8bd45255e69710a9f8420a8f3d814abe88b68f8ervdj6   0/1
Completed   0          4h33m
catalog-controller-manager-7fd5b7b987-69s4n                 2/2     Running     0
4h32m
cs-memcached-7622r                                  1/1     Running     0          4h33m
```

추가 리소스

- 고급 사용 사례는 opm CLI를 직접 사용하는 방법에 대한 자세한 내용은 사용자 정의 카탈로그 관리를 참조하십시오.

5.8.4. Operator Lifecycle Manager에서 Operator 업그레이드 테스트

인덱스 이미지 및 카탈로그 소스를 수동으로 관리하지 않아도 Operator SDK에서 OLM(Operator Lifecycle Manager) 통합을 사용하여 Operator 업그레이드를 신속하게 테스트할 수 있습니다.

run bundle-upgrade 하위 명령은 최신 버전의 번들 이미지를 지정하여 설치된 Operator가 최신 버전으로 업그레이드되도록 트리거하는 작업을 자동화합니다.

사전 요구 사항

- *run bundle 하위 명령을 사용하거나 기존 OLM 설치와 함께 OLM과 함께 Operator 설치*

- *번들 이미지에 설치된 Operator의 최신 버전이 표시됨*

프로세스

1. *OLM을 사용하여 Operator가 아직 설치되지 않은 경우 run bundle 하위 명령을 사용하거나 기존 OLM 설치와 함께 이전 버전을 설치합니다.*

참고

OLM을 사용하여 이전 버전의 번들이 설치된 경우 카탈로그 소스에서 참조하는 인덱스 이미지에 업그레이드하려는 최신 번들이 없어야 합니다. 그렇지 않으면 패키지 및 CSV(클러스터 서비스 버전)를 제공하는 인덱스에서 최신 번들을 이미 참조하므로 run bundle-upgrade 하위 명령을 실행하면 레지스트리 pod가 실패합니다.

예를 들어 이전 번들 이미지를 지정하여 Memcached Operator에 다음 run bundle 하위 명령을 사용할 수 있습니다.

```
$ operator-sdk run bundle <registry>/<user>/memcached-operator:v0.0.1
```

출력 예

```
INFO[0006] Creating a File-Based Catalog of the bundle "quay.io/demo/memcached-
operator:v0.0.1"
INFO[0008] Generated a valid File-Based Catalog
INFO[0012] Created registry pod: quay-io-demo-memcached-operator-v1-0-1
INFO[0012] Created CatalogSource: memcached-operator-catalog
INFO[0012] OperatorGroup "operator-sdk-og" created
INFO[0012] Created Subscription: memcached-operator-v0-0-1-sub
INFO[0015] Approved InstallPlan install-h9666 for the Subscription: memcached-
operator-v0-0-1-sub
INFO[0015] Waiting for ClusterServiceVersion "my-project/memcached-
operator.v0.0.1" to reach 'Succeeded' phase
INFO[0015] Waiting for ClusterServiceVersion ""my-project/memcached-
operator.v0.0.1" to appear
INFO[0026] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1"
phase: Pending
INFO[0028] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1"
phase: Installing
```

```
INFO[0059] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1"
phase: Succeeded
INFO[0059] OLM has successfully installed "memcached-operator.v0.0.1"
```

2. *최신 **Operator** 버전에 번들 이미지를 지정하여 설치한 **Operator**를 업그레이드합니다.*

```
$ operator-sdk run bundle-upgrade <registry>/<user>/memcached-operator:v0.0.2
```

출력 예

```
INFO[0002] Found existing subscription with name memcached-operator-v0-0-1-sub
and namespace my-project
INFO[0002] Found existing catalog source with name memcached-operator-catalog
and namespace my-project
INFO[0008] Generated a valid Upgraded File-Based Catalog
INFO[0009] Created registry pod: quay-io-demo-memcached-operator-v0-0-2
INFO[0009] Updated catalog source memcached-operator-catalog with address and
annotations
INFO[0010] Deleted previous registry pod with name "quay-io-demo-memcached-
operator-v0-0-1"
INFO[0041] Approved InstallPlan install-gvcjh for the Subscription: memcached-
operator-v0-0-1-sub
INFO[0042] Waiting for ClusterServiceVersion "my-project/memcached-
operator.v0.0.2" to reach 'Succeeded' phase
INFO[0019] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2"
phase: Pending
INFO[0042] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2"
phase: InstallReady
INFO[0043] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2"
phase: Installing
INFO[0044] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2"
phase: Succeeded
INFO[0044] Successfully upgraded to "memcached-operator.v0.0.2"
```

3. *설치된 **Operator**를 정리합니다.*

```
$ operator-sdk cleanup memcached-operator
```

추가 리소스

- OLM을 사용한 기존 Operator 설치

5.8.5. OpenShift Container Platform 버전과 Operator 호환성 제어

중요

Kubernetes는 후속 릴리스에서 제거된 특정 API를 주기적으로 사용하지 않습니다. Operator가 더 이상 사용되지 않는 API를 사용하는 경우 OpenShift Container Platform 클러스터가 제거된 Kubernetes 버전으로 업그레이드된 후 더 이상 작동하지 않을 수 있습니다.

Operator 작성자는 Kubernetes 문서에서 더 이상 사용되지 않는 API 마이그레이션 가이드를 검토하고 더 이상 사용되지 않거나 삭제된 API를 사용하지 않도록 Operator 프로젝트를 최신 상태로 유지하는 것이 좋습니다. Operator와 호환되지 않는 향후 버전의 OpenShift Container Platform을 릴리스하기 전에 Operator를 업데이트하는 것이 좋습니다.

OpenShift Container Platform 버전에서 API가 제거되면 제거된 API를 계속 사용하는 클러스터 버전에서 실행되는 Operator가 더 이상 제대로 작동하지 않게 됩니다. Operator 작성자는 Operator 사용자의 중단을 방지하기 위해 API 사용 중단 및 제거를 수용하도록 Operator 프로젝트를 업데이트해야 합니다.

작은 정보

Operator의 이벤트 경고를 확인하여 현재 사용 중인 API에 대한 경고가 있는지 확인할 수 있습니다. 다음 릴리스에서 제거될 사용 중인 API를 감지하면 다음 경고가 실행됩니다.

APIRemovedInNextReleaseInUse

OpenShift Container Platform의 다음 릴리스에서 제거될 API

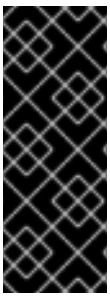
APIRemovedInNextEUSReleaseInUse

OpenShift Container Platform EUS (Extended Update Support)의 다음 릴리스에서 제거될 API

클러스터 관리자가 다음 버전의 OpenShift Container Platform으로 업그레이드하기 전에 Operator를 설치한 경우 다음 클러스터 버전과 호환되는 Operator 버전이 설치되어 있는지 확인해야 합니다. 이전 버전의 OpenShift Container Platform에서 계속 사용할 수 있도록 Operator 프로젝트를 더 이상 사용되지 않거나 제거된 API와 함께 게시해야 하는 경우에도 Operator 프로젝트를 업데이트하는 것이 좋습니다.

*다음 절차에서는 관리자가 호환되지 않는 **OpenShift Container Platform** 버전에 **Operator** 버전을 설치하지 않도록 하는 데 도움이 됩니다. 이러한 단계를 수행하면 관리자가 현재 클러스터에 설치된 **Operator** 버전과 호환되지 않는 최신 버전의 **OpenShift Container Platform**으로 업그레이드할 수 없습니다.*

*이 절차는 특정 **OpenShift Container Platform** 버전에서 현재 버전의 **Operator**가 제대로 작동하지 않는 경우에도 유용합니다. **Operator**를 배포해야 하는 클러스터 버전을 정의하면 **Operator**가 허용 범위 외부에 있는 클러스터 버전의 카탈로그에 표시되지 않아야 합니다.*



중요

*클러스터 관리자가 **API**가 더 이상 지원되지 않는 향후 **OpenShift Container Platform** 버전으로 업그레이드할 때 더 이상 사용되지 않는 **API**를 사용하는 **Operator**는 중요한 워크로드에 부정적인 영향을 미칠 수 있습니다. **Operator**가 더 이상 사용되지 않는 **API**를 사용하는 경우 최대한 빨리 **Operator** 프로젝트에서 다음 설정을 구성해야 합니다.*

사전 요구 사항

- *기존 **Operator** 프로젝트*

프로세스

1. *특정 **Operator** 번들이 지원되지 않고 특정 클러스터 버전 이후의 **OpenShift Container Platform**에서 올바르게 작동하지 않는 경우 **Operator**가 호환되는 최대 버전의 **OpenShift Container Platform**을 구성합니다. **Operator** 프로젝트의 **CSV(**클러스터 서비스 버전**)**에서 **olm.maxOpenShiftVersion** 주석을 설정하여 설치된 **Operator**를 호환 버전으로 업그레이드하기 전에 관리자가 클러스터를 업그레이드하지 못하도록 합니다.*

중요

***Operator** 번들 버전이 이후 버전에서 작동할 수 없는 경우에만 **olm.maxOpenShiftVersion** 주석을 사용해야 합니다. 클러스터 관리자는 솔루션이 설치된 클러스터를 업그레이드할 수 없습니다. 이후 버전 및 유효한 업그레이드 경로를 제공하지 않으면 클러스터 관리자가 **Operator**를 제거하고 클러스터 버전을 업그레이드할 수 있습니다.*

***olm.maxOpenShiftVersion** 주석이 있는 **CSV**의 예*

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

```
kind: ClusterServiceVersion
metadata:
  annotations:
    "olm.properties": '[{"type": "olm.maxOpenShiftVersion", "value": "
<cluster_version>"}]' (1)
```

(1)

Operator와 호환되는 최대 OpenShift Container Platform 클러스터 버전을 지정합니다. 예를 들어 value를 4.9로 설정하면 이 번들을 클러스터에 설치할 때 OpenShift Container Platform 버전 4.9 이후 버전으로 클러스터를 업그레이드할 수 없습니다.

2. 번들이 Red Hat 제공 Operator 카탈로그에 배포되도록 설계된 경우 다음 속성을 설정하여 Operator에 대해 호환되는 OpenShift Container Platform 버전을 구성합니다. 이 구성을 사용하면 Operator가 OpenShift Container Platform의 대상 호환 버전을 대상으로 하는 카탈로그에만 포함됩니다.

참고

이 단계는 Red Hat 제공 카탈로그에서 Operator를 게시할 때만 유효합니다. 번들이 사용자 지정 카탈로그의 배포 전용인 경우 이 단계를 건너뛸 수 있습니다. 자세한 내용은 "Red Hat 제공 Operator 카탈로그"를 참조하십시오.

a. 프로젝트의 bundle/metadata/annotations.yaml 파일에com.redhat.openshift.versions 주석을 설정합니다.

호환 버전이 있는 bundle/metadata/annotations.yaml 파일의 예

```
com.redhat.openshift.versions: "v4.7-v4.9" (1)
```

(1)

범위 또는 단일 버전으로 설정합니다.

b.

번들이 호환되지 않는 **OpenShift Container Platform** 버전으로 전달되지 않도록 하려면 **Operator** 번들 이미지에 있는 적절한 **com.redhat.openshift.versions** 라벨을 사용하여 인덱스 이미지가 생성되어야 합니다. 예를 들어 **Operator SDK**를 사용하여 프로젝트가 생성된 경우 **bundle.Dockerfile** 파일을 업데이트합니다.

호환 버전이 있는 **bundle.Dockerfile**의 예

```
LABEL com.redhat.openshift.versions="<versions>" 1
```

1

범위 또는 단일 버전으로 설정합니다 (예: **v4.7-v4.9**). 이 설정은 **Operator**를 배포해야 하는 클러스터 버전을 정의하고 범위 외부에 있는 클러스터 버전의 카탈로그에는 **Operator**가 표시되지 않습니다.

이제 새 버전의 **Operator**를 번들하고 업데이트된 버전을 배포 카탈로그에 게시할 수 있습니다.

추가 리소스

- 인증된 **Operator** 빌드 가이드에서 **OpenShift** 버전 관리
- 설치된 **Operator** 업데이트
- **Red Hat** 제공 **Operator** 카탈로그

5.8.6. 추가 리소스

- 번들 형식에 대한 자세한 내용은 **Operator Framework** 패키징 형식을 참조하십시오.
- **opm** 명령을 사용하여 번들 이미지를 인덱스 이미지에 추가하는 방법에 대한 자세한 내용은 사용자 정의 카탈로그 관리를 참조하십시오.
-

설치된 Operator의 업그레이드 작동 방식에 대한 자세한 내용은 Operator Lifecycle Manager 워크플로 를 참조하십시오.

5.9. POD 보안 승인 준수

Pod 보안 허용 은 Kubernetes Pod 보안 표준을 구현한 것입니다. Pod 보안 허용 은 Pod의 동작을 제한합니다. 전역 또는 네임스페이스 수준에서 정의된 Pod 보안 승인을 준수하지 않는 Pod는 클러스터에 허용되지 않으며 실행할 수 없습니다.

Operator 프로젝트에서 에스컬레이션된 권한을 실행할 필요가 없는 경우 제한된 Pod 보안 수준으로 설정된 네임스페이스에서 워크로드가 실행되도록 할 수 있습니다. Operator 프로젝트를 실행하려면 에스컬레이션된 권한이 필요한 경우 다음과 같은 보안 컨텍스트 구성을 설정해야 합니다.

- Operator의 네임스페이스에 허용되는 Pod 보안 승인 수준
- 워크로드 서비스 계정에 허용되는 SCC(보안 컨텍스트 제약 조건)

자세한 내용은 Pod 보안 승인 이해 및 관리를 참조하십시오.

5.9.1. Pod 보안 표준을 사용한 보안 컨텍스트 제약 조건 동기화

OpenShift Container Platform에는 Kubernetes Pod 보안 승인이 포함되어 있습니다. 전체적으로 권한 프로파일이 적용되며 제한된 프로필은 경고 및 감사에 사용됩니다.

글로벌 Pod 보안 승인 제어 구성 외에도 Pod 보안 허용 제어 경고 및 감사 레이블을 지정된 네임스페이스에 있는 서비스 계정의 SCC 권한에 따라 네임스페이스에 적용하는 컨트롤러가 있습니다.

중요

클러스터 페이로드의 일부로 정의된 네임스페이스에는 Pod 보안 승인 동기화가 영구적으로 비활성화되어 있습니다. 필요에 따라 다른 네임스페이스에서 Pod 보안 승인 동기화를 활성화할 수 있습니다.

컨트롤러는 각 네임스페이스에서 보안 컨텍스트 제약 조건을 사용하도록 ServiceAccount 오브젝트 권한을 검사합니다. SCC(보안 컨텍스트 제약 조건)는 필드 값을 기반으로 Pod 보안 프로필에 매핑됩니다. 컨트롤러는 이러한 변환된 프로필을 사용합니다. Pod 보안 허용 경고 및 감사 레이블은 Pod가 생성될

*때 경고 및 감사 로깅을 방지하기 위해 네임스페이스에 있는 가장 권한 있는 **Pod** 보안 프로필로 설정됩니다.*

네임스페이스 레이블 지정은 네임스페이스 로컬 서비스 계정 권한 고려를 기반으로 합니다.

***Pod**를 직접 적용하면 **Pod**를 실행하는 사용자의 **SCC** 권한을 사용할 수 있습니다. 그러나 사용자 권한은 자동 레이블 지정 중에 고려되지 않습니다.*

5.9.2. 제한된 Pod 보안 수준으로 설정된 네임스페이스에서 Operator 워크로드가 실행되도록 합니다.

***Operator** 프로젝트를 다양한 배포 및 환경에서 실행할 수 있도록 제한된 **Pod** 보안 수준으로 설정된 네임스페이스에서 실행하도록 **Operator**의 워크로드를 구성합니다.*

주의

***runAsUser** 필드를 비워 두어야 합니다. 이미지에 특정 사용자가 필요한 경우 제한된 **SCC**(보안 컨텍스트 제약 조건) 및 제한된 **Pod** 보안 적용에서 실행할 수 없습니다.*

절차

- *제한된 **Pod** 보안 수준으로 설정된 네임스페이스에서 실행되도록 **Operator** 워크로드를 구성하려면 다음 예와 유사한 **Operator**의 네임스페이스 정의를 편집합니다.*

중요

***Operator**의 네임스페이스 정의에 **seccomp** 프로필을 설정하는 것이 좋습니다. 그러나 **OpenShift Container Platform 4.10**에서는 **seccomp** 프로필 설정이 지원되지 않습니다.*

  - *OpenShift Container Platform 4.11 이상에서만 실행해야 하는 Operator 프로젝트의 경우 다음 예와 유사한 Operator의 네임스페이스 정의를 편집합니다.*

***config/manager/manager.yaml** 파일 예*

```
...
spec:
 securityContext:
  seccompProfile:
   type: RuntimeDefault (1)
  runAsNonRoot: true
 containers:
  - name: <operator_workload_container>
    securityContext:
     allowPrivilegeEscalation: false
     capabilities:
      drop:
       - ALL
...
```

(1)

seccomp 프로필 유형을 RuntimeDefault 로 설정하면 SCC는 기본적으로 네임스페이스의 Pod 보안 프로파일로 설정됩니다.

- *OpenShift Container Platform 4.10에서도 실행해야 하는 Operator 프로젝트의 경우 다음 예와 유사한 Operator의 네임스페이스 정의를 편집합니다.*

config/manager/manager.yaml 파일 예

```
...
spec:
 securityContext: (1)
  runAsNonRoot: true
 containers:
  - name: <operator_workload_container>
    securityContext:
     allowPrivilegeEscalation: false
     capabilities:
      drop:
       - ALL
...
```

1

seccomp 프로필 유형을 설정되지 않은 상태로 두면 Operator 프로젝트가 OpenShift Container Platform 4.10에서 실행될 수 있습니다.

추가 리소스

- *보안 컨텍스트 제약 조건 관리*

5.9.3. 에스컬레이션된 권한이 필요한 Operator 워크로드에 대한 Pod 보안 승인 관리

Operator 프로젝트에서 실행할 수 있는 권한을 에스컬레이션해야 하는 경우 Operator의 CSV(클러스터 서비스 버전)를 편집해야 합니다.

절차

1. *다음 예와 유사하게 보안 컨텍스트 구성을 Operator CSV에서 필요한 권한 수준으로 설정합니다.*

   네트워크 관리자 권한이 있는 &lt ;operator_name>.clusterserviceversion.yaml 파일의 예

```
...
containers:
  - name: my-container
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        add:
          - "NET_ADMIN"
...
```

2. *다음 예와 유사하게 Operator의 워크로드가 필요한 SCC(보안 컨텍스트 제약 조건)를 사용하도록 허용하는 서비스 계정 권한을 설정합니다.*

   예: <operator_name>.clusterserviceversion.yaml 파일

```
...
 install:
   spec:
    clusterPermissions:
    - rules:
      - apiGroups:
        - security.openshift.io
        resourceNames:
        - privileged
        resources:
        - securitycontextconstraints
        verbs:
        - use
      serviceAccountName: default
...
```

3. *Operator의 CSV 설명을 편집하여 Operator 프로젝트에 다음 예와 유사한 에스컬레이션 권한이 필요한 이유를 설명합니다.*

예: <operator_name>.clusterserviceversion.yaml 파일

```
...
spec:
 apiservicedefinitions:{}
 ...
description: The <operator_name> requires a privileged pod security admission label
set on the Operator's namespace. The Operator's agents require escalated
permissions to restart the node if the node needs remediation.
```

5.9.4. 추가 리소스

- *Pod 보안 승인 이해 및 관리*

5.10. 스코어 카드 툴을 사용하여 OPERATOR 검증

Operator 작성자는 Operator SDK의 스코어 카드 툴을 사용하여 다음 작업을 수행할 수 있습니다.

- *Operator 프로젝트에 구문 오류가 없고 올바르게 패키지되었는지 확인합니다.*

- *Operator를 개선할 수 있는 방법에 대한 제안 사항 검토*

5.10.1. 스코어 카드 툴 정보

Operator SDK bundle validate 하위 명령은 콘텐츠 및 구조에 대한 로컬 번들 디렉터리 및 원격 번들 이미지를 검증할 수 있지만 scorecard 명령을 사용하면 구성 파일 및 테스트 이미지를 기반으로 Operator에서 테스트를 실행할 수 있습니다. 이러한 테스트는 스코어 카드에 의해 실행되도록 구성된 테스트 이미지 내에서 구현됩니다.

스코어 카드는 OpenShift Container Platform과 같이 구성된 Kubernetes 클러스터에 대한 액세스 권한을 사용하여 실행된다고 가정합니다. 스코어 카드는 각 테스트를 Pod 내에서 실행하며 해당 Pod에서 로그가 집계되고 테스트 결과가 콘솔로 전송됩니다. 스코어 카드에는 기본 테스트 및 OLM(Operator Lifecycle Manager) 테스트가 내장되어 있으며 사용자 정의 테스트 정의를 실행하는 방법도 제공합니다.

스코어 카드 워크플로

1. *관련 CR(사용자 정의 리소스) 및 Operator에 필요한 모든 리소스를 생성합니다.*

2. *Operator 배포에 프록시 컨테이너를 생성하여 API 서버에 대한 호출을 기록하고 테스트를 실행합니다.*

3. *CR의 매개변수 검사*

스코어 카드 테스트에서는 테스트 중인 Operator의 상태를 가정하지 않습니다. Operator에 대한 Operator 및 CR 생성은 스코어 카드 자체의 범위를 벗어납니다. 그러나 테스트가 리소스 생성을 위해 설계된 경우 필요한 모든 리소스를 생성할 수 있습니다.

scorecard 명령 구문

```
$ operator-sdk scorecard <bundle_dir_or_image> [flags]
```

스코어 카드에는 Operator 번들에 대한 디스크상의 경로 또는 번들 이미지 이름에 대한 위치 인수가 필요합니다.

플래그에 대한 자세한 내용을 보려면 다음을 실행합니다.

```
$ operator-sdk scorecard -h
```

5.10.2. 스코어 카드 구성

스코어 카드 툴에서는 여러 글로벌 구성 옵션과 내부 플러그인을 구성할 수 있는 구성을 사용합니다. 테스트는 bundle/ 디렉터리에 있는 make bundle 명령으로 생성되는 config.yaml 구성 파일로 구동됩니다.

```
./bundle
...
└── tests
    └── scorecard
        └── config.yaml
```

스코어 카드 구성 파일 예제

```
kind: Configuration
apiversion: scorecard.operatorframework.io/v1alpha3
metadata:
  name: config
stages:
- parallel: true
  tests:
  - image: quay.io/operator-framework/scorecard-test:v1.22.2
    entrypoint:
    - scorecard-test
    - basic-check-spec
    labels:
      suite: basic
      test: basic-check-spec-test
  - image: quay.io/operator-framework/scorecard-test:v1.22.2
    entrypoint:
    - scorecard-test
    - olm-bundle-validation
    labels:
      suite: olm
      test: olm-bundle-validation-test
```

구성 파일은 스코어 카드로 실행할 수 있는 각 테스트를 정의합니다. 스코어 카드 구성 파일의 다음 필드는 다음과 같이 테스트를 정의합니다.

구성 필드	설명
image	테스트를 구현하는 컨테이너 이미지 이름 테스트
entrypoint	테스트를 실행하기 위해 테스트 이미지에서 호출되는 명령 및 인수
labels	실행할 테스트를 선택하는 스코어 카드 정의 또는 사용자 정의 라벨

5.10.3. 기본 제공 스코어 카드 테스트

*스코어 카드는 도구 모음(기본 테스트 도구 모음 및 **OLM(Operator Lifecycle Manager)** 도구 모음)으로 준비된 사전 정의 테스트와 함께 제공됩니다.*

표 5.16. 기본 테스트 모음

테스트	설명	짧은 이름
Spec Block Exists	이 테스트에서는 클러스터에서 생성된 모든 CR(사용자 정의 리소스)에 **spec** 블록이 있는지 확인합니다.	**basic-check-spec-test**

표 5.17. OLM 테스트 도구 모음

테스트	설명	짧은 이름
Bundle Validation	이 테스트에서는 스코어 카드로 전달되는 번들에 있는 번들 매니페스트를 검증합니다. 번들 콘텐츠에 오류가 포함된 경우 테스트 결과 출력에 검증기 로그 및 검증 라이브러리의 오류 메시지가 포함됩니다.	**olm-bundle-validation-test**
Provided APIs Have Validation	이 테스트에서는 제공된 CR의 CRD(사용자 정의 리소스 정의)에 검증 섹션이 포함되어 있고 CR에서 탐지된 각 **spec** 및 **status** 필드에 대한 검증이 있는지 확인합니다.	**olm-crds-have-validation-test**

테스트	설명	짧은 이름
Owned CRDs Have Resources Listed	이 테스트는 **cr-manifest** 옵션을 통해 제공된 각 CR의 CRD에서 CSV(ClusterServiceVersion)의 **owned** CRD 섹션에 **resources** 하위 섹션이 있는지 확인합니다. 테스트에서 resources 섹션에 나열되지 않은 사용된 리소스를 탐지하면 테스트 종료 시 제안 사항에 해당 리소스를 나열합니다. 이 테스트를 통과하기 위해서는 사용자가 초기 코드 생성 후 resources 섹션을 작성해야 합니다.	**olm-crds-have-resources-test**
Spec Fields With Descriptors	이 테스트에서는 CR **spec** 섹션의 모든 필드에 CSV에 나열된 해당 설명자가 있는지 확인합니다.	**olm-spec-descriptors-test**
Status Fields With Descriptors	이 테스트에서는 CR **status** 섹션의 모든 필드에 CSV에 나열된 해당 설명자가 있는지 확인합니다.	**olm-status-descriptors-test**

5.10.4. 스코어 카드 툴 실행

*기본 **Kuryrstomize** 파일 세트는 **init** 명령을 실행한 후 **Operator SDK**에서 생성합니다. 생성된 기본 **bundle/tests/scorecard/config.yaml** 파일은 즉시 사용하여 **Operator**에 대해 스코어 카드 툴을 실행하거나 이 파일을 테스트 사양으로 수정할 수 있습니다.*

사전 요구 사항

- ***Operator SDK**를 사용하여 **Operator** 프로젝트 생성*

절차

1. ***Operator**에 대한 번들 매니페스트 및 메타데이터를 생성하거나 다시 생성합니다.*

```
$ make bundle
```

*이 명령을 수행하면 테스트를 실행하는 데 **scorecard** 명령에서 사용하는 번들 메타데이터에 스코어 카드 주석이 자동으로 추가됩니다.*

2. ***Operator** 번들에 대한 디스크상의 경로 또는 번들 이미지 이름에 대한 스코어 카드를 실행합니다.*

```
$ operator-sdk scorecard <bundle_dir_or_image>
```

5.10.5. 스코어 카드 출력

*scorecard 명령의 **--output** 플래그는 스코어 카드 결과 출력 형식을 **text** 또는 **json** 중 하나로 지정합니다.*

예 5.15. JSON 출력 조각의 예

```
{
  "apiVersion": "scorecard.operatorframework.io/v1alpha3",
  "kind": "TestList",
  "items": [
   {
     "kind": "Test",
     "apiVersion": "scorecard.operatorframework.io/v1alpha3",
     "spec": {
       "image": "quay.io/operator-framework/scorecard-test:v1.22.2",
       "entrypoint": [
         "scorecard-test",
         "olm-bundle-validation"
       ],
       "labels": {
         "suite": "olm",
         "test": "olm-bundle-validation-test"
       }
     },
     "status": {
       "results": [
         {
           "name": "olm-bundle-validation",
           "log": "time=\"2020-06-10T19:02:49Z\" level=debug msg=\"Found manifests
directory\" name=bundle-test\ntime=\"2020-06-10T19:02:49Z\" level=debug msg=\"Found
metadata directory\" name=bundle-test\ntime=\"2020-06-10T19:02:49Z\" level=debug
msg=\"Getting mediaType info from manifests directory\" name=bundle-test\ntime=\"2020-
06-10T19:02:49Z\" level=info msg=\"Found annotations file\" name=bundle-
test\ntime=\"2020-06-10T19:02:49Z\" level=info msg=\"Could not find optional
dependencies file\" name=bundle-test\n",
           "state": "pass"
         }
       ]
     }
   }
  ]
}
```

예 5.16. 텍스트 출력 조각의 예

```
--------------------------------------------------------------------------------
Image:      quay.io/operator-framework/scorecard-test:v1.22.2
Entrypoint: [scorecard-test olm-bundle-validation]
Labels:
 "suite":"olm"
 "test":"olm-bundle-validation-test"
Results:
 Name: olm-bundle-validation
```

```
State: pass
Log:
 time="2020-07-15T03:19:02Z" level=debug msg="Found manifests directory"
name=bundle-test
 time="2020-07-15T03:19:02Z" level=debug msg="Found metadata directory"
name=bundle-test
 time="2020-07-15T03:19:02Z" level=debug msg="Getting mediaType info from manifests
directory" name=bundle-test
 time="2020-07-15T03:19:02Z" level=info msg="Found annotations file" name=bundle-test
 time="2020-07-15T03:19:02Z" level=info msg="Could not find optional dependencies file"
name=bundle-test
```

참고

출력 형식의 사양은 Test 유형 레이아웃과 일치합니다.

5.10.6. 테스트 선택

스코어 카드 테스트는 **--selector CLI** 플래그를 일련의 라벨 문자열로 설정하여 선택합니다. 선택기 플래그를 지정하지 않으면 스코어 카드 구성 파일에 포함된 테스트가 모두 실행됩니다.

테스트는 순차적으로 실행되고 테스트 결과는 스코어 카드에 의해 집계되어 표준 출력 또는 **stdout**에 기록됩니다.

절차

1. 단일 테스트(예: **basic-check-spec-test**)를 선택하려면 **--selector** 플래그를 사용하여 테스트를 지정합니다.

```
$ operator-sdk scorecard <bundle_dir_or_image> \
    -o text \
    --selector=test=basic-check-spec-test
```

2. 테스트 도구 모음(예: **olm**)을 선택하려면 모든 **OLM** 테스트에서 사용하는 라벨을 지정합니다.

```
$ operator-sdk scorecard <bundle_dir_or_image> \
    -o text \
    --selector=suite=olm
```

3. 여러 개의 테스트를 선택하려면 다음 구문을 사용하여 **selector** 플래그로 테스트 이름을 지정합니다.

```
$ operator-sdk scorecard <bundle_dir_or_image> \
  -o text \
  --selector='test in (basic-check-spec-test,olm-bundle-validation-test)'
```

5.10.7. 병렬 테스트 활성화

Operator 작성자는 스코어 카드 구성 파일을 사용하여 테스트에 별도의 단계를 정의할 수 있습니다. 단계는 구성 파일에 정의된 순서에 따라 순차적으로 실행됩니다. 단계에는 테스트 목록과 구성 가능한 parallel 설정이 포함되어 있습니다.

기본적으로 또는 특정 단계에서 명시적으로 parallel을 false로 설정한 경우 단계의 테스트는 구성 파일에 정의된 순서에 따라 순차적으로 실행됩니다. 테스트를 한 번에 하나씩 실행하면 두 테스트가 서로 상호 작용하며 충돌하지 않도록 하는 데 유용합니다.

그러나 테스트를 완전히 격리하도록 설계하면 병렬화할 수 있습니다.

절차

- 격리된 테스트 세트를 병렬로 실행하려면 동일한 단계에 테스트 세트를 포함하고 parallel을 true로 설정합니다.

```
apiVersion: scorecard.operatorframework.io/v1alpha3
kind: Configuration
metadata:
  name: config
stages:
- parallel: true 1
  tests:
  - entrypoint:
    - scorecard-test
    - basic-check-spec
    image: quay.io/operator-framework/scorecard-test:v1.22.2
    labels:
      suite: basic
      test: basic-check-spec-test
  - entrypoint:
    - scorecard-test
    - olm-bundle-validation
    image: quay.io/operator-framework/scorecard-test:v1.22.2
    labels:
      suite: olm
      test: olm-bundle-validation-test
```

1

병렬 테스트 사용

병렬 단계의 테스트는 모두 동시에 실행되고 스코어 카드는 테스트가 모두 완료될 때까지 기다린 후 다음 단계를 진행합니다. 이 경우 테스트가 훨씬 빨라질 수 있습니다.

5.10.8. 사용자 정의 스코어 카드 테스트

스코어 카드 툴에서는 다음과 같은 필수 규칙을 따르는 사용자 정의 테스트를 실행할 수 있습니다.

- 테스트를 컨테이너 이미지 내에서 구현함
- 테스트에서 명령 및 인수를 포함하는 진입점 허용
- 테스트에서 테스트 출력과 관련 없는 로그를 기록하지 않고 **JSON** 형식으로 **v1alpha3** 스코어 카드 출력 생성
- 테스트에서 **/bundle**의 공유 마운트 옵션에 있는 번들 콘텐츠를 가져올 수 있음
- 테스트에서 클러스터 내 클라이언트 연결을 사용하여 **Kubernetes API**에 액세스할 수 있음

테스트 이미지가 위 지침을 따르는 경우 다른 프로그래밍 언어로 사용자 정의 테스트를 작성할 수 있습니다.

다음 예제는 **Go**에서 작성한 사용자 정의 테스트 이미지입니다.

예 5.17. 사용자 정의 스코어 카드 테스트 예제

```
// Copyright 2020 The Operator-SDK Authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
```

```
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
 "encoding/json"
 "fmt"
 "log"
 "os"

 scapiv1alpha3 "github.com/operator-framework/api/pkg/apis/scorecard/v1alpha3"
 apimanifests "github.com/operator-framework/api/pkg/manifests"
)

// This is the custom scorecard test example binary
// As with the Redhat scorecard test image, the bundle that is under
// test is expected to be mounted so that tests can inspect the
// bundle contents as part of their test implementations.
// The actual test is to be run is named and that name is passed
// as an argument to this binary.  This argument mechanism allows
// this binary to run various tests all from within a single
// test image.

const PodBundleRoot = "/bundle"

func main() {
 entrypoint := os.Args[1:]
 if len(entrypoint) == 0 {
  log.Fatal("Test name argument is required")
 }

 // Read the pod's untar'd bundle from a well-known path.
 cfg, err := apimanifests.GetBundleFromDir(PodBundleRoot)
 if err != nil {
  log.Fatal(err.Error())
 }

 var result scapiv1alpha3.TestStatus

 // Names of the custom tests which would be passed in the
 // `operator-sdk` command.
 switch entrypoint[0] {
 case CustomTest1Name:
  result = CustomTest1(cfg)
 case CustomTest2Name:
  result = CustomTest2(cfg)
 default:
  result = printValidTests()
 }

 // Convert scapiv1alpha3.TestResult to json.
 prettyJSON, err := json.MarshalIndent(result, "", "    ")
```

```
 if err != nil {
  log.Fatal("Failed to generate json", err)
 }
 fmt.Printf("%s\n", string(prettyJSON))

}

// printValidTests will print out full list of test names to give a hint to the end user on what
the valid tests are.
func printValidTests() scapiv1alpha3.TestStatus {
 result := scapiv1alpha3.TestResult{}
 result.State = scapiv1alpha3.FailState
 result.Errors = make([]string, 0)
 result.Suggestions = make([]string, 0)

 str := fmt.Sprintf("Valid tests for this image include: %s %s",
  CustomTest1Name,
  CustomTest2Name)
 result.Errors = append(result.Errors, str)
 return scapiv1alpha3.TestStatus{
  Results: []scapiv1alpha3.TestResult{result},
 }
}

const (
 CustomTest1Name = "customtest1"
 CustomTest2Name = "customtest2"
)

// Define any operator specific custom tests here.
// CustomTest1 and CustomTest2 are example test functions. Relevant operator specific
// test logic is to be implemented in similarly.

func CustomTest1(bundle *apimanifests.Bundle) scapiv1alpha3.TestStatus {
 r := scapiv1alpha3.TestResult{}
 r.Name = CustomTest1Name
 r.State = scapiv1alpha3.PassState
 r.Errors = make([]string, 0)
 r.Suggestions = make([]string, 0)
 almExamples := bundle.CSV.GetAnnotations()["alm-examples"]
 if almExamples == "" {
  fmt.Println("no alm-examples in the bundle CSV")
 }

 return wrapResult(r)
}

func CustomTest2(bundle *apimanifests.Bundle) scapiv1alpha3.TestStatus {
 r := scapiv1alpha3.TestResult{}
 r.Name = CustomTest2Name
 r.State = scapiv1alpha3.PassState
 r.Errors = make([]string, 0)
 r.Suggestions = make([]string, 0)
 almExamples := bundle.CSV.GetAnnotations()["alm-examples"]
 if almExamples == "" {
  fmt.Println("no alm-examples in the bundle CSV")
```

```
 }
 return wrapResult(r)
}

func wrapResult(r scapiv1alpha3.TestResult) scapiv1alpha3.TestStatus {
 return scapiv1alpha3.TestStatus{
  Results: []scapiv1alpha3.TestResult{r},
 }
}
```

5.11. OPERATOR 번들 검증

Operator 작성자는 Operator SDK에서 bundle validate 명령을 실행하여 Operator 번들의 콘텐츠 및 형식을 검증할 수 있습니다. 원격 Operator 번들 이미지 또는 로컬 Operator 번들 디렉터리에서 명령을 실행할 수 있습니다.

5.11.1. bundle validate 명령 정보

Operator SDK scorecard 명령은 구성 파일 및 테스트 이미지를 기반으로 Operator에서 테스트를 실행할 수 있지만 bundle validate 하위 명령은 콘텐츠 및 구조에 대해 로컬 번들 디렉터리 및 원격 번들 이미지를 검증할 수 있습니다.

bundle validate 명령 구문

```
$ operator-sdk bundle validate <bundle_dir_or_image> <flags>
```

참고

bundle validate 명령은 make bundle 명령을 사용하여 번들을 빌드할 때 자동으로 실행됩니다.

번들 이미지는 원격 레지스트리에서 가져와서 검증되기 전에 로컬로 빌드됩니다. 로컬 번들 디렉터리에는 Operator 메타데이터 및 매니페스트가 포함되어야 합니다. 번들 메타데이터 및 매니페스트에는 다음 번들 레이아웃과 유사한 구조가 있어야 합니다.

번들 레이아웃의 예

```
./bundle
├── manifests
│   ├── cache.my.domain_memcacheds.yaml
│   └── memcached-operator.clusterserviceversion.yaml
└── metadata
    └── annotations.yaml
```

번들 테스트는 오류를 탐지하지 않는 경우 유효성 검사를 통과하고 종료 코드 **0** 을 사용하여 완료합니다.

출력 예

```
INFO[0000] All validation tests have completed successfully
```

테스트가 실패하고 오류가 감지된 경우 종료 코드 **1** 을 사용하여 완료합니다.

출력 예

```
ERRO[0000] Error: Value cache.example.com/v1alpha1, Kind=Memcached: CRD
"cache.example.com/v1alpha1, Kind=Memcached" is present in bundle "" but not defined in
CSV
```

경고가 감지된 번들 테스트는 오류가 발견되지 않은 경우 종료 코드 **0** 으로 유효성 검사를 전달할 수 있습니다. 테스트는 오류에서만 실패합니다.

출력 예

```
WARN[0000] Warning: Value : (memcached-operator.v0.0.1) annotations not found
INFO[0000] All validation tests have completed successfully
```

bundle validate 하위 명령에 대한 자세한 내용을 보려면 다음을 실행합니다.

```
$ operator-sdk bundle validate -h
```

5.11.2. 기본 제공되는 bundle validate 테스트

Operator SDK는 제품군으로 정렬된 사전 정의된 검증기와 함께 제공됩니다. 검증기를 지정하지 않고 bundle validate 명령을 실행하면 기본 테스트가 실행됩니다. 기본 테스트에서는 번들이 Operator 프레임워크 커뮤니티에서 정의한 사양을 준수하는지 확인합니다. 자세한 내용은 "Bundle 형식"을 참조하십시오.

선택적 검증기를 실행하여 OperatorHub 호환성 또는 더 이상 사용되지 않는 Kubernetes API와 같은 문제를 테스트할 수 있습니다. 선택적 검증기는 항상 기본 테스트 외에 실행됩니다.

선택적 테스트 모음의 bundle validate 명령 구문

```
$ operator-sdk bundle validate <bundle_dir_or_image>
  --select-optional <test_label>
```

표 5.18. addtional bundle validate validators

이름	설명	레이블
Operator 프레임워크	이 검증기에서는 Operator 프레임워크에서 제공하는 전체 검증기 모음에 대해 Operator 번들을 테스트합니다.	**suite=operatorframework**
OperatorHub	이 검증기에서는 OperatorHub와의 호환성을 위해 Operator 번들을 테스트합니다.	**name=operatorhub**
모범 사례	이 검증기를 통해 Operator 번들이 Operator 프레임워크에서 정의한 모범 사례를 준수하는지 테스트합니다. 빈 CRD 설명 또는 지원되지 않는 OLM(Operator Lifecycle Manager) 리소스와 같은 문제를 확인합니다.	**name=good-practices**

추가 리소스

- 번들 형식

5.11.3. bundle validate 명령 실행

기본 검증기에서는 **bundle validate** 명령을 입력할 때마다 테스트를 실행합니다. **--select-optional** 플래그를 사용하여 선택적 검증기를 실행할 수 있습니다. 선택적 검증기에서는 기본 테스트 외에 테스트를 실행합니다.

사전 요구 사항

- Operator SDK를 사용하여 Operator 프로젝트 생성

절차

1. 로컬 번들 디렉터리에 대해 기본 검증기를 실행하려면 **Operator** 프로젝트 디렉터리에서 다음 명령을 입력합니다.

```
$ operator-sdk bundle validate ./bundle
```

2. 원격 **Operator** 번들 이미지에 대해 기본 검증기를 실행하려면 다음 명령을 입력합니다.

```
$ operator-sdk bundle validate \
  <bundle_registry>/<bundle_image_name>:<tag>
```

다음과 같습니다.

<bundle_registry>

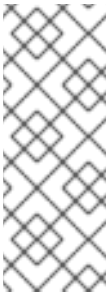
quay.io/example 과 같이 번들이 호스팅되는 레지스트리를 지정합니다.

<bundle_image_name>

memcached-operator 와 같은 번들 이미지의 이름을 지정합니다.

<tag>

번들 이미지의 태그(예: **v1.22.2**)를 지정합니다.



참고

Operator 번들 이미지를 검증하려면 원격 레지스트리에서 이미지를 호스팅해야 합니다. Operator SDK는 이미지를 가져와서 테스트를 실행하기 전에 로컬로 빌드합니다. bundle validate 명령은 로컬 번들 이미지 테스트를 지원하지 않습니다.

3. Operator 번들에 대해 추가 검증기를 실행하려면 다음 명령을 입력합니다.

```
$ operator-sdk bundle validate \
  <bundle_dir_or_image> \
  --select-optional <test_label>
```

다음과 같습니다.

<bundle_dir_or_image>

로컬 번들 디렉터리 또는 원격 번들 이미지를 지정합니다(예: ~/projects/memcached 또는 quay.io/example/memcached-operator:v1.22).

<test_label>

name=good-practices 와 같이 실행할 검증기의 이름을 지정합니다.

출력 예

```
ERRO[0000] Error: Value apiextensions.k8s.io/v1, Kind=CustomResource:
unsupported media type registry+v1 for bundle object
WARN[0000] Warning: Value k8sevent.v0.0.1: owned CRD
"k8sevents.k8s.k8sevent.com" has an empty description
```

5.12. 고가용성 또는 단일 노드 클러스터 감지 및 지원

OpenShift Container Platform 클러스터는 여러 노드를 사용하는 HA(고가용성) 모드로 구성하거나 단일 노드를 사용하는 비-HA 모드로 구성할 수 있습니다. 단일 노드 OpenShift라고도 하는 단일 노드 클

러스터에는 보다 보수적인 리소스 제약 조건이 있을 수 있습니다. 따라서 단일 노드 클러스터에 설치된 Operator를 적절하게 조정하고 계속 실행할 수 있어야 합니다.

OpenShift Container Platform에 제공된 클러스터 고가용성 모드 API에 액세스하여 Operator 작성자는 Operator SDK를 사용하여 Operator가 HA 또는 비HA 모드 중 하나의 클러스터 인프라 토폴로지를 감지할 수 있습니다. 감지된 클러스터 토폴로지를 사용하여 Operator 및 관리하는 Operand 또는 워크로드 모두에 대해 리소스 요구 사항을 토폴로지에 가장 적합한 프로필로 자동 전환하는 사용자 정의 Operator 논리를 개발할 수 있습니다.

5.12.1. 클러스터 고가용성 모드 API 정보

OpenShift Container Platform은 Operator에서 인프라 토폴로지를 감지하는 데 사용할 수 있는 클러스터 고가용성 모드 API를 제공합니다. 인프라 API는 인프라와 관련된 클러스터 전체 정보를 보유합니다. OLM(Operator Lifecycle Manager)에서 관리하는 Operator는 고가용성 모드를 기반으로 Operand 또는 관리되는 워크로드를 다르게 구성해야 하는 경우 인프라 API를 사용할 수 있습니다.

인프라 API에서 infrastructureTopology 상태는 컨트롤 플레인 노드에서 실행되지 않는 인프라 서비스의 기대치를 표현하며, 일반적으로 master 이외의 role 값에 대한 노드 선택기로 표시됩니다. controlPlaneTopology 상태는 일반적으로 컨트롤 플레인 노드에서 실행되는 Operand에 대한 기대치를 나타냅니다.

두 상태의 기본 설정은 HighlyAvailable 로, Operator가 여러 노드 클러스터에 있는 동작을 나타냅니다. SingleReplica 설정은 단일 노드 OpenShift라고도 하는 단일 노드 클러스터에서 사용되며 Operator는 고가용성 작업을 위해 Operand를 구성해서는 안 함을 나타냅니다.

OpenShift Container Platform 설치 프로그램은 다음 규칙에 따라 생성 시 클러스터의 복제본 수를 기반으로 controlPlaneTopology 및 infrastructureTopology 상태 필드를 설정합니다.

- 컨트롤 플레인 복제본 수가 3 미만이면 controlPlaneTopology 상태가 SingleReplica로 설정됩니다. 그렇지 않으면 HighlyAvailable로 설정됩니다.
- 작업자 복제본 수가 0이면 컨트롤 플레인 노드도 작업자로 구성됩니다. 따라서 infrastructureTopology 상태는 controlPlaneTopology 상태와 동일합니다.
- 작업자 복제본 수가 1이면 infrastructureTopology가 SingleReplica로 설정됩니다. 그렇지 않으면 HighlyAvailable로 설정됩니다.

5.12.2. Operator 프로젝트의 API 사용 예

Operator 작성자는 다음 예와 같이 일반적인 Kubernetes 구문 및 controller-runtime 라이브러리를 사용하여 Operator 프로젝트를 업데이트하여 Infrastructure API에 액세스할 수 있습니다.

controller-runtime 라이브러리 예

```
// Simple query
 nn := types.NamespacedName{
 Name: "cluster",
 }
 infraConfig := &configv1.Infrastructure{}
 err = crClient.Get(context.Background(), nn, infraConfig)
 if err != nil {
 return err
 }
 fmt.Printf("using crclient: %v\n", infraConfig.Status.ControlPlaneTopology)
 fmt.Printf("using crclient: %v\n", infraConfig.Status.InfrastructureTopology)
```

Kubernetes 생성 예

```
operatorConfigInformer :=
configinformer.NewSharedInformerFactoryWithOptions(configClient, 2*time.Second)
 infrastructureLister = operatorConfigInformer.Config().V1().Infrastructures().Lister()
 infraConfig, err := configClient.ConfigV1().Infrastructures().Get(context.Background(),
"cluster", metav1.GetOptions{})
 if err != nil {
 return err
 }
// fmt.Printf("%v\n", infraConfig)
 fmt.Printf("%v\n", infraConfig.Status.ControlPlaneTopology)
 fmt.Printf("%v\n", infraConfig.Status.InfrastructureTopology)
```

5.13. PROMETHEUS를 사용하여 기본 제공 모니터링 구성

이 가이드에서는 Prometheus Operator를 사용하여 Operator SDK에서 제공하는 기본 제공 모니터링 지원과 Go 기반 및 Ansible 기반 Operator 작성자의 세부 정보를 설명합니다.

5.13.1. Prometheus Operator 지원

Prometheus는 오픈 소스 시스템 모니터링 및 경고 툴킷입니다. Prometheus Operator는 OpenShift Container Platform과 같은 Kubernetes 기반 클러스터에서 실행되는 Prometheus 클러스터를 생성, 구성, 관리합니다.

Helper 함수는 기본적으로 Operator SDK에 있으며 Prometheus Operator가 배포된 클러스터에서 사용하기 위해 생성한 Go 기반 Operator의 지표를 자동으로 설정합니다.

5.13.2. Go 기반 Operator에 대한 사용자 정의 지표 노출

Operator 작성자는 controller-runtime/pkg/metrics 라이브러리의 글로벌 Prometheus 레지스트리를 사용하여 사용자 지정 지표를 게시할 수 있습니다.

사전 요구 사항

- Operator SDK를 사용하여 Go 기반 Operator가 생성됨
- OpenShift Container Platform 클러스터에 기본적으로 배포된 Prometheus Operator

절차

1. Operator SDK 프로젝트에서 config/default/kustomization.yaml 파일에서 다음 행의 주석을 제거합니다.

```
../prometheus
```

2. Operator에서 추가 지표를 게시하는 사용자 정의 컨트롤러 클래스를 생성합니다. 다음 예제에서는 위젯 및 위젯Failures 수집기를 전역 변수로 선언한 다음 컨트롤러의 패키지에 init() 함수에 등록합니다.

예 5.18. controllers/memcached_controller_test_metrics.go file

```
package controllers

import (
 "github.com/prometheus/client_golang/prometheus"
 "sigs.k8s.io/controller-runtime/pkg/metrics"
)


var (
    widgets = prometheus.NewCounter(
```

```
        prometheus.CounterOpts{
            Name: "widgets_total",
            Help: "Number of widgets processed",
        },
    )
    widgetFailures = prometheus.NewCounter(
        prometheus.CounterOpts{
            Name: "widget_failures_total",
            Help: "Number of failed widgets",
        },
    )
)

func init() {
    // Register custom metrics with the global prometheus registry
    metrics.Registry.MustRegister(widgets, widgetFailures)
}
```

3. *지표에 대한 비즈니스 로직을 결정하는 기본 컨트롤러 클래스의 조정 루프에서 이러한 수집기를 기록합니다.*

예 5.19. controllers/memcached_controller.go file

```
func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error) {
 ...
 ...
 // Add metrics
 widgets.Inc()
 widgetFailures.Inc()

 return ctrl.Result{}, nil
}
```

4. *Operator를 빌드하고 내보냅니다.*

```
$ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
```

5. *Operator를 배포합니다.*

```
$ make deploy IMG=<registry>/<user>/<image_name>:<tag>
```

6. *OpenShift Container Platform 클러스터의 Prometheus 인스턴스에서 Operator의 서비스 모니터를 스크랩할 수 있도록 역할 및 역할 바인딩 정의를 생성합니다.*

서비스 계정에 네임스페이스 메트릭을 스크랩할 수 있는 권한이 있도록 역할을 할당해야 합니다.

예 5.20. config/prometheus/role.yaml role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-k8s-role
  namespace: <operator_namespace>
rules:
  - apiGroups:
      - ""
    resources:
      - endpoints
      - pods
      - services
      - nodes
      - secrets
    verbs:
      - get
      - list
      - watch
```

예 5.21. config/prometheus/rolebinding.yaml role binding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus-k8s-rolebinding
  namespace: memcached-operator-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus-k8s-role
subjects:
  - kind: ServiceAccount
    name: prometheus-k8s
    namespace: openshift-monitoring
```

7. 배포된 Operator에 대한 역할 및 역할 바인딩을 적용합니다.

```
$ oc apply -f config/prometheus/role.yaml
```

```
$ oc apply -f config/prometheus/rolebinding.yaml
```

8. 스크랩할 네임스페이스의 라벨을 설정하여 OpenShift 클러스터 모니터링이 가능합니다.

```
$ oc label namespace <operator_namespace> openshift.io/cluster-monitoring="true"
```

검증

- OpenShift Container Platform 웹 콘솔에서 메트릭을 쿼리하고 확인합니다. 사용자 정의 컨트롤러 클래스에 설정된 이름(예: widgets_total 및 widget_failures_total)을 사용할 수 있습니다.

5.13.3. Ansible 기반 Operator에 대한 사용자 정의 지표 노출

Operator 작성자는 Ansible 기반 Operator를 생성하는 경우 Operator SDK의 osdk_metrics 모듈을 사용하여 사용자 정의 Operator 및 Operand 메트릭, 출력 이벤트, 로깅을 지원할 수 있습니다.

사전 요구 사항

- Operator SDK를 사용하여 Ansible 기반 Operator 생성
- OpenShift Container Platform 클러스터에 기본적으로 배포된 Prometheus Operator

절차

1. Ansible 기반 Operator를 생성합니다. 이 예제에서는 testmetrics.com 도메인을 사용합니다.

```
$ operator-sdk init \
  --plugins=ansible \
  --domain=testmetrics.com
```

2. 지표 API를 생성합니다. 이 예에서는 Testmetrics 라는 유형을 사용합니다.

```
$ operator-sdk create api \
  --group metrics \
  --version v1 \
  --kind Testmetrics \
  --generate-role
```

3. roles/testmetrics/tasks/main.yml 파일을 편집하고 osdk_metrics 모듈을 사용하여

Operator 프로젝트에 대한 사용자 정의 지표를 생성합니다.

예 5.22. roles/testmetrics/tasks/main.yml 파일의 예

```
---
# tasks file for Memcached
- name: start k8sstatus
  k8s:
    definition:
      kind: Deployment
      apiVersion: apps/v1
      metadata:
        name: '{{ ansible_operator_meta.name }}-memcached'
        namespace: '{{ ansible_operator_meta.namespace }}'
      spec:
        replicas: "{{size}}"
        selector:
          matchLabels:
            app: memcached
        template:
          metadata:
            labels:
              app: memcached
          spec:
            containers:
            - name: memcached
              command:
              - memcached
              - -m=64
              - -o
              - modern
              - -v
              image: "docker.io/memcached:1.4.36-alpine"
              ports:
                - containerPort: 11211

- osdk_metric:
    name: my_thing_counter
    description: This metric counts things
    counter: {}

- osdk_metric:
    name: my_counter_metric
    description: Add 3.14 to the counter
    counter:
      increment: yes

- osdk_metric:
    name: my_gauge_metric
    description: Create my gauge and set it to 2.
    gauge:
      set: 2

- osdk_metric:
    name: my_histogram_metric
    description: Observe my histogram
```

```
    histogram:
      observe: 2

  - osdk_metric:
      name: my_summary_metric
      description: Observe my summary
      summary:
        observe: 2
```

검증

1. *클러스터에서 **Operator**를 실행합니다. 예를 들어 **"run as a deployment"** 메서드를 사용하려면 다음을 수행합니다.*

   a. *Operator 이미지를 빌드하고 레지스트리로 내보냅니다.*

   ```
   $ make docker-build docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```

   b. *클러스터에 **Operator**를 설치합니다.*

   ```
   $ make install
   ```

   c. *Operator를 배포합니다.*

   ```
   $ make deploy IMG=<registry>/<user>/<image_name>:<tag>
   ```

2. *Testmetrics CR(사용자 정의 리소스)을 생성합니다.*

   a. *CR 사양을 정의합니다.*

   예 5.23. config/samples/metrics_v1_testmetrics.yaml 파일의 예

   ```
   apiVersion: metrics.testmetrics.com/v1
   kind: Testmetrics
   metadata:
     name: testmetrics-sample
   spec:
     size: 1
   ```

.

b. *오브젝트를 생성합니다.*

```
$ oc create -f config/samples/metrics_v1_testmetrics.yaml
```

3. *Pod 세부 정보를 가져옵니다.*

```
$ oc get pods
```

출력 예

```
NAME                                          READY   STATUS    RESTARTS   AGE
ansiblemetrics-controller-manager-<id>  2/2     Running   0          149m
testmetrics-sample-memcached-<id>       1/1     Running   0          147m
```

4. *끝점 세부 정보를 가져옵니다.*

```
$ oc get ep
```

출력 예

```
NAME                                             ENDPOINTS          AGE
ansiblemetrics-controller-manager-metrics-service   10.129.2.70:8443   150m
```

5. *사용자 정의 메트릭 토큰을 요청합니다.*

```
$ token=`oc create token prometheus-k8s -n openshift-monitoring`
```

6. *메트릭 값을 확인합니다.*

a.

my_counter_metric 값을 확인합니다.

```
$ oc exec ansiblemetrics-controller-manager-<id> -- curl -k -H "Authoriza
tion: Bearer $token" 'https://10.129.2.70:8443/metrics' | grep  my_counter
```

출력 예

```
HELP my_counter_metric Add 3.14 to the counter
TYPE my_counter_metric counter
my_counter_metric 2
```

b.

my_gauge_metric 값을 확인합니다.

```
$ oc exec ansiblemetrics-controller-manager-<id> -- curl -k -H "Authoriza
tion: Bearer $token" 'https://10.129.2.70:8443/metrics' | grep  gauge
```

출력 예

```
HELP my_gauge_metric Create my gauge and set it to 2.
```

c.

my_histogram_metric 및 my_summary_metric 값을 확인합니다.

```
$ oc exec ansiblemetrics-controller-manager-<id> -- curl -k -H "Authoriza
tion: Bearer $token" 'https://10.129.2.70:8443/metrics' | grep  Observe
```

출력 예

```
HELP my_histogram_metric Observe my histogram
HELP my_summary_metric Observe my summary
```

5.14. 리더 선택 방식 구성

Operator의 라이프사이클 동안, 예를 들어 Operator에 대한 업그레이드를 롤아웃할 때 언제든지 여러 개의 인스턴스를 실행할 수 있습니다. 이러한 시나리오에서는 리더를 선택하여 여러 Operator 인스턴스 간 경합을 방지해야 합니다. 그러면 하나의 리더 인스턴스에서만 조정을 처리하고 다른 인스턴스는 비활성화되지만 리더가 아래로 내려가면 이어서 작업을 수행할 수 있습니다.

두 가지 리더 선택 구현 방법 중 선택할 수 있으며 각각 고유한 장단점이 있습니다.

Leader-for-life

리더 Pod는 삭제되는 경우에만 가비지 컬렉션을 사용하여 리더십을 포기합니다. 이 구현에서는 실수로 두 개의 인스턴스가 리더로 실행될 가능성을 방지합니다. 이러한 상태를 스플릿 브레인이라고 합니다. 그러나 이 방법을 사용하면 새 리더 선택이 지연될 수 있습니다. 예를 들어 리더 Pod가 응답하지 않거나 분할된 노드에 있는 경우 pod-eviction-timeout은 리더 Pod가 노드에서 삭제되고 아래로 내려가는 데 걸리는 시간을 지정하는데 기본값은 5m입니다. 자세한 내용은 leader-for-life Go 설명서를 참조하십시오.

Leader-with-lease

리더 Pod는 주기적으로 리더 리스를 갱신하고 리스를 갱신할 수 없는 경우 리더십을 포기합니다. 이 구현에서는 기존 리더가 격리되었을 때 새 리더로 더 빠르게 전환할 수 있지만 특정 상황에서 스플릿 브레인이 발생할 가능성이 있습니다. 자세한 내용은 Leader-with-lease Go 설명서를 참조하십시오.

Operator SDK에서는 기본적으로 Leader-for-life 구현을 사용합니다. 두 가지 접근 방식에 대한 관련 Go 설명서를 참조하여 사용 사례에 적합한 장단점을 고려하도록 합니다.

5.14.1. Operator 리더 선택 예

다음 예제에서는 Operator, Leader-for-life 및 Leader-with-lease에 두 명의 리더 선택 옵션을 사용하는 방법을 보여줍니다.

5.14.1.1. Leader-for-life 선택

Leader-for-life 선택 방식을 구현하면 leader.Become() 호출 시 memcached-operator-lock이라는 구성 맵을 생성하여 Operator에서 리더가 될 때까지 재시도하지 못하도록 합니다.

```
import (
  ...
  "github.com/operator-framework/operator-sdk/pkg/leader"
)

func main() {
  ...
  err = leader.Become(context.TODO(), "memcached-operator-lock")
  if err != nil {
    log.Error(err, "Failed to retry for leader lock")
    os.Exit(1)
  }
  ...
}
```

Operator가 클러스터 내에서 실행되지 않는 경우 leader.Become()은 Operator의 이름을 탐지할 수 없기 때문에 리더 선택을 건너뛰기 위해 오류 없이 단순히 반환됩니다.

5.14.1.2. Leader-with-lease 선택

리더 선택을 위해 Manager 옵션을 사용하여 Leader-with-lease 구현을 활성화할 수 있습니다.

```
import (
  ...
  "sigs.k8s.io/controller-runtime/pkg/manager"
)

func main() {
  ...
  opts := manager.Options{
    ...
    LeaderElection: true,
    LeaderElectionID: "memcached-operator-lock"
  }
  mgr, err := manager.New(cfg, opts)
  ...
}
```

Operator가 클러스터에서 실행되지 않으면 Manager에서 리더 선택을 위한 구성 맵을 생성하기 위해 Operator의 네임스페이스를 탐지할 수 없기 때문에 시작 시 오류를 반환합니다. Manager에 LeaderElectionNamespace 옵션을 설정하여 이 네임스페이스를 덮어쓸 수 있습니다.

5.15. GO 기반 OPERATOR의 오브젝트 정리 유틸리티

operator-lib 정리 유틸리티를 사용하면 Go 기반 Operator가 더 이상 필요하지 않은 경우 오브젝트를 정리하거나 정리할 수 있습니다. Operator 작성자는 유틸리티를 사용하여 사용자 정의 후크 및 전략을 생

성할 수도 있습니다.

5.15.1. operator-lib 정리 유틸리티 정보

작업 또는 Pod와 같은 오브젝트는 Operator 라이프 사이클의 정상적인 부분으로 생성됩니다. 클러스터 관리자 또는 Operator에서 이러한 오브젝트를 제거하지 않으면 클러스터에 남아 있고 리소스를 사용할 수 있습니다.

이전에는 불필요한 오브젝트를 정리할 때 다음 옵션을 사용할 수 있었습니다.

- Operator 작성자는 Operator를 위한 고유한 정리 솔루션을 생성해야 했습니다.
- 클러스터 관리자는 자체적으로 오브젝트를 정리해야 했습니다.

operator-lib 정리 유틸리티는 지정된 네임스페이스의 Kubernetes 클러스터에서 오브젝트를 제거합니다. 라이브러리는 Operator 프레임워크의 일부로 operator-lib 라이브러리 의 버전 0.9.0 에 추가되었습니다.

5.15.2. 유틸리티 구성 정리

operator-lib 정리 유틸리티는 Go로 작성되었으며 Go 기반 Operator를 위한 공통 정리 전략을 포함합니다.

설정 예

```
cfg = Config{
        log:          logf.Log.WithName("prune"),
        DryRun:       false,
        Clientset:    client,
        LabelSelector: "app=<operator_name>",
        Resources: []schema.GroupVersionKind{
                {Group: "", Version: "", Kind: PodKind},
        },
        Namespaces: []string{"default"},
        Strategy: StrategyConfig{
                Mode:          MaxCountStrategy,
                MaxCountSetting: 1,
```

```
        },
        PreDeleteHook: myhook,
}
```

정리 유틸리티 구성 파일은 다음 필드를 사용하여 정리 작업을 정의합니다.

구성 필드	Description
log	라이브러리 로그 메시지를 처리하는 데 사용되는 로거입니다.
DryRun	리소스를 제거해야 하는지 여부를 결정하는 부울입니다. **true** 로 설정하면 유틸리티가 실행되지만 리소스를 제거하지는 않습니다.
Clientset	Kubernetes API 호출에 사용되는 클라이언트-go Kubernetes ClientSet입니다.
LabelSelector	정리할 리소스를 찾는 데 사용되는 Kubernetes 라벨 선택기 표현식입니다.
리소스	Kubernetes 리소스 종류. 현재 **PodKind** 및 **JobKind** 가 지원됩니다.
네임스페이스	리소스를 검색할 Kubernetes 네임스페이스 목록입니다.
전략	실행할 정리 전략입니다.
strategy.Mode	**MaxCountStrategy**,**MaxAgeStrategy** 또는 **CustomStrategy** 가 현재 지원됩니다.
Strategy.MaxCountSetting	정리 유틸리티 실행 후에도 유지해야 하는 리소스 수를 지정하는 **MaxCountStrategy** 의 정수 값입니다.
Strategy.MaxAgeSetting	go **time.Duration** 문자열 값(예:**48h**)은 정리할 리소스의 기간을 지정합니다.
Strategy.CustomSettings	사용자 정의 전략 함수에 전달할 수 있는 값의 맵을 이동합니다.
PreDeleteHook	선택 사항: 리소스를 정리하기 전에 호출할 함수를 찾습니다.
customStrategy	선택 사항: 사용자 정의 정리 전략을 구현하는 Go 함수

실행 정리

*정리 구성에서 **execute** 함수를 실행하여 정리 작업을 호출할 수 있습니다.*

```
err := cfg.Execute(ctx)
```

cron 패키지를 사용하거나 트리거 이벤트와 함께 정리 유틸리티를 호출하여 정리 작업을 호출할 수도 있습니다.

5.16. 번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션

Operator에 대한 레거시 패키지 매니페스트 형식 지원은 OpenShift Container Platform 4.8 이상에서 제거됩니다. 패키지 매니페스트 형식을 사용하여 처음 생성된 Operator 프로젝트가 있는 경우 Operator SDK를 사용하여 프로젝트를 번들 형식으로 마이그레이션할 수 있습니다. 번들 형식은 OpenShift Container Platform 4.6부터 OLM(Operator Lifecycle Manager)의 기본 패키징 형식입니다.

5.16.1. 패키징 형식 마이그레이션 정보

Operator SDK pkgman-to-bundle 명령은 OLM(Operator Lifecycle Manager) 패키지 매니페스트를 번들로 마이그레이션하는 데 도움이 됩니다. 명령은 입력 패키지 매니페스트 디렉터리를 사용하고 입력 디렉터리에 있는 각 매니페스트 버전에 대한 번들을 생성합니다. 그런 다음 생성된 각 번들에 대한 번들 이미지를 빌드할 수도 있습니다.

예를 들어 패키지 매니페스트 형식의 프로젝트에 대해 다음 packagemanifests/ 디렉터리를 고려하십시오.

패키지 매니페스트 형식 레이아웃 예제

```
packagemanifests/
└── etcd
    ├── 0.0.1
    │   ├── etcdcluster.crd.yaml
    │   └── etcdoperator.clusterserviceversion.yaml
    ├── 0.0.2
    │   ├── etcdbackup.crd.yaml
    │   ├── etcdcluster.crd.yaml
    │   ├── etcdoperator.v0.0.2.clusterserviceversion.yaml
    │   └── etcdrestore.crd.yaml
    └── etcd.package.yaml
```

마이그레이션을 실행하면 bundle/ 디렉터리에 다음 번들이 생성됩니다.

Bundle 형식 레이아웃의 예

```
bundle/
├── bundle-0.0.1
│   ├── bundle.Dockerfile
│   ├── manifests
│   │   ├── etcdcluster.crd.yaml
│   │   └── etcdoperator.clusterserviceversion.yaml
│   ├── metadata
│   │   └── annotations.yaml
│   └── tests
│       └── scorecard
│           └── config.yaml
└── bundle-0.0.2
    ├── bundle.Dockerfile
    ├── manifests
    │   ├── etcdbackup.crd.yaml
    │   ├── etcdcluster.crd.yaml
    │   ├── etcdoperator.v0.0.2.clusterserviceversion.yaml
    │   └── etcdrestore.crd.yaml
    ├── metadata
    │   └── annotations.yaml
    └── tests
        └── scorecard
            └── config.yaml
```

생성된 이 레이아웃을 기반으로 두 번들의 번들 이미지도 다음 이름으로 빌드됩니다.

- quay.io/example/etcd:0.0.1
- quay.io/example/etcd:0.0.2

추가 리소스

- Operator 프레임워크 패키지 형식

5.16.2. 번들 형식으로 패키지 매니페스트 프로젝트 마이그레이션

Operator 작성자는 Operator SDK를 사용하여 패키지 매니페스트 형식 Operator 프로젝트를 번들 형

식 프로젝트로 마이그레이션할 수 있습니다.

사전 요구 사항

- Operator SDK CLI가 설치됨
- 패키지 매니페스트 형식의 Operator SDK를 사용하여 처음에 Operator 프로젝트 생성

절차

- Operator SDK를 사용하여 패키지 매니페스트 프로젝트를 번들 형식으로 마이그레이션하고 번들 이미지를 생성합니다.

```
$ operator-sdk pkgman-to-bundle <package_manifests_dir> \ 1
  [--output-dir <directory>] \ 2
  --image-tag-base <image_name_base> 3
```

1. 프로젝트의 패키지 매니페스트 디렉터리(예: packagemanifests/ 또는 manifests/)의 위치를 지정합니다.
2. 선택 사항: 기본적으로 생성된 번들은 bundle/ 디렉터리에 디스크에 로컬로 작성됩니다. --output-dir 플래그를 사용하여 대체 위치를 지정할 수 있습니다.
3. 번들에 사용할 quay.io/example/etcd와 같은 이미지 이름의 기반을 제공하도록 --image-tag-base 플래그를 설정합니다. 이미지 태그가 번들 버전에 따라 설정되므로 태그 없이 이름을 입력합니다. 예를 들어 전체 번들 이미지 이름이 <image_name_base>:<bundle_version> 형식으로 생성됩니다.

검증

- 생성된 번들 이미지가 성공적으로 실행되는지 확인합니다.

```
$ operator-sdk run bundle <bundle_image_name>:<tag>
```

출력 예

```
INFO[0025] Successfully created registry pod: quay-io-my-etcd-0-9-4
INFO[0025] Created CatalogSource: etcd-catalog
INFO[0026] OperatorGroup "operator-sdk-og" created
INFO[0026] Created Subscription: etcdoperator-v0-9-4-sub
INFO[0031] Approved InstallPlan install-5t58z for the Subscription: etcdoperator-v0-9-4-sub
INFO[0031] Waiting for ClusterServiceVersion "default/etcdoperator.v0.9.4" to reach 'Succeeded' phase
INFO[0032]   Waiting for ClusterServiceVersion "default/etcdoperator.v0.9.4" to appear
INFO[0048]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Pending
INFO[0049]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Installing
INFO[0064]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Succeeded
INFO[0065] OLM has successfully installed "etcdoperator.v0.9.4"
```

5.17. OPERATOR SDK CLI 참조

Operator SDK CLI(명령줄 인터페이스)는 Operator를 더 쉽게 작성할 수 있도록 설계된 개발 키트입니다.

Operator SDK CLI 구문

```
$ operator-sdk <command> [<subcommand>] [<argument>] [<flags>]
```

Kubernetes 기반 클러스터(예: OpenShift Container Platform)에 대한 클러스터 관리자 액세스 권한이 있는 Operator 작성자는 Operator SDK CLI를 사용하여 Go, Ansible 또는 Helm을 기반으로 자체 Operator를 개발할 수 있습니다. Kubebuilder는 Go 기반 Operator의 스캐폴드 솔루션으로 Operator SDK에 포함되어 있습니다. 즉 기존 Kubebuilder 프로젝트를 그대로 Operator SDK와 함께 사용할 수 있으며 계속 작업할 수 있습니다.

5.17.1. 번들

operator-sdk bundle 명령은 Operator 번들 메타데이터를 관리합니다.

5.17.1.1. 검증

bundle validate 하위 명령은 Operator 번들을 검증합니다.

표 5.19. bundle validate 플래그

플래그	Description
-h, --help	**bundle validate** 하위 명령에 대한 도움말 출력입니다.
--index-builder(문자열)	번들 이미지를 가져오고 압축 해제하는 툴입니다. 번들 이미지를 검증할 때만 사용됩니다. 사용 가능한 옵션은 **docker**(기본값), **podman** 또는 **none**입니다.
--list-optional	사용 가능한 선택적 검증기를 모두 나열합니다. 이 플래그를 설정하면 검증기가 실행되지 않습니다.
--select-optional(문자열)	실행할 선택적 검증기를 선택하는 라벨 선택기입니다. **--list-optional** 플래그를 사용하여 실행하는 경우 사용 가능한 선택적 검증기를 나열합니다.

5.17.2. cleanup

operator-sdk cleanup 명령은 run 명령을 사용하여 배포한 Operator용으로 생성된 리소스를 삭제하고 제거합니다.

표 5.20. cleanup 플래그

플래그	Description
-h, --help	**run bundle** 하위 명령에 대한 도움말 출력입니다.
--kubeconfig(문자열)	CLI 요청에 사용할 **kubeconfig** 파일 경로입니다.
-n,--namespace (문자열)	이 플래그가 있는 경우 CLI 요청을 실행할 네임스페이스입니다.
--timeout <duration>	실패 전 명령이 완료될 때까지 대기하는 시간입니다. 기본값은 **2m0s**입니다.

5.17.3. 완료

operator-sdk completion 명령은 CLI 명령을 더 신속하고 쉽게 실행할 수 있도록 쉘 완료를 생성합니다.

표 5.21. completion 하위 명령

하위 명령	Description
bash	bash 완료를 생성합니다.
zsh	zsh 완료를 생성합니다.

표 5.22. completion 플래그

플래그	Description
-h, --help	사용법 도움말 출력입니다.

예를 들면 다음과 같습니다.

```
$ operator-sdk completion bash
```

출력 예

```
# bash completion for operator-sdk                  -*- shell-script -*-
...
# ex: ts=4 sw=4 et filetype=sh
```

5.17.4. create

operator-sdk create 명령은 Kubernetes API를 생성하거나 스캐폴드하는 데 사용됩니다.

5.17.4.1. api

create api 하위 명령은 Kubernetes API를 스캐폴드합니다. 하위 명령은 init 명령을 사용하여 초기화한 프로젝트에서 실행해야 합니다.

표 5.23. create api 플래그

플래그	Description
-h, **--help**	**run bundle** 하위 명령에 대한 도움말 출력입니다.

5.17.5. generate

operator-sdk generate 명령은 특정 생성기를 호출하여 코드 또는 매니페스트를 생성합니다.

5.17.5.1. 번들

generate bundle 하위 명령은 Operator 프로젝트에 대해 일련의 번들 매니페스트, 메타데이터, bundle.Dockerfile 파일을 생성합니다.

참고

일반적으로 generate kustomize manifests 하위 명령을 먼저 실행하여 generate bundle 하위 명령에 사용되는 입력 Kustomize 베이스를 생성합니다. 그러나 초기화된 프로젝트에서 make bundle 명령을 사용하여 이러한 명령을 순서대로 실행하도록 자동화할 수 있습니다.

표 5.24. generate bundle 플래그

플래그	Description
--channels(문자열)	번들이 속한 채널의 쉼표로 구분된 목록입니다. 기본값은 **alpha**입니다.
--crds-dir(문자열)	**CustomResoureDefinition** 매니페스트의 루트 디렉터리입니다.
--default-channel(문자열)	번들의 기본 채널입니다.
--deploy-dir(문자열)	배포 및 RBAC와 같은 Operator 매니페스트용 루트 디렉터리입니다. 이 디렉터리는 **--input-dir** 플래그로 전달되는 디렉터리와 다릅니다.
-h, **--help**	**generate bundle**에 대한 도움말입니다.
--input-dir(문자열)	기존 번들을 읽을 디렉터리입니다. 이 디렉터리는 번들 **manifests** 디렉터리의 상위이며 **--deploy-dir** 디렉터리와 다릅니다.
--kustomize-dir(문자열)	번들 매니페스트용 Kustomize 베이스 및 **kustomization.yaml** 파일이 포함된 디렉터리입니다. 기본 경로는 **config/manifests**입니다.
--manifests	번들 매니페스트를 생성합니다.
--metadata	번들 메타데이터 및 Dockerfile을 생성합니다.
--output-dir(문자열)	번들을 작성할 디렉터리입니다.

플래그	Description
--overwrite	번들 메타데이터 및 Dockerfile이 있는 경우 덮어씁니다. 기본값은 **true**입니다.
--package(문자열)	번들의 패키지 이름입니다.
-q, --quiet	자동 모드로 실행됩니다.
--stdout	번들 매니페스트를 표준 출력에 작성합니다.
--version(문자열)	생성된 번들에 있는 Operator의 의미 체계 버전입니다. 새 번들을 생성하거나 Operator를 업그레이드하는 경우에만 설정됩니다.

추가 리소스

- ***make bundle** 명령을 사용하여 **generate bundle** 하위 명령을 호출하는 것을 포함하는 전체 프로시저는 Operator 번들링을 참조하십시오.*

5.17.5.2. kustomize

***generate kustomize** 하위 명령에는 **Operator**에 대한 Kustomize 데이터를 생성하는 하위 명령이 포함되어 있습니다.*

5.17.5.2.1. 매니페스트

***generate kustomize manifests** 하위 명령은 다른 **Operator SDK** 명령에서 번들 매니페스트를 빌드하는 데 사용하는 **Kustomize** 베이스 및 **kustomization.yaml** 파일을 **config/manifests** 디렉터리에 생성하거나 다시 생성합니다. 베이스가 존재하지 않거나 **--interactive=false** 플래그를 설정하지 않은 경우 이 명령은 기본적으로 매니페스트 베이스의 중요한 구성 요소인 **UI** 메타데이터를 대화형으로 요청합니다.*

*표 5.25. **generate kustomize manifests** 플래그*

플래그	Description
--apis-dir(문자열)	API 유형 정의를 위한 루트 디렉터리입니다.
-h, --help	**generate kustomize manifests**에 대한 도움말입니다.
--input-dir(문자열)	기존 Kustomize 파일이 있는 디렉터리입니다.
--interactive	**false**로 설정하면 Kustomize 베이스가 없는 경우 사용자 정의 메타데이터를 수락하도록 대화형 명령 프롬프트가 표시됩니다.
--output-dir(문자열)	Kustomize 파일을 작성할 디렉터리입니다.

플래그	Description
--package(문자열)	패키지 이름입니다.
-q, **--quiet**	자동 모드로 실행됩니다.

5.17.6. init

***operator-sdk init** 명령은 **Operator** 프로젝트를 초기화하고 지정된 플러그인 의 기본 프로젝트 디렉터리 레이아웃을 생성하거나 스캐폴드 합니다.*

이 명령은 다음 파일을 작성합니다.

- *상용구 라이센스 파일*
- *도메인 및 리포지토리가 있는 **PROJECT** 파일*
- *프로젝트를 빌드할 **Makefile***
- *프로젝트 종속 항목이 있는 **go.mod** 파일*
- *매니페스트를 사용자 정의하는 **kustomization.yaml** 파일*
- *관리자 매니페스트용 이미지를 사용자 정의하는 패치 파일*
- ***Prometheus** 지표를 활성화하는 패치 파일*
- *실행할 **main.go** 파일*

*표 **5.26. init** 플래그*

플래그	Description
--help, -h	**init** 명령에 대한 도움말 출력입니다.
--plugins(문자열)	프로젝트를 초기화할 플러그인의 이름 및 버전(선택 사항)입니다. 사용 가능한 플러그인은 **ansible.sdk.operatorframework.io/v1**,**go.kubebuilder.io/v2**,**go.kubebuilder.io/v3**, **helm.sdk.operatorframework.io/v1** 입니다.
--project-version	프로젝트 버전입니다. 사용 가능한 값은 **2** 및 기본값인 **3-alpha** 입니다.

5.17.7. run

***operator-sdk run** 명령은 다양한 환경에서 Operator를 시작할 수 있는 옵션을 제공합니다.*

5.17.7.1. 번들

***run bundle** 하위 명령은 OLM(Operator Lifecycle Manager)을 사용하여 번들 형식으로 Operator를 배포합니다.*

표 5.27. run bundle 플래그

플래그	Description
--index-image(문자열)	번들을 삽입할 인덱스 이미지입니다. 기본 이미지는 **quay.io/operator-framework/upstream-opm-builder:latest**입니다.
--install-mode <install_mode_value>	Operator CSV(클러스터 서비스 버전)에서 지원되는 설치 모드(예: **AllNamespaces** 또는 **SingleNamespace**)입니다.
--timeout <duration>	설치 제한 시간입니다. 기본값은 **2m0s**입니다.
--kubeconfig(문자열)	CLI 요청에 사용할 **kubeconfig** 파일 경로입니다.
-n,--namespace (문자열)	이 플래그가 있는 경우 CLI 요청을 실행할 네임스페이스입니다.
-h, --help	**run bundle** 하위 명령에 대한 도움말 출력입니다.

추가 리소스

- *가능한 설치 모드에 대한 자세한 내용은 Operator group 멤버십 을 참조하십시오.*

5.17.7.2. bundle-upgrade

***run bundle-upgrade** 하위 명령은 이전에 **OLM(Operator Lifecycle Manager)**을 사용하여 번들 형식으로 설치한 **Operator**를 업그레이드합니다.*

표 5.28. run bundle-upgrade 플래그

플래그	Description
--timeout <duration>	업그레이드 제한 시간입니다. 기본값은 **2m0s**입니다.
--kubeconfig(문자열)	CLI 요청에 사용할 **kubeconfig** 파일 경로입니다.
-n,--namespace (문자열)	이 플래그가 있는 경우 CLI 요청을 실행할 네임스페이스입니다.
-h, --help	**run bundle** 하위 명령에 대한 도움말 출력입니다.

5.17.8. scorecard

***operator-sdk scorecard** 명령은 스코어 카드 툴을 실행하여 **Operator** 번들을 검증하고 개선을 위해 제안 사항을 제공합니다. 이 명령은 하나의 인수를 사용하며, 인수는 번들 이미지이거나 매니페스트 및 메타데이터가 포함된 디렉터리입니다. 인수에 이미지 태그가 있으면 이미지가 원격으로 존재해야 합니다.*

표 5.29. scorecard 플래그

플래그	Description
-c, --config(문자열)	스코어 카드 구성 파일 경로입니다. 기본 경로는 **bundle/tests/scorecard/config.yaml**입니다.
-h, --help	**scorecard** 명령에 대한 도움말 출력입니다.
--kubeconfig(문자열)	**kubeconfig** 파일 경로입니다.
-L, --list	실행할 수 있는 테스트를 나열합니다.
-n, --namespace(문자열)	테스트 이미지를 실행할 네임스페이스입니다.
-o, --output(문자열)	결과 출력 형식입니다. 사용 가능한 값은 **text**(기본값) 및 **json**입니다.
-l, --selector(문자열)	실행할 테스트를 결정하는 라벨 선택기입니다.
-s, --service-account(문자열)	테스트에 사용할 서비스 계정입니다. 기본값은 **default**입니다.

플래그	Description
-x, --skip-cleanup	테스트가 실행된 후 리소스 정리를 비활성화합니다.
-w, --wait-time <duration>	테스트가 완료될 때까지 대기할 시간(초)입니다(예: **35s**). 기본값은 **30s**입니다.

추가 리소스

- *스코어 카드 툴 실행에 대한 자세한 내용은 스코어 카드 툴을 사용하여 Operator 검증을 참조하십시오.*

6장. 클러스터 OPERATOR 참조

이 참조 가이드에서는 OpenShift Container Platform의 아키텍처 기반 역할을 하는 Red Hat에서 제공하는 클러스터 Operator 를 인덱싱합니다. 클러스터 Operator는 별도로 명시하지 않는 한 기본적으로 설치되고 CVO(Cluster Version Operator)에 의해 관리됩니다. 컨트롤 플레인 아키텍처에 대한 자세한 내용은 OpenShift Container Platform의 Operator 를 참조하십시오.

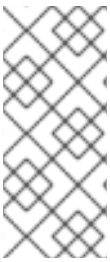
클러스터 관리자는 관리 → 클러스터 설정 페이지에서 OpenShift Container Platform 웹 콘솔에서 클러스터 Operator를 볼 수 있습니다.

참고

클러스터 Operator는 OLM(Operator Lifecycle Manager) 및 OperatorHub에서 관리하지 않습니다. OLM 및 OperatorHub는 OpenShift Container Platform에서 선택적 애드온 Operator를 설치 및 실행하는 데 사용되는 Operator 프레임워크 의 일부입니다.

다음 클러스터 Operator 중 일부는 설치 전에 비활성화할 수 있습니다. 자세한 내용은 클러스터 기능 보기를 참조하십시오.

6.1. CLUSTER BAREMETAL OPERATOR



참고

Cluster Baremetal Operator는 설치 중에 클러스터 관리자가 비활성화할 수 있는 선택적 클러스터 기능입니다. 선택적 클러스터 기능에 대한 자세한 내용은 설치 후 구성의 "클러스터 기능"을 참조하십시오.

목적

CBO(Cluster Baremetal Operator)는 베어 메탈 서버를 OpenShift Container Platform 컴퓨팅 노드를 실행할 준비가 된 완전히 작동하는 작업자 노드에 가져오는 데 필요한 모든 구성 요소를 배포합니다. CBO를 사용하면 BMO(Bare Metal Operator) 및 Ironic 컨테이너로 구성된 metal3 배포가 OpenShift Container Platform 클러스터 내의 컨트롤 플레인 노드 중 하나에서 실행됩니다. CBO는 또한 감시하고 적절한 조치를 수행하는 리소스에 대한 OpenShift Container Platform 업데이트를 수신 대기합니다.

프로젝트

cluster-baremetal-operator

추가 리소스

- 베어 메탈 기능

6.2. BARE METAL EVENT RELAY

목적

OpenShift Bare Metal Event Relay는 베어 메탈 이벤트 릴레이의 라이프사이클을 관리합니다. Bare Metal Event Relay를 사용하면 Redfish 하드웨어 이벤트를 사용하여 모니터링되는 클러스터 이벤트 유형을 구성할 수 있습니다.

구성 오브젝트

이 명령을 사용하여 설치 후 구성을 편집할 수 있습니다(예: 웹 후크 포트). 다음을 사용하여 구성 오브젝트를 편집할 수 있습니다.

```
$ oc -n [namespace] edit cm hw-event-proxy-operator-manager-config
```

```
apiVersion: controller-runtime.sigs.k8s.io/v1alpha1
kind: ControllerManagerConfig
health:
  healthProbeBindAddress: :8081
metrics:
  bindAddress: 127.0.0.1:8080
webhook:
  port: 9443
leaderElection:
  leaderElect: true
  resourceName: 6e7a703c.redhat-cne.org
```

프로젝트

hw-event-proxy-operator

CRD

프록시를 통해 베어 메탈 클러스터에서 실행되는 애플리케이션은 HardwareEvent CR을 사용하여 보고한 온도 임계값, 팬 실패, 디스크 손실, 정전 및 메모리 장애와 같은 Redfish 하드웨어 변경 및 실패에 신속하게 대응할 수 있습니다.

hardwareevents.event.redhat-cne.org:

- 범위: 네임스페이스

- CR: HardwareEvent
- 검증: 예

추가 리소스

- Redfish 하드웨어 이벤트 모니터링

6.3. CLOUD CREDENTIAL OPERATOR

목적

CCO(Cloud Credential Operator)는 클라우드 공급자 자격 증명을 Kubernetes CRD(사용자 지정 리소스 정의)로 관리합니다. CCO는 CredentialsRequest CR(사용자 정의 리소스)에서 동기화되어 OpenShift Container Platform 구성 요소가 클러스터를 실행하는 데 필요한 특정 권한이 있는 클라우드 공급자 자격 증명을 요청할 수 있습니다.

install-config.yaml 파일에서 credentialsMode 매개변수에 다양한 값을 설정하면 CCO를 여러 모드에서 작동하도록 구성할 수 있습니다. 모드를 지정하지 않거나 credentialsMode 매개변수를 빈 문자열("")로 설정하면 CCO가 기본 모드에서 작동합니다.

프로젝트

openshift-cloud-credential-operator

CRD

- credentialsrequests.cloudcredential.openshift.io
  - 범위: 네임스페이스
  - CR: CredentialsRequest
  - 검증: 예

구성 오브젝트

구성이 필요하지 않습니다.

추가 리소스

- CredentialsRequest 사용자 정의 리소스
- Cloud Credential Operator 정보

6.4. CLUSTER AUTHENTICATION OPERATOR

목적

Cluster Authentication Operator는 클러스터에서 Authentication 사용자 정의 리소스를 설치 및 유지 관리하고 다음을 실행하여 확인할 수 있습니다.

```
$ oc get clusteroperator authentication -o yaml
```

프로젝트

cluster-authentication-operator

6.5. CLUSTER AUTOSCALER OPERATOR

목적

Cluster Autoscaler Operator는 cluster-api 공급자를 사용하여 OpenShift Cluster Autoscaler 배포를 관리합니다.

프로젝트

cluster-autoscaler-operator

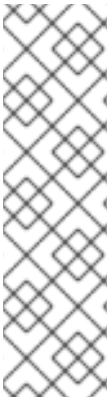
CRD

- ClusterAutoscaler: 클러스터의 구성 자동 스케일러 인스턴스를 제어하는 단일 생성 리소스입니다. Operator는 관리형 네임스페이스에서 WATCH_NAMESPACE 환경 변수의 값인 default라는 ClusterAutoscaler 리소스에만 응답합니다.
- MachineAutoscaler: 이 리소스는 노드 그룹을 대상으로 하며 그룹, min 및 max 크기에 대해 자동 스케일링을 활성화하고 구성하는 주석을 관리합니다. 현재는 MachineSet 오브젝트만 대상

으로 할 수 있습니다.

6.6. CLUSTER CLOUD CONTROLLER MANAGER OPERATOR

목적



참고

이 Operator는 Microsoft Azure Stack Hub에서만 완전 지원됩니다.

Alibaba Cloud, AWS(Amazon Web Services), GCP(Google Cloud Platform), IBM Cloud, Microsoft Azure, RHOSP(Red Hat OpenStack Platform) 및 VMware vSphere 에 대한 기술 프리뷰 로 사용할 수 있습니다.

Cluster Cloud Controller Manager Operator는 OpenShift Container Platform 상단에 배포된 클라우드 컨트롤러 관리자를 관리하고 업데이트합니다. Operator는 Kubebuilder 프레임워크 및 controller-runtime 라이브러리를 기반으로 합니다. CVO(Cluster Version Operator)를 통해 설치됩니다.

여기에는 다음 구성 요소가 포함됩니다.

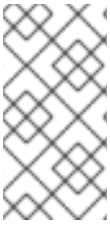
- Operator
- 클라우드 구성 관찰자

기본적으로 Operator는 metrics 서비스를 통해 Prometheus 지표를 표시합니다.

프로젝트

cluster-cloud-controller-manager-operator

6.7. CLUSTER CAPI OPERATOR



참고

이 Operator는 AWS(Amazon Web Services) 및 GCP(Google Cloud Platform)의 기술 프리뷰 로 제공됩니다.

목적

Cluster CAPI Operator는 Cluster API 리소스의 라이프사이클을 유지 관리합니다. 이 Operator는 OpenShift Container Platform 클러스터 내에서 클러스터 API 프로젝트 배포와 관련된 모든 관리 작업을 담당합니다.

프로젝트

cluster-capi-operator

CRD

- *awsmachines.infrastructure.cluster.x-k8s.io*
  - *범위: 네임스페이스*
  - *CR: awsmachine*
  - *검증: 아니요*
- *gcpmachines.infrastructure.cluster.x-k8s.io*
  - *범위: 네임스페이스*
  - *CR: gcpmachine*
  - *검증: 아니요*
- *awsmachinetemplates.infrastructure.cluster.x-k8s.io*

- 범위: 네임스페이스
- CR: awsmachinetemplate
- 검증: 아니요

- gcpmachinetemplates.infrastructure.cluster.x-k8s.io
  - 범위: 네임스페이스
  - CR: gcpmachinetemplate
  - 검증: 아니요

6.8. CLUSTER CONFIG OPERATOR

목적

Cluster Config Operator는 config.openshift.io와 관련된 다음 작업을 수행합니다.

- CRD를 생성합니다.
- 초기 사용자 정의 리소스를 렌더링합니다.
- 마이그레이션을 처리합니다.

프로젝트

cluster-config-operator

6.9. CLUSTER CSI SNAPSHOT CONTROLLER OPERATOR

목적

Cluster CSI Snapshot Controller Operator는 **CSI Snapshot Controller**를 설치하고 유지 관리합니다. **CSI Snapshot** 컨트롤러는 **VolumeSnapshot CRD** 오브젝트를 모니터링하고 볼륨 스냅샷의 생성 및 삭제 라이프사이클을 관리합니다.

프로젝트

cluster-csi-snapshot-controller-operator

6.10. CLUSTER IMAGE REGISTRY OPERATOR

목적

Cluster Image Registry Operator는 **OpenShift** 이미지 레지스트리의 단일 인스턴스를 관리합니다. 스토리지 생성을 포함하여 레지스트리의 모든 구성을 관리합니다.

처음 시작 시 **Operator**는 클러스터에서 탐지된 구성에 따라 기본 **image-registry** 리소스 인스턴스를 생성합니다. 이는 클라우드 공급자를 기반으로 사용할 클라우드 스토리지 유형을 나타냅니다.

완전한 **image-registry** 리소스를 정의하는 데 사용할 수 있는 정보가 충분하지 않으면 불완전한 리소스가 정의되고 **Operator**는 누락된 항목에 대한 정보를 사용하여 리소스 상태를 업데이트합니다.

Cluster Image Registry Operator는 **openshift-image-registry** 네임스페이스에서 실행되며 해당 위치의 레지스트리 인스턴스도 관리합니다. 레지스트리의 모든 설정 및 워크로드 리소스는 해당 네임 스페이스에 있습니다.

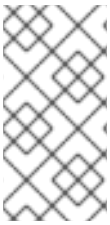
프로젝트

cluster-image-registry-operator

6.11. CLUSTER MACHINE APPROVER OPERATOR

목적

Cluster Machine Approver Operator는 클러스터 설치 후 새 작업자 노드에 대해 요청된 **CSR**을 자동으로 승인합니다.



참고

컨트롤 플레인 노드의 경우 부트스트랩 노드의 **approve-csr** 서비스는 클러스터 부트스트랩 단계 중에 모든 **CSR**을 자동으로 승인합니다.

프로젝트

cluster-machine-approver-operator

6.12. CLUSTER MONITORING OPERATOR

목적

Cluster Monitoring Operator는 OpenShift Container Platform 상단에 배포된 Prometheus 기반 클러스터 모니터링 스택을 관리하고 업데이트합니다.

프로젝트

openshift-monitoring

CRD

- *alertmanagers.monitoring.coreos.com*
  - *범위: 네임스페이스*
  - *CR: alertmanager*
  - *검증: 예*
- *prometheuses.monitoring.coreos.com*
  - *범위: 네임스페이스*
  - *CR: prometheus*
  - *검증: 예*
- *prometheusrules.monitoring.coreos.com*

- 범위: 네임스페이스
- **CR: prometheusrule**
- 검증: 예

- **servicemonitors.monitoring.coreos.com**
  - 범위: 네임스페이스
  - **CR: servicemonitor**
  - 검증: 예

구성 오브젝트

```
$ oc -n openshift-monitoring edit cm cluster-monitoring-config
```

6.13. CNO(CLUSTER NETWORK OPERATOR)

목적

Cluster Network Operator는 **OpenShift Container Platform** 클러스터에 네트워킹 구성 요소를 설치 및 업그레이드합니다.

6.14. CLUSTER SAMPLES OPERATOR

참고

Cluster Samples Operator는 설치 중에 클러스터 관리자가 비활성화할 수 있는 선택적 클러스터 기능입니다. 선택적 클러스터 기능에 대한 자세한 내용은 설치 후 구성의 "클러스터 기능"을 참조하십시오.

목적

Cluster Samples Operator는 **openshift** 네임스페이스에 저장된 샘플 이미지 스트림 및 템플릿을 관리

합니다.

*처음 시작 시 **Operator**는 기본 샘플 구성 리소스를 생성하여 이미지 스트림 및 템플릿을 생성하기 시작합니다. 구성 오브젝트는 키가 **cluster**이고 유형이 **configs.samples**인 클러스터 범위 지정 오브젝트입니다.*

*이미지 스트림은 **registry.redhat.io**의 이미지를 가리키는 **RHCOS(Red Hat Enterprise Linux CoreOS)** 기반 **OpenShift Container Platform** 이미지 스트림입니다. 마찬가지로 템플릿은 **OpenShift Container Platform** 템플릿으로 분류된 템플릿입니다.*

*Cluster Samples Operator 배포는 **openshift-cluster-samples-operator** 네임스페이스에 포함됩니다. 시작 시 **OpenShift** 이미지 레지스트리 및 **API** 서버의 이미지 스트림 가져오기 논리에서 **registry.redhat.io** 로 인증하는 데 설치 풀 시크릿을 사용합니다. 관리자는 샘플 이미지 스트림에 사용된 레지스트리를 변경하는 경우 **openshift** 네임스페이스에 추가 보안을 생성할 수 있습니다. 생성한 경우 이러한 보안에는 이미지를 손쉽게 가져오는 데 필요한 **docker**의 **config.json** 콘텐츠가 포함됩니다.*

*Cluster Samples Operator 이미지에는 관련 **OpenShift Container Platform** 릴리스의 이미지 스트림 및 템플릿 정의가 포함되어 있습니다. **Cluster Samples Operator**는 샘플을 생성한 후 호환되는 **OpenShift Container Platform** 버전을 나타내는 주석을 추가합니다. **Operator**는 이 주석을 사용하여 각 샘플이 호환되는 릴리스 버전과 일치하는지 확인합니다. 인벤토리 외부 샘플은 건너뛰기한 샘플과 마찬가지로 무시됩니다.*

*Operator에서 관리하는 모든 샘플에 대한 수정은 버전 주석을 수정하거나 삭제하지 않는 한 허용됩니다. 그러나 업그레이드 시 버전 주석이 변경되면 샘플이 최신 버전으로 업데이트되므로 이러한 수정 사항이 대체될 수 있습니다. **Jenkins** 이미지는 설치의 이미지 페이로드에 포함되어 있으며 이미지 스트림에 직접 태그가 지정됩니다.*

샘플 리소스에는 삭제 시 다음을 정리하는 종료자가 포함됩니다.

- *Operator에서 관리하는 이미지 스트림*
- *Operator에서 관리하는 템플릿*
- *Operator에서 생성하는 구성 리소스*
- *클러스터 상태 리소스*

샘플 리소스를 삭제하면 Cluster Samples Operator에서 기본 구성을 사용하여 리소스를 다시 생성합니다.

프로젝트

cluster-samples-operator

추가 리소스

- OpenShift 샘플 기능

6.15. CLUSTER STORAGE OPERATOR

목적

Cluster Storage Operator는 OpenShift Container Platform 클러스터 수준 스토리지 기본값을 설정합니다. 이를 통해 OpenShift Container Platform 클러스터에 기본 스토리지 클래스가 있는지 확인합니다.

프로젝트

cluster-storage-operator

설정

구성이 필요하지 않습니다.

참고

- Cluster Storage Operator는 AWS(Amazon Web Services) 및 RHOSP(Red Hat OpenStack Platform)를 지원합니다.
- 생성한 스토리지 클래스는 주석을 편집하여 기본이 아닌 상태로 설정할 수 있지만 Operator에서 실행하는 동안에는 스토리지 클래스를 삭제할 수 없습니다.

6.16. CLUSTER VERSION OPERATOR

목적

클러스터 Operator는 특정 클러스터 기능 영역을 관리합니다. CVO(Cluster Version Operator)는 기본적으로 OpenShift Container Platform에 설치된 클러스터 Operator의 라이프사이클을 관리합니다.

CVO도 OpenShift Update Service를 통해 현재 구성 요소 버전 및 그래프의 정보를 기반으로 유효한 업데이트 및 업데이트 경로를 확인합니다.

프로젝트

cluster-version-operator

추가 리소스

- OpenShift Container Platform의 Operator

6.17. CONSOLE OPERATOR

목적

Console Operator에서 클러스터에 OpenShift Container Platform 웹 콘솔을 설치하고 유지 관리합니다.

프로젝트

console-operator

6.18. DNS OPERATOR

목적

DNS Operator는 CoreDNS를 배포 및 관리하고 Pod에 이름 확인 서비스를 제공하여 OpenShift Container Platform에서 DNS 기반 Kubernetes 서비스 검색을 사용할 수 있도록 합니다.

Operator는 클러스터 구성을 기반으로 작동하는 기본 배포를 생성합니다.

- 기본 클러스터 도메인은 cluster.local입니다.
- CoreDNS Corefile 또는 Kubernetes 플러그인 구성은 아직 지원되지 않습니다.

DNS Operator는 고정 IP를 사용하여 서비스로 노출되는 Kubernetes 데몬 세트로 CoreDNS를 관리합니다. CoreDNS는 클러스터의 모든 노드에서 실행됩니다.

프로젝트

cluster-dns-operator

6.19. ETCD 클러스터 OPERATOR

목적

etcd 클러스터 Operator는 etcd 클러스터 스케일링을 자동화하고 etcd 모니터링 및 지표를 활성화하여 재해 복구 절차를 간소화합니다.

프로젝트

cluster-etcd-operator

CRD

- **etcds.operator.openshift.io**
  - 범위: 클러스터
  - CR: etcd
  - 검증: 예

구성 오브젝트

```
$ oc edit etcd cluster
```

6.20. INGRESS OPERATOR

목적

Ingress Operator는 OpenShift Container Platform 라우터를 구성하고 관리합니다.

프로젝트

openshift-ingress-operator

CRD

- **clusteringresses.ingress.openshift.io**

clusteringresses.operator.openshift.io

- 범위: 네임스페이스
- CR: clusteringresses
- 검증: 아니요

구성 오브젝트

- 클러스터 구성
  - 유형 이름: clusteringresses.ingress.openshift.io
  - 인스턴스 이름: default
  - 보기 명령:

```
$ oc get clusteringresses.ingress.openshift.io -n openshift-ingress-operator
default -o yaml
```

참고

Ingress Operator는 openshift-ingress 프로젝트에서 라우터를 설정하고 라우터용 배포를 생성합니다.

```
$ oc get deployment -n openshift-ingress
```

Ingress Operator는 network/cluster 상태의 clusterNetwork[].cidr 을 사용하여 관리형 Ingress 컨트롤러(라우터)가 작동해야 하는 모드(IPv4, IPv6 또는 이중 스택)를 결정합니다. 예를 들어 clusterNetwork에 v6 cidr 만 포함된 경우 Ingress 컨트롤러가 IPv6 전용 모드에서 작동합니다.

다음 예제에서는 클러스터 네트워크가 하나뿐이고 네트워크는 IPv4 cidr:이므로 Ingress Operator에서 관리하는 Ingress 컨트롤러가 IPv4 전용 모드에서 실행됩니다.

```
$ oc get network/cluster -o jsonpath='{.status.clusterNetwork[*]}'
```

출력 예

```
map[cidr:10.128.0.0/14 hostPrefix:23]
```

6.21. INSIGHTS OPERATOR

목적

Insights Operator는 OpenShift Container Platform 구성 데이터를 수집하여 이를 Red Hat으로 보냅니다. 데이터는 클러스터가 노출될 수 있는 문제에 대한 사전 대응적인 인사이트 권장 사항을 생성하는 데 사용됩니다. 이러한 통찰력은 console.redhat.com 의 Insights Advisor를 통해 클러스터 관리자에게 전달됩니다.

프로젝트

insights-operator

설정

구성이 필요하지 않습니다.

참고

Insights Operator는 OpenShift Container Platform Telemetry를 환영합니다.

추가 리소스

- Insights Operator 및 Telemetry에 대한 자세한 내용은 원격 상태 모니터링 정보

6.22. KUBERNETES API SERVER OPERATOR

목적

Kubernetes API Server Operator는 OpenShift Container Platform 상단에 배포된 Kubernetes API 서버를 관리하고 업데이트합니다. 이 Operator는 OpenShift Container Platform library-go 프레임워크를 기반으로 하며 CVO(Cluster Version Operator)를 사용하여 설치됩니다.

프로젝트

openshift-kube-apiserver-operator

CRD

- *kubeapiservers.operator.openshift.io*
  - *범위: 클러스터*
  - *CR: kubeapiserver*
  - *검증: 예*

구성 오브젝트

```
$ oc edit kubeapiserver
```

6.23. KUBERNETES CONTROLLER MANAGER OPERATOR

목적

Kubernetes Controller Manager Operator는 OpenShift Container Platform 상단에 배포된 Kubernetes Controller Manager를 관리하고 업데이트합니다. 이 Operator는 OpenShift Container Platform library-go 프레임워크를 기반으로 하며 CVO(Cluster Version Operator)를 사용하여 설치됩니다.

여기에는 다음 구성 요소가 포함됩니다.

- *Operator*
- *부트스트랩 매니페스트 렌더러*
- *정적 Pod 기반 설치*
- *구성 관찰자*

기본적으로 Operator는 metrics 서비스를 통해 Prometheus 지표를 표시합니다.

프로젝트

cluster-kube-controller-manager-operator

6.24. KUBERNETES SCHEDULER OPERATOR

목적

Kubernetes Scheduler Operator는 OpenShift Container Platform 상단에 배포된 Kubernetes Scheduler를 관리하고 업데이트합니다. 이 Operator는 OpenShift Container Platform library-go 프레임워크를 기반으로 하며 CVO(Cluster Version Operator)를 사용하여 설치됩니다.

Kubernetes Scheduler Operator에는 다음 구성 요소가 포함됩니다.

- Operator
- 부트스트랩 매니페스트 렌더러
- 정적 Pod 기반 설치
- 구성 관찰자

기본적으로 Operator는 metrics 서비스를 통해 Prometheus 지표를 표시합니다.

프로젝트

cluster-kube-scheduler-operator

설정

Kubernetes Scheduler의 구성은 다음을 병합한 결과입니다.

- 기본 구성.

- 사양 **schedulers.config.openshift.io**에서 관찰된 구성

이러한 구성은 모두 스파스 구성으로, 마지막에 유효한 구성을 형성하기 위해 병합된, 무효화된 **JSON** 조각입니다.

6.25. KUBERNETES STORAGE 버전 MIGRATOR OPERATOR

목적

Kubernetes Storage Version Migrator Operator는 기본 스토리지 버전의 변경 사항을 감지하고, 스토리지 버전이 변경될 때 리소스 유형에 대한 마이그레이션 요청을 생성하고 마이그레이션 요청을 처리합니다.

프로젝트

cluster-kube-storage-version-migrator-operator

6.26. MACHINE API OPERATOR

목적

Machine API Operator는 **Kubernetes API**를 확장하는 특정 용도의 **CRD**(사용자 정의 리소스 정의), 컨트롤러, **RBAC** 오브젝트의 라이프사이클을 관리합니다. 이를 통해 클러스터에서 원하는 머신 상태를 선언합니다.

프로젝트

machine-api-operator

CRD

- **MachineSet**
- **Machine**
- **MachineHealthCheck**

6.27. MACHINE CONFIG OPERATOR

목적

*Machine Config Operator*는 커널과 *kubelet* 사이의 모든 것을 포함하여 기본 운영 체제 및 컨테이너 런타임의 구성 및 업데이트를 관리하고 적용합니다.

다음의 네 가지 구성 요소가 있습니다.

- *machine-config-server:* 클러스터에 가입하는 새 머신에 *Ignition* 설정을 제공합니다.
- *machine-config-controller: MachineConfig* 객체에 의해 정의된 설정으로 머신 업그레이드를 조정합니다. 머신 세트의 업그레이드를 개별적으로 제어하는 옵션이 제공됩니다.
- *machine-config-daemon:* 업데이트 중에 새로운 머신 설정을 적용합니다. 머신 상태를 요청한 머신 구성에 대해 검증하고 확인합니다.
- *machine-config:* 처음으로 머신을 설치, 시작 및 업데이트하기위한 완전한 머신 구성 소스를 제공합니다.

중요

현재는 머신 구성 서버 끝점을 차단하거나 제한할 수 있는 방법이 없습니다. 기존 구성 또는 상태가 없는 새로 프로비저닝된 머신이 구성을 가져올 수 있도록 머신 구성 서버를 네트워크에 노출해야 합니다. 이 모델에서 신뢰의 루트는 *CSR*(인증서 서명 요청) 끝점으로, *kubelet*이 클러스터에 가입하기 위해 승인하기 위해 인증서 서명 요청을 보내는 위치입니다. 이로 인해 시크릿 및 인증서와 같은 중요한 정보를 배포하는 데 머신 구성을 사용해서는 안 됩니다.

머신 구성 서버 끝점, 포트 *22623* 및 *22624*가 베어 메탈 시나리오에서 보호되도록 하려면 고객이 적절한 네트워크 정책을 구성해야 합니다.

추가 리소스

- *OpenShift SDN 네트워크 플러그인 정보*.

프로젝트

openshift-machine-config-operator

6.28. MARKETPLACE OPERATOR

참고

Marketplace Operator는 설치 중에 클러스터 관리자가 비활성화할 수 있는 선택적 클러스터 기능입니다. 선택적 클러스터 기능에 대한 자세한 내용은 설치 후 구성의 "클러스터 기능"을 참조하십시오.

목적

Marketplace Operator는 클러스터의 기본 **OLM(Operator Lifecycle Manager)** 카탈로그 세트를 사용하여 클러스터 외부 **Operator**를 클러스터로 가져오는 프로세스를 간소화합니다. **Marketplace Operator**가 설치되면 **openshift-marketplace** 네임스페이스가 생성됩니다. **OLM**은 클러스터의 모든 네임스페이스에서 **openshift-marketplace** 네임스페이스에 설치된 카탈로그 소스를 사용할 수 있도록 합니다.

프로젝트

operator-marketplace

추가 리소스

- 마켓플레이스 기능

6.29. NODE TUNING OPERATOR

목적

Node Tuning Operator는 **TuneD** 데몬을 오케스트레이션하여 노드 수준 튜닝을 관리하고 **Performance Profile** 컨트롤러를 사용하여 짧은 대기 시간 성능을 달성하는 데 도움이 됩니다. 대부분의 고성능 애플리케이션에는 일정 수준의 커널 튜닝이 필요합니다. **Node Tuning Operator**는 노드 수준 **sysctls** 사용자에게 통합 관리 인터페이스를 제공하며 사용자의 필요에 따라 지정되는 사용자 정의 튜닝을 추가할 수 있는 유연성을 제공합니다.

Operator는 **OpenShift Container Platform**의 컨테이너화된 **TuneD** 데몬을 **Kubernetes** 데몬 세트로 관리합니다. 클러스터에서 실행되는 모든 컨테이너화된 **TuneD** 데몬에 사용자 정의 튜닝 사양이 데몬이 이해할 수 있는 형식으로 전달되도록 합니다. 데몬은 클러스터의 모든 노드에서 노드당 하나씩 실행됩니다.

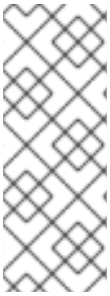
컨테이너화된 **TuneD** 데몬을 통해 적용되는 노드 수준 설정은 프로필 변경을 트리거하는 이벤트 시 또는 컨테이너화된 **TuneD** 데몬이 종료 신호를 수신하고 처리하여 정상적으로 종료될 때 롤백됩니다.

Node Tuning Operator는 **Performance Profile** 컨트롤러를 사용하여 자동 튜닝을 구현하여

OpenShift Container Platform 애플리케이션에 대해 짧은 대기 시간 성능을 실현합니다. 클러스터 관리자는 다음과 같은 노드 수준 설정을 정의하도록 성능 프로필을 구성합니다.

- *커널을 kernel-rt로 업데이트*
- *하우스키핑을 위해 CPU를 선택합니다.*
- *워크로드 실행을 위한 CPU 선택.*

버전 4.1 이상에서는 Node Tuning Operator가 표준 OpenShift Container Platform 설치에 포함되어 있습니다.



참고

이전 버전의 OpenShift Container Platform에서는 OpenShift 애플리케이션에 대해 짧은 대기 시간 성능을 실현하기 위해 자동 튜닝을 구현하는 데 Performance Addon Operator를 사용했습니다. OpenShift Container Platform 4.11 이상에서는 이 기능은 Node Tuning Operator의 일부입니다.

프로젝트

cluster-node-tuning-operator

추가 리소스

- *OCP 노드의 짧은 대기 시간 튜닝*

6.30. OPENSHIFT API SERVER OPERATOR

목적

OpenShift API Server Operator는 클러스터에서 openshift-apiserver를 설치하고 유지 관리합니다.

프로젝트

openshift-apiserver-operator

CRD

- *openshiftapiservers.operator.openshift.io*
  - *범위: 클러스터*
  - *CR: openshiftapiserver*
  - *검증: 예*

6.31. OPENSHIFT CONTROLLER MANAGER OPERATOR

목적

OpenShift Controller Manager Operator는 클러스터에서 OpenShiftControllerManager 사용자 정의 리소스를 설치하고 유지 관리하며 다음을 실행하여 확인할 수 있습니다.

```
$ oc get clusteroperator openshift-controller-manager -o yaml
```

CRD(사용자 정의 리소스 정의) openshiftcontrollermanagers.operator.openshift.io는 다음을 실행하여 클러스터에서 확인할 수 있습니다.

```
$ oc get crd openshiftcontrollermanagers.operator.openshift.io -o yaml
```

프로젝트

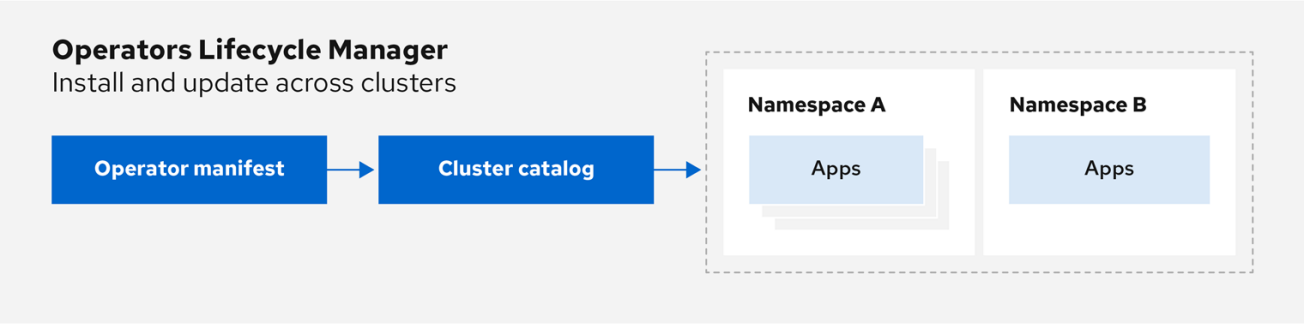
cluster-openshift-controller-manager-operator

6.32. OPERATOR LIFECYCLE MANAGER OPERATOR

목적

OLM(Operator Lifecycle Manager)은 OpenShift Container Platform 클러스터에서 실행되는 Kubernetes 네이티브 애플리케이션(Operator) 및 관련 서비스의 라이프사이클을 설치, 업데이트, 관리하는 데 도움이 됩니다. Operator 프레임워크의 일부로, 효과적이고 자동화되었으며 확장 가능한 방식으로 Operator를 관리하도록 설계된 오픈 소스 툴킷입니다.

그림 6.1. Operator Lifecycle Manager 워크플로



OLM은 OpenShift Container Platform 4.11에서 기본적으로 실행되므로 클러스터 관리자가 클러스터에서 실행되는 Operator를 설치, 업그레이드 및 액세스할 수 있도록 지원합니다. OpenShift Container Platform 웹 콘솔은 클러스터 관리자가 Operator를 설치할 수 있는 관리 화면을 제공하고, 클러스터에 제공되는 Operator 카탈로그를 사용할 수 있는 액세스 권한을 특정 프로젝트에 부여합니다.

개발자의 경우 분야별 전문가가 아니어도 셀프서비스 경험을 통해 데이터베이스, 모니터링, 빅 데이터 서비스의 인스턴스를 프로비저닝하고 구성할 수 있습니다. Operator에서 해당 지식을 제공하기 때문입니다.

CRD

OLM(Operator Lifecycle Manager)은 OLM Operator와 Catalog Operator의 두 Operator로 구성됩니다.

각 Operator는 OLM 프레임워크의 기반이 되는 CRD(사용자 정의 리소스 정의)를 관리합니다.

표 6.1. OLM 및 Catalog Operator에서 관리하는 CRD

리소스	짧은 이름	소유자	Description
ClusterServiceVersion(CSV)	**csv**	OLM	애플리케이션 메타데이터: 이름, 버전, 아이콘, 필수 리소스, 설치 등입니다.
InstallPlan	**ip**	카탈로그	CSV를 자동으로 설치하거나 업그레이드하기 위해 생성하는 계산된 리소스 목록입니다.
CatalogSource	**catsrc**	카탈로그	애플리케이션을 정의하는 CSV, CRD, 패키지의 리포지토리입니다.
서브스크립션	**sub**	카탈로그	패키지의 채널을 추적하여 CSV를 최신 상태로 유지하는 데 사용됩니다.

리소스	짧은 이름	소유자	Description
OperatorGroup	**og**	OLM	동일한 네임스페이스에 배포된 모든 Operator를 **OperatorGroup** 오브젝트로 구성하여 네임스페이스 목록 또는 클러스터 수준에서 CR(사용자 정의 리소스)을 조사합니다.

*또한 각 **Operator**는 다음 리소스를 생성합니다.*

표 6.2. OLM 및 Catalog Operator에서 생성하는 리소스

리소스	소유자
Deployments	OLM
ServiceAccounts	
(Cluster)Roles	
(Cluster)RoleBindings	
CRD(**CustomResourceDefinitions**)	카탈로그
ClusterServiceVersions	

OLM Operator

***CSV**에 지정된 필수 리소스가 클러스터에 제공되면 **OLM Operator**는 **CSV** 리소스에서 정의하는 애플리케이션을 배포합니다.*

***OLM Operator**는 필수 리소스 생성과는 관련이 없습니다. **CLI** 또는 **Catalog Operator**를 사용하여 이러한 리소스를 수동으로 생성하도록 선택할 수 있습니다. 이와 같은 분리를 통해 사용자는 애플리케이션에 활용하기 위해 선택하는 **OLM** 프레임워크의 양을 점차 늘리며 구매할 수 있습니다.*

***OLM Operator**에서는 다음 워크플로를 사용합니다.*

1. *네임스페이스에서 **CSV**(클러스터 서비스 버전)를 조사하고 해당 요구 사항이 충족되는지 확인합니다.*
2. *요구사항이 충족되면 **CSV**에 대한 설치 전략을 실행합니다.*

참고

설치 전략을 실행하기 위해서는 **CSV**가 **Operator group**의 활성 멤버여야 합니다.

Catalog Operator

Catalog Operator는 **CSV**(클러스터 서비스 버전) 및 **CSV**에서 지정하는 필수 리소스를 확인하고 설치합니다. 또한 채널에서 패키지 업데이트에 대한 카탈로그 소스를 조사하고 원하는 경우 사용 가능한 최신 버전으로 자동으로 업그레이드합니다.

채널에서 패키지를 추적하려면 원하는 패키지를 구성하는 **Subscription** 오브젝트, 채널, 업데이트를 가져오는 데 사용할 **CatalogSource** 오브젝트를 생성하면 됩니다. 업데이트가 확인되면 사용자를 대신하여 네임스페이스에 적절한 **InstallPlan** 오브젝트를 기록합니다.

Catalog Operator에서는 다음 워크플로를 사용합니다.

1. 클러스터의 각 카탈로그 소스에 연결합니다.
2. 사용자가 생성한 설치 계획 중 확인되지 않은 계획이 있는지 조사하고 있는 경우 다음을 수행합니다.
    a. 요청한 이름과 일치하는 **CSV**를 찾아 확인된 리소스로 추가합니다.
    b. 각 관리 또는 필수 **CRD**의 경우 **CRD**를 확인된 리소스로 추가합니다.
    c. 각 필수 **CRD**에 대해 이를 관리하는 **CSV**를 확인합니다.
3. 확인된 설치 계획을 조사하고 사용자의 승인에 따라 또는 자동으로 해당 계획에 대해 검색된 리소스를 모두 생성합니다.
4. 카탈로그 소스 및 서브스크립션을 조사하고 이에 따라 설치 계획을 생성합니다.

카탈로그 레지스트리

Catalog 레지스트리는 클러스터에서 생성할 CSV 및 CRD를 저장하고 패키지 및 채널에 대한 메타데이터를 저장합니다.

패키지 매니페스트는 패키지 ID를 CSV 세트와 연결하는 카탈로그 레지스트리의 항목입니다. 패키지 내에서 채널은 특정 CSV를 가리킵니다. CSV는 교체하는 CSV를 명시적으로 참조하므로 패키지 매니페스트는 Catalog Operator에 각 중간 버전을 거쳐 CSV를 최신 버전으로 업데이트하는 데 필요한 모든 정보를 제공합니다.

추가 리소스

- OLM(Operator Lifecycle Manager) 이해

6.33. OPENSHIFT SERVICE CA OPERATOR

목적

OpenShift Service CA Operator는 Kubernetes 서비스용 인증서 제공 및 관리.

프로젝트

openshift-service-ca-operator

6.34. VSPHERE PROBLEM DETECTOR OPERATOR

목적

vSphere Problem Detector Operator는 스토리지와 관련된 일반적인 설치 및 잘못된 구성 문제를 위해 vSphere에 배포된 클러스터를 확인합니다.

참고

vSphere Problem Detector Operator는 Cluster Storage Operator가 vSphere에 클러스터가 배포되었음을 탐지하는 경우에만 Cluster Storage Operator에 의해 시작됩니다.

설정

구성이 필요하지 않습니다.

참고

- Operator는 vSphere에서 OpenShift Container Platform 설치를 지원합니다.

- *Operator는 vsphere-cloud-credentials를 사용하여 vSphere와 통신합니다.*
- *Operator는 스토리지와 관련된 검사를 수행합니다.*

추가 리소스

- *vSphere Problem Detector Operator 사용*