OpenShift Container Platform 4.6

애플리케이션

OpenShift Container Platform에서 애플리케이션 생성 및 관리

Last Updated: 2023-04-08

OpenShift Container Platform 4.6 애플리케이션

OpenShift Container Platform에서 애플리케이션 생성 및 관리

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

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 실행 중인 사용자 프로비저닝 애플리케이션 인스턴스를 생성하고 관리하는 다양한 방법을 설명합니다. 여기에는 Open Service Broker API를 사용하여 프로젝트 및 프로비저닝 애플리케이션을 작업하는 방법이 포함됩니다.

차례

1장. 애플리케이션 빌드 개요

OpenShift Container Platform을 사용하면 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 애플리케이션을 생성, 편집, 삭제 및 관리할 수 있습니다.

1.1. 프로젝트 작업

프로젝트를 사용하면 별도로 애플리케이션을 구성하고 관리할 수 있습니다. OpenShift Container Platform에서 프로젝트를 생성, 보기, 삭제하는 등 전체 프로젝트 라이프사이클을 관리할 수 있습니다.

프로젝트를 생성한 후 개발자 화면을 사용하여 사용자에게 프로젝트에 대한 액세스 권한을 부여하거나 취소할 수 있습니다. 새 프로젝트의 자동 프로비저닝에 사용되는 프로젝트 템플릿을 생성하는 동안 프로젝트 구성 리소스를 편집할 수도 있습니다.

CLI를 사용하면 OpenShift Container Platform API에 요청을 가장하여 다른 사용자로 프로젝트를 생성할 수 있습니다. 새 프로젝트를 생성하도록 요청할 때 OpenShift Container Platform은 끝점을 사용하여 사용자 지정 템플릿에 따라 프로젝트를 프로비저닝합니다. 클러스터 관리자는 인증된 사용자 그룹이 새 프로젝트를 자체 프로비저닝하지 못하도록 할 수 있습니다.

1.2. 애플리케이션 작업

1.2.1. 애플리케이션 생성

애플리케이션을 생성하려면 프로젝트를 생성하거나 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있어야 합니다. 웹 콘솔의 개발자 화면,설치된 Operator 또는 OpenShift Container Platform CLI 를 사용하여 애플리케이션을 생성할 수 있습니다. Git, JAR 파일, devfile 또는 개발자 카탈로그에서 프로젝트에 추가할 애플리케이션을 소싱할 수 있습니다.

OpenShift Container Platform CLI를 사용하여 애플리케이션을 생성하는 소스 또는 바이너리 코드, 이미지 및 템플릿이 포함된 구성 요소를 사용할 수도 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 클러스터 관리자가 설치한 Operator에서 애플리케이션을 생성할 수 있습니다.

1.2.2. 애플리케이션 유지 관리

애플리케이션을 생성한 후 웹 콘솔을 사용하여 프로젝트 또는 애플리케이션 지표를 모니터링할 수 있습니다. 웹 콘솔을 사용하여 애플리케이션을 편집하거나 삭제할 수도 있습니다. 애플리케이션이 실행 중이면 모든 애플리케이션 리소스가 사용되지 않습니다. 클러스터 관리자는 이러한 확장 가능한 리소스를 유휴 상태로 설정하여 리소스 사용을 줄일 수 있습니다.

1.2.3. 애플리케이션 배포

Deployment 또는 **DeploymentConfig** 오브젝트 를 사용하여 애플리케이션을 배포하고 웹 콘솔에서 관리할 수 있습니다. 변경 중 또는 애플리케이션을 업그레이드하는 동안 다운타임을 줄이는 데 도움이 되는 배포 전략을 생성할 수 있습니다.

OpenShift Container Platform 클러스터에 애플리케이션 및 서비스 배포를 간소화하는 소프트웨어 패키지 관리자인 Helm 을 사용할 수도 있습니다.

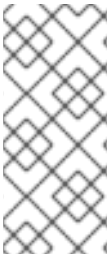
1.3. RED HAT MARKETPLACE 사용

Red Hat Marketplace 는 퍼블릭 클라우드 및 온프레미스에서 실행되는 컨테이너 기반 환경에 대해 인증된 소프트웨어를 검색하고 액세스할 수 있는 오픈 클라우드 마켓플레이스입니다.

2장. 프로젝트

2.1. 노드 작업

사용자 커뮤니티는 *프로젝트*를 통해 다른 커뮤니티와 별도로 콘텐츠를 구성하고 관리할 수 있습니다.



참고

openshift- 및 **kube-**로 시작하는 프로젝트는 기본 프로젝트입니다. 이러한 프로젝트는 Pod 및 기타 인프라 구성 요소로 실행되는 클러스터 구성 요소를 호스팅합니다. 따라서 OpenShift Container Platform에서는 **oc new-project** 명령을 사용하여 **openshift-** 또는 **kube-**로 시작하는 프로젝트를 생성할 수 없습니다. 클러스터 관리자는 **oc adm new-project** 명령을 사용하여 이러한 프로젝트를 생성할 수 있습니다.

참고

기본 네임스페이스(**default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, **openshift**) 중 하나에서 생성된 Pod에는 SCC를 할당할 수 없습니다. 이러한 네임스페이스는 Pod 또는 서비스를 실행하는 데 사용할 수 없습니다.

2.1.1. 웹 콘솔을 사용하여 프로젝트 생성

클러스터 관리자가 허용한 경우 새 프로젝트를 생성할 수 있습니다.

참고

openshift- 및 **kube-**로 시작하는 프로젝트는 OpenShift Container Platform에서 중요한 것으로 간주합니다. 따라서 OpenShift Container Platform에서는 웹 콘솔을 사용하여 **openshift-**로 시작하는 프로젝트를 생성할 수 없습니다.

참고

기본 네임스페이스(**default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, **openshift**) 중 하나에서 생성된 Pod에는 SCC를 할당할 수 없습니다. 이러한 네임스페이스는 Pod 또는 서비스를 실행하는 데 사용할 수 없습니다.

프로세스

1. 홈 → 프로젝트로 이동합니다.
2. **Create Project**를 클릭합니다.
3. 프로젝트 세부 정보를 입력합니다.
4. **Create**를 클릭합니다.

2.1.2. 웹 콘솔의 개발자 화면을 사용하여 프로젝트 만들기

OpenShift Container Platform 웹 콘솔의 개발자 화면을 사용하여 클러스터에 프로젝트를 생성할 수 있습니다.

참고

openshift- 및 **kube-**로 시작하는 프로젝트는 OpenShift Container Platform에서 중요한 것으로 간주합니다. 따라서 OpenShift Container Platform에서는 개발자 화면을 사용하여 **openshift-** 또는 **kube-**로 시작하는 프로젝트를 생성할 수 없습니다. 클러스터 관리자는 **oc adm new-project** 명령을 사용하여 이러한 프로젝트를 생성할 수 있습니다.

참고

기본 네임스페이스(**default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, **openshift**) 중 하나에서 생성된 Pod에는 SCC를 할당할 수 없습니다. 이러한 네임스페이스는 Pod 또는 서비스를 실행하는 데 사용할 수 없습니다.

사전 요구 사항

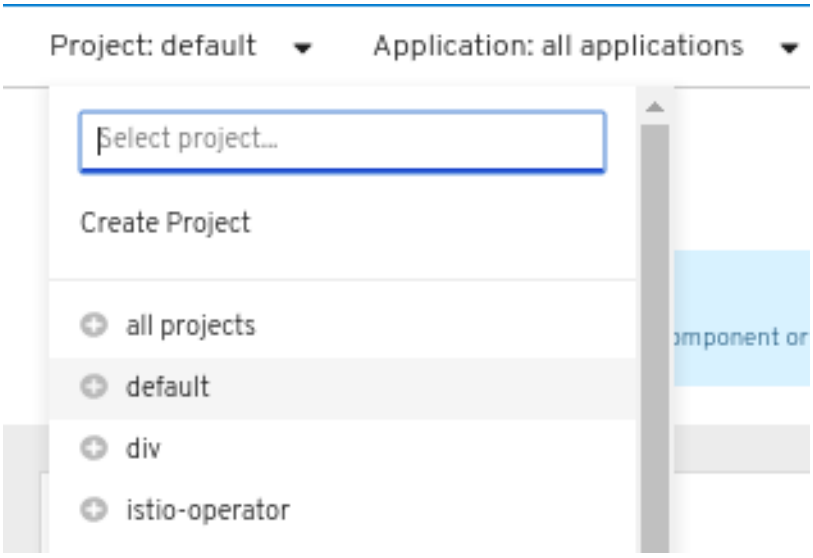
- OpenShift Container Platform에서 프로젝트, 애플리케이션 및 기타 워크로드를 생성할 적절한 역할과 권한이 있는지 확인합니다.

프로세스

다음과 같이 개발자 화면을 사용하여 프로젝트를 생성할 수 있습니다.

1. 프로젝트 드롭다운 메뉴를 클릭하여 사용 가능한 전체 프로젝트 목록을 확인합니다. 프로젝트 생성을 선택합니다.

   그림 **2.1.** 프로젝트 생성

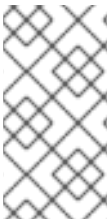


2. 프로젝트 생성 대화 상자에서 이름 필드에 **myproject**와 같은 고유한 이름을 입력합니다.
3. 선택 사항: 프로젝트의 **Display Name** (표시 이름) 및 **Description** (설명) 세부 정보를 추가합니다.
4. 생성을 클릭합니다.
5. 왼쪽 탐색 패널을 사용하여 프로젝트 보기로 이동하여 프로젝트 대시보드를 확인합니다.
6. 선택 사항:
   - 클러스터의 프로젝트를 모두 나열하려면 화면 위쪽에 있는 프로젝트 드롭다운 메뉴를 사용하여 모든 프로젝트를 선택합니다.

- 프로젝트 세부 정보를 확인하려면 세부 정보 탭을 사용합니다.
- 프로젝트에 대한 적절한 권한이 있는 경우 프로젝트 액세스 탭을 사용하여 *관리자*를 제공하거나 취소하고 프로젝트에 대한 권한을 *편집* 및 *확인*할 수 있습니다.

2.1.3. CLI를 사용하여 프로젝트 생성

클러스터 관리자가 허용한 경우 새 프로젝트를 생성할 수 있습니다.



참고

openshift- 및 **kube-**로 시작하는 프로젝트는 OpenShift Container Platform에서 중요한 것으로 간주합니다. 따라서 OpenShift Container Platform에서는 **oc new-project** 명령을 사용하여 **openshift-** 또는 **kube-**로 시작하는 프로젝트를 생성할 수 없습니다. 클러스터 관리자는 **oc adm new-project** 명령을 사용하여 이러한 프로젝트를 생성할 수 있습니다.

참고

기본 네임스페이스(**default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, **openshift**) 중 하나에서 생성된 Pod에는 SCC를 할당할 수 없습니다. 이러한 네임스페이스는 Pod 또는 서비스를 실행하는 데 사용할 수 없습니다.

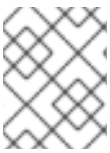
프로세스

- 다음을 실행합니다.

```
$ oc new-project <project_name> \
    --description="<description>" --display-name="<display_name>"
```

예를 들면 다음과 같습니다.

```
$ oc new-project hello-openshift \
    --description="This is an example project" \
    --display-name="Hello OpenShift"
```



참고

생성할 수 있는 프로젝트 수는 시스템 관리자가 제한할 수 있습니다. 제한에 도달한 후 새 프로젝트를 생성하려면 기존 프로젝트를 삭제해야 할 수 있습니다.

2.1.4. 웹 콘솔을 사용하여 프로젝트 보기

프로세스

1. 홈 → 프로젝트로 이동합니다.
2. 확인할 프로젝트를 선택합니다.
   이 페이지에서 **Workloads(**워크로드)를 클릭하여 프로젝트의 워크로드를 확인합니다.

2.1.5. CLI를 사용하여 프로젝트 보기

프로젝트를 볼 때는 권한 부여 정책에 따라 볼 수 있는 액세스 권한이 있는 프로젝트만 볼 수 있습니다.

프로세스

1. 프로젝트 목록을 보려면 다음을 실행합니다.

```
$ oc get projects
```

2. CLI 작업을 위해 현재 프로젝트에서 다른 프로젝트로 변경할 수 있습니다. 그러면 지정된 프로젝트가 프로젝트 범위의 콘텐츠를 조작하는 모든 후속 작업에서 사용됩니다.

```
$ oc project <project_name>
```

2.1.6. 개발자 화면을 사용하여 프로젝트에 액세스 권한 제공

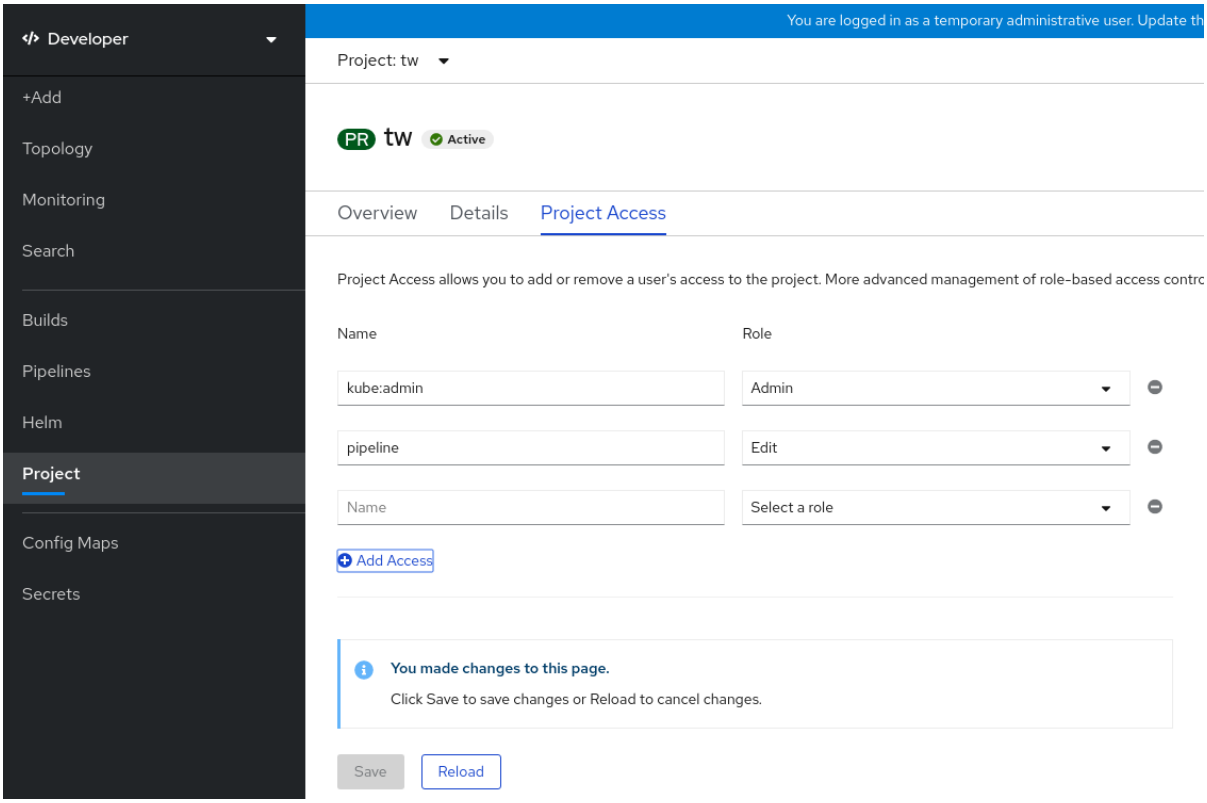
개발자 화면의 프로젝트 보기를 사용하여 프로젝트에 대한 액세스 권한을 부여하거나 취소할 수 있습니다.

프로세스

프로젝트에 사용자를 추가하고 관리자, 편집 또는 보기 액세스 권한을 제공하려면 다음을 수행합니다.

1. 개발자 화면에서 프로젝트 보기로 이동합니다.
2. 프로젝트 페이지에서 프로젝트 액세스 탭을 선택합니다.
3. 액세스 추가를 클릭하여 기본 권한에 새 권한 행을 추가합니다.

그림 2.2. 프로젝트 권한



4. 사용자 이름을 입력하고 역할 선택 드롭다운 목록을 클릭하고 적절한 역할을 선택합니다.
5. 저장을 클릭하여 새 권한을 추가합니다.

또는 다음을 수행할 수도 있습니다.

- 역할 선택 드롭다운 목록을 사용하여 기존 사용자의 액세스 권한을 수정합니다.
- 액세스 제거 아이콘을 사용하여 프로젝트에 대한 기존 사용자의 액세스 권한을 완전히 제거합니다.

참고

고급 역할 기반 액세스 제어는 관리자 화면의 역할 및 역할 바인딩 보기에서 관리됩니다.

2.1.7. 프로젝트에 추가

프로세스

1. 웹 콘솔 탐색 메뉴 위쪽의 컨텍스트 선택기에서 개발자를 선택합니다.
2. +추가를 클릭합니다.
3. 페이지 위쪽에서 추가할 프로젝트 이름을 선택합니다.
4. 프로젝트에 추가하는 방법을 클릭한 다음 워크플로를 따릅니다.

2.1.8. 웹 콘솔을 사용하여 프로젝트 상태 확인

프로세스

1. 홈 → 프로젝트로 이동합니다.
2. 상태를 확인할 프로젝트를 선택합니다.

2.1.9. CLI를 사용하여 프로젝트 상태 확인

프로세스

1. 다음을 실행합니다.

```
$ oc status
```

이 명령은 현재 프로젝트에 대한 고급 개요에 해당 구성 요소와 그 관계를 제공합니다.

2.1.10. 웹 콘솔을 사용하여 프로젝트 삭제

OpenShift Container Platform 웹 콘솔을 사용하여 프로젝트를 삭제할 수 있습니다.

참고

프로젝트를 삭제할 수 있는 권한이 없는 경우 프로젝트 삭제 옵션이 제공되지 않습니다.

프로세스

1. 홈 → 프로젝트로 이동합니다.
2. 프로젝트 목록에서 삭제할 프로젝트를 찾습니다.

3. 프로젝트 목록 맨 오른쪽에 있는 옵션 메뉴  에서 프로젝트 삭제를 선택합니다.

4. 프로젝트 삭제 창이 열리면 삭제할 프로젝트 이름을 필드에 입력합니다.

5. 삭제를 클릭합니다.

2.1.11. CLI를 사용하여 프로젝트 삭제

프로젝트를 삭제하면 서버에서 프로젝트 상태를 활성에서 종료로 업데이트합니다. 그런 다음 서버는 프로젝트를 완전히 제거하기 전에 상태가 종료인 프로젝트의 모든 콘텐츠를 지웁니다. 프로젝트 상태가 종료인 동안에는 프로젝트에 새 콘텐츠를 추가할 수 없습니다. CLI 또는 웹 콘솔에서 프로젝트를 삭제할 수 있습니다.

프로세스

1. 다음을 실행합니다.

```
$ oc delete project <project_name>
```

2.2. 다른 사용자로 프로젝트 생성

가장 기능을 사용하면 다른 사용자로 프로젝트를 생성할 수 있습니다.

2.2.1. API 가장

OpenShift Container Platform API에 대한 요청을 다른 사용자가 보낸 것처럼 작동하도록 구성할 수 있습니다. 자세한 내용은 Kubernetes 설명서의 사용자 가장을 참조하십시오.

2.2.2. 프로젝트를 만들 때 사용자 가장

프로젝트 요청을 생성할 때 다른 사용자로 가장할 수 있습니다. **system:authenticated:oauth**는 프로젝트 요청을 생성할 수 있는 유일한 부트스트랩 그룹이므로 이 그룹을 가장해야 합니다.

프로세스

- 다른 사용자를 대신하여 프로젝트 요청을 생성하려면 다음을 실행합니다.

```
$ oc new-project <project> --as=<user> \
    --as-group=system:authenticated --as-group=system:authenticated:oauth
```

2.3. 프로젝트 생성 구성

OpenShift Container Platform에서 *프로젝트*는 관련 오브젝트를 그룹화하고 격리하는 데 사용됩니다. 웹 콘솔 또는 **oc new-project** 명령을 사용하여 새 프로젝트를 생성하라는 요청이 생성되면 OpenShift Container Platform의 끝점을 사용하여 사용자 지정할 수 있는 템플릿에 따라 프로젝트를 프로비저닝합니다.

클러스터 관리자는 개발자 및 서비스 계정이 자체 프로젝트를 생성하거나 *자체 프로비저닝*하는 방법을 허용하고 구성할 수 있습니다.

2.3.1. 프로젝트 생성 정보

OpenShift Container Platform API 서버는 클러스터의 프로젝트 구성 리소스에서 **projectRequestTemplate** 매개변수로 식별하는 프로젝트 템플릿을 기반으로 새 프로젝트를 자동으로 프로비저닝합니다. 매개변수가 정의되지 않은 경우 API 서버는 요청된 이름으로 프로젝트를 생성하는 기본 템플릿을 생성하고 요청하는 사용자에게 해당 프로젝트의 **admin** 역할을 할당합니다.

프로젝트 요청을 제출하면 API에서 다음 매개변수를 템플릿으로 대체합니다.

표 2.1. 기본 프로젝트 템플릿 매개변수

매개변수	설명
PROJECT_NAME	프로젝트 이름입니다. 필수 항목입니다.
PROJECT_DISPLAYNAME	프로젝트의 표시 이름입니다. 비어 있을 수 있습니다.
PROJECT_DESCRIPTION	프로젝트에 대한 설명입니다. 비어 있을 수 있습니다.
PROJECT_ADMIN_USER	관리 사용자의 사용자 이름입니다.
PROJECT_REQUESTING_USER	요청하는 사용자의 사용자 이름입니다.

API에 대한 액세스 권한은 **self-provisioner** 역할 및 **self-provisioner** 클러스터 역할 바인딩을 가진 개발자에게 부여됩니다. 이 역할은 기본적으로 인증된 모든 개발자에게 제공됩니다.

2.3.2. 새 프로젝트의 템플릿 수정

클러스터 관리자는 사용자 정의 요구 사항을 사용하여 새 프로젝트를 생성하도록 기본 프로젝트 템플릿을 수정할 수 있습니다.

사용자 정의 프로젝트 템플릿을 만들려면:

프로세스

1. **cluster-admin** 권한이 있는 사용자로 로그인합니다.
2. 기본 프로젝트 템플릿을 생성합니다.

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 텍스트 편집기를 사용하여 오브젝트를 추가하거나 기존 오브젝트를 수정하여 생성된 **template.yaml** 파일을 수정합니다.
4. 프로젝트 템플릿은 **openshift-config** 네임스페이스에서 생성해야 합니다. 수정된 템플릿을 불러옵니다.

```
$ oc create -f template.yaml -n openshift-config
```

5. 웹 콘솔 또는 CLI를 사용하여 프로젝트 구성 리소스를 편집합니다.

- 웹 콘솔에 액세스:
  i. 관리 → 클러스터 설정으로 이동합니다.
  ii. 전역 구성을 클릭하여 모든 구성 리소스를 확인합니다.
  iii. 프로젝트 항목을 찾아 **YAML** 편집을 클릭합니다.
- CLI 사용:
  i. 다음과 같이 **project.config.openshift.io/cluster** 리소스를 편집합니다.

```
$ oc edit project.config.openshift.io/cluster
```

6. **projectRequestTemplate** 및 **name** 매개변수를 포함하도록 **spec** 섹션을 업데이트하고 업로드된 프로젝트 템플릿의 이름을 설정합니다. 기본 이름은 **project-request**입니다.

사용자 정의 프로젝트 템플릿이 포함된 프로젝트 구성 리소스

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. 변경 사항을 저장한 후 새 프로젝트를 생성하여 변경 사항이 성공적으로 적용되었는지 확인합니다.

2.3.3. 프로젝트 자체 프로비저닝 비활성화

인증된 사용자 그룹이 새 프로젝트를 자체 프로비저닝하지 못하도록 할 수 있습니다.

프로세스

1. **cluster-admin** 권한이 있는 사용자로 로그인합니다.
2. 다음 명령을 실행하여 **self-provisioners** 클러스터 역할 바인딩 사용을 확인합니다.

```
$ oc describe clusterrolebinding.rbac self-provisioners
```

출력 예

```
Name:  self-provisioners
Labels:  <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name    Namespace
  ---- ----    ---------
  Group system:authenticated:oauth
```

self-provisioners 섹션의 제목을 검토합니다.

3. 그룹 **system:authenticated:oauth**에서 **self-provisioner** 클러스터 역할을 제거합니다.

   - **self-provisioners** 클러스터 역할 바인딩에서 **self-provisioner** 역할을 **system:authenticated:oauth** 그룹에만 바인딩하는 경우 다음 명령을 실행합니다.

     ```
     $ oc patch clusterrolebinding.rbac self-provisioners -p '{"subjects": null}'
     ```

   - **self-provisioners** 클러스터 역할 바인딩에서 **self-provisioner** 역할을 **system:authenticated:oauth** 그룹 이외에도 추가 사용자, 그룹 또는 서비스 계정에 바인딩하는 경우 다음 명령을 실행합니다.

     ```
     $ oc adm policy \
         remove-cluster-role-from-group self-provisioner \
         system:authenticated:oauth
     ```

4. 역할이 자동으로 업데이트되지 않도록 **self-provisioners** 클러스터 역할 바인딩을 편집합니다. 역할이 자동으로 업데이트되면 클러스터 역할이 기본 상태로 재설정됩니다.

   - CLI를 사용하여 역할 바인딩을 업데이트하려면 다음을 수행합니다.

     i. 다음 명령을 실행합니다.

     ```
     $ oc edit clusterrolebinding.rbac self-provisioners
     ```

     ii. 표시된 역할 바인딩에서 다음 예와 같이 **rbac.authorization.kubernetes.io/autoupdate** 매개변수 값을 **false**로 설정합니다.

     ```
     apiVersion: authorization.openshift.io/v1
     kind: ClusterRoleBinding
     metadata:
       annotations:
         rbac.authorization.kubernetes.io/autoupdate: "false"
       ...
     ```

   - 단일 명령을 사용하여 역할 바인딩을 업데이트하려면 다음을 실행합니다.

     ```
     $ oc patch clusterrolebinding.rbac self-provisioners -p '{ "metadata": { "annotations": {
     "rbac.authorization.kubernetes.io/autoupdate": "false" } } }'
     ```

5. 인증된 사용자로 로그인하여 프로젝트를 더 이상 자체 프로비저닝할 수 없는지 확인합니다.

   ```
   $ oc new-project test
   ```

   출력 예

   ```
   Error from server (Forbidden): You may not request a new project via this API.
   ```

   조직과 관련된 더 유용한 지침을 제공하도록 이 프로젝트 요청 메시지를 사용자 정의하는 것이 좋습니다.

2.3.4. 프로젝트 요청 메시지 사용자 정의

프로젝트를 자체 프로비저닝할 수 없는 개발자 또는 서비스 계정이 웹 콘솔 또는 CLI를 사용하여 프로젝트 생성을 요청하면 기본적으로 다음과 같은 오류 메시지가 반환됩니다.

```
You may not request a new project via this API.
```

클러스터 관리자는 이 메시지를 사용자 지정할 수 있습니다. 조직과 관련된 새 프로젝트를 요청하는 방법에 대한 추가 지침을 제공하려면 업데이트하는 것이 좋습니다. 예를 들면 다음과 같습니다.

- 프로젝트를 요청하려면 **projectname@example.com**을 통해 시스템 관리자에게 문의하십시오.
- 새 프로젝트를 요청하려면 **https://internal.example.com/openshift-project-request**에 있는 프로젝트 요청 양식을 작성합니다.

프로젝트 요청 메시지를 사용자 지정하려면 다음을 수행합니다.

프로세스

1. 웹 콘솔 또는 CLI를 사용하여 프로젝트 구성 리소스를 편집합니다.
   - 웹 콘솔에 액세스:
     i. 관리 → 클러스터 설정으로 이동합니다.
     ii. 전역 구성을 클릭하여 모든 구성 리소스를 확인합니다.
     iii. 프로젝트 항목을 찾아 **YAML** 편집을 클릭합니다.
   - CLI 사용:
     i. **cluster-admin** 권한이 있는 사용자로 로그인합니다.
     ii. 다음과 같이 **project.config.openshift.io/cluster** 리소스를 편집합니다.

     ```
     $ oc edit project.config.openshift.io/cluster
     ```

2. **projectRequestMessage** 매개변수를 포함하도록 **spec** 섹션을 업데이트하고 해당 값을 사용자 정의 메시지로 설정합니다.

   사용자 정의 프로젝트 요청 메시지가 포함된 프로젝트 구성 리소스

   ```
   apiVersion: config.openshift.io/v1
   kind: Project
   metadata:
     ...
   spec:
     projectRequestMessage: <message_string>
   ```

   예를 들면 다음과 같습니다.

   ```
   apiVersion: config.openshift.io/v1
   kind: Project
   metadata:
     ...
   ```

```
spec:
  projectRequestMessage: To request a project, contact your system administrator at
projectname@example.com.
```

3. 변경 사항을 저장한 후 프로젝트를 자체 프로비저닝하여 변경 사항이 성공적으로 적용되었는지 확인할 수 없는 개발자 또는 서비스 계정으로 새 프로젝트를 생성합니다.

3장. 애플리케이션 라이프사이클 관리

3.1. 개발자 화면을 사용하여 애플리케이션 생성

웹 콘솔의 개발자 화면은 +추가 보기에서 애플리케이션 및 관련 서비스를 생성하고 OpenShift Container Platform에 배포할 수 있는 다음 옵션을 제공합니다.

- **Git**에서: OpenShift Container Platform에서 애플리케이션을 생성, 빌드, 배포하려면 이 옵션을 사용하여 Git 리포지토리의 기존 코드베이스를 가져옵니다.
- 컨테이너 이미지: 이미지 스트림 또는 레지스트리의 기존 이미지를 사용하여 OpenShift Container Platform에 배포합니다.
- **Dockerfile**에서: Git 리포지토리에서 dockerfile을 가져와 애플리케이션을 빌드하고 배포합니다.
- **YAML**: 편집기를 사용하여 YAML 또는 JSON 정의를 추가하여 리소스를 생성하고 수정합니다.
- 카탈로그에서: 개발자 카탈로그를 살펴보고 이미지 빌더에 필요한 애플리케이션, 서비스 또는 소스를 선택하고 프로젝트에 추가합니다.
- 데이터베이스: 필요한 데이터베이스 서비스를 선택하고 애플리케이션에 추가하려면 개발자 카탈로그를 참조하십시오.
- **Operator** 지원: 개발자 카탈로그를 탐색하여 Operator 관리 서비스를 선택하고 배포합니다.
- **Helm** 차트: 개발자 카탈로그를 살펴보고 필요한 Helm 차트를 선택하여 애플리케이션 및 서비스 배포를 간소화합니다.

Pipeline,**Event Source**, **Import Virtual Machines** 와 같은 특정 옵션은 각각 OpenShift Pipelines Operator, OpenShiftServerless Operator 및 OpenShift Virtualization Operator 가 설치된 경우에만 표시됩니다.

3.1.1. 사전 요구 사항

개발자 화면을 사용하여 애플리케이션을 생성하려면 다음 조건을 충족해야 합니다.

- 웹 콘솔에 로그인 했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

서버리스 애플리케이션을 생성하려면 위 사전 요구 사항에 다음 조건이 추가됩니다.

- OpenShift Serverless Operator를 설치 했습니다.
- **knative-serving** 네임스페이스에 **KnativeServing** 리소스를 생성 했습니다.

3.1.2. 샘플 애플리케이션 생성

개발자 화면의 +추가 작업에서 기본 샘플 애플리케이션을 사용하면 애플리케이션을 빠르게 생성, 빌드, 배포할 수 있습니다.

다음 절차에서는 개발자 화면의 샘플 옵션을 사용하여 샘플 애플리케이션을 생성하는 방법을 설명합니다.

프로세스

1. +추가 보기에서 샘플을 클릭하여 샘플 페이지를 확인합니다.

2. 샘플 페이지에서 사용 가능한 샘플 애플리케이션 중 하나를 선택하여 샘플 애플리케이션 생성 양식을 확인합니다.

3. 샘플 애플리케이션 생성 양식에서 다음을 수행합니다.

   - 이름 필드에는 기본적으로 배포 이름이 표시됩니다. 필요에 따라 이 이름을 수정할 수 있습니다.

   - 빌더 이미지 버전에는 빌더 이미지가 기본적으로 선택되어 있습니다. 빌더 이미지 버전 드롭다운 목록을 사용하여 이 이미지 버전을 수정할 수 있습니다.

   - 샘플 Git 리포지토리 URL이 기본적으로 추가됩니다.

4. 생성을 클릭하여 샘플 애플리케이션을 생성합니다. 샘플 애플리케이션의 빌드 상태가 토폴로지 보기에 표시됩니다. 샘플 애플리케이션이 생성되면 애플리케이션에 추가된 배포를 볼 수 있습니다.

3.1.3. Git에서 코드베이스를 가져와 애플리케이션 생성

개발자 화면에서는 GitHub의 기존 코드베이스를 사용하여 OpenShift Container Platform에 애플리케이션을 생성, 빌드, 배포할 수 있습니다.

다음 절차에서는 개발자 화면의 **Git**에서 옵션을 통해 애플리케이션을 생성합니다.

프로세스

1. +추가 보기에서 **Git**에서를 클릭하여 **Git**에서 가져오기 양식을 확인합니다.

2. **Git** 섹션에서 애플리케이션을 생성하는 데 사용할 코드베이스의 Git 리포지토리 URL을 입력합니다. 예를 들어 이 샘플 Node.js 애플리케이션의 URL **https://github.com/sclorg/nodejs-ex**를 입력합니다. 그런 다음 URL을 검증합니다.

3. 선택 사항: **Show Advanced Git Options**(고급 **Git** 옵션 표시)를 클릭하여 다음과 같은 세부 정보를 추가할 수 있습니다.

   - **Git** 참조: 애플리케이션을 빌드하는 데 사용할 특정 분기의 코드, 태그 또는 커밋을 가리킵니다.

   - 컨텍스트 디렉터리: 애플리케이션을 빌드하는 데 사용할 애플리케이션 소스 코드의 하위 디렉터리를 지정합니다.

   - 소스 시크릿: 프라이빗 리포지토리에서 소스 코드를 가져올 수 있는 자격 증명이 포함된 시크릿 이름을 생성합니다.

4. 빌더 섹션에서 URL이 검증되면 적절한 빌더 이미지가 탐지되어 별표로 표시된 후 자동으로 선택됩니다. **https://github.com/sclorg/nodejs-ex** Git URL의 경우 Node.js 빌더 이미지가 기본적으로 선택됩니다. 빌더 이미지가 자동으로 탐지되지 않으면 빌더 이미지를 선택합니다. 필요한 경우 빌더 이미지 버전 드롭다운 목록을 사용하여 버전을 변경할 수 있습니다.

5. 일반 섹션에서 다음을 수행합니다.

   a. 애플리케이션 필드에서 애플리케이션 그룹화에 대한 고유 이름을 입력합니다(예: **myapp**). 애플리케이션 이름이 네임스페이스에서 고유해야 합니다.

   b. 기존 애플리케이션이 없는 경우 이 애플리케이션에 대해 생성된 리소스를 확인하는 이름 필

드는 Git 리포지토리 URL에 따라 자동으로 채워집니다. 기존 애플리케이션이 있는 경우에는 기존 애플리케이션 내에 구성 요소를 배포하거나 새 애플리케이션을 생성하거나 구성 요소를 할당하지 않은 상태로 유지하도록 선택할 수 있습니다.

참고

리소스 이름은 네임스페이스에서 고유해야 합니다. 오류가 발생하면 리소스 이름을 수정합니다.

6. 리소스 섹션에서 다음 옵션을 선택합니다.
   - 배포: 일반 Kubernetes 형식으로 애플리케이션을 생성합니다.
   - 배포 구성: OpenShift 스타일 애플리케이션을 생성합니다.
   - **Knative** 서비스: 마이크로 서비스를 생성합니다.

참고

Knative 서비스 옵션은 **Serverless Operator**가 클러스터에 설치된 경우에만 **Git**에서 가져오기 양식에 표시됩니다. 자세한 내용은 OpenShift Serverless 설치 설명서를 참조하십시오.

7. 파이프라인 섹션에서 파이프라인 추가를 선택한 다음 파이프라인 시각화 표시를 클릭하여 애플리케이션의 파이프라인을 확인합니다.
8. 공개적으로 제공되는 URL을 사용하여 애플리케이션에 액세스할 수 있도록 고급 옵션 섹션의 애플리케이션의 경로 만들기가 기본적으로 선택되어 있어 습니다. 공개 경로에 애플리케이션을 노출하지 않으려면 확인란을 지우면 됩니다.
9. 선택 사항: 다음 고급 옵션을 사용하여 애플리케이션을 추가로 사용자 지정할 수 있습니다.

라우팅

라우팅 링크를 클릭하면 다음 작업을 수행할 수 있습니다.

- 경로의 호스트 이름을 사용자 지정합니다.
- 라우터에서 감시하는 경로를 지정합니다.
- 드롭다운 목록에서 트래픽의 대상 포트를 선택합니다.
- **Secure Route** 확인란을 선택하여 경로를 보호합니다. 필요한 TLS 종료 유형을 선택하고 해당 드롭다운 목록에서 안전하지 않은 트래픽에 대한 정책을 설정합니다.

참고

서버리스 애플리케이션의 경우 Knative 서비스는 위의 모든 라우팅 옵션을 관리합니다. 그러나 필요한 경우 트래픽에 대한 대상 포트를 사용자 지정할 수 있습니다. 대상 포트를 지정하지 않으면 기본 포트 **8080**이 사용됩니다.

상태 점검

애플리케이션에 준비 상태, 활성 상태, 시작 프로브를 추가하려면 상태 점검 링크를 클릭합니다. 모든 프로브에는 기본 데이터가 미리 채워져 있습니다. 기본 데이터가 포함된 프로브를 추가하거나 필요에 따라 사용자 지정할 수 있습니다.

상태 프로브를 사용자 지정하려면 다음을 수행합니다.

- 필요한 경우 준비 상태 프로브 추가를 클릭하여 컨테이너에서 요청을 처리할 준비가 되었는지 확인하도록 매개변수를 수정하고 확인 표시를 선택하여 프로브를 추가합니다.
- 필요한 경우 활성 상태 프로브 추가를 클릭하여 컨테이너가 아직 실행되고 있는지 확인하도록 매개변수를 수정하고 확인 표시를 선택하여 프로브를 추가합니다.
- 필요한 경우 시작 프로브 추가를 클릭하여 컨테이너 내 애플리케이션이 시작되었는지 확인하도록 매개변수를 수정하고 확인 표시를 선택하여 프로브를 추가합니다.
  각 프로브의 드롭다운 목록에서 요청 유형을 **HTTP GET**, 컨테이너 명령 또는 **TCP** 소켓으로 지정할 수 있습니다. 선택한 요청 유형에 따라 양식이 변경됩니다. 그런 다음 기타 매개변수(예: 프로브의 성공 및 실패 임계값, 컨테이너를 시작한 후 첫 번째 프로브를 수행할 때까지의 시간(초), 프로브 빈도, 시간 제한 값)에 대한 기본값을 수정할 수 있습니다.

빌드 구성 및 배포

빌드 구성 및 배포 링크를 클릭하여 해당 구성 옵션을 확인합니다. 일부 옵션은 기본적으로 선택되어 있습니다. 필요한 트리거 및 환경 변수를 추가하여 추가로 사용자 지정할 수 있습니다.
서버리스 애플리케이션의 경우 DeploymentConfig 대신 Knative 구성 리소스에서 배포에 필요한 상태를 유지 관리하므로 배포 옵션이 표시되지 않습니다.

스케일링

처음에 배포할 애플리케이션의 Pod 수 또는 인스턴스 수를 정의하려면 스케일링 링크를 클릭합니다. Knative 서비스를 생성하는 경우 다음 설정을 구성할 수도 있습니다.

- 자동 스케일러로 설정할 수 있는 Pod 수 상한값과 하한값을 설정합니다. 하한값을 지정하지 않는 경우 기본값은 0입니다.
- 애플리케이션 인스턴스에 한 번에 필요한 동시 요청 수에 대한 소프트 제한을 정의합니다. 자동 스케일링을 위한 구성에 사용하는 것이 좋습니다. 지정하지 않으면 클러스터 구성에 지정된 값이 사용됩니다.
- 애플리케이션 인스턴스에 한 번에 허용된 동시 요청 수에 대한 하드 제한을 정의합니다. 이는 리버전 템플릿에 구성됩니다. 지정하지 않으면 기본값은 클러스터 구성에 지정된 값입니다.

리소스 제한

컨테이너 실행 시 컨테이너에서 사용하도록 보장하거나 허용하는 **CPU** 및 메모리 리소스의 양을 설정하려면 리소스 제한 링크를 클릭합니다.

라벨

라벨 링크를 클릭하여 애플리케이션에 사용자 정의 라벨을 추가합니다.

1. 생성을 클릭하여 애플리케이션을 생성하고 토폴로지 보기에서 빌드 상태를 확인합니다.

3.1.4. 추가 리소스

- OpenShift Serverless의 Knative 라우팅 설정에 대한 자세한 내용은 라우팅 을 참조하십시오.
- OpenShift Serverless의 Knative 자동 스케일링 설정에 대한 자세한 내용은 자동 스케일링 을 참조하십시오.
- OpenShift Serverless의 도메인 매핑 설정에 대한 자세한 내용은 Knative 서비스의 사용자 정의 도메인 구성을 참조하십시오.

3.2. 설치된 OPERATOR에서 애플리케이션 생성

*Operator*는 Kubernetes 애플리케이션을 패키징, 배포 및 관리하는 방법입니다. 클러스터 관리자가 설치한 Operator를 사용하여 OpenShift Container Platform에서 애플리케이션을 생성할 수 있습니다.

이 가이드에서는 개발자에게 OpenShift Container Platform 웹 콘솔을 사용하여 설치된 Operator에서 애플리케이션을 생성하는 예제를 보여줍니다.

추가 리소스

- Operator의 작동 방식 및 Operator Lifecycle Manager를 OpenShift Container Platform에 통합하는 방법에 대한 자세한 내용은 Operator 가이드를 참조하십시오.

3.2.1. 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 4.6 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-etcd**라는 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 셀프 서비스 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.

3.3. CLI를 사용하여 애플리케이션 생성

OpenShift Container Platform CLI를 사용하여 소스 또는 바이너리 코드, 이미지, 템플릿이 포함된 구성 요소에서 OpenShift Container Platform 애플리케이션을 생성할 수 있습니다.

new-app으로 생성되는 오브젝트 세트는 입력을 통해 전달되는 아티팩트(소스 리포지토리, 이미지 또는 템플릿)에 따라 다릅니다.

3.3.1. 소스 코드에서 애플리케이션 생성

new-app 명령을 사용하면 로컬 또는 원격 Git 리포지토리의 소스 코드에서 애플리케이션을 생성할 수 있습니다.

new-app 명령은 소스 코드에서 자체적으로 새 애플리케이션 이미지를 생성하는 빌드 구성을 생성합니다. 또한 **new-app** 명령은 새 이미지를 배포하는 **Deployment** 오브젝트와 이미지를 실행하는 배포에 부하 분산된 액세스를 제공하는 서비스를 생성합니다.

OpenShift Container Platform에서는 파이프라인 또는 소스 빌드 전략 사용 여부를 자동으로 탐지하고, 소스 빌드의 경우 적절한 언어 빌더 이미지를 탐지합니다.

3.3.1.1. 로컬

로컬 디렉터리의 Git 리포지토리에서 애플리케이션을 생성하려면 다음을 실행합니다.

```
$ oc new-app /<path to source code>
```

참고

로컬 Git 리포지토리를 사용하는 경우 리포지토리에 OpenShift Container Platform 클러스터에서 액세스할 수 있는 URL을 가리키는 **origin**이라는 원격이 있어야 합니다. 확인되는 원격이 없는 경우 **new-app** 명령을 실행하면 바이너리 빌드가 생성됩니다.

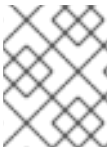
3.3.1.2. 원격

원격 Git 리포지토리에서 애플리케이션을 생성하려면 다음을 실행합니다.

```
$ oc new-app https://github.com/sclorg/cakephp-ex
```

프라이빗 원격 Git 리포지토리에서 애플리케이션을 생성하려면 다음을 실행합니다.

```
$ oc new-app https://github.com/youruser/yourprivaterepo --source-secret=yoursecret
```



참고

프라이빗 원격 Git 리포지토리를 사용하는 경우 **--source-secret** 플래그를 사용하면 빌드 구성에 삽입할 기존 소스 복제 시크릿을 지정하여 리포지토리에 액세스할 수 있습니다.

--context-dir 플래그를 지정하여 소스 코드 리포지토리의 하위 디렉터리를 사용할 수 있습니다. 원격 Git 리포지토리 및 컨텍스트 하위 디렉터리에서 애플리케이션을 생성하려면 다음을 실행합니다.

```
$ oc new-app https://github.com/sclorg/s2i-ruby-container.git \
    --context-dir=2.0/test/puma-test-app
```

또한 원격 URL을 지정하면 URL 끝에 **#<branch_name>**을 추가하여 사용할 Git 분기를 지정할 수 있습니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

3.3.1.3. 빌드 전략 탐지

새 애플리케이션을 생성할 때 Jenkins 파일이 소스 리포지토리의 루트 또는 지정된 컨텍스트 디렉터리에 있는 경우 OpenShift Container Platform에서는 파이프라인 빌드 전략을 생성합니다. 그렇지 않으면 소스 빌드 전략을 생성합니다.

--strategy 플래그를 **pipeline** 또는 **source**에 설정하여 빌드 전략을 재정의합니다.

```
$ oc new-app /home/user/code/myapp --strategy=docker
```

참고

oc 명령을 실행하려면 빌드 소스가 포함된 파일이 원격 Git 리포지토리에 제공되어야 합니다. 모든 소스 빌드에 **git remote -v**를 사용해야 합니다.

3.3.1.4. 언어 탐지

소스 빌드 전략을 사용하는 경우 **new-app**은 리포지토리의 루트 또는 지정된 컨텍스트 디렉터리에 특정 파일이 있는지에 따라 사용할 언어 빌더를 결정합니다.

표 3.1. **new-app**에서 탐지하는 언어

언어	파일
dotnet	**project.json**, ***.csproj**
jee	**pom.xml**
nodejs	**app.json**, **package.json**

언어	파일
perl	**cpanfile**, **index.pl**
php	**composer.json**, **index.php**
python	**requirements.txt**, **setup.py**
ruby	**Gemfile**, **Rakefile**, **config.ru**
scala	**build.sbt**
golang	**Godeps**, **main.go**

언어를 탐지한 후 **new-app**은 OpenShift Container Platform 서버에서 탐지한 언어와 일치하는 **supports** 주석이 있는 이미지 스트림 태그 또는 탐지된 언어의 이름과 일치하는 이미지 스트림을 검색합니다. 일치 항목이 없는 경우 **new-app**은 Docker Hub 레지스트리에서 이름을 기반으로 탐지된 언어와 일치하는 이미지를 검색합니다.

이미지, 이미지 스트림 또는 컨테이너 사양 중 하나, 리포지토리를 구분자 **~**를 사용하여 지정하면 빌더에서 특정 소스 리포지토리에 사용하는 이미지를 재정의할 수 있습니다. 이 작업이 완료되면 빌드 전략 탐지 및 언어 탐지가 수행되지 않습니다.

예를 들어 원격 리포지토리의 소스에 **myproject/my-ruby** 이미지 스트림을 사용하려면 다음을 실행합니다.

```
$ oc new-app myproject/my-ruby~https://github.com/openshift/ruby-hello-world.git
```

로컬 리포지토리의 소스에 **openshift/ruby-20-centos7:latest** 컨테이너 이미지 스트림을 사용하려면 다음을 실행합니다.

```
$ oc new-app openshift/ruby-20-centos7:latest~/home/user/code/my-ruby-app
```

참고

언어 탐지 기능에는 리포지토리를 복제하고 검사할 수 있도록 Git 클라이언트를 로컬에 설치해야 합니다. Git을 사용할 수 없는 경우 **<image>~<repository>** 구문으로 리포지토리에 사용할 빌더 이미지를 지정하여 언어 탐지 단계가 수행되지 않도록 수 있습니다.

-i <image> <repository> 호출을 실행하려면 **new-app**에서 아티팩트 유형이 무엇인지 확인하기 위해 **repository**를 복제해야 하므로 Git을 사용할 수 없는 경우 이 명령이 실패합니다.

-i <image> --code <repository> 호출을 실행하려면 **image**를 소스 코드의 빌더로 사용해야 하는지 아니면 데이터베이스 이미지의 경우와 같이 별도로 배포해야 하는지 확인하기 위해 **new-app**에서 **repository**를 복제해야 합니다.

3.3.2. 이미지에서 애플리케이션 생성

기존 이미지에서 애플리케이션을 배포할 수 있습니다. 이미지는 OpenShift Container Platform 서버의 이미지 스트림, 특정 레지스트리의 이미지 또는 로컬 Docker 서버의 이미지에서 가져올 수 있습니다.

new-app 명령은 전달된 인수에 지정된 이미지 유형을 확인합니다. 그러나 컨테이너 이미지에는 **--docker-image** 인수를, 이미지 스트림에는 **-i|--image-stream** 인수를 사용하여 **new-app**에 이미지 유형을 명시적으로 표시할 수 있습니다.

참고

로컬 Docker 리포지토리에서 이미지를 지정하는 경우 OpenShift Container Platform 클러스터 노드에서 동일한 이미지를 사용할 수 있는지 확인해야 합니다.

3.3.2.1. Docker Hub MySQL 이미지

Docker Hub MySQL 이미지에서 애플리케이션을 생성합니다. 예를 들면 다음과 같습니다.

```
$ oc new-app mysql
```

3.3.2.2. 프라이빗 레지스트리의 이미지

프라이빗 레지스트리의 이미지를 사용하여 애플리케이션을 생성하고 전체 컨테이너 이미지 사양을 지정합니다.

```
$ oc new-app myregistry:5000/example/myimage
```

3.3.2.3. 기존 이미지 스트림 및 선택적 이미지 스트림 태그

기존 이미지 스트림 및 선택적 이미지 스트림 태그에서 애플리케이션을 생성합니다.

```
$ oc new-app my-stream:v1
```

3.3.3. 템플릿에서 애플리케이션 생성

템플릿 이름을 인수로 지정하여 이전에 저장된 템플릿 또는 템플릿 파일에서 애플리케이션을 생성할 수 있습니다. 예를 들어 샘플 애플리케이션 템플릿을 저장하고 이 템플릿을 사용하여 애플리케이션을 생성할 수 있습니다.

현재 프로젝트의 템플릿 라이브러리에 애플리케이션 템플릿을 업로드합니다. 다음 예제에서는 **examples/sample-app/application-template-stibuild.json**이라는 파일에서 애플리케이션 템플릿을 업로드합니다.

```
$ oc create -f examples/sample-app/application-template-stibuild.json
```

그런 다음 애플리케이션 템플릿을 참조하여 새 애플리케이션을 생성합니다. 이 예에서 템플릿 이름은 **ruby-helloworld-sample**입니다.

```
$ oc new-app ruby-helloworld-sample
```

OpenShift Container Platform에 먼저 저장하지 않고 로컬 파일 시스템에서 템플릿 파일을 참조하여 새 애플리케이션을 생성하려면 **-f|--file** 인수를 사용하십시오. 예를 들면 다음과 같습니다.

```
$ oc new-app -f examples/sample-app/application-template-stibuild.json
```

3.3.3.1. 템플릿 매개변수

템플릿을 기반으로 애플리케이션을 생성할 때 **-p|--param** 인수를 사용하여 템플릿에 정의된 매개변수 값을 설정합니다.

```
$ oc new-app ruby-helloworld-sample \
    -p ADMIN_USERNAME=admin -p ADMIN_PASSWORD=mypassword
```

템플릿을 인스턴스화할 때 해당 매개변수를 파일에 저장한 다음 해당 파일을 **--param-file**과 함께 사용할 수 있습니다. 표준 입력에서 매개변수를 읽으려면 **--param-file=-**을 사용합니다. 다음은 **helloworld.params**라는 예제 파일입니다.

```
ADMIN_USERNAME=admin
ADMIN_PASSWORD=mypassword
```

템플릿을 인스턴스화할 때 파일에서 매개변수를 참조합니다.

```
$ oc new-app ruby-helloworld-sample --param-file=helloworld.params
```

3.3.4. 애플리케이션 생성 수정

new-app 명령은 생성된 애플리케이션을 빌드, 배포, 실행하는 OpenShift Container Platform 오브젝트를 생성합니다. 일반적으로 이러한 오브젝트는 현재 프로젝트에서 생성되고 입력 소스 리포지토리 또는 입력 이미지에서 파생된 이름이 할당됩니다. 그러나 **new-app**을 사용하면 이 동작을 수정할 수 있습니다.

표 3.2. **new-app** 출력 오브젝트

개체	설명
BuildConfig	명령줄에 지정된 각 소스 리포지토리에 대해 **BuildConfig** 오브젝트가 생성됩니다. **BuildConfig** 오브젝트는 사용할 전략, 소스 위치, 빌드 출력 위치를 지정합니다.
ImageStreams	**BuildConfig** 오브젝트의 경우 일반적으로 두 개의 이미지 스트림이 생성됩니다. 하나는 입력 이미지를 나타냅니다. 소스 빌드에서 이 이미지는 빌더 이미지입니다. **Docker** 빌드에서는 **출처** 이미지에 해당합니다. 두 번째 이미지는 출력 이미지를 나타냅니다. 컨테이너 이미지가 **new-app**에 입력으로 지정되면 해당 이미지에도 이미지 스트림이 생성됩니다.
DeploymentConfig	**DeploymentConfig** 오브젝트는 빌드 출력 또는 지정된 이미지를 배포하기 위해 생성됩니다. **new-app** 명령은 결과 **DeploymentConfig** 오브젝트에 포함되어 있는 컨테이너에 지정된 모든 Docker 볼륨에 대해 **emptyDir** 볼륨을 생성합니다.
Service	**new-app** 명령은 입력 이미지에서 노출된 포트를 탐지합니다. 가장 낮은 숫자의 노출된 포트를 사용하여 해당 포트를 노출하는 서비스를 생성합니다. 다른 포트를 공개하려면 **new-app**을 완료한 후 **oc expose** 명령을 사용하여 추가 서비스를 생성하기만 하면 됩니다.
기타	다른 오브젝트는 템플릿을 인스턴스화할 때 템플릿에 따라 생성할 수 있습니다.

3.3.4.1. 환경 변수 지정

템플릿, 소스 또는 이미지에서 애플리케이션을 생성할 때 **-e|--env** 인수를 사용하여 런타임 시 환경 변수를 애플리케이션 컨테이너에 전달할 수 있습니다.

```
$ oc new-app openshift/postgresql-92-centos7 \
```

```
    -e POSTGRESQL_USER=user \
    -e POSTGRESQL_DATABASE=db \
    -e POSTGRESQL_PASSWORD=password
```

이 변수는 **--env-file** 인수를 사용하여 파일에서 읽을 수도 있습니다. 다음은 **postgresql.env**라는 예제 파일입니다.

```
POSTGRESQL_USER=user
POSTGRESQL_DATABASE=db
POSTGRESQL_PASSWORD=password
```

파일에서 변수를 읽습니다.

```
$ oc new-app openshift/postgresql-92-centos7 --env-file=postgresql.env
```

또한 **--env-file=-**을 사용하여 환경 변수를 표준 입력에 제공할 수 있습니다.

```
$ cat postgresql.env | oc new-app openshift/postgresql-92-centos7 --env-file=-
```

참고

new-app 처리의 일부로 생성된 모든 **BuildConfig** 오브젝트는 **-e|--env** 또는 **--env-file** 인수로 전달되는 환경 변수를 통해 업데이트되지 않습니다.

3.3.4.2. 빌드 환경 변수 지정

템플릿, 소스 또는 이미지에서 애플리케이션을 생성할 때 **--build-env** 인수를 사용하여 런타임 시 환경 변수를 빌드 컨테이너에 전달할 수 있습니다.

```
$ oc new-app openshift/ruby-23-centos7 \
    --build-env HTTP_PROXY=http://myproxy.net:1337/ \
    --build-env GEM_HOME=~/.gem
```

이 변수는 **--build-env-file** 인수를 사용하여 파일에서 읽을 수도 있습니다. 다음은 **ruby.env**라는 예제 파일입니다.

```
HTTP_PROXY=http://myproxy.net:1337/
GEM_HOME=~/.gem
```

파일에서 변수를 읽습니다.

```
$ oc new-app openshift/ruby-23-centos7 --build-env-file=ruby.env
```

또한 **--build-env-file=-**을 사용하여 환경 변수를 표준 입력에 제공할 수 있습니다.

```
$ cat ruby.env | oc new-app openshift/ruby-23-centos7 --build-env-file=-
```

3.3.4.3. 라벨 지정

소스, 이미지 또는 템플릿에서 애플리케이션을 생성할 때는 **-l|--label** 인수를 사용하여 생성된 오브젝트에 라벨을 추가할 수 있습니다. 라벨을 사용하면 애플리케이션과 관련된 오브젝트를 전체적으로 선택, 구성, 삭제할 수 있습니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world -l name=hello-world
```

3.3.4.4. 생성하지 않고 출력 보기

new-app 명령 실행의 시험 실행을 보려면 **-o|--output** 인수를 **yaml** 또는 **json** 값과 함께 사용하면 됩니다. 그런 다음 출력을 사용하여 생성된 오브젝트를 미리 보거나 편집할 수 있는 파일로 리디렉션할 수 있습니다. 만족하는 경우 **oc create**를 사용하여 OpenShift Container Platform 오브젝트를 생성할 수 있습니다.

new-app 아티팩트를 파일에 출력하려면 다음을 실행합니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world \
   -o yaml > myapp.yaml
```

파일을 편집합니다.

```
$ vi myapp.yaml
```

파일을 참조하여 새 애플리케이션을 생성합니다.

```
$ oc create -f myapp.yaml
```

3.3.4.5. 다양한 이름으로 오브젝트 생성

new-app으로 생성한 오브젝트는 일반적으로 소스 리포지토리 또는 해당 오브젝트를 생성하는 데 사용된 이미지의 이름을 따서 이름이 지정됩니다. 명령에 **--name** 플래그를 추가하여 생성한 오브젝트의 이름을 설정할 수 있습니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world --name=myapp
```

3.3.4.6. 다른 프로젝트에서 오브젝트 생성

일반적으로 **new-app**은 현재 프로젝트에서 오브젝트를 생성합니다. 그러나 **-n|--namespace** 인수를 사용하면 다른 프로젝트에서 오브젝트를 생성할 수 있습니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world -n myproject
```

3.3.4.7. 여러 오브젝트 생성

new-app 명령을 사용하면 **new-app**에 다양한 매개변수를 지정하는 애플리케이션을 여러 개 생성할 수 있습니다. 명령 줄에서 지정된 라벨은 단일 명령으로 생성된 모든 개체에 적용됩니다. 환경 변수는 소스 또는 이미지에서 생성한 모든 구성 요소에 적용됩니다.

소스 리포지토리 및 Docker Hub 이미지에서 애플리케이션을 생성하려면 다음을 실행합니다.

```
$ oc new-app https://github.com/openshift/ruby-hello-world mysql
```

참고

소스 코드 리포지토리와 빌더 이미지가 별도의 인수로 지정되면 **new-app**에서 빌더 이미지를 소스 코드 저장소의 빌더로 사용합니다. 이를 원하지 않는 경우 ~ 구분자를 사용하여 소스에 필요한 빌더 이미지를 지정합니다.

3.3.4.8. 단일 Pod에서 이미지 및 소스 그룹화

new-app 명령을 사용하면 단일 Pod에 여러 이미지를 함께 배포할 수 있습니다. 함께 그룹화할 이미지를 지정하려면 **+** 구분자를 사용합니다. **--group** 명령줄 인수를 사용하여 함께 그룹화해야 하는 이미지를 지정할 수도 있습니다. 소스 리포지토리에서 빌드한 이미지를 기타 이미지와 함께 그룹화하려면 그룹에 해당 빌더 이미지를 지정합니다.

```
$ oc new-app ruby+mysql
```

소스에서 빌드한 이미지를 외부 이미지와 함께 배포하려면 다음을 실행합니다.

```
$ oc new-app \
    ruby~https://github.com/openshift/ruby-hello-world \
    mysql \
    --group=ruby+mysql
```

3.3.4.9. 이미지, 템플릿 및 기타 입력 검색

oc new-app 명령에 대한 이미지, 템플릿 및 기타 입력을 검색하려면 **--search** 및 **--list**플래그를 추가합니다. 예를 들어 PHP를 포함하는 모든 이미지 또는 템플릿을 찾으려면 다음을 실행합니다.

```
$ oc new-app --search php
```

3.4. 토폴로지 보기를 사용하여 애플리케이션 구성 보기

웹 콘솔의 개발자 화면에 있는 토폴로지 보기에는 프로젝트 내의 모든 애플리케이션과 해당 빌드 상태, 애플리케이션에 연결된 구성 요소 및 서비스가 그래픽으로 표시되어 있습니다.

3.4.1. 사전 요구 사항

토폴로지 보기에서 애플리케이션을 확인하고 애플리케이션과 상호 작용하려면 다음을 수행합니다.

- 웹 콘솔에 로그인 했습니다.
- 프로젝트에 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 적절한 역할과 권한이 있습니다.
- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션을 생성하고 배포했습니다.
- 개발자 화면에 있습니다.

3.4.2. 애플리케이션의 토폴로지 보기

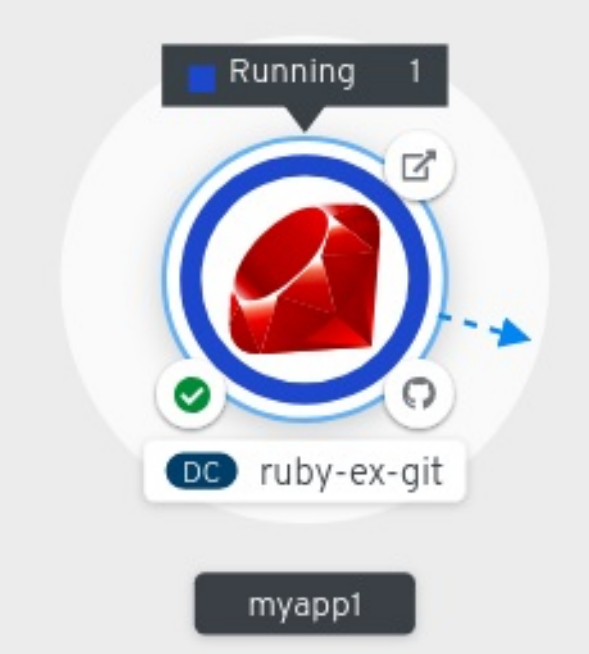
개발자 화면의 왼쪽 탐색 패널을 사용하여 토폴로지 보기로 이동할 수 있습니다. 애플리케이션을 생성한 후에는 토폴로지 보기로 자동으로 안내됩니다. 여기서 애플리케이션 Pod의 상태를 확인하고, 공용 URL에서 애플리케이션에 빠르게 액세스하고, 소스 코드에 액세스하여 수정하고, 마지막 빌드의 상태를 확인할

수 있습니다. 확대 및 축소하여 특정 애플리케이션에 대한 세부 정보를 확인할 수 있습니다.

Pod의 상태 또는 단계는 실행 중( ), 준비 안 됨 ( ), 경고( ), 실패( ), 보류 중( ), 성공( ), 종료( ) 또는 알 수 없음( )으로 다양한 색상과 툴팁으로 표시됩니다. Pod 상태에 대한 자세한 내용은 Kubernetes 설명서 를 참조하십시오.

애플리케이션을 생성하고 이미지를 배포하면 해당 상태가 보류 중으로 표시됩니다. 애플리케이션을 빌드한 후에는 실행 중으로 표시됩니다.

그림 3.1. 애플리케이션 토폴로지



애플리케이션 리소스 이름에는 다음과 같이 다양한 유형의 리소스 오브젝트에 대한 표시가 추가됩니다.

- CJ: **CronJob**
- D: **Deployment**
- DC: **DeploymentConfig**
- DS: **DaemonSet**
- J: **Job**
- P: **Pod**
- SS: **StatefulSet**
- (Knative): 서버리스 애플리케이션

참고

서버리스 애플리케이션은 토폴로지 보기에 로드하고 표시되는 데 다소 시간이 걸립니다. 서버리스 애플리케이션을 생성할 때 먼저 서비스 리소스를 생성한 다음 리비전을 생성합니다. 그런 다음 토폴로지 보기에 배포되고 표시됩니다. 이 작업이 유일한 워크로드인 경우 추가 페이지로 리디렉션될 수 있습니다. 리비전이 배포되면 토폴로지 보기에 서버리스 애플리케이션이 표시됩니다.

3.4.3. 애플리케이션 및 구성 요소와 상호 작용

웹 콘솔의 개발자 화면에 있는 토폴로지 보기에는 애플리케이션 및 구성 요소와 상호 작용할 수 있는 다음 옵션이 있습니다.

- 공개 URL의 경로를 통해 노출되는 애플리케이션을 확인하려면 **URL** 열기( )를 클릭합니다.
- 소스 코드에 액세스하여 수정하려면 소스 코드 편집을 클릭합니다.

참고

이 기능은 **Git**에서, 카탈로그에서, **Dockerfile**에서 옵션을 사용하여 애플리케이션을 생성할 때만 사용할 수 있습니다.

- 최신 빌드의 이름과 해당 상태를 확인하려면 Pod의 왼쪽 아래 아이콘 위에 커서를 올려 놓습니다. 애플리케이션 빌드 상태는 신규( ), 보류 중( ), 실행 중( ), 완료됨( ), 실패( ), 취소됨( )으로 표시됩니다.
- 화면 오른쪽 상단에 나열된 바로 가기 메뉴를 사용하여 토폴로지 보기에서 구성 요소를 탐색합니다.
- **List View**(나열 보기) 아이콘을 사용하여 모든 애플리케이션 목록을 확인하고 **Topology View**(토폴로지 보기) 아이콘을 사용하여 토폴로지 보기로 다시 전환합니다.
- **Find by name** (이름으로 찾기) 필드를 사용하여 쿼리와 일치하는 구성 요소 이름이 있는 구성 요소를 선택합니다. 검색 결과가 표시 영역 외부에 표시될 수 있습니다. 왼쪽 아래에 있는 툴바에서 화면에 맞추기를 클릭하여 모든 구성 요소를 표시하도록 토폴로지 보기의 크기를 조정합니다.
- 다양한 애플리케이션 그룹화에 대한 토폴로지 보기를 구성하려면 표시 옵션 드롭다운 목록을 사용합니다. 해당 옵션은 프로젝트에 배포된 구성 요소 유형에 따라 제공됩니다.
  - **Pod** 개수: 구성 요소 아이콘에 구성 요소의 포드 수를 표시하려면 선택합니다.
  - 이벤트 소스: 이벤트 소스를 표시하거나 숨기려면 전환합니다.
  - 가상 머신: 토글을 선택하여 가상 시스템을 표시하거나 숨길 수 있습니다.
  - 레이블: 구성 요소 레이블을 표시하거나 숨기려면 전환합니다.
  - 애플리케이션 그룹화 : 애플리케이션 그룹을 애플리케이션 그룹 개요와 연결된 경고가 포함된 카드로 축소하려면 지웁니다.
  - **Helm** 릴리스: Helm 릴리스로 배포된 구성 요소를 지정된 릴리스의 개요가 포함된 카드로 축소하려면 지웁니다.
  - **Knative** 서비스: Knative 서비스 구성 요소를 지정된 구성 요소의 개요가 포함된 카드로 축소하려면 지웁니다.
  - **Operator** 그룹화는 Operator와 함께 배포된 구성 요소를 지정된 그룹의 개요가 포함된 카드로 축소하기 위해 지웁니다.
- Pod의 상태 또는 단계는 다음과 같이 다양한 색상 및 툴팁으로 표시됩니다.
  - 실행 중 ( ): 포드는 노드에 바인딩되고 모든 컨테이너가 생성됩니다. 하나 이상의 컨테이너가 계속 실행 중이거나 시작 또는 다시 시작하는 중입니다.

- 준비되지 않음 (): 컨테이너가 여러 개 실행되고 있지만 일부 컨테이너가 준비되지 않은 포드.
- 경고(): 포드의 컨테이너가 종료되고 있지만 종료에 성공하지 못했습니다. 일부 컨테이너는 다른 상태일 수 있습니다.
- 실패(): 포드의 모든 컨테이너가 종료되었지만 하나 이상의 컨테이너가 실패로 종료되었습니다. 즉 컨테이너는 0이 아닌 상태로 종료되었거나 시스템에 의해 종료되었습니다.
- 보류 중(): Pod는 Kubernetes 클러스터에서 허용되지만 하나 이상의 컨테이너가 설정되어 실행할 준비가 되어 있지 않습니다. 여기에는 Pod가 네트워크를 통해 컨테이너 이미지를 다운로드하는 데 소요되는 시간뿐만 아니라 Pod를 예약 대기하는 시간이 포함됩니다.
- 성공(): 포드의 모든 컨테이너가 성공적으로 종료되고 다시 시작되지 않습니다.
- 종료 중(): Pod가 삭제되면 일부 kubectl 명령에서 종료하는 것으로 표시됩니다. 종료 중 상태는 Pod 단계 중 하나가 아닙니다. Pod에는 정상 종료 기간이 부여되며 기본값은 30초입니다.
- 알 수 없음(): 포드의 상태를 가져올 수 없습니다. 일반적으로 이 단계는 Pod가 실행되어야 하는 노드와 통신하는 동안 오류로 인해 발생합니다.

3.4.4. 애플리케이션 Pod 스케일링 및 빌드와 경로 확인

토폴로지 보기에는 개요 패널에 배포된 구성 요소의 세부 정보가 있습니다. 다음과 같이 개요 및 리소스 탭을 사용하여 애플리케이션 Pod를 스케일링하고 빌드 상태, 서비스, 경로를 확인할 수 있습니다.

- 구성 요소 노드를 클릭하면 오른쪽에 개요 패널이 표시됩니다. 개요 탭에서는 다음을 수행할 수 있습니다.
  - 위쪽 및 아래쪽 화살표를 사용하여 Pod 수를 스케일링하여 애플리케이션 인스턴스 수를 수동으로 늘리거나 줄입니다. 서버리스 애플리케이션의 경우 유휴 상태에서는 Pod가 자동으로 0으로 축소되고 채널 트래픽에 따라 확장됩니다.
  - 애플리케이션의 라벨, 주석, 상태를 확인합니다.
- 다음을 수행하려면 리소스 탭을 클릭합니다.
  - 모든 Pod 목록을 확인하고 해당 상태 및 액세스 로그를 본 후 Pod 세부 정보를 확인할 Pod를 클릭합니다.
  - 빌드, 해당 상태, 액세스 로그를 확인하고 필요한 경우 새 빌드를 시작합니다.
  - 구성 요소에서 사용하는 서비스 및 경로를 참조하십시오.

  서버리스 애플리케이션의 경우 리소스 탭에는 해당 구성 요소에 사용된 리버전, 경로, 구성 정보가 있습니다.

3.4.5. 애플리케이션 내의 여러 구성 요소 그룹화

Add(추가) 페이지를 사용하여 프로젝트에 여러 구성 요소 또는 서비스를 추가하고 토폴로지 페이지를 사용하여 애플리케이션 그룹 내에서 애플리케이션과 리소스를 그룹화할 수 있습니다. 다음 절차에서는 Node.js 구성 요소가 있는 기존 애플리케이션에 MongoDB 데이터베이스 서비스를 추가합니다.

사전 요구 사항

- 개발자 화면을 사용하여 OpenShift Container Platform에서 Node.js 애플리케이션을 생성하고 배포했는지 확인합니다.

프로세스

1. 다음과 같이 MongoDB 서비스를 생성하여 프로젝트에 배포합니다.

   a. 개발자 화면에서 **Add** (추가) 보기로 이동하여 **Database** (데이터베이스) 옵션을 선택하여 애플리케이션에 구성 요소 또는 서비스로 추가할 수 있는 여러 옵션이 있는 개발자 카탈로그를 확인합니다.

   b. **MongoDB** 옵션을 클릭하여 서비스 세부 정보를 확인합니다.

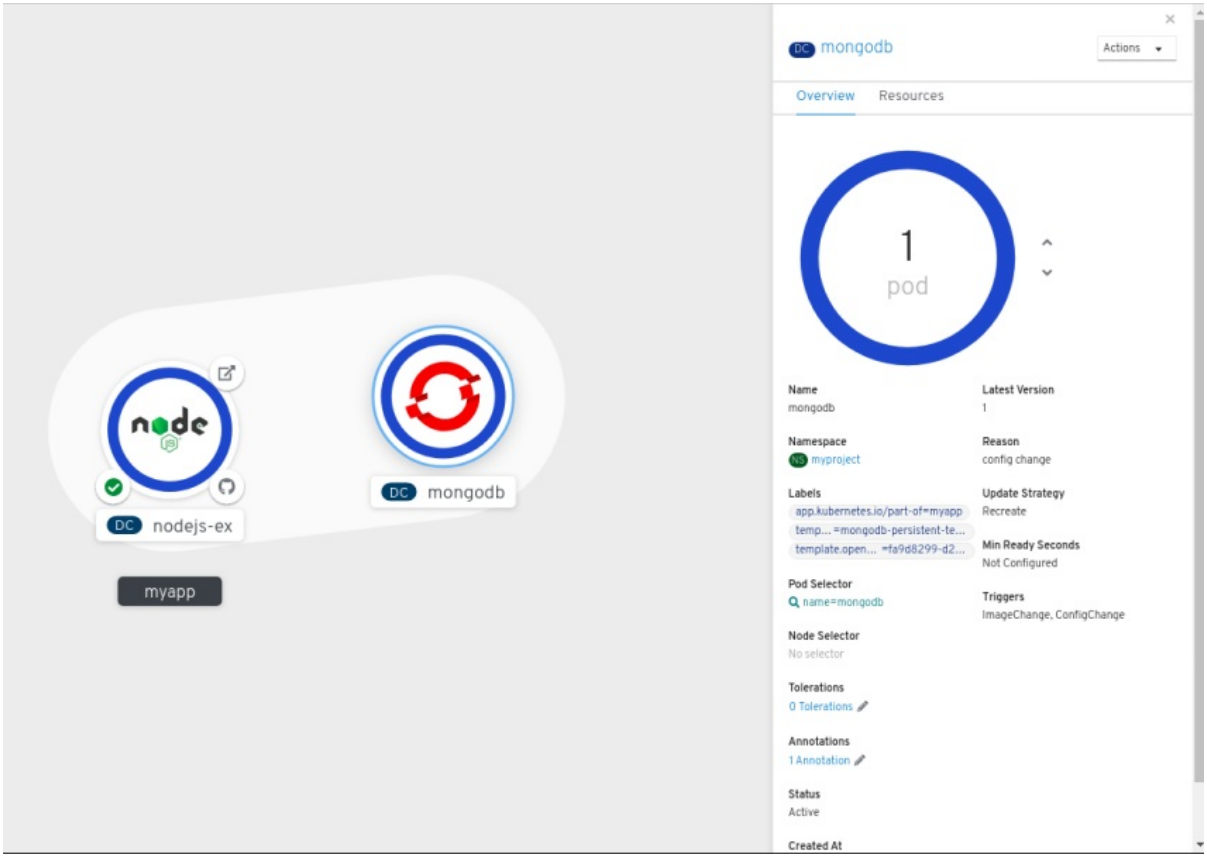
   c. 템플릿 인스턴스화 를 클릭하여 MongoDB 서비스 세부 정보가 자동으로 채워진 템플릿을 확인하고 만들기를 클릭하여 서비스를 만듭니다.

2. 왼쪽 탐색 패널에서 **Topology(**토폴로지)를 클릭하여 프로젝트에 배포된 MongoDB 서비스를 확인합니다.

3. MongoDB 서비스를 기존 애플리케이션 그룹에 추가하려면 **mongodb** 포드를 선택하여 애플리케이션으로 끌어 옵니다. MongoDB 서비스는 기존 애플리케이션 그룹에 추가됩니다.

4. 구성 요소를 끌어서 애플리케이션 그룹에 추가하면 구성 요소에 필요한 레이블이 자동으로 추가됩니다. MongoDB 서비스 노드를 클릭하여 개요 패널의 레이블 섹션에 추가된 **app.kubernetes.io/part-of=myapp** 레이블을 확인합니다.

   그림 **3.2.** 애플리케이션 그룹화



또는 다음과 같이 애플리케이션에 구성 요소를 추가할 수도 있습니다.

1. 애플리케이션에 MongoDB 서비스를 추가하려면 **mongodb** 포드를 클릭하여 오른쪽에 있는 개요 패널을 확인합니다.

2. 패널 오른쪽 상단에 있는 작업 드롭다운 메뉴를 클릭하고 **Edit Application Grouping** (애플리케이션 그룹화 편집)을 선택합니다.

3. **Edit Application Grouping** (애플리케이션 그룹화 편집) 대화 상자에서 애플리케이션 선택 드롭다운 목록을 클릭하고 적절한 애플리케이션 그룹을 선택합니다.

4. **Save(**저장)를 클릭하여 애플리케이션 그룹에 추가된 MongoDB 서비스를 확인합니다.

구성 요소를 선택하고 **Shift** 키를 누른 상태로 애플리케이션 그룹 밖으로 드래그하면 애플리케이션 그룹에서 구성 요소를 제거할 수 있습니다.

3.4.6. 애플리케이션 내 및 애플리케이션 간 구성 요소 연결

애플리케이션 내에서 여러 구성 요소를 그룹화하는 것 외에도 토폴로지 보기를 사용하여 구성 요소를 서로 연결할 수 있습니다. 바인딩 커넥터 또는 시각적 커넥터를 사용하여 구성 요소를 연결할 수 있습니다.

대상 노드가 Operator 지원 서비스인 경우에만 구성 요소 간 바인딩 연결을 설정할 수 있습니다. 이러한 연결은 해당 대상 노드로 화살표를 드래그할 때 표시되는 바인딩 커넥터 생성 툴팁으로 표시됩니다. 바인딩 커넥터를 사용하여 애플리케이션을 서비스에 연결하면 서비스 바인딩 요청이 생성됩니다. 그러면 Service Binding Operator 컨트롤러에서 중간 시크릿을 사용하여 필요한 바인딩 데이터를 애플리케이션 배포에 환경 변수로 삽입합니다. 요청이 성공하면 애플리케이션이 재배포되어 연결된 구성 요소 간 상호 작용을 설정합니다.

시각적 커넥터는 구성 요소 간 시각적 연결만 설정하고 연결하려는 의도를 표시합니다. 구성 요소 간 상호 작용은 설정되지 않습니다. 대상 노드가 Operator에서 지원하는 서비스가 아닌 경우 대상 노드로 화살표를 드래그하면 시각적 커넥터 생성 툴팁이 표시됩니다.

3.4.6.1. 구성 요소 간 시각적 연결 생성

시각적 커넥터를 사용하여 애플리케이션 구성 요소를 연결하려는 의도를 나타낼 수 있습니다.

이 절차에서는 MongoDB 서비스와 Node.js 애플리케이션 간 시각적 연결을 생성하는 예제를 보여줍니다.

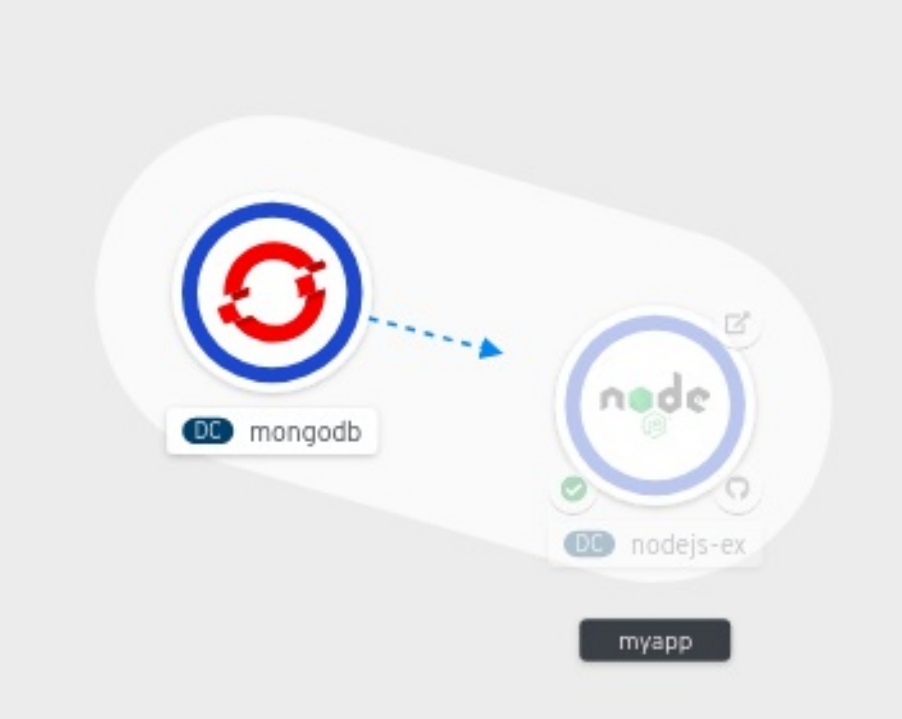
사전 요구 사항

- 개발자 화면을 사용하여 Node.js 애플리케이션을 생성하고 배포했는지 확인합니다.
- 개발자 화면을 사용하여 MongoDB 서비스를 생성하고 배포했는지 확인합니다.

프로세스

1. MongoDB 서비스 위에 커서를 올리면 노드에서 출발하는 화살표가 표시됩니다.

그림 3.3. 커넥터

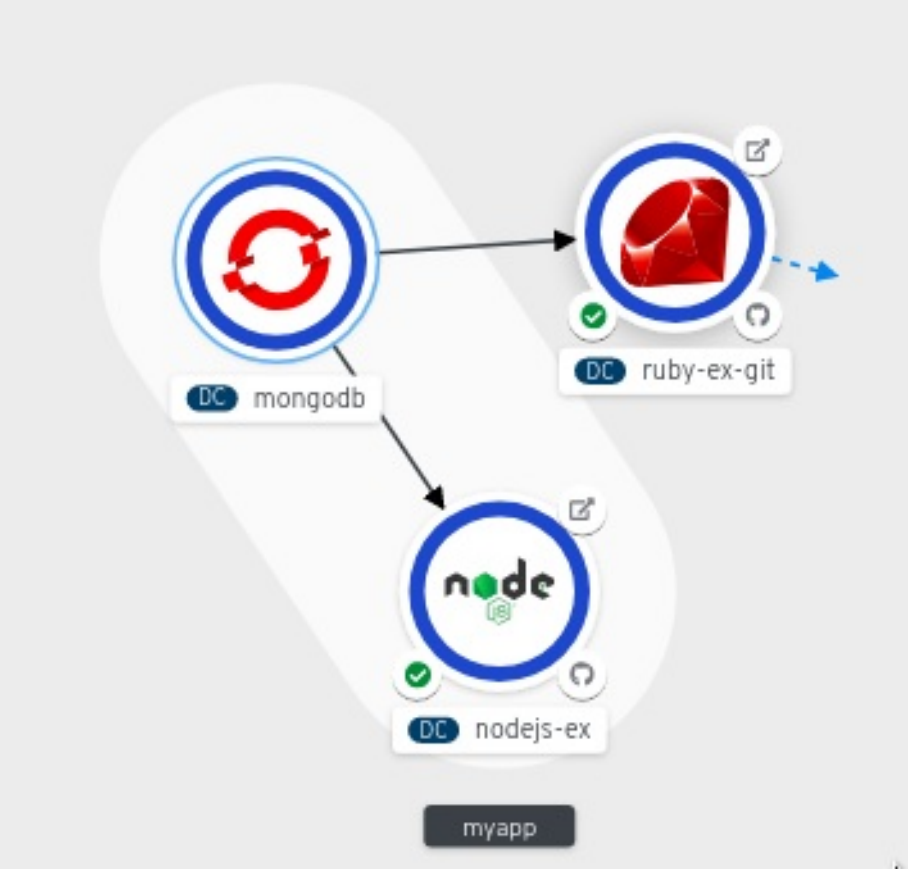


2. 화살표를 클릭하고 Node.js 구성 요소 쪽으로 드래그하여 MongoDB 서비스와 연결합니다.

3. MongoDB 서비스를 클릭하여 개요 패널을 확인합니다. 서비스에 추가된 **Key = app.openshift.io/connects-to** 및 Value = **[{"apiVersion":"apps.openshift.io/v1","kind":"DeploymentConfig","name":"nodejs-ex"}]** 주석을 확인하려면 주석 섹션에서 편집 아이콘을 클릭합니다.

마찬가지로 기타 애플리케이션 및 구성 요소를 생성하고 이들 간의 연결을 설정할 수 있습니다.

그림 3.4. 여러 애플리케이션 연결



3.4.6.2. 구성 요소 간 바인딩 연결 생성

중요

서비스 바인딩은 기술 프리뷰 기능 전용입니다. Technology Preview 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 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

참고

현재 **etcd** 및 **PostgresSQL Database** Operator의 서비스 인스턴스와 같은 몇 가지 특정 Operator를 바인딩할 수 있습니다.

Operator 지원 구성 요소를 사용하여 바인딩 연결을 설정할 수 있습니다.

이 절차에서는 PostgreSQL 데이터베이스 서비스와 Node.js 애플리케이션 간 바인딩 연결을 생성하는 예를 보여줍니다. PostgreSQL Database Operator에서 지원하는 서비스로 바인딩 연결을 생성하려면 먼저 **CatalogSource** 리소스를 사용하여 **OperatorHub**에 Red Hat 제공 PostgreSQL Database Operator를 추가한 후에 Operator를 설치해야 합니다. 그러면 PostreSQL Database Operator에서 시크릿, 구성 맵, 상태, 사양 속성에 바인딩 정보를 노출하는 데이터베이스 리소스를 생성하고 관리합니다.

사전 요구 사항

- 개발자 화면을 사용하여 Node.js 애플리케이션을 생성하고 배포했는지 확인합니다.
- **OperatorHub** 에서 **Service Binding Operator**를 설치했는지 확인합니다.

절차

1. Red Hat에서 제공하는 PostgresSQL Database Operator를 **OperatorHub**에 추가하는 **CatalogSource** 리소스를 생성합니다.
   a. **+**추가 보기에서 **YAML** 옵션을 클릭하여 **YAML** 가져오기 화면을 확인합니다.
   b. 다음 YAML 파일을 추가하여 **CatalogSource** 리소스를 적용합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
   name: sample-db-operators
   namespace: openshift-marketplace
spec:
   sourceType: grpc
   image: quay.io/redhat-developer/sample-db-operators-olm:v1
   displayName: Sample DB OLM registry
   updateStrategy:
      registryPoll:
         interval: 30m
```

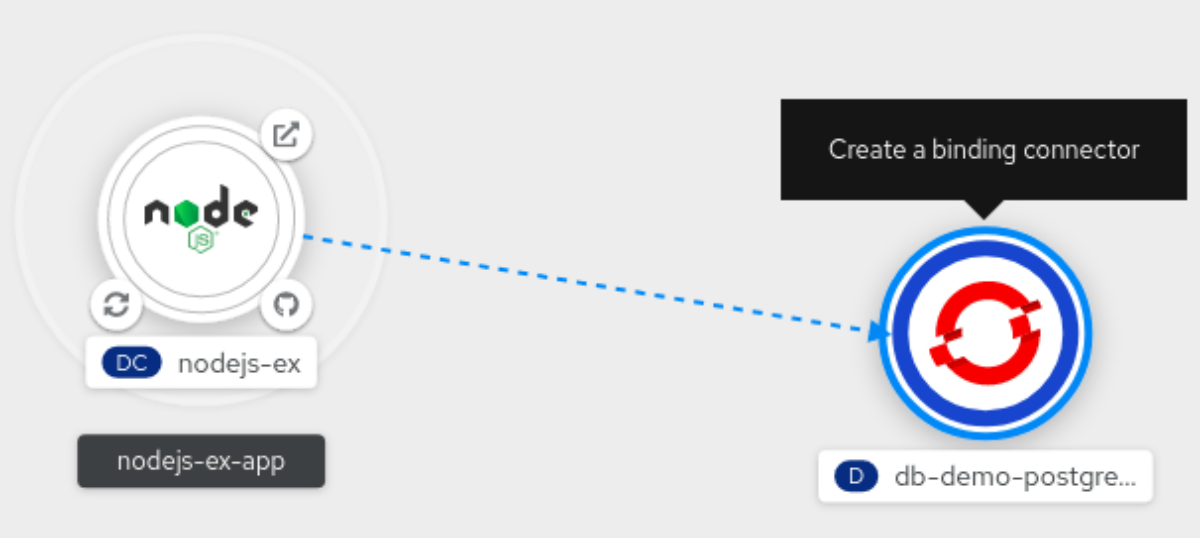
   c. 생성을 클릭하여 클러스터에서 **CatalogSource** 리소스를 생성합니다.
2. Red Hat에서 제공하는 **PostgreSQL Database** Operator를 설치합니다.
   a. 콘솔의 관리자 화면에서 **Operator → OperatorHub**로 이동합니다.
   b. 데이터베이스 카테고리에서 **PostgreSQL Database** Operator를 선택하고 설치합니다.
3. 애플리케이션에 대한 DB(데이터베이스) 인스턴스를 생성합니다.
   a. 개발자 화면으로 전환하고 적절한 프로젝트(예: **test-project**)에 있는지 확인합니다.
   b. **+**추가 보기에서 **YAML** 옵션을 클릭하여 **YAML** 가져오기 화면을 확인합니다.
   c. 편집기에 서비스 인스턴스 YAML을 추가하고 생성을 클릭하여 서비스를 배포합니다. 다음은 서비스 YAML의 예입니다.

```
apiVersion: postgresql.baiju.dev/v1alpha1
kind: Database
metadata:
 name: db-demo
spec:
 image: docker.io/postgres
 imageName: postgres
 dbName: db-demo
```

   DB 인스턴스가 토폴로지 보기에 배포됩니다.
4. 토폴로지 보기에서 Node.js 구성 요소 위에 커서를 올리면 노드에서 출발하는 화살표가 표시됩니다.

5. 이 화살표를 클릭하고 **db-demo-postgresql** 서비스를 향해 끌어와서 Node.js 애플리케이션과 바인딩 연결합니다. 서비스 바인딩 요청이 생성되고 **Service Binding Operator** 컨트롤러에서 DB 연결 정보를 애플리케이션 배포에 환경 변수로 삽입합니다. 요청이 성공하면 애플리케이션이 재배포되고 연결이 설정됩니다.

   그림 3.5. 바인딩 커넥터



3.4.7. 토폴로지 보기에 사용되는 라벨 및 주석

토폴로지 보기에서는 다음 라벨 및 주석을 사용합니다.

노드에 표시되는 아이콘

노드의 아이콘은 **app.openshift.io/runtime** 라벨과 다음으로 **app.kubernetes.io/name** 라벨을 사용하여 일치하는 아이콘을 찾아 정의됩니다. 이러한 일치는 사전 정의된 아이콘 세트를 사용하여 수행됩니다.

소스 코드 편집기 또는 소스에 대한 링크

app.openshift.io/vcs-uri 주석은 소스 코드 편집기에 대한 링크를 생성하는 데 사용됩니다.

노드 커넥터

app.openshift.io/connects-to 주석은 노드를 연결하는 데 사용됩니다.

앱 그룹화

app.kubernetes.io/part-of=<appname> 라벨은 애플리케이션, 서비스, 구성 요소를 그룹화하는 데 사용됩니다.

OpenShift Container Platform 애플리케이션에서 사용해야 하는 라벨 및 주석에 대한 자세한 내용은 OpenShift 애플리케이션의 라벨 및 주석에 대한 지침 을 참조하십시오.

3.5. 애플리케이션 편집

토폴로지 보기를 사용하여 생성하는 애플리케이션의 구성 및 소스 코드를 편집할 수 있습니다.

3.5.1. 사전 요구 사항

- 프로젝트에 OpenShift Container Platform에서 애플리케이션을 생성하고 수정할 적절한 역할과 권한이 있습니다.
- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션을 생성하고 배포했습니다.

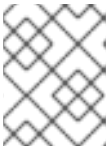
- 웹 콘솔에 로그인하여 개발자 화면으로 전환했습니다.

3.5.2. 개발자 화면을 사용하여 애플리케이션의 소스 코드 편집

개발자 화면에서 토폴로지 보기를 사용하여 애플리케이션의 소스 코드를 편집할 수 있습니다.

프로세스

- 토폴로지 보기에서 배포된 애플리케이션의 오른쪽 아래에 표시되는 소스 코드 편집 아이콘을 클릭하여 소스 코드에 액세스한 후 수정합니다.



참고

이 기능은 **Git**에서, 카탈로그에서, **Dockerfile**에서 옵션을 사용하여 애플리케이션을 생성할 때만 사용할 수 있습니다.

Eclipse Che Operator가 클러스터에 설치된 경우 Che 작업 공간( )이 생성되고 소스 코드를 편집할 수 있는 작업 공간으로 이동합니다. 이 Operator가 설치되지 않은 경우 소스 코드가 호스팅되는 Git 리포지토리( )로 이동합니다.

3.5.3. 개발자 화면을 사용하여 애플리케이션 구성 편집

개발자 화면에서 토폴로지 보기를 사용하여 애플리케이션의 구성을 편집할 수 있습니다.

참고

현재 개발자 화면의 추가 워크플로에 있는 **Git**에서, 컨테이너 이미지, 카탈로그에서 또는 **Dockerfile**에서 옵션을 사용하여 생성한 애플리케이션 구성만 편집할 수 있습니다. CLI 또는 추가 워크플로의 **YAML** 옵션을 사용하여 생성한 애플리케이션 구성은 편집할 수 없습니다.

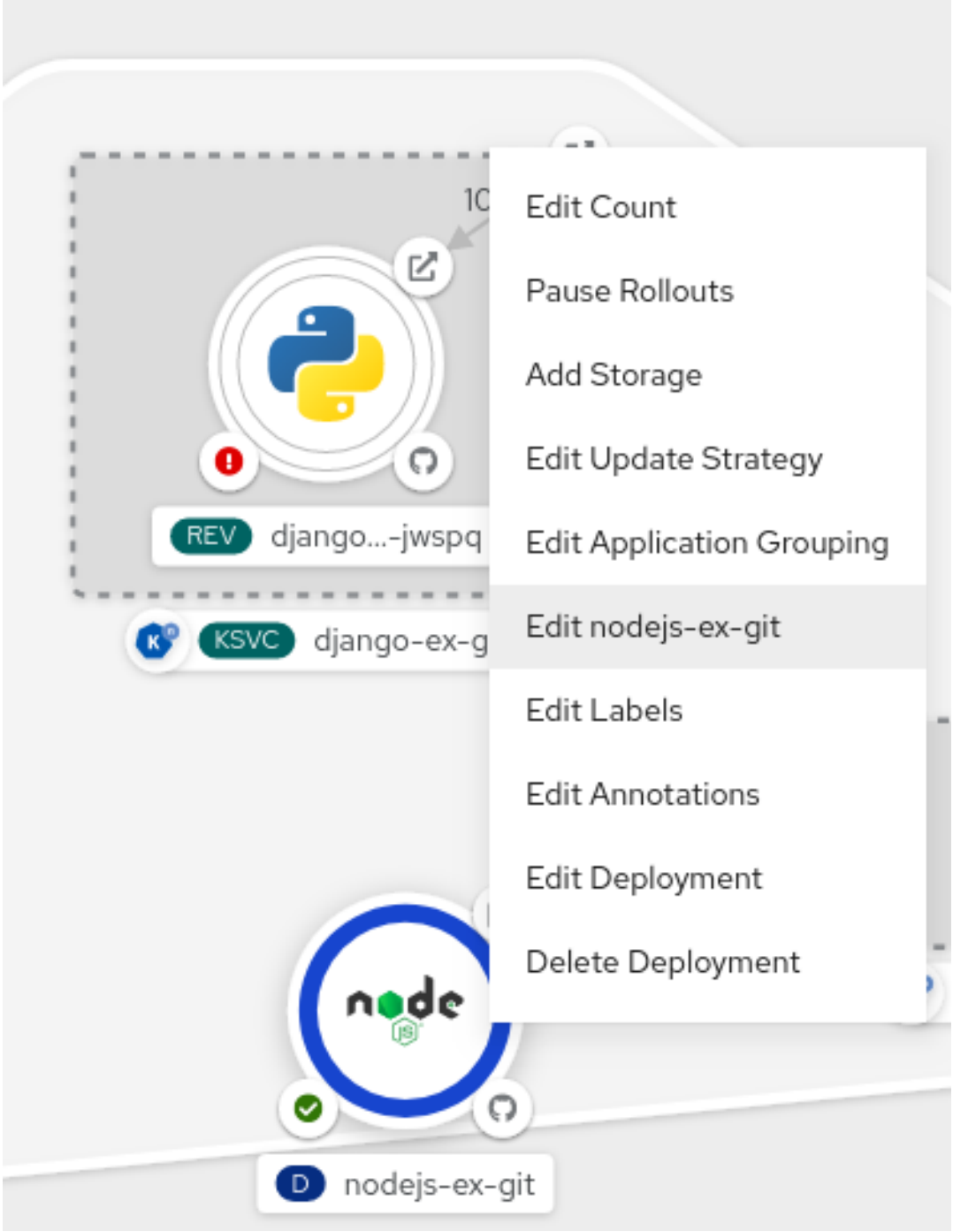
사전 요구 사항

추가 워크플로의 **Git**에서, 컨테이너 이미지, 카탈로그에서 또는 **Dockerfile**에서 옵션을 사용하여 애플리케이션을 생성했는지 확인합니다.

프로세스

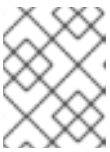
1. 애플리케이션을 생성하고 애플리케이션이 토폴로지 보기에 표시되면 애플리케이션을 마우스 오른쪽 버튼으로 클릭하여 사용 가능한 편집 옵션을 확인합니다.

그림 3.6. 애플리케이션 편집



2. ***애플리케이션 이름*** 편집을 클릭하여 애플리케이션을 생성하는 데 사용한 추가 워크플로를 확인합니다. 해당 양식은 애플리케이션을 생성하는 동안 추가한 값으로 미리 채워져 있습니다.

3. 애플리케이션에 필요한 값을 편집합니다.

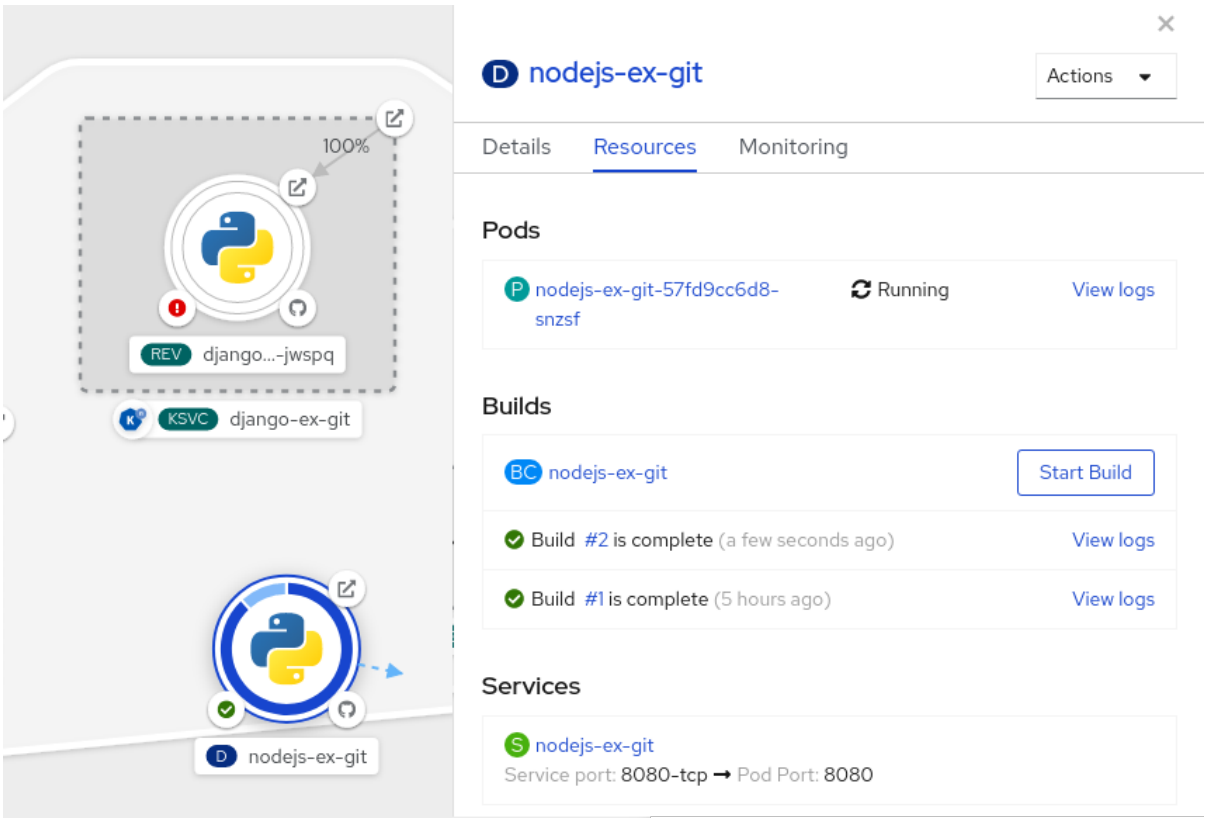


참고

일반 섹션의 이름 필드, CI/CD 파이프라인 또는 고급 옵션 섹션의 애플리케이션의 경로 만들기 필드는 편집할 수 없습니다.

4. 저장을 클릭하여 빌드를 재시작하고 새 이미지를 배포합니다.

그림 3.7. 애플리케이션 편집 및 재배포



3.6. 개발자 화면을 사용하여 HELM 차트 작업

3.6.1. Helm 이해

Helm은 OpenShift Container Platform 클러스터에 대한 애플리케이션 및 서비스 배포를 간소화하는 소프트웨어 패키지 관리자입니다.

Helm은 *차트*라는 패키징 형식을 사용합니다. Helm 차트는 OpenShift Container Platform 리소스에 대해 설명하는 파일 컬렉션입니다.

클러스터에 있는 차트의 실행 중인 인스턴스를 *릴리스*라고 합니다. 클러스터에 차트가 설치될 때마다 새 릴리스가 생성됩니다.

차트를 설치하거나 릴리스를 업그레이드하거나 롤백할 때마다 번호가 증가한 리버전이 생성됩니다.

3.6.1.1. 주요 기능

Helm은 다음을 수행할 수 있는 기능을 제공합니다.

- 차트 리포지토리에 저장된 대규모 차트 컬렉션에서 검색합니다.
- 기존 차트를 수정합니다.
- OpenShift Container Platform 또는 Kubernetes 리소스를 사용하여 자체 차트를 생성합니다.
- 애플리케이션을 차트로 패키징하고 공유합니다.

웹 콘솔에서 개발자 화면을 사용하여 개발자 카탈로그에 나열된 Helm 차트에서 차트를 선택하고 설치할 수 있습니다. 이러한 차트를 사용하여 Helm 릴리스를 생성하고 릴리스를 업그레이드, 롤백, 설치 제거할 수 있습니다.

3.6.2. 사전 요구 사항

- 웹 콘솔에 로그인하여 개발자 화면으로 전환했습니다.

3.6.3. Helm 차트 설치

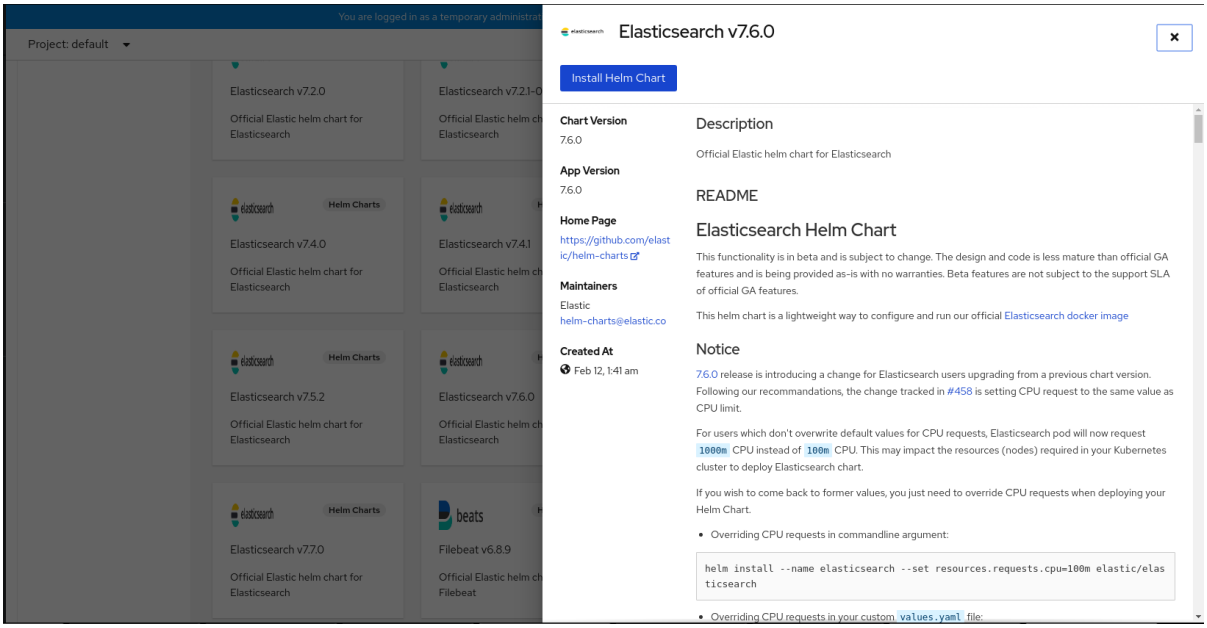
개발자 화면 또는 CLI를 사용하여 Helm 릴리스를 생성하고 웹 콘솔의 개발자 화면에서 확인할 수 있습니다.

프로세스

개발자 카탈로그에 제공된 Helm 차트에서 Helm 릴리스를 생성하려면 다음을 수행합니다.

1. 개발자 화면에서 +추가 보기로 이동하여 프로젝트를 선택합니다. 그런 다음 **Helm** 차트 옵션을 클릭하여 개발자 카탈로그의 모든 Helm 차트를 확인합니다.
2. 차트를 선택하고 해당 차트에 대한 설명, README 및 기타 세부 정보를 읽습니다.
3. **Helm** 차트 설치를 클릭합니다.

   그림 3.8. 개발자 카탈로그의 **Helm** 차트



4. **Helm** 차트 설치 페이지에서 다음을 수행합니다.
   a. 릴리스 이름 필드에 고유한 릴리스 이름을 입력합니다.
   b. 차트 버전 드롭다운 목록에서 필요한 차트 버전을 선택합니다.
   c. 보기에서 또는 **YAML** 보기를 사용하여 Helm 차트를 구성합니다.

      참고

      가능한 경우 **YAML** 보기 및 양식 보기를 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

   d. 설치를 클릭하여 Helm 릴리스를 생성합니다. 릴리스가 표시되는 토폴로지 보기로 리디렉션됩니다. Helm 차트에 릴리스 노트가 있는 경우 이 차트가 사전에 선택되고 오른쪽 패널에 해당 릴리스의 릴리스 노트가 표시됩니다.

측면 패널의 작업 버튼을 사용하거나 Helm 릴리스를 마우스 오른쪽 버튼으로 클릭하여 Helm 릴리스를 업그레이드, 롤백 또는 설치 제거할 수 있습니다.

3.6.4. Helm 릴리스 업그레이드

Helm 릴리스를 업그레이드하여 새 차트 버전으로 업그레이드하거나 릴리스 구성을 업데이트할 수 있습니다.

프로세스

1. 토폴로지 보기에서 Helm 릴리스를 선택하여 측면 패널을 확인합니다.
2. 작업 → **Helm** 릴리스 업그레이드를 클릭합니다.
3. **Helm** 릴리스 업그레이드 페이지에서 업그레이드할 차트 버전을 선택한 다음 업그레이드를 클릭하면 또 하나의 Helm 릴리스가 생성됩니다. **Helm** 릴리스 페이지에 두 가지 리버전이 표시됩니다.

3.6.5. Helm 릴리스 롤백

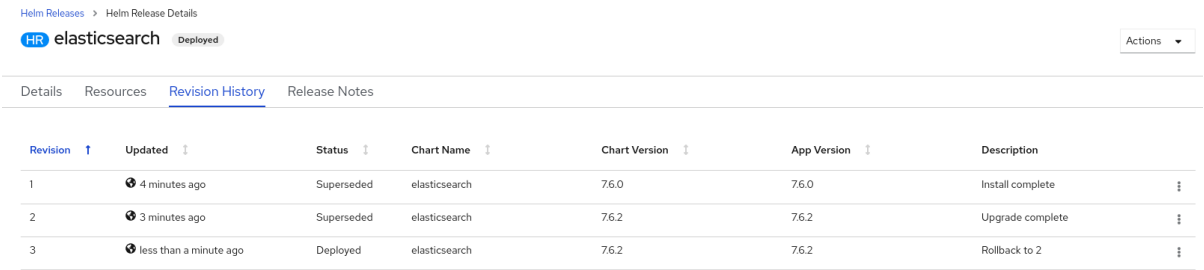
릴리스가 실패하면 Helm 릴리스를 이전 버전으로 롤백할 수 있습니다.

프로세스

Helm 보기를 사용하여 릴리스를 롤백하려면 다음을 수행합니다.

1. 개발자 화면에서 **Helm** 보기로 이동하여 네임스페이스에서 **Helm** 릴리스를 확인합니다.
2. 나열된 릴리스 옆에 있는 옵션 메뉴 를 클릭하고 롤백을 선택합니다.
3. 롤백 **Helm** 릴리스 페이지에서 롤백할 리버전을 선택하고 롤백을 클릭합니다.
4. **Helm** 릴리스 페이지에서 차트를 클릭하여 해당 릴리스의 세부 정보 및 리소스를 확인합니다.
5. 리버전 내역 탭으로 이동하여 차트의 리버전을 모두 확인합니다.

   그림 3.9. Helm 리버전 내역



6. 필요한 경우 특정 리버전 옆에 있는 옵션 메뉴 를 사용하여 롤백할 리버전을 선택할 수 있습니다.

3.6.6. Helm 릴리스 설치 제거

프로세스

1. 토폴로지 보기에서 Helm 릴리스를 마우스 오른쪽 버튼으로 클릭하고 **Helm** 릴리스 설치 제거를 선택합니다.

2. 확인 프롬프트에서 차트 이름을 입력하고 설치 제거를 클릭합니다.

3.7. 애플리케이션 삭제

프로젝트에서 생성한 애플리케이션을 삭제할 수 있습니다.

3.7.1. 개발자 화면을 사용하여 애플리케이션 삭제

개발자 화면에서 토폴로지 보기를 사용하여 애플리케이션 및 모든 관련 구성 요소를 삭제할 수 있습니다.

1. 삭제할 애플리케이션을 클릭하면 측면 패널에 애플리케이션의 리소스 세부 정보가 표시됩니다.

2. 패널 오른쪽에 위에 표시된 작업 드롭다운 메뉴를 클릭하고 애플리케이션 삭제를 선택하면 확인 대화 상자가 표시됩니다.

3. 애플리케이션 이름을 입력하고 삭제를 클릭하여 삭제합니다.

삭제할 애플리케이션을 마우스 오른쪽 버튼으로 클릭하고 애플리케이션 삭제를 클릭하여 삭제할 수도 있습니다.

4장. 배포

4.1. DEPLOYMENT 및 DEPLOYMENTCONFIG 오브젝트 이해

OpenShift Container Platform의 **Deployment** 및 **DeploymentConfig** API 오브젝트에서는 일반적인 사용자 애플리케이션을 세밀하게 관리할 수 있도록 유사하지만 서로 다른 두 가지 방법을 제공합니다. 이러한 오브젝트는 다음과 같이 별도의 API 오브젝트로 구성됩니다.

- **DeploymentConfig** 또는 **Deployment** 오브젝트: 특정 애플리케이션 구성 요소에 대해 원하는 상태를 Pod 템플릿으로 설명합니다.
- **DeploymentConfig** 오브젝트: 특정 시점의 배포 상태 레코드가 Pod 템플릿으로 포함되는 *복제 컨트롤러*가 한 개 이상 포함됩니다. 마찬가지로 배포 오브젝트에는 복제 컨트롤러를 대체하는 *복제본 세트*가 한 개 이상 포함됩니다.
- 하나 이상의 Pod: 특정 버전의 애플리케이션 인스턴스를 나타냅니다.

4.1.1. 배포 블록 빌드

배포 및 배포 구성은 각각 기본 Kubernetes API 오브젝트인 **ReplicaSet** 및 **ReplicationController**를 빌딩 블록으로 사용하여 활성화됩니다.

사용자가 **DeploymentConfig** 오브젝트 또는 배포에서 다루는 복제 컨트롤러, 복제본 세트 또는 Pod를 조작할 필요가 없습니다. 배포 시스템을 통해 변경 사항이 적절하게 전파됩니다.

작은 정보

기존 배포 전략이 사용 사례에 적합하지 않고 배포 라이프사이클 중 수동 단계를 실행해야 하는 경우에는 사용자 정의 배포 전략을 생성해야 합니다.

다음 섹션에서는 이러한 오브젝트에 대해 자세히 설명합니다.

4.1.1.1. 복제 컨트롤러

복제 컨트롤러를 사용하면 항상 지정된 수의 Pod 복제본이 실행됩니다. Pod가 종료되거나 삭제되면 복제 컨트롤러가 작동하여 정의된 수까지 추가로 인스턴스화합니다. 마찬가지로 필요한 것보다 많은 Pod가 실행되고 있는 경우에는 정의된 수에 맞게 필요한 개수의 Pod를 삭제합니다.

복제 컨트롤러 구성은 다음과 같이 구성됩니다.

- 런타임에 조정할 수 있는 원하는 복제본 수
- 복제된 Pod를 생성할 때 사용할 **Pod** 정의
- 관리형 Pod를 확인하는 선택기

선택기는 복제 컨트롤러에서 관리하는 Pod에 할당한 라벨 세트입니다. 이러한 라벨은 복제 컨트롤러에서 인스턴스화하는 **Pod** 정의에 포함됩니다. 복제 컨트롤러에서는 필요에 따라 조정할 수 있도록 선택기를 사용하여 이미 실행 중인 Pod의 인스턴스 수를 결정합니다.

복제 컨트롤러에서 로드나 트래픽을 추적하지 않으므로 로드 또는 트래픽을 기반으로 자동 스케일링하지 않습니다. 대신 외부 Autoscaler에서 복제본 수를 조정해야 합니다.

다음은 복제 컨트롤러 정의의 예입니다.

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 (1)
  selector: (2)
    name: frontend
  template: (3)
    metadata:
      labels: (4)
        name: frontend (5)
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

(1) 실행할 Pod의 사본 수입니다.

(2) 실행할 Pod의 라벨 선택기입니다.

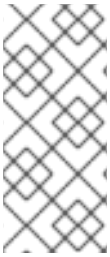
(3) 컨트롤러에서 생성하는 Pod용 템플릿입니다.

(4) Pod의 라벨에는 라벨 선택기의 해당 항목이 포함되어야 합니다.

(5) 매개변수를 확장한 후 최대 이름 길이는 63자입니다.

4.1.1.2. 복제본 세트

복제 컨트롤러와 유사하게 **ReplicaSet**은 지정된 수의 Pod 복제본을 언제든지 실행할 수 있는 기본 Kubernetes API 오브젝트입니다. 복제본 세트와 복제 컨트롤러의 차이점은 복제본 세트는 세트 기반 선택기 요구 사항을 지원하는 반면 복제 컨트롤러는 일치 기반 선택기 요구 사항만 지원한다는 점입니다.



참고

사용자 정의 업데이트 오케스트레이션이 필요한 경우에만 복제본 세트를 사용하거나 업데이트가 필요하지 않습니다. 그러지 않으면 배포를 사용하십시오. 복제본 세트는 독립적으로 사용할 수 있지만 배포에서 Pod 생성, 삭제, 업데이트를 오케스트레이션하는 데 사용합니다. 배포는 복제본 세트를 자동으로 관리하고 Pod에 선언적 업데이트를 제공하며 생성한 복제본 세트를 수동으로 관리할 필요가 없습니다.

다음은 **ReplicaSet** 정의의 예입니다.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend-1
  labels:
    tier: frontend
```

```
spec:
  replicas: 3
  selector: (1)
    matchLabels: (2)
      tier: frontend
    matchExpressions: (3)
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

(1) 리소스 집합에 대한 레이블 쿼리입니다. **matchLabels** 및 **matchExpressions**의 결과는 논리적으로 결합됩니다.

(2) 선택기와 일치하는 라벨을 사용하여 리소스를 지정하는 일치 기반 선택기입니다.

(3) 키를 필터링하는 세트 기반 선택기입니다. 키가 **tier**이고 값이 **frontend**인 모든 리소스를 선택합니다.

4.1.2. DeploymentConfig 오브젝트

OpenShift Container Platform은 복제 컨트롤러를 기반으로 소프트웨어 개발 및 배포 라이프사이클에 대한 지원을 **DeploymentConfig** 오브젝트의 개념으로 확장합니다. 가장 간단한 경우 **DeploymentConfig** 오브젝트는 새 복제 컨트롤러를 생성하고 이 컨트롤러에서 Pod를 시작할 수 있도록 설정합니다.

그러나 **DeploymentConfig** 오브젝트의 OpenShift Container Platform 배포에서는 이미지의 기존 배포에서 새 버전으로 전환하는 기능도 제공하고 복제 컨트롤러를 생성하기 전이나 후에 실행할 후크도 정의합니다.

DeploymentConfig 배포 시스템에서는 다음 기능을 제공합니다.

- 실행 중인 애플리케이션에 대한 템플릿인 **DeploymentConfig** 오브젝트
- 이벤트에 대한 응답으로 자동 배포를 실행하는 트리거
- 이전 버전에서 새 버전으로의 전환을 위한 사용자 정의 가능 배포 전략. 전략은 일반적으로 배포 프로세스라는 Pod 내에서 실행됩니다.
- 배포 라이프사이클 중 다른 시점에서 사용자 정의 동작을 실행하는 일련의 후크(라이프사이클 후크)
- 배포가 실패하는 경우 롤백을 수동 또는 자동으로 지원하기 위한 애플리케이션 버전 관리
- 복제본 수동 스케일링 및 자동 스케일링

DeploymentConfig 오브젝트를 생성하면 **DeploymentConfig** 오브젝트의 Pod 템플릿을 나타내는 복제 컨트롤러가 생성됩니다. 배포가 변경되면 최신 Pod 템플릿을 사용하여 새 복제 컨트롤러가 생성되고 배포 프로세스가 실행되어 이전 복제 컨트롤러가 축소되고 새 복제 컨트롤러가 확장됩니다.

애플리케이션 인스턴스는 생성 시 서비스 로드 밸런서와 라우터 모두에서 자동으로 추가 및 제거됩니다. 애플리케이션에서 **TERM** 신호를 수신할 때 정상 종료를 지원하는 경우 실행 중인 사용자 연결을 정상적으로 완료할 수 있는 기회를 제공할 수 있습니다.

OpenShift Container Platform **DeploymentConfig** 오브젝트는 다음과 같은 세부 정보를 정의합니다.

1. **ReplicationController** 정의의 요소
2. 새 배포를 자동으로 생성하는 트리거
3. 배포 간 전환을 위한 전략
4. 라이프사이클 후크

배포가 수동 또는 자동으로 트리거될 때마다 배포자 Pod에서 배포를 관리합니다(이전 복제 컨트롤러 축소, 새 복제 컨트롤러 확장, 후크 실행 포함). 배포 Pod는 배포 로그를 유지하기 위해 배포 완료 후 무기한으로 유지됩니다. 배포가 다른 배포로 대체되면 필요한 경우 쉽게 롤백할 수 있도록 이전 복제 컨트롤러가 유지됩니다.

DeploymentConfig 정의의 예

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
  - type: ConfigChange 1
  - imageChangeParams:
      automatic: true
      containerNames:
      - helloworld
      from:
        kind: ImageStreamTag
        name: hello-openshift:latest
    type: ImageChange 2
  strategy:
    type: Rolling 3
```

1. 구성 변경 트리거를 사용하면 배포 구성의 Pod 템플릿에서 변경이 탐지될 때마다 새 복제 컨트롤러가 생성됩니다.
2. 이름이 지정된 이미지 스트림에 새 버전의 백업 이미지가 제공될 때마다 이미지 변경 트리거를 통해 새 배포가 생성됩니다.
3. 기본 **Rolling** 전략을 사용하면 배포 사이에 다운타임 없이 전환할 수 있습니다.

4.1.3. 배포

Kubernetes는 OpenShift Container Platform에서 **Deployment**라는 최상위 기본 API 오브젝트 유형을 제공합니다. **Deployment** 오브젝트는 OpenShift Container Platform 관련 **DeploymentConfig** 오브젝트의 하위 항목 역할을 합니다.

Deployment 오브젝트는 **DeploymentConfig** 오브젝트와 같이 특정 애플리케이션 구성 요소의 원하는 상태를 Pod 템플릿으로 설명합니다. 배포에서는 Pod 라이프사이클을 오케스트레이션하는 복제본 세트를 생성합니다.

예를 들어 다음 배포 정의에서는 하나의 **hello-openshift** Pod를 가져오는 복제본 세트를 생성합니다.

배포 정의

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
      - name: hello-openshift
        image: openshift/hello-openshift:latest
        ports:
        - containerPort: 80
```

4.1.4. Deployment 및 DeploymentConfig 오브젝트 비교

OpenShift Container Platform에서는 Kubernetes **Deployment** 오브젝트와 OpenShift Container Platform 제공 **DeploymentConfig** 오브젝트가 모두 지원되지만 **DeploymentConfig** 오브젝트에서 제공하는 특정 기능 또는 동작이 필요하지 않는 한 **Deployment** 오브젝트를 사용하는 것이 좋습니다.

다음 섹션은 두 오브젝트 유형의 차이점에 대해 더 자세히 설명하여 사용할 유형을 결정하는 데 도움이 됩니다.

4.1.4.1. 설계

Deployment 및 **DeploymentConfig** 오브젝트의 중요한 차이점은 각 설계에서 롤아웃 프로세스에 대해 선택한 CAP theorem의 속성입니다. **DeploymentConfig** 오브젝트는 일관성을 선호하는 반면 **Deployments** 오브젝트는 일관성보다 가용성을 우선합니다.

DeploymentConfig 오브젝트의 경우 배포자 Pod를 실행하는 노드가 종료되면 대체되지 않습니다. 이 프로세스는 노드가 다시 온라인 상태가 될 때까지 기다리거나 수동으로 삭제됩니다. 노드를 수동으로 삭제하면 해당 Pod도 삭제됩니다. 즉 kubelet에서 연결된 Pod를 삭제해야 하므로 롤아웃을 해제하기 위해 Pod를 삭제할 수 없습니다.

그러나 배포 롤아웃은 컨트롤러 관리자에서 구동됩니다. 컨트롤러 관리자는 마스터에서 고가용성 모드로 실행되고 리더 선택 알고리즘을 사용하여 일관성보다 가용성을 중시합니다. 오류가 발생하는 동안 기타 마스터가 동일한 배포에서 동시에 작업할 수 있지만 이러한 문제는 오류 발생 직후 조정됩니다.

4.1.4.2. DeploymentConfig 오브젝트별 기능

자동 롤백
현재 배포에서는 배포에 실패하는 경우 성공적으로 배포된 마지막 복제본 세트로 자동 롤백되지 않습니다.

Trigger
배포의 Pod 템플릿이 변경될 때마다 새 롤아웃이 자동으로 트리거된다는 점에서 배포에는 암시적 구성 변경 트리거가 포함되어 있습니다. Pod 템플릿 변경 시 새 롤아웃을 수행하지 않으려면 배포를 정지하십시오.

```
$ oc rollout pause deployments/<name>
```

라이프사이클 후크
배포에서는 아직 라이프사이클 후크를 지원하지 않습니다.

사용자 정의 전략
배포에서는 사용자가 지정하는 사용자 정의 배포 전략을 아직 지원하지 않습니다.

4.1.4.3. 배포별 기능

롤오버
Deployment 오브젝트에 대한 배포 프로세스는 모든 새 롤아웃에 대해 배포자 Pod를 사용하는 **DeploymentConfig** 오브젝트와 달리 컨트롤러 반복문에 의해 실행됩니다. 즉 **Deployment** 오브젝트에는 활성 상태의 복제본 세트가 가능한 한 많이 있을 수 있습니다. 결국 배포 컨트롤러에서 이전 복제본 세트를 축소하고 최신 복제본 세트를 확장합니다.

DeploymentConfig 오브젝트에서는 최대 1개의 배포자 Pod를 실행할 수 있습니다. 그러지 않으면 여러 배포자가 최신이라고 생각하는 복제 컨트롤러를 확장하면서 충돌하게 됩니다. 이로 인해 어느 시점에 두 개의 복제 컨트롤러만 활성화할 수 있습니다. 결국 **Deployment** 오브젝트에 대한 신속한 롤아웃이 더 빨라집니다.

비례 스케일링
배포 컨트롤러는 **Deployment** 오브젝트가 소유한 새 복제본 세트 및 이전 복제본 세트의 크기에 대한 유일한 정보 소스이므로 지속적인 롤아웃을 확장할 수 있습니다. 추가 복제본은 각 복제본 세트의 크기에 비례하여 배포됩니다.

롤아웃이 진행 중이면 컨트롤러에서 새 복제 컨트롤러 크기에 대한 배포자 프로세스에 문제가 발생하므로 **DeploymentConfig** 오브젝트를 확장할 수 없습니다.

롤아웃 중 정지
배포는 언제든지 정지할 수 있습니다. 따라서 지속적인 롤아웃도 정지할 수 있습니다. 반면 현재 배포자 Pod는 정지할 수 없으므로 롤아웃 중 배포를 정지하려고 해도 배포자 프로세스는 영향을 받지 않고 완료될 때까지 계속 수행됩니다.

4.2. 배포 프로세스 관리

4.2.1. DeploymentConfig 오브젝트 관리

DeploymentConfig 오브젝트는 OpenShift Container Platform 웹 콘솔의 워크로드 페이지 또는 **oc** CLI에서 관리할 수 있습니다. 다음 절차에서는 달리 명시하지 않는 한 CLI 사용법을 보여줍니다.

4.2.1.1. 배포 시작

롤아웃을 시작하여 애플리케이션 배포 프로세스를 시작할 수 있습니다.

프로세스

1. 기존 **DeploymentConfig** 오브젝트에서 새 배포 프로세스를 시작하려면 다음 명령을 실행합니다.

```
$ oc rollout latest dc/<name>
```

참고

배포 프로세스가 이미 진행 중인 경우 명령에서 메세지를 표시하고 새 복제 컨트롤러가 배포되지 않습니다.

4.2.1.2. 배포 보기

배포를 보고 애플리케이션의 모든 사용 가능한 리버전에 대한 기본 정보를 얻을 수 있습니다.

프로세스

1. 현재 실행 중인 배포 프로세스를 포함하여 제공된 **DeploymentConfig** 오브젝트에 대해 최근 생성된 모든 복제 컨트롤러에 대한 세부 정보를 보려면 다음 명령을 실행합니다.

```
$ oc rollout history dc/<name>
```

2. 버전과 관련된 세부 정보를 보려면 **--revision** 플래그를 추가합니다.

```
$ oc rollout history dc/<name> --revision=1
```

3. **DeploymentConfig** 오브젝트 및 이 오브젝트의 최신 버전에 대한 자세한 내용을 보려면 **oc describe** 명령을 사용합니다.

```
$ oc describe dc <name>
```

4.2.1.3. 배포 재시도

DeploymentConfig 오브젝트의 현재 리버전을 배포하지 못한 경우 배포 프로세스를 재시작할 수 있습니다.

프로세스

1. 실패한 배포 프로세스를 재시작하려면 다음을 수행합니다.

```
$ oc rollout retry dc/<name>
```

최신 버전이 성공적으로 배포된 경우 명령에서 메시지를 표시하고 배포 프로세스가 다시 수행되지 않습니다.

참고

배포를 다시 시도하면 배포 프로세스가 다시 시작되고 새 배포 리버전이 생성되지 않습니다. 재시작한 복제 컨트롤러의 구성은 실패한 경우의 구성과 동일합니다.

4.2.1.4. 배포 롤백

롤백은 애플리케이션을 이전 리버전으로 되돌리고 REST API, CLI 또는 웹 콘솔을 사용하여 수행할 수 있습니다.

프로세스

1. 마지막으로 성공적으로 배포된 구성 리버전으로 롤백하려면 다음을 수행합니다.

```
$ oc rollout undo dc/<name>
```

DeploymentConfig 오브젝트의 템플릿이 undo 명령에 지정된 배포 리버전과 일치하도록 되돌아가고 새 복제 컨트롤러가 시작됩니다. 리버전이 **--to-revision**을 사용하여 지정되지 않은 경우 마지막으로 성공적으로 배포된 리버전이 사용됩니다.

2. 롤백이 완료되면 실수로 새 배포 프로세스가 시작되지 않도록 **DeploymentConfig** 오브젝트에서 이미지 변경 트리거가 비활성화됩니다.
이미지 변경 트리거를 다시 활성화하려면 다음을 수행합니다.

```
$ oc set triggers dc/<name> --auto
```

참고

배포 구성에서는 최신 배포 프로세스가 실패하는 경우 마지막으로 성공한 구성 리버전으로의 자동 롤백도 지원합니다. 이 경우 배포되지 않은 최신 템플릿은 시스템에서 그대로 유지되며 사용자가 해당 구성을 수정할 수 있습니다.

4.2.1.5. 컨테이너 내에서 명령 실행

이미지의 **ENTRYPOINT** 를 오버레이하여 컨테이너의 시작 동작을 수정하는 명령을 컨테이너에 추가할 수 있습니다. 이 명령은 지정된 시점에 배포당 한 번만 실행할 수 있는 라이프사이클 후크와는 다릅니다.

프로세스

1. **DeploymentConfig** 오브젝트의 **spec** 필드에 **command** 매개변수를 추가합니다. 또한 **command**(**command**가 존재하지 않는 경우 **ENTRYPOINT**)를 수정하는 **args** 필드를 추가할 수 있습니다.

```
spec:
  containers:
  - name: <container_name>
    image: 'image'
    command:
      - '<command>'
    args:
      - '<argument_1>'
      - '<argument_2>'
      - '<argument_3>'
```

예를 들어 **-jar** 및 **/opt/app-root/springboots2idemo.jar** 인수를 사용하여 **java** 명령을 실행하려면 다음을 수행합니다.

```
spec:
  containers:
  - name: example-spring-boot
    image: 'image'
    command:
      - java
    args:
      - '-jar'
      - /opt/app-root/springboots2idemo.jar
```

4.2.1.6. 배포 로그 보기

프로세스

1. 지정된 **DeploymentConfig** 오브젝트의 최신 리버전 로그를 스트리밍하려면 다음을 수행합니다.

```
$ oc logs -f dc/<name>
```

최신 리버전이 실행 중이거나 실패한 경우 명령에서 Pod 배포를 담당하는 프로세스의 로그를 반환합니다. 성공하는 경우 애플리케이션 Pod의 로그를 반환합니다.

2. 또한 실패한 이전 배포 프로세스(복제 컨트롤러 및 배포 Pod)가 존재하고 이를 수동으로 정리하거나 삭제하지 않은 경우에만 이러한 프로세스의 로그를 볼 수 있습니다.

```
$ oc logs --version=1 dc/<name>
```

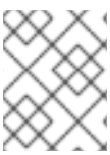
4.2.1.7. 배포 트리거

DeploymentConfig 오브젝트에는 클러스터 내부 이벤트에 대한 응답으로 새 배포 프로세스 생성을 유도하는 트리거가 포함될 수 있습니다.

주의

DeploymentConfig 오브젝트에 트리거가 정의되어 있지 않은 경우 기본적으로 구성 변경 트리거가 추가됩니다. Trigger가 빈 필드로 정의되면 배포를 수동으로 시작해야 합니다.

구성 변경 배포 트리거
구성 변경 트리거를 사용하면 **DeploymentConfig** 오브젝트의 Pod 템플릿에서 구성 변경이 탐지될 때마다 새 복제 컨트롤러가 생성됩니다.



참고

DeploymentConfig 오브젝트에 구성 변경 트리거를 정의하면 **DeploymentConfig** 오브젝트 자체를 생성한 직후 첫 번째 복제 컨트롤러가 자동으로 생성되고 정지되지 않습니다.

구성 변경 배포 트리거

```
triggers:
  - type: "ConfigChange"
```

이미지 변경 배포 트리거
이미지 변경 트리거를 사용하면 이미지 스트림 태그 내용이 변경될 때마다(새 버전의 이미지를 내보낼 때) 새 복제 컨트롤러가 생성됩니다.

이미지 변경 배포 트리거

```
triggers:
  - type: "ImageChange"
    imageChangeParams:
      automatic: true (1)
      from:
        kind: "ImageStreamTag"
        name: "origin-ruby-sample:latest"
        namespace: "myproject"
      containerNames:
        - "helloworld"
```

(1) **imageChangeParams.automatic** 필드를 **false**로 설정하면 트리거가 비활성화됩니다.

위 예제에서 **origin-ruby-sample** 이미지 스트림의 **latest** 태그 값이 변경되고 새 이미지 값이 **DeploymentConfig** 오브젝트의 **helloworld** 컨테이너에 지정된 현재 이미지와 다른 경우 **helloworld** 컨테이너의 새 이미지를 사용하여 새 복제 컨트롤러가 생성됩니다.

참고

DeploymentConfig 오브젝트에 이미지 변경 트리거가 정의되고(구성 변경 트리거와 **automatic=false** 또는 **automatic=true** 사용) 이미지 변경 트리거가 가리키는 이미지 스트림 태그가 아직 존재하지 않는 경우 빌드에서 이미지를 가져오거나 이미지 스트림 태그로 내보내는 즉시 초기 배포 프로세스가 자동으로 시작됩니다.

4.2.1.7.1. 배포 트리거 설정

프로세스

1. **oc set triggers** 명령을 사용하여 **DeploymentConfig** 오브젝트에 대한 배포 트리거를 설정할 수 있습니다. 예를 들어 이미지 변경 트리거를 설정하려면 다음 명령을 사용하십시오.

```
$ oc set triggers dc/<dc_name> \
    --from-image=<project>/<image>:<tag> -c <container_name>
```

4.2.1.8. 배포 리소스 설정

배포는 노드에서 리소스(메모리, CPU, 임시 스토리지)를 사용하는 Pod에 의해 완료됩니다. 기본적으로 Pod는 바인딩되지 않은 노드 리소스를 사용합니다. 그러나 프로젝트에서 기본 컨테이너 제한을 지정하는 경우 Pod는 해당 제한까지 리소스를 사용합니다.

참고

배포를 위한 최소 메모리 제한은 12MB입니다. 메모리를 할당할 수 없음 Pod 이벤트로 인해 컨테이너가 시작되지 않으면 메모리 제한이 너무 낮은 것입니다. 메모리 제한을 늘리거나 제거합니다. 제한을 제거하면 Pod에서 바인딩되지 않은 노드 리소스를 사용할 수 있습니다.

리소스 제한을 배포 전략의 일부로 지정하여 리소스 사용을 제한할 수도 있습니다. 배포 리소스는 재현, 롤링 또는 사용자 정의 배포 전략과 함께 사용할 수 있습니다.

프로세스

1. 다음 예에서 각 **resources**, **cpu**, **memory**, **ephemeral-storage**는 선택 사항입니다.

```
type: "Recreate"
resources:
  limits:
    cpu: "100m" (1)
    memory: "256Mi" (2)
    ephemeral-storage: "1Gi" (3)
```

(1) **CPU** 는 CPU 단위입니다. **100m**은 0.1 CPU 단위(100 * 1e-3)를 나타냅니다.

(2) 메모리 는 바이트 단위입니다. **256Mi** 는 268435456 바이트(256 * 2 ^ 20)를 나타냅니다.

(3) **ephemeral-storage** 는 바이트 단위입니다. **1GI**는 1073741824바이트(2^ 30)를 나타냅니다.

그러나 프로젝트에 할당량을 정의한 경우 다음 두 항목 중 하나가 필요합니다.

- **requests**가 명시적으로 설정된 **resources** 섹션:

```
type: "Recreate"
resources:
  requests: (1)
    cpu: "100m"
    memory: "256Mi"
    ephemeral-storage: "1Gi"
```

(1) **requests** 오브젝트에 할당량의 리소스 목록에 해당하는 리소스 목록이 포함되어 있습니다.

- 프로젝트에 정의된 제한 범위: **LimitRange** 오브젝트의 기본값은 배포 프로세스 중 생성된 Pod에 적용됩니다.

배포 리소스를 설정하려면 위 옵션 중 하나를 선택하십시오. 그러지 않으면 할당량을 충족하지 못하여 배포 Pod가 생성되지 않습니다.

추가 리소스

- 리소스 제한 및 요청에 대한 자세한 내용은 애플리케이션 메모리 관리를 참조하십시오.

4.2.1.9. 수동 스케일링

롤백 외에도 수동으로 스케일링하여 복제본 수를 세부적으로 제어할 수 있습니다.

참고

autoscale 명령을 사용하여 Pod를 자동 스케일링할 수도 있습니다.

프로세스

1. **DeploymentConfig** 오브젝트를 수동으로 스케일링하려면 **oc scale** 명령을 사용합니다. 예를 들어 다음 명령은 **frontend DeploymentConfig** 오브젝트의 복제본 수를 **3**으로 설정합니다.

```
$ oc scale dc frontend --replicas=3
```

결국 복제본 수는 **DeploymentConfig** 오브젝트의 **frontend**에서 구성한 배포의 원하는 현재 상태로 전달됩니다.

4.2.1.10. DeploymentConfig 오브젝트에서 프라이빗 리포지토리에 액세스

프라이빗 리포지토리에서 이미지에 액세스할 수 있도록 **DeploymentConfig** 오브젝트에 시크릿을 추가할 수 있습니다. 이 절차에서는 OpenShift Container Platform 웹 콘솔 방법을 보여줍니다.

프로세스

1. 새 프로젝트를 생성합니다.
2. 워크로드 페이지에서 프라이빗 이미지 리포지토리에 액세스하는 데 필요한 인증 정보가 포함된 시크릿을 생성합니다.
3. **DeploymentConfig** 오브젝트를 생성합니다.
4. **DeploymentConfig** 오브젝트 편집기 페이지에서 시크릿 가져오기를 설정하고 변경 사항을 저장합니다.

4.2.1.11. 특정 노드에 Pod 할당

라벨이 지정된 노드와 함께 노드 선택기를 사용하여 Pod 배치를 제어할 수 있습니다.

클러스터 관리자는 Pod 배치를 특정 노드로 제한하기 위해 프로젝트의 기본 노드 선택기를 설정할 수 있습니다. 개발자는 **Pod** 구성에 노드 선택기를 설정하여 노드를 추가로 제한할 수 있습니다.

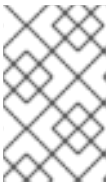
프로세스

1. Pod를 생성할 때 노드 선택기를 추가하려면 **Pod** 구성을 편집하고 **nodeSelector** 값을 추가합니다. 이 값은 단일 **Pod** 구성 또는 **Pod** 템플릿에 추가할 수 있습니다.

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    disktype: ssd
...
```

노드 선택기가 제 위치에 있으면 생성된 Pod가 라벨이 지정된 노드에 할당됩니다. 여기 지정된 라벨은 클러스터 관리자가 추가한 라벨과 함께 사용됩니다.

예를 들어 클러스터 관리자가 프로젝트에 **type=user-node** 및 **region=east** 라벨을 추가하고 위의 **disktype: ssd** 라벨을 Pod에 추가하는 경우 Pod는 세 라벨이 모두 있는 노드에서만 예약됩니다.



참고

라벨은 하나의 값으로만 설정할 수 있으므로 관리자가 설정한 기본값이 **region=east**인 **Pod** 구성에 **region=west**인 노드 선택기를 설정하면 예약되지 않는 Pod가 생성됩니다.

4.2.1.12. 다른 서비스 계정으로 Pod 실행

기본 계정 이외의 서비스 계정으로 Pod를 실행할 수 있습니다.

프로세스

1. **DeploymentConfig** 오브젝트를 편집합니다.

```
$ oc edit dc/<deployment_config>
```

2. **serviceAccount** 및 **serviceAccountName** 매개변수를 **spec** 필드에 추가하고 사용할 서비스 계정을 지정합니다.

```
spec:
  securityContext: {}
  serviceAccount: <service_account>
  serviceAccountName: <service_account>
```

4.3. 배포 전략 사용

*배포 전략*은 애플리케이션을 변경하거나 업그레이드하는 방법입니다. 사용자가 개선 작업을 거의 알아채지 못하도록 다운타임 없이 변경하는 것이 목표입니다.

최종 사용자는 일반적으로 라우터에서 처리한 경로를 통해 애플리케이션에 액세스하므로 배포 전략에서는 **DeploymentConfig** 오브젝트 기능 또는 라우팅 기능에 중점을 둘 수 있습니다. 배포에 중점을 둔 전략은 애플리케이션을 사용하는 모든 경로에 영향을 미칩니다. 라우터 기능을 사용하는 전략은 개별 경로를 대상으로 합니다.

대다수의 배포 전략은 **DeploymentConfig** 오브젝트를 통해 지원되며 일부 추가 전략은 라우터 기능을 통해 지원됩니다. 이 섹션에서는 배포 전략에 대해 설명합니다.

배포 전략 선택

배포 전략을 선택할 때는 다음을 고려하십시오.

- 장기 실행 연결은 정상적으로 처리해야 합니다.
- 데이터베이스 변환은 복잡할 수 있으며 애플리케이션과 함께 수행하고 롤백해야 합니다.
- 애플리케이션이 마이크로 서비스와 기존 구성 요소의 하이브리드인 경우 전환을 완료하기 위해 다운타임이 필요할 수 있습니다.
- 이 작업을 수행하려면 인프라가 있어야 합니다.

- 격리되지 않은 테스트 환경이 있는 경우 새 버전과 이전 버전을 모두 중단할 수 있습니다.

배포 전략에서는 준비 상태 점검을 통해 새 Pod를 사용할 준비가 되었는지 확인합니다. 준비 상태 점검이 실패하면 **DeploymentConfig** 오브젝트에서 타임아웃될 때까지 Pod를 다시 실행하려고 합니다. 기본 타임아웃은 **dc.spec.strategy.*params**의 **TimeoutSeconds**에 설정된 값인 **10m**입니다.

4.3.1. 롤링 전략

롤링 배포를 통해 이전 버전의 애플리케이션 인스턴스를 새 버전의 애플리케이션 인스턴스로 대체합니다. 롤링 전략은 **DeploymentConfig** 오브젝트에 전략이 지정되지 않은 경우 사용되는 기본 배포 전략입니다.

롤링 배포는 일반적으로 새 Pod가 준비 상태 점검을 통해 준비 상태가 될 때까지 기다린 후 이전 구성 요소를 축소합니다. 심각한 문제가 발생하는 경우 롤링 배포가 중단될 수 있습니다.

롤링 배포를 사용하는 경우는 다음과 같습니다.

- 애플리케이션을 업데이트하는 동안 다운타임이 발생하지 않도록 하려는 경우
- 애플리케이션에서 이전 코드 및 새 코드가 동시에 실행되도록 지원하는 경우

롤링 배포에서는 코드의 이전 버전과 새 버전이 동시에 실행됩니다. 이를 위해서는 일반적으로 애플리케이션에서 N-1 호환성을 처리해야 합니다.

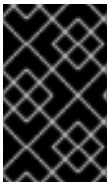
롤링 전략 정의의 예

```
strategy:
  type: Rolling
  rollingParams:
    updatePeriodSeconds: 1 (1)
    intervalSeconds: 1 (2)
    timeoutSeconds: 120 (3)
    maxSurge: "20%" (4)
    maxUnavailable: "10%" (5)
    pre: {} (6)
    post: {}
```

1. 개별 Pod 업데이트 사이에 대기하는 시간입니다. 이 값을 지정하지 않는 경우 기본값은 **1**입니다.
2. 업데이트 후 배포 상태를 폴링할 때까지 대기하는 시간입니다. 이 값을 지정하지 않는 경우 기본값은 **1**입니다.
3. 스케일링 이벤트를 중지하기 전에 대기하는 시간입니다. 선택 사항이며 기본값은 **600**입니다. 여기서 *중지*하는 경우 이전에 완료된 배포로 자동으로 롤백합니다.
4. **maxSurge**는 선택 사항이며 지정하지 않는 경우 기본값은 **25%**입니다. 다음 절차 아래의 정보를 참조하십시오.
5. **maxUnavailable**은 선택 사항이며 지정하지 않는 경우 기본값은 **25%**입니다. 다음 절차 아래의 정보를 참조하십시오.
6. **pre**와 **post**는 둘 다 라이프사이클 후크입니다.

롤링 전략:

1. **pre** 라이프사이클 후크를 실행합니다.
2. 서지 수에 따라 새 복제 컨트롤러를 확장합니다.
3. 사용할 수 없는 최대 수에 따라 이전 복제 컨트롤러를 축소합니다.
4. 새 복제 컨트롤러가 원하는 복제본 수에 도달하고 이전 복제 컨트롤러가 0으로 스케일링될 때까지 이 스케일링을 반복합니다.
5. **post** 라이프사이클 후크를 실행합니다.



중요

축소할 때는 롤링 전략이 Pod가 준비될 때까지 기다립니다. 따라서 추가 스케일링이 가용성에 영향을 미치는지 확인할 수 있습니다. 확장된 Pod가 준비되지 않으면 배포 프로세스가 결국 타임아웃되어 배포에 실패합니다.

maxUnavailable 매개변수는 업데이트 중 사용할 수 없는 최대 Pod 수입니다. **maxSurge** 매개변수는 원래 Pod 수 이상으로 예약할 수 있는 최대 Pod 수입니다. 두 매개변수 모두 백분율 (예: **10%**) 또는 절대 값 (예: **2**)으로 설정할 수 있습니다. 기본값은 둘 다 **25%**입니다.

이러한 매개변수를 사용하면 가용성 및 속도를 위해 배포를 조정할 수 있습니다. 예를 들면 다음과 같습니다.

- **maxUnavailable*=0** 및 **maxSurge*=20%**를 사용하면 업데이트 및 빠른 확장 중 전체 용량을 유지 관리할 수 있습니다.
- **maxUnavailable*=10%** 및 **maxSurge*=0**은 추가 용량을 사용하지 않고 업데이트를 수행합니다 (내부 업데이트).
- **maxUnavailable*=10%** 및 **maxSurge*=10%**는 빠르게 확장 및 축소하고 약간의 용량 손실 가능성이 있습니다.

일반적으로 빠른 롤아웃을 원한다면 **maxSurge**를 사용합니다. 리소스 할당량을 고려해야 하고 부분적인 비가용성을 허용할 수 있는 경우에는 **maxUnavailable**을 사용합니다.

4.3.1.1. 카나리아 배포

OpenShift Container Platform의 모든 롤링 배포는 *카나리아 배포*입니다. 이전 인스턴스를 모두 교체하기 전에 새 버전(카나리아)을 테스트합니다. 준비 상태 점검이 실패하면 카나리아 인스턴스가 제거되고 **DeploymentConfig** 오브젝트가 자동으로 롤백됩니다.

준비 상태 점검은 애플리케이션 코드의 일부이며 새 인스턴스를 사용할 준비가 되었는지 확인하는 데 필요할 정도로 정교할 수 있습니다. 더 복잡한 애플리케이션 점검을 구현해야 하는 경우(예: 실제 사용자 워크로드를 새 인스턴스로 전송) 사용자 정의 배포를 구현하거나 Blue-Green 배포 전략을 사용하는 것이 좋습니다.

4.3.1.2. 롤링 배포 생성

롤링 배포는 OpenShift Container Platform의 기본 유형입니다. CLI를 사용하여 롤링 배포를 생성할 수 있습니다.

프로세스

1. Quay.io에 있는 배포 이미지 예제를 기반으로 애플리케이션을 생성합니다.

```
$ oc new-app quay.io/openshifttest/deployment-example:latest
```

2. 라우터가 설치된 경우 경로를 통해 애플리케이션을 제공하거나 서비스 IP를 직접 사용합니다.

```
$ oc expose svc/deployment-example
```

3. **deployment-example.<project>.<router_domain>**에서 애플리케이션을 검색하여 **v1** 이미지가 표시되는지 확인합니다.

4. **DeploymentConfig** 오브젝트를 세 개의 복제본으로 확장합니다.

```
$ oc scale dc/deployment-example --replicas=3
```

5. 예제의 새 버전에 **latest** 태그를 지정하여 새 배포를 자동으로 트리거합니다.

```
$ oc tag deployment-example:v2 deployment-example:latest
```

6. 브라우저에서 **v2** 이미지가 표시될 때까지 페이지를 새로 고칩니다.

7. CLI를 사용하는 경우 다음 명령은 버전 1의 Pod 수와 버전 2의 Pod 수를 보여줍니다. 웹 콘솔에서 Pod가 점차 v2에 추가되고 v1에서 제거됩니다.

```
$ oc describe dc deployment-example
```

배포 프로세스 중 새 복제 컨트롤러가 점점 확장됩니다. 새 Pod가 (준비 상태 점검을 통과하여) **ready** 상태로 표시되면 배포 프로세스가 계속됩니다.

Pod가 준비 상태가 되지 않으면 프로세스가 중단되고 배포가 이전 버전으로 롤백됩니다.

4.3.1.3. 개발자 화면을 사용하여 롤링 배포 시작

사전 요구 사항

- 웹 콘솔의 개발자 화면에 있는지 확인합니다.
- 추가 보기를 사용하여 애플리케이션을 생성하고 토폴로지 보기에 배포되었는지 확인합니다.

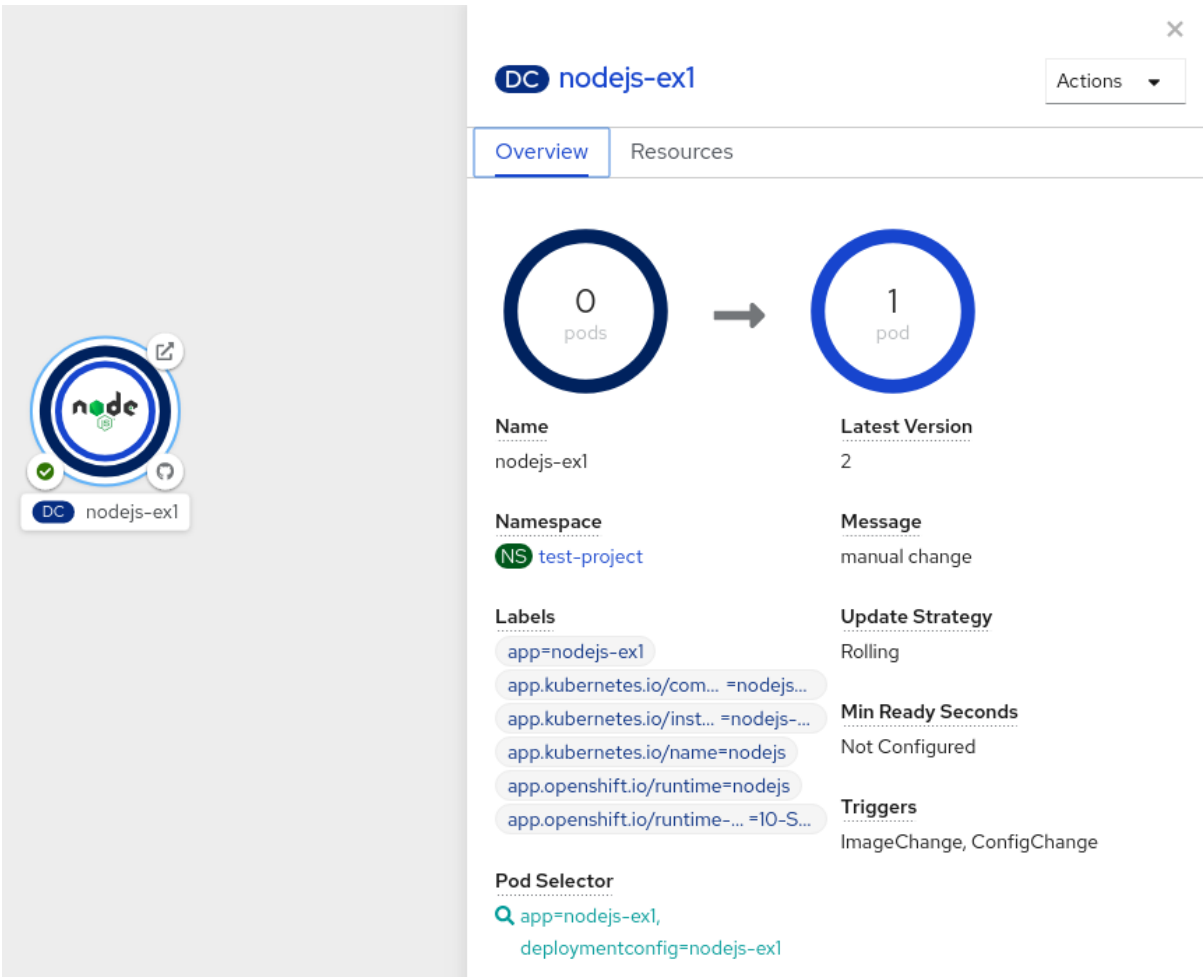
프로세스

롤링 배포를 시작하여 애플리케이션을 업그레이드하려면 다음을 수행합니다.

1. 개발자 화면의 토폴로지 보기에서 애플리케이션 노드를 클릭하여 측면 패널의 개요 탭을 확인합니다. 업데이트 전략은 기본값인 롤링 전략으로 설정되어 있습니다.

2. 작업 드롭다운 메뉴에서 롤아웃 시작을 선택하여 롤링 업데이트를 시작합니다. 롤링 배포에서는 새 버전의 애플리케이션을 실행한 다음 이전 버전을 종료합니다.

그림 4.1. 롤링 업데이트



추가 리소스

- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션 생성 및 배포
- 프로젝트에서 애플리케이션 보기

4.3.2. 재현 전략

재현 전략에는 기본 롤아웃 동작이 있고 배포 프로세스에 코드를 삽입하는 라이프사이클 후크를 지원합니다.

재현 전략 정의의 예

```
strategy:
  type: Recreate
  recreateParams: (1)
    pre: {} (2)
    mid: {}
    post: {}
```

(1) **recreateParams**는 선택 사항입니다.

(2) **pre**, **mid**, **post**는 라이프사이클 후크입니다.

재현 전략은 다음과 같습니다.

1. **pre** 라이프사이클 후크를 실행합니다.
2. 이전 배포를 0으로 축소합니다.
3. **mid** 라이프사이클 후크를 실행합니다.
4. 새 배포를 확장합니다.
5. **post** 라이프사이클 후크를 실행합니다.

중요

확장하는 동안 배포의 복제본 수가 두 개 이상인 경우 배포를 완전히 확장하기 전에 첫 번째 배포 복제본의 준비 상태를 검증합니다. 첫 번째 복제본의 검증이 실패하면 배포가 실패로 간주됩니다.

재현 배포를 사용하는 경우는 다음과 같습니다.

- 새 코드가 시작되려면 마이그레이션 또는 기타 데이터 변환 작업을 실행해야 하는 경우
- 새 버전과 이전 버전의 애플리케이션 코드가 동시에 실행되는 것을 지원하지 않는 경우
- 여러 복제본에서 공유할 수 없는 RWO 볼륨을 사용하려는 경우

재현 배포에서는 잠시 동안 애플리케이션 인스턴스가 실행되지 않기 때문에 다운타임이 발생합니다. 그러나 기존 코드와 새 코드가 동시에 실행되지 않습니다.

4.3.3. 개발자 화면을 사용하여 재현 배포 시작

웹 콘솔의 개발자 화면을 사용하여 배포 전략을 기본 롤링 업데이트에서 재현 업데이트로 전환할 수 있습니다.

사전 요구 사항

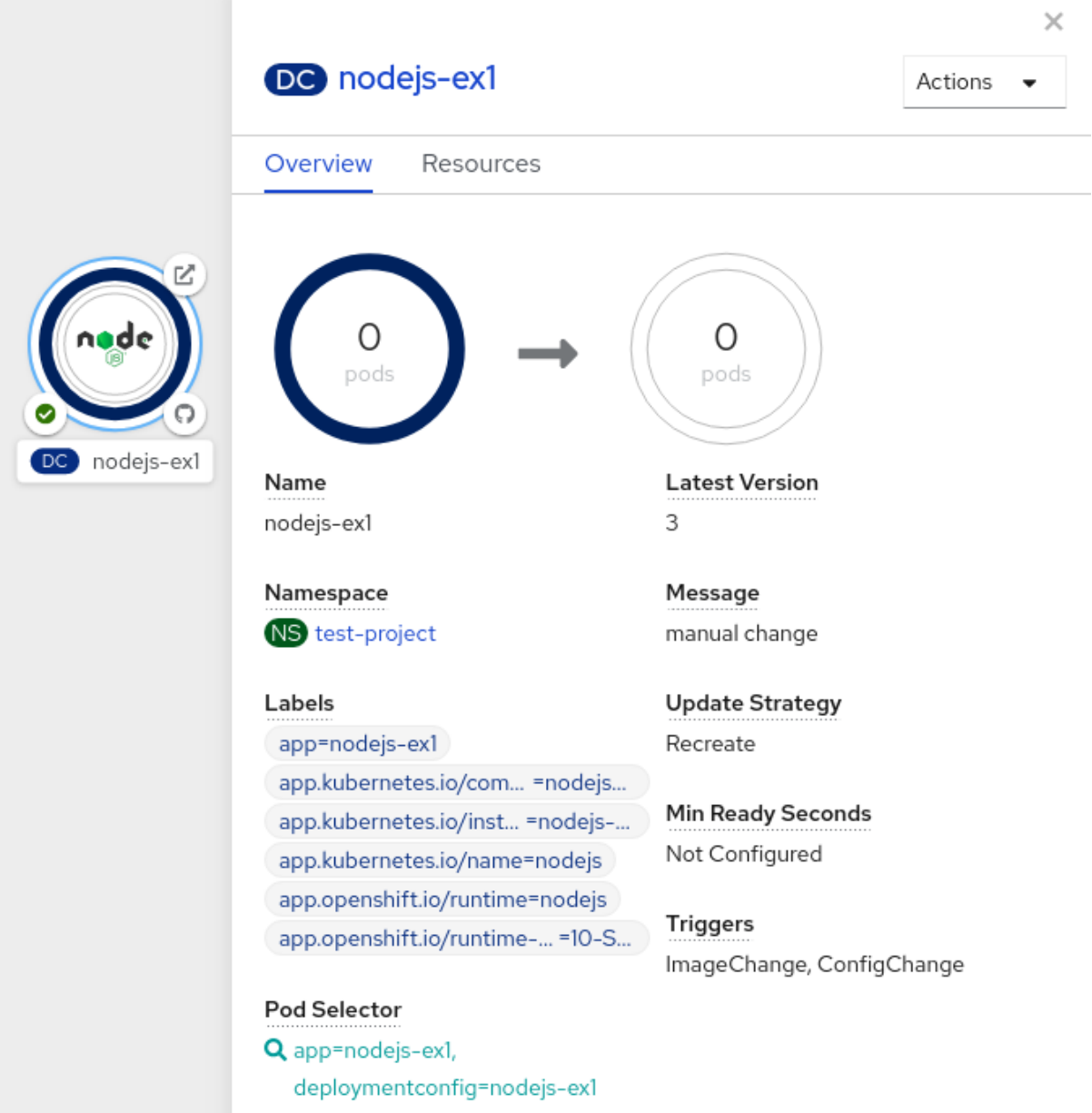
- 웹 콘솔의 개발자 화면에 있는지 확인합니다.
- 추가 보기를 사용하여 애플리케이션을 생성하고 토폴로지 보기에 배포되었는지 확인합니다.

프로세스

재현 업데이트 전략으로 전환하고 애플리케이션을 업그레이드하려면 다음을 수행합니다.

1. 작업 드롭다운 메뉴에서 배포 구성 편집을 선택하여 애플리케이션의 배포 구성 세부 정보를 확인합니다.
2. YAML 편집기에서 **spec.strategy.type**을 **Recreate**로 변경하고 저장을 클릭합니다.
3. 토폴로지 보기에서 노드를 선택하여 측면 패널의 개요 탭을 확인합니다. 이제 업데이트 전략이 재현으로 설정되어 있습니다.
4. 작업 드롭다운 메뉴에서 롤아웃 시작을 선택하면 재현 전략을 사용하여 업데이트가 시작됩니다. 재현 전략에서는 먼저 이전 버전 애플리케이션에 대한 Pod를 종료한 다음 새 버전에 대한 Pod를 구동합니다.

그림 4.2. 재현 업데이트



추가 리소스

- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션 생성 및 배포
- 프로젝트에서 애플리케이션 보기

4.3.4. 사용자 정의 전략

사용자 정의 전략을 사용하면 고유의 배포 동작을 제공할 수 있습니다.

사용자 정의 전략 정의의 예

```
strategy:
  type: Custom
  customParams:
    image: organization/strategy
    command: [ "command", "arg1" ]
```

```
    environment:
      - name: ENV_1
        value: VALUE_1
```

위 예에서 **organization/strategy** 컨테이너 이미지는 배포 동작을 제공합니다. 선택 사항인 **command** 배열은 이미지의 **Dockerfile**에 지정된 모든 **CMD** 지시문을 재정의합니다. 제공되는 선택적 환경 변수는 전략 프로세스의 실행 환경에 추가됩니다.

또한 OpenShift Container Platform에서는 배포 프로세스에 다음 환경 변수를 제공합니다.

환경 변수	설명
OPENSHIFT_DEPLOYMENT_NAME	복제 컨트롤러의 새 배포 이름입니다.
OPENSHIFT_DEPLOYMENT_NAMESPACE	새 배포의 이름 공간입니다.

처음에는 새 배포의 복제본 수가 0입니다. 이 전략은 사용자의 요구에 가장 적합한 논리를 사용하여 새 배포를 활성화하는 작업을 담당합니다.

또는 **customParams** 오브젝트를 사용하여 기존 배포 전략에 사용자 정의 배포 논리를 삽입합니다. 사용자 정의 쉘 스크립트 논리를 제공하고 **openshift-deploy** 바이너리를 호출합니다. 사용자는 사용자 정의 배포 컨테이너 이미지를 제공할 필요가 없습니다. 이 경우 기본 OpenShift Container Platform 배포자 이미지가 대신 사용됩니다.

```
strategy:
  type: Rolling
  customParams:
    command:
    - /bin/sh
    - -c
    - |
      set -e
      openshift-deploy --until=50%
      echo Halfway there
      openshift-deploy
      echo Complete
```

그러면 다음 배포가 생성됩니다.

```
Started deployment #2
--> Scaling up custom-deployment-2 from 0 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
    Scaling custom-deployment-2 up to 1
--> Reached 50% (currently 50%)
Halfway there
--> Scaling up custom-deployment-2 from 1 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
    Scaling custom-deployment-1 down to 1
    Scaling custom-deployment-2 up to 2
```

```
    Scaling custom-deployment-1 down to 0
--> Success
Complete
```

사용자 정의 배포 전략 프로세스에서 OpenShift Container Platform API 또는 Kubernetes API에 액세스해야 하는 경우에는 전략을 실행하는 컨테이너에서 인증을 위해 컨테이너 내에 제공되는 서비스 계정 토큰을 사용하면 됩니다.

4.3.5. 라이프사이클 후크

롤링 및 재현 전략에서는 *라이프사이클 후크* 또는 배포 후크를 지원하므로 전략 내의 미리 정의한 지점에서 배포 프로세스에 동작을 삽입할 수 있습니다.

pre 라이프사이클 후크의 예

```
pre:
  failurePolicy: Abort
  execNewPod: {} 1
```

1 **execNewPod**는 Pod 기반 라이프사이클 후크입니다.

모든 후크에는 후크에 오류가 발생했을 때 전략에서 취해야 하는 조치를 정의하는 *실패 정책*이 있습니다.

Abort	후크가 실패하면 배포 프로세스가 실패로 간주됩니다.
Retry	성공할 때까지 후크를 다시 실행합니다.
Ignore	모든 후크 오류를 무시하고 배포를 계속 진행합니다.

후크에는 후크 실행 방법을 설명하는 유형별 필드가 있습니다. 현재는 Pod 기반 후크가 지원되는 유일한 후크 유형으로 **execNewPod**필드에 지정되어 있습니다.

Pod 기반 라이프사이클 후크
Pod 기반 라이프사이클 후크는 **DeploymentConfig** 오브젝트의 템플릿에서 파생된 새 Pod에서 후크 코드를 실행합니다.

간소화된 다음 예제 배포에서는 롤링 전략을 사용합니다. 간결성을 위해 트리거 및 몇 가지 기타 사소한 세부 정보는 생략되었습니다.

```
kind: DeploymentConfig
apiVersion: v1
metadata:
  name: frontend
spec:
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
        - name: helloworld
          image: openshift/origin-ruby-sample
```

```
replicas: 5
selector:
  name: frontend
strategy:
  type: Rolling
  rollingParams:
    pre:
      failurePolicy: Abort
      execNewPod:
        containerName: helloworld (1)
        command: [ "/usr/bin/command", "arg1", "arg2" ] (2)
        env: (3)
          - name: CUSTOM_VAR1
            value: custom_value1
        volumes:
          - data (4)
```

(1) **helloworld** 이름은 **spec.template.spec.containers[0].name**을 나타냅니다.

(2) 이 **command**는 **openshift/origin-ruby-sample** 이미지로 정의된 모든 **ENTRYPOINT**를 재정의합니다.

(3) **env**는 후크 컨테이너의 선택적 환경 변수 세트입니다.

(4) **volume**은 후크 컨테이너의 선택적 볼륨 참조 세트입니다.

이 예제에서 **pre** 후크는 **helloworld** 컨테이너의 **openshift/origin-ruby-sample** 이미지를 사용하여 새 Pod에서 실행됩니다. 후크 Pod에는 다음과 같은 속성이 있습니다.

- 후크 명령은 **/usr/bin/command arg1 arg2**입니다.
- 후크 컨테이너에는 **CUSTOM_VAR1=custom_value1** 환경 변수가 있습니다.
- 후크 실패 정책이**Abort**이므로 후크가 실패하면 배포 프로세스가 실패합니다.
- 후크 Pod는 **DeploymentConfig** 오브젝트 Pod의 **data** 볼륨을 상속합니다.

4.3.5.1. 라이프사이클 후크 설정

CLI를 사용하여 배포에 라이프사이클 후크 또는 배포 후크를 설정할 수 있습니다.

프로세스

1. **oc set deployment-hook** 명령을 사용하여 원하는 후크 유형을 **--pre**, **--mid** 또는 **--post**로 설정합니다. 예를 들어 사전 배포 후크를 설정하려면 다음을 실행합니다.

```
$ oc set deployment-hook dc/frontend \
    --pre -c helloworld -e CUSTOM_VAR1=custom_value1 \
    --volumes data --failure-policy=abort -- /usr/bin/command arg1 arg2
```

4.4. 경로 기반 배포 전략 사용

배포 전략에서는 애플리케이션을 개발하는 방식을 제공합니다. 일부 전략에서는 **Deployment** 오브젝트를 사용하여 애플리케이션으로 확인되는 모든 경로의 사용자가 확인할 수 있는 변경을 수행합니다. 이 섹

션에 설명된 것과 같은 기타 고급 전략에서는 **Deployment** 오브젝트와 함께 라우터 기능을 사용하여 특정 경로에 영향을 미칩니다.

가장 일반적인 경로 기반 전략은 *Blue-Green 배포*를 사용하는 것입니다. 테스트 및 평가를 위해 새 버전(Green 버전)이 제공되지만 사용자는 여전히 안정적인 버전(Blue 버전)을 사용합니다. 사용자는 준비가 되면 Green 버전으로 전환합니다. 문제가 발생하면 Blue 버전으로 다시 전환할 수 있습니다.

일반적인 대체 전략은 동시에 활성 상태인 *A/B 버전*을 사용하는 것입니다. 즉 일부 사용자는 한 버전을 사용하고 나머지 사용자는 다른 버전을 사용합니다. 이 전략은 사용자 인터페이스 변경 및 기타 기능을 실험하여 사용자 피드백을 받는 데 사용할 수 있습니다. 또한 문제가 제한된 수의 사용자에게 영향을 미치는 프로덕션 컨텍스트에서 작업이 적절한지 확인하는 데 사용할 수도 있습니다.

카나리아 배포는 새 버전을 테스트하지만 문제가 발견되면 신속하게 이전 버전으로 대체합니다. 이는 위의 두 전략 모두에서 수행할 수 있습니다.

경로 기반 배포 전략에서는 서비스의 Pod 수를 확장하지 않습니다. 원하는 성능 특성을 유지 관리하려면 배포 구성을 확장해야 할 수 있습니다.

4.4.1. 프록시 shard 및 트래픽 분할

프로덕션 환경에서는 특정 shard에 전달되는 트래픽 분배를 정확하게 제어할 수 있습니다. 많은 수의 인스턴스를 처리할 때는 개별 shard의 상대적 스케일을 사용하여 백분율 기반 트래픽을 구현할 수 있습니다. 이는 수신하는 트래픽을 다른 곳에서 실행되는 별도의 서비스 또는 애플리케이션으로 전달하거나 분할하는 *프록시/shard*와 잘 결합합니다.

가장 간단한 구성에서 프록시는 변경되지 않은 요청을 전달합니다. 더 복잡한 설정에서는 들어오는 요청을 복제하여 애플리케이션의 로컬 인스턴스와 별도의 클러스터에 보내 결과를 비교할 수 있습니다. 기타 패턴에는 DR 설치 캐시를 준비 상태로 유지하거나 분석을 위해 들어오는 트래픽을 샘플링하는 작업이 포함됩니다.

모든 TCP(또는 UDP) 프록시는 원하는 shard에서 실행할 수 있습니다. **oc scale** 명령을 사용하여 프록시 shard에서 요청을 처리하는 상대적 인스턴스 수를 변경합니다. 더 복잡한 트래픽 관리에는 비례 분산 기능을 사용하여 OpenShift Container Platform 라우터를 사용자 정의하는 것이 좋습니다.

4.4.2. N-1 호환성

새 코드와 이전 코드가 동시에 포함된 애플리케이션에서는 새 코드에서 작성한 데이터를 이전 버전의 코드에서 읽고 처리(또는 정상적으로 무시)할 수 있도록 주의해야 합니다. 이는 *스키마 진화*라고도 하며 복잡한 문제에 해당합니다.

디스크, 데이터베이스, 임시 캐시에 저장된 데이터 또는 사용자 브라우저 세션의 일부 등 다양한 형태를 취할 수 있습니다. 대부분의 웹 애플리케이션에서는 롤링 배포를 지원할 수 있지만 이를 처리할 수 있도록 애플리케이션을 테스트하고 설계하는 것이 중요합니다.

일부 애플리케이션의 경우 이전 코드와 새 코드가 나란히 실행되는 기간이 짧기 때문에 버그 또는 일부 실패한 사용자 트랜잭션이 허용됩니다. 다른 애플리케이션의 경우 실패 패턴으로 인해 전체 애플리케이션이 작동하지 않을 수 있습니다.

N-1 호환성을 검증하는 한 가지 방법은 A/B 배포를 사용하는 것입니다. 즉 테스트 환경에서 제어되는 방식으로 이전 코드와 새 코드를 동시에 실행한 후 새 배포로 이동하는 트래픽으로 인해 이전 배포가 실패하지 않는지 확인하는 것입니다.

4.4.3. 정상적인 종료

OpenShift Container Platform 및 Kubernetes는 부하 분산 순환 작업에서 애플리케이션 인스턴스를 제거하기 전에 종료할 시간을 제공합니다. 그러나 애플리케이션은 종료하기 전에 사용자 연결을 명확하게 종료해야 합니다.

종료 시 OpenShift Container Platform은 컨테이너의 프로세스에 **TERM** 신호를 보냅니다. 애플리케이션 코드는 **SIGTERM**을 수신하면 새 연결을 허용하지 않습니다. 이로 인해 로드 밸런서에서 트래픽을 활성 상태의 다른 인스턴스로 라우팅합니다. 그러면 애플리케이션 코드는 열려 있는 모든 연결이 닫힐 때까지 기다리거나 종료되기 전 다음 기회에 개별 연결을 정상적으로 종료합니다.

정상 종료 기간이 만료된 후 종료되지 않은 프로세스에는 **KILL** 신호가 전송되어 프로세스가 즉시 종료됩니다. Pod 또는 Pod 템플릿의 **terminationGracePeriodSeconds** 속성은 정상 종료 기간(기본값 30초)을 제어하고 필요에 따라 애플리케이션별로 사용자 지정할 수 있습니다.

4.4.4. Blue-Green 배포

Blue-Green 배포에는 두 가지 버전의 애플리케이션을 동시에 실행하고 프로덕션 내 버전(Blue 버전)에서 최신 버전(Green 버전)으로 트래픽을 이동하는 작업이 포함됩니다. 롤링 전략 또는 전환 서비스를 경로에서 사용할 수 있습니다.

대다수의 애플리케이션이 영구 데이터에 의존하기 때문에 *N-1 호환성*을 지원하는 애플리케이션이 있어야 합니다. 그러면 데이터 계층 복사본을 두 개 생성하여 데이터를 공유하고 데이터베이스, 저장소 또는 디스크 간 실시간 마이그레이션을 구현할 수 있습니다.

새 버전을 테스트하는 데 사용되는 데이터를 떠올려 보십시오. 데이터가 프로덕션 데이터라면 새 버전의 버그로 인해 프로덕션 버전에 장애가 발생할 수 있습니다.

4.4.4.1. Blue-Green 배포 설정

Blue-Green 배포에서는 두 개의 **Deployment** 오브젝트를 사용합니다. 둘 다 실행되고 프로덕션의 항목은 경로에서 지정하는 서비스에 따라 달라지며 각 **Deployment** 오브젝트는 다른 서비스에 노출됩니다.

참고

경로는 웹(HTTP 및 HTTPS) 트래픽을 위한 것이므로 이 기술은 웹 애플리케이션에 적합합니다.

새 버전의 새 경로를 생성하고 테스트할 수 있습니다. 준비되었을 때 새 서비스를 가리키도록 프로덕션 경로의 서비스를 변경하면 새(Green) 버전이 활성화됩니다.

필요한 경우 서비스를 이전 버전(Blue)으로 전환하여 롤백할 수 있습니다.

프로세스

1. 독립된 애플리케이션 구성 요소 두 개를 생성합니다.
   a. **example-blue** 서비스에서 **v1** 이미지를 실행하는 예제 애플리케이션의 복사본을 생성합니다.

```
$ oc new-app openshift/deployment-example:v1 --name=example-blue
```

   b. **example-green** 서비스에서 **v2** 이미지를 사용하는 두 번째 복사본을 생성합니다.

```
$ oc new-app openshift/deployment-example:v2 --name=example-green
```

2. 이전 서비스를 가리키는 경로를 생성합니다.

```
$ oc expose svc/example-blue --name=bluegreen-example
```

3. **bluegreen-example-<project>.<router_domain>** 에서 애플리케이션을 검색하여 **v1** 이미지가 표시되는지 확인합니다.

4. 경로를 편집하고 서비스 이름을 **example-green**으로 변경합니다.

```
$ oc patch route/bluegreen-example -p '{"spec":{"to":{"name":"example-green"}}}'
```

5. 경로가 변경되었는지 확인하려면 **v2** 이미지가 표시될 때까지 브라우저를 새로 고칩니다.

4.4.5. A/B 배포

A/B 배포 전략을 사용하면 프로덕션 환경에서 제한된 방식으로 새 버전의 애플리케이션을 시도할 수 있습니다. 프로덕션 버전에서 대부분의 사용자 요청을 가져오고 제한된 요청만 새 버전으로 이동하도록 지정할 수 있습니다.

각 버전에 대한 요청 분배를 사용자가 제어하므로 테스트를 진행하면서 새 버전에 대한 요청 비율을 늘려 결국 이전 버전의 사용을 중지할 수 있습니다. 각 버전에 대한 요청 부하를 조정할 때 필요한 성능을 제공하기 위해 각 서비스의 Pod 수도 스케일링해야 할 수 있습니다.

소프트웨어 업그레이드 외에도 이 기능을 사용하여 사용자 인터페이스의 버전을 실험할 수 있습니다. 일부 사용자는 이전 버전을 보유하고 있고 일부는 새 버전을 보유하고 있기 때문에 다양한 버전에 대한 사용자의 반응을 평가하여 설계 결정을 알릴 수 있습니다.

이 배포가 효과적이려면 이전 버전과 새 버전이 동시에 실행될 수 있을 정도로 충분히 유사해야 합니다. 버그 수정 릴리즈 및 새 기능이 이전 기능을 방해하지 않는 경우 일반적으로 사용됩니다. 버전이 제대로 작동하려면 N-1 호환성이 필요합니다.

OpenShift Container Platform은 웹 콘솔과 CLI를 통해 N-1 호환성을 지원합니다.

4.4.5.1. A/B 테스트의 부하 분산

사용자는 다양한 서비스를 통해 경로를 설정합니다. 각 서비스는 애플리케이션의 버전을 처리합니다.

각 서비스에는 **weight**가 할당되고 각 서비스에 대한 일부 요청에는 **service_weight**을 **sum_of_weights**으로 나눈 값이 할당됩니다. 각 서비스의 **weight**는 서비스 끝점에 분배되므로 끝점 가중치의 합계는 서비스 **weight**입니다.

이 경로는 최대 4개의 서비스를 제공할 수 있습니다. 서비스 **weight**는 **0~256** 사이일 수 있습니다. **weight**가 **0**인 서비스는 부하 분산에 참여하지 않지만 기존 영구 연결을 계속 제공합니다. 서비스 **weight**가 **0**이 아닌 경우 각 끝점의 최소 **weight**는 **1**입니다. 이로 인해 끝점이 많은 서비스는 **weight**가 의도한 것보다 높을 수 있습니다. 이 경우 Pod 수를 줄이면 예상되는 부하 분산 **weight**를 얻을 수 있습니다.

프로세스

A/B 환경을 설정하려면 다음을 수행합니다.

1. 애플리케이션 두 개를 생성하고 서로 다른 이름을 지정합니다. 각각 **Deployment** 오브젝트를 생성합니다. 두 애플리케이션은 동일한 프로그램의 버전에 해당합니다. 하나는 일반적으로 현재 프로덕션 버전이며 다른 하나는 제안된 새 버전입니다.

   a. 첫 번째 애플리케이션을 생성합니다. 다음 예제에서는 **ab-example-a**라는 애플리케이션을 생성합니다.

```
$ oc new-app openshift/deployment-example --name=ab-example-a
```

b. 두 번째 애플리케이션을 생성합니다.

```
$ oc new-app openshift/deployment-example:v2 --name=ab-example-b
```

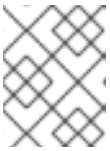
두 애플리케이션 모두 배포되고 서비스가 생성됩니다.

2. 경로를 통해 외부에서 애플리케이션을 사용할 수 있도록 설정합니다. 이 시점에서는 둘 중 하나를 노출할 수 있습니다. 현재 프로덕션 버전을 먼저 노출하고 나중에 경로를 수정하여 새 버전을 추가하는 것이 편리합니다.

```
$ oc expose svc/ab-example-a
```

ab-example-a.<project>.<router_domain>에서 애플리케이션을 검색하여 예상 버전이 표시되는지 확인합니다.

3. 경로를 배포할 때 라우터는 서비스에 지정된 **weights**에 따라 트래픽을 분산합니다. 이 시점에는 기본값이 **weight=1**인 단일 서비스가 있으므로 모든 요청이 해당 서비스로 이동합니다. 기타 서비스를 **alternateBackends**로 추가하고 **weights**를 조정하면 A/B 설정이 활성화됩니다. 이 작업은 **oc set route-backends** 명령을 실행하거나 경로를 편집하여 수행할 수 있습니다.
oc set route-backend를 **0**으로 설정하면 서비스가 부하 분산에 참여하지 않지만 기존 영구 연결은 계속 제공합니다.



참고

경로를 변경하면 다양한 서비스에 대한 트래픽 부분만 변경됩니다. 예상되는 부하를 처리하기 위해 Pod 수를 조정하려면 배포를 스케일링해야 할 수 있습니다.

경로를 편집하려면 다음을 실행합니다.

```
$ oc edit route <route_name>
```

출력 예

```
...
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin
spec:
  host: ab-example.my-project.my-domain
  to:
    kind: Service
    name: ab-example-a
    weight: 10
  alternateBackends:
  - kind: Service
    name: ab-example-b
    weight: 15
...
```

4.4.5.1.1. 웹 콘솔을 사용하여 기존 경로의 가중치 관리

프로세스

1. 네트워킹 → 경로 페이지로 이동합니다.
2. 편집할 경로 옆에 있는 작업 메뉴  를 클릭하고 경로 편집을 선택합니다.
3. YAML 파일을 편집합니다. 다른 대상 참조 오브젝트에 대한 대상의 상대적 가중치를 지정하는 **weight** 값을 **0**에서 **256** 사이의 정수가 되도록 업데이트합니다. 값 **0**은 이 백엔드에 대한 요청을 비활성화합니다. 기본값은 **100**입니다. 옵션에 대한 자세한 내용을 보려면 **oc explain routes.spec.alternateBackends**를 실행합니다.
4. 저장을 클릭합니다.

4.4.5.1.2. 웹 콘솔을 사용하여 새 경로의 가중치 관리

1. 네트워킹 → 경로 페이지로 이동합니다.
2. 경로 생성을 클릭합니다.
3. 경로 이름을 입력합니다.
4. 서비스를 선택합니다.
5. 대체 서비스 추가를 클릭합니다.
6. 가중치 및 대체 서비스 가중치 값을 입력합니다. 다른 대상과 비교했을 때 상대적 가중치를 나타내는 **0**에서 **255** 사이의 숫자를 입력합니다. 기본값은 **100**입니다.
7. 대상 포트를 선택합니다.
8. **Create**를 클릭합니다.

4.4.5.1.3. CLI를 사용하여 가중치 관리

프로세스

1. 경로에서 부하를 분산하는 서비스와 해당 가중치를 관리하려면 **oc set route-backends** 명령을 사용합니다.

```
$ oc set route-backends ROUTENAME \
    [--zero|--equal] [--adjust] SERVICE=WEIGHT[%] [...] [options]
```

예를 들어 다음 명령은 **ab-example-a**를 **weight=198**인 기본 서비스로 설정하고 **ab-example-b**는 **weight=2**인 첫 번째 대체 서비스로 설정합니다.

```
$ oc set route-backends ab-example ab-example-a=198 ab-example-b=2
```

즉 99%의 트래픽은 서비스 **ab-example-a**로 전송되고 1%는 서비스 **ab-example-b**로 전송됩니다.

이 명령에서는 배포를 스케일링하지 않습니다. 요청 부하를 처리할 수 있는 Pod를 충분히 확보하기 위해서는 스케일링해야 할 수 있습니다.

2. 플래그 없이 명령을 실행하여 현재 구성을 확인합니다.

```
$ oc set route-backends ab-example
```

출력 예

```
NAME                KIND     TO          WEIGHT
routes/ab-example   Service  ab-example-a 198 (99%)
routes/ab-example   Service  ab-example-b 2   (1%)
```

3. 자체 또는 기본 서비스에 대한 개별 서비스의 가중치를 변경하려면 **--adjust** 플래그를 사용합니다. 백분율을 지정하면 기본 또는 첫 번째 대체(기본을 지정하는 경우)를 기준으로 서비스가 조정됩니다. 다른 백엔드가 있는 경우 해당 가중치는 변경된 값에 비례하여 유지됩니다.
다음 예제에서는 **ab-example-a** 및 **ab-example-b** 서비스의 가중치를 변경합니다.

```
$ oc set route-backends ab-example --adjust ab-example-a=200 ab-example-b=10
```

또는 백분율을 지정하여 서비스 가중치를 변경합니다.

```
$ oc set route-backends ab-example --adjust ab-example-b=5%
```

백분율 선언 앞에 **+**를 지정하면 현재 설정을 기준으로 가중치를 조정할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc set route-backends ab-example --adjust ab-example-b=+15%
```

--equal 플래그는 모든 서비스의 **weight**를 **100**으로 설정합니다.

```
$ oc set route-backends ab-example --equal
```

--zero 플래그는 모든 서비스의 **weight**를 **0**으로 설정합니다. 그러면 모든 요청에서 503 오류가 반환됩니다.

참고

일부 라우터에서는 다중 또는 가중치 적용 백엔드를 지원하지 않습니다.

4.4.5.1.4. 하나의 서비스, 여러 Deployment 오브젝트

절차

1. 새 애플리케이션을 생성하여 모든 shard에 공통된 라벨 **ab-example=true**를 추가합니다.

```
$ oc new-app openshift/deployment-example --name=ab-example-a --as-deployment-
config=true --labels=ab-example=true --env=SUBTITLE\=shardA
$ oc delete svc/ab-example-a
```

애플리케이션이 배포되고 서비스가 생성됩니다. 이것이 첫 번째 shard입니다.

2. 경로를 통해 애플리케이션을 사용 가능하게 하거나 서비스 IP를 직접 사용합니다.

```
$ oc expose deployment ab-example-a --name=ab-example --selector=ab-example\=true
$ oc expose service ab-example
```

3. **ab-example-<project_name>.<router_domain>**에서 애플리케이션을 검색하여 **v1** 이미지가 표시되는지 확인합니다.

4. 첫 번째 shard와 동일한 소스 이미지 및 라벨을 기반으로 하지만 태그 버전이 다르고 환경 변수가 고유한 두 번째 shard를 생성합니다.

```
$ oc new-app openshift/deployment-example:v2 \
    --name=ab-example-b --labels=ab-example=true \
    SUBTITLE="shard B" COLOR="red" --as-deployment-config=true
$ oc delete svc/ab-example-b
```

5. 이 시점에 두 Pod 세트 모두 경로에 제공됩니다. 그러나 두 브라우저(연결을 열린 상태로 유지하여) 및 라우터(기본적으로 쿠키를 통해)에서 백엔드 서버에 대한 연결을 유지하려고 하기 때문에 두 shard가 반환되지 않을 수 있습니다.
브라우저를 하나 또는 다른 shard에 강제 적용하려면 다음을 수행합니다.

   a. **oc scale** 명령을 사용하여 **ab-example-a**의 복제본 수를 **0**으로 줄입니다.

```
$ oc scale dc/ab-example-a --replicas=0
```

      브라우저를 새로고침하여 **v2** 및 **shard B**(빨간색)를 표시합니다.

   b. **ab-example-a**를 복제본 수 **1**로, **ab-example-b**를 **0**으로 스케일링합니다.

```
$ oc scale dc/ab-example-a --replicas=1; oc scale dc/ab-example-b --replicas=0
```

      브라우저를 새로고침하여 **v1** 및 **shard A**(파란색)를 표시합니다.

6. 둘 중 하나의 shard에서 배포를 트리거하면 해당 shard의 Pod만 영향을 받습니다. **Deployment** 오브젝트에서 **SUBTITLE** 환경 변수를 변경하여 배포를 트리거할 수 있습니다.

```
$ oc edit dc/ab-example-a
```

   또는

```
$ oc edit dc/ab-example-b
```

5장. 할당량

5.1. 프로젝트당 리소스 할당량

ResourceQuota 오브젝트로 정의하는 *리소스 할당량*은 프로젝트당 집계 리소스 사용을 제한하는 제약 조건을 제공합니다. 이를 통해 프로젝트에서 생성할 수 있는 오브젝트의 수량을 유형별로 제한하고 해당 프로젝트의 리소스에서 사용할 수 있는 컴퓨팅 리소스 및 스토리지의 총량도 제한할 수 있습니다.

이 가이드에서는 리소스 할당량이 작동하는 방식, 클러스터 관리자가 프로젝트별로 리소스 할당량을 설정하고 관리하는 방법, 개발자와 클러스터 관리자가 이를 확인하는 방법을 설명합니다.

5.1.1. 할당량으로 관리하는 리소스

다음 내용에서는 할당량으로 관리할 수 있는 컴퓨팅 리소스 및 오브젝트 유형 세트를 설명합니다.

참고

status.phase in (Failed, Succeeded)이 True인 경우 Pod는 터미널 상태에 있습니다.

표 5.1. 할당량으로 관리하는 컴퓨팅 리소스

리소스 이름	설명
cpu	터미널이 아닌 상태에서 모든 Pod의 CPU 요청 합계는 이 값을 초과할 수 없습니다. **cpu** 및 **requests.cpu** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
memory	터미널이 아닌 상태에서 모든 Pod의 메모리 요청 합계는 이 값을 초과할 수 없습니다. **memory** 및 **requests.memory** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
ephemeral-storage	터미널이 아닌 상태에서 모든 Pod의 로컬 임시 스토리지 요청 합계는 이 값을 초과할 수 없습니다. **ephemeral-storage** 및 **requests.ephemeral-storage** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
requests.cpu	터미널이 아닌 상태에서 모든 Pod의 CPU 요청 합계는 이 값을 초과할 수 없습니다. **cpu** 및 **requests.cpu** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
requests.memory	터미널이 아닌 상태에서 모든 Pod의 메모리 요청 합계는 이 값을 초과할 수 없습니다. **memory** 및 **requests.memory** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
requests.ephemeral-storage	터미널이 아닌 상태에서 모든 Pod의 임시 스토리지 요청 합계는 이 값을 초과할 수 없습니다. **ephemeral-storage** 및 **requests.ephemeral-storage** 는 동일한 값이며 서로 바꿔 사용할 수 있습니다.
limits.cpu	터미널이 아닌 상태에서 모든 Pod의 CPU 제한 합계는 이 값을 초과할 수 없습니다.
limits.memory	터미널이 아닌 상태에서 모든 Pod의 메모리 제한 합계는 이 값을 초과할 수 없습니다.

리소스 이름	설명
limits.ephemeral-storage	터미널이 아닌 상태에서 모든 Pod의 임시 스토리지 제한 합계는 이 값을 초과할 수 없습니다.

표 5.2. 할당량으로 관리되는 스토리지 리소스

리소스 이름	설명
requests.storage	상태와 관계없이 모든 영구 볼륨 클레임의 스토리지 요청 합계는 이 값을 초과할 수 없습니다.
persistentvolumeclaims	프로젝트에 존재할 수 있는 총 영구 볼륨 클레임 수입니다.
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	상태와 관계없이 일치하는 스토리지 클래스가 있는 모든 영구 볼륨 클레임의 스토리지 요청 합계는 이 값을 초과할 수 없습니다.
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	프로젝트에 존재할 수 있는, 일치하는 스토리지 클래스가 있는 총 영구 볼륨 클레임 수입니다.

표 5.3. 할당량으로 관리하는 오브젝트 수

리소스 이름	설명
pods	프로젝트에 존재할 수 있는 터미널이 아닌 상태의 총 Pod 수입니다.
replicationcontrollers	프로젝트에 존재할 수 있는 총 복제 컨트롤러 수입니다.
resourcequotas	프로젝트에 존재할 수 있는 총 리소스 할당량 수입니다.
services	프로젝트에 존재할 수 있는 총 서비스 수입니다.
services.loadbalancers	프로젝트에 존재할 수 있는 **LoadBalancer** 유형의 총 서비스 수입니다.
services.nodeports	프로젝트에 존재할 수 있는 **NodePort** 유형의 총 서비스 수입니다.
secrets	프로젝트에 존재할 수 있는 총 시크릿 수입니다.
configmaps	프로젝트에 존재할 수 있는 총 **ConfigMap** 오브젝트 수입니다.

리소스 이름	설명
persistentvolumeclaims	프로젝트에 존재할 수 있는 총 영구 볼륨 클레임 수입니다.
openshift.io/imagestreams	프로젝트에 존재할 수 있는 총 이미지 스트림 수입니다.

5.1.2. 할당량 범위

각 할당량에는 일련의 관련 *범위*가 있을 수 있습니다. 특정 할당량은 열거된 범위의 교집합과 일치하는 경우에만 리소스 사용량을 측정합니다.

할당량에 범위를 추가하면 할당량을 적용할 수 있는 리소스 세트가 제한됩니다. 허용된 설정을 벗어난 리소스를 지정하면 검증 오류가 발생합니다.

범위	설명
Terminating	**spec.activeDeadlineSeconds©= 0**인 Pod와 일치합니다.
NotTerminating	**spec.activeDeadlineSeconds**가 **nil**인 Pod와 일치합니다.
BestEffort	**cpu** 또는 **memory**에 대해 최상의 작업 품질을 제공하는 Pod와 일치합니다.
NotBestEffort	**cpu** 및 **memory**에 최상의 작업 품질을 제공하지 않는 Pod와 일치합니다.

BestEffort 범위는 할당량을 제한하여 다음 리소스를 제한합니다.

- **pods**

Terminating, **NotTerminating**, **NotBestEffort** 범위는 할당량을 제한하여 다음 리소스를 추적합니다.

- **pods**
- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**
- **limits.cpu**
- **ephemeral-storage**

- **requests.ephemeral-storage**
- **limits.ephemeral-storage**

5.1.3. 할당량 적용

프로젝트에 대한 리소스 할당량이 처음 생성되면 프로젝트에서 업데이트된 사용량 통계를 계산할 때까지 할당량 제약 조건을 위반할 수 있는 새 리소스 생성 기능을 제한합니다.

할당량이 생성되고 사용량 통계가 업데이트되면 프로젝트에서 새 콘텐츠 생성을 허용합니다. 리소스를 생성하거나 수정할 때는 리소스 생성 또는 수정 요청에 따라 할당량 사용이 즉시 증가합니다.

리소스를 삭제할 때는 프로젝트에 대한 다음 할당량 통계 전체 재계산 중 할당량 사용이 감소합니다. 구성 가능한 시간에 따라 현재 관찰되는 시스템 값으로 할당량 사용을 줄이는 데 걸리는 시간이 결정됩니다.

프로젝트 수정이 할당량 사용 제한을 초과하면 서버에서 작업을 거부하고 사용자에게 할당량 제약 조건 위반에 대해 설명하는 적절한 오류 메시지와 현재 시스템에서 관찰되는 사용 통계를 반환합니다.

5.1.4. 요청과 제한 비교

컴퓨팅 리소스를 할당할 때 각 컨테이너에서 CPU, 메모리, 임시 스토리지 각각에 대한 요청 및 제한 값을 지정할 수 있습니다. 할당량은 이러한 값 중을 제한할 수 있습니다.

할당량에 **requests.cpu** 또는 **requests.memory**에 대해 지정된 값이 있는 경우 들어오는 모든 컨테이너에서 해당 리소스를 명시적으로 요청해야 합니다. 할당량에 **limits.cpu** 또는 **limits.memory**에 대해 지정된 값이 있는 경우 들어오는 모든 컨테이너에서 해당 리소스에 대한 제한을 명시적으로 지정해야 합니다.

5.1.5. 리소스 할당량 정의의 예

core-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" 1
    persistentvolumeclaims: "4" 2
    replicationcontrollers: "20" 3
    secrets: "10" 4
    services: "10" 5
    services.loadbalancers: "2" 6
```

1. 프로젝트에 존재할 수 있는 총 **ConfigMap** 오브젝트 수입니다.
2. 프로젝트에 존재할 수 있는 총 PVC(영구 볼륨 클레임) 수입니다.
3. 프로젝트에 존재할 수 있는 총 복제 컨트롤러 수입니다.
4. 프로젝트에 존재할 수 있는 총 시크릿 수입니다.
5. 프로젝트에 존재할 수 있는 총 서비스 수입니다.

6 프로젝트에 존재할 수 있는 **LoadBalancer** 유형의 총 서비스 수입니다.

openshift-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" 1
```

1 프로젝트에 존재할 수 있는 총 이미지 스트림 수입니다.

compute-resources.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4" 1
    requests.cpu: "1" 2
    requests.memory: 1Gi 3
    requests.ephemeral-storage: 2Gi 4
    limits.cpu: "2" 5
    limits.memory: 2Gi 6
    limits.ephemeral-storage: 4Gi 7
```

1 프로젝트에 존재할 수 있는 터미널이 아닌 상태의 총 Pod 수입니다.

2 터미널이 아닌 상태에서 모든 Pod의 CPU 요청 합계는 코어 1개를 초과할 수 없습니다.

3 터미널이 아닌 상태에서 모든 Pod의 메모리 요청 합계는 1Gi를 초과할 수 없습니다.

4 터미널이 아닌 상태에서 모든 Pod의 임시 스토리지 요청 합계는 2Gi를 초과할 수 없습니다.

5 터미널이 아닌 상태에서 모든 Pod의 CPU 제한 합계는 코어 2개를 초과할 수 없습니다.

6 터미널이 아닌 상태에서 모든 Pod의 메모리 제한 합계는 2Gi를 초과할 수 없습니다.

7 터미널이 아닌 상태에서 모든 Pod의 임시 스토리지 제한 합계는 4Gi를 초과할 수 없습니다.

besteffort.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
```

```
  hard:
    pods: "1" (1)
  scopes:
  - BestEffort (2)
```

(1) 상태가 터미널이 아니고 서비스 품질이 **BestEffort**인, 프로젝트에 존재할 수 있는 총 Pod 수입니다.

(2) 메모리 또는 CPU에 대한 서비스 품질이 **BestEffort**와 일치하는 Pod로만 할당량을 제한합니다.

compute-resources-long-running.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" (1)
    limits.cpu: "4" (2)
    limits.memory: "2Gi" (3)
    limits.ephemeral-storage: "4Gi" (4)
  scopes:
  - NotTerminating (5)
```

(1) 터미널이 아닌 상태의 총 Pod 수입니다.

(2) 터미널이 아닌 상태에서 모든 Pod의 CPU 제한 합계는 이 값을 초과할 수 없습니다.

(3) 터미널이 아닌 상태에서 모든 Pod의 메모리 제한 합계는 이 값을 초과할 수 없습니다.

(4) 터미널이 아닌 상태에서 모든 Pod의 임시 스토리지 제한 합계는 이 값을 초과할 수 없습니다.

(5) 할당량을 **spec.activeDeadlineSeconds**가 **nil**로 설정된 일치하는 Pod로만 제한합니다. 빌드 Pod는 **RestartNever** 정책을 적용하지 않는 한 **NotTerminating**에 해당합니다.

compute-resources-time-bound.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" (1)
    limits.cpu: "1" (2)
    limits.memory: "1Gi" (3)
    limits.ephemeral-storage: "1Gi" (4)
  scopes:
  - Terminating (5)
```

(1) 종료 상태의 총 Pod 수입니다.

2 종료 상태의 모든 Pod에서 CPU 제한 합계는 이 값을 초과할 수 없습니다.

3 종료 상태의 모든 Pod에서 메모리 제한 합계는 이 값을 초과할 수 없습니다.

4 종료 상태의 모든 Pod에서 임시 스토리지 제한 합계는 이 값을 초과할 수 없습니다.

5 할당량을 **spec.activeDeadlineSeconds >=0**인 일치하는 Pod로만 제한합니다. 예를 들어 이 할당량은 빌드 또는 배포자 Pod에 대해 부과되지만 웹 서버 또는 데이터베이스와 같이 오래 실행되는 Pod는 아닙니다.

storage-consumption.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" 1
    requests.storage: "50Gi" 2
    gold.storageclass.storage.k8s.io/requests.storage: "10Gi" 3
    silver.storageclass.storage.k8s.io/requests.storage: "20Gi" 4
    silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" 5
    bronze.storageclass.storage.k8s.io/requests.storage: "0" 6
    bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" 7
```

1 프로젝트의 총 영구 볼륨 클레임 수

2 프로젝트의 모든 영구 볼륨 클레임에서 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.

3 프로젝트의 모든 영구 볼륨 클레임에서 골드 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.

4 프로젝트의 모든 영구 볼륨 클레임에서 실버 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.

5 프로젝트의 모든 영구 볼륨 클레임에서 실버 스토리지 클래스의 총 클레임 수는 이 값을 초과할 수 없습니다.

6 프로젝트의 모든 영구 볼륨 클레임에서 브론즈 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다. 이 값을 **0**으로 설정하면 브론즈 스토리지 클래스에서 스토리지를 요청할 수 없습니다.

7 프로젝트의 모든 영구 볼륨 클레임에서 브론즈 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다. 이 값을 **0**으로 설정하면 브론즈 스토리지 클래스에서 클레임을 생성할 수 없습니다.

5.1.6. 할당량 생성

할당량을 생성하여 지정된 프로젝트에서 리소스 사용량을 제한할 수 있습니다.

프로세스

1. 파일에 할당량을 정의합니다.
2. 이 파일을 사용하여 할당량을 생성하고 프로젝트에 적용합니다.

```
$ oc create -f <file> [-n <project_name>]
```

예를 들면 다음과 같습니다.

```
$ oc create -f core-object-counts.yaml -n demoproject
```

5.1.6.1. 오브젝트 수 할당량 생성

모든 OpenShift Container Platform 표준 네임스페이스 리소스 유형(예: **BuildConfig**, **DeploymentConfig** 개체)에 대해 오브젝트 수 할당량을 생성할 수 있습니다. 오브젝트 할당량 수는 모든 표준 네임스페이스 리소스 유형에 정의된 할당량을 지정합니다.

리소스 할당량을 사용할 때 오브젝트는 생성 시 할당량에 대해 부과됩니다. 이러한 유형의 할당량은 스토리지 소진을 방지하는 데 유용합니다. 할당량은 프로젝트 내에 충분한 예비 리소스가 있는 경우에만 생성할 수 있습니다.

프로세스

리소스에 대한 오브젝트 수 할당량을 구성하려면 다음을 수행합니다.

1. 다음 명령을 실행합니다.

```
$ oc create quota <name> \
    --hard=count/<resource>.<group>=<quota>,count/<resource>.<group>=<quota> 1
```

1 **<resource>** 변수는 리소스 이름이고 **<group>**은 API 그룹입니다(해당하는 경우). 리소스 및 관련 API 그룹 목록에 **oc api-resources** 명령을 사용합니다.

예를 들면 다음과 같습니다.

```
$ oc create quota test \
    --
hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secr
ets=4
```

출력 예

```
resourcequota "test" created
```

이 예제에서는 나열된 리소스가 클러스터에 있는 각 프로젝트의 하드 제한으로 제한됩니다.

2. 할당량이 생성되었는지 확인합니다.

```
$ oc describe quota test
```

출력 예

```
Name:                   test
```

```
Namespace:                    quota
Resource                     Used  Hard
--------                     ----  ----
count/deployments.extensions  0     2
count/pods                    0     3
count/replicasets.extensions  0     4
count/secrets                 0     4
```

5.1.6.2. 확장 리소스에 대한 리소스 할당량 설정

확장 리소스에는 리소스 과다 할당이 허용되지 않으므로 할당량의 해당 확장 리소스에 **requests** 및 **limits**를 지정해야 합니다. 현재는 확장 리소스에 접두사 **requests.**가 있는 할당량 항목만 허용됩니다. 다음은 GPU 리소스 **nvidia.com/gpu**에 대한 리소스 할당량을 설정하는 방법에 대한 예제 시나리오입니다.

프로세스

1. 클러스터의 노드에서 사용 가능한 GPU 수를 결정합니다. 예를 들면 다음과 같습니다.

```
# oc describe node ip-172-31-27-209.us-west-2.compute.internal | egrep
'Capacity|Allocatable|gpu'
```

출력 예

```
                    openshift.com/gpu-accelerator=true
Capacity:
 nvidia.com/gpu:  2
Allocatable:
 nvidia.com/gpu:  2
  nvidia.com/gpu  0          0
```

이 예에서는 GPU 2개를 사용할 수 있습니다.

2. 네임스페이스 **nvidia**에 할당량을 설정합니다. 이 예에서 할당량은 **1**입니다.

```
# cat gpu-quota.yaml
```

출력 예

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: gpu-quota
  namespace: nvidia
spec:
  hard:
    requests.nvidia.com/gpu: 1
```

3. 할당량을 생성합니다.

```
# oc create -f gpu-quota.yaml
```

출력 예

```
resourcequota/gpu-quota created
```

4. 네임스페이스에 올바른 할당량이 설정되어 있는지 확인합니다.

```
# oc describe quota gpu-quota -n nvidia
```

출력 예

```
Name:                   gpu-quota
Namespace:              nvidia
Resource                Used  Hard
--------                ----  ----
requests.nvidia.com/gpu  0     1
```

5. 단일 GPU를 요청하는 Pod를 정의합니다. 다음 예제 정의 파일은 **gpu-pod.yaml**이라고 합니다.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: gpu-pod-
  namespace: nvidia
spec:
  restartPolicy: OnFailure
  containers:
  - name: rhel7-gpu-pod
    image: rhel7
    env:
      - name: NVIDIA_VISIBLE_DEVICES
        value: all
      - name: NVIDIA_DRIVER_CAPABILITIES
        value: "compute,utility"
      - name: NVIDIA_REQUIRE_CUDA
        value: "cuda>=5.0"
    command: ["sleep"]
    args: ["infinity"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

6. Pod를 생성합니다.

```
# oc create -f gpu-pod.yaml
```

7. Pod가 실행 중인지 확인합니다.

```
# oc get pods
```

출력 예

```
NAME            READY    STATUS     RESTARTS   AGE
gpu-pod-s46h7   1/1      Running    0          1m
```

8. 할당량 **Used**의 카운터가 올바른지 확인합니다.

```
# oc describe quota gpu-quota -n nvidia
```

출력 예

```
Name:                   gpu-quota
Namespace:              nvidia
Resource                Used  Hard
--------                ----  ----
requests.nvidia.com/gpu  1     1
```

9. **nvidia** 네임스페이스에서 두 번째 GPU Pod를 생성합니다. 노드에 GPU가 2개 있으므로 기술적으로 가능합니다.

```
# oc create -f gpu-pod.yaml
```

출력 예

```
Error from server (Forbidden): error when creating "gpu-pod.yaml": pods "gpu-pod-f7z2w" is forbidden: exceeded quota: gpu-quota, requested: requests.nvidia.com/gpu=1, used: requests.nvidia.com/gpu=1, limited: requests.nvidia.com/gpu=1
```

이 허용되지 않음 오류 메시지는 할당량이 GPU 1개이고 이 Pod에서 할당량을 초과하는 두 번째 GPU를 할당하려고 했기 때문에 예상된 것입니다.

5.1.7. 할당량 보기

웹 콘솔에서 프로젝트의 할당량 페이지로 이동하면 프로젝트 할당량에 정의된 모든 하드 제한과 관련된 사용량 통계를 볼 수 있습니다.

CLI를 사용하여 할당량 세부 정보를 볼 수도 있습니다.

프로세스

1. 프로젝트에 정의된 할당량 목록을 가져옵니다. 예를 들어 **demoproject**라는 프로젝트의 경우 다음과 같습니다.

```
$ oc get quota -n demoproject
```

출력 예

```
NAME                AGE
besteffort          11m
compute-resources   2m
core-object-counts  29m
```

2. 관심 있는 할당량을 입력합니다. 예를 들어 **core-object-counts** 할당량은 다음과 같습니다.

```
$ oc describe quota core-object-counts -n demoproject
```

출력 예

```
Name:   core-object-counts
Namespace:  demoproject
Resource  Used Hard
--------  ---- ----
configmaps  3 10
persistentvolumeclaims 0 4
replicationcontrollers 3 20
secrets   9 10
services  2 10
```

5.1.8. 명시적 리소스 할당량 구성

새 프로젝트에 특정 리소스 할당량을 적용하려면 프로젝트 요청 템플릿에서 명시적 리소스 할당량을 구성합니다.

사전 요구 사항

- cluster-admin 역할을 가진 사용자로 클러스터에 액세스합니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. 프로젝트 요청 템플릿에 리소스 할당량 정의를 추가합니다.
   - 클러스터에 프로젝트 요청 템플릿이 없는 경우 다음을 수행합니다.
     a. 부트스트랩 프로젝트 템플릿을 생성하고 **template.yaml**이라는 파일에 출력합니다.

     ```
     $ oc adm create-bootstrap-project-template -o yaml > template.yaml
     ```

     b. **template.yaml**에 리소스 할당량 정의를 추가합니다. 다음 예제에서는 'storage-consumption'이라는 리소스 할당량을 정의합니다. 정의는 템플릿의 **parameters:** 섹션 앞에 추가해야 합니다.

     ```
     - apiVersion: v1
       kind: ResourceQuota
       metadata:
         name: storage-consumption
         namespace: ${PROJECT_NAME}
       spec:
         hard:
           persistentvolumeclaims: "10" (1)
           requests.storage: "50Gi" (2)
           gold.storageclass.storage.k8s.io/requests.storage: "10Gi" (3)
           silver.storageclass.storage.k8s.io/requests.storage: "20Gi" (4)
           silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" (5)
           bronze.storageclass.storage.k8s.io/requests.storage: "0" (6)
           bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" (7)
     ```

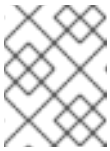
     (1) 프로젝트의 총 영구 볼륨 클레임 수입니다.

     (2) 프로젝트의 모든 영구 볼륨 클레임에서 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.

3. 프로젝트의 모든 영구 볼륨 클레임에서 골드 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.
4. 프로젝트의 모든 영구 볼륨 클레임에서 실버 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다.
5. 프로젝트의 모든 영구 볼륨 클레임에서 실버 스토리지 클래스의 총 클레임 수는 이 값을 초과할 수 없습니다.
6. 프로젝트의 모든 영구 볼륨 클레임에서 브론즈 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다. 이 값을 **0**으로 설정하면 브론즈 스토리지 클래스에서 스토리지를 요청할 수 없습니다.
7. 프로젝트의 모든 영구 볼륨 클레임에서 브론즈 스토리지 클래스에 요청된 스토리지 합계는 이 값을 초과할 수 없습니다. 이 값을 **0**으로 설정하면 브론즈 스토리지 클래스에서 클레임을 생성할 수 없습니다.

c. **openshift-config** 네임스페이스의 수정된 **template.yaml** 파일에서 프로젝트 요청 템플릿을 생성합니다.

```
$ oc create -f template.yaml -n openshift-config
```

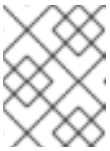


참고

구성을 **kubectl.kubernetes.io/last-applied-configuration** 주석으로 포함하려면 **oc create** 명령에 **--save-config** 옵션을 추가합니다.

기본적으로 이 템플릿을 **project-request**라고 합니다.

- 클러스터에 프로젝트 요청 템플릿이 이미 있는 경우 다음을 수행합니다.



참고

구성 파일을 사용하여 클러스터 내의 오브젝트를 선언적 또는 명령적으로 관리하는 경우 대신 해당 파일을 통해 기존 프로젝트 요청 템플릿을 편집합니다.

a. **openshift-config** 네임스페이스의 템플릿을 나열합니다.

```
$ oc get templates -n openshift-config
```

b. 기존 프로젝트 요청 템플릿을 편집합니다.

```
$ oc edit template <project_request_template> -n openshift-config
```

c. 위의 **storage-consumption** 예제와 같이 기존 템플릿에 리소스 할당량 정의를 추가합니다. 정의는 템플릿의 **parameters:** 섹션 앞에 추가해야 합니다.

2. 프로젝트 요청 템플릿을 생성한 경우 클러스터의 프로젝트 구성 리소스에서 해당 템플릿을 참조합니다.

a. 편집할 프로젝트 구성 리소스에 액세스합니다.

- 웹 콘솔 사용:

i. 관리 → 클러스터 설정으로 이동합니다.

ii. 전역 구성을 클릭하여 모든 구성 리소스를 확인합니다.

iii. 프로젝트 항목을 찾아 **YAML** 편집을 클릭합니다.

- CLI 사용:

i. 다음과 같이 **project.config.openshift.io/cluster** 리소스를 편집합니다.

```
$ oc edit project.config.openshift.io/cluster
```

b. **projectRequestTemplate** 및 **name** 매개변수를 포함하도록 프로젝트 구성 리소스의 **spec** 섹션을 업데이트합니다. 다음 예제에서는 기본 프로젝트 요청 템플릿 이름 **project-request**를 참조합니다.

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: project-request
```

3. 프로젝트가 생성될 때 리소스 할당량이 적용되는지 확인합니다.

a. 프로젝트를 생성합니다.

```
$ oc new-project <project_name>
```

b. 프로젝트의 리소스 할당량을 나열합니다.

```
$ oc get resourcequotas
```

c. 리소스 할당량을 자세히 설명합니다.

```
$ oc describe resourcequotas <resource_quota_name>
```

5.2. 다중 프로젝트의 리소스 할당량

ClusterResourceQuota 오브젝트에서 정의하는 다중 프로젝트 할당량은 여러 프로젝트에서 할당량을 공유할 수 있습니다. 선택한 각 프로젝트에서 사용하는 리소스를 집계하고 이 집계 정보를 사용하여 선택한 모든 프로젝트의 리소스를 제한합니다.

이 가이드에서는 클러스터 관리자가 다중 프로젝트에서 리소스 할당량을 설정하고 관리하는 방법을 설명합니다.

5.2.1. 할당량 생성 중 다중 프로젝트 선택

할당량을 생성할 때 주석 선택이나 라벨 선택 또는 둘 다에 따라 다중 프로젝트를 선택할 수 있습니다.

프로세스

1. 주석에 따라 프로젝트를 선택하려면 다음 명령을 실행합니다.

```
$ oc create clusterquota for-user \
    --project-annotation-selector openshift.io/requester=<user_name> \
    --hard pods=10 \
    --hard secrets=20
```

이렇게 하면 다음과 같은 **ClusterResourceQuota** 오브젝트가 생성됩니다.

```
apiVersion: quota.openshift.io/v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: 1
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: 2
      openshift.io/requester: <user_name>
    labels: null 3
status:
  namespaces: 4
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: 5
    hard:
      pods: "10"
      secrets: "20"
    used:
      pods: "1"
      secrets: "9"
```

1. 선택한 프로젝트를 통해 적용할 **ResourceQuotaSpec** 오브젝트입니다.
2. 주석에 대한 간단한 키-값 선택기입니다.
3. 프로젝트를 선택하는 데 사용할 수 있는 라벨 선택기입니다.
4. 선택한 각 프로젝트의 현재 할당량 사용을 설명하는 네임스페이스별 맵입니다.
5. 선택한 모든 프로젝트에서 집계한 사용량입니다.

이 다중 프로젝트 할당량 문서는 기본 프로젝트 요청 끝점을 사용하여 **<user_name>**에서 요청하는 모든 프로젝트를 제어합니다. Pod 10개와 시크릿 20개로 제한되어 있습니다.

2. 마찬가지로 라벨에 따라 프로젝트를 선택하려면 다음 명령을 실행합니다.

```
-
```

```
$ oc create clusterresourcequota for-name \ 1
    --project-label-selector=name=frontend \ 2
    --hard=pods=10 --hard=secrets=20
```

1 **clusterresourcequota** 및 **clusterquota** 는 모두 동일한 명령의 별칭입니다. **for-name** 은 **ClusterResourceQuota** 오브젝트의 이름입니다.

2 라벨에 따라 프로젝트를 선택하려면 **--project-label-selector=key=value** 형식을 사용하여 키-값 쌍을 제공합니다.

이렇게 하면 다음과 같은 **ClusterResourceQuota** 오브젝트 정의가 생성됩니다.

```
apiVersion: quota.openshift.io/v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend
```

5.2.2. 적용 가능한 클러스터 리소스 할당량 보기

프로젝트 관리자는 자신의 프로젝트를 제한하는 다중 프로젝트 할당량을 생성하거나 수정할 수 없지만 관리자는 자신의 프로젝트에 적용되는 다중 프로젝트 할당량 문서를 볼 수 있습니다. 프로젝트 관리자는 **AppliedClusterResourceQuota** 리소스를 통해 이 작업을 수행할 수 있습니다.

프로세스

1. 프로젝트에 적용되는 할당량을 보려면 다음을 실행합니다.

```
$ oc describe AppliedClusterResourceQuota
```

출력 예

```
Name:   for-user
Namespace:  <none>
Created:  19 hours ago
Labels:   <none>
Annotations:  <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource  Used  Hard
--------  ----  ----
pods      1     10
secrets   9     20
```

5.2.3. 선택 단위

할당량 배분을 요청할 때의 잠금 고려 사항으로 인해 다중 프로젝트 할당량에 따라 선택되는 활성 프로젝트의 수는 중요한 고려 사항입니다. 단일 다중 프로젝트 할당량에서 프로젝트를 100개 이상 선택하면 해당 프로젝트의 API 서버 응답성에 불리한 영향을 미칠 수 있습니다.

6장. 애플리케이션과 함께 구성 맵 사용

구성 맵을 사용하면 컨테이너화된 애플리케이션을 이식할 수 있도록 구성 아티팩트를 이미지 콘텐츠에서 분리할 수 있습니다.

다음 섹션에서는 구성 맵과 이를 생성하고 사용하는 방법을 정의합니다.

구성 맵 생성에 대한 자세한 내용은 구성 맵 생성 및 사용을 참조하십시오.

6.1. 구성 맵 이해

많은 애플리케이션에서는 구성 파일, 명령줄 인수 및 환경 변수를 조합한 구성이 필요합니다. OpenShift Container Platform에서 컨테이너화된 애플리케이션을 이식하기 위해 이러한 구성 아티팩트는 이미지 콘텐츠와 분리됩니다.

ConfigMap 오브젝트는 컨테이너를 OpenShift Container Platform과 무관하게 유지하면서 구성 데이터를 사용하여 컨테이너를 삽입하는 메커니즘을 제공합니다. 구성 맵은 개별 속성 또는 전체 구성 파일 또는 JSON Blob과 같은 세분화된 정보를 저장하는 데 사용할 수 있습니다.

ConfigMap API 오브젝트에는 Pod에서 사용하거나 컨트롤러와 같은 시스템 구성 요소의 구성 데이터를 저장하는 데 사용할 수 있는 구성 데이터의 키-값 쌍이 있습니다. 예를 들면 다음과 같습니다.

ConfigMap 오브젝트 정의

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data: 1
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw 2
```

1 구성 데이터를 포함합니다.

2 UTF8이 아닌 데이터를 포함한 파일을 가리킵니다(예: 바이너리 Java 키 저장소 파일). Base 64에 파일 데이터를 입력합니다.

참고

이미지와 같은 바이너리 파일에서 구성 맵을 생성할 때 **binaryData** 필드를 사용할 수 있습니다.

다양한 방법으로 Pod에서 구성 데이터를 사용할 수 있습니다. 구성 맵을 다음과 같이 사용할 수 있습니다.

- 컨테이너에서 환경 변수 값 채우기

- 컨테이너에서 명령줄 인수 설정
- 볼륨에 구성 파일 채우기

사용자 및 시스템 구성 요소는 구성 데이터를 구성 맵에 저장할 수 있습니다.

구성 맵은 보안과 유사하지만 민감한 정보가 포함되지 않은 문자열 작업을 더 편리하게 지원하도록 설계되었습니다.

구성 맵 제한 사항
Pod에서 콘텐츠를 사용하기 전에 구성 맵을 생성해야 합니다.

컨트롤러는 누락된 구성 데이터를 허용하도록 작성할 수 있습니다. 상황에 따라 구성 맵을 사용하여 구성된 개별 구성 요소를 참조하십시오.

ConfigMap 오브젝트는 프로젝트에 있습니다.

동일한 프로젝트의 Pod에서만 참조할 수 있습니다.

Kubelet은 **API** 서버에서 가져오는 **Pod**에 대한 구성 맵만 지원합니다.

여기에는 CLI를 사용하거나 복제 컨트롤러에서 간접적으로 생성되는 모든 Pod가 포함됩니다. OpenShift Container Platform 노드의 **--manifest-url** 플래그, **--config** 플래그 또는 해당 REST API를 사용하여 생성한 Pod를 포함하지 않으며 이는 Pod를 생성하는 일반적인 방법이 아니기 때문입니다.

6.2. 사용 사례: POD에서 구성 맵 사용

다음 섹션에서는 Pod에서 **ConfigMap** 오브젝트를 사용할 때 몇 가지 사용 사례에 대해 설명합니다.

6.2.1. 구성 맵을 사용하여 컨테이너에서 환경 변수 채우기

구성 맵은 컨테이너의 개별 환경 변수를 채우거나 유효한 환경 변수 이름을 형성하는 모든 키에서 컨테이너에 있는 환경 변수를 채우는 데 사용할 수 있습니다.

예를 들어 다음 구성 맵을 고려하십시오.

두 개의 환경 변수가 있는 ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config 1
  namespace: default 2
data:
  special.how: very 3
  special.type: charm 4
```

1 구성 맵의 이름입니다.

2 구성 맵이 있는 프로젝트입니다. 구성 맵은 동일한 프로젝트의 Pod에서만 참조할 수 있습니다.

3 4 삽입할 환경 변수입니다.

하나의 환경 변수가 있는 ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config (1)
  namespace: default
data:
  log_level: INFO (2)
```

(1) 구성 맵의 이름입니다.

(2) 삽입할 환경 변수입니다.

절차

- **configMapKeyRef** 섹션을 사용하여 Pod에서 이 **ConfigMap**의 키를 사용할 수 있습니다.

  특정 환경 변수를 삽입하도록 구성된 샘플 Pod 사양

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env: (1)
        - name: SPECIAL_LEVEL_KEY (2)
          valueFrom:
            configMapKeyRef:
              name: special-config (3)
              key: special.how (4)
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config (5)
              key: special.type (6)
              optional: true (7)
      envFrom: (8)
        - configMapRef:
            name: env-config (9)
  restartPolicy: Never
```

(1) **ConfigMap**에서 지정된 환경 변수를 가져오는 스탠자입니다.

(2) 키 값을 삽입하는 Pod 환경 변수의 이름입니다.

(3) (5) 특정 환경 변수를 끌어올 **ConfigMap**의 이름입니다.

(4) (6) **ConfigMap**에서 가져올 환경 변수입니다.

7 환경 변수를 선택적으로 만듭니다. 선택적으로 지정된 **ConfigMap** 및 키가 없는 경우에도 Pod가 시작됩니다.

8 **ConfigMap**에서 모든 환경 변수를 가져오는 스탠자입니다.

9 모든 환경 변수를 가져올 **ConfigMap**의 이름입니다.

이 Pod가 실행되면 Pod 로그에 다음 출력이 포함됩니다.

```
SPECIAL_LEVEL_KEY=very
log_level=INFO
```

참고

SPECIAL_TYPE_KEY=charm은 예제 출력에 나열되지 않습니다. **optional: true**가 설정되어 있기 때문입니다.

6.2.2. 구성 맵을 사용하여 컨테이너 명령에 대한 명령줄 인수 설정

구성 맵을 사용하여 컨테이너에서 명령 또는 인수 값을 설정할 수도 있습니다. 이는 Kubernetes 대체 구문 **$(VAR_NAME)**을 사용하여 수행됩니다. 다음 구성 맵을 고려하십시오.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

절차

- 컨테이너의 명령에 값을 삽입하려면 환경 변수 사용 사례에서 ConfigMap을 사용하는 것처럼 환경 변수로 사용할 키를 사용해야 합니다. 그런 다음 **$(VAR_NAME)** 구문을 사용하여 컨테이너의 명령에서 참조할 수 있습니다.

  특정 환경 변수를 삽입하도록 구성된 샘플 Pod 사양

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
1
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
```

```
            name: special-config
            key: special.how
      - name: SPECIAL_TYPE_KEY
        valueFrom:
          configMapKeyRef:
            name: special-config
            key: special.type
  restartPolicy: Never
```

1 환경 변수로 사용할 키를 사용하여 컨테이너의 명령에 값을 삽입합니다.

이 Pod가 실행되면 test-container 컨테이너에서 실행되는 echo 명령의 출력은 다음과 같습니다.

```
very charm
```

6.2.3. 구성 맵을 사용하여 볼륨에 콘텐츠 삽입

구성 맵을 사용하여 볼륨에 콘텐츠를 삽입할 수 있습니다.

ConfigMap CR(사용자 정의 리소스)의 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

절차

구성 맵을 사용하여 볼륨에 콘텐츠를 삽입하는 몇 가지 다른 옵션이 있습니다.

- 구성 맵을 사용하여 콘텐츠를 볼륨에 삽입하는 가장 기본적인 방법은 키가 파일 이름이고 파일의 콘텐츠가 키의 값인 파일로 볼륨을 채우는 것입니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "cat", "/etc/config/special.how" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config 1
  restartPolicy: Never
```

```
-
```

1 키가 포함된 파일입니다.

이 Pod가 실행되면 cat 명령의 출력은 다음과 같습니다.

```
very
```

- 구성 맵 키가 프로젝션되는 볼륨 내 경로를 제어할 수도 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "cat", "/etc/config/path/to/special-key" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
        - key: special.how
          path: path/to/special-key 1
  restartPolicy: Never
```

1 구성 맵 키의 경로입니다.

이 Pod가 실행되면 cat 명령의 출력은 다음과 같습니다.

```
very
```

7장. 개발자 화면을 사용하여 프로젝트 및 애플리케이션 지표 모니터링

개발자 화면의 모니터링 보기에서는 프로젝트 또는 애플리케이션 지표(예: CPU, 메모리, 대역폭 사용량, 네트워크 관련 정보)를 모니터링할 수 있는 옵션을 제공합니다.

7.1. 사전 요구 사항

- OpenShift Container Platform에서 애플리케이션을 생성하고 배포 했습니다.
- 웹 콘솔에 로그인하여 개발자 화면으로 전환했습니다.

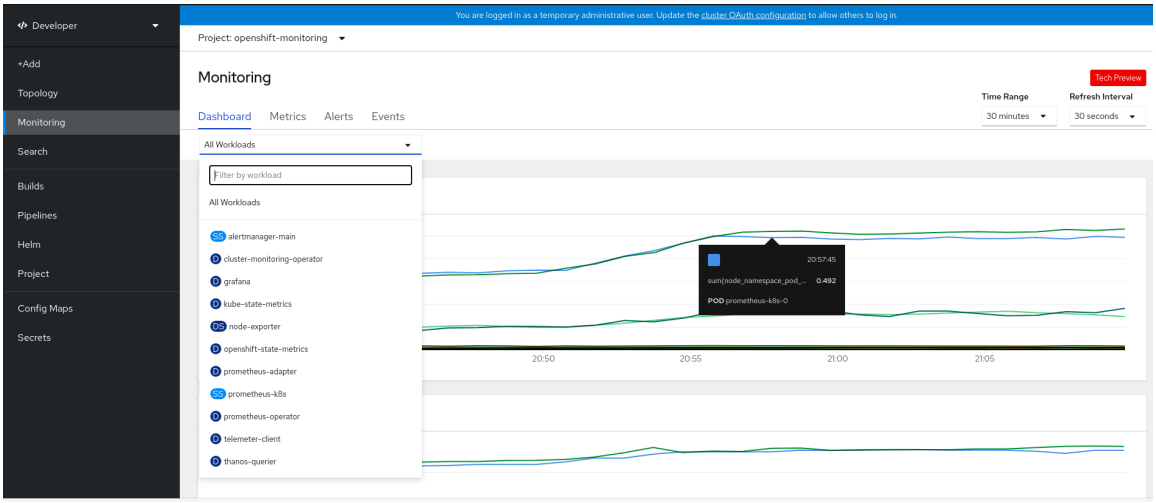
7.2. 프로젝트 지표 모니터링

프로젝트에 애플리케이션을 생성하고 배포한 후 웹 콘솔에서 개발자 화면을 사용하여 프로젝트의 지표를 확인할 수 있습니다.

프로세스

1. 개발자 화면의 왼쪽 탐색 패널에서 모니터링을 클릭하여 프로젝트의 대시보드, 지표, 알림, 이벤트를 확인합니다.
   - **Dashboard** 탭을 사용하여 CPU, 메모리, 대역폭 사용량, 네트워크 관련 정보(예: 전송 및 수신된 패킷 비율, 누락된 패킷 비율)를 보여주는 그래프를 확인합니다.

     그림 7.1. 모니터링 대시보드

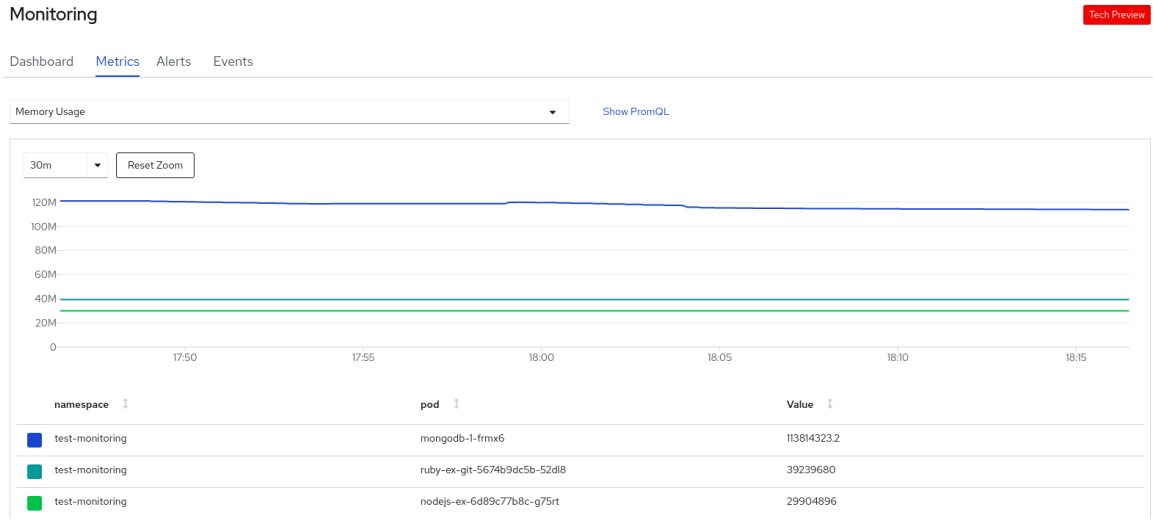


     자세한 내용을 확인하려면 다음 옵션을 사용합니다.

     - 선택한 워크로드에 대해 필터링된 지표를 확인하려면 모든 워크로드 목록에서 워크로드를 선택합니다.
     - 데이터 캡처 기간을 결정하려면 시간 범위 목록에서 옵션을 선택합니다.
     - 데이터를 새로 고친 후 기간을 결정하려면 새로 고침 간격 목록에서 옵션을 선택합니다.
     - Pod에 대한 특정 세부 정보를 확인하려면 그래프 위에 커서를 올려놓습니다.
     - 지표 페이지의 특정 지표에 대한 세부 정보를 확인하려면 표시된 그래프 중 하나를 클릭합니다.

- 필요한 프로젝트 지표를 쿼리하려면 지표 탭을 사용합니다.

그림 7.2. 지표 모니터링



a. 쿼리 선택 목록에서 프로젝트에 필요한 세부 정보를 필터링할 옵션을 선택합니다. 프로젝트의 모든 애플리케이션 Pod에 대해 필터링된 지표가 그래프에 표시됩니다. 프로젝트의 Pod도 아래에 나열됩니다.

b. 쿼리 결과를 추가로 필터링하기 위해 특정 Pod의 지표를 제거하려면 Pod 목록에서 색상이 지정된 사각형 상자를 지웁니다.

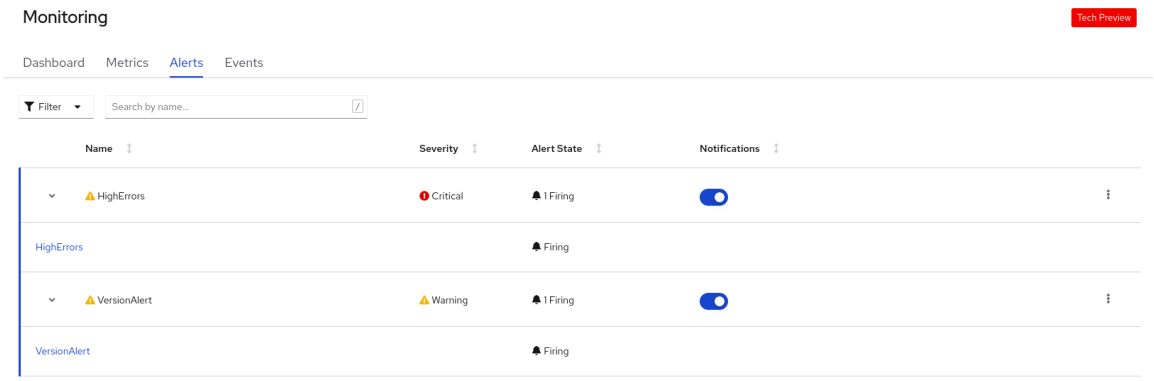
c. Prometheus 쿼리를 보려면 **PromQL** 표시를 클릭합니다. 쿼리를 사용자 지정하고 해당 네임스페이스에 표시할 지표를 필터링하기 위해 프롬프트의 도움말을 사용하여 이 쿼리를 추가로 수정할 수 있습니다.

d. 데이터가 표시되는 기간을 설정하려면 드롭다운 목록을 사용합니다. 확대/축소 재설정을 클릭하여 기본 시간 범위를 설정할 수 있습니다.

e. 필요한 경우 쿼리 선택 목록에서 사용자 정의 쿼리를 선택하여 사용자 정의 Prometheus 쿼리를 생성하고 관련 지표를 필터링합니다.

- 알림 탭에서는 프로젝트 내 애플리케이션에 대한 알림을 트리거하는 규칙을 확인하고 프로젝트에서 실행되는 알림을 파악한 후 필요한 경우 해당 알림을 음소거할 수 있습니다.

그림 7.3. 알림 모니터링



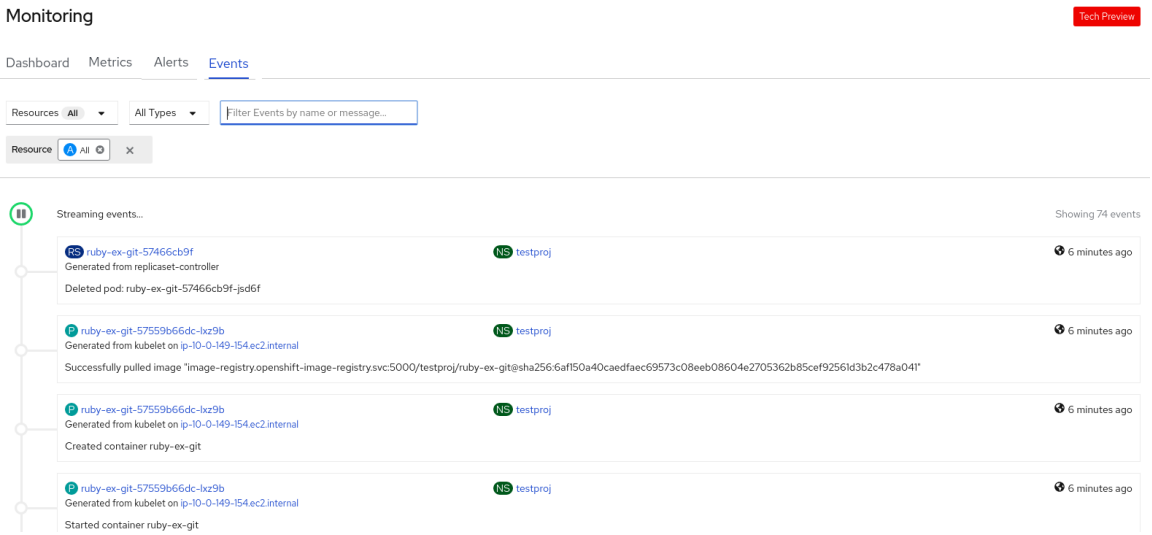
- 필터 목록을 사용하여 알림 상태 및 심각도로 알림을 필터링합니다.
- 알림을 클릭하여 해당 알림의 세부 정보 페이지로 이동합니다. 경고 세부 정보 페이지에서 지표 보기를 클릭하여 경고에 대한 지표를 확인할 수 있습니다.

- 알림 규칙 옆에 있는 알림 토글을 사용하여 해당 규칙에 대한 모든 알림을 음소거한 다음 음소거 기간 목록에서 알림을 음소거할 기간을 선택합니다. 알림 토글을 보려면 알림을 편집할 수 있는 권한이 있어야 합니다.

- 알림 규칙 세부 정보를 확인하려면 알림 규칙 옆에 있는 옵션 메뉴 를 사용합니다.

- 프로젝트의 이벤트를 보려면 이벤트 탭을 사용합니다.

그림 7.4. 모니터링 이벤트



다음 옵션을 사용하여 표시되는 이벤트를 필터링할 수 있습니다.

- 리소스 목록에서 리소스를 선택하여 해당 리소스에 대한 이벤트를 확인합니다.
- 모든 유형 목록에서 이벤트 유형을 선택하여 해당 유형과 관련된 이벤트를 확인합니다.
- 이름 또는 메시지로 이벤트 필터링 필드를 사용하여 특정 이벤트를 검색합니다.

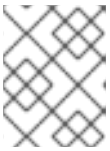
7.3. 애플리케이션 지표 모니터링

프로젝트에 애플리케이션을 생성하고 배포한 후에는 개발자 화면의 토폴로지 보기를 사용하여 애플리케이션에 대한 알림 및 지표를 확인할 수 있습니다. 애플리케이션에 대한 중요 및 경고 알림은 토폴로지 보기의 워크로드 노드에 표시됩니다.

프로세스

워크로드에 대한 알림을 보려면 다음을 수행합니다.

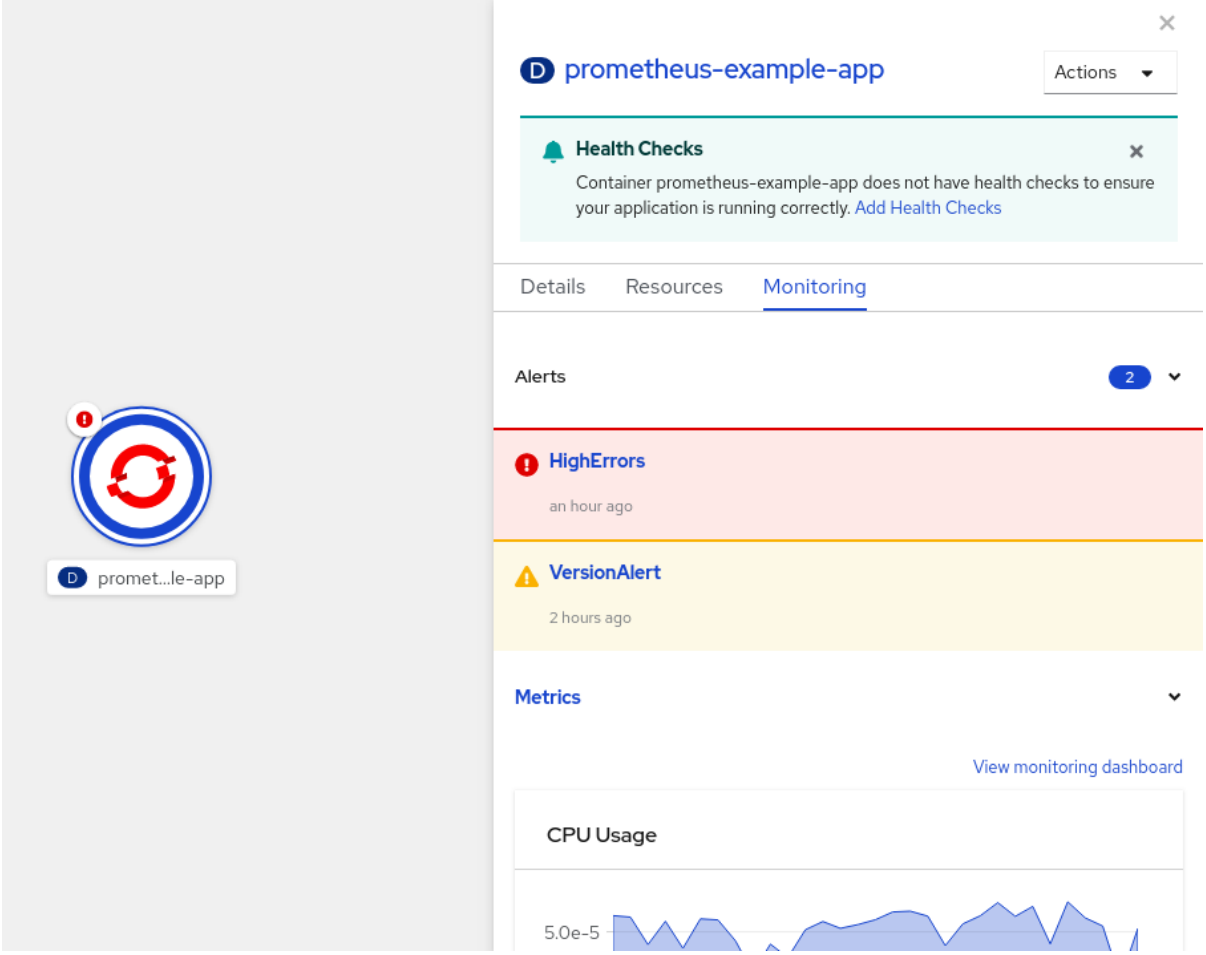
1. 토폴로지 보기에서 워크로드를 클릭하여 오른쪽 패널의 워크로드 세부 정보를 확인합니다.
2. 모니터링 탭을 클릭하여 애플리케이션에 대한 심각 및 경고 알림, 지표(예: CPU, 메모리, 대역폭 사용량)에 대한 그래프, 애플리케이션과 관련된 모든 이벤트를 확인합니다.



참고

토폴로지 보기에는 실행 중 상태의 중요 및 경고 알림만 표시됩니다. 음소거됨, 보류 중, 실행되지 않음 상태의 알림은 표시되지 않습니다.

그림 7.5. 애플리케이션 지표 모니터링



a. 오른쪽 패널에 나열된 알림을 클릭하여 알림 세부 정보 페이지에서 알림 세부 정보를 확인합니다.

b. 차트 중 하나를 클릭하여 지표 탭으로 이동하여 애플리케이션에 대한 세부 지표를 확인합니다.

c. 모니터링 대시보드 보기를 클릭하여 해당 애플리케이션의 모니터링 대시보드를 확인합니다.

7.4. 추가 리소스

- 모니터링 개요

8장. 상태 점검을 사용하여 애플리케이션 상태 모니터링

소프트웨어 시스템에서는 일시적인 문제(예: 일시적인 연결 끊김, 구성 오류 또는 외부 종속 항목의 문제)로 인해 구성 요소가 비정상 상태가 될 수 있습니다. OpenShift Container Platform 애플리케이션에는 비정상 상태의 컨테이너를 탐지하고 처리하는 다양한 옵션이 있습니다.

8.1. 상태 점검 이해

상태 점검에서는 준비 상태, 활성 상태, 시작 상태 점검을 함께 사용하여 실행 중인 컨테이너에서 정기적으로 진단을 수행합니다.

상태 점검을 수행할 컨테이너가 포함된 Pod 사양에 프로브를 1개 이상 포함할 수 있습니다.

참고

기존 Pod에서 상태 점검을 추가하거나 편집하려면 Pod **DeploymentConfig** 오브젝트를 편집하거나 웹 콘솔에서 개발자 화면을 사용해야 합니다. CLI에서는 기존 Pod의 상태 점검을 추가하거나 편집할 수 없습니다.

Readiness 프로브

*준비 상태 프로브*는 컨테이너가 서비스 요청을 수락할 준비가 되었는지 확인합니다. 컨테이너에 대한 준비 상태 프로브가 실패하면 kubelet에서 사용 가능한 서비스 끝점 목록에서 Pod를 제거합니다.
프로브는 실패 후에도 Pod를 계속 검사합니다. Pod를 사용할 수 있게 되면 kubelet은 사용 가능한 서비스 끝점 목록에 Pod를 추가합니다.

활성 상태 점검

*활성 상태 프로브*는 컨테이너가 계속 실행 중인지 확인합니다. 교착 상태와 같은 상태로 인해 활성 상태 프로브가 실패하면 kubelet에서 컨테이너를 종료합니다. 그런 다음 Pod는 재시작 정책에 따라 작업을 수행합니다.
예를 들어 **restartPolicy**가 **Always** 또는 **OnFailure**인 Pod의 활성 상태 프로브는 컨테이너를 종료한 후 다시 시작합니다.

Startup 프로브

*시작 프로브*는 컨테이너 내 애플리케이션이 시작되었는지를 나타냅니다. 기타 모든 프로브는 시작 프로브가 성공할 때까지 비활성화됩니다. 시작 프로브가 지정된 기간 내에 성공하지 못하면 kubelet에서 컨테이너를 종료하고 컨테이너에 Pod의 **restartPolicy**가 적용됩니다.
일부 애플리케이션에는 첫 번째 초기화 시 추가 시작 시간이 필요할 수 있습니다. 시작 프로브를 활성 상태 또는 준비 상태 프로브와 함께 사용하여 **failureThreshold** 및 **periodSeconds** 매개변수를 사용하여 해당 프로브를 긴 시작 시간을 처리할 수 있습니다.

예를 들어 실패 30회의 **failureThreshold**와 10초의 **periodSeconds**를 사용하면(30 * 10s = 300s) 최대 5분 동안 시작 프로브를 활성 상태 프로브에 추가할 수 있습니다. 시작 프로브가 처음으로 성공하면 활성 상태 프로브가 시작됩니다.

다음과 같은 테스트 유형을 사용하여 활성 상태 프로브, 준비 상태 프로브, 시작 프로브를 구성할 수 있습니다.

- HTTP **GET**: HTTP **GET** 테스트를 사용하는 경우 테스트는 웹 후크를 사용하여 컨테이너의 상태를 결정합니다. HTTP 응답 코드가 **200**에서 **399** 사이인 경우 테스트가 성공한 것입니다.
  HTTP **GET** 테스트는 완전히 초기화되었을 때 HTTP 상태 코드를 반환하는 애플리케이션에 사용할 수 있습니다.

- 컨테이너 명령: 컨테이너 명령 테스트를 사용하는 경우 프로브는 컨테이너 내부에서 명령을 실행합니다. 테스트가 상태 **0**으로 종료되면 프로브가 성공한 것입니다.

- TCP 소켓: TCP 소켓 테스트를 사용하는 경우 프로브는 컨테이너에 소켓을 열려고 합니다. 컨테이너는 프로브에서 연결을 설정할 수 있는 경우에만 정상 상태로 간주됩니다. TCP 소켓 테스트는 초기화가 완료된 후 수신 대기를 시작하는 애플리케이션에 사용할 수 있습니다.

다음과 같은 다양한 필드를 구성하여 프로브 동작을 제어할 수 있습니다.

- **initialDelaySeconds**: 컨테이너를 시작한 후 프로브를 예약할 수 있는 시간(초)입니다. 기본값은 0입니다.

- **periodSeconds**: 프로브 수행 사이의 지연 시간(초)입니다. 기본값은 **10**입니다. 이 값은 **timeoutSeconds** 보다 커야 합니다.

- **timeoutSeconds**: 프로브가 시간 초과되고 컨테이너가 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 **1**입니다. 이 값은 **periodSeconds** 보다 작아야 합니다.

- **successThreshold**: 실패 후 프로브에서 컨테이너 상태를 성공으로 재설정해야 하는 횟수입니다. 활성 상태 프로브는 값이 **1**이어야 합니다. 기본값은 **1**입니다.

- **failureThreshold**: 프로브가 실패할 수 있는 횟수입니다. 기본값은 3입니다. 지정된 시도 횟수 후에는 다음이 수행됩니다.

  - 활성 상태 프로브의 경우 컨테이너가 재시작됩니다.

  - 준비 상태 프로브의 경우 Pod가 **Unready**로 표시됩니다.

  - 시작 프로브의 경우 컨테이너가 종료되고 Pod의 **restartPolicy**가 적용됩니다.

프로브 예

다음은 오브젝트 사양에 나타나는 다양한 프로브 샘플입니다.

Pod 사양에 컨테이너 명령 준비 상태 프로브가 포함된 샘플 준비 상태 프로브

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
...
spec:
  containers:
  - name: goproxy-app 1
    args:
    image: k8s.gcr.io/goproxy:0.1 2
    readinessProbe: 3
      exec: 4
        command: 5
        - cat
        - /tmp/healthy
...
```

1 컨테이너 이름입니다.

2 배포할 컨테이너 이미지입니다.

3 Readiness 프로브입니다.

4 컨테이너 명령 테스트입니다.

5 컨테이너에서 실행할 명령입니다.

Pod 사양에 컨테이너 명령 테스트가 포함된 샘플 컨테이너 명령 시작 프로브 및 활성 상태 프로브

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
...
spec:
  containers:
  - name: goproxy-app 1
    args:
    image: k8s.gcr.io/goproxy:0.1 2
    livenessProbe: 3
      httpGet: 4
        scheme: HTTPS 5
        path: /healthz
        port: 8080 6
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    startupProbe: 7
      httpGet: 8
        path: /healthz
        port: 8080 9
      failureThreshold: 30 10
      periodSeconds: 10 11
...
```

1 컨테이너 이름입니다.

2 배포할 컨테이너 이미지를 지정합니다.

3 활성 상태 프로브

4 HTTP **GET** 테스트입니다.

5 인터넷 스키마: **HTTP** 또는 **HTTPS**. 기본값은 **HTTP**입니다.

6 컨테이너가 수신 대기 중인 포트입니다.

7 시작 프로브입니다.

8 HTTP **GET** 테스트입니다.

9 컨테이너가 수신 대기 중인 포트입니다.

10 실패 후 프로브에 시도할 수 있는 횟수입니다.

11 프로브를 수행할 시간(초)입니다.

Pod 사양의 타임아웃을 사용하는 컨테이너 명령 테스트가 포함된 샘플 활성 상태 프로브

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
...
spec:
  containers:
  - name: goproxy-app 1
    args:
    image: k8s.gcr.io/goproxy:0.1 2
    livenessProbe: 3
      exec: 4
        command: 5
        - /bin/bash
        - '-c'
        - timeout 60 /opt/eap/bin/livenessProbe.sh
      periodSeconds: 10 6
      successThreshold: 1 7
      failureThreshold: 3 8
...
```

1 컨테이너 이름입니다.

2 배포할 컨테이너 이미지를 지정합니다.

3 활성 상태 프로브입니다.

4 프로브 유형으로 여기서는 컨테이너 명령 프로브입니다.

5 컨테이너 내부에서 실행할 명령줄입니다.

6 프로브를 수행하는 빈도(초)입니다.

7 실패 후 성공으로 표시하는 데 필요한 연속 성공 횟수입니다.

8 실패 후 프로브에 시도할 수 있는 횟수입니다.

배포에 TCP 소켓 테스트를 사용하는 샘플 준비 상태 프로브 및 활성 상태 프로브

```
kind: Deployment
apiVersion: apps/v1
...
spec:
```

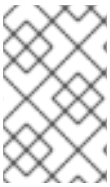
```
...
  template:
    spec:
      containers:
        - resources: {}
          readinessProbe: 1
            tcpSocket:
              port: 8080
            timeoutSeconds: 1
            periodSeconds: 10
            successThreshold: 1
            failureThreshold: 3
          terminationMessagePath: /dev/termination-log
          name: ruby-ex
          livenessProbe: 2
            tcpSocket:
              port: 8080
            initialDelaySeconds: 15
            timeoutSeconds: 1
            periodSeconds: 10
            successThreshold: 1
            failureThreshold: 3
...
```

1 준비 상태 프로브입니다.

2 활성 상태 프로브입니다.

8.2. CLI를 사용하여 상태 점검 구성

준비 상태 프로브, 활성 상태 프로브, 시작 프로브를 구성하려면 상태 점검을 수행할 컨테이너가 포함된 Pod의 사양에 프로브를 한 개 이상 추가합니다.



참고

기존 Pod에서 상태 점검을 추가하거나 편집하려면 Pod **DeploymentConfig** 오브젝트를 편집하거나 웹 콘솔에서 개발자 화면을 사용해야 합니다. CLI에서는 기존 Pod의 상태 점검을 추가하거나 편집할 수 없습니다.

프로세스

컨테이너에 대한 프로브를 추가하려면 다음을 수행합니다.

1. 프로브를 한 개 이상 추가할 **Pod** 오브젝트를 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
spec:
  containers:
  - name: my-container 1
```

```
    args:
    image: k8s.gcr.io/goproxy:0.1 (2)
    livenessProbe: (3)
      tcpSocket: (4)
        port: 8080 (5)
      initialDelaySeconds: 15 (6)
      periodSeconds: 20 (7)
      timeoutSeconds: 10 (8)
    readinessProbe: (9)
      httpGet: (10)
        host: my-host (11)
        scheme: HTTPS (12)
        path: /healthz
        port: 8080 (13)
    startupProbe: (14)
      exec: (15)
        command: (16)
        - cat
        - /tmp/healthy
      failureThreshold: 30 (17)
      periodSeconds: 20 (18)
      timeoutSeconds: 10 (19)
```

(1) 컨테이너 이름을 지정합니다.

(2) 배포할 컨테이너 이미지를 지정합니다.

(3) 선택 사항: 활성 상태 프로브를 생성합니다.

(4) 수행할 테스트를 지정합니다. 여기서는 TCP 소켓 테스트를 지정합니다.

(5) 컨테이너가 수신 대기 중인 포트를 지정합니다.

(6) 컨테이너를 시작한 후 프로브를 예약할 수 있을 때까지 걸리는 시간을 초 단위로 지정합니다.

(7) 프로브를 수행할 시간(초)을 지정합니다. 기본값은 **10**입니다. 이 값은 **timeoutSeconds** 보다 커야 합니다.

(8) 프로브가 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 **1**입니다. 이 값은 **periodSeconds** 보다 작아야 합니다.

(9) 선택 사항: 준비 상태 프로브 생성.

(10) 수행할 테스트 유형을 지정합니다. 여기서는 HTTP 테스트를 지정합니다.

(11) 호스트 IP 주소를 지정합니다. **host**가 정의되지 않은 경우 **PodIP**가 사용됩니다.

(12) **HTTP** 또는 **HTTPS**를 지정합니다. **scheme**가 정의되지 않은 경우 **HTTP** 스키마가 사용됩니다.

(13) 컨테이너가 수신 대기 중인 포트를 지정합니다.

(14) 선택 사항: 시작 프로브 생성.

15 수행할 테스트 유형을 지정합니다. 여기서는 컨테이너 실행 프로브를 지정합니다.

16 컨테이너에서 실행할 명령을 지정합니다.

17 실패 후 프로브에 시도할 수 있는 횟수를 지정합니다.

18 프로브를 수행할 시간(초)을 지정합니다. 기본값은 **10**입니다. 이 값은 **timeoutSeconds** 보다 커야 합니다.

19 프로브가 실패한 것으로 간주되는 비활성 시간(초)입니다. 기본값은 **1**입니다. 이 값은 **periodSeconds** 보다 작아야 합니다.

참고

initialDelaySeconds 값이 **periodSeconds** 값보다 작으면 타이머 관련 문제로 인해 첫 번째 준비 상태 프로브가 두 기간 사이의 어느 시점에 수행됩니다.

timeoutSeconds 값은 **periodSeconds** 값보다 작아야 합니다.

2. **Pod** 오브젝트를 생성합니다.

```
$ oc create -f <file-name>.yaml
```

3. 상태 점검 Pod의 상태를 확인합니다.

```
$ oc describe pod health-check
```

출력 예

```
Events:
  Type    Reason     Age   From                                Message
  ----    ------     ----  ----                                -------
  Normal  Scheduled  9s    default-scheduler                   Successfully assigned openshift-
logging/liveness-exec to ip-10-0-143-40.ec2.internal
  Normal  Pulling    2s    kubelet, ip-10-0-143-40.ec2.internal  pulling image
"k8s.gcr.io/liveness"
  Normal  Pulled     1s    kubelet, ip-10-0-143-40.ec2.internal  Successfully pulled image
"k8s.gcr.io/liveness"
  Normal  Created    1s    kubelet, ip-10-0-143-40.ec2.internal  Created container
  Normal  Started    1s    kubelet, ip-10-0-143-40.ec2.internal  Started container
```

다음은 컨테이너를 재시작한 실패한 프로브의 출력입니다.

비정상 상태의 컨테이너가 있는 샘플 활성 상태 점검 출력

```
$ oc describe pod pod1
```

출력 예

```
....

Events:
```

```
  Type     Reason          Age                From                                        Message
  ----     ------          ----               ----                                        -------
  Normal   Scheduled       <unknown>                                                      Successfully
assigned aaa/liveness-http to ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
  Normal   AddedInterface  47s                multus                                      Add eth0
[10.129.2.11/23]
  Normal   Pulled          46s                kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Successfully pulled image "k8s.gcr.io/liveness" in 773.406244ms
  Normal   Pulled          28s                kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Successfully pulled image "k8s.gcr.io/liveness" in 233.328564ms
  Normal   Created         10s (x3 over 46s)  kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Created container liveness
  Normal   Started         10s (x3 over 46s)  kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Started container liveness
  Warning  Unhealthy       10s (x6 over 34s)  kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-
snzrj  Liveness probe failed: HTTP probe failed with statuscode: 500
  Normal   Killing         10s (x2 over 28s)  kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Container liveness failed liveness probe, will be restarted
  Normal   Pulling         10s (x3 over 47s)  kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Pulling image "k8s.gcr.io/liveness"
  Normal   Pulled          10s                kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snzrj
Successfully pulled image "k8s.gcr.io/liveness" in 244.116568ms
```

8.3. 개발자 화면을 사용하여 애플리케이션 상태 모니터링

개발자 화면을 통해 컨테이너에 세 가지 유형의 상태 프로브를 추가하여 애플리케이션이 정상인지 확인할 수 있습니다.

- 준비 상태 프로브를 사용하여 컨테이너가 요청을 처리할 준비가 되었는지 확인합니다.
- 활성 상태 프로브를 사용하여 컨테이너가 실행 중인지 확인합니다.
- 시작 프로브를 사용하여 컨테이너 내 애플리케이션이 시작되었는지 확인합니다.

애플리케이션을 생성하고 배포하는 동안 또는 애플리케이션을 배포한 후 상태 점검을 추가할 수 있습니다.

8.4. 개발자 화면을 사용하여 상태 점검 추가

토폴로지 보기를 사용하여 배포된 애플리케이션에 상태 점검을 추가할 수 있습니다.

사전 요구 사항

- 웹 콘솔에서 개발자 화면으로 전환했습니다.
- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션을 생성하고 배포했습니다.

프로세스

1. 토폴로지 보기에서 애플리케이션 노드를 클릭하여 측면 패널을 확인합니다. 애플리케이션의 원활한 실행을 위해 컨테이너에 상태 점검을 추가하지 않은 경우 상태 점검을 추가할 수 있는 링크가 포함된 상태 점검 알림이 표시됩니다.
2. 표시된 알림에서 상태 점검 추가 링크를 클릭합니다.

3. 또는 작업 드롭다운 목록을 클릭한 후 상태 점검 추가를 선택해도 됩니다. 컨테이너에 이미 상태 점검이 있는 경우 추가 옵션 대신 상태 점검 편집 옵션이 표시됩니다.

4. 컨테이너를 여러 개 배포한 경우 상태 점검 추가 양식에서 컨테이너 드롭다운 목록을 사용하여 컨테이너가 적절하게 선택되었는지 확인합니다.

5. 필요한 상태 프로브 링크를 클릭하여 컨테이너에 추가합니다. 상태 점검을 위한 기본 데이터는 미리 채워집니다. 기본 데이터를 사용하여 프로브를 추가하거나 값을 추가로 사용자 지정한 후 추가할 수 있습니다. 예를 들어 컨테이너가 요청을 처리할 준비가 되었는지 확인하는 준비 상태 프로브를 추가하려면 다음을 수행합니다.

   a. 준비 상태 프로브 추가를 클릭하여 프로브에 대한 매개변수가 포함된 양식을 확인합니다.

   b. 유형 드롭다운 목록을 클릭하여 추가할 요청 유형을 선택합니다. 예를 들어 이 경우에는 컨테이너 명령을 선택하여 컨테이너 내에서 실행할 명령을 선택합니다.

   c. 명령 필드에서 인수 **cat**을 추가합니다. 마찬가지로 검사를 위해 여러 인수를 추가할 수 있습니다(예: 다른 인수 **/tmp/healthy** 추가).

   d. 필요에 따라 다른 매개변수의 기본값을 유지하거나 수정합니다.

   참고

   timeout 값은 **period** 값보다 작아야 합니다. **timeout** 기본값은 **1**입니다. **Period** 기본값은 **10**입니다.

   e. 양식 하단의 확인 표시를 클릭합니다. 준비 상태 프로브 추가됨 메시지가 표시됩니다.

6. 추가를 클릭하여 상태 점검을 추가합니다. 토폴로지 보기로 리디렉션되고 컨테이너가 재시작됩니다.

7. 측면 패널에서 **Pod** 섹션에 배포된 Pod를 클릭하여 프로브가 추가되었는지 확인합니다.

8. **Pod** 세부 정보 페이지에서 컨테이너 섹션에 나열된 컨테이너를 클릭합니다.

9. 컨테이너 세부 정보 페이지에서 준비 상태 프로브(**Exec Command cat /tmp/healthy**)가 컨테이너에 추가되었는지 확인합니다.

8.5. 개발자 화면을 사용하여 상태 점검 편집

토폴로지 보기를 사용하여 애플리케이션에 추가된 상태 점검을 편집 또는 수정하거나 더 많은 상태 점검을 추가할 수 있습니다.

사전 요구 사항

- 웹 콘솔에서 개발자 화면으로 전환했습니다.
- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션을 생성하고 배포했습니다.
- 애플리케이션에 상태 점검을 추가했습니다.

프로세스

1. 토폴로지 보기에서 애플리케이션을 마우스 오른쪽 버튼으로 클릭하고 상태 점검 편집을 선택합니다. 또는 측면 패널에서 작업 드롭다운 목록을 클릭하고 상태 점검 편집을 선택합니다.

2. 상태 점검 편집 페이지에서 다음을 수행합니다.

   - 이전에 추가한 상태 프로브를 제거하려면 옆에 있는 빼기(-) 기호를 클릭합니다.
   - 기존 프로브의 매개변수를 편집하려면 다음을 수행합니다.
     a. 이전에 추가한 프로브 옆에 있는 프로브 편집 링크를 클릭하여 프로브의 매개변수를 확인합니다.
     b. 필요에 따라 매개변수를 수정하고 확인 표시를 클릭하여 변경 사항을 저장합니다.
   - 기존 상태 점검 외에 새 상태 프로브를 추가하려면 프로브 링크 추가를 클릭합니다. 예를 들어 컨테이너가 실행 중인지 확인하는 활성 상태 프로브를 추가하려면 다음을 수행합니다.
     a. 활성 상태 프로브를 클릭하여 프로브에 대한 매개변수가 포함된 양식을 확인합니다.
     b. 필요에 따라 프로브 매개변수를 편집합니다.

     참고

     timeout 값은 **period** 값보다 작아야 합니다. **timeout** 기본값은 **1**입니다. **Period** 기본값은 **10**입니다.

     c. 양식 하단의 확인 표시를 클릭합니다. 활성 상태 프로브 추가됨 메시지가 표시됩니다.

3. 저장을 클릭하여 수정 사항을 저장하고 컨테이너에 프로브를 추가합니다. 토폴로지 보기로 리디렉션됩니다.

4. 측면 패널에서 **Pod** 섹션에 배포된 Pod를 클릭하여 프로브가 추가되었는지 확인합니다.

5. **Pod** 세부 정보 페이지에서 컨테이너 섹션에 나열된 컨테이너를 클릭합니다.

6. 컨테이너 세부 정보 페이지에서 활성 상태 프로브(**HTTP Get 10.129.4.65:8080/**)가 이전의 기존 프로브 외에 컨테이너에도 추가되었는지 확인합니다.

8.6. 개발자 화면을 사용하여 상태 점검 실패 모니터링

애플리케이션 상태 점검이 실패하는 경우 토폴로지 보기를 사용하여 이러한 상태 점검 위반을 모니터링할 수 있습니다.

사전 요구 사항

- 웹 콘솔에서 개발자 화면으로 전환했습니다.
- 개발자 화면을 사용하여 OpenShift Container Platform에서 애플리케이션을 생성하고 배포했습니다.
- 애플리케이션에 상태 점검을 추가했습니다.

프로세스

1. 토폴로지 보기에서 애플리케이션 노드를 클릭하여 측면 패널을 확인합니다.

2. 모니터링 탭을 클릭하여 이벤트(경고) 섹션에서 상태 점검 실패를 확인합니다.

3. 이벤트(Warning) 옆에 있는 아래쪽 화살표를 클릭하여 상태 점검 실패 세부 정보를 확인합니다.

추가 리소스

- 웹 콘솔에서 개발자 화면으로 전환하는 방법에 대한 자세한 내용은 개발자 화면 정보를 참조하십시오.
- 애플리케이션을 생성하고 배포하는 동안 상태 점검을 추가하는 방법에 대한 자세한 내용은 개발자 화면을 사용하여 애플리케이션 생성 섹션의 고급 옵션을 참조하십시오.

9장. 애플리케이션을 유휴 상태로 설정

클러스터 관리자는 리소스 사용을 줄이기 위해 애플리케이션을 유휴 상태로 설정할 수 있습니다. 이는 비용이 리소스 사용과 연결된 퍼블릭 클라우드에 클러스터를 배포할 때 유용합니다.

확장 가능 리소스가 사용되지 않은 경우 OpenShift Container Platform은 해당 리소스를 검색하고 복제본 수를 **0**으로 스케일링하여 유휴 상태로 설정합니다. 다음에 네트워크 트래픽이 리소스로 전달되면 복제본이 확장되어 리소스의 유휴 상태가 해제되고 정상적인 작업이 계속됩니다.

애플리케이션은 서비스 및 기타 확장 가능한 리소스(예: 배포 구성)로 구성됩니다. 애플리케이션을 유휴 상태로 설정하는 작업에서는 관련 리소스를 모두 유휴 상태로 설정합니다.

9.1. 애플리케이션을 유휴 상태로 설정

애플리케이션을 유휴 상태로 설정하려면 서비스와 관련된 확장 가능 리소스(배포 구성, 복제 컨트롤러 등)를 찾아야 합니다. 애플리케이션을 유휴 상태로 설정하는 작업에서는 서비스를 검색하여 유휴 상태로 표시하고 리소스를 복제본 수 0개로 축소합니다.

oc idle 명령을 사용하여 단일 서비스를 유휴 상태로 설정하거나 **--resource-names-file** 옵션을 사용하여 여러 서비스를 유휴 상태로 설정할 수 있습니다.

9.1.1. 단일 서비스를 유휴 상태로 설정

프로세스

1. 단일 서비스를 유휴 상태로 설정하려면 다음을 실행합니다.

```
$ oc idle <service>
```

9.1.2. 여러 서비스를 유휴 상태로 설정

여러 서비스를 유휴 상태로 설정하는 것은 애플리케이션이 프로젝트 내의 여러 서비스에 걸쳐 있거나 동일한 프로젝트 내의 여러 애플리케이션을 대량으로 유휴 상태로 설정하기 위해 스크립트와 함께 사용하여 여러 서비스를 유휴 상태로 설정할 때 유용합니다.

프로세스

1. 각 행에 서비스 목록이 포함된 파일을 생성합니다.
2. **--resource-names-file** 옵션을 사용하여 서비스를 유휴 상태로 설정합니다.

```
$ oc idle --resource-names-file <filename>
```

참고

© 명령은 단일 프로젝트로 제한됩니다. 클러스터 전체의 애플리케이션을 유휴 상태로 설정하려면 각 프로젝트에 개별적으로 **idle** 명령을 실행합니다.

9.2. 애플리케이션 유휴 상태 해제

애플리케이션 서비스는 네트워크 트래픽을 수신할 때 다시 활성화되고 이전 상태로 확장됩니다. 여기에는 서비스에 대한 트래픽과 경로를 통과하는 트래픽이 모두 포함됩니다.

리소스를 확장하여 애플리케이션의 유휴 상태를 수동으로 해제할 수도 있습니다.

프로세스

1. DeploymentConfig를 확장하려면 다음을 실행합니다.

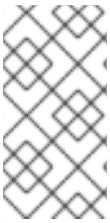
```
$ oc scale --replicas=1 dc <dc_name>
```

참고

라우터에 의한 자동 유휴 상태 해제는 현재 기본 HAProxy 라우터에서만 지원합니다.

참고

Kuryr-Kubernetes를 SDN으로 구성하면 서비스는 자동 유휴 상태를 지원하지 않습니다.



참고

OpenShift Container Platform 4.6에서는 leader **ovnkube-master** 프로세스가 재시작되거나 **ovnkube-master** 리더가 변경된 후 네트워크 트래픽을 수신하는 경우 유휴 상태의 서비스가 다시 활성화되지 않습니다. 자세한 내용은 Red Hat OpenShift Container Platform 4에서 ovnkube-master를 다시 시작한 후 Idled 서비스가 자동으로 발생할 수 없습니다.

10장. 리소스 회수를 위한 오브젝트 정리

시간이 지남에 따라 OpenShift Container Platform에서 생성된 API 오브젝트는 애플리케이션을 빌드하고 배포할 때와 같이 일반 사용자 작업을 통해 클러스터의 etcd 데이터 저장소에 누적될 수 있습니다.

클러스터 관리자는 더 이상 필요하지 않은 이전 버전의 오브젝트를 클러스터에서 주기적으로 정리할 수 있습니다. 예를 들어 이미지를 정리하면 더 이상 사용하지 않지만 디스크 공간을 차지하고 있는 오래된 이미지와 계층을 삭제할 수 있습니다.

10.1. 기본 정리 작업

CLI 그룹은 공통 상위 명령 아래의 작업을 정리합니다.

```
$ oc adm prune <object_type> <options>
```

이 명령은 다음을 지정합니다.

- 작업을 수행할 **<object_type>**(예: **groups**, **builds**, **deployments** 또는 **images**)
- 해당 오브젝트 유형을 정리하도록 지원하는 **<options>**

10.2. 그룹 정리

관리자는 다음 명령을 실행하여 외부 공급자의 그룹 레코드를 정리할 수 있습니다.

```
$ oc adm prune groups \
    --sync-config=path/to/sync/config [<options>]
```

표 10.1. **oc adm prune groups** 플래그

옵션	설명
--confirm	시험 실행하는 대신 정리를 수행해야 함을 나타냅니다.
--blacklist	그룹 블랙리스트 파일의 경로입니다.
--whitelist	그룹 허용 목록 파일의 경로입니다.
--sync-config	동기화 구성 파일의 경로입니다.

프로세스

1. prune 명령에서 삭제하는 그룹을 확인하려면 다음 명령을 실행합니다.

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml
```

2. 정리 작업을 수행하려면 **--confirm** 플래그를 추가합니다.

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml --confirm
```

10.3. 배포 리소스 정리

수명 및 상태로 인해 시스템에서 더 이상 필요하지 않은 배포와 관련된 리소스를 정리할 수 있습니다.

다음 명령은 DeploymentConfig 오브젝트 와 관련된 복제 컨트롤러를 정리합니다.

```
$ oc adm prune deployments [<options>]
```

표 10.2. **oc adm prune deployment** 플래그

옵션	설명
--confirm	시험 실행하는 대신 정리를 수행해야 함을 나타냅니다.
--keep-complete=<N>	**DeploymentConfig** 오브젝트마다 상태가 **Complete** 이고 복제본 수가 0인 마지막 **N** 개의 복제 컨트롤러를 유지합니다. 기본값은 **5** 입니다.
--keep-failed=<N>	**DeploymentConfig** 오브젝트마다 상태가 **Failed** 이고 복제본 수가 0인 마지막 **N** 개의 복제 컨트롤러를 유지합니다. 기본값은 **1** 입니다.
--keep-younger-than=<duration>	현재 시간을 기준으로 < **duration>** 이내의 복제 컨트롤러를 정리하지 마십시오. 유효한 측정 단위에는 나노초(ns), 마이크로초(**m**), 밀리초(**ms**), 초(S), 분(m), 시간(**h**)이 포함됩니다. 기본값은 **60m** 입니다**.**
--orphans	더 이상 **DeploymentConfig** 오브젝트가 없는 모든 복제 컨트롤러를 정리하고 상태가 **Complete** 또는 **Failed** 인 경우 복제본 수가 0입니다.

프로세스

1. 정리 작업에서 삭제하는 내용을 확인하려면 다음 명령을 실행합니다.

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

2. 실제로 정리 작업을 수행하려면 **--confirm** 플래그를 추가합니다.

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

10.4. 빌드 정리

관리자는 다음 명령을 실행하여 수명 및 상태로 인해 시스템에서 더 이상 필요하지 않은 빌드를 정리할 수 있습니다.

```
$ oc adm prune builds [<options>]
```

표 10.3. **oc adm prune builds** 플래그

옵션	설명
--confirm	시험 실행하는 대신 정리를 수행해야 함을 나타냅니다.
--orphans	빌드 구성이 더 이상 존재하지 않거나 상태가 완료됨, 실패, 오류 또는 취소됨인 모든 빌드를 정리합니다.
--keep-complete=<N>	빌드 구성별로 상태가 완료된 마지막 빌드 **N** 개를 유지합니다. 기본값은 **5** 입니다.
--keep-failed=<N>	빌드 구성마다 상태가 failed, error 또는 canceled인 마지막 빌드 **N** 개를 유지합니다. 기본값은 **1**입니다.
--keep-younger-than=<duration>	현재 시간을 기준으로 < **duration>** 이내의 오브젝트를 정리하지 마십시오. 기본값은 **60m** 입니다.

프로세스

1. 정리 작업에서 삭제하는 내용을 확인하려면 다음 명령을 실행합니다.

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

2. 실제로 정리 작업을 수행하려면 **--confirm** 플래그를 추가합니다.

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

참고

개발자는 빌드 구성을 수정하여 자동 빌드 정리를 활성화할 수 있습니다.

추가 리소스

- 고급 빌드 수행 → 빌드 정리

10.5. 이미지 자동 정리

기간, 상태 또는 제한 초과로 인해 시스템에서 더 이상 필요하지 않은 이미지가 자동으로 정리됩니다. 클러스터 관리자는 사용자 정의 리소스 정리를 구성하거나 일시 중단할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- **oc** CLI를 설치합니다.

프로세스

- **imagepruners.imageregistry.operator.openshift.io/cluster**라는 오브젝트에 다음 **spec** 및 **status** 필드가 있는지 확인합니다.

```
spec:
  schedule: 0 0 * * * (1)
  suspend: false (2)
  keepTagRevisions: 3 (3)
  keepYoungerThanDuration: 60m (4)
  keepYoungerThan: 3600000000000 (5)
  resources: {} (6)
  affinity: {} (7)
  nodeSelector: {} (8)
  tolerations: [] (9)
  successfulJobsHistoryLimit: 3 (10)
  failedJobsHistoryLimit: 3 (11)
status:
  observedGeneration: 2 (12)
  conditions: (13)
  - type: Available
    status: "True"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Ready
    message: "Periodic image pruner has been created."
  - type: Scheduled
    status: "True"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Scheduled
    message: "Image pruner job has been scheduled."
  - type: Failed
    staus: "False"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Succeeded
    message: "Most recent image pruning job succeeded."
```

(1) 일정: **CronJob** 형식의 일정. 선택적 필드이며 기본값은 매일 자정입니다.

(2) **suspend**: **true** 로 설정하면 정리를 실행하는 **CronJob** 이 일시 중단됩니다. 선택적 필드이며 기본값은 **false**입니다. 새 클러스터의 초깃값은 **false**입니다.

(3) **keepTagRevisions**: 유지할 태그당 버전 수입니다. 선택적 필드이며 기본값은 **3**입니다. 초깃값은 **3**입니다.

(4) **keepYoungerThanDuration**: 이 기간보다 짧은 이미지를 유지합니다. 이 필드는 선택적 필드입니다. 값을 지정하지 않으면 **keepYoungerThan** 또는 기본값인 **60m**(60분)이 사용됩니다.

(5) **keepYoungerThan**: 더 이상 사용되지 않음. **keepYoungerThanDuration**과 동일하지만 기간은 나노초 단위의 정수로 지정됩니다. 이 필드는 선택적 필드입니다. **keepYoungerThanDuration**이 설정되면 이 필드는 무시됩니다.

(6) **resources**: 표준 Pod 리소스 요청 및 제한 이 필드는 선택적 필드입니다.

(7) 유사성: 표준 Pod 유사성입니다. 이 필드는 선택적 필드입니다.

(8) **nodeSelector**: 표준 Pod 노드 선택기입니다. 이 필드는 선택적 필드입니다.

9 허용 오차: 표준 Pod 허용 오차입니다. 이 필드는 선택적 필드입니다.

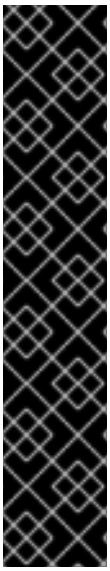
10 **successfulJobsHistoryLimit**: 유지할 성공 작업의 최대 수입니다. 지표를 보고하려면 **>= 1**이어야 합니다. 선택적 필드이며 기본값은 **3**입니다. 초깃값은 **3**입니다.

11 **failedJobsHistoryLimit**: 유지할 실패한 작업의 최대 수입니다. 지표를 보고하려면 **>= 1**이어야 합니다. 선택적 필드이며 기본값은 **3**입니다. 초깃값은 **3**입니다.

12 **observedGeneration**: Operator에서 관찰한 생성입니다.

13 조건: 다음 유형의 표준 조건 오브젝트입니다.

- 사용 가능: 정리 작업이 생성되었는지를 나타냅니다. 이유는 준비 또는 오류일 수 있습니다.
- **Scheduled**: 다음 정리 작업이 예약되었는지를 나타냅니다. 이유는 스케줄링, 일시 중지 또는 오류일 수 있습니다.
- 실패: 가장 최근 정리 작업이 실패했는지를 나타냅니다.



중요

정리기 관리에 필요한 Image Registry Operator의 동작은 Image Registry Operator의 **ClusterOperator** 오브젝트에 지정된 **managementState**와는 별개입니다. Image Registry Operator가 **Managed** 상태가 아닌 경우에도 이미지 정리기를 정리 사용자 정의 리소스에서 설정 및 관리할 수 있습니다.

그러나 이미지 레지스트리 Operator의 **managementState**는 배포된 이미지 pruner 작업의 동작을 변경합니다.

- **Managed**: 이미지 pruner의 **--prune-registry** 플래그가 **true**로 설정됩니다.
- **Removed**: 이미지 pruner의 **--prune-registry** 플래그가 **false** 로 설정되어 etcd의 이미지 메타 데이터만 정리합니다.
- **Unmanaged** : 이미지 pruner의 **--prune-registry** 플래그가 **false**로 설정됩니다.

10.6. 수동으로 이미지 제거

정리 사용자 정의 리소스를 사용하면 이미지를 자동으로 정리할 수 있습니다. 그러나 관리자는 기간, 상태 또는 제한 초과로 인해 시스템에서 더 이상 필요하지 않은 이미지를 수동으로 정리할 수 있습니다. 이미지를 수동으로 정리하는 방법은 다음 두 가지입니다.

- 클러스터에서 **Job** 또는 **CronJob**으로 이미지 정리를 실행합니다.
- **oc adm prune images** 명령을 실행합니다.

사전 요구 사항

- 이미지를 정리하려면 먼저 액세스 토큰이 있는 사용자로 CLI에 로그인해야 합니다. 사용자에게 **system:image-pruner** 클러스터 이상의 역할(예: **cluster-admin**)이 있어야 합니다.
- 이미지 레지스트리를 노출합니다.

프로세스

수명, 상태 또는 제한 초과로 인해 시스템에서 더 이상 필요하지 않은 이미지를 수동으로 정리하려면 다음 방법 중 하나를 사용합니다.

- **pruner** 서비스 계정에 대한 YAML 파일을 생성하여 클러스터에서 **Job** 또는 **CronJob**으로 이미지 정리를 실행합니다. 예를 들면 다음과 같습니다.

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

출력 예

```
kind: List
apiVersion: v1
items:
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: pruner
    namespace: openshift-image-registry
- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: openshift-image-registry-pruner
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: system:image-pruner
  subjects:
  - kind: ServiceAccount
    name: pruner
    namespace: openshift-image-registry
- apiVersion: batch/v1beta1
  kind: CronJob
  metadata:
    name: image-pruner
    namespace: openshift-image-registry
  spec:
    schedule: "0 0 * * *"
    concurrencyPolicy: Forbid
    successfulJobsHistoryLimit: 1
    failedJobsHistoryLimit: 3
    jobTemplate:
      spec:
        template:
          spec:
            restartPolicy: OnFailure
            containers:
            - image: "quay.io/openshift/origin-cli:4.1"
              resources:
                requests:
                  cpu: 1
                  memory: 1Gi
              terminationMessagePolicy: FallbackToLogsOnError
              command:
              - oc
              args:
```

```
        - adm
        - prune
        - images
        - --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
        - --keep-tag-revisions=5
        - --keep-younger-than=96h
        - --confirm=true
        name: image-pruner
      serviceAccountName: pruner
```

- **oc adm prune images [<options>]** 명령을 실행합니다.

```
$ oc adm prune images [<options>]
```

--prune-registry=false를 사용하지 않는 한 이미지를 정리하면 통합 레지스트리에서 데이터가 제거됩니다.

--namespace 플래그를 사용하여 이미지를 정리하면 이미지는 제거되지 않고 스트림만 제거됩니다. 이미지는 네임스페이스가 아닌 리소스입니다. 따라서 특정 네임스페이스로 정리를 제한하면 현재 사용량을 계산할 수 없습니다.

기본적으로 통합 레지스트리는 스토리지에 대한 요청 수를 줄이고 요청 처리 속도를 높이기 위해 Blob의 메타데이터를 캐시합니다. 정리를 수행해도 통합 레지스트리 캐시는 업데이트되지 않습니다. 캐시에 메타데이터가 있는 정리된 계층은 내보낼 수 없기 때문에 정리 후에도 정리된 계층이 포함된 이미지는 손상됩니다. 따라서 정리 후 캐시를 지우려면 레지스트리를 재배포해야 합니다.

```
$ oc rollout restart deployment/image-registry -n openshift-image-registry
```

통합 레지스트리에서 Redis 캐시를 사용하는 경우 데이터베이스를 수동으로 정리해야 합니다.

정리 후 레지스트리를 재배포하는 것이 옵션이 아닌 경우 캐시를 영구적으로 비활성화해야 합니다.

oc adm prune images 작업에는 레지스트리 경로가 필요합니다. 레지스트리 경로는 기본적으로 생성되지 않습니다.

이미지 CLI 구성 옵션 정리 테이블에는 **oc adm prune images <options>** 명령과 함께 사용할 수 있는 옵션이 설명되어 있습니다.

표 10.4. 이미지 CLI 구성 옵션 정리

옵션	설명
--all	레지스트리로 내보내지 않았지만 pullthrough에 의해 미러링된 이미지를 포함합니다. 기본적으로 활성화되어 있습니다. 통합 레지스트리로 내보낸 이미지로 정리를 제한하려면 **--all=false**를 전달합니다.
--certificate-authority	OpenShift Container Platform 관리 레지스트리와 통신할 때 사용할 인증 기관 파일의 경로입니다. 기본값은 현재 사용자 구성 파일의 인증 기관 데이터입니다. 제공되는 경우 보안 연결이 시작됩니다.

옵션	설명
--confirm	테스트를 실행하는 대신 정리를 수행해야 함을 나타냅니다. 이를 위해서는 통합 컨테이너 이미지 레지스트리에 대한 올바른 경로가 필요합니다. 이 명령을 클러스터 네트워크 외부에서 실행하는 경우 **--registry-url**을 사용하여 경로를 제공해야 합니다.
--force-insecure	이 옵션에는 주의가 필요합니다. HTTP를 통해 호스팅되거나 잘못된 HTTPS 인증서가 있는 컨테이너 레지스트리에 비보안 연결을 허용합니다.
--keep-tag-revisions=<N>	이미지 스트림별 이미지 리버전을 태그당 최대 **N**개(기본값 **3**)로 유지합니다.
--keep-younger-than=<duration>	현재 시간을 기준으로 **<duration>** 이내의 이미지를 정리하지 않습니다. 또는 현재 시간을 기준으로 **<duration>**(기본값 **60**분) 이내의 다른 오브젝트에서 참조하는 이미지를 정리하지 않습니다.
--prune-over-size-limit	동일한 프로젝트에 정의된 최소 제한을 초과하는 모든 이미지를 정리합니다. 이 플래그는 **--keep-tag-revisions** 또는 **--keep-younger-than**과 결합할 수 없습니다.
--registry-url	레지스트리에 연결할 때 사용하는 주소입니다. 이 명령에서는 관리되는 이미지 및 이미지 스트림으로 결정되는 클러스터 내부 URL을 사용하려고 합니다. 실패하는 경우(레지스트리를 해석하거나 연결할 수 없음) 이 플래그를 사용하여 작동하는 대체 경로를 제공해야 합니다. 레지스트리 호스트 이름은 특정 연결 프로토콜을 적용하는 **https://** 또는 **http://** 접두사로 지정할 수 있습니다.
--prune-registry	이 옵션은 다른 옵션에서 지정하는 조건과 함께 OpenShift Container Platform Image API 오브젝트에 해당하는 레지스트리 데이터를 정리할지 여부를 제어합니다. 기본적으로 이미지 정리 작업은 Image API 오브젝트와 레지스트리의 해당 데이터를 모두 처리합니다. 이 옵션은 이미지 오브젝트 수를 줄이기 위해 etcd 콘텐츠만 제거하고 레지스트리 스토리지는 정리하지 않으려는 경우 또는 레지스트리의 적절한 유지보수 기간 동안 레지스트리를 하드 정리하여 이미지 정리 작업을 별도로 수행하려는 경우 유용합니다.

10.6.1. 이미지 정리 조건

수동으로 정리한 이미지에 조건을 적용할 수 있습니다.

- OpenShift Container Platform에서 관리하는 이미지 또는 주석 **openshift.io/image.managed**가 있는 이미지를 제거하려면 다음을 수행합니다.
    - 최소 **--keep-younger-than**분 이내에 생성했으며 다음 중 어느 것도 현재 이 이미지를 참조하지 않습니다.

        - **--keep-younger-than**분 이내에 생성한 Pod
        - **--keep-younger-than**분 이내에 생성한 이미지 스트림
        - 실행 중인 Pod
        - 보류 중인 Pod
        - 복제 컨트롤러
        - 배포
        - 배포 구성
        - 복제본 세트
        - 빌드 구성
        - 빌드
        - **stream.status.tags[].items**의 **--keep-tag-revisions** 최근 항목
    - 동일한 프로젝트에 정의된 최소 제한을 초과하고 현재 다음 항목에서 참조하지 않습니다.
        - 실행 중인 Pod
        - 보류 중인 Pod
        - 복제 컨트롤러
        - 배포
        - 배포 구성
        - 복제본 세트
        - 빌드 구성
        - 빌드

- 외부 레지스트리에서 정리를 지원하지 않습니다.
- 이미지를 정리하면 **status.tags**에 이미지에 대한 참조가 있는 모든 이미지 스트림에서 이미지에 대한 참조가 모두 제거됩니다.
- 이미지에서 더 이상 참조하지 않는 이미지 계층이 제거됩니다.

참고

--prune-over-size-limit 플래그는 **--keep-tag-revisions** 플래그 또는 **--keep-younger-than** 플래그와 결합할 수 없습니다. 해당 플래그를 결합하면 이 작업을 수행할 수 없다는 정보가 반환됩니다.

--prune-registry=false를 사용하여 OpenShift Container Platform Image API 오브젝트와 레지스트리의 이미지 데이터를 제거하는 작업을 분리한 후 레지스트리를 하드 정리하면 타이밍 기간이 줄어들고 하나의 명령을 사용하여 정리할 때보다 더 안전합니다. 그러나 시차가 완전히 제거되지는 않습니다.

예를 들어 정리 작업에서 정리할 이미지를 확인하는 동안 이미지를 참조하는 Pod를 계속 생성할 수 있습니다. 삭제된 콘텐츠에 대한 참조를 줄일 수 있도록 정리 작업 중 생성되어 해당 이미지를 참조할 수 있는 API 오브젝트를 계속 추적해야 합니다.

--prune-registry 옵션을 사용하지 않거나 **--prune-registry=true**옵션을 사용하여 다시 정리하면 이전에 **--prune-registry=false**를 사용하여 정리한 이미지의 이미지 레지스트리와 연결된 스토리지가 정리되지 않습니다. **--prune-registry=false**를 사용하여 정리한 이미지는 레지스트리 하드 정리를 통해서만 레지스트리 스토리지에서 삭제할 수 있습니다.

10.6.2. 이미지 정리 작업 실행

프로세스

1. 정리 작업에서 삭제하는 항목을 확인하려면 다음을 수행합니다.
   a. 태그 리버전을 3개까지 유지하고 60분 이내의 리소스(이미지, 이미지 스트림, Pod)를 유지합니다.

      ```
      $ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
      ```

   b. 정의된 제한을 초과하는 모든 이미지를 정리합니다.

      ```
      $ oc adm prune images --prune-over-size-limit
      ```

2. 위 단계의 옵션을 사용하여 정리 작업을 수행하려면 다음 단계를 수행합니다.

   ```
   $ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm
   ```

   ```
   $ oc adm prune images --prune-over-size-limit --confirm
   ```

10.6.3. 보안 또는 비보안 연결 사용

보안 연결은 선호되고 권장되는 접근 방식입니다. 필수 인증서 확인과 함께 HTTPS 프로토콜을 통해 수행됩니다. **prune** 명령에서는 가능한 경우 항상 필수 인증서 확인을 사용하려고 합니다. 불가능한 경우 일부 사례에서 비보안 연결로 대체할 수 있는데 이는 위험합니다. 이 경우 인증서 확인을 건너뛰거나 일반 HTTP 프로토콜을 사용합니다.

다음의 경우 **--certificate-authority**를 지정하지 않는 한 비보안 연결로 대체할 수 있습니다.

1. **prune** 명령이 **--force-insecure** 옵션과 함께 실행됩니다.
2. 제공된 **registry-url**이 **http://** 스키마로 시작됩니다.
3. 제공된 **registry-url**이 로컬 링크 주소이거나 **localhost**입니다.
4. 현재 사용자의 구성에서 비보안 연결을 허용합니다. 이는 사용자가 **--insecure-skip-tls-verify**를 사용하여 로그인하거나 메시지가 표시되었을 때 비보안 연결을 선택했기 때문에 발생할 수 있습니다.

중요

OpenShift Container Platform에서 사용하는 인증 기관과 다른 인증 기관에서 레지스트리를 보호하는 경우 **--certificate-authority** 플래그를 사용하여 지정해야 합니다. 그러지 않으면 **prune** 명령이 오류와 함께 실패합니다.

10.6.4. 이미지 정리 문제

이미지가 정리되지 않음
이미지가 계속 누적되고 **prune** 명령을 실행해도 예상한 이미지의 일부만 제거되는 경우 이미지를 정리 후보로 간주하기 위해 적용해야 하는 이미지 정리 조건을 이해하고 있는지 확인하십시오.

제거할 이미지가 각 태그 내역에서 선택한 태그 리버전 임계값보다 위에 있어야 합니다. 예를 들어 **sha:abz**라는 오래되고 더 이상 사용되지 않는 이미지를 고려해 보십시오. 이미지에 태그가 지정된 네임스페이스 **N**에서 다음 명령을 실행하면 **myapp**이라는 단일 이미지 스트림에서 이미지에 태그가 세 번 지정됩니다.

```
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range $ti, $tag := $is.status.tags}}'\
  '{{range $ii, $item := $tag.items}}{{if eq $item.image """sha:abz"'\
  $""}}{{$is.metadata.name}}:{{$tag.tag}} at position {{$ii}} out of {{len $tag.items}}\n'\
  '{{end}}{{end}}{{end}}{{end}}'
```

출력 예

```
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

기본 옵션을 사용하면 **myapp:v2.1-may-2016** 태그 기록의 위치 **0**에서 이미지 정리가 수행되지 않으므로 이미지가 정리되지 않습니다. 정리할 이미지에 대해 관리자는 다음 중 하나를 수행해야 합니다.

- **oc adm prune images** 명령을 사용하여 **--keep-tag-revisions=0**을 지정합니다.

주의

이 작업에서는 태그가 지정된 임계값 미만이거나 지정된 임계값 미만의 오브젝트에서 참조하는 경우를 제외하고 기본 이미지가 있는 모든 네임스페이스에서 태그를 모두 제거합니다.

- 위치가 리버전 임계값 미만인 **istags**를 모두 삭제합니다. 즉 **myapp:v2.1** 및 **myapp:v2.1-may-2016**을 삭제합니다.
- 새 빌드를 동일한 **istag**로 내보내거나 다른 이미지에 태그를 지정하여 내역에서 이미지를 더 이동합니다. 이전 릴리스 태그에는 이 작업이 적절하지 않을 수 있습니다.

이미지를 정의되지 않은 시간 동안 보존해야 하는 경우를 제외하고 이름에 특정 이미지 빌드의 날짜 또는 시간이 포함된 태그는 사용하지 않도록 합니다. 이러한 태그는 내역에 하나의 이미지만 있어 정리되지 않습니다.

비보안 레지스트리에 대한 보안 연결 사용

oc adm prune images 명령 출력에 다음과 유사한 메시지가 표시되면 레지스트리가 안전하지 않은 것이며 **oc adm prune images** 클라이언트에서 보안 연결을 사용하려고 합니다.

```
error: error communicating with registry: Get https://172.30.30.30:5000/healthz: http: server gave HTTP response to HTTPS client
```

- 해결책은 레지스트리를 보호하는 것이 좋습니다. 또는 명령에 **--force-insecure**를 추가하여 클라이언트에서 비보안 연결을 사용하도록 할 수 있지만 이 방법은 권장되지 않습니다.

보안 레지스트리에 대해 비보안 연결 사용
oc adm prune images 명령 출력에 다음 오류 중 하나가 표시되면 연결 확인을 위해 **oc adm prune images** 클라이언트에서 사용하는 인증 기관이 아닌 다른 인증 기관에서 서명한 인증서를 사용하여 레지스트리를 보호하고 있음을 나타냅니다.

```
error: error communicating with registry: Get http://172.30.30.30:5000/healthz: malformed HTTP response "\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get https://172.30.30.30:5000/healthz: x509: certificate signed by unknown authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response "\x15\x03\x01\x00\x02\x02"]
```

기본적으로 사용자의 구성 파일에 저장된 인증 기관 데이터가 사용됩니다. Master API와의 통신도 마찬가지입니다.

--certificate-authority 옵션을 사용하여 컨테이너 이미지 레지스트리 서버에 대한 올바른 인증 기관을 제공합니다.

잘못된 인증 기관 사용
다음 오류는 보안 컨테이너 이미지 레지스트리의 인증서에 서명하는 데 사용하는 인증 기관이 클라이언트에서 사용하는 기관과 다르다는 것을 나타냅니다.

```
error: error communicating with registry: Get https://172.30.30.30:5000/: x509: certificate signed by unknown authority
```

--certificate-authority 플래그를 사용하여 올바른 정보를 제공하십시오.

해결 방법으로 **--force-insecure** 플래그를 대신 추가할 수도 있습니다. 하지만 이 방법은 권장되지 않습니다.

추가 리소스

- 레지스트리 액세스
- 레지스트리 공개
- 레지스트리 경로를 생성하는 방법에 대한 자세한 내용은 OpenShift Container Platform의 Image Registry Operator를 참조하십시오.

10.7. 레지스트리 하드 정리

OpenShift Container Registry는 OpenShift Container Platform 클러스터의 etcd에서 참조하지 않는 Blob을 누적할 수 있습니다. 따라서 기본 이미지 정리 절차를 수행할 수 없습니다. 이를 *고립된 Blob*이라고 합니다.

고립된 Blob은 다음과 같은 시나리오에서 발생할 수 있습니다.

- etcd의 이미지만 제거하고 레지스트리 스토리지의 이미지는 제거하지 않는 **oc delete image <sha256:image-id>** 명령을 사용하여 이미지를 수동으로 삭제합니다.
- 데몬 오류로 인해 시작된 레지스트리로 내보내서 일부 Blob은 업로드되지만 (최종 구성 요소로 업로드되는) 이미지 매니페스트는 업로드되지 않습니다. 모든 고유 이미지 Blob이 고립됩니다.
- OpenShift Container Platform에서 할당량 제한 때문에 이미지를 거부합니다.
- 표준 이미지 정리기에서 이미지 매니페스트를 삭제하지만 관련 Blob을 삭제하기 전에 중단됩니다.
- 의도한 Blob을 제거하지 못하는 레지스트리 정리기 버그로 인해 해당 Blob을 참조하는 이미지 오브젝트가 제거되고 Blob이 고립됩니다.

기본 이미지 정리와 별도의 절차인 레지스트리 *하드 정리*를 수행하면 클러스터 관리자가 고립된 Blob을 제거할 수 있습니다. OpenShift Container Registry의 스토리지 공간이 부족하고 고립된 Blob이 있다고 생각되면 하드 정리를 수행해야 합니다.

이 작업은 드물게 수행해야 하며 상당한 수의 고립이 새롭게 생성되었다는 증거가 있는 경우에만 필요합니다. 이러한 경우가 아니라면 생성되는 이미지 수에 따라 하루에 한 번과 같이 일정한 간격으로 표준 이미지를 정리하면 됩니다.

프로세스

레지스트리에서 고립된 Blob을 하드 정리하려면 다음을 수행합니다.

1. 로그인합니다.
   CLI를 사용하여 **kubeadmin** 또는 **openshift-image-registry** 네임스페이스에 액세스할 수 있는 권한이 있는 다른 사용자로 클러스터에 로그인합니다.
2. 기본 이미지 정리를 실행합니다.
   기본 이미지 정리에서는 더 이상 필요하지 않은 추가 이미지를 제거합니다. 하드 정리는 이미지를 자체적으로 제거하지 않습니다. 레지스트리 스토리지에 저장된 Blob만 제거합니다. 따라서 하드 정리 전에 기본 이미지 정리를 실행해야 합니다.
3. 레지스트리를 읽기 전용 모드로 전환합니다.
   레지스트리가 읽기 전용 모드로 실행되지 않는 경우 정리 작업과 동시에 발생하는 모든 내보내기 작업에서 다음 중 하나를 수행합니다.
   - 실패 및 새로운 고립 발생
   - 이미지를 가져올 수 없지만 성공(참조된 Blob 중 일부가 삭제되었기 때문에)

   내보내기 작업은 레지스트리가 다시 읽기-쓰기 모드로 전환될 때까지 실패합니다. 따라서 하드 정리는 신중하게 계획해야 합니다.

   레지스트리를 읽기 전용 모드로 전환하려면 다음을 수행합니다.

   a. **configs.imageregistry.operator.openshift.io/cluster**에서 **spec.readOnly**를 **true**로 설정합니다.

   ```
   $ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec":
   {"readOnly":true}}' --type=merge
   ```

4. **system:image-pruner** 역할을 추가합니다.
   일부 리소스를 나열하는 데에는 레지스트리 인스턴스를 실행하는 데 사용하는 서비스 계정에 추가 권한이 필요합니다.

a. 서비스 계정 이름을 가져옵니다.

```
$ service_account=$(oc get -n openshift-image-registry \
    -o jsonpath='{.spec.template.spec.serviceAccountName}' deploy/image-registry)
```

b. **system:image-pruner** 클러스터 역할을 서비스 계정에 추가합니다.

```
$ oc adm policy add-cluster-role-to-user \
    system:image-pruner -z \
    ${service_account} -n openshift-image-registry
```

5. 선택 사항: 시험 실행 모드에서 정리기를 실행합니다.
제거되는 Blob 수를 보려면 시험 실행 모드에서 하드 정리기를 실행합니다. 실제로는 변경되지 않습니다. 다음 예제에서는 **image-registry-3-vhndw**라는 이미지 레지스트리 Pod를 참조합니다.

```
$ oc -n openshift-image-registry exec pod/image-registry-3-vhndw -- /bin/sh -c
'/usr/bin/dockerregistry -prune=check'
```

또는 정리 후보에 대한 정확한 경로를 가져오도록 로깅 수준을 높입니다.

```
$ oc -n openshift-image-registry exec pod/image-registry-3-vhndw -- /bin/sh -c
'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -prune=check'
```

출력 예

```
time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune (dry-run mode)"
distribution_version="v2.4.1+unknown" kubernetes_version=v1.6.1+$Format:%h$
openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663fa5"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete blob:
sha256:0022d49612807cb348cabc562c072ef34d756adfe0100a61952cbcb87ee6578a"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a37c"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7663"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeeec207e4fad2a5bafd28afbef57e"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5e06"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data
```

6. 하드 정리를 실행합니다.

image-registry Pod에서 실행 중인 하나의 인스턴스 내에서 다음 명령을 실행하여 하드 정리를 실행합니다. 다음 예제에서는 **image-registry-3-vhndw**라는 이미지 레지스트리 Pod를 참조합니다.

```
$ oc -n openshift-image-registry exec pod/image-registry-3-vhndw -- /bin/sh -c
'/usr/bin/dockerregistry -prune=delete'
```

출력 예

```
Deleted 13374 blobs
Freed up 2.835 GiB of disk space
```

7. 레지스트리를 다시 읽기-쓰기 모드로 전환합니다.
정리가 완료되면 레지스트리를 다시 읽기-쓰기 모드로 전환할 수 있습니다.
configs.imageregistry.operator.openshift.io/cluster에서 **spec.readOnly**를 **false**로 설정합니다.

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec":{"readOnly":false}}' -
-type=merge
```

10.8. CRON 작업 정리

cron 작업으로 작업을 정리할 수 있지만 실패한 작업은 제대로 처리하지 못할 수 있습니다. 따라서 클러스터 관리자가 직접 정기적으로 작업을 정리해야 합니다. 또한 cron 작업에 대한 액세스 권한을 신뢰할 수 있는 소수의 사용자 그룹으로 제한하고 cron 작업에서 너무 많은 작업과 Pod를 생성하지 않도록 적절한 할당량을 설정해야 합니다.

추가 리소스

- 작업을 사용하여 Pod에서 작업 실행
- 다중 프로젝트의 리소스 할당량
- RBAC를 사용하여 권한 정의 및 적용

11장. RED HAT MARKETPLACE 사용

Red Hat Marketplace는 퍼블릭 클라우드 및 온프레미스 환경에서 실행되는 컨테이너 기반 환경에 대해 인증된 소프트웨어를 쉽게 검색하고 해당 소프트웨어에 액세스할 수 있는 오픈 클라우드 마켓플레이스입니다.

11.1. RED HAT MARKETPLACE의 기능

클러스터 관리자는 Red Hat Marketplace를 사용하여 OpenShift Container Platform에서 소프트웨어를 관리하고, 개발자에게 애플리케이션 인스턴스를 배포할 셀프 서비스 액세스 권한을 제공하며, 애플리케이션 사용량과 할당량을 관련시킵니다.

11.1.1. Marketplace에 OpenShift Container Platform 클러스터 연결

클러스터 관리자는 Marketplace에 연결된 OpenShift Container Platform 클러스터에 공통 애플리케이션 세트를 설치할 수 있습니다. 또한 Marketplace를 사용하여 서브스크립션 또는 할당량 대비 클러스터 사용량을 추적할 수 있습니다. Marketplace를 사용하여 추가하는 사용자는 제품 사용량이 추적되고 해당 조직에 요금이 청구됩니다.

클러스터 연결 프로세스 중 이미지 레지스트리 시크릿을 업데이트하고 카탈로그를 관리하며 애플리케이션 사용량을 보고하는 Marketplace Operator가 설치됩니다.

11.1.2. 애플리케이션 설치

클러스터 관리자는 OpenShift Container Platform의 OperatorHub 내에서 또는 Marketplace 웹 애플리케이션에서 Marketplace 애플리케이션을 설치 할 수 있습니다.

웹 콘솔에서 **Operator** > 설치된 **Operator**를 클릭하여 설치된 애플리케이션에 액세스할 수 있습니다.

11.1.3. 다른 화면에서 애플리케이션 배포

웹 콘솔의 관리자 및 개발자 화면에서 Marketplace 애플리케이션을 배포할 수 있습니다.

개발자 화면
개발자는 개발자 화면을 사용하여 새로 설치된 기능에 액세스할 수 있습니다.

예를 들어 데이터베이스 Operator가 설치되면 개발자가 프로젝트 내의 카탈로그에서 인스턴스를 생성할 수 있습니다. 데이터베이스 사용량이 집계되어 클러스터 관리자에게 보고됩니다.

이 화면에는 Operator 설치 및 애플리케이션 사용량 추적이 포함되지 않습니다.

관리자 화면
클러스터 관리자는 관리자 화면에서 Operator 설치 및 애플리케이션 사용량 정보에 액세스할 수 있습니다.

설치된 **Operator** 목록에서 CRD(사용자 정의 리소스 정의)를 검색하여 애플리케이션 인스턴스를 시작할 수도 있습니다.