OpenShift Container Platform 4.12

이미지

OpenShift Container Platform에서 이미지와 이미지 스트림 생성 및 관리

Last Updated: 2024-10-16

OpenShift Container Platform 4.12 이미지

OpenShift Container Platform에서 이미지와 이미지 스트림 생성 및 관리

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 이미지와 이미지 스트림을 생성하고 관리하는 데 필요한 지침을 제공합니다. 템플릿 사용에 대한 지침도 제공합니다.

차례

1장. 이미지 개요

1.1. 컨테이너, 이미지 및 이미지 스트림 이해

컨테이너화된 소프트웨어 생성 및 관리를 시작하려는 경우 컨테이너, 이미지 및 이미지 스트림이라는 개념을 이해하는 것이 중요합니다. 이미지에는 실행할 준비가 된 소프트웨어 모음이 들어 있으며 컨테이너는 컨테이너 이미지의 실행 중인 인스턴스입니다. 이미지 스트림은 동일한 기본 이미지의 다양한 버전을 저장하는 방법을 제공합니다. 이러한 다양한 버전은 동일한 이미지 이름에 다양한 태그로 나타냅니다.

1.2. 이미지

OpenShift Container Platform의 컨테이너는 OCI 또는 Docker 형식의 컨테이너 *이미지*를 기반으로 합니다. 이미지는 단일 컨테이너를 실행하는 데 필요한 모든 요구 사항과 필요성 및 기능에 대해 설명하는 메타데이터가 포함되어 있는 바이너리입니다.

이미지는 패키징 기술로 생각할 수 있습니다. 컨테이너를 생성할 때 컨테이너에 추가 액세스 권한을 부여하지 않는 경우 컨테이너는 이미지에 정의된 리소스에만 액세스할 수 있습니다. OpenShift Container Platform은 여러 호스트의 여러 컨테이너에 동일한 이미지를 배포하고 컨테이너 간에 부하를 분산하여 서비스 패키지 이미지에 중복성과 수평 확장을 제공할 수 있습니다.

사용자가 podman 또는 **docker** CLI를 사용하여 직접 이미지를 빌드할 수 있으나, 기존 이미지에 코드 또는 구성을 추가하여 새 이미지를 생성할 수 있는 빌더 이미지를 OpenShift Container Platform에서 제공하기도 합니다.

시간이 지남에 따라 여러 애플리케이션이 개발되므로 단일 이미지 이름이 실제로는 동일한 이미지의 여러 다양한 버전을 참조하는 경우가 있습니다. 이미지의 다양한 각 버전은 해당 해시 (**fd44297e2ddb050ec4f...**과 같은 긴 16진수)로 고유하게 참조되며, 일반적으로 **fd44297e2ddb**와 같이 12자로 단축됩니다.

컨테이너 이미지를 생성,관리 및 사용할 수 있습니다.

1.3. 이미지 레지스트리

이미지 레지스트리는 컨테이너 이미지를 저장하고 제공할 수 있는 콘텐츠 서버입니다. 예를 들면 다음과 같습니다.

```
registry.redhat.io
```

레지스트리에는 하나 이상의 태그된 이미지가 포함된 하나 이상의 이미지 리포지터리 컬렉션이 포함되어 있습니다. Red Hat은 **registry.redhat.io**의 레지스트리를 구독자에게 제공합니다. OpenShift Container Platform은 사용자 정의 컨테이너 이미지를 관리하기 위해 자체 OpenShift 이미지 레지스트리를 제공할 수도 있습니다.

1.4. 이미지 리포지터리

이미지 리포지터리는 관련 컨테이너 이미지와 이러한 이미지를 식별하는 태그의 컬렉션입니다. 예를 들어 OpenShift Container Platform Jenkins 이미지는 다음 리포지터리에 있습니다.

```
docker.io/openshift/jenkins-2-centos7
```

1.5. 이미지 태그

이미지 태그는 리포지터리의 컨테이너 이미지에 적용되는 레이블로, 이미지 스트림에서 특정 이미지를 다른 이미지와 구별하는 역할을 합니다. 일반적으로 태그는 일종의 버전 번호를 나타냅니다. 예를 들어 다음에서는 **:v3.11.59-2**가 태그입니다.

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

이미지에 태그를 더 추가할 수 있습니다. 예를 들어 이미지에 **:v3.11.59-2** 및 **:latest** 태그가 할당될 수 있습니다.

OpenShift Container Platform은 **docker tag** 명령과 유사하지만 이미지가 아닌 이미지 스트림에서 작동하는 **oc tag** 명령을 제공합니다.

1.6. 이미지 ID

이미지 ID는 이미지를 가져오는 데 사용할 수 있는 SHA(Secure Hash Algorithm) 코드입니다. SHA 이미지 ID는 변경할 수 없습니다. 특정 SHA 식별자는 항상 정확히 동일한 컨테이너 이미지 콘텐츠를 참조합니다. 예를 들면 다음과 같습니다.

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

1.7. 컨테이너

OpenShift Container Platform 애플리케이션의 기본 단위는 컨테이너라고 합니다. Linux 컨테이너 기술은 실행 중인 프로세스를 격리하여 지정된 리소스와만 상호 작용하도록 제한하기 위한 간단한 메커니즘입니다. 컨테이너라는 단어는 컨테이너 이미지의 실행 중이거나 정지된 특정 인스턴스로 정의됩니다.

많은 애플리케이션 인스턴스는 서로의 프로세스, 파일, 네트워크 등을 보지 않으며 단일 호스트의 컨테이너에서 실행될 수 있습니다. 컨테이너를 임의의 워크로드에 사용할 수 있기는 하지만 일반적으로 각 컨테이너는 웹 서버나 데이터베이스 같은 마이크로 서비스라고 하는 단일 서비스를 제공하는 경우가 많습니다.

Linux 커널은 수년간 컨테이너 기술을 위한 기능을 통합해 왔습니다. Docker 프로젝트에서는 호스트의 Linux 컨테이너를 위한 편리한 관리 인터페이스를 개발했습니다. 최근에는 Open Container Initiative 에서 컨테이너 형식 및 컨테이너 런타임에 대한 오픈 표준을 개발했습니다. OpenShift Container Platform 및 Kubernetes는 다중 호스트 설치에서 OCI 및 Docker 형식 컨테이너를 오케스트레이션하는 기능을 추가합니다.

OpenShift Container Platform을 사용할 때 컨테이너 런타임과 직접 상호 작용하지는 않지만 OpenShift Container Platform에서의 역할과 애플리케이션이 컨테이너 내에서 작동하는 방식을 이해하려면 기능 및 용어를 이해하는 것이 중요합니다.

podman 같은 도구는 컨테이너를 직접 실행하고 관리하는 데 사용되는 **docker** 명령줄 도구를 교체할 수 있습니다. **podman**을 사용하면 OpenShift Container Platform과 별도로 컨테이너를 시험할 수 있습니다.

1.8. 이미지 스트림을 사용하는 이유

이미지 스트림 및 관련 태그는 OpenShift Container Platform 내에서 컨테이너 이미지를 참조하는 데 필요한 추상을 제공합니다. 이미지 스트림 및 해당 태그를 사용하면 사용 가능한 이미지를 볼 수 있으며 리포지터리의 이미지가 변경되어도 필요한 특정 이미지를 사용하도록 할 수 있습니다.

이미지 스트림에는 실제 이미지 데이터가 포함되어 있지 않지만 이미지 리포지터리와 유사하게 관련 이미지에 대한 단일 가상 뷰가 있습니다.

새 이미지가 추가되는 경우 이미지 스트림의 알림을 보고 빌드 또는 배포를 각각 수행하여 대응하도록 빌드 및 배포를 구성할 수 있습니다.

예를 들어 배포에서 특정 이미지를 사용하고 있는데 해당 이미지의 새 버전이 생성되는 경우 배포가 자동으로 수행되어 새 버전의 이미지를 가져올 수 있습니다.

하지만 배포 또는 빌드에서 사용하는 이미지 스트림 태그가 업데이트되지 않으면 컨테이너 이미지 레지스트리의 컨테이너 이미지가 업데이트되어도 빌드 또는 배포에서는 알려진 정상 이미지인 이전 이미지를 계속 사용합니다.

소스 이미지를 저장할 수 있는 위치는 다음과 같습니다.

- OpenShift Container Platform 통합 레지스트리
- 외부 레지스트리(예: registry.redhat.io 또는 quay.io)
- OpenShift Container Platform 클러스터의 다른 이미지 스트림

이미지 스트림 태그를 참조하는 오브젝트(예: 빌드 또는 배포 구성)를 정의할 때 리포지터리가 아닌 이미지 스트림 태그를 가리킵니다. 애플리케이션을 빌드하거나 배포할 때 OpenShift Container Platform은 이 이미지 스트림 태그로 리포지터리를 조회하여 관련된 이미지 ID를 찾은 후 바로 그 이미지를 사용합니다.

이미지 스트림 메타데이터는 다른 클러스터 정보와 함께 etcd 인스턴스에 저장됩니다.

이미지 스트림을 사용하면 다음과 같은 여러 중요한 이점이 있습니다.

- 명령줄을 사용하여 다시 푸시하지 않고도 태그하고, 태그를 롤백하고, 이미지를 빠르게 처리할 수 있습니다.
- 새 이미지가 레지스트리로 푸시되면 빌드 및 배포를 트리거할 수 있습니다. OpenShift Container Platform에는 Kubernetes 오브젝트 등 다른 리소스에 대한 일반 트리거도 있습니다.
- 정기적인 다시 가져오기를 위해 태그를 표시할 수 있습니다. 소스 이미지가 변경되면 해당 변경 사항이 이미지 스트림에 반영되고, 이후 빌드 또는 배포 구성에 따라 빌드 및/또는 배포 흐름이 트리거됩니다.
- 세분화된 액세스 제어를 사용하여 이미지를 공유하고 팀 전체에 이미지를 빠르게 배포할 수 있습니다.
- 소스 이미지가 변경되면 이미지 스트림 태그는 여전히 알려진 양호한 버전의 이미지를 가리키므로 애플리케이션이 예기치 않게 손상되지 않도록 합니다.
- 이미지 스트림 오브젝트에 대한 권한을 통해 이미지를 보고 사용할 수 있는 사람에 대한 보안을 구성할 수 있습니다.
- 클러스터 수준에서 이미지를 읽거나 나열할 수 있는 권한이 없는 사용자도 이미지 스트림을 사용하여 프로젝트의 태그된 이미지를 검색할 수 있습니다.

이미지 스트림을 관리하고, Kubernetes 리소스가 포함된 이미지 스트림을 사용 하고, 이미지 스트림 업데이트에서 업데이트를 트리거할 수 있습니다.

1.9. 이미지 스트림 태그

이미지 스트림 태그는 이미지 스트림의 이미지에 대한 이름 지정된 포인터입니다. 이미지 스트림 태그는 컨테이너 이미지 태그와 유사합니다.

1.10. 이미지 스트림 이미지

이미지 스트림 이미지를 사용하면 태그된 특정 이미지 스트림에서 특정 컨테이너 이미지를 검색할 수 있습니다. 이미지 스트림 이미지는 특정 이미지 SHA 식별자에 대한 일부 메타데이터를 함께 가져오는 API 리소스 오브젝트입니다.

1.11. 이미지 스트림 트리거

이미지 스트림 태그가 변경되면 이미지 스트림 트리거가 특정 작업이 발생하도록 합니다. 예를 들어 가져오기로 인해 태그 값이 변경되고 이어서 해당 태그를 수신하는 배포, 빌드 또는 기타 리소스에서 트리거가 실행될 수 있습니다.

1.12. CLUSTER SAMPLES OPERATOR 사용 방법

초기 시작 중에 Operator는 기본 샘플 리소스를 생성하여 이미지 스트림 및 템플릿 생성을 시작합니다. Cluster Samples Operator를 사용하여 **openshift** 네임스페이스에 저장된 샘플 이미지 스트림 및 템플릿을 관리할 수 있습니다.

클러스터 관리자는 Cluster Samples Operator를 사용하여 다음을 수행할 수 있습니다.

- Operator를 구성합니다.
- 대체 레지스트리와 함께 Operator를 사용합니다.

1.13. 템플릿 정보

템플릿은 복제할 오브젝트 정의입니다. 템플릿을 사용하여 구성을 빌드하고 배포할 수 있습니다.

1.14. RUBY ON RAILS 사용 방법

개발자는 Ruby on Rails 를 사용하여 다음을 수행할 수 있습니다.

- 애플리케이션을 작성합니다.
  - 데이터베이스를 설정합니다.
  - 시작 페이지를 생성합니다.
  - OpenShift Container Platform에 대한 애플리케이션을 구성합니다.
  - Git에 애플리케이션을 저장합니다.
- OpenShift Container Platform에서 애플리케이션을 배포합니다.
  - 데이터베이스 서비스를 생성합니다.
  - 프런트 엔드 서비스를 생성합니다.
  - 애플리케이션의 경로를 생성합니다.

2장. CLUSTER SAMPLES OPERATOR 구성

openshift 네임스페이스에서 작동하는 Cluster Samples Operator는 RHEL(Red Hat Enterprise Linux) 기반 OpenShift Container Platform 이미지 스트림 및 OpenShift Container Platform 템플릿을 설치하고 업데이트합니다.

2.1. CLUSTER SAMPLES OPERATOR 이해

설치 중에 Operator는 기본 구성 오브젝트를 생성한 후 샘플 이미지 스트림 및 빠른 시작 템플릿을 비롯한 템플릿을 생성합니다.

참고

인증 정보가 필요한 다른 레지스트리에서 이미지 스트림을 원활하게 가져오기 위해, 클러스터 관리자는 이미지 가져오기에 필요한 **openshift** 네임스페이스에서 Docker **config.json** 파일의 콘텐츠가 포함된 추가 시크릿을 생성할 수 있습니다.

Cluster Samples Operator 구성은 클러스터 전체 리소스이며 배포는 **openshift-cluster-samples-operator** 네임스페이스에 들어 있습니다.

Cluster Samples Operator 이미지에는 관련 OpenShift Container Platform 릴리스의 이미지 스트림 및 템플릿 정의가 포함되어 있습니다. 각 샘플이 생성되거나 업데이트되면 Cluster Samples Operator에 OpenShift Container Platform 버전을 나타내는 주석이 포함됩니다. Operator는 이 주석을 사용하여 각 샘플이 릴리스 버전과 일치하는지 확인합니다. 인벤토리 외부 샘플은 건너뛰기한 샘플과 마찬가지로 무시됩니다. Operator가 관리하는 샘플에서 버전 주석을 수정하거나 삭제해야 하는 수정 사항은 자동으로 취소됩니다.

참고

Jenkins 이미지는 설치의 이미지 페이로드에 포함되어 있으며 이미지 스트림에 직접 태그됩니다.

Cluster Samples Operator 구성 리소스에는 삭제 시 다음 항목을 정리하는 종료자가 포함되어 있습니다.

- Operator가 관리하는 이미지 스트림
- Operator가 관리하는 템플릿
- Operator가 생성한 구성 리소스
- 클러스터 상태 리소스

샘플 리소스를 삭제하면 Cluster Samples Operator에서 기본 구성을 사용하여 리소스를 다시 생성합니다.

2.1.1. Cluster Samples Operator의 관리 상태 사용

Cluster Samples Operator는 기본적으로 또는 글로벌 프록시가 구성된 경우 **Managed** 상태로 부트 스트랩됩니다. **Managed** 상태에서는 Cluster Samples Operator가 레지스트리에서 샘플 이미지 스트림과 이미지를 가져오고 필수 샘플 템플릿이 설치되었는지 확인하기 위해 적극적으로 리소스를 관리하며 구성 요소를 활성 상태로 유지합니다.

Cluster Samples Operator가 **Removed**으로 부트 스트랩되는 상황은 다음과 같습니다.

- 새로 설치 후 초기 시작에서 3분이 경과해도 Cluster Samples Operator가 registry.redhat.io에 접속할 수 없는 경우
- Cluster Samples Operator가 IPv6 네트워크에 있음을 탐지하는 경우
- 이미지 컨트롤러 구성 매개변수 가 기본 이미지 레지스트리를 사용하거나 **samplesRegistry** 설정에서 지정한 이미지 레지스트리를 사용하여 이미지 스트림을 생성하지 못하도록 합니다.

참고

OpenShift Container Platform의 경우 기본 이미지 레지스트리는 **registry.redhat.io** 입니다.

그러나 Cluster Samples Operator가 IPv6 네트워크에 있고 OpenShift Container Platform 글로벌 프록시가 구성되어 있음을 감지하면 IPv6 검사가 모든 검사를 대체합니다. 결과적으로 Cluster Samples Operator가 **Removed** 로 부트 스트랩됩니다.

중요

IPv6 설치는 현재 registry.redhat.io에서 지원되지 않습니다. Cluster Samples Operator는 대부분의 샘플 이미지 스트림과 이미지를 registry.redhat.io에서 가져옵니다.

2.1.1.1. 제한된 네트워크 설치

registry.redhat.io에 액세스할 수 없을 때 **Removed**로 부트 스트랩하면 네트워크 제한이 이미 있는 경우 제한된 네트워크 설치를 원활하게 수행할 수 있습니다. 네트워크 액세스가 제한되어 있을 때 **Removed** 상태로 부트 스트랩하면 클러스터 관리자가 샘플이 필요한지 판단할 시간이 더 늘어납니다. 관리 상태가 **Removed**로 설정되면 Cluster Samples Operator가 샘플 이미지 스트림 가져오기에 실패했다는 경고를 제출하지 않기 때문입니다. Cluster Samples Operator가 **Managed**로 표시될 때 샘플 이미지 스트림을 설치하려고 하는 경우 실패한 가져오기가 있으면 초기 설치 후 2시간이 경과했을 때 경고가 시작됩니다.

2.1.1.2. 초기 네트워크 액세스를 통한 제한된 네트워크 설치

반대로, 네트워크 액세스가 가능한 동안 제한된 네트워크 또는 연결 해제된 클러스터로 설정할 클러스터가 먼저 설치되는 경우 Cluster Samples Operator는 **registry.redhat.io**의 콘텐츠에 액세스할 수 있기 때문에 해당 콘텐츠를 설치합니다. 필요한 샘플을 결정하고 이미지 미러를 설정하는 등의 작업을 수행할 때까지 샘플 설치를 미루도록 Cluster Samples Operator를 **Removed** 상태로 부트 스트랩하려면 대체 레지스트리를 통한 Samples Operator 사용 및 노드 사용자 정의 지침에 따릅니다. 둘 다 추가 리소스 섹션에 연결되어 있으며 Cluster Samples Operator 기본 구성을 재정의하여 처음에는 **Removed**로 표시되도록 합니다.

다음 추가 YAML 파일을 **openshift-install create manifest**를 통해 생성된 **openshift** 디렉토리에 지정해야 합니다.

managementState: Removed 상태의 Cluster Samples Operator YAML 파일 예

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  architectures:
  - x86_64
  managementState: Removed
```

2.1.2. Cluster Samples Operator의 이미지 스트림 가져오기 추적 및 오류 복구

샘플 이미지 스트림을 생성하거나 업데이트한 후 Cluster Samples Operator는 각 이미지 스트림 태그의 이미지 가져오기 진행 상황을 모니터링합니다.

가져오기에 실패하거나, Cluster Samples Operator 구성이 변경되어 해당 이미지 스트림이 **skippedImagestreams** 목록에 추가되거나, 관리 상태가 **Removed**로 변경되는 경우 Cluster Samples Operator는 **oc import-image** 명령에서 사용하는 것과 동일한 이미지 스트림 이미지 가져오기 API를 통해 가져오기가 성공할 때까지 약 15분마다 가져오기를 다시 시도합니다.

추가 리소스

- 설치 중에 Cluster Samples Operator를 제거하는 경우 대체 레지스트리와 함께 Cluster Samples Operator를 사용하여 콘텐츠를 가져온 다음 Cluster Samples Operator를 **Managed** 로 설정하여 샘플을 가져올 수 있습니다.
- 초기 네트워크 액세스가 포함된 제한된 네트워크 설치에서 필요한 샘플을 결정할 때까지 샘플 설치를 미루도록 Cluster Samples Operator가 **Removed** 상태로 부트 스트랩되도록 하려면 노드 사용자 정의 지침에 따라 Cluster Samples Operator 기본 구성을 재정의하고 처음에는 **Removed** 로 표시되도록 합니다.
  - 연결이 끊긴 환경의 샘플을 호스팅하려면 대체 레지스트리가 있는 Cluster Samples Operator 사용 지침을 따르십시오.

2.1.3. 미러링을 위한 Cluster Samples Operator 지원

설치 프로세스 중에 OpenShift Container Platform은 **openshift-cluster-samples-operator** 네임스페이스에 **imagestreamtag-to-image**라는 구성 맵을 생성합니다. **imagestreamtag-to-image** 구성 맵에는 각 이미지 스트림 태그에 대한 이미지 채우기 항목이 포함되어 있습니다.

구성 맵의 데이터 필드에 있는 각 항목의 키 형식은 **<image_stream_name>_<image_stream_tag_name>**입니다.

OpenShift Container Platform의 연결이 끊긴 설치 프로세스 중에 Cluster Samples Operator의 상태가 **Removed**로 설정됩니다. **Managed**로 변경하려면 샘플이 설치됩니다.

참고

네트워크 제한 또는 중단된 환경에서 샘플을 사용하려면 네트워크 외부의 서비스에 액세스해야 할 수 있습니다. 일부 예제 서비스에는 GitHub, Maven Central, npm, RubyGems, PyPi 등이 있습니다. 클러스터 샘플 Operator의 오브젝트가 필요한 서비스에 도달할 수 있도록 하는 추가 단계가 있을 수 있습니다.

이 구성 맵을 사용하여 이미지 스트림을 가져오려면 이미지를 미러링해야 하는 이미지 참조로 사용할 수 있습니다.

- Cluster Samples Operator가 **Removed**로 설정된 경우 미러링된 레지스트리를 생성하거나 사용할 기존 미러링된 레지스트리를 확인할 수 있습니다.
- 새 구성 맵을 가이드로 사용하여 미러링된 레지스트리에 샘플을 미러링합니다.
- Cluster Samples Operator 구성 개체의 **skippedImagestreams** 필드에 미러링되지 않은 이미지 스트림을 추가합니다.
- Cluster Samples Operator 구성 개체의 **samplesRegistry** 를 미러링된 레지스트리로 설정합니다.

- 그런 다음 Cluster Samples Operator를 **Managed**로 설정하여 미러링된 이미지 스트림을 설치합니다.

자세한 내용은 대체 레지스트리 또는 미러링된 레지스트리에서 Cluster Samples Operator 이미지 스트림 사용을 참조하십시오.

2.2. CLUSTER SAMPLES OPERATOR 구성 매개변수

샘플 리소스에서는 다음 구성 필드를 제공합니다.

매개변수	설명
managementState	**Managed**: Cluster Samples Operator가 구성에 지시된 대로 샘플을 업데이트합니다. **Unmanaged**: Cluster Samples Operator가 구성 리소스 오브젝트와**openshift** 네임스페이스에 있는 이미지 스트림 또는 템플릿에 대한 업데이트를 무시합니다. **Removed**: Cluster Samples Operator가**openshift** 네임스페이스에 있는 **Managed** 이미지 스트림 및 템플릿 세트를 제거합니다. 클러스터 관리자가 생성한 새 샘플이나 건너뛰기한 목록의 샘플을 무시합니다. 제거가 완료되면 Cluster Samples Operator는 **Unmanaged** 상태인 것처럼 작동하며 샘플 리소스, 이미지 스트림 또는 템플릿에 대한 감시 이벤트를 무시합니다.
samplesRegistry	이미지 콘텐츠에 대해 이미지 스트림이 액세스하는 레지스트리를 지정할 수 있습니다. **samplesRegistry의** 기본값은 OpenShift Container Platform의 경우 **registry.redhat.io**입니다. **참고** **Samples Registry**가 명시적으로 설정되지 않았거나 빈 문자열을 남겼거나 registry.redhat.io로 설정된 경우 가져오기 액세스에 대한 시크릿이 없으면 RHEL 콘텐츠의 생성 또는 업데이트가 시작되지 않습니다. 두 경우 모두 인증 정보가 필요한 registry.redhat.io 외부에서 이미지 가져오기를 수행합니다. **Samples Registry**가 빈 문자열 또는 registry.redhat.io 이외의 값으로 재정의되면 RHEL 컨텐츠의 생성 또는 업데이트가 풀 시크릿의 존재 여부에 따라 제어되지 않습니다.
architectures	아키텍처 유형을 선택하는 자리 표시자입니다.
skippedImagestreams	Cluster Samples Operator 인벤토리에 있지만 클러스터 관리자가 Operator에서 무시하거나 관리하지 않도록 하려는 이미지 스트림입니다. 이미지 스트림 이름 목록을 이 매개변수에 추가할 수 있습니다. 예를 들어 **["httpd","perl"]**가 있습니다.
skippedTemplates	Cluster Samples Operator 인벤토리에 있지만 클러스터 관리자가 Operator에서 무시하거나 관리하지 않도록 하려는 템플릿입니다.

초기 샘플 리소스 오브젝트가 생성되기 전에 시크릿, 이미지 스트림 및 템플릿 감시 이벤트가 추가될 수 있으며 Cluster Samples Operator는 그러한 이벤트를 탐지하여 다시 큐에 지정합니다.

2.2.1. 구성 제한 사항

Cluster Samples Operator에서 여러 아키텍처 지원을 시작하면 **Managed** 상태에서는 아키텍처 목록을 변경할 수 없습니다.

아키텍처 값을 변경하려면 클러스터 관리자가 다음 작업을 수행해야 합니다.

- **Management State**를 **Removed**로 표시하고 해당 변경 사항을 저장합니다.
- 후속 변경에서 아키텍처를 편집하고 **Management State**를 **Managed**로 다시 변경합니다.

Cluster Samples Operator는 **Removed** 상태에서도 여전히 시크릿을 처리합니다. **Removed** 로 전환하기 전에, **Managed**로 전환하기 전에 **Removed** 또는 **Managed** 상태로 전환한 후 시크릿을 생성할 수 있습니다. **Managed**로 전환한 후 시크릿을 생성하면 시크릿 이벤트가 처리될 때까지 샘플 생성이 지연됩니다. 이 경우 전환하기 전에 모든 샘플을 제거하도록 선택하여 레지스트리 변경을 원활하게 수행할 수 있습니다. 전환 전에 모든 샘플을 제거할 필요는 없습니다.

2.2.2. 조건

샘플 리소스는 다음 조건을 해당 상태에서 유지보수합니다.

Condition	설명
SamplesExists	**openshift** 네임스페이스에서 샘플이 생성되었음을 나타냅니다.
ImageChangesInProgress	이미지 스트림이 생성 또는 업데이트되었으나 일부 태그 사양 생성 및 태그 상태 생성이 일치하지 않는 경우 **True**입니다. 모든 생성이 일치하거나 가져오기 중에 복구할 수 없는 오류가 발생한 경우 **False**입니다. 마지막으로 표시된 오류는 메시지 필드에 있습니다. 보류 중인 이미지 스트림 목록은 이유 필드에 있습니다. 이 조건은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다.
ConfigurationValid	이전에 언급한 제한된 변경 사항이 제출되었는지 여부에 따라 **True** 또는 **False**입니다.
RemovePending	**Management State: Removed** 설정이 보류 중이나 Cluster Samples Operator에서 삭제가 완료되기를 기다리는 중임을 나타냅니다.
ImportImageErrorsExist	해당 태그 중 하나에 대한 이미지 가져오기 단계에서 이미지 스트림에 오류가 발생했는지를 나타냅니다. 오류가 발생한 경우 **True**입니다. 오류가 있는 이미지 스트림 목록은 이유 필드에 있습니다. 보고된 각 오류에 대한 세부 정보는 메시지 필드에 있습니다.
MigrationInProgress	Cluster Samples Operator에서 현재 샘플 세트가 설치된 버전이 Cluster Samples Operator 버전과 다름을 탐지하는 경우 **True**입니다. 이 조건은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다.

2.3. CLUSTER SAMPLES OPERATOR 구성에 액세스

제공된 매개변수로 파일을 편집하여 Cluster Samples Operator를 구성할 수 있습니다.

전제 조건

- OpenShift CLI(**oc**)를 설치합니다.

절차

- Cluster Samples Operator 구성에 액세스합니다.

```
$ oc edit configs.samples.operator.openshift.io/cluster -o yaml
```

Cluster Samples Operator 구성은 다음 예와 유사합니다.

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
# ...
```

2.4. CLUSTER SAMPLES OPERATOR에서 더 이상 사용되지 않는 이미지 스트림 태그 제거

Cluster Samples Operator는 더 이상 사용되지 않는 이미지 스트림 태그를 사용하는 배포가 있을 수 있으므로 이미지 스트림에 더 이상 사용되지 않는 이미지 스트림 태그를 남겨 둡니다.

oc tag 명령을 사용하여 이미지 스트림을 편집하여 더 이상 사용되지 않는 이미지 스트림 태그를 제거할 수 있습니다.

참고

샘플 공급자가 이미지 스트림에서 제거된 이미지 스트림 태그는 초기 설치에 포함되어 있지 않습니다.

사전 요구 사항

- **oc** CLI를 설치했습니다.

절차

- **oc tag** 명령을 사용하여 이미지 스트림을 편집하여 더 이상 사용되지 않는 이미지 스트림 태그를 제거합니다.

```
$ oc tag -d <image_stream_name:tag>
```

출력 예

```
Deleted tag default/<image_stream_name:tag>.
```

추가 리소스

- 인증 정보 구성에 대한 자세한 내용은 이미지 풀 시크릿 사용을 참조하십시오.

3장. 대체 레지스트리를 통한 CLUSTER SAMPLES OPERATOR 사용

먼저 미러 레지스트리를 생성하여 대체 레지스트리를 통해 Cluster Samples Operator를 사용할 수 있습니다.

중요

필요한 컨테이너 이미지를 얻으려면 인터넷에 액세스해야 합니다. 이 절차에서는 네트워크와 인터넷에 모두 액세스할 수 있는 미러 호스트에 미러 레지스트리를 배치합니다.

3.1. 미러 레지스트리 정보

OpenShift Container Platform 설치 및 후속 제품 업데이트에 Red Hat Quay, JFrog Artifactory, Sonatype Nexus Repository 또는 Harbor와 같은 컨테이너 미러 레지스트리에 필요한 이미지를 미러링할 수 있습니다. 대규모 컨테이너 레지스트리에 액세스할 수 없는 경우 OpenShift Container Platform 서브스크립션에 포함된 소규모 컨테이너 레지스트리인 *Red Hat OpenShift에 미러 레지스트리*를 사용할 수 있습니다.

Red Hat Quay, the *mirror registry for Red Hat OpenShift*, Artifactory, Sonatype Nexus Repository, 또는 Harbor와 같이 Docker v2-2를 지원하는 컨테이너 레지스트리를 사용할 수 있습니다. 선택한 레지스트리에 관계없이 인터넷상의 Red Hat 호스팅 사이트의 콘텐츠를 격리된 이미지 레지스트리로 미러링하는 절차는 동일합니다. 콘텐츠를 미러링한 후 미러 레지스트리에서 이 콘텐츠를 검색하도록 각 클러스터를 설정합니다.

중요

OpenShift 이미지 레지스트리는 미러링 프로세스 중에 필요한 태그 없이 푸시를 지원하지 않으므로 대상 레지스트리로 사용할 수 없습니다.

*Red Hat OpenShift의 미러 레지스트리*가 아닌 컨테이너 레지스트리를 선택하는 경우 프로비저닝하는 클러스터의 모든 시스템에서 액세스할 수 있어야 합니다. 레지스트리에 연결할 수 없는 경우 설치, 업데이트 또는 워크로드 재배치와 같은 일반 작업이 실패할 수 있습니다. 따라서 고가용성 방식으로 미러 레지스트리를 실행해야하며 미러 레지스트리는 최소한 OpenShift Container Platform 클러스터의 프로덕션 환경의 가용성조건에 일치해야 합니다.

미러 레지스트리를 OpenShift Container Platform 이미지로 채우면 다음 두 가지 시나리오를 수행할 수 있습니다. 호스트가 인터넷과 미러 레지스트리에 모두 액세스할 수 있지만 클러스터 노드에 액세스 할 수 없는 경우 해당 머신의 콘텐츠를 직접 미러링할 수 있습니다. 이 프로세스를 *connected mirroring*(미러링 연결)이라고 합니다. 그러한 호스트가 없는 경우 이미지를 파일 시스템에 미러링한 다음 해당 호스트 또는 이동식 미디어를 제한된 환경에 배치해야 합니다. 이 프로세스를 *미러링 연결 해제*라고 합니다.

미러링된 레지스트리의 경우 가져온 이미지의 소스를 보려면 CRI-O 로그의 **Trying to access** 로그 항목을 검토해야 합니다. 노드에서 **crictl images** 명령을 사용하는 등의 이미지 가져오기 소스를 보는 다른 방법은 미러링되지 않은 이미지 이름을 표시합니다.

참고

Red Hat은 OpenShift Container Platform에서 타사 레지스트리를 테스트하지 않습니다.

추가 정보

이미지 소스를 보기 위해 CRI-O 로그를 보는 방법에 대한 자세한 내용은 이미지 가져오기 소스 보기를 참조하십시오.

3.1.1. 미러 호스트 준비

미러 레지스트리를 생성하려면 먼저 미러 호스트를 준비해야 합니다.

3.1.2. 바이너리를 다운로드하여 OpenShift CLI 설치

명령줄 인터페이스를 사용하여 OpenShift Container Platform과 상호 작용하기 위해 OpenShift CLI(**oc**)를 설치할 수 있습니다. Linux, Windows 또는 macOS에 **oc**를 설치할 수 있습니다.

중요

이전 버전의 **oc**를 설치한 경우, OpenShift Container Platform 4.12의 모든 명령을 완료하는 데 해당 버전을 사용할 수 없습니다. 새 버전의 **oc**를 다운로드하여 설치합니다.

Linux에서 OpenShift CLI 설치
다음 절차를 사용하여 Linux에서 OpenShift CLI(**oc**) 바이너리를 설치할 수 있습니다.

프로세스

1. Red Hat 고객 포털에서 OpenShift Container Platform 다운로드 페이지 로 이동합니다.
2. **제품 변형** 드롭다운 목록에서 아키텍처를 선택합니다.
3. **버전** 드롭다운 목록에서 적절한 버전을 선택합니다.
4. **OpenShift v4.12 Linux Client** 항목 옆에 있는 **지금 다운로드**를 클릭하고 파일을 저장합니다.
5. 아카이브의 압축을 풉니다.

```
$ tar xvf <file>
```

6. **oc** 바이너리를 **PATH**에 있는 디렉터리에 배치합니다.
   PATH를 확인하려면 다음 명령을 실행합니다.

```
$ echo $PATH
```

검증

- OpenShift CLI를 설치한 후 **oc** 명령을 사용할 수 있습니다.

```
$ oc <command>
```

Windows에서 OpenSfhit CLI 설치
다음 절차에 따라 Windows에 OpenShift CLI(**oc**) 바이너리를 설치할 수 있습니다.

프로세스

1. Red Hat 고객 포털에서 OpenShift Container Platform 다운로드 페이지 로 이동합니다.
2. **버전** 드롭다운 목록에서 적절한 버전을 선택합니다.

3. **OpenShift v4.12 Windows Client** 항목 옆에 있는 **지금 다운로드**를 클릭하고 파일을 저장합니다.
4. ZIP 프로그램으로 아카이브의 압축을 풉니다.
5. **oc** 바이너리를 **PATH**에 있는 디렉터리로 이동합니다.
   PATH를 확인하려면 명령 프롬프트를 열고 다음 명령을 실행합니다.

```
C:\> path
```

검증

- OpenShift CLI를 설치한 후 **oc** 명령을 사용할 수 있습니다.

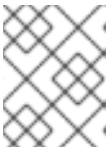
```
C:\> oc <command>
```

macOS에 OpenShift CLI 설치

다음 절차에 따라 macOS에서 OpenShift CLI(**oc**) 바이너리를 설치할 수 있습니다.

프로세스

1. Red Hat 고객 포털에서 OpenShift Container Platform 다운로드 페이지 로 이동합니다.
2. **버전** 드롭다운 목록에서 적절한 버전을 선택합니다.
3. **OpenShift v4.12 macOS Client** 항목 옆에 있는 **지금 다운로드**를 클릭하고 파일을 저장합니다.



참고

macOS ARM64의 경우 **OpenShift v4.12 macOS ARM64 Client** 항목을 선택합니다.

4. 아카이브의 압축을 해제하고 압축을 풉니다.
5. **oc** 바이너리 PATH의 디렉터리로 이동합니다.
   PATH를 확인하려면 터미널을 열고 다음 명령을 실행합니다.

```
$ echo $PATH
```

검증

- OpenShift CLI를 설치한 후 **oc** 명령을 사용할 수 있습니다.

```
$ oc <command>
```

3.2. 이미지를 미러링할 수 있는 인증 정보 설정

Red Hat에서 미러로 이미지를 미러링할 수 있는 컨테이너 이미지 레지스트리 인증 정보 파일을 생성합니다.

사전 요구 사항

- 연결이 끊긴 환경에서 사용할 미러 레지스트리를 구성했습니다.

절차

설치 호스트에서 다음 단계를 수행합니다.

1. Red Hat OpenShift Cluster Manager에서 **registry.redhat.io** 풀 시크릿을 다운로드합니다.
2. 풀 시크릿을 JSON 형식으로 복사합니다.

```
$ cat ./pull-secret | jq . > <path>/<pull_secret_file_in_json> (1)
```

(1) 풀 시크릿을 저장할 폴더의 경로와 생성한 JSON 파일의 이름을 지정합니다.

파일의 내용은 다음 예와 유사합니다.

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3. 미러 레지스트리에 대한 base64로 인코딩된 사용자 이름 및 암호 또는 토큰을 생성합니다.

```
$ echo -n '<user_name>:<password>' | base64 -w0 (1)
BGVtbYk3ZHAtqXs=
```

(1) **<user_name>** 및 **<password>**의 경우 레지스트리에 설정한 사용자 이름 및 암호를 지정합니다.

4. JSON 파일을 편집하고 레지스트리를 설명하는 섹션을 추가합니다.

```
  "auths": {
    "<mirror_registry>": { (1)
      "auth": "<credentials>", (2)
      "email": "you@example.com"
    }
  },
```

1. **<mirror_registry>**의 경우 미러 레지스트리가 콘텐츠를 제공하는데 사용하는 레지스트리 도메인 이름 및 포트 (선택 사항)를 지정합니다. 예: **registry.example.com** 또는

2. **<credentials>**의 경우 미러 레지스트리에 대해 base64로 인코딩된 사용자 이름 및 암호를 지정합니다.

파일은 다음 예제와 유사합니다.

```
{
  "auths": {
    "registry.example.com": {
      "auth": "BGVtbYk3ZHAtqXs=",
      "email": "you@example.com"
    },
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3.3. OPENSHIFT CONTAINER PLATFORM 이미지 저장소 미러링

클러스터 설치 또는 업그레이드 중에 사용할 OpenShift Container Platform 이미지 저장소를 레지스트리에 미러링합니다.

사전 요구 사항

- 미러 호스트가 인터넷에 액세스할 수 있습니다.
- 네트워크가 제한된 환경에서 사용할 미러 레지스트리를 설정하고 설정한 인증서 및 인증 정보에 액세스할 수 있습니다.
- Red Hat OpenShift Cluster Manager에서 풀 시크릿 을 다운로드하여 미러 저장소에 대한 인증을 포함하도록 수정했습니다.
- 자체 서명된 인증서를 사용하는 경우 인증서에 주체 대체 이름을 지정했습니다.

절차

미러 호스트에서 다음 단계를 완료합니다.

1. OpenShift Container Platform 다운로드 페이지 를 확인하여 설치할 OpenShift Container Platform 버전을 확인하고 Repository Tags 페이지에서 해당 태그를 지정합니다.

2. 필요한 환경 변수를 설정합니다.

   a. 릴리스 버전을 내보냅니다.

   ```
   $ OCP_RELEASE=<release_version>
   ```

   <release_version>에 대해 설치할 OpenShift Container Platform 버전에 해당하는 태그를 지정합니다 (예: **4.5.4**).

   b. 로컬 레지스트리 이름 및 호스트 포트를 내보냅니다.

   ```
   $ LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>'
   ```

   <local_registry_host_name>의 경우 미러 저장소의 레지스트리 도메인 이름을 지정하고 **<local_registry_host_port>**의 경우 콘텐츠를 제공하는데 사용되는 포트를 지정합니다.

   c. 로컬 저장소 이름을 내보냅니다.

   ```
   $ LOCAL_REPOSITORY='<local_repository_name>'
   ```

   <local_repository_name>의 경우 레지스트리에 작성할 저장소 이름 (예: **ocp4/openshift4**)을 지정합니다.

   d. 미러링할 저장소 이름을 내보냅니다.

   ```
   $ PRODUCT_REPO='openshift-release-dev'
   ```

   프로덕션 환경의 릴리스의 경우 **openshift-release-dev**를 지정해야 합니다.

   e. 레지스트리 풀 시크릿의 경로를 내보냅니다.

   ```
   $ LOCAL_SECRET_JSON='<path_to_pull_secret>'
   ```

   생성한 미러 레지스트리에 대한 풀 시크릿의 절대 경로 및 파일 이름을 **<path_to_pull_secret>**에 지정합니다.

   f. 릴리스 미러를 내보냅니다.

   ```
   $ RELEASE_NAME="ocp-release"
   ```

   프로덕션 환경의 릴리스의 경우 **ocp-release**를 지정해야 합니다.

   g. 서버의 아키텍처 유형 (예: **x86_64** 또는 **aarch** 64)을 내보냅니다.

   ```
   $ ARCHITECTURE=<server_architecture>
   ```

   h. 미러링된 이미지를 호스트할 디렉터리의 경로를 내보냅니다.

   ```
   $ REMOVABLE_MEDIA_PATH=<path> (1)
   ```

   (1) 초기 슬래시 (/) 문자를 포함하여 전체 경로를 지정합니다.

3. 미러 레지스트리에 버전 이미지를 미러링합니다.

- 미러 호스트가 인터넷에 액세스할 수 없는 경우 다음 작업을 수행합니다.

  i. 이동식 미디어를 인터넷에 연결된 시스템에 연결합니다.

  ii. 미러링할 이미지 및 설정 매니페스트를 확인합니다.

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE} --dry-run
```

  iii. 이전 명령의 출력에서 전체 **imageContentSources** 섹션을 기록합니다. 미러에 대한 정보는 미러링된 저장소에 고유하며 설치 중에 **imageContentSources** 섹션을 **install-config.yaml** 파일에 추가해야 합니다.

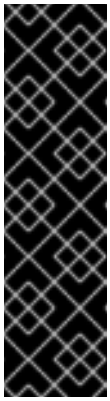
  iv. 이동식 미디어의 디렉터리에 이미지를 미러링합니다.

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} --to-
dir=${REMOVABLE_MEDIA_PATH}/mirror
quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE}
```

  v. 미디어를 네트워크가 제한된 환경으로 가져와서 이미지를 로컬 컨테이너 레지스트리에 업로드합니다.

```
$ oc image mirror -a ${LOCAL_SECRET_JSON} --from-
dir=${REMOVABLE_MEDIA_PATH}/mirror
"file://openshift/release:${OCP_RELEASE}*"
${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} (1)
```

  (1) **REMOVABLE_MEDIA_PATH**의 경우 이미지를 미러링 할 때 지정한 것과 동일한 경로를 사용해야 합니다.



중요

oc image mirror 를 실행하면 다음과 같은 오류가 발생할 수 있습니다. **error: unable to retrieve source image**. 이 오류는 이미지 인덱스에 이미지 레지스트리에 더 이상 존재하지 않는 이미지에 대한 참조가 포함된 경우 발생합니다. 이미지 인덱스에서는 이러한 이미지를 실행하는 사용자가 업그레이드 그래프의 최신 지점으로 업그레이드 경로를 실행할 수 있도록 이전 참조를 유지할 수 있습니다. 임시 해결 방법으로 **--skip-missing** 옵션을 사용하여 오류를 무시하고 이미지 인덱스를 계속 다운로드할 수 있습니다. 자세한 내용은 Service Mesh Operator 미러링 실패 에서 참조하십시오.

- 로컬 컨테이너 레지스트리가 미러 호스트에 연결된 경우 다음 작업을 수행합니다.

  i. 다음 명령을 사용하여 릴리스 이미지를 로컬 레지스트리에 직접 푸시합니다.

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}
```

이 명령은 요약된 릴리스 정보를 가져오며, 명령 출력에는 클러스터를 설치할 때 필요한 **imageContentSources** 데이터가 포함됩니다.

ii. 이전 명령의 출력에서 전체 **imageContentSources** 섹션을 기록합니다. 미러에 대한 정보는 미러링된 저장소에 고유하며 설치 중에 **imageContentSources** 섹션을 **install-config.yaml** 파일에 추가해야 합니다.

참고

미러링 프로세스 중에 이미지 이름이 Quay.io에 패치되고 podman 이미지는 부트스트랩 가상 머신의 레지스트리에 Quay.io를 표시합니다.

4. 미러링된 콘텐츠를 기반으로 설치 프로그램을 생성하려면 콘텐츠를 추출하여 릴리스 배포에 고정합니다.

- 미러 호스트가 인터넷에 액세스할 수 없는 경우 다음 명령을 실행합니다.

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --icsp-file=<file> \ --
command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}"
```

- 로컬 컨테이너 레지스트리가 미러 호스트에 연결된 경우 다음 명령을 실행합니다.

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}"
```

중요

선택한 OpenShift Container Platform 버전에 올바른 이미지를 사용하려면 미러링된 콘텐츠에서 설치 프로그램을 배포해야 합니다.

인터넷이 연결된 컴퓨터에서 이 단계를 수행해야 합니다.

5. 설치 프로그램에서 제공하는 인프라를 사용하는 클러스터의 경우 다음 명령을 실행합니다.

```
$ openshift-install
```

3.4. 대체 레지스트리 또는 미러링된 레지스트리에서 CLUSTER SAMPLES OPERATOR 이미지 스트림 사용

Cluster Samples Operator에 의해 관리되는 **openshift** 네임스페이스에 있는 대부분의 이미지 스트림은 registry.redhat.io의 Red Hat 레지스트리에 있는 이미지를 참조합니다.

참고

설치 페이로드의 일부인 **cli**, **installer**, **must-gather** 및 **test** 이미지 스트림은 Cluster Samples Operator가 관리하지 않습니다. 이러한 내용은 이 절차에서 다루지 않습니다.

중요

연결이 끊긴 환경에서 Cluster Samples Operator를 **Managed** 로 설정해야 합니다. 이미지 스트림을 설치하려면 미러링된 레지스트리가 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 미러 레지스트리의 풀 시크릿을 생성합니다.

프로세스

1. 미러링할 특정 이미지 스트림의 이미지에 액세스합니다. 예를 들면 다음과 같습니다.

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep registry.redhat.io
```

2. 필요한 이미지 스트림과 관련된 registry.redhat.io에서 이미지를 미러링합니다.

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-25-rhel7:latest
```

3. 클러스터의 이미지 구성 오브젝트를 생성합니다.

```
$ oc create configmap registry-config --from-file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. 클러스터의 이미지 설정 오브젝트에서 미러에 필요한 신뢰할 수 있는 CA를 추가합니다.

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
```

5. 미러 설정에 정의된 미러 위치의 **hostname** 부분을 포함하도록 Cluster Samples Operator 설정 오브젝트에서 **samplesRegistry** 필드를 업데이트합니다.

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

참고

현재 이미지 스트림 가져오기 프로세스에서 미러 또는 검색 메커니즘이 사용되지 않기 때문에 이 작업이 필요합니다.

6. Cluster Samples Operator 구성 오브젝트의 **skippedImagestreams** 필드에 미러링되지 않은 이미지 스트림을 추가합니다. 또는 샘플 이미지 스트림을 모두 지원할 필요가 없는 경우 Cluster Samples Operator 구성 오브젝트에서 Cluster Samples Operator를 **Removed** 로 설정합니다.

참고

이미지 스트림 가져오기가 실패했으나 Cluster Samples Operator가 주기적으로 재시도하거나 재시도하지 않는 것처럼 보이면 Cluster Samples Operator는 경고를 발행합니다.

openshift 네임스페이스의 여러 템플릿은 이미지 스트림을 참조합니다. 따라서 **Removed**를 사용하여 이미지 스트림과 템플릿을 모두 제거하면 누락된 이미지 스트림으로 인해 기능이 제대로 작동하지 않을 경우 템플릿을 사용할 가능성이 없어집니다.

3.4.1. 미러링을 위한 Cluster Samples Operator 지원

설치 프로세스 중에 OpenShift Container Platform은 **openshift-cluster-samples-operator** 네임스페이스에 **imagestreamtag-to-image**라는 구성 맵을 생성합니다. **imagestreamtag-to-image** 구성 맵에는 각 이미지 스트림 태그에 대한 이미지 채우기 항목이 포함되어 있습니다.

구성 맵의 데이터 필드에 있는 각 항목의 키 형식은 **<image_stream_name>_<image_stream_tag_name>**입니다.

OpenShift Container Platform의 연결이 끊긴 설치 프로세스 중에 Cluster Samples Operator의 상태가 **Removed**로 설정됩니다. **Managed**로 변경하려면 샘플이 설치됩니다.

참고

네트워크 제한 또는 중단된 환경에서 샘플을 사용하려면 네트워크 외부의 서비스에 액세스해야 할 수 있습니다. 일부 예제 서비스에는 GitHub, Maven Central, npm, RubyGems, PyPi 등이 있습니다. 클러스터 샘플 Operator의 오브젝트가 필요한 서비스에 도달할 수 있도록 하는 추가 단계가 있을 수 있습니다.

이 구성 맵을 사용하여 이미지 스트림을 가져오려면 이미지를 미러링해야 하는 이미지 참조로 사용할 수 있습니다.

- Cluster Samples Operator가 **Removed**로 설정된 경우 미러링된 레지스트리를 생성하거나 사용할 기존 미러링된 레지스트리를 확인할 수 있습니다.
- 새 구성 맵을 가이드로 사용하여 미러링된 레지스트리에 샘플을 미러링합니다.
- Cluster Samples Operator 구성 개체의 **skippedImagestreams** 필드에 미러링되지 않은 이미지 스트림을 추가합니다.
- Cluster Samples Operator 구성 개체의 **samplesRegistry** 를 미러링된 레지스트리로 설정합니다.
- 그런 다음 Cluster Samples Operator를 **Managed**로 설정하여 미러링된 이미지 스트림을 설치합니다.

자세한 내용은 대체 레지스트리 또는 미러링된 레지스트리에서 Cluster Samples Operator 이미지 스트림 사용을 참조하십시오.

4장. 이미지 생성

사용자를 지원하도록 준비되어 있는 미리 빌드된 이미지를 기반으로 자체 컨테이너 이미지를 생성하는 방법에 대해 알아봅니다. 이 프로세스에는 이미지를 작성하고, 이미지 메타데이터를 정의하고, 이미지를 테스트하고, 사용자 정의 빌더 워크플로를 통해 OpenShift Container Platform과 함께 사용할 이미지를 생성하는 작업에 대한 모범 사례 학습이 포함되어 있습니다. 이미지를 생성한 후 OpenShift 이미지 레지스트리로 푸시할 수 있습니다.

4.1. 컨테이너 모범 사례 학습

OpenShift Container Platform에서 실행할 컨테이너 이미지를 생성하는 경우 해당 이미지 사용자가 좋은 경험을 얻도록 하기 위해 이미지 작성자로서 고려해야 할 여러 모범 사례가 있습니다. 이미지는 변경 불가능하며 그대로 사용하도록 설정되어 있으므로 다음 지침에 따라 OpenShift Container Platform에서 이미지의 활용률과 사용 편의성을 높일 수 있습니다.

4.1.1. 일반 컨테이너 이미지 지침

다음 지침은 이미지가 OpenShift Container Platform에서 사용되는지 여부와 상관없이 일반적인 컨테이너 이미지를 생성하는 경우 적용됩니다.

이미지 재사용
가능한 경우 이미지는 **FROM** 문을 사용하는 적절한 업스트림 이미지를 기반으로 합니다. 이렇게 하면 이미지가 업데이트되는 경우 사용자가 직접 종속성을 업데이트할 필요 없이 이미지가 업스트림 이미지의 보안 수정 사항을 쉽게 찾을 수 있습니다.

또한 **FROM** 명령어에 태그 (예: **rhel:rhel7**)를 사용하여 해당 이미지의 기반이 되는 이미지 버전을 사용자에게 정확히 알려 주십시오. **latest** 이외의 태그를 사용하면 손상을 유발하는 변경사항이 이미지에 적용되어 **latest** 버전의 업스트림 이미지에 포함되는 일이 없습니다.

태그 내에서 호환성 유지보수
자체 이미지를 태그하는 경우 태그 내에서 이전 버전과의 호환성을 유지보수합니다. 예를 들어 **image** 라는 이미지를 제공하고 현재 버전 **1.0** 이 포함된 경우 **image:v1** 태그를 제공할 수 있습니다. 이미지를 업데이트하면 원래 이미지와 계속 호환되는 한 새 이미지 이미지 (**v1**) 및 이 태그의 다운스트림 소비자는 손상 없이 업데이트를 가져올 수 있습니다.

나중에 호환되지 않는 업데이트를 릴리스하는 경우 새 태그(예: **image:v2**)로 전환합니다. 이렇게 하면 다운스트림 사용자가 마음대로 새 버전으로 이동하면서도 호환되지 않는 새 이미지로 인해 의도치 않게 손상되는 일이 없게 할 수 있습니다. **image:latest** 를 사용하는 모든 다운스트림 사용자는 호환되지 않는 변경 사항이 도입될 위험이 있습니다.

여러 프로세스 방지
하나의 컨테이너에서 데이터베이스 및 **SSHD** 같은 서비스를 여러 개 시작하지 마십시오. 컨테이너는 간단하며 쉽게 서로 연결하여 여러 프로세스를 오케스트레이션할 수 있으므로 그렇게 할 필요가 없습니다. OpenShift Container Platform을 사용하면 관련 이미지를 단일 pod로 그룹화하여 쉽게 공동 배치하고 공동 관리할 수 있습니다.

컨테이너는 이러한 공동 배치를 통해 통신에 필요한 네트워크 네임스페이스 및 스토리지를 공유할 수 있습니다. 각 이미지를 덜 자주, 독립적으로 업데이트할 수 있으므로 업데이트에서도 중단이 덜 발생합니다. 생성된 프로세스에 대한 라우팅 신호를 관리할 필요가 없으므로 단일 프로세스를 사용하면 신호 처리 흐름이 더욱 명확해집니다.

래퍼 스크립트에 exec 사용
많은 이미지에서는 소프트웨어 실행을 위한 프로세스를 시작하기 전에 래퍼 스크립트를 사용하여 설정을 수행합니다. 이미지에서 이러한 스크립트를 사용하는 경우 해당 스크립트는 **exec**를 사용하여 소프트웨어가 스크립트 프로세스를 교체하도록 합니다. **exec**를 사용하지 않으면 컨테이너 런타임에서 보낸 신호가

소프트웨어 프로세스가 아닌 래퍼 스크립트로 갑니다. 이는 원하는 작동이 아닙니다.

어떤 서버의 프로세스를 시작하는 래퍼 스크립트가 있는 경우입니다. 예를 들어 **podman run -i**를 사용하여 컨테이너를 시작하여 래퍼 스크립트를 실행하면 프로세스가 시작됩니다. **CTRL+C** 를 사용하여 컨테이너를 종료하려면 다음을 수행합니다. 래퍼 스크립트에서 **exec**를 사용하여 서버 프로세스를 시작한 경우 **podman** 은 SIGINT를 서버 프로세스로 보내며 모든 작업이 예상대로 수행됩니다. 래퍼 스크립트에서 **exec**를 사용하지 않은 경우 **podman**은 래퍼 스크립트의 프로세스로 SIGINT를 보내며 프로세스는 아무 일도 없던 것처럼 계속 실행됩니다.

또한 프로세스는 컨테이너에서 실행되는 경우 **PID 1**로 실행됩니다. 즉, 기본 프로세스가 종료되면 전체 컨테이너가 중지되어 **PID 1** 프로세스에서 시작한 하위 프로세스가 모두 취소됩니다.

임시 파일 정리
빌드 프로세스 중에 생성한 임시 파일을 모두 제거합니다. 이러한 파일에는 **ADD** 명령으로 추가된 파일도 포함됩니다. 예를 들어 **yum install** 작업을 수행한 후에는 **yum clean** 명령을 실행합니다.

다음과 같이 **RUN** 문을 생성하면 **yum** 캐시가 이미지 계층에 생성되는 것을 방지할 수 있습니다.

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clean all -y
```

이 문을 다음과 같이 작성할 수도 있습니다.

```
RUN yum -y install mypackage
RUN yum -y install myotherpackage && yum clean all -y
```

그러면 첫 번째 **yum** 호출에서 해당 계층에 추가 파일을 남기는데 나중에 **yum clean** 작업을 실행해도 해당 파일을 제거할 수 없습니다. 추가 파일이 최종 이미지에 표시되지 않아도 기본 계층에 남아 있습니다.

현재 컨테이너 빌드 프로세스에서는 이전 계층에서 어떤 항목을 제거한 경우에도 이후 계층에서 명령을 실행하여 이미지가 사용하는 공간을 줄일 수 없습니다. 향후에는 변경될 수도 있습니다. 즉, 이후 계층에서 **rm** 명령을 수행하는 경우 파일이 숨겨져 있어도 다운로드할 이미지의 전체 크기가 줄어들지 않습니다. 따라서 **yum clean** 예에서처럼 가능한 경우 파일을 생성한 명령과 동일한 명령에서 파일을 제거하여 계층에 쓰이지 않도록 하는 것이 가장 좋습니다.

또한 단일 **RUN** 문에서 여러 명령을 수행하면 이미지의 계층 수가 줄어들어 다운로드 및 추출 시간이 단축됩니다.

적절한 순서로 명령어 배치
컨테이너 빌더에서는 **Dockerfile**을 읽고 맨 위부터 아래로 명령어를 실행합니다. 성공적으로 실행되는 모든 명령은 다음에 이 이미지 또는 다른 이미지가 빌드될 때 재사용할 수 있는 계층을 생성합니다. **Dockerfile**의 맨 위에는 거의 변경되지 않을 명령어를 배치하는 것이 매우 중요합니다. 이렇게 하면 상위 계층 변경에 의해 캐시가 무효화되지 않으므로 다음에 동일한 이미지를 매우 빠르게 빌드할 수 있습니다.

예를 들어 반복할 파일을 설치하는 **ADD** 명령과 패키지를 **yum install**하는 **RUN** 명령이 포함된 **Dockerfile**에서 작업하는 경우 다음과 같이 **ADD** 명령을 마지막에 배치하는 것이 가장 좋습니다.

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

이렇게 하면 **myfile**을 편집하고 **podman build** 또는 **docker build**를 다시 실행할 때마다 시스템은 **yum** 명령에는 캐시된 계층을 재사용하고 **ADD** 작업에서만 새 계층을 생성합니다.

그러지 않고 다음과 같이 **Dockerfile**을 작성할 수 있습니다.

```
FROM foo
ADD myfile /test/myfile
RUN yum -y install mypackage && yum clean all -y
```

그러면 **myfile**을 변경하고 **podman build** 또는 **docker build**를 다시 실행할 때마다 **ADD** 작업으로 **RUN** 계층 캐시가 무효화되어 **yum** 작업도 다시 실행해야 합니다.

중요한 포트 표시
EXPOSE 명령어는 컨테이너의 포트를 호스트 시스템 및 다른 컨테이너에서 사용할 수 있도록 합니다. **podman run** 호출을 통해 포트가 노출되도록 지정할 수 있으나 **Dockerfile**에 EXPOSE 명령어를 사용하면 소프트웨어를 실행하는 데 필요한 포트를 명시적으로 선언하여 사용자 및 소프트웨어 둘 다 더욱 쉽게 이미지를 사용할 수 있습니다.

- 노출된 포트는 이미지에서 생성된 컨테이너와 관련 있는 **podman ps** 아래에 표시됩니다.
- 노출된 포트는 **podman inspect**에서 반환된 이미지의 메타데이터에도 있습니다.
- 한 컨테이너를 다른 컨테이너에 연결하면 노출된 포트가 연결됩니다.

환경 변수 설정
ENV 명령어로 환경 변수를 설정하는 것이 좋습니다. 한 예로 프로젝트 버전 설정이 있습니다. 이렇게 하면 사용자가 **Dockerfile**을 보지 않고도 쉽게 버전을 찾을 수 있습니다. 또 다른 예로는 **JAVA_HOME** 같은 다른 프로세스에서 사용할 수 있는 시스템의 경로 알림이 있습니다.

기본 암호 설정 방지
기본 암호를 설정하지 않도록 합니다. 많은 사용자가 이미지를 확장한 후 기본 암호를 제거하거나 변경하는 것을 잊어 버립니다. 이렇게 되면 프로덕션 단계의 사용자에게 잘 알려진 암호가 할당되는 경우 보안 문제로 이어질 수 있습니다. 암호는 환경 변수를 사용하여 구성할 수 있습니다.

기본 암호를 설정하도록 선택하는 경우 컨테이너가 시작될 때 적절한 경고 메시지가 표시되도록 해야 합니다. 이 메시지에서는 사용자에게 기본 암호 값을 알려주고 설정할 환경 변수와 같은 암호 변경 방법을 설명해야 합니다.

sshd 실행 방지
이미지에서는 **sshd**를 실행하지 않는 것이 가장 좋습니다. **podman exec** 또는 **docker exec** 명령을 사용하여 로컬 호스트에서 실행 중인 컨테이너에 액세스할 수 있습니다. 또는 **oc exec** 명령 또는 **oc rsh** 명령을 사용하여 OpenShift Container Platform 클러스터에서 실행 중인 컨테이너에 액세스할 수 있습니다. 이미지에서 **sshd**를 설치하고 실행하면 추가적인 공격 벡터와 보안 패치 요구 사항이 발생합니다.

영구 데이터 볼륨 사용
이미지에서는 영구 데이터 볼륨을 사용합니다. 이렇게 하면 OpenShift Container Platform에서 컨테이너를 실행하는 노드에 네트워크 스토리지를 마운트하며 컨테이너가 새 노드로 이동하면 해당 노드에 이 스토리지가 다시 연결됩니다. 모든 영구 스토리지 요구에 이 볼륨을 사용하면 컨테이너를 다시 시작하거나 이동해도 콘텐츠가 보존됩니다. 이미지가 컨테이너 내 임의의 위치에 데이터를 쓰는 경우 해당 콘텐츠를 보존할 수 없습니다.

컨테이너가 삭제된 후에도 보존해야 하는 데이터는 모두 볼륨에 써야 합니다. 컨테이너 엔진은 컨테이너의 ephemeral 스토리지에 데이터를 쓰지 않는 모범 사례를 엄격하게 적용하기 위해 사용할 수 있는 **readonly** 플래그를 컨테이너에서 지원합니다. 지금 이 기능을 기반으로 이미지를 설계하면 나중에 더욱 쉽게 이미지를 활용할 수 있습니다.

Dockerfile에서 명시적으로 볼륨을 정의하면 이미지 사용자가 이미지를 실행할 때 정의해야 하는 볼륨을 쉽게 파악할 수 있습니다.

OpenShift Container Platform에서 볼륨을 사용하는 방법에 대한 자세한 내용은 Kubernetes 문서를 참조하십시오.

참고

영구 볼륨을 사용하더라도 이미지의 각 인스턴스에는 자체 볼륨이 있으며 파일 시스템은 인스턴스 간에 공유되지 않습니다. 즉, 볼륨은 클러스터에서 상태를 공유하는 데 사용할 수 없습니다.

4.1.2. OpenShift Container Platform 특정 지침

다음은 특별히 OpenShift Container Platform에서 사용할 컨테이너 이미지를 생성하는 경우 적용되는 지침입니다.

4.1.2.1. S2I(source-to-image)에 대해 이미지 활성화

타사에서 제공한 애플리케이션 코드를 실행하도록 하려는 이미지(예: 개발자가 제공한 Ruby 코드를 실행하도록 설계된 Ruby 이미지)의 경우 이미지를 활성화하여 S2I(Source-to-Image) 빌드 도구로 작업할 수 있습니다. S2I는 입력으로 애플리케이션 소스 코드를 사용하는 이미지를 쓰고 출력으로 어셈블된 애플리케이션을 실행하는 새 이미지를 생성하는 작업을 쉽게 수행할 수 있도록 하는 프레임워크입니다.

4.1.2.2. 임의의 사용자 ID 지원

기본적으로 OpenShift Container Platform에서는 임의로 할당된 사용자 ID를 사용하여 컨테이너를 실행합니다. 이렇게 하면 컨테이너 엔진 취약점으로 인해 컨테이너를 벗어나는 프로세스에 대해 추가적인 보안이 제공되므로 호스트 노드에서의 권한이 에스컬레이션됩니다.

이미지에서 임의의 사용자로 실행하도록 지원하려면 이미지의 프로세스에 의해 쓰일 수 있는 디렉토리와 파일을 루트 그룹에서 소유해야 하며 해당 그룹에서 읽을 수 있고 쓸 수 있어야 합니다. 실행될 파일에도 그룹 실행 권한이 있어야 합니다.

Dockerfile에 다음을 추가하면 루트 그룹의 사용자가 빌드 된 이미지의 디렉토리 및 파일에 액세스할 수 있도록 디렉토리 및 파일 권한이 설정됩니다.

```
RUN chgrp -R 0 /some/directory && \
    chmod -R g=u /some/directory
```

컨테이너 사용자는 항상 루트 그룹의 멤버이므로 컨테이너 사용자는 이러한 파일을 읽고 쓸 수 있습니다.

주의

컨테이너의 중요한 영역에 대한 디렉터리 및 파일 권한을 변경할 때는 주의해야 합니다. **/etc/passwd** 파일과 같은 민감한 영역에 적용되는 경우 이러한 변경으로 인해 의도하지 않은 사용자가 이러한 파일을 수정하여 컨테이너 또는 호스트가 노출될 수 있습니다. CRI-O는 컨테이너의 **/etc/passwd** 파일에 임의의 사용자 ID를 삽입할 수 있도록 지원합니다. 따라서 권한을 변경할 필요가 없습니다.

또한 컨테이너 이미지에 **/etc/passwd** 파일이 없어야 합니다. 이 경우 CRI-O 컨테이너 런타임에서 임의의 UID를 **/etc/passwd** 파일에 삽입하지 못합니다. 이러한 경우 컨테이너는 활성 UID를 해결하는 데 문제가 발생할 수 있습니다. 이 요구 사항을 충족하지 못하면 컨테이너화된 특정 애플리케이션의 기능에 영향을 미칠 수 있습니다.

또한, 컨테이너에서 실행되는 프로세스는 권한이 있는 사용자로 실행되지 않으므로 권한이 있는 포트(1024 미만의 포트)에서 수신 대기해서는 안 됩니다.

중요

S2I 이미지에 숫자 사용자를 사용한 **USER** 선언이 포함되어 있지 않으면 기본적으로 빌드는 실패합니다. OpenShift Container Platform에서 빌드하는 데 이름 지정된 사용자 또는 root **0** 사용자를 사용하는 이미지를 허용하려면 프로젝트의 빌더 서비스 계정 **system:serviceaccount:<your-project>:builder** 를 **anyuid** 보안 컨텍스트 제약 조건(SCC)에 추가할 수 있습니다. 또는 모든 이미지를 임의의 사용자로 실행하도록 허용할 수 있습니다.

4.1.2.3. 이미지 간 통신을 위해 서비스 사용

데이터베이스 이미지에 액세스하여 데이터를 저장하고 검색해야 하는 웹 프런트 엔드 이미지와 같이, 이미지가 다른 이미지에서 제공한 서비스와 통신해야 하는 경우 이미지는 OpenShift Container Platform 서비스를 사용해야 합니다. 서비스에서는 컨테이너가 중지되거나, 시작되거나, 이동될 때 액세스가 변경되지 않도록 정적 끝점을 제공합니다. 요청에 대한 부하 분산도 서비스에서 제공합니다.

4.1.2.4. 공통 라이브러리 제공

타사에서 제공한 애플리케이션 코드를 실행하기 위한 이미지의 경우 해당 플랫폼에 일반적으로 사용되는 라이브러리를 이미지에 포함해야 합니다. 특히, 해당 플랫폼과 함께 사용되는 공통 데이터베이스에 필요한 데이터베이스 드라이버를 제공하십시오. 예를 들어 Java 프레임워크 이미지를 생성하는 경우 MySQL 및 PostgreSQL용 JDBC 드라이버를 제공하십시오. 이렇게 하면 애플리케이션 어셈블리 시간 동안 공통 종속성을 다운로드할 필요가 없으므로 애플리케이션 이미지 빌드 속도가 빨라집니다. 애플리케이션 개발자가 모든 종속성이 충족되는지 확인하는 데 필요한 작업도 간소화됩니다.

4.1.2.5. 구성에 환경 변수 사용

이미지 사용자는 이미지를 기반으로 다운스트림 이미지를 생성하지 않고도 이미지를 구성할 수 있어야 합니다. 즉, 환경 변수를 사용하여 런타임 구성을 처리해야 합니다. 간단한 구성의 경우 실행 중인 프로세스에서 직접 환경 변수를 사용할 수 있습니다. 보다 복잡한 구성 또는 이 방법을 지원하지 않는 런타임의 경우 시작 시 처리되는 템플릿 구성 파일을 정의하여 런타임을 구성하십시오. 이러한 처리에서는 환경 변수를 통해 제공되는 값이 구성 파일로 대체되거나 구성 파일에서 설정할 옵션을 결정하는 데 사용될 수 있습니다.

또한, 환경 변수를 사용하여 인증서 및 키와 같은 시크릿을 컨테이너에 전달할 수 있으며 이 방법을 권장합니다. 이렇게 하면 시크릿 값이 이미지에 커밋되어 컨테이너 이미지 레지스트리로 유출되지 않습니다.

환경 변수를 제공하면 이미지 사용자가 이미지 위에 새 계층을 도입하지 않고도 데이터베이스 설정, 암호 및 성능 튜닝과 같은 동작을 사용자 정의할 수 있습니다. 대신, pod를 정의할 때 환경 변수 값을 정의하고 이미지를 다시 빌드하지 않고도 해당 설정을 변경할 수 있습니다.

매우 복잡한 시나리오의 경우 런타임 시 컨테이너에 마운트되는 볼륨을 사용하여 구성을 제공할 수도 있습니다. 하지만 이 방법으로 작업을 수행하도록 선택하는 경우 필요한 볼륨 또는 구성이 없으면 시작할 때 이미지에서 명확한 오류 메시지를 제공하도록 해야 합니다.

이 주제는 데이터 소스와 같은 구성이 서비스 끝점 정보를 제공하는 환경 변수의 관점에서 정의되어야 한다는 내용의 이미지 간 통신에 서비스 사용 주제와 관련이 있습니다. 이러한 방식을 선택하면 애플리케이션은 애플리케이션 이미지를 수정하지 않고도 OpenShift Container Platform 환경에 정의된 데이터 소스 서비스를 동적으로 사용할 수 있습니다.

또한, 컨테이너의 **cgroups** 설정을 검사하여 튜닝을 수행해야 합니다. 그러면 이미지가 사용 가능한 메모리, CPU 및 기타 리소스에 맞게 자체적으로 튜닝됩니다. 예를 들어 Java 기반 이미지는 **cgroup** 최대 메모리 매개변수를 기반으로 힙을 튜닝하여 이미지가 제한을 초과하지 않고 메모리 부족 오류가 발생하지 않

도록 합니다.

4.1.2.6. 이미지 메타데이터 설정

이미지 메타데이터를 정의하면 OpenShift Container Platform에서 컨테이너 이미지를 더 효율적으로 사용할 수 있으므로 OpenShift Container Platform에서 이미지를 사용하는 개발자를 위해 더 효율적인 환경을 구축할 수 있습니다. 예를 들어 이미지에 대한 유용한 설명을 제공하거나 기타 필요한 이미지를 제안하는 메타데이터를 추가할 수 있습니다.

4.1.2.7. 클러스터링

이미지의 여러 인스턴스를 실행한다는 것이 어떤 의미인지를 완전하게 이해하고 있어야 합니다. 가장 간단한 사례로, 서비스의 부하 분산 기능이 이미지의 모든 인스턴스에 대한 라우팅 트래픽을 처리하는 것을 들 수 있습니다. 하지만 세션 복제에서와 같이 리더 선택을 수행하거나 상태를 장애 조치하려면 많은 프레임워크에서 정보를 공유해야 합니다.

OpenShift Container Platform에서 실행하는 경우 인스턴스에서 이러한 통신을 수행할 방법을 고려해 보십시오. pod는 서로 직접 통신할 수 있지만 pod가 시작되거나, 중지되거나, 이동될 때마다 해당 IP 주소는 변경됩니다. 따라서 클러스터링 스키마가 동적인 것이 중요합니다.

4.1.2.8. 로깅

로깅은 모두 표준 출력으로 보내는 것이 가장 좋습니다. OpenShift Container Platform은 컨테이너에서 표준 출력을 수집하여 표준 출력을 볼 수 있는 중앙의 로깅 서비스로 보냅니다. 로그 콘텐츠를 분리해야 하는 경우 출력에 적절한 키워드 접두사를 붙여 메시지를 필터링할 수 있도록 합니다.

이미지가 파일에 로깅되면 사용자는 수동 작업을 통해 실행 중인 컨테이너에 들어가서 로그 파일을 검색하거나 확인해야 합니다.

4.1.2.9. liveness 및 readiness 프로브

이미지와 함께 사용할 수 있는 liveness 및 readiness 프로브 예를 문서화하십시오. 사용자는 이러한 프로브를 통해 컨테이너에서 트래픽을 처리할 준비가 될 때까지 트래픽이 컨테이너로 라우팅되지 않으며 프로세스가 비정상 상태가 되면 컨테이너가 다시 시작될 것이라는 확신을 가지고 이미지를 배포할 수 있습니다.

4.1.2.10. 템플릿

이미지와 함께 템플릿 예를 제공하십시오. 템플릿은 사용자가 정상적으로 작동하는 구성을 통해 이미지를 빠르게 배포할 수 있는 간편한 방법입니다. 완전성을 갖추려면 문서화한 liveness 및 readiness 프로브가 이미지와 함께 템플릿에 포함되어야 합니다.

4.2. 이미지에 메타데이터 포함

이미지 메타데이터를 정의하면 OpenShift Container Platform에서 컨테이너 이미지를 더 효율적으로 사용할 수 있으므로 OpenShift Container Platform에서 이미지를 사용하는 개발자를 위해 더 효율적인 환경을 구축할 수 있습니다. 예를 들어 이미지에 대한 유용한 설명을 제공하거나 기타 필요한 이미지를 제안하는 메타데이터를 추가할 수 있습니다.

이 주제에서는 현재의 사용 사례에 필요한 메타데이터만 정의합니다. 향후 메타데이터 또는 사용 사례가 더 추가될 수 있습니다.

4.2.1. 이미지 메타데이터 정의

Dockerfile에서 **LABEL** 명령어를 사용하여 이미지 메타데이터를 정의할 수 있습니다. 레이블은 이미지 또는 컨테이너에 연결된 키 값 쌍이라는 점에서 환경 변수와 유사합니다. 레이블은 실행 중인 애플리케이션에 표시되지 않으며 이미지 및 컨테이너를 빠르게 조회하는 데 사용될 수 있다는 점에서 환경 변수와 다릅니다.

LABEL 명령어에 대한 자세한 내용은 Docker 문서를 참조하십시오.

레이블 이름은 일반적으로 네임스페이스가 지정됩니다. 레이블을 찾아 사용할 프로젝트를 반영하여 네임스페이스를 적절히 설정해야 합니다. OpenShift Container Platform의 경우 네임스페이스는 **io.openshift**로 설정되어야 하며 Kubernetes의 경우 네임스페이스는 **io.k8s**입니다.

형식에 대한 자세한 내용은 Docker 사용자 정의 메타데이터 문서를 참조하십시오.

표 4.1. 지원되는 메타데이터

변수	설명
io.openshift.tags	이 레이블에는 쉼표로 구분된 문자열 값 목록으로 표시되는 태그 목록이 있습니다. 태그는 컨테이너 이미지를 광범위한 기능 영역으로 분류하는 방법입니다. 태그는 UI 및 생성 도구에서 애플리케이션 생성 프로세스 중 관련 컨테이너 이미지를 제안하는 데 도움이 됩니다. `LABEL io.openshift.tags  mongodb,mongodb24,nosql`
io.openshift.wants	지정된 태그가 있는 컨테이너 이미지가 아직 없는 경우 생성 도구 및 UI가 이와 관련된 제안을 하는 데 사용할 수 있는 태그 목록을 지정합니다. 예를 들어 컨테이너 이미지에 **mysql** 및 **redis**가 필요한데 **redis** 태그가 있는 컨테이너 이미지가 없는 경우 이 이미지를 배포에 추가하라는 UI 제안이 표시될 수 있습니다. `LABEL io.openshift.wants  mongodb,redis`
io.k8s.description	이 레이블을 사용하면 컨테이너 이미지 사용자에게 이 이미지가 제공하는 서비스 또는 기능에 대한 더 자세한 정보를 제공할 수 있습니다. 그러면 UI가 컨테이너 이미지 이름과 함께 이 설명을 사용하여 더 쉽게 이해할 수 있는 정보를 일반 사용자에게 제공할 수 있습니다. `LABEL io.k8s.description The MySQL 5.5 Server with master-slave replication support`
io.openshift.non-scalable	이미지는 이 변수를 사용하여 확장을 지원하지 않도록 제안할 수 있습니다. 그러면 UI가 이 내용을 해당 이미지 사용자에게 전달합니다. 확장 불가능하다는 것은 기본적으로 **replicas** 값이 처음에 **1**보다 크게 설정되지 않아야 한다는 것을 의미합니다. `LABEL io.openshift.non-scalable  true`

변수	설명

io.openshift.min-memory 및 **io.openshift.min-cpu**	이 레이블은 컨테이너 이미지가 제대로 작동하려면 리소스가 얼마나 필요한지를 제안합니다. 이 컨테이너 이미지를 배포하면 사용자 할당량이 초과된다는 UI 경고 메시지가 표시될 수 있습니다. 값은 Kubernetes 수량과 호환되어야 합니다.
	`LABEL io.openshift.min-memory 16Gi`
	`LABEL io.openshift.min-cpu 4`

4.3. S2I(SOURCE-TO-IMAGE)를 사용하여 소스 코드에서 이미지 생성

S2I(Source-to-Image)는 애플리케이션 소스 코드를 입력으로 사용하고 어셈블된 애플리케이션을 실행하는 새 이미지를 출력으로 생성하는 이미지를 쉽게 작성할 수 있는 프레임워크입니다.

재현 가능한 컨테이너 이미지를 빌드하는 데 S2I를 사용하는 주요 장점은 개발자가 쉽게 사용할 수 있다는 점입니다. 빌더 이미지 작성자는 이미지에서 최상의 S2I 성능, 빌드 프로세스, S2I 스크립트를 제공하도록 두 가지 기본 개념을 이해해야 합니다.

4.3.1. S2I(Source-to-Image) 빌드 프로세스 이해

빌드 프로세스는 최종 컨테이너 이미지로 통합되는 다음 세 가지 기본 요소로 구성됩니다.

- 소스
- S2I(Source-to-Image) 스크립트
- 빌더 이미지

S2I는 첫 번째 **FROM** 명령으로 빌더 이미지가 포함된 Dockerfile을 생성합니다. 그런 다음 S2I에서 생성된 Dockerfile은 Buildah로 전달됩니다.

4.3.2. S2I(Source-to-Image) 스크립트를 작성하는 방법

스크립트를 빌더 이미지 내에서 실행할 수 있는 경우 모든 프로그래밍 언어로 S2I(Source-to-Image) 스크립트를 작성할 수 있습니다. S2I는 **assemble/run/save-artifacts** 스크립트를 제공하는 다양한 옵션을 지원합니다. 이러한 위치는 모두 다음 순서에 따라 각 빌드에서 확인합니다.

1. 빌드 구성에 지정된 스크립트입니다.
2. 애플리케이션 소스 **.s2i/bin** 디렉터리에 있는 스크립트입니다.
3. 라벨이 **io.openshift.s2i.scripts-url**인 기본 이미지 URL에 있는 스크립트입니다.

이미지에 지정된 **io.openshift.s2i.scripts-url** 라벨과 빌드 구성에 지정된 스크립트 모두 다음 양식 중 하나를 취할 수 있습니다.

- **image:///path_to_scripts_dir**: S2I 스크립트가 있는 디렉터리에 대한 이미지 내부의 절대 경로입니다.
- **file:///path_to_scripts_dir**: S2I 스크립트가 있는 호스트의 디렉터리에 대한 상대 또는 절대 경로입니다.
- **http(s)://path_to_scripts_dir**: S2I 스크립트가 있는 디렉터리에 대한 URL입니다.

표 4.2. S2I 스크립트

스크립트	설명
assemble	**assemble** 스크립트는 소스에서 애플리케이션 아티팩트를 빌드하여 이미지 내부의 적절한 디렉터리에 배치합니다. 이 스크립트는 필수입니다. 이 스크립트의 워크플로는 다음과 같습니다. 1. 선택 사항: 빌드 아티팩트를 복구합니다. 증분 빌드를 지원하려면 **save-artifacts**도 정의해야 합니다. 2. 애플리케이션 소스를 원하는 위치에 배치합니다. 3. 애플리케이션 아티팩트를 빌드합니다. 4. 아티팩트를 실행할 수 있는 적절한 위치에 설치합니다.
run	**run** 스크립트는 애플리케이션을 실행합니다. 이 스크립트는 필수입니다.
save-artifacts	**save-artifacts** 스크립트는 이어지는 빌드 프로세스의 속도를 높일 수 있는 모든 종속 항목을 수집합니다. 이 스크립트는 선택 사항입니다. 예를 들면 다음과 같습니다. - Ruby의 경우 Bundler에서 설치한 **gems**입니다. - Java의 경우 **.m2** 콘텐츠입니다. 이러한 종속 항목은 **tar** 파일로 수집되어 표준 출력으로 스트리밍됩니다.
usage	**usage** 스크립트를 사용하면 사용자에게 이미지를 올바르게 사용하는 방법을 알릴 수 있습니다. 이 스크립트는 선택 사항입니다.
test/run	**test/run** 스크립트를 사용하면 이미지가 올바르게 작동하는지 확인하는 프로세스를 생성할 수 있습니다. 이 스크립트는 선택 사항입니다. 해당 프로세스의 제안된 흐름은 다음과 같습니다. 1. 이미지를 빌드합니다. 2. 이미지를 실행하여 **usage** 스크립트를 확인합니다. 3. **s2i build**를 실행하여 **assemble** 스크립트를 확인합니다. 4. 선택 사항: **s2i build**를 다시 실행하여 **save-artifacts** 및 **assemble** 스크립트에서 아티팩트 기능을 저장 및 복원하는지 확인합니다. 5. 이미지를 실행하여 테스트 애플리케이션이 작동하는지 확인합니다. **참고** **test/run** 스크립트로 빌드한 테스트 애플리케이션을 배치하도록 제안된 위치는 이미지 리포지토리의 **test/test-app** 디렉터리입니다.

S2I 스크립트의 예

다음 예제 S2I 스크립트는 Bash로 작성됩니다. 각 예에서는 **tar** 콘텐츠가 **/tmp/s2i** 디렉터리에 압축 해제되어 있다고 가정합니다.

assemble 스크립트:

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run 스크립트:

```
#!/bin/bash

# run the application
/opt/application/run.sh
```

save-artifacts 스크립트:

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage 스크립트:

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

추가 리소스

- S2I 이미지 생성 튜토리얼

4.4. S2I(SOURCE-TO-IMAGE) 이미지 테스트 정보

S2I(Source-to-Image) 빌더 이미지 작성자는 S2I 이미지를 로컬에서 테스트하고 자동 테스트와 지속적인 통합을 위해 OpenShift Container Platform 빌드 시스템을 사용할 수 있습니다.

S2I 빌드를 성공적으로 실행하려면 S2I에 **assemble** 및 **run** 스크립트가 있어야 합니다. **save-artifacts** 스크립트를 제공하면 빌드 아티팩트를 재사용하고 **usage** 스크립트를 제공하면 S2I 외부에서 컨테이너 이미지를 실행하는 경우 사용 정보가 콘솔에 인쇄되도록 합니다.

S2I 이미지 테스트는 기본 컨테이너 이미지가 변경되거나 명령에 사용된 도구가 업데이트되어도 설명한 모든 명령이 제대로 작동하는지 확인하기 위한 것입니다.

4.4.1. 테스트 요구 사항 이해

test 스크립트의 표준 위치는 **test/run**입니다. 이 스크립트는 OpenShift Container Platform S2I 이미지 빌더에 의해 호출되며 간단한 Bash 스크립트일 수도 있고 정적 Go 바이너리일 수도 있습니다.

test/run 스크립트는 S2I 빌드를 수행하므로 **$PATH**에서 S2I 바이너리를 사용할 수 있어야 합니다. 필요한 경우 S2I README의 설치 지침을 따르십시오.

S2I는 애플리케이션 소스 코드와 빌더 이미지를 결합하므로 테스트를 하려면 소스가 실행 가능한 컨테이너 이미지로 변환되는지 확인할 샘플 애플리케이션 소스가 있어야 합니다. 샘플 애플리케이션은 단순하되 **assemble** 및 **run** 스크립트의 중요한 단계를 수행해야 합니다.

4.4.2. 스크립트 및 도구 생성

S2I 도구는 새로운 S2I 이미지 생성 프로세스를 더욱 빠르게 수행할 수 있도록 강력한 생성 도구와 함께 제공됩니다. **s2i create** 명령에서는 모든 필요한 S2I 스크립트와 테스트 도구를 **Makefile**과 함께 생성합니다.

```
$ s2i create <image_name> <destination_directory>
```

생성된 **test/run** 스크립트는 유용하게 조정되어야 하지만 개발을 시작하기에 좋은 시작점을 제공합니다.

참고

s2i create 명령으로 생성된 **test/run** 스크립트를 사용하려면 샘플 애플리케이션 소스가 **test/test-app** 디렉토리 내에 있어야 합니다.

4.4.3. 로컬에서 테스트

S2I 이미지 테스트를 로컬에서 실행하는 가장 쉬운 방법은 생성된 **Makefile**을 사용하는 것입니다.

s2i create 명령을 사용하지 않은 경우 다음 **Makefile** 템플릿을 복사하고 **IMAGE_NAME** 매개변수를 이미지 이름으로 교체할 수 있습니다.

샘플 Makefile

```
IMAGE_NAME = openshift/ruby-20-centos7
CONTAINER_ENGINE := $(shell command -v podman 2> /dev/null | echo docker)

build:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME) .
```

```
.PHONY: test
test:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME)-candidate .
 IMAGE_NAME=$(IMAGE_NAME)-candidate test/run
```

4.4.4. 기본 테스트 워크플로

test 스크립트에서는 테스트할 이미지를 이미 빌드했다고 가정합니다. 필요한 경우 먼저 S2I 이미지를 빌드합니다. 다음 명령 중 하나를 실행합니다.

- Podman을 사용하는 경우 다음 명령을 실행합니다.

```
$ podman build -t <builder_image_name>
```

- Docker를 사용하는 경우 다음 명령을 실행합니다.

```
$ docker build -t <builder_image_name>
```

다음 단계에서는 S2I 이미지 빌더를 테스트하는 기본 워크플로를 설명합니다.

1. **usage** 스크립트가 작동 중인지 확인합니다.
   - Podman을 사용하는 경우 다음 명령을 실행합니다.

```
$ podman run <builder_image_name> .
```

   - Docker를 사용하는 경우 다음 명령을 실행합니다.

```
$ docker run <builder_image_name> .
```

2. 이미지를 빌드합니다.

```
$ s2i build file:///path-to-sample-app _<BUILDER_IMAGE_NAME>_
_<OUTPUT_APPLICATION_IMAGE_NAME>_
```

3. 선택사항: **save-artifacts**를 지원하는 경우 2단계를 다시 한번 실행하여 아티팩트 저장 및 복원이 제대로 작동하는지 확인합니다.
4. 컨테이너를 실행합니다.
   - Podman을 사용하는 경우 다음 명령을 실행합니다.

```
$ podman run <output_application_image_name>
```

   - Docker를 사용하는 경우 다음 명령을 실행합니다.

```
$ docker run <output_application_image_name>
```

5. 컨테이너가 실행 중이고 애플리케이션이 응답하는지 확인합니다.

일반적으로 이러한 단계를 실행하면 빌더 이미지가 예상대로 작동하는지 충분히 확인할 수 있습니다.

4.4.5. 이미지 빌드에 OpenShift Container Platform 사용

새로운 S2I 빌더 이미지를 구성하는 **Dockerfile** 및 기타 아티팩트가 있으면 git 리포지터리에 배치한 후 OpenShift Container Platform을 사용하여 이미지를 빌드하고 푸시할 수 있습니다. 리포지터리를 가리키는 Docker 빌드를 정의합니다.

OpenShift Container Platform 인스턴스가 공용 IP 주소에서 호스트된 경우 S2I 빌더 이미지 GitHub 리포지터리로 푸시할 때마다 빌드가 트리거될 수 있습니다.

ImageChangeTrigger를 사용하면 업데이트된 S2I 빌더 이미지를 기반으로 하는 애플리케이션의 다시 빌드를 트리거할 수도 있습니다.

5장. 이미지 관리

5.1. 이미지 관리 개요

OpenShift Container Platform을 사용하면 이미지 레지스트리 위치, 해당 레지스트리에 대한 인증 요구 사항, 빌드 및 배포의 바람직한 작동 방식에 따라 이미지와 상호 작용하고 이미지 스트림을 설정할 수 있습니다.

5.1.1. 이미지 개요

이미지 스트림은 태그로 식별되는 임의 숫자의 컨테이너 이미지로 구성됩니다. 컨테이너 이미지 리포지터리와 유사하게 관련 이미지에 대한 단일 가상 뷰를 제공합니다.

빌드 및 배포는 이미지 스트림을 모니터링하다가 새 이미지가 추가되거나 이미지가 수정되면 알림을 받고 빌드 또는 배포를 수행하여 각각에 대응할 수 있습니다.

5.2. 이미지 태그 지정

다음 섹션에서는 OpenShift Container Platform 이미지 스트림과 해당 태그로 작업하기 위해 컨테이너 이미지 컨텍스트에서 이미지 태그를 사용하는 데 필요한 개요 및 지침을 제공합니다.

5.2.1. 이미지 태그

이미지 태그는 리포지터리의 컨테이너 이미지에 적용되는 레이블로, 이미지 스트림에서 특정 이미지를 다른 이미지와 구별하는 역할을 합니다. 일반적으로 태그는 일종의 버전 번호를 나타냅니다. 예를 들어 다음에서는 **:v3.11.59-2**가 태그입니다.

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

이미지에 태그를 더 추가할 수 있습니다. 예를 들어 이미지에 **:v3.11.59-2** 및 **:latest** 태그가 할당될 수 있습니다.

OpenShift Container Platform은 **docker tag** 명령과 유사하지만 이미지가 아닌 이미지 스트림에서 작동하는 **oc tag** 명령을 제공합니다.

5.2.2. 이미지 태그 규칙

이미지는 시간이 지남에 따라 개선되며 해당 태그를 사용하여 이러한 개선을 반영합니다. 일반적으로 이미지 태그는 항상 빌드된 최신 이미지를 가리킵니다.

v2.0.1-may-2019와 같이 태그 이름에 너무 많은 정보가 포함되어 있으면 태그는 이미지의 한 버전만 가리키며 업데이트되지 않습니다. 기본 이미지 정리 옵션을 사용하는 경우 이러한 이미지는 절대 제거되지 않습니다. 크기가 매우 큰 클러스터에서는 모든 수정된 이미지에 대한 새로운 태그 생성 스키마가 아주 오래된 이미지에 대한 초과 태그 메타데이터로 etcd 데이터 저장소를 채울 수 있습니다.

태그 이름이 **v2.0**이면 이미지가 수정될 가능성이 높습니다. 그러면 태그 기록이 길어지고, 따라서 이미지 정리기가 오래되고 사용되지 않는 이미지를 제거할 가능성이 높습니다.

태그 이름 지정 규칙은 사용자가 결정하지만 아래에서 **<image_name>:<image_tag>** 형식으로 된 몇 가지 예를 볼 수 있습니다.

표 5.1. 이미지 태그 이름 지정 규칙

설명	예
버전	**myimage:v2.0.1**
아키텍처	**myimage:v2.0-x86_64**
기본 이미지	**myimage:v1.2-centos7**
최신(불안정한 상태가 될 수 있음)	**myimage:latest**
안정된 최신 상태	**myimage:stable**

태그 이름에 날짜가 필요한 경우 오래되고 지원되지 않는 이미지와 **istags**를 정기적으로 검사하여 제거하십시오. 그러지 않으면 오래된 이미지 유지로 인해 리소스 사용량이 증가할 수 있습니다.

5.2.3. 이미지 스트림에 태그 추가

OpenShift Container Platform의 이미지 스트림은 태그로 식별되는 0개 이상의 컨테이너 이미지로 구성됩니다.

사용할 수 있는 태그 유형은 다양합니다. 기본 동작에서는 시간의 특정 이미지를 가리키는 **permanent** 태그를 사용합니다. **permanent** 태그가 사용 중이고 소스가 변경되는 경우 대상에 대해 태그가 변경되지 않습니다.

tracking 태그는 소스 태그를 가져오는 동안 대상 태그의 메타데이터가 업데이트되었음을 나타냅니다.

절차

- **oc tag** 명령을 사용하여 이미지 스트림에 태그를 추가할 수 있습니다.

```
$ oc tag <source> <destination>
```

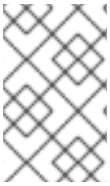
  예를 들어 **ruby** 이미지 스트림 **static-2.0** 태그가 항상 **ruby** 이미지 스트림 **2.0** 태그의 현재 이미지를 참조하도록 구성하려면 다음을 사용합니다.

```
$ oc tag ruby:2.0 ruby:static-2.0
```

  이 명령을 사용하면 **ruby** 이미지 스트림에 **static-2.0**이라는 새 이미지 스트림 태그가 생성됩니다. 새 태그는 **oc tag**가 실행될 때 **ruby:2.0** 이미지 스트림 태그가 가리키는 이미지 ID를 직접 참조하며 태그가 가리키는 이미지는 변경되지 않습니다.

- 소스 태그가 변경될 때 대상 태그가 업데이트되도록 하려면 **--alias=true** 플래그를 사용합니다.

```
$ oc tag --alias=true <source> <destination>
```



참고

영구 별칭(예: **latest** 또는 **stable**)을 생성하려면 추적 태그를 사용하십시오. 이 태그는 단일 이미지 스트림 내에서만 제대로 작동합니다. 이미지 스트림 간 별칭을 생성하려고 하면 오류가 발생합니다.

- **--scheduled=true** 플래그를 추가하여 대상 태그를 정기적으로 새로 고치거나 다시 가져오도록 할 수 있습니다. 기간은 시스템 수준에서 전역적으로 구성됩니다.

- **--reference** 플래그는 가져오지 않는 이미지 스트림 태그를 생성합니다. 태그는 영구적으로 소스 위치를 가리킵니다.
  OpenShift Container Platform이 항상 통합 레지스트리에서 태그된 이미지를 가져오도록 지정하려면 **--reference-policy=local**을 사용하십시오. 레지스트리는 가져오기 기능을 사용하여 클라이언트에 이미지를 제공합니다. 기본적으로 이미지 Blob은 레지스트리에 의해 로컬로 미러링됩니다. 따라서 다음에 필요할 때 더 빨리 가져올 수 있습니다. 이 플래그를 사용하면 이미지 스트림에 비보안 주석이 있거나 태그에 비보안 가져오기 정책이 있는 한 컨테이너 런타임에 **--insecure-registry**를 제공할 필요 없이 비보안 레지스트리에서 가져올 수도 있습니다.

5.2.4. 이미지 스트림에서 태그 제거

이미지 스트림에서 태그를 제거할 수 있습니다.

절차

- 이미지 스트림에서 태그를 완전히 제거하려면 다음을 실행합니다.

```
$ oc delete istag/ruby:latest
```

  또는 다음을 수행합니다.

```
$ oc tag -d ruby:latest
```

5.2.5. 이미지 스트림의 이미지 참조

태그를 사용하면 다음 참조 유형이 사용된 이미지 스트림의 이미지를 참조할 수 있습니다.

표 5.2. 이미지 스트림 참조 유형

참조 유형	설명
ImageStreamTag	**ImageStreamTag**는 지정된 이미지 스트림 및 태그의 이미지를 참조하거나 검색하는 데 사용됩니다.
ImageStreamImage	**ImageStreamImage**는 지정된 이미지 스트림 및 이미지 **sha** ID의 이미지를 참조하거나 검색하는 데 사용됩니다.
DockerImage	**DockerImage**는 지정된 외부 레지스트리의 이미지를 참조하거나 검색하는 데 사용됩니다. 해당 이름에 표준 Docker **pull specification**을 사용합니다.

이미지 스트림 정의 예를 보면 **ImageStreamTag** 정의와 **DockerImage**에 대한 참조는 포함되어 있으나 **ImageStreamImage**와 관련된 항목은 포함되어 있지 않음을 알 수 있습니다.

이미지 스트림으로 이미지를 가져오거나 태그하는 경우 **ImageStreamImage** 오브젝트가 OpenShift Container Platform에서 자동으로 생성되기 때문입니다. 이미지 스트림을 생성하는 데 사용하는 이미지 스트림 정의에서는 **ImageStreamImage** 오브젝트를 명시적으로 정의할 필요가 없습니다.

절차

- 지정된 이미지 스트림 및 태그의 이미지를 참조하려면 **ImageStreamTag**를 사용합니다.

```
<image_stream_name>:<tag>
```

- 지정된 이미지 스트림 및 이미지 **sha** ID의 이미지를 참조하려면 **ImageStreamImage**를 사용합니다.

```
<image_stream_name>@<id>
```

<id>는 다이제스트라고도 하는 특정 이미지의 변경 불가능한 식별자입니다.

- 지정된 외부 레지스트리의 이미지를 참조하거나 검색하려면 **DockerImage**를 사용합니다.

```
openshift/ruby-20-centos7:2.0
```

> **참고**
>
> 태그가 지정되지 않은 경우 **latest** 태그가 사용된 것으로 가정합니다.

다음과 같은 타사 레지스트리를 참조할 수도 있습니다.

```
registry.redhat.io/rhel7:latest
```

또는 다이제스트가 있는 이미지도 참조할 수 있습니다.

```
centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b2
8e
```

5.3. 이미지 가져오기 정책

pod의 각 컨테이너에는 컨테이너 이미지가 있습니다. 이미지를 생성하여 레지스트리로 푸시한 후 Pod에서 해당 이미지를 참조할 수 있습니다.

5.3.1. 이미지 가져오기 정책 개요

OpenShift Container Platform은 컨테이너를 생성할 때 컨테이너 **imagePullPolicy**를 사용하여 컨테이너를 시작하기 전에 이미지를 가져와야 하는지 결정합니다. **imagePullPolicy**에 가능한 값은 다음 세 가지입니다.

표 5.3. imagePullPolicy 값

값	설명
Always	항상 이미지를 가져옵니다.
IfNotPresent	이미지가 아직 노드에 없는 경우에만 이미지를 가져옵니다.

값	설명
Never	이미지를 가져오지 않습니다.

컨테이너 **imagePullPolicy** 매개변수가 지정되지 않은 경우 OpenShift Container Platform은 이미지 태그를 기반으로 이 매개변수를 설정합니다.

1. 태그가 **latest**인 경우 OpenShift Container Platform은 기본적으로 **imagePullPolicy**를 **Always**로 설정합니다.
2. 태그가 그 이외의 값인 경우 OpenShift Container Platform은 기본적으로 **imagePullPolicy**를 **IfNotPresent**로 설정합니다.

5.4. 이미지 풀 시크릿 사용

OpenShift 이미지 레지스트리를 사용하고 동일한 프로젝트에 있는 이미지 스트림에서 이미지를 가져오는 경우 pod 서비스 계정에 이미 올바른 권한이 있어야 하며 추가 작업이 필요하지 않습니다.

그러나 OpenShift Container Platform 프로젝트 또는 보안 레지스트리에서 이미지를 참조하는 것과 같은 다른 시나리오의 경우 추가 구성 단계가 필요합니다.

Red Hat OpenShift Cluster Manager에서 이미지 풀 시크릿 을 가져올 수 있습니다. 이 풀 시크릿을 **pullSecret** 이라고 합니다.

이 풀 시크릿을 사용하여 OpenShift Container Platform 구성 요소에 대한 컨테이너 이미지를 제공하는 인증 기관, Quay.io 및 registry.redhat.io 에서 제공하는 서비스로 인증합니다.

5.4.1. 프로젝트 간에 pod가 이미지를 참조할 수 있도록 허용

OpenShift 이미지 레지스트리를 사용하는 경우 **project-a** 의 pod가 **project-b** 의 이미지를 참조하도록 허용하려면 **project-a** 의 서비스 계정을 **project-b** 의 **system:image-puller** 역할에 바인딩해야 합니다.

참고

Pod 서비스 계정 또는 네임스페이스를 생성할 때 docker pull secret을 사용하여 서비스 계정이 프로비저닝될 때까지 기다립니다. 서비스 계정이 완전히 프로비저닝되기 전에 pod를 생성하면 Pod가 OpenShift 이미지 레지스트리에 액세스하지 못합니다.

절차

1. **project-a**의 pod가 **project-b**의 이미지를 참조하도록 허용하려면 **project-a**의 서비스 계정을 **project-b**의 **system:image-puller** 역할에 바인딩합니다.

```
$ oc policy add-role-to-user \
    system:image-puller system:serviceaccount:project-a:default \
    --namespace=project-b
```

   해당 역할을 추가한 후에는 기본 서비스 계정을 참조하는 **project-a**의 pod가 **project-b**의 이미지를 가져올 수 있습니다.

2. **project-a**의 모든 서비스 계정에 액세스를 허용하려면 다음 그룹을 사용합니다.

```
$ oc policy add-role-to-group \
    system:image-puller system:serviceaccounts:project-a \
    --namespace=project-b
```

5.4.2. Pod에서 다른 보안 레지스트리의 이미지를 참조하도록 허용

Docker 클라이언트의 **.dockercfg $HOME/.docker/config.json** 파일은 이전에 보안 또는 비보안 레지스트리에 로그인한 적이 있는 경우 인증 정보를 저장하는 Docker 인증 정보 파일입니다.

OpenShift 이미지 레지스트리가 아닌 보안 컨테이너 이미지를 가져오려면 Docker 인증 정보에서 풀 시크릿을 생성하여 서비스 계정에 추가해야 합니다.

Docker 인증 정보 파일과 연결된 풀 암호에는 각각 고유한 인증 정보 세트가 있는 동일한 레지스트리에 대한 여러 참조가 포함될 수 있습니다.

config.json 파일 예

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}
```

pull secret의 예

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

절차

- 보안 레지스트리의 **.dockercfg** 파일이 이미 있는 경우 다음을 실행하여 해당 파일에서 시크릿을 생성할 수 있습니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockercfg=<path/to/.dockercfg> \
    --type=kubernetes.io/dockercfg
```

- **$HOME/.docker/config.json** 파일이 있는 경우 다음을 실행합니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

- 보안 레지스트리의 Docker 인증 정보 파일이 아직 없는 경우 다음을 실행하여 시크릿을 생성할 수 있습니다.

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

- Pod의 이미지 가져오기에 시크릿을 사용하려면 서비스 계정에 이 시크릿을 추가해야 합니다. 이 예제의 서비스 계정 이름은 Pod에서 사용하는 서비스 계정 이름과 일치해야 합니다. **default**는 기본 서비스 계정입니다.

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5.4.2.1. 위임된 인증을 사용하여 개인 레지스트리에서 가져오기

개인 레지스트리는 별도의 서비스에 인증을 위임할 수 있습니다. 이 경우 인증 및 레지스트리 끝점 둘 다에 대해 이미지 풀 시크릿을 정의해야 합니다.

프로세스

1. 위임된 인증 서버에 대한 시크릿을 생성합니다.

```
$ oc create secret docker-registry \
    --docker-server=sso.redhat.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    redhat-connect-sso

secret/redhat-connect-sso
```

2. 개인 레지스트리에 대한 시크릿을 생성합니다.

```
$ oc create secret docker-registry \
    --docker-server=privateregistry.example.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    private-registry

secret/private-registry
```

5.4.3. 글로벌 클러스터 풀 시크릿 업데이트

현재 풀 시크릿을 교체하거나 새 풀 시크릿을 추가하여 클러스터의 글로벌 풀 시크릿을 업데이트할 수 있습니다.

중요

클러스터를 다른 소유자로 전송하려면 먼저 OpenShift Cluster Manager Hybrid Cloud Console 에서 전송을 시작한 다음 클러스터에서 풀 시크릿을 업데이트해야 합니다. OpenShift Cluster Manager에서 전송을 시작하지 않고 클러스터의 풀 시크릿을 업데이트하면 클러스터에서 OpenShift Cluster Manager에서 Telemetry 지표 보고를 중지합니다.

클러스터 소유권 양도에 대한 자세한 내용은 Red Hat OpenShift Cluster Manager 설명서의 "클러스터 소유권 전송"을 참조하십시오.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

절차

1. 선택 사항: 기존 풀 시크릿에 새 풀 시크릿을 추가하려면 다음 단계를 완료합니다.

   a. 다음 명령을 입력하여 풀 시크릿을 다운로드합니다.

   ```
   $ oc get secret/pull-secret -n openshift-config --template='{{index .data
   ".dockerconfigjson" | base64decode}}' ><pull_secret_location> (1)
   ```

   (1) 풀 시크릿 파일에 경로를 제공합니다.

   b. 다음 명령을 입력하여 새 풀 시크릿을 추가합니다.

   ```
   $ oc registry login --registry="<registry>" \ (1)
   --auth-basic="<username>:<password>" \ (2)
   --to=<pull_secret_location> (3)
   ```

   (1) 새 레지스트리를 제공합니다. 동일한 레지스트리에 여러 리포지토리를 포함할 수 있습니다 (예: **--registry="<registry/my-namespace/my-repository>**).

   (2) 새 레지스트리의 인증 정보를 제공합니다.

   (3) 풀 시크릿 파일에 경로를 제공합니다.

   또는 가져오기 시크릿 파일에 대한 수동 업데이트를 수행할 수 있습니다.

2. 다음 명령을 입력하여 클러스터의 글로벌 풀 시크릿을 업데이트합니다.

   ```
   $ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
   <pull_secret_location> (1)
   ```

   (1) 새 풀 시크릿 파일의 경로를 제공합니다.

이 업데이트는 모든 노드로 롤아웃되며 클러스터 크기에 따라 작업에 약간의 시간이 걸릴 수 있습니다.

참고

OpenShift Container Platform 4.7.4부터 글로벌 풀 시크릿을 변경해도 더 이상 노드 드레이닝 또는 재부팅이 트리거되지 않습니다.

6장. 이미지 스트림 관리

이미지 스트림은 지속적으로 컨테이너 이미지를 생성하고 업데이트할 수단을 제공합니다. 이미지가 개선되면 태그를 사용하여 새 버전 번호를 할당하고 변경 사항을 추적할 수 있습니다. 이 문서에서는 이미지 스트림을 관리하는 방법을 설명합니다.

6.1. 이미지 스트림을 사용하는 이유

이미지 스트림 및 관련 태그는 OpenShift Container Platform 내에서 컨테이너 이미지를 참조하는 데 필요한 추상을 제공합니다. 이미지 스트림 및 해당 태그를 사용하면 사용 가능한 이미지를 볼 수 있으며 리포지터리의 이미지가 변경되어도 필요한 특정 이미지를 사용하도록 할 수 있습니다.

이미지 스트림에는 실제 이미지 데이터가 포함되어 있지 않지만 이미지 리포지터리와 유사하게 관련 이미지에 대한 단일 가상 뷰가 있습니다.

새 이미지가 추가되는 경우 이미지 스트림의 알림을 보고 빌드 또는 배포를 각각 수행하여 대응하도록 빌드 및 배포를 구성할 수 있습니다.

예를 들어 배포에서 특정 이미지를 사용하고 있는데 해당 이미지의 새 버전이 생성되는 경우 배포가 자동으로 수행되어 새 버전의 이미지를 가져올 수 있습니다.

하지만 배포 또는 빌드에서 사용하는 이미지 스트림 태그가 업데이트되지 않으면 컨테이너 이미지 레지스트리의 컨테이너 이미지가 업데이트되어도 빌드 또는 배포에서는 알려진 정상 이미지인 이전 이미지를 계속 사용합니다.

소스 이미지를 저장할 수 있는 위치는 다음과 같습니다.

- OpenShift Container Platform 통합 레지스트리
- 외부 레지스트리(예: registry.redhat.io 또는 quay.io)
- OpenShift Container Platform 클러스터의 다른 이미지 스트림

이미지 스트림 태그를 참조하는 오브젝트(예: 빌드 또는 배포 구성)를 정의할 때 리포지터리가 아닌 이미지 스트림 태그를 가리킵니다. 애플리케이션을 빌드하거나 배포할 때 OpenShift Container Platform은 이 이미지 스트림 태그로 리포지터리를 조회하여 관련된 이미지 ID를 찾은 후 바로 그 이미지를 사용합니다.

이미지 스트림 메타데이터는 다른 클러스터 정보와 함께 etcd 인스턴스에 저장됩니다.

이미지 스트림을 사용하면 다음과 같은 여러 중요한 이점이 있습니다.

- 명령줄을 사용하여 다시 푸시하지 않고도 태그하고, 태그를 롤백하고, 이미지를 빠르게 처리할 수 있습니다.
- 새 이미지가 레지스트리로 푸시되면 빌드 및 배포를 트리거할 수 있습니다. OpenShift Container Platform에는 Kubernetes 오브젝트 등 다른 리소스에 대한 일반 트리거도 있습니다.
- 정기적인 다시 가져오기를 위해 태그를 표시할 수 있습니다. 소스 이미지가 변경되면 해당 변경 사항이 이미지 스트림에 반영되고, 이후 빌드 또는 배포 구성에 따라 빌드 및/또는 배포 흐름이 트리거됩니다.
- 세분화된 액세스 제어를 사용하여 이미지를 공유하고 팀 전체에 이미지를 빠르게 배포할 수 있습니다.
- 소스 이미지가 변경되면 이미지 스트림 태그는 여전히 알려진 양호한 버전의 이미지를 가리키므로 애플리케이션이 예기치 않게 손상되지 않도록 합니다.

- 이미지 스트림 오브젝트에 대한 권한을 통해 이미지를 보고 사용할 수 있는 사람에 대한 보안을 구성할 수 있습니다.
- 클러스터 수준에서 이미지를 읽거나 나열할 수 있는 권한이 없는 사용자도 이미지 스트림을 사용하여 프로젝트의 태그된 이미지를 검색할 수 있습니다.

6.2. 이미지 스트림 구성

ImageStream 오브젝트 파일에는 다음 요소가 포함되어 있습니다.

이미지 스트림 오브젝트 정의

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample (2)
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d (3)
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 (4)
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest (5)
```

1. 이미지 스트림의 이름입니다.
2. 이 이미지 스트림에서 이미지를 추가 또는 업데이트하기 위해 새 이미지를 푸시할 수 있는 Docker 리포지터리 경로입니다.
3. 이 이미지 스트림 태그가 현재 참조하는 SHA 식별자입니다. 이 이미지 스트림 태그를 참조하는 리소스는 이 식별자를 사용합니다.
4. 이 이미지 스트림 태그가 이전에 참조한 SHA 식별자입니다. 이전 이미지로 롤백하는 데 사용할 수 있습니다.
5. 이미지 스트림 태그 이름

6.3. 이미지 스트림 이미지

이미지 스트림 이미지는 이미지 스트림 내에서 특정 이미지 ID를 가리킵니다.

이미지 스트림 이미지를 사용하면 태그된 특정 이미지 스트림에서 이미지에 대한 메타데이터를 검색할 수 있습니다.

이미지 스트림 이미지 오브젝트는 이미지 스트림으로 이미지를 가져오거나 태그할 때마다 OpenShift Container Platform에서 자동으로 생성됩니다. 이미지 스트림을 생성하는 데 사용하는 이미지 스트림 정의에서는 이미지 스트림 태그 오브젝트를 명시적으로 정의할 필요가 없습니다.

이미지 스트림 이미지는 리포지터리의 이미지 스트림 이름과 이미지 ID로 구성됩니다. 이름 및 ID는 **@** 기호로 구분됩니다.

```
<image-stream-name>@<image-id>
```

ImageStream 개체 예의 이미지를 참조하려면 이미지 스트림 이미지가 다음과 유사해야 합니다.

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.4. 이미지 스트림 태그

이미지 스트림 태그는 이미지 스트림의 이미지에 대한 이름 지정된 포인터입니다. **istag**로 축약됩니다. 이미지 스트림 태그는 지정된 이미지 스트림 및 태그의 이미지를 참조하거나 검색하는 데 사용됩니다.

이미지 스트림 태그는 로컬 또는 외부 관리 이미지를 참조할 수 있습니다. 태그가 가리켰던 모든 이미지의 스택으로 표시되는 이미지 기록이 이미지 스트림 태그에 포함되어 있습니다. 새 이미지 또는 기존 이미지가 특정 이미지 스트림 태그 아래에 태그될 때마다 기록 스택의 첫 번째 위치에 배치됩니다. 이전에 맨 위 위치를 차지했던 이미지는 두 번째 위치에서 사용할 수 있습니다. 이렇게 하면 태그가 다시 과거 이미지를 가리키도록 손쉽게 롤백할 수 있습니다.

다음 이미지 스트림 태그는 **ImageStream** 오브젝트에서 가져온 것입니다.

기록에 두 개의 이미지가 있는 이미지 스트림 태그

```
kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: my-image-stream
# ...
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest
# ...
```

이미지 스트림 태그는 영구 태그일 수도 있고 추적 태그일 수도 있습니다.

- 영구 태그는 Python 3.5 같은 특정 버전의 이미지를 가리키는 버전 특정 태그입니다.
- 추적 태그는 다른 이미지 스트림 태그를 따르는 참조 태그이며, symlink와 매우 비슷하게 나중에 이 태그를 업데이트하여 따를 이미지를 변경할 수 있습니다. 이러한 새 수준은 이전 버전과 호환되지 않을 수 있습니다.
  예를 들어 OpenShift Container Platform과 함께 제공되는 **latest** 이미지 스트림 태그는 추적 태그입니다. 즉, **latest** 이미지 스트림 태그 사용자는 새로운 수준을 사용할 수 있게 되면 이미지에서 제공하는 프레임워크의 최신 수준으로 업데이트됩니다. **v3.10**에 대한 **latest** 이미지 스트림 태그는 언제든 **v3.11** 로 변경될 수 있습니다. 이러한 **latest** 이미지 스트림 태그는 Docker **latest** 태그와 다르게 동작한다는 사실을 알고 있는 것이 중요합니다. Docker의 경우 **latest** 이미지 스트림 태그가 Docker 리포지터리의 최신 이미지를 가리키지 않습니다. 다른 이미지 스트림 태그를 가리키며, 이것은 이미지의 최신 버전이 아닐 수도 있습니다. 예를 들어 **latest** 이미지 스트림 태그가 이미지의 **v3.10**을 가리키는 경우 **3.11** 버전이 릴리스되면 **latest** 태그가 **v3.11**로 자동 업데이트되지 않고 **v3.11** 이미지 스트림 태그를 가리키도록 수동 업데이트될 때까지 **v3.10**으로 남아 있습니다.

참고

추적 태그는 단일 이미지 스트림으로 제한되며 다른 이미지 스트림을 참조할 수 없습니다.

고유한 요구 사항에 맞게 자체 이미지 스트림 태그를 생성할 수 있습니다.

이미지 스트림 태그는 콜론으로 구분된 이미지 스트림 이름 및 태그로 구성됩니다.

```
<imagestream name>:<tag>
```

예를 들어 앞의 **ImageStream** 오브젝트 예에 있는 **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** 이미지를 참조하려면 이미지 스트림 태그는 다음과 같습니다.

```
origin-ruby-sample:latest
```

6.5. 이미지 스트림 변경 트리거

이미지 스트림 트리거를 사용하면 새 버전의 업스트림 이미지가 준비될 때 빌드 및 배포가 자동으로 호출됩니다.

예를 들어 이미지 스트림 태그가 수정되면 빌드 및 배포를 자동으로 시작할 수 있습니다. 이 과정은 해당하는 특정 이미지 스트림 태그를 모니터링하다가 변경이 탐지되면 빌드 또는 배포에 알리는 방식으로 이루어집니다.

6.6. 이미지 스트림 매핑

통합 레지스트리에서 새 이미지를 수신하면 이미지 스트림 매핑을 생성하여 OpenShift Container Platform으로 보내서 이미지의 프로젝트, 이름, 태그 및 이미지 메타데이터를 제공합니다.

참고

이미지 스트림 매핑 구성은 고급 기능입니다.

이러한 정보는 새 이미지를 생성하고(아직 없는 경우) 이미지를 이미지 스트림에 태그하는 데 사용됩니다. OpenShift Container Platform은 명령, 진입점, 환경 변수 등 각 이미지에 대한 완전한 메타데이터를 저장합니다. OpenShift Container Platform의 이미지는 변경 불가능하며 이름의 최대 길이는 63자입니다.

다음 이미지 스트림 매핑 예에서는 **test/origin-ruby-sample:latest**로 이미지가 태그됩니다.

이미지 스트림 매핑 오브젝트 정의

```
apiVersion: image.openshift.io/v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Labels:
```

```
        build-date: 2015-12-23
        io.k8s.description: Platform for building and running Ruby 2.2 applications
        io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
        io.openshift.build.commit.author: Ben Parees <bparees@users.noreply.github.com>
        io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
        io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
        io.openshift.build.commit.message: 'Merge pull request #51 from php-coder/fix_url_and_sti'
        io.openshift.build.commit.ref: master
        io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
        io.openshift.build.source-location: https://github.com/openshift/ruby-hello-world.git
        io.openshift.builder-base-version: 8d95148
        io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: builder,ruby,ruby22
        io.s2i.scripts-url: image:///usr/libexec/s2i
        license: GPLv2
        name: CentOS Base Image
        vendor: CentOS
      User: "1001"
      WorkingDir: /opt/app-root/src
    Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
    ContainerConfig:
      AttachStdout: true
      Cmd:
      - /bin/sh
      - -c
      - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Hostname: ruby-sample-build-1-build
      Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
      OpenStdin: true
      StdinOnce: true
      User: "1001"
      WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
```

```
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.7. 이미지 스트림으로 작업

다음 섹션에서는 이미지 스트림 및 이미지 스트림 태그를 사용하는 방법에 대해 설명합니다.

6.7.1. 이미지 스트림에 대한 정보 얻기

이미지 스트림에 대한 일반 정보와 해당 이미지 스트림이 가리키는 모든 태그에 대한 자세한 정보를 얻을 수 있습니다.

절차

- 이미지 스트림에 대한 일반 정보와 해당 이미지 스트림이 가리키는 모든 태그에 대한 자세한 정보를 얻습니다.

  ```
  $ oc describe is/<image-name>
  ```

  예를 들면 다음과 같습니다.

  ```
  $ oc describe is/python
  ```

  출력 예

  ```
  Name:   python
  Namespace:  default
  Created:  About a minute ago
  Labels:   <none>
  Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
  Docker Pull Spec: docker-registry.default.svc:5000/default/python
  Image Lookup:  local=false
  Unique Images:  1
  Tags:   1

  3.5
    tagged from centos/python-35-centos7

    * centos/python-35-
  centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
        About a minute ago
  ```

- 특정 이미지 스트림 태그에 대한 사용 가능한 모든 정보를 얻습니다.

  ```
  $ oc describe istag/<image-stream>:<tag-name>
  ```

  예를 들면 다음과 같습니다.

```
$ oc describe istag/python:latest
```

출력 예

```
Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name:  sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created: 2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author:  <none>
Arch:  amd64
Entrypoint: container-entrypoint
Command: /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
User:  1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```

참고

표시된 것보다 더 많은 정보가 출력됩니다.

6.7.2. 이미지 스트림에 태그 추가

이미지 스트림에 태그를 더 추가할 수 있습니다.

절차

- 'oc tag' 명령을 사용하여 기존 태그 중 하나를 가리키는 태그를 추가합니다.

```
$ oc tag <image-name:tag1> <image-name:tag2>
```

예를 들면 다음과 같습니다.

```
$ oc tag python:3.5 python:latest
```

출력 예

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

- 이미지 스트림에 두 개의 태그 즉, 외부 컨테이너 이미지를 가리키는 하나의 태그(**3.5**)와 첫 번째 태그를 기반으로 생성되어 동일한 이미지를 가리키는 다른 태그(**latest**)가 있는지 확인합니다.

```
$ oc describe is/python
```

출력 예

```
Name:   python
Namespace:  default
```

```
Created:  5 minutes ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   2

latest
  tagged from
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      5 minutes ago
```

6.7.3. 외부 이미지에 대해 태그 추가

외부 이미지에 대해 태그를 추가할 수 있습니다.

프로세스

- 모든 태그 관련 작업에 **oc tag** 명령을 사용하여 내부 또는 외부 이미지를 가리키는 태그를 추가합니다.

```
$ oc tag <repository/image> <image-name:tag>
```

예를 들어 이 명령은 **python** 이미지 스트림에서 **docker.io/python:3.6.0** 이미지를 **3.6** 태그에 매핑합니다.

```
$ oc tag docker.io/python:3.6.0 python:3.6
```

출력 예

```
Tag python:3.6 set to docker.io/python:3.6.0.
```

외부 이미지에 보안이 설정되어 있으면 해당 레지스트리에 액세스하는 데 사용할 인증 정보가 포함된 시크릿을 생성해야 합니다.

6.7.4. 이미지 스트림 태그 업데이트

이미지 스트림의 다른 태그를 반영하도록 태그를 업데이트할 수 있습니다.

절차

- 태그를 업데이트합니다.

```
$ oc tag <image-name:tag> <image-name:latest>
```

예를 들어 다음에서는 이미지 스트림의 **3.6** 태그를 반영하도록 **latest** 태그를 업데이트합니다.

```
$ oc tag python:3.6 python:latest
```

출력 예

```
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

6.7.5. 이미지 스트림 태그 제거

이미지 스트림에서 이전 태그를 제거할 수 있습니다.

절차

- 이미지 스트림에서 이전 태그를 제거합니다.

```
$ oc tag -d <image-name:tag>
```

예를 들면 다음과 같습니다.

```
$ oc tag -d python:3.6
```

출력 예

```
Deleted tag default/python:3.6
```

Cluster Samples Operator에서 더 이상 사용되지 않는 이미지 스트림 태그를 처리하는 방법에 대한 자세한 내용은 Cluster Samples Operator에서 더 이상 사용되지 않는 이미지 스트림 태그 제거를 참조하십시오.

6.7.6. 주기적으로 이미지 스트림 태그 가져오기 구성

외부 컨테이너 이미지 레지스트리로 작업하는 경우 예를 들어 최신 보안 업데이트를 받기 위해 이미지를 정기적으로 다시 가져오려면 **--scheduled** 플래그를 사용할 수 있습니다.

프로세스

1. 이미지 가져오기를 스케줄링합니다.

```
$ oc tag <repository/image> <image-name:tag> --scheduled
```

예를 들면 다음과 같습니다.

```
$ oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

출력 예

```
Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

■

이 명령은 OpenShift Container Platform이 특정 이미지 스트림 태그를 주기적으로 업데이트하도록 합니다. 이 기간은 기본적으로 15분으로 설정되는 클러스터 전체 설정입니다.

2. 정기 점검을 없애고 위 명령을 다시 실행하되 **--scheduled** 플래그를 생략합니다. 이렇게 하면 동작이 기본값으로 재설정됩니다.

```
$ oc tag <repositiory/image> <image-name:tag>
```

6.8. 개인 레지스트리에서 이미지 및 이미지 스트림 가져 오기

인증이 필요한 개인 이미지 레지스트리에서 태그 및 이미지 메타 데이터를 가져 오도록 이미지 스트림을 구성할 수 있습니다. 이 절차는 Cluster Samples Operator가 콘텐츠를 registry.redhat.io 이외의 항목에서 가져오는 데 사용하는 레지스트리를 변경하는 경우에 적용됩니다.

참고

안전하지 않거나 보안된 레지스트리에서 가져올 때 시크릿에 정의 된 레지스트리 URL에는 **:80** 포트 접미사가 포함되어야 합니다. 그렇지 않으면 레지스트리에서 가져 오려고 할 때 시크릿을 사용할 수 없습니다.

프로세스

1. 다음 명령을 입력하여 인증 정보를 저장하는 데 사용되는**secret** 개체를 만들어야 합니다.

```
$ oc create secret generic <secret_name> --from-file=.dockerconfigjson=
<file_absolute_path> --type=kubernetes.io/dockerconfigjson
```

2. 시크릿을 구성한 후 새 이미지 스트림을 생성하거나 **oc import-image** 명령을 입력합니다.

```
$ oc import-image <imagestreamtag> --from=<image> --confirm
```

가져 오기 프로세스 중에 OpenShift Container Platform은 시크릿을 선택하여 원격 당사자에게 제공합니다.

6.8.1. Pod에서 다른 보안 레지스트리의 이미지를 참조하도록 허용

Docker 클라이언트의 **.dockercfg $HOME/.docker/config.json** 파일은 이전에 보안 또는 비보안 레지스트리에 로그인한 적이 있는 경우 인증 정보를 저장하는 Docker 인증 정보 파일입니다.

OpenShift 이미지 레지스트리가 아닌 보안 컨테이너 이미지를 가져오려면 Docker 인증 정보에서 풀 시크릿을 생성하여 서비스 계정에 추가해야 합니다.

Docker 인증 정보 파일과 연결된 풀 암호에는 각각 고유한 인증 정보 세트가 있는 동일한 레지스트리에 대한 여러 참조가 포함될 수 있습니다.

config.json 파일 예

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
```

```
      },
      "quay.io":{
        "auth":"b3Blb=",
        "email":"you@example.com"
      },
      "quay.io/repository-main":{
        "auth":"b3Blb=",
        "email":"you@example.com"
      }
    }
}
```

pull secret의 예

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

절차

- 보안 레지스트리의 **.dockercfg** 파일이 이미 있는 경우 다음을 실행하여 해당 파일에서 시크릿을 생성할 수 있습니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockercfg=<path/to/.dockercfg> \
    --type=kubernetes.io/dockercfg
```

- **$HOME/.docker/config.json** 파일이 있는 경우 다음을 실행합니다.

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

- 보안 레지스트리의 Docker 인증 정보 파일이 아직 없는 경우 다음을 실행하여 시크릿을 생성할 수 있습니다.

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

- Pod의 이미지 가져오기에 시크릿을 사용하려면 서비스 계정에 이 시크릿을 추가해야 합니다. 이 예제의 서비스 계정 이름은 Pod에서 사용하는 서비스 계정 이름과 일치해야 합니다. **default**는 기본 서비스 계정입니다.

```
$ oc secrets link default <pull_secret_name> --for=pull
```

6.9. IMAGESTREAMIMPORT를 통해 매니페스트 목록 가져오기

ImageStreamImport 리소스를 사용하여 다른 컨테이너 이미지 레지스트리에서 클러스터로 이미지 매니페스트를 찾아 가져올 수 있습니다. 개별 이미지 또는 전체 이미지 리포지토리를 가져올 수 있습니다.

다음 절차에 따라 **ImageStreamImport** 오브젝트를 통해 **importMode** 값이 있는 매니페스트 목록을 가져옵니다.

절차

1. **ImageStreamImport** YAML 파일을 생성하고 매니페스트 목록으로 가져올 태그에서 **importMode** 매개변수를 **PreserveOriginal** 로 설정합니다.

```
apiVersion: image.openshift.io/v1
kind: ImageStreamImport
metadata:
  name: app
  namespace: myapp
spec:
  import: true
  images:
  - from:
      kind: DockerImage
      name: <registry>/<user_name>/<image_name>
    to:
      name: latest
    referencePolicy:
      type: Source
    importPolicy:
      importMode: "PreserveOriginal"
```

2. 다음 명령을 실행하여 **ImageStreamImport** 를 생성합니다.

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

6.9.1. importMode 구성 필드

다음 표에서는 **importMode** 값에 사용할 수 있는 구성 필드를 설명합니다.

매개변수	설명

매개변수	설명
Legacy	**importMode** 의 기본값입니다. 활성화되면 매니페스트 목록이 삭제되고 단일 하위 매니페스트가 가져옵니다. 플랫폼은 다음과 같은 우선 순위 순서로 선택됩니다. 1. 태그 주석 2. 컨트롤 플레인 아키텍처 3. Linux/AMD64 4. 목록의 첫 번째 매니페스트
PreserveOriginal	활성 상태이면 원래 매니페스트가 유지됩니다. 매니페스트 목록의 경우 매니페스트 목록과 모든 하위 매니페스트를 가져옵니다.

7장. KUBERNETES 리소스가 포함된 이미지 스트림 사용

OpenShift Container Platform 네이티브 리소스인 이미지 스트림은 **Build** 또는 **DeploymentConfigs** 리소스와 같은 OpenShift Container Platform에서 사용할 수 있는 모든 네이티브 리소스와 함께 작동합니다. 또한 **Job**,**ReplicationController**,**ReplicaSet** 또는 Kubernetes **Deployment** 리소스와 같은 네이티브 Kubernetes 리소스와 함께 작동하도록 할 수도 있습니다.

7.1. KUBERNETES 리소스가 포함된 이미지 스트림 활성화

Kubernetes 리소스가 포함된 이미지 스트림을 사용하는 경우 리소스와 동일한 프로젝트에 상주하는 이미지 스트림만 참조할 수 있습니다. 이미지 스트림 참조는 **ruby:2.5**와 같은 단일 세그먼트 값으로 구성되어야 합니다. 여기서, **ruby**는 **2.5**라는 태그가 있고 참조하는 리소스와 동일한 프로젝트에 상주하는 이미지 스트림의 이름입니다.

참고

이 기능은 **default** 네임스페이스나 **openshift-** 또는 **kube-** 네임스페이스에서 사용할 수 없습니다.

Kubernetes 리소스가 포함된 이미지 스트림을 활성화하는 방법은 다음 두 가지가 있습니다.

- 특정 리소스에서 이미지 스트림 분석을 활성화합니다. 이렇게 하면 해당 리소스만 이미지 필드에서 이미지 스트림 이름을 사용할 수 있습니다.
- 이미지 스트림에서 이미지 스트림 해석을 활성화합니다. 이렇게 하면 해당 이미지 스트림을 가리키는 모든 리소스가 이미지 필드에서 그 이미지 스트림 이름을 사용할 수 있습니다.

절차

oc set image-lookup을 통해 특정 리소스에서 이미지 스트림 분석을 활성화하거나 이미지 스트림에서 이미지 스트림 분석을 활성화할 수 있습니다.

1. 모든 리소스가 **mysql**이라는 이미지 스트림을 참조하도록 허용하려면 다음 명령을 입력합니다.

```
$ oc set image-lookup mysql
```

이렇게 하면 **Imagestream.spec.lookupPolicy.local** 필드가 True로 설정됩니다.

이미지 조회가 활성화된 이미지 스트림

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/display-name: mysql
  name: mysql
  namespace: myproject
spec:
  lookupPolicy:
    local: true
```

활성화되는 경우 이미지 스트림 내 모든 태그에 대해 해당 동작이 활성화됩니다.

2. 그런 다음 이미지 스트림을 조회하여 옵션이 설정되었는지 확인할 수 있습니다.

```
$ oc set image-lookup imagestream --list
```

특정 리소스에서 이미지 조회를 활성화할 수 있습니다.

- **mysql**이라는 Kubernetes 배포가 이미지 스트림을 사용하도록 허용하려면 다음 명령을 실행합니다.

  ```
  $ oc set image-lookup deploy/mysql
  ```

  이렇게 하면 배포에 **alpha.image.policy.openshift.io/resolve-names** 주석이 설정됩니다.

  이미지 조회가 활성화된 배포

  ```
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: mysql
    namespace: myproject
  spec:
    replicas: 1
    template:
      metadata:
        annotations:
          alpha.image.policy.openshift.io/resolve-names: '*'
      spec:
        containers:
        - image: mysql:latest
          imagePullPolicy: Always
          name: mysql
  ```

이미지 조회를 비활성화할 수 있습니다.

- 이미지 조회를 비활성화하려면 **--enabled=false**를 전달합니다.

  ```
  $ oc set image-lookup deploy/mysql --enabled=false
  ```

8장. 이미지 스트림 변경 시 업데이트 트리거

이미지 스트림 태그가 새 이미지를 참조하도록 업데이트되면 OpenShift Container Platform은 이전 이미지를 사용하는 리소스로 새 이미지를 롤아웃하는 작업을 자동으로 수행할 수 있습니다. 이미지 스트림 태그를 참조하는 리소스 유형에 따라 다양한 방식으로 이 동작을 구성할 수 있습니다.

8.1. OPENSHIFT CONTAINER PLATFORM 리소스

이미지 스트림 태그를 변경하여 OpenShift Container Platform 배포 구성 및 빌드 구성이 자동으로 트리거될 수 있습니다. 트리거된 작업은 업데이트된 이미지 스트림 태그에서 참조하는 이미지의 새 값을 사용하여 실행할 수 있습니다.

8.2. KUBERNETES 리소스 트리거

Kubernetes 리소스에는 API 정의의 일부로 트리거 제어를 위한 필드 집합을 포함하는 배포 및 빌드 구성과 달리 트리거를 위한 필드가 없습니다. 대신 OpenShift Container Platform에서 주석을 사용하여 트리거를 요청할 수 있습니다.

주석은 다음과 같이 정의됩니다.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    image.openshift.io/triggers:
      [
       {
         "from": {
           "kind": "ImageStreamTag", (1)
           "name": "example:latest", (2)
           "namespace": "myapp" (3)
         },
         "fieldPath": "spec.template.spec.containers[?(@.name==\"web\")].image", (4)
         "paused": false (5)
       },
      # ...
      ]
# ...
```

1. 필수: **kind**는 트리거할 리소스이며 **ImageStreamTag**여야 합니다.
2. 필수: **name**은 이미지 스트림 태그의 이름이어야 합니다.
3. 선택 사항: **namespace**는 기본적으로 개체의 네임스페이스로 설정됩니다.
4. 필수: **fieldPath**는 변경할 JSON 경로입니다. 이 필드는 제한되어 ID 또는 인덱스로 컨테이너와 정확히 일치하는 JSON 경로식만 허용됩니다. Pod의 경우 JSON 경로는 **spec.containers[?(@.name='web')].image** 입니다.
5. 선택 사항: **paused**는 트리거가 일시 정지되었는지 여부이며 기본값은 **false**입니다. 이 트리거를 일시적으로 비활성화하려면 **paused**를 **true**로 설정합니다.

코어 Kubernetes 리소스 중 하나에 pod 템플릿과 이 주석이 모두 포함된 경우 OpenShift Container Platform은 트리거에서 참조하는 이미지 스트림 태그와 현재 연결된 이미지를 사용하여 개체의 업데이트를 시도합니다. 업데이트는 지정된 **fieldPath**에 대해 수행됩니다.

pod 템플릿 및 주석을 모두 포함하는 코어 Kubernetes 리소스의 예는 다음과 같습니다.

- **CronJobs**
- **Deployments**
- **StatefulSets**
- **DaemonSets**
- **Jobs**
- **ReplicationControllers**
- **Pods**

8.3. KUBERNETES 리소스에 이미지 트리거 설정

배포에 이미지 트리거를 추가할 때 **oc set triggers** 명령을 사용할 수 있습니다. 예를 들어 이 절차의 샘플 명령은 이미지 변경 트리거를 **example** 이라는 배포에 추가하여 **example:latest** 이미지 스트림 태그가 업데이트되면 배포 내의 **web** 컨테이너가 새 이미지 값으로 업데이트됩니다. 이 명령은 배포 리소스에 올바른 **image.openshift.io/triggers** 주석을 설정합니다.

절차

- **oc set triggers** 명령을 입력하여 Kubernetes 리소스를 트리거합니다.

```
$ oc set triggers deploy/example --from-image=example:latest -c web
```

트리거 주석이 있는 배포의 예

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    image.openshift.io/triggers: '[{"from":
{"kind":"ImageStreamTag","name":"example:latest"},"fieldPath":"spec.template.spec.containers[
?(@.name==\"container\")].image"}]'
# ...
```

배포를 일시 정지하지 않으면 이 pod 템플릿 업데이트로 인해 새 이미지 값으로 배포가 자동으로 수행됩니다.

9장. 이미지 구성 리소스

다음 절차에 따라 이미지 레지스트리를 구성합니다.

9.1. 이미지 컨트롤러 구성 매개변수

image.config.openshift.io/cluster 리소스에는 이미지를 처리하는 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 **cluster**입니다. **spec**에서는 다음 구성 매개변수를 제공합니다.

참고

DisableScheduledImport,**MaxImagesBulkImportedPerRepository**,**MaxScheduledImportsPerMinute**,**ScheduledImageImportMinimumIntervalSeconds**,**InternalRegistryHostname** 과 같은 매개변수는 구성할 수 없습니다.

매개변수	설명
allowedRegistriesForImport	일반 사용자가 이미지를 가져올 수 있는 컨테이너 이미지 레지스트리를 제한합니다. 이 목록은 유효한 이미지를 포함한다고 신뢰할 수 있으며 애플리케이션을 가져올 수 있도록 하려는 레지스트리로 설정합니다. 이미지를 생성할 권한이 있는 사용자 또는 API의 **ImageStreamMappings**는 이 정책의 영향을 받지 않습니다. 일반적으로 클러스터 관리자에게만 적절한 권한이 있습니다. 이 목록의 모든 요소에는 레지스트리 도메인 이름으로 지정된 레지스트리 위치가 포함되어 있습니다. **domainName**: 레지스트리의 도메인 이름을 지정합니다. 레지스트리에서 비표준 **80** 또는 **443** 포트를 사용하는 경우 도메인 이름에도 포트가 포함되어야 합니다. **insecure**: 비보안은 레지스트리가 안전한지 안전하지 않은지를 나타냅니다. 달리 지정하지 않으면 기본적으로 레지스트리는 안전한 것으로 간주됩니다.
additionalTrustedCA	**ImageStream import**, **pod image pull**, **openshift-image-registry pullthrough** 및 빌드 중에 신뢰해야 하는 추가 CA가 포함된 구성 맵에 대한 참조입니다. 이 구성 맵의 네임스페이스는 **openshift-config**입니다. 구성 맵 형식에서는 신뢰할 추가 레지스트리 CA마다 레지스트리 호스트 이름을 키로 사용하고 PEM으로 인코딩된 인증서를 값으로 사용합니다.
externalRegistryHostnames	기본 외부 이미지 레지스트리의 호스트 이름을 제공합니다. 외부 호스트 이름은 이미지 레지스트리가 외부에 노출되는 경우에만 설정되어야 합니다. 첫 번째 값은 이미지 스트림의 **publicDockerImageRepository** 필드에서 사용됩니다. 값은 **hostname[:port]** 형식이어야 합니다.

매개변수	설명
registrySources	빌드 및 pod 이미지에 액세스하는 경우 컨테이너 런타임에서 개별 레지스트리를 처리하는 방법을 결정할 구성이 포함되어 있습니다. 비보안 액세스 허용 여부를 예로 들 수 있습니다. 내부 클러스터 레지스트리에 대한 구성은 포함되어 있지 않습니다. **insecureRegistries**: 유효한 TLS 인증서가 없거나 HTTP 연결만 지원하는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. **blockedRegistries**: 이미지 풀 및 푸시 작업이 거부되는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. 다른 모든 레지스트리는 허용됩니다. **allowedRegistries**: 이미지 풀 및 푸시 작업을 허용하는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. 다른 모든 레지스트리는 차단됩니다. **containerRuntimeSearchRegistries**: 이미지의 짧은 이름을 사용하여 이미지 풀 및 푸시 작업을 허용하는 레지스트리입니다. 다른 모든 레지스트리는 차단됩니다. **blockedRegistries** 또는 **allowedRegistries**를 설정할 수 있으나 둘 다 설정할 수는 없습니다.

주의

allowedRegistries 매개변수가 정의되면 명시적으로 나열되지 않은 경우 **registry.redhat.io** 및 **quay.io** 레지스트리 및 기본 OpenShift 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 Pod 실패를 방지하기 위해 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리 및 **internalRegistryHostname**을 포함한 모든 레지스트리를 **allowedRegistries** 목록에 추가합니다. 연결 해제된 클러스터의 경우 미러 레지스트리도 추가해야 합니다.

image.config.openshift.io/cluster 리소스의 **상태** 필드에는 클러스터에서 관찰된 값이 들어 있습니다.

매개변수	설명
internalRegistryHostname	**internalRegistryHostname**을 제어하는 Image Registry Operator가 설정합니다. 기본 OpenShift 이미지 레지스트리의 호스트 이름을 설정합니다. 값은 **hostname[:port]** 형식이어야 합니다. 이전 버전과의 호환성을 위해 **OPENSHIFT_DEFAULT_REGISTRY** 환경 변수를 계속 사용할 수 있지만 이 설정을 통해 환경 변수가 재정의됩니다.
externalRegistryHostnames	Image Registry Operator가 설정하며, 이미지 레지스트리가 외부에 노출되는 경우 이미지 레지스트리의 외부 호스트 이름을 제공합니다. 첫 번째 값은 이미지 스트림의 **publicDockerImageRepository** 필드에서 사용됩니다. 값은 **hostname[:port]** 형식이어야 합니다.

9.2. 이미지 레지스트리 설정 구성

image.config.openshift.io/cluster 사용자 지정 리소스 (CR)를 편집하여 이미지 레지스트리 설정을 구성할 수 있습니다. 레지스트리 변경 사항이 **image.config.openshift.io/cluster** CR에 적용되는 경우 MCO(Machine Config Operator)는 다음과 같은 순차적 작업을 수행합니다.

1. 노드 차단
2. CRI-O를 다시 시작하여 변경 사항 적용
3. 노드 차단 해제

> **참고**
>
> MCO는 변경 사항을 감지하면 노드를 재시작하지 않습니다.

절차

1. 다음과 같이 **project.config.openshift.io/cluster** 사용자 정의 리소스를 편집합니다.

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

다음은 **image.config.openshift.io/cluster** CR의 예입니다.

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: (2)
    - domainName: quay.io
```

```
    insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. **Image**: 이미지 처리 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 **cluster**입니다.

2. **allowedRegistriesForImport**: 일반 사용자가 이미지를 가져올 수 있는 컨테이너 이미지 레지스트리를 제한합니다. 이 목록은 유효한 이미지를 포함한다고 신뢰할 수 있으며 애플리케이션을 가져올 수 있도록 하려는 레지스트리로 설정합니다. 이미지를 생성할 권한이 있는 사용자 또는 API의 **ImageStreamMappings**는 이 정책의 영향을 받지 않습니다. 일반적으로 클러스터 관리자에게만 적절한 권한이 있습니다.

3. **additionalTrustedCA**: 이미지 스트림 가져오기, pod 이미지 가져오기, **openshift-image-registry** 풀스루 및 빌드 중에 신뢰해야 하는 추가 CA (인증 기관)가 포함된 구성 맵에 대한 참조입니다. 이 구성 맵의 네임스페이스는 **openshift-config**입니다. 구성 맵 형식에서는 신뢰할 추가 레지스트리 CA마다 레지스트리 호스트 이름을 키로 사용하고 PEM 인증서를 값으로 사용합니다.

4. **registrySources**: 빌드 및 pod 이미지에 액세스할 때 컨테이너 런타임에서 개별 레지스트리를 허용하는지 여부를 결정하는 구성이 포함되어 있습니다. **allowedRegistries** 매개변수 또는 **blockedRegistries** 매개변수 중 하나를 설정할 수 있지만 둘 다 설정할 수는 없습니다. 이미지 단축 이름을 사용하는 레지스트리를 허용하는 비보안 레지스트리 또는 레지스트리에 대한 액세스를 허용할지 여부를 정의할 수도 있습니다. 이 예에서는 사용할 수 있는 레지스트리를 정의하는 **allowedRegistries** 매개변수를 사용합니다. 비보안 레지스트리 **insecure.com** 도 허용됩니다. **registrySources** 매개변수에 내부 클러스터 레지스트리에 대한 구성이 포함되어 있지 않습니다.

참고

allowedRegistries 매개변수가 정의되면 명시적으로 나열되지 않은 경우 registry.redhat.io 및 quay.io 레지스트리 및 기본 OpenShift 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 Pod 실패를 방지하기 위해 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리와 **internalRegistryHostname**을 **allowedRegistries** 목록에 추가해야 합니다. **registry.redhat.io** 및 **quay.io** 레지스트리를 **blockedRegistries** 목록에 추가하지 마십시오.

allowedRegistries, **blockedRegistries** 또는 **insecureRegistries** 매개변수를 사용하는 경우 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**.

가능한 보안 위험을 줄이려면 안전하지 않은 외부 레지스트리의 사용을 피해야합니다.

2. 변경 사항이 적용되었는지 확인하려면 노드를 나열합니다.

```
$ oc get nodes
```

출력 예

```
NAME                                         STATUS                     ROLES                AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker
65m   v1.25.4+77bec7a
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane
74m   v1.25.4+77bec7a
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane        75m
v1.25.4+77bec7a
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker               65m
v1.25.4+77bec7a
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker               63m
v1.25.4+77bec7a
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane        73m
v1.25.4+77bec7a
```

9.2.1. 특정 레지스트리 추가

image.config.openshift.io/cluster 사용자 정의 리소스(CR)를 편집하여 이미지 가져오기 및 푸시 작업에 허용되는 레지스트리 목록을 추가할 수 있습니다. OpenShift Container Platform은 이 CR에 대한 변경 사항을 클러스터의 모든 노드에 적용합니다.

이미지를 풀하거나 푸시할 때 컨테이너 런타임은 **image.config.openshift.io/cluster** CR에서 **registrySources** 매개변수 아래에 나열된 레지스트리를 검색합니다. **allowedRegistries** 매개변수 아래에 레지스트리 목록을 생성한 경우 컨테이너 런타임은 해당 레지스트리만 검색합니다. 목록에 없는 레지스트리는 차단됩니다.

> **주의**
>
> **allowedRegistries** 매개변수가 정의되면 명시적으로 나열되지 않은 경우 **registry.redhat.io** 및 **quay.io** 레지스트리 및 기본 OpenShift 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 Pod 실패를 방지하기 위해 사용자 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리와 **internalRegistryHostname** 을 **allowedRegistries** 목록에 추가합니다. 연결 해제된 클러스터의 경우 미러 레지스트리도 추가해야 합니다.

절차

1. 다음과 같이 **project.config.openshift.io/cluster** CR을 편집합니다.

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

다음은 허용 목록을 포함한 **image.config.openshift.io/cluster** CR의 예입니다.

```
apiVersion: config.openshift.io/v1
```

```
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: (1)
    allowedRegistries: (2)
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 빌드 및 pod 이미지에 액세스하는 경우 컨테이너 런타임에서 개별 레지스트리를 처리하는 방법을 결정할 구성이 포함되어 있습니다. 내부 클러스터 레지스트리에 대한 구성은 포함되어 있지 않습니다.
2. 이미지 가져오기 및 푸시 작업에 사용할 레지스트리를 지정하고 선택적으로 해당 레지스트리의 리포지토리를 지정합니다. 다른 모든 레지스트리는 차단됩니다.

참고

allowedRegistries 매개변수 또는 **blockedRegistries** 매개변수 중 하나를 설정할 수 있지만 둘 다 설정할 수는 없습니다.

MCO(Machine Config Operator)는 **image.config.openshift.io/cluster** 리소스를 통해 레지스트리에 대한 변경 사항을 확인합니다. MCO가 변경 사항을 감지하면 MCP(머신 구성 풀)의 노드에서 롤아웃을 트리거합니다. 허용되는 레지스트리 목록은 각 노드의 **/etc/containers/policy.json** 파일에서 이미지 서명 정책을 업데이트하는 데 사용됩니다. **/etc/containers/policy.json** 파일을 변경하려면 노드가 드레이닝할 필요가 없습니다.

검증

- 다음 명령을 입력하여 노드 목록을 가져옵니다.

```
$ oc get nodes
```

출력 예

```
NAME            STATUS   ROLES                  AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 다음 명령을 실행하여 노드에서 디버그 모드로 전환합니다.

```
$ oc debug node/<node_name>
```

2. 메시지가 표시되면 **chroot /host** 를 터미널에 입력합니다.

```
sh-4.4# chroot /host
```

3. 다음 명령을 입력하여 레지스트리가 정책 파일에 추가되었는지 확인합니다.

```
sh-5.1# cat /etc/containers/policy.json | jq '.'
```

다음 정책은 example.com, quay.io 및 registry.redhat.io 레지스트리의 이미지만 이미지 풀 및 푸시할 수 있음을 나타냅니다.

예 9.1. 이미지 서명 정책 파일 예

```
{
  "default":[
    {
      "type":"reject"
    }
  ],
  "transports":{
    "atomic":{
      "example.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "image-registry.openshift-image-registry.svc:5000":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "insecure.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "quay.io":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "reg4.io/myrepo/myapp:latest":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "registry.redhat.io":[
        {
          "type":"insecureAcceptAnything"
        }
      ]
    },
    "docker":{
      "example.com":[
        {
```

```
            "type":"insecureAcceptAnything"
          }
        ],
        "image-registry.openshift-image-registry.svc:5000":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "insecure.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "quay.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "reg4.io/myrepo/myapp:latest":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "registry.redhat.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      },
      "docker-daemon":{
        "":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      }
    }
  }
```

참고

클러스터에서 **registrySources.insecureRegistries** 매개변수를 사용하는 경우 비보안 레지스트리가 허용 목록에 포함되어 있는지 확인합니다.

예를 들면 다음과 같습니다.

```
spec:
  registrySources:
    insecureRegistries:
    - insecure.com
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com
    - image-registry.openshift-image-registry.svc:5000
```

9.2.2. 특정 레지스트리 차단

image.config.openshift.io/cluster CR(사용자 정의 리소스)을 편집하여 레지스트리 내의 모든 레지스트리를 차단할 수 있습니다. OpenShift Container Platform은 이 CR에 대한 변경 사항을 클러스터의 모든 노드에 적용합니다.

이미지를 풀하거나 푸시할 때 컨테이너 런타임은 **image.config.openshift.io/cluster** CR에서 **registrySources** 매개변수 아래에 나열된 레지스트리를 검색합니다. **blockedRegistries** 매개변수 아래에 레지스트리 목록을 생성한 경우 컨테이너 런타임에서 해당 레지스트리를 검색하지 않습니다. 다른 모든 레지스트리는 허용됩니다.

주의

Pod 실패를 방지하려면 사용자 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리를 **blockedRegistries** 목록에 추가하지 마십시오.

절차

1. 다음과 같이 **project.config.openshift.io/cluster** CR을 편집합니다.

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

다음은 차단 목록이 포함된 **image.config.openshift.io/cluster** 리소스의 예입니다.

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
```

```
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: (1)
    blockedRegistries: (2)
    - untrusted.com
    - reg1.io/myrepo/myapp:latest
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 빌드 및 pod 이미지에 액세스하는 경우 컨테이너 런타임에서 개별 레지스트리를 처리하는 방법을 결정할 구성이 포함되어 있습니다. 내부 클러스터 레지스트리에 대한 구성은 포함되어 있지 않습니다.
2. 이미지 가져오기 및 푸시 작업에 사용해서는 안 되는 레지스트리를 지정하고 선택적으로 해당 레지스트리에서 리포지토리를 지정합니다. 다른 모든 레지스트리는 허용됩니다.

참고

blockedRegistries 레지스트리 또는 **allowedRegistries** 레지스트리 중 하나를 설정할 수 있지만 둘 다 설정할 수는 없습니다.

MCO(Machine Config Operator)는 **image.config.openshift.io/cluster** 리소스를 통해 레지스트리에 대한 변경 사항을 확인합니다. MCO가 변경사항을 감지하면 노드를 비우고, 변경 사항을 적용한 다음 노드를 분리합니다. 노드가 **Ready** 상태가 되면 차단된 레지스트리에 대한 변경이 각 노드의 **/etc/containers/registries.conf** 파일에 표시됩니다.

검증

- 다음 명령을 입력하여 노드 목록을 가져옵니다.

```
$ oc get nodes
```

출력 예

```
NAME              STATUS   ROLES                  AGE   VERSION
<node_name>         Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 다음 명령을 실행하여 노드에서 디버그 모드로 전환합니다.

```
$ oc debug node/<node_name>
```

2. 메시지가 표시되면 **chroot /host** 를 터미널에 입력합니다.

```
sh-4.4# chroot /host
```

3. 다음 명령을 입력하여 레지스트리가 정책 파일에 추가되었는지 확인합니다.

```
sh-5.1# cat etc/containers/registries.conf
```

다음 예에서는 **untrusted.com** 레지스트리의 이미지가 이미지 풀 및 푸시를 위해 비활성화되어 있음을 나타냅니다.

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "untrusted.com"
  blocked = true
```

9.2.2.1. 페이로드 레지스트리 차단

미러링 구성에서 ICSP(**ImageContentSourcePolicy**) 오브젝트를 사용하여 연결이 끊긴 환경의 업스트림 페이로드 레지스트리를 차단할 수 있습니다. 다음 예제 절차에서는 **quay.io/openshift- payload** 페이로드 레지스트리를 차단하는 방법을 보여줍니다.

절차

1. ICSP(**ImageContentSourcePolicy**) 오브젝트를 사용하여 미러 구성을 생성하여 인스턴스의 레지스트리에 페이로드를 미러링합니다. 다음 예제 ICSP 파일은 **internal-mirror.io/openshift-payload** 를 미러링합니다.

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: my-icsp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - internal-mirror.io/openshift-payload
    source: quay.io/openshift-payload
```

2. 오브젝트가 노드에 배포된 후 **/etc/containers/registries.conf** 파일을 확인하여 미러 구성이 설정되었는지 확인합니다.

   출력 예

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

3. 다음 명령을 사용하여 **image.config.openshift.io** 사용자 정의 리소스 파일을 편집합니다.

```
$ oc edit image.config.openshift.io cluster
```

4. 페이로드 레지스트리를 차단하려면 **image.config.openshift.io** 사용자 정의 리소스 파일에 다음 구성을 추가합니다.

```
spec:
  registrySources:
    blockedRegistries:
     - quay.io/openshift-payload
```

검증

- 노드에서 **/etc/containers/registries.conf** 파일을 확인하여 업스트림 페이로드 레지스트리가 차단되었는지 확인합니다.

  출력 예

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  blocked = true
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

9.2.3. 안전하지 않은 레지스트리 허용

image.config.openshift.io/cluster 사용자 정의 리소스(CR)를 편집하여 비보안 레지스트리를 추가하거나 선택적으로 레지스트리 내에 개별 저장소를 추가할 수 있습니다. OpenShift Container Platform은 이 CR에 대한 변경 사항을 클러스터의 모든 노드에 적용합니다.

유효한 SSL 인증서를 사용하지 않거나 HTTPS 연결이 필요하지 않은 레지스트리는 안전하지 않은 레지스트리로 간주됩니다.

> **주의**
>
> 가능한 보안 위험을 줄이려면 안전하지 않은 외부 레지스트리의 사용을 피해야합니다.

절차

1. 다음과 같이 **project.config.openshift.io/cluster** CR을 편집합니다.

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

   다음은 안전하지 않은 레지스트리 목록이있는 **image.config.openshift.io/cluster** CR의 예입니다.

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
```

```
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: (1)
    insecureRegistries: (2)
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com (3)
    - reg4.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 빌드 및 pod 이미지에 액세스하는 경우 컨테이너 런타임에서 개별 레지스트리를 처리하는 방법을 결정할 구성이 포함되어 있습니다. 내부 클러스터 레지스트리에 대한 구성은 포함되어 있지 않습니다.
2. 안전하지 않은 레지스트리를 지정합니다. 해당 레지스트리에 리포지토리를 지정할 수 있습니다.
3. 안전하지 않은 모든 레지스트리가 **allowedRegistries** 목록에 포함되어 있는지 확인합니다.

> **참고**
>
> **allowedRegistries** 매개변수가 정의되면 명시적으로 나열되지 않은 경우 registry.redhat.io 및 quay.io 레지스트리 및 기본 OpenShift 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하면 Pod 실패를 방지하기 위해 사용자 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리와 **internalRegistryHostname** 을 포함하여 모든 레지스트리를 **allowedRegistries** 목록에 추가합니다. 연결 해제된 클러스터의 경우 미러 레지스트리도 추가해야 합니다.

MCO(Machine Config Operator)는 레지스트리에 대한 변경 사항이 있는지 **image.config.openshift.io/cluster**CR를 감시하고 변경 사항이 탐지되면 노드를 비우고 차단을 해제합니다. 노드가 **Ready** 상태가 되면 안전하지 않고 차단된 레지스트리에 대한 변경이 각 노드의 **/etc/containers/registries.conf** 파일에 표시됩니다.

검증

- 레지스트리가 정책 파일에 추가되었는지 확인하려면 노드에서 다음 명령을 사용하십시오.

```
$ cat /etc/containers/registries.conf
```

다음 예는 **insecure.com** 레지스트리의 이미지가 안전하지 않으며, 이미지 풀 및 푸시가 허용된다는 것을 나타냅니다.

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "insecure.com"
  insecure = true
```

9.2.4. 이미지의 단축 이름을 허용하는 레지스트리 추가

image.config.openshift.io/cluster 사용자 정의 리소스(CR)를 편집하여 이미지의 단축 이름을 검색하기 위해 레지스트리를 추가할 수 있습니다. OpenShift Container Platform은 이 CR에 대한 변경 사항을 클러스터의 모든 노드에 적용합니다.

이미지 단축 이름을 사용하면 가져오기 사양에 정규화된 도메인 이름을 포함하지 않고도 이미지를 검색할 수 있습니다. 예를 들어 **registry.access.redhat.com/rhe7/etcd** 대신 **rhel7/etcd**를 사용할 수 있습니다.

전체 경로를 사용하지 않는 경우 단축 이름을 사용할 수 있습니다. 예를 들어, 클러스터가 DNS가 자주 변경되는 여러 내부 레지스트리를 참조하는 경우 각 변경에 따라 풀 사양에서 정규화된 도메인 이름을 업데이트해야 합니다. 이 경우 이미지 단축 이름을 사용하는 것이 유용할 수 있습니다.

이미지를 풀하거나 푸시할 때 컨테이너 런타임은 **image.config.openshift.io/cluster** CR에서 **registrySources** 매개변수 아래에 나열된 레지스트리를 검색합니다. **containerRuntimeSearchRegistries** 매개변수 아래에 레지스트리 목록을 생성한 경우, 단축 이름으로 이미지를 가져오면 컨테이너 런타임에서 해당 레지스트리를 검색합니다.

주의

공용 레지스트리에 인증이 필요한 경우 이미지가 배포되지 않을 수 있으므로 공용 레지스트리에서 이미지 단축 이름을 사용하는 것이 좋습니다. 공용 레지스트리와 함께 정규화된 이미지 이름을 사용합니다.

Red Hat 내부 또는 개인 레지스트리는 일반적으로 이미지의 단축 이름을 사용할 수 있습니다.

containerRuntimeSearchRegistries 매개변수(**registry.redhat.io**,**docker.io** 및 **quay.io** 레지스트리 포함) 아래에 공용 레지스트리를 나열하는 경우 목록의 모든 레지스트리에 인증 정보를 노출하고 네트워크 및 레지스트리 공격이 발생할 위험이 있습니다. 글로벌 풀 시크릿에 정의된 대로 이미지를 가져오기 위해 하나의 풀 시크릿만 있을 수 있으므로 해당 시크릿은 해당 목록의 모든 레지스트리에 대해 인증하는 데 사용됩니다. 따라서 목록에 공용 레지스트리를 포함하는 경우 보안 위험이 발생합니다.

각 공개 레지스트리에 다른 인증 정보가 필요하고 클러스터에 글로벌 풀 시크릿의 공개 레지스트리가 나열되지 않는 경우 **containerRuntimeSearchRegistries** 매개변수 아래에 여러 개의 공용 레지스트리를 나열할 수 없습니다.

인증이 필요한 퍼블릭 레지스트리의 경우 레지스트리에 글로벌 풀 시크릿에 저장된 인증 정보가 있는 경우에만 이미지의 단축 이름을 사용할 수 있습니다.

MCO(Machine Config Operator)는 **image.config.openshift.io/cluster** 리소스를 통해 레지스트리에 대한 변경 사항을 확인합니다. MCO가 변경사항을 감지하면 노드를 비우고, 변경 사항을 적용한 다음 노드를 분리합니다. 노드가 **Ready** 상태가 된 후 **containerRuntimeSearchRegistries** 매개변수가 추가되면 MCO가 나열된 레지스트리를 사용하여 각 노드에 **/etc/containers/registries.conf.d** 디렉터리에 파일을 생성합니다. 파일은 **/etc/containers/registries.conf** 파일의 정규화되지 않은 검색 레지스트리 목록을 덮어씁니다. 정규화되지 않은 검색 레지스트리의 기본 목록으로 돌아갈 수 없습니다.

containerRuntimeSearchRegistries 매개변수는 Podman 및 CRI-O 컨테이너 엔진에서만 작동합니다. 목록의 레지스트리는 빌드 및 이미지 스트림에서 아닌 Pod 사양에서만 사용할 수 있습니다.

절차

1. 다음과 같이 **project.config.openshift.io/cluster** CR을 편집합니다.

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

다음은 **image.config.openshift.io/cluster** CR의 예입니다.

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    containerRuntimeSearchRegistries: (1)
    - reg1.io
    - reg2.io
    - reg3.io
    allowedRegistries: (2)
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io
    - reg2.io
    - reg3.io
    - image-registry.openshift-image-registry.svc:5000
...
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

(1) 이미지 단축 이름으로 사용할 레지스트리를 지정합니다. 가능한 보안 위험을 줄이기 위해 내부 또는 개인 레지스트리만 있는 이미지의 단축 이름을 사용해야 합니다.

2 **containerRuntimeSearchRegistries**에 나열된 모든 레지스트리가 **allowedRegistries** 목록에 포함되어 있는지 확인합니다.

> **참고**
>
> **allowedRegistries** 매개변수가 정의되면 명시적으로 나열되지 않은 경우 **registry.redhat.io** 및 **quay.io** 레지스트리 및 기본 OpenShift 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 Pod 실패를 방지하기 위해 사용자 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리와 **internalRegistryHostname** 을 **allowedRegistries** 목록에 포함한 모든 레지스트리를 추가합니다. 연결 해제된 클러스터의 경우 미러 레지스트리도 추가해야 합니다.

검증

- 다음 명령을 입력하여 노드 목록을 가져옵니다.

```
$ oc get nodes
```

출력 예

```
NAME              STATUS   ROLES                  AGE   VERSION
<node_name>         Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 다음 명령을 실행하여 노드에서 디버그 모드로 전환합니다.

```
$ oc debug node/<node_name>
```

2. 메시지가 표시되면 **chroot /host** 를 터미널에 입력합니다.

```
sh-4.4# chroot /host
```

3. 다음 명령을 입력하여 레지스트리가 정책 파일에 추가되었는지 확인합니다.

```
sh-5.1# cat /etc/containers/registries.conf.d/01-image-searchRegistries.conf
```

출력 예

```
unqualified-search-registries = ['reg1.io', 'reg2.io', 'reg3.io']
```

9.2.5. 이미지 레지스트리 액세스를 위한 추가 신뢰 저장소 구성

image.config.openshift.io/cluster 사용자 지정 리소스에는 이미지 레지스트리 액세스 중에 신뢰할 수 있는 추가 인증 기관이 포함된 구성 맵에 대한 참조가 포함될 수 있습니다.

사전 요구 사항

- 인증 기관(CA)은 PEM으로 인코딩되어야 합니다.

절차

openshift-config 네임 스페이스에 구성 맵을 만들고 **image.config.openshift.io** 사용자 지정 리소스에서 **AdditionalTrustedCA**의 해당 이름을 사용하여 외부 레지스트리에 연결할 때 신뢰할 수있는 추가 CA를 제공할 수 있습니다.

구성 맵 키는 이 CA가 신뢰할 수있는 포트가 있는 레지스트리의 호스트 이름이며 PEM 인증서 콘텐츠는 신뢰할 수 있는 각 추가 레지스트리 CA의 값입니다.

이미지 레지스트리 CA 구성 맵의 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | (1)
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1. 레지스트리에 **registry-with-port.example.com:5000** 같은 포트가 있는 경우 **:**이 **..**로 교체되어야 합니다.

다음 절차에 따라 추가 CA를 구성할 수 있습니다.

1. 추가 CA를 구성하려면 다음을 실행합니다.

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

9.2.6. 이미지 레지스트리 저장소 미러링 설정

컨테이너 레지스트리 저장소 미러링을 설정하면 다음을 수행할 수 있습니다.

- 소스 이미지 레지스트리의 저장소에서 이미지를 가져오기 위해 요청을 리디렉션하고 미러링된 이미지 레지스트리의 저장소에서 이를 해석하도록 OpenShift Container Platform 클러스터를 설정합니다.
- 하나의 미러가 다운된 경우 다른 미러를 사용할 수 있도록 각 대상 저장소에 대해 여러 미러링된 저장소를 확인합니다.

다음은 OpenShift Container Platform의 저장소 미러링의 몇 가지 속성입니다.

- 이미지 풀은 레지스트리 다운타임에 탄력적으로 대처할 수 있습니다.

- 연결이 끊긴 환경의 클러스터는 중요한 위치(예: quay.io)에서 이미지를 가져오고 회사 방화벽 뒤의 레지스트리에서 요청된 이미지를 제공하도록 할 수 있습니다.
- 이미지 가져오기 요청이 있으면 특정한 레지스트리 순서로 가져오기를 시도하며 일반적으로 영구 레지스트리는 마지막으로 시도합니다.
- 입력한 미러링 정보는 OpenShift Container Platform 클러스터의 모든 노드에서 **/etc/containers/registries.conf** 파일에 추가됩니다.
- 노드가 소스 저장소에서 이미지를 요청하면 요청된 컨텐츠를 찾을 때 까지 미러링된 각 저장소를 차례로 시도합니다. 모든 미러가 실패하면 클러스터는 소스 저장소를 시도합니다. 성공하면 이미지를 노드로 가져올 수 있습니다.

저장소 미러링은 다음과 같은 방법으로 설정할 수 있습니다.

- OpenShift Container Platform 설치 시
  OpenShift Container Platform에 필요한 컨테이너 이미지를 가져온 다음 해당 이미지를 회사 방화벽 뒤에 배치하면 연결이 끊긴 환경에 있는 데이터 센터에 OpenShift Container Platform을 설치할 수 있습니다.
- OpenShift Container Platform 설치 후
  OpenShift Container Platform 설치 시 미러링을 설정하지 않고 **ImageContentSourcePolicy** 개체를 사용하여 나중에 설정할 수 있습니다.

다음 절차에서는 다음을 식별하는 **ImageContentSourcePolicy** 오브젝트를 생성하는 사후 설치 미러 구성을 제공합니다.

- 미러링하려는 컨테이너 이미지 저장소의 소스
- 소스 저장소에서 요청된 컨텐츠를 제공하는 각 미러 저장소에 대한 개별 항목

참고

ImageContentSourcePolicy 개체가 있는 클러스터에 대한 글로벌 풀 시크릿만 구성할 수 있습니다. 프로젝트에 풀 시크릿을 추가할 수 없습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. 미러링된 저장소를 설정합니다.
   - Red Hat Quay Repository Mirroring 에 설명된대로 Red Hat Quay를 사용하여 미러링된 저장소를 설정합니다. Red Hat Quay를 사용하면 한 저장소에서 다른 저장소로 이미지를 복사하고 시간이 지남에 따라 해당 저장소를 반복해서 자동으로 동기화할 수 있습니다.
   - **skopeo**와 같은 툴을 사용하여 소스 디렉토리에서 미러링된 저장소로 이미지를 수동으로 복사합니다.
     예를 들어, Red Hat Enterprise Linux(RHEL) 7 또는 RHEL 8 시스템에 skopeo RPM 패키지를 설치한 후 다음 예와 같이 **skopeo** 명령을 사용합니다.

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi8/ubi-
     minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
     ```

```
a6 \
docker://example.io/example/ubi-minimal
```

이 예제에는 **example.io**라는 컨테이너 이미지 레지스트리가 있으며, **registry.access.redhat.com**에서 **ubi8/ubi-minimal** 이미지를 복사할 **example**이라는 이미지 저장소가 있습니다. 레지스트리를 생성한 후 OpenShift Container Platform 클러스터를 설정하여 소스 저장소의 요청을 미러링된 저장소로 리디렉션할 수 있습니다.

2. OpenShift Container Platform 클러스터에 로그인합니다.

3. **ImageContentSourcePolicy** 파일(예: **registryrepomirror.yaml**)을 생성하고 소스 및 미러를 특정 레지스트리 및 저장소 쌍과 이미지로 교체합니다.

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal (1)
    - example.com/example/ubi-minimal (2)
    source: registry.access.redhat.com/ubi8/ubi-minimal (3)
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 (4)
  - mirrors:
    - mirror.example.com
    source: registry.redhat.io (5)
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage (6)
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example (7)
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com (8)
```

1. 이미지 레지스트리 및 저장소의 이름을 가리킵니다.

2. 각 대상 저장소에 대해 여러 미러 리포지토리를 나타냅니다. 하나의 미러가 다운된 경우 대상 저장소에서 다른 미러를 사용할 수 있습니다.

3. 미러링된 컨텐츠를 포함하는 레지스트리 및 저장소를 가리킵니다.

4. 해당 네임스페이스의 이미지를 사용하도록 레지스트리 내에서 네임스페이스를 구성할 수 있습니다. 레지스트리 도메인을 소스로 사용하는 경우 **ImageContentSourcePolicy** 리소스가 레지스트리의 모든 리포지토리에 적용됩니다.

5. 레지스트리 이름을 구성하면 **ImageContentSourcePolicy** 리소스가 소스 레지스트리에서 미러 레지스트리로 모든 리포지토리에 적용됩니다.

6. **mirror.example.net/image@sha256:...** 이미지를 가져옵니다.

7. 미러 **mirror.example.net/myimage@sha256:...** 에서 소스 레지스트리 네임스페이스의 **myimage** 이미지를 가져옵니다.

8. 미러 레지스트리 **mirror.example.net/registry-example-com/example/myimage@sha256:...** 에서 이미지 **registry.example.com/example/myimage**를 가져옵니다. **ImageContentSourcePolicy** 리소스는 소스 레지스트리에서 미러 레지스트리 **mirror.example.net/registry-example-com** 으로 모든 리포지토리에 적용됩니다.

4. 새 **ImageContentSourcePolicy** 개체를 생성합니다.

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

ImageContentSourcePolicy 개체가 생성된 후 새 설정이 각 노드에 배포된 클러스터는 소스 저장소에 대한 요청에 미러링된 저장소를 사용하기 시작합니다.

5. 미러링된 설정이 적용되었는지 확인하려면 노드 중 하나에서 다음을 수행하십시오.

   a. 노드를 나열합니다.

```
$ oc get node
```

출력 예

```
NAME                          STATUS                ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-138-148.ec2.internal   Ready                     master   11m  v1.25.0
ip-10-0-139-122.ec2.internal   Ready                     master   11m  v1.25.0
ip-10-0-147-35.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-153-12.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-154-10.ec2.internal    Ready                    master   11m  v1.25.0
```

Imagecontentsourcepolicy 리소스는 노드를 재시작하지 않습니다.

   b. 디버깅 프로세스를 시작하고 노드에 액세스합니다.

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

출력 예

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

   c. 루트 디렉토리를 **/host** 로 변경합니다.

```
sh-4.2# chroot /host
```

   d. **/etc/containers/registries.conf** 파일을 체크하여 변경 사항이 적용되었는지 확인합니다.

```
sh-4.2# cat /etc/containers/registries.conf
```

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi8/ubi-minimal"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal"

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"

[[registry]]
  prefix = ""
  location = "registry.example.com"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net"

[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net/image"

[[registry]]
  prefix = ""
  location = "registry.redhat.io"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.com"

[[registry]]
  prefix = ""
  location = "registry.redhat.io/openshift4"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
```

e. 소스의 이미지 다이제스트를 노드로 가져와 실제로 미러링에 의해 해결되는지 확인합니다. **ImageContentSourcePolicy** 개체는 이미지 태그가 아닌 이미지 다이제스트만 지원합니다.

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6
```

저장소 미러링 문제 해결

저장소 미러링 절차가 설명대로 작동하지 않는 경우 저장소 미러링 작동 방법에 대한 다음 정보를 사용하여 문제를 해결하십시오.

- 가져온 이미지는 첫 번째 작동 미러를 사용하여 공급합니다.
- 주요 레지스트리는 다른 미러가 작동하지 않는 경우에만 사용됩니다.
- 시스템 컨텍스트에서 **Insecure** 플래그가 폴백으로 사용됩니다.
- **/etc/containers/registries.conf** 파일 형식이 최근에 변경되었습니다. 현재 버전은 TOML 형식의 버전 2입니다.

추가 리소스

- 글로벌 풀 시크릿에 대한 자세한 내용은 글로벌 클러스터 풀 시크릿 업데이트를 참조하십시오.

10장. 템플릿 사용

다음 섹션에서는 템플릿에 대한 개요와 템플릿 사용 및 생성 방법을 알려줍니다.

10.1. 템플릿 이해

템플릿은 오브젝트 세트를 설명합니다. 이러한 오브젝트를 매개변수화하고 처리하여 OpenShift Container Platform에서 생성할 오브젝트 목록을 만들 수 있습니다. 서비스, 빌드 구성, 배포 구성과 같이 프로젝트 내에서 생성할 권한이 있는 모든 항목을 템플릿을 수정하여 생성할 수 있습니다. 템플릿은 템플릿에 정의된 모든 오브젝트에 적용할 레이블 세트를 정의할 수도 있습니다.

CLI를 사용하거나 템플릿을 프로젝트 또는 글로벌 템플릿 라이브러리에 업로드한 경우 웹 콘솔을 사용하여 템플릿에서 오브젝트 목록을 생성할 수 있습니다.

10.2. 템플릿 업로드

템플릿을 정의하는 JSON 또는 YAML 파일이 있는 경우 CLI를 사용하여 템플릿을 프로젝트에 업로드할 수 있습니다. 그러면 해당 프로젝트에 대한 적절한 액세스 권한이 있는 사용자가 반복해서 사용할 수 있도록 템플릿이 프로젝트에 저장됩니다. 자체 템플릿 작성에 대한 지침은 이 항목의 뒷부분에 제공됩니다.

프로세스

- 다음 방법 중 하나를 사용하여 템플릿을 업로드합니다.
  - 템플릿을 현재 프로젝트의 템플릿 라이브러리에 업로드하고 다음 명령으로 JSON 또는 YAML 파일을 전달합니다.

    ```
    $ oc create -f <filename>
    ```

  - 프로젝트 이름과 **-n** 옵션을 사용하여 템플릿을 다른 프로젝트에 업로드합니다.

    ```
    $ oc create -f <filename> -n <project>
    ```

이제 웹 콘솔 또는 CLI를 사용하여 템플릿을 선택할 수 있습니다.

10.3. 웹 콘솔을 사용하여 애플리케이션 생성

웹 콘솔을 사용하여 템플릿에서 애플리케이션을 생성할 수 있습니다.

프로세스

1. 웹 콘솔 탐색 메뉴 위쪽의 컨텍스트 선택기에서 **개발자**를 선택합니다.
2. 원하는 프로젝트에서 **+추가**를 클릭합니다.
3. **개발자 카탈로그** 타일에서 **모든 서비스를** 클릭합니다.
4. 유형에서 **빌더** 이미지를 클릭하여 사용 가능한 빌더 이미지를 확인합니다.

참고

여기 설명된 것처럼, 주석에 builder 태그가 표시된 이미지 스트림 태그만 이 목록에 나타납니다.

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
# ...
  tags:
    - name: "2.6"
      annotations:
        description: "Build and run Ruby 2.6 applications"
        iconClass: "icon-ruby"
        tags: "builder,ruby" (1)
        supports: "ruby:2.6,ruby"
        version: "2.6"
# ...
```

1. 여기에 빌더를 포함하면 이 이미지 스트림 태그는 웹 콘솔에 빌더로 표시됩니다.

5. 새 애플리케이션 화면에서 설정을 수정하여 애플리케이션을 지원하도록 오브젝트를 구성합니다.

10.4. CLI를 사용하여 템플릿에서 오브젝트 생성

CLI를 사용하여 템플릿을 처리하고 생성된 구성을 사용하여 오브젝트를 생성할 수 있습니다.

10.4.1. 레이블 추가

레이블은 pod 같은 생성된 오브젝트를 관리하고 구성하는 데 사용됩니다. 템플릿에 지정된 레이블은 템플릿에서 생성된 모든 오브젝트에 적용됩니다.

프로세스

- 명령줄에서 템플릿에 레이블을 추가합니다.

```
$ oc process -f <filename> -l name=otherLabel
```

10.4.2. 매개변수 나열

재정의할 수 있는 매개변수 목록은 템플릿의 **parameters** 섹션에 나열되어 있습니다.

프로세스

1. 다음 명령을 사용하여 사용할 파일을 지정하면 CLI로 매개변수를 나열할 수 있습니다.

```
$ oc process --parameters -f <filename>
```

또는 템플릿이 이미 업로드된 경우 다음을 실행합니다.

```
$ oc process --parameters -n <project> <template_name>
```

예를 들어 다음은 기본 **openshift** 프로젝트에서 퀵 스타트 템플릿 중 하나에 대한 매개변수를 나열하는 경우의 출력을 보여줍니다.

```
$ oc process --parameters -n openshift rails-postgresql-example
```

출력 예

```
NAME                    DESCRIPTION
GENERATOR          VALUE
SOURCE_REPOSITORY_URL       The URL of the repository with your application
source code                                                  https://github.com/sclorg/rails-ex.git
SOURCE_REPOSITORY_REF       Set this to a branch name, tag or other ref of your
repository if you are not using the default branch
CONTEXT_DIR                 Set this to the relative path to your project if it is not in the
root of your repository
APPLICATION_DOMAIN          The exposed hostname that will route to the Rails
service                                                      rails-postgresql-
example.openshiftapps.com
GITHUB_WEBHOOK_SECRET       A secret string used to configure the GitHub
webhook                                             expression          [a-zA-Z0-9]{40}
SECRET_KEY_BASE             Your secret key for verifying the integrity of signed
cookies                                             expression          [a-z0-9]{127}
APPLICATION_USER            The application user that is used within the sample
application to authorize access on pages                            openshift
APPLICATION_PASSWORD        The application password that is used within the
sample application to authorize access on pages                          secret
DATABASE_SERVICE_NAME       Database service name
postgresql
POSTGRESQL_USER             database username
expression          user[A-Z0-9]{3}
POSTGRESQL_PASSWORD         database password
expression          [a-zA-Z0-9]{8}
POSTGRESQL_DATABASE         database name
root
POSTGRESQL_MAX_CONNECTIONS   database max connections
10
POSTGRESQL_SHARED_BUFFERS    database shared buffers
12MB
```

출력에서는 템플릿이 처리될 때 생성기와 같이 정규식으로 생성되는 여러 매개변수를 식별합니다.

10.4.3. 오브젝트 목록 생성

CLI를 사용하면 템플릿 정의 파일을 처리하여 오브젝트 목록을 표준 출력으로 반환할 수 있습니다.

프로세스

1. 템플릿 정의 파일을 처리하여 오브젝트 목록을 표준 출력으로 반환합니다.

```
$ oc process -f <filename>
```

또는 템플릿이 현재 프로젝트에 이미 업로드된 경우 다음을 실행합니다.

```
$ oc process <template_name>
```

2. 템플릿을 처리하고 출력을 **oc create**로 파이핑하여 템플릿에서 오브젝트를 생성합니다.

```
$ oc process -f <filename> | oc create -f -
```

또는 템플릿이 현재 프로젝트에 이미 업로드된 경우 다음을 실행합니다.

```
$ oc process <template> | oc create -f -
```

3. 재정의하려는 각 **<name>=<value>** 쌍에 **-p** 옵션을 추가하여 파일에 정의된 매개변수 값을 재정의할 수 있습니다. 매개변수 참조는 템플릿 항목 내의 텍스트 필드에 표시될 수 있습니다.
예를 들어 다음에서는 템플릿의 **POSTGRESQL_USER** 및 **POSTGRESQL_DATABASE** 매개변수가 재정의되어 사용자 정의된 환경 변수가 있는 구성을 출력합니다.

   a. 템플릿에서 오브젝트 목록을 생성합니다.

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase
```

   b. JSON 파일은 처리된 출력을 **oc create** 명령으로 파이핑하여 템플릿을 업로드하지 않고 직접 적용하거나 파일로 리디렉션할 수 있습니다.

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase \
    | oc create -f -
```

   c. 많은 수의 매개변수가 있는 경우 파일에 저장한 후 해당 파일을 **oc process**로 전달할 수 있습니다.

```
$ cat postgres.env
POSTGRESQL_USER=bob
POSTGRESQL_DATABASE=mydatabase
```

```
$ oc process -f my-rails-postgresql --param-file=postgres.env
```

   d. **"-"**를 **--param-file**의 인수로 사용하여 표준 출력에서 환경을 읽을 수도 있습니다.

```
$ sed s/bob/alice/ postgres.env | oc process -f my-rails-postgresql --param-file=-
```

10.5. 업로드된 템플릿 수정

프로젝트에 이미 업로드된 템플릿을 편집할 수 있습니다.

프로세스

- 이미 업로드된 템플릿을 수정합니다.

```
$ oc edit template <template>
```

10.6. 인스턴트 앱 및 빠른 시작 템플릿 사용

OpenShift Container Platform은 다양한 기본 즉시 앱 및 퀵 스타트 템플릿을 제공하므로 다양한 언어를

위한 새 애플리케이션 생성을 쉽게 시작할 수 있습니다. Rails(Ruby), Django(Python), Node.js, CakePHP(PHP) 및 Dancer(Perl)에 대한 템플릿이 제공됩니다. 클러스터 관리자가 기본 글로벌 **openshift** 프로젝트에서 이러한 템플릿을 생성한 경우 해당 템플릿에 액세스할 수 있습니다.

기본적으로 템플릿은 필요한 애플리케이션 코드가 포함된 GitHub의 공용 소스 리포지터리를 사용하여 빌드합니다.

절차

1. 다음을 사용하여 사용 가능한 기본 즉시 앱 및 퀵 스타트 템플릿을 나열할 수 있습니다.

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

2. 소스를 수정하고 자체 애플리케이션 버전을 빌드하려면 다음을 수행합니다.

   a. 템플릿의 기본 **SOURCE_REPOSITORY_URL** 매개변수에서 참조하는 리포지터리를 포크합니다.

   b. 템플릿에서 생성하는 경우 기본값 대신 포크를 지정하여 **SOURCE_REPOSITORY_URL** 매개변수 값을 재정의합니다.
   이렇게 하면 템플릿에 의해 생성된 빌드 구성이 이제 애플리케이션 코드의 포크를 가리키므로 코드를 수정하고 원하는 대로 애플리케이션을 다시 빌드할 수 있습니다.

참고

일부 인스턴트 앱 및 퀵 스타트 템플릿은 데이터베이스 배포 구성을 정의합니다. 정의된 구성은 데이터베이스 컨텐츠에 ephemeral 스토리지를 사용합니다. 어떤 이유로든 데이터베이스 pod가 다시 시작되면 데이터베이스 데이터가 모두 손실되므로 이러한 템플릿은 설명용으로만 사용해야 합니다.

10.6.1. 퀵 스타트 템플릿

퀵 스타트 템플릿은 OpenShift Container Platform에서 실행되는 애플리케이션의 기본 예입니다. 퀵 스타트는 다양한 언어와 프레임워크로 제공되며 템플릿 내에 정의되어 있으며 일련의 서비스, 빌드 구성 및 배포 구성으로 구성됩니다. 이 템플릿은 애플리케이션을 빌드하고 배포하는 데 필요한 이미지 및 소스 리포지터리를 참조합니다.

퀵 스타트를 살펴보려면 템플릿에서 애플리케이션을 생성합니다. 관리자가 이미 OpenShift Container Platform 클러스터에 이러한 템플릿을 이미 설치했을 수 있으며, 이 경우 간단히 웹 콘솔에서 선택할 수 있습니다.

퀵 스타트는 애플리케이션 소스 코드가 포함된 소스 리포지터리를 나타냅니다. 퀵 스타트를 사용자 정의하려면 리포지터리를 포크하고 템플릿에서 애플리케이션을 생성할 때 기본 소스 리포지토리 이름을 분기된 리포지터리로 대체합니다. 그러면 제공된 소스 예 대신 소스 코드를 사용하여 수행되는 빌드가 생성됩니다. 그런 다음, 소스 리포지터리에서 코드를 업데이트하고 새 빌드를 시작하여 배포된 애플리케이션에 변경 사항이 반영된 것을 확인할 수 있습니다.

10.6.1.1. 웹 프레임워크 퀵 스타트 템플릿

이러한 퀵 스타트 템플릿은 표시된 프레임워크 및 언어의 기본 애플리케이션을 제공합니다.

- CakePHP: PHP 웹 프레임워크(MySQL 데이터베이스 포함)
- Dancer: Perl 웹 프레임워크(MySQL 데이터베이스 포함)

- Django: Python 웹 프레임워크(PostgreSQL 데이터베이스 포함)
- NodeJS: NodeJS 웹 애플리케이션(MongoDB 데이터베이스 포함)
- Rails: Ruby 웹 프레임워크(PostgreSQL 데이터베이스 포함)

10.7. 템플릿 작성

애플리케이션의 모든 오브젝트를 쉽게 다시 생성할 수 있도록 새 템플릿을 정의할 수 있습니다. 템플릿은 해당 개체의 생성을 안내하는 일부 메타데이터와 함께 생성되는 오브젝트를 정의합니다.

다음은 간단한 템플릿 오브젝트 정의의 예입니다(YAML).

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: redis-template
  annotations:
    description: "Description"
    iconClass: "icon-redis"
    tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
parameters:
- description: Password used for Redis authentication
  from: '[A-Z0-9]{8}'
  generate: expression
  name: REDIS_PASSWORD
labels:
  redis: master
```

10.7.1. 템플릿 설명 작성

템플릿 설명은 템플릿의 기능을 사용자에게 알려주고 웹 콘솔에서 검색할 때 템플릿을 찾도록 도와줍니다. 템플릿 이름 이외의 추가 메타데이터는 선택사항이지만 있으면 유용합니다. 메타데이터에는 일반적인 설명 정보 외에도 태그 세트가 포함되어 있습니다. 유용한 태그에는 템플릿과 관련된 언어의 이름이 포함되어 있습니다(예: Java, PHP, Ruby 등).

다음은 템플릿 설명 메타데이터의 예입니다.

```
kind: Template
apiVersion: template.openshift.io/v1
```

```
metadata:
  name: cakephp-mysql-example (1)
  annotations:
    openshift.io/display-name: "CakePHP MySQL Example (Ephemeral)" (2)
    description: >-
      An example CakePHP application with a MySQL database. For more information
      about using this template, including OpenShift considerations, see
      https://github.com/sclorg/cakephp-ex/blob/master/README.md.


      WARNING: Any data stored will be lost upon pod destruction. Only use this
      template for testing." (3)
    openshift.io/long-description: >-
      This template defines resources needed to develop a CakePHP application,
      including a build configuration, application DeploymentConfig, and
      database DeploymentConfig.  The database is stored in
      non-persistent storage, so this configuration should be used for
      experimental purposes only. (4)
    tags: "quickstart,php,cakephp" (5)
    iconClass: icon-php (6)
    openshift.io/provider-display-name: "Red Hat, Inc." (7)
    openshift.io/documentation-url: "https://github.com/sclorg/cakephp-ex" (8)
    openshift.io/support-url: "https://access.redhat.com" (9)
message: "Your admin credentials are ${ADMIN_USERNAME}:${ADMIN_PASSWORD}" (10)
```

1. 템플릿의 고유한 이름입니다.
2. 사용자 인터페이스에서 사용할 수 있는 간단하고 사용자에게 친숙한 이름입니다.
3. 템플릿에 대한 설명입니다. 사용자가 배포 사항을 이해할 수 있도록 충분한 세부 정보와 배포 전에 알아야 할 경고 사항을 포함합니다. README 파일과 같은 추가 정보에 대한 링크도 제공해야 합니다. 단락을 생성하기 위해 줄 바꿈이 포함될 수 있습니다.
4. 추가 템플릿 설명입니다. 예를 들어 서비스 카탈로그에 의해 표시될 수 있습니다.
5. 검색 및 그룹화에 필요한 템플릿과 연관된 태그입니다. 제공된 카탈로그 카테고리 중 하나에 포함할 태그를 추가합니다. 콘솔 상수 파일에 있는 **CATALOG_CATEGORIES**의 **id** 및 **categoryAliases**를 참조합니다. 카테고리는 전체 클러스터에 맞게 사용자 정의할 수도 있습니다.
6. 웹 콘솔에서 템플릿과 함께 표시되는 아이콘입니다.

예 10.1. 사용 가능한 아이콘

- **icon-3scale**
- **icon-aerogear**
- **icon-amq**
- **icon-angularjs**
- **icon-ansible**
- **icon-apache**

- **icon-beaker**
- **icon-camel**
- **icon-capedwarf**
- **icon-cassandra**
- **icon-catalog-icon**
- **icon-clojure**
- **icon-codeigniter**
- **icon-cordova**
- **icon-datagrid**
- **icon-datavirt**
- **icon-debian**
- **icon-decisionserver**
- **icon-django**
- **icon-dotnet**
- **icon-drupal**
- **icon-eap**
- **icon-elastic**
- **icon-erlang**
- **icon-fedora**
- **icon-freebsd**
- **icon-git**
- **icon-github**
- **icon-gitlab**
- **icon-glassfish**
- **icon-go-gopher**
- **icon-golang**
- **icon-grails**
- **icon-hadoop**
- **icon-haproxy**

- **icon-helm**
- **icon-infinispan**
- **icon-jboss**
- **icon-jenkins**
- **icon-jetty**
- **icon-joomla**
- **icon-jruby**
- **icon-js**
- **icon-knative**
- **icon-kubevirt**
- **icon-laravel**
- **icon-load-balancer**
- **icon-mariadb**
- **icon-mediawiki**
- **icon-memcached**
- **icon-mongodb**
- **icon-mssql**
- **icon-mysql-database**
- **icon-nginx**
- **icon-nodejs**
- **icon-openjdk**
- **icon-openliberty**
- **icon-openshift**
- **icon-openstack**
- **icon-other-linux**
- **Ironic-other-known**
- **icon-perl**
- **icon-phalcon**
- **icon-php**

- **icon-play**
- **iconpostgresql**
- **icon-processserver**
- **icon-python**
- **icon-quarkus**
- **icon-rabbitmq**
- **icon-rails**
- **icon-redhat**
- **icon-redis**
- **icon-rh-integration**
- **icon-rh-spring-boot**
- **icon-rh-tomcat**
- **icon-ruby**
- **icon-scala**
- **icon-serverlessfx**
- **icon-shadowman**
- **icon-spring-boot**
- **icon-spring**
- **icon-sso**
- **icon-stackoverflow**
- **icon-suse**
- **icon-symfony**
- **icon-tomcat**
- **icon-ubuntu**
- **icon-vertx**
- **icon-wildfly**
- **icon-windows**
- **icon-wordpress**
- **icon-xamarin**

```
    - icon-zend
```

7. 템플릿을 제공하는 사람 또는 조직의 이름입니다.
8. 템플릿에 대한 추가 문서를 참조하는 URL입니다.
9. 템플릿에 대한 지원을 받을 수 있는 URL입니다.
10. 이 템플릿이 인스턴스화될 때 표시되는 지시 메시지입니다. 이 필드는 새로 생성된 리소스 사용 방법을 사용자에게 알려주어야 합니다. 생성된 인증 정보 및 기타 매개변수가 출력에 포함될 수 있도록 표시 전에 메시지에서 매개변수 대체가 수행됩니다. 사용자가 따라야 하는 다음 단계 문서에 대한 링크를 포함합니다.

10.7.2. 템플릿 레이블 작성

템플릿에는 레이블 세트가 포함될 수 있습니다. 이러한 레이블은 템플릿이 인스턴스화될 때 생성되는 각 오브젝트에 추가됩니다. 이런 방식으로 레이블을 정의하면 사용자가 특정 템플릿에서 생성된 모든 오브젝트를 쉽게 찾아서 관리할 수 있습니다.

다음은 템플릿 오브젝트 레이블의 예입니다.

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" (1)
  app: "${NAME}" (2)
```

1. 이 템플릿에서 생성된 모든 오브젝트에 적용되는 레이블입니다.
2. 이 템플릿에서 생성된 모든 오브젝트에 적용되는 매개변수화된 레이블입니다. 매개변수 확장은 레이블 키와 값 둘 다에서 수행됩니다.

10.7.3. 템플릿 매개변수 작성

매개 변수를 사용하면 템플릿을 인스턴스화할 때 값을 제공하거나 생성할 수 있습니다. 그러면 해당 값이 매개변수가 참조될 때마다 대체됩니다. 참조는 오브젝트 목록 필드의 어떤 필드에서든 정의할 수 있습니다. 임의의 암호를 생성하거나 템플릿을 사용자 정의하는 데 필요한 호스트 이름 또는 기타 사용자 특정 값을 제공할 수 있는 데 유용합니다. 매개변수는 다음 두 가지 방법으로 참조할 수 있습니다.

- 템플릿에 있는 임의의 문자열 필드에 **${PARAMETER_NAME}** 형식의 값을 배치하여 문자열 값으로 참조합니다.
- 템플릿에서 임의의 필드 대신 **${{PARAMETER_NAME}}** 형식의 값을 배치하여 JSON 또는 YAML 값으로 참조합니다.

${PARAMETER_NAME}** 구문을 사용하는 경우 여러 매개변수 참조가 단일 필드에서 결합될 수 있으며 참조는 **"http://${PARAMETER_1}${PARAMETER_2}" 같이 고정된 데이터에 내에 포함될 수 있습니다. 매개변수 값이 둘 다 대체되며 결과 값은 인용된 문자열이 됩니다.

${{PARAMETER_NAME}} 구문을 사용하는 경우 단일 매개변수 참조만 허용되며 선행 및 후행 문자는 허용되지 않습니다. 대체가 수행된 후 결과가 유효한 JSON 오브젝트인 경우 결과 값이 인용되지 않습니다. 결과가 유효한 JSON 값이 아닌 경우 결과 값이 인용되고 표준 문자열로 처리됩니다.

단일 매개변수는 템플릿 내에서 여러 번 참조될 수 있으며 단일 템플릿 내에서 두 대체 구문을 사용하여 참조될 수도 있습니다.

다른 값을 제공하지 않은 경우 사용되는 기본값을 제공할 수 있습니다.

다음은 명시적 값을 기본값으로 설정하는 예입니다.

```
parameters:
  - name: USERNAME
    description: "The user name for Joe"
    value: joe
```

매개변수 값은 다음과 같이 매개변수 정의에 지정된 규칙을 기반으로 생성될 수도 있습니다(예: 매개변수 값 생성).

```
parameters:
  - name: PASSWORD
    description: "The random user password"
    generate: expression
    from: "[a-zA-Z0-9]{12}"
```

이전 예에서 처리가 완료되면 모든 대문자 및 소문자 영문자와 숫자로 구성된 12자 길이의 임의의 암호가 생성됩니다.

사용 가능한 구문은 완전한 정규식 구문이 아닙니다. 하지만 **\w**, **\d**, **\a**, and **\A** 수정자를 사용할 수 있습니다.

- **[\w]{10}** 은 10개의 영문자, 숫자 및 밑줄을 생성합니다. 이는 PCRE 표준을 따르며**[a-zA-Z0-9_]{10}** 과 동일합니다.
- **[\D]{10}** 은 10개의 숫자를 생성합니다.**[0-9]{10}** 과 동일합니다.
- **[\a]{10}** 은 10개의 알파벳 문자를 생성합니다.**[a-zA-Z]{10}** 과 동일합니다.
- **[\a]{10}** 은 10개의 구두점 또는 기호 문자를 생성합니다. 이는**[~!@#$%\^&*()\-_+={}\[\]\\|<,>.?/"';:`]{10}**과 동일합니다.

참고

템플릿이 YAML 또는 JSON로 작성된 경우, 수정자가 포함된 문자열 유형에 따라 두 번째 백슬래시를 사용하여 백슬래시를 이스케이프해야 할 수 있습니다. 다음 예에서와 같습니다.

수정자가 포함된 YAML 템플릿 예

```
parameters:
- name: singlequoted_example
  generate: expression
  from: '[\A]{10}'
- name: doublequoted_example
  generate: expression
  from: "[\\A]{10}"
```

수정자가 포함된 JSON 템플릿 예

```
{
   "parameters": [
     {
      "name": "json_example",
      "generate": "expression",
      "from": "[\\A]{10}"
     }
   ]
}
```

다음은 매개변수 정의 및 참조가 포함된 전체 템플릿의 예입니다.

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
  - kind: BuildConfig
    apiVersion: build.openshift.io/v1
    metadata:
      name: cakephp-mysql-example
      annotations:
        description: Defines how to build the application
    spec:
      source:
        type: Git
        git:
          uri: "${SOURCE_REPOSITORY_URL}" (1)
          ref: "${SOURCE_REPOSITORY_REF}"
        contextDir: "${CONTEXT_DIR}"
  - kind: DeploymentConfig
    apiVersion: apps.openshift.io/v1
    metadata:
      name: frontend
    spec:
      replicas: "${{REPLICA_COUNT}}" (2)
parameters:
```

```
  - name: SOURCE_REPOSITORY_URL (3)
    displayName: Source Repository URL (4)
    description: The URL of the repository with your application source code (5)
    value: https://github.com/sclorg/cakephp-ex.git (6)
    required: true (7)
  - name: GITHUB_WEBHOOK_SECRET
    description: A secret string used to configure the GitHub webhook
    generate: expression (8)
    from: "[a-zA-Z0-9]{40}" (9)
  - name: REPLICA_COUNT
    description: Number of replicas to run
    value: "2"
    required: true
message: "... The GitHub webhook secret is ${GITHUB_WEBHOOK_SECRET} ..." (10)
```

1. 이 값은 템플릿이 인스턴스화될 때 **SOURCE_REPOSITORY_URL** 매개변수 값으로 교체됩니다.
2. 이 값은 템플릿이 인스턴스화될 때 인용되지 않은 **REPLICA_COUNT** 매개변수 값으로 교체됩니다.
3. 매개변수의 이름입니다. 이 값은 템플릿 내에서 매개변수를 참조하는 데 사용됩니다.
4. 사용자에게 친숙한 매개변수 이름입니다. 이는 사용자에게 표시됩니다.
5. 매개변수에 대한 설명입니다. 예상 값에 대한 제약 조건을 비롯하여 매개변수의 목적에 대한 자세한 정보를 제공합니다. 설명은 콘솔의 텍스트 표준을 따르는 완전한 문장을 사용해야 합니다. 표시 이름과 중복되지 않게 하십시오.
6. 템플릿을 인스턴스화할 때 사용자가 값을 재정의하지 않는 경우 사용되는 매개변수의 기본값입니다. 암호 등에 기본값을 사용하지 말고 생성된 매개변수를 시크릿과 조합하여 사용하십시오.
7. 이 매개변수가 필수임을 나타냅니다. 즉, 빈 값으로 이 매개변수를 재정의할 수 없습니다. 매개변수가 기본값 또는 생성된 값을 제공하지 않으면 값을 제공해야 합니다.
8. 값이 생성되어 있는 매개변수입니다.
9. 생성기에 대한 입력입니다. 이 경우 생성기는 대문자 및 소문자를 포함하여 40자 길이의 영숫자 값을 생성합니다.
10. 매개변수가 템플릿 메시지에 포함될 수 있습니다. 생성된 값에 대해 알려줍니다.

10.7.4. 템플릿 오브젝트 목록 작성

템플릿의 주요 부분은 템플릿이 인스턴스화될 때 생성되는 오브젝트 목록입니다. 빌드 구성, 배포 구성 또는 서비스와 같은 유효한 API 오브젝트일 수 있습니다. 오브젝트는 정확히 여기에 정의된 대로 생성되며 매개변수 값은 생성 전에 대체됩니다. 이러한 오브젝트 정의는 앞에서 정의한 매개변수를 참조할 수 있습니다.

다음은 오브젝트 목록의 예입니다.

```
kind: "Template"
apiVersion: "v1"
metadata:
  name: my-template
objects:
  - kind: "Service" (1)
```

```
apiVersion: "v1"
metadata:
  name: "cakephp-mysql-example"
  annotations:
    description: "Exposes and load balances the application pods"
spec:
  ports:
    - name: "web"
      port: 8080
      targetPort: 8080
  selector:
    name: "cakephp-mysql-example"
```

1 이 템플릿에서 생성된 서비스의 정의입니다.

참고

오브젝트 정의 메타데이터에 고정된 **namespace** 필드 값이 포함된 경우 템플릿을 인스턴스화하는 동안 이 필드가 정의에서 제거됩니다. **namespace** 필드에 매개변수 참조가 포함되어 있는 경우 사용자가 해당 네임스페이스에서 오브젝트를 생성할 수 있는 권한이 있다고 가정하면 매개변수 대체가 값을 해석한 모든 네임스페이스에서 일반 매개변수 대체가 수행되고 오브젝트가 생성됩니다.

10.7.5. 템플릿을 바인딩 가능으로 표시

Template Service Broker는 해당 카탈로그에서 인식하는 템플릿 오브젝트마다 하나의 서비스를 알립니다. 기본적으로 이러한 각 서비스는 바인드 가능 상태로 알려집니다. 즉, 일반 사용자가 프로비저닝된 서비스에 대해 바인딩할 수 있습니다.

절차

템플릿 작성자는 일반 사용자가 지정된 템플릿에서 프로비저닝된 서비스에 대해 바인딩하지 못하도록 할 수 있습니다.

- **template.openshift.io/bindable: "false"** 주석을 템플릿에 추가하여 일반 사용자가 지정된 템플릿에서 프로비저닝된 서비스에 대해 바인딩하지 못하도록 합니다.

10.7.6. 템플릿 오브젝트 필드 노출

템플릿 작성자는 템플릿의 특정 오브젝트 필드가 노출되어야 함을 나타낼 수 있습니다. Template Service Broker는 **ConfigMap**, **Secret**, **Service**, **Route** 오브젝트에서 노출된 필드를 인식하고, 브로커가 지원하는 서비스를 사용자가 바인딩할 때 노출된 필드의 값을 반환합니다.

오브젝트의 필드를 하나 이상 노출하려면 **template.openshift.io/expose-** 또는 **template.openshift.io/base64-expose-**로 접두사를 붙인 주석을 템플릿의 오브젝트에 추가합니다.

접두사가 제거된 각 주석 키는 전달되어 **bind** 응답의 키가 됩니다.

각 주석 값은 Kubernetes JSONPath 표현식이며, 바인딩 시 이를 해석하여 **bind** 응답에서 값을 반환해야 하는 오브젝트 필드를 식별합니다.

참고

Bind 응답 키/값 쌍은 시스템의 다른 부분에서 환경 변수로 사용될 수 있습니다. 따라서 접두사가 제거된 모든 주석 키는 유효한 환경 변수 이름이 되도록 하는 것이 좋습니다. 즉 **A-Z**, **a-z** 또는 _로 시작하고 뒤에 **A-Z**, **a-z**, **0-9** 또는 _ 문자가 0개 이상 나와야 합니다.

참고

백슬래시로 이스케이프되지 않으면 Kubernetes JSONPath 구현에서는 ., @ 및 메타 문자와 같은 기타 문자를 표현식 내 위치에 관계없이 문자로 해석합니다. 따라서 예를 들어 **my.key**라는 **ConfigMap** 데이터를 참조하기 위해 필요한 JSONPath 표현식은 **{.data['my\.key']}**입니다. JSONPath 표현식이 YAML로 작성되는 방법에 따라 추가 백슬래시가 필요할 수도 있습니다(예: **"{.data['my\\.key']}"**).

다음은 노출되는 다양한 오브젝트 필드의 예입니다.

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: ConfigMap
  apiVersion: v1
  metadata:
    name: my-template-config
    annotations:
      template.openshift.io/expose-username: "{.data['my\\.username']}"
  data:
    my.username: foo
- kind: Secret
  apiVersion: v1
  metadata:
    name: my-template-config-secret
    annotations:
      template.openshift.io/base64-expose-password: "{.data['password']}"
  stringData:
    password: <password>
- kind: Service
  apiVersion: v1
  metadata:
    name: my-template-service
    annotations:
      template.openshift.io/expose-service_ip_port: "{.spec.clusterIP}:{.spec.ports[?
(.name==\"web\")].port}"
  spec:
    ports:
    - name: "web"
      port: 8080
- kind: Route
  apiVersion: route.openshift.io/v1
  metadata:
    name: my-template-route
    annotations:
```

```
    template.openshift.io/expose-uri: "http://{.spec.host}{.spec.path}"
  spec:
    path: mypath
```

위의 부분적인 템플릿을 기반으로 하는 **bind** 작업에 대한 응답 예는 다음과 같습니다.

```
{
  "credentials": {
    "username": "foo",
    "password": "YmFy",
    "service_ip_port": "172.30.12.34:8080",
    "uri": "http://route-test.router.default.svc.cluster.local/mypath"
  }
}
```

프로세스

- **template.openshift.io/expose-** 주석을 사용하여 필드 값을 문자열로 반환합니다. 이 주석을 사용하면 임의의 바이너리 데이터를 처리하지는 않지만 편리합니다.
- 바이너리 데이터를 반환하려면 **template.openshift.io/base64-expose-** 주석을 사용하여 데이터를 반환하기 전에 base64로 인코딩합니다.

10.7.7. 템플릿 준비 상태 대기

템플릿 작성자는 템플릿 내의 특정 오브젝트가 일정 시간 대기한 뒤에 서비스 카탈로그, Template Service Broker 또는 **TemplateInstance** API의 템플릿 인스턴스화가 완료된 것으로 간주된다고 표시할 수 있습니다.

이 기능을 사용하려면 템플릿에서 다음 주석으로 하나 이상의 오브젝트를 **Build**, **BuildConfig**, **Deployment**, **DeploymentConfig**, **Job** 또는 **StatefulSet** 유형으로 표시하십시오.

```
"template.alpha.openshift.io/wait-for-ready": "true"
```

이 주석이 표시된 모든 오브젝트가 준비되었다고 보고할 때까지 템플릿 인스턴스화는 완료되지 않습니다. 마찬가지로 주석이 있는 오브젝트 보고서 중 실패하는 보고서가 있거나 템플릿이 1시간이라는 고정된 제한 시간 내에 준비되지 않으면 템플릿 인스턴스화가 실패합니다.

인스턴스화를 위해 각 오브젝트 유형의 준비 및 실패는 다음과 같이 정의됩니다.

유형	준비	실패
Build	오브젝트가 완료 단계 보고	오브젝트가 취소, 오류 또는 실패 단계 보고
BuildConfig	관련된 최신 빌드 오브젝트가 완료 단계 보고	관련된 최신 빌드 오브젝트가 취소, 오류 또는 실패 단계 보고
배포	오브젝트는 사용 가능한 새 복제본 세트 및 배포를 보고합니다. 이 명령은 오브젝트에 정의된 준비 상태 프로브를 따릅니다.	오브젝트가 진행 상태를 False로 보고

유형	준비	실패
DeploymentConfig	오브젝트가 사용 가능한 새 복제 컨트롤러 및 배포를 보고합니다. 이 명령은 오브젝트에 정의된 준비 상태 프로브를 따릅니다.	오브젝트가 진행 상태를 False로 보고
작업	오브젝트가 완료 보고	오브젝트가 하나 이상의 실패가 발생했음을 보고
StatefulSet	오브젝트가 준비된 모든 복제본을 보고합니다. 이 명령은 오브젝트에 정의된 준비 상태 프로브를 따릅니다.	해당 없음

다음은 **wait-for-ready** 주석을 사용하는 템플릿 추출의 예입니다. 더 많은 예는 OpenShift Container Platform 퀵 스타트 템플릿에서 찾을 수 있습니다.

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: ...
    annotations:
      # wait-for-ready used on BuildConfig ensures that template instantiation
      # will fail immediately if build fails
      template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: DeploymentConfig
  apiVersion: apps.openshift.io/v1
  metadata:
    name: ...
    annotations:
      template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: Service
  apiVersion: v1
  metadata:
    name: ...
  spec:
    ...
```

추가 권장 사항

- 메모리, CPU 및 스토리지 기본 크기를 설정하여 애플리케이션에 원활한 실행을 위한 리소스가 충분히 제공되도록 합니다.

- **latest** 태그가 주요 버전 간에 사용되는 경우 이미지의 해당 태그를 참조하지 마십시오. 새 이미지를 해당 태그로 푸시하면 실행 중인 애플리케이션이 중단될 수도 있습니다.
- 좋은 템플릿은 템플릿 배포 후 수정할 필요 없이 깔끔하게 빌드되고 배포됩니다.

10.7.8. 기존 오브젝트에서 템플릿 생성

전체 템플릿을 처음부터 작성하지 않고 프로젝트에서 기존 오브젝트를 YAML 형식으로 내보낸 후 템플릿 형식으로 매개변수 및 기타 사용자 정의를 추가하여 YAML을 수정할 수 있습니다.

프로세스

- 프로젝트의 오브젝트를 YAML 형식으로 내보냅니다.

```
$ oc get -o yaml all > <yaml_filename>
```

  all을 사용하지 않고 특정 리소스 유형 또는 여러 리소스를 대체할 수도 있습니다. 더 많은 예를 보려면 **oc get -h**를 실행하십시오.

  oc get -o yaml all에 포함된 오브젝트 유형은 다음과 같습니다.

  - **BuildConfig**
  - **Build**
  - **DeploymentConfig**
  - **ImageStream**
  - **Pod**
  - **ReplicationController**
  - 경로
  - 서비스

참고

all 별칭을 사용하는 것은 다른 클러스터 및 버전마다 내용이 다를 수 있으므로 권장되지 않습니다. 대신 필요한 모든 리소스를 지정합니다.

11장. RUBY ON RAILS 사용

Ruby on Rails는 Ruby로 작성된 웹 프레임워크입니다. 이 안내서에서는 OpenShift Container Platform에서 Rails 4를 사용하는 방법을 설명합니다.

> **주의**
>
> 전체 자습서를 살펴보고 OpenShift Container Platform에서 애플리케이션을 실행하는 데 필요한 모든 단계에 대한 개요를 알아보십시오. 문제가 발생하면 전체 자습서를 읽어보고 다시 돌아가 문제를 해결하십시오. 이전 단계를 검토하여 모든 단계가 제대로 실행되었는지 확인하는 것도 유용할 수 있습니다.

11.1. 사전 요구 사항

- Ruby 및 Rails에 대한 기본 지식
- 로컬로 설치된 Ruby 2.0.0+, Rubygems, Bundler 버전
- Git에 대한 기본 지식
- OpenShift Container Platform 4의 실행 중인 인스턴스
- OpenShift Container Platform 인스턴스가 실행 중이고 사용 가능한지 확인합니다. 또한 이메일 주소와 암호로 로그인할 수 있도록 **oc** CLI 클라이언트가 설치되어 있고 명령 셸에서 명령에 액세스할 수 있는지 확인합니다.

11.2. 데이터베이스 설정

Rails 애플리케이션은 거의 항상 데이터베이스와 함께 사용됩니다. 로컬 개발의 경우 PostgreSQL 데이터베이스를 사용합니다.

절차

1. 데이터베이스를 설치합니다.

```
$ sudo yum install -y postgresql postgresql-server postgresql-devel
```

2. 데이터베이스를 초기화합니다.

```
$ sudo postgresql-setup initdb
```

이 명령은 데이터를 저장할 **/var/lib/pgsql/data** 디렉토리를 생성합니다.

3. 데이터베이스를 시작합니다.

```
$ sudo systemctl start postgresql.service
```

4. 데이터베이스가 실행 중이면 **rails** 사용자를 생성합니다.

```
$ sudo -u postgres createuser -s rails
```

생성된 사용자에게는 암호가 없습니다.

11.3. 애플리케이션 작성

Rails 애플리케이션을 처음부터 시작하는 경우 Rails gem을 먼저 설치해야 합니다. 그런 다음, 애플리케이션 작성을 진행할 수 있습니다.

프로세스

1. Rails gem을 설치합니다.

```
$ gem install rails
```

출력 예

```
Successfully installed rails-4.3.0
1 gem installed
```

2. Rails gem을 설치한 후 PostgreSQL을 데이터베이스로 사용하는 새 애플리케이션을 생성합니다.

```
$ rails new rails-app --database=postgresql
```

3. 새 애플리케이션 디렉토리로 변경합니다.

```
$ cd rails-app
```

4. 이미 애플리케이션이 있으면 **pg**(postgresql) gem이 **Gemfile**에 있는지 확인합니다. 없는 경우 gem을 추가하여 **Gemfile**을 편집합니다.

```
gem 'pg'
```

5. 종속성이 모두 포함된 새 **Gemfile.lock**을 생성합니다.

```
$ bundle install
```

6. **pg** gem과 함께 **postgresql** 데이터베이스를 사용하는 것 외에도**config/database.yml**에서 **postgresql** 어댑터를 사용하고 있는지 확인해야 합니다.
config/database.yml 파일의 **default** 섹션이 다음과 같이 표시되도록 업데이트되었는지 확인합니다.

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: 5
  host: localhost
  username: rails
  password: <password>
```

7. 애플리케이션의 개발 및 테스트 데이터베이스를 생성합니다.

```
$ rake db:create
```

PostgreSQL 서버에 **development** 및 **test** 데이터베이스가 생성됩니다.

11.3.1. 시작 페이지 생성

Rails 4는 더 이상 프로덕션에서 정적 **public/index.html** 페이지를 제공하지 않으므로 새 루트 페이지를 생성해야 합니다.

사용자 정의 시작 페이지를 생성하려면 다음 단계를 수행해야 합니다.

- 인덱스 작업을 사용하여 컨트롤러 생성
- 시작 컨트롤러 인덱스 작업에 대한 뷰 페이지 생성
- 생성된 컨트롤러 및 뷰를 통해 애플리케이션 루트 페이지를 제공할 경로 생성

Rails는 필요한 모든 단계를 수행하는 생성기를 제공합니다.

절차

1. Rails 생성기를 실행합니다.

```
$ rails generate controller welcome index
```

필요한 모든 파일이 생성됩니다.

2. **config/routes.rb** 파일의 2행을 다음과 같이 편집합니다.

```
root 'welcome#index'
```

3. rails 서버를 실행하여 페이지가 사용 가능한지 확인합니다.

```
$ rails server
```

브라우저에서 http://localhost:3000으로 가서 페이지를 확인해야 합니다. 페이지가 표시되지 않으면 서버로 출력되는 로그를 확인하여 디버그합니다.

11.3.2. OpenShift Container Platform의 애플리케이션 구성

애플리케이션이 OpenShift Container Platform에서 실행되는 PostgreSQL 데이터베이스 서비스와 통신하도록 하려면 데이터베이스 서비스 생성 시 나중에 정의할 환경 변수를 사용하도록 **config/database.yml**의 **default** 섹션을 편집해야 합니다.

절차

- 다음과 같이 사전 정의된 변수를 사용하여 **config/database.yml**의 **default** 섹션을 편집합니다.

  config/database YAML 파일 샘플

```
<% user = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ? "root" :
ENV["POSTGRESQL_USER"] %>
<% password = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ?
ENV["POSTGRESQL_ADMIN_PASSWORD"] : ENV["POSTGRESQL_PASSWORD"] %>
```

```
<% db_service = ENV.fetch("DATABASE_SERVICE_NAME","").upcase %>

default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see rails configuration guide
  # http://guides.rubyonrails.org/configuring.html#database-pooling
  pool: <%= ENV["POSTGRESQL_MAX_CONNECTIONS"] || 5 %>
  username: <%= user %>
  password: <%= password %>
  host: <%= ENV["#{db_service}_SERVICE_HOST"] %>
  port: <%= ENV["#{db_service}_SERVICE_PORT"] %>
  database: <%= ENV["POSTGRESQL_DATABASE"] %>
```

11.3.3. Git에 애플리케이션 저장

OpenShift Container Platform에서 애플리케이션을 빌드하려면 일반적으로 소스 코드를 git 리포지터리에 저장해야 하므로 **git**이 아직 없는 경우 설치해야 합니다.

전제 조건

- **git**을 설치해야 합니다.

프로세스

1. **ls -1** 명령을 실행하여 Rails 애플리케이션 디렉토리에 있는지 확인합니다. 명령 출력은 다음과 같아야 합니다.

```
$ ls -1
```

출력 예

```
app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.rdoc
test
tmp
vendor
```

2. Rails 앱 디렉토리에서 다음 명령을 실행하여 코드를 초기화하고 **git**으로 커밋합니다.

```
$ git init
```

```
$ git add .
```

```
$ git commit -m "initial commit"
```

애플리케이션이 커밋된 후에는 원격 리포지터리로 푸시해야 합니다. GitHub 계정으로 새 리포지터리를 생성합니다.

3. **git** 리포지터리를 가리키는 remote를 설정합니다.

```
$ git remote add origin git@github.com:<namespace/repository-name>.git
```

4. 애플리케이션을 원격 git 리포지터리로 푸시합니다.

```
$ git push
```

11.4. OPENSHIFT CONTAINER PLATFORM에 애플리케이션 배포

OpenShift Container Platform에 애플리케이션을 배포할 수 있습니다.

rails-app 프로젝트를 생성하면 자동으로 새 프로젝트 네임스페이스로 전환됩니다.

OpenShift Container Platform에서 애플리케이션을 배포하려면 다음 세 단계를 수행해야 합니다.

- OpenShift Container Platform의 PostgreSQL 이미지에서 데이터베이스 서비스를 생성합니다.
- 데이터베이스 서비스와 연결된 OpenShift Container Platform의 Ruby 2.0 빌더 이미지 및 Ruby on Rails 소스 코드에서 프런트 엔드 서비스를 생성합니다.
- 애플리케이션 경로를 생성합니다.

절차

- Ruby on Rails 애플리케이션을 배포하려면 애플리케이션을 위한 새 프로젝트를 생성합니다.

```
$ oc new-project rails-app --description="My Rails application" --display-name="Rails
Application"
```

11.4.1. 데이터베이스 서비스 생성

Rails 애플리케이션에는 실행 중인 데이터베이스 서비스가 필요합니다. 이 서비스의 경우 PostgreSQL 데이터베이스 이미지를 사용합니다.

데이터베이스 서비스를 생성하려면 **oc new-app** 명령을 사용합니다. 데이터베이스 컨테이너 내에서 사용할 몇 가지 필요한 환경 변수를 이 명령으로 전달해야 합니다. 이러한 환경 변수는 사용자 이름, 암호 및 데이터베이스 이름을 설정하는 데 필요합니다. 이러한 환경 변수 값은 원하는 대로 변경할 수 있습니다. 변수는 다음과 같습니다.

- POSTGRESQL_DATABASE
- POSTGRESQL_USER
- POSTGRESQL_PASSWORD

이러한 변수를 설정하면 다음과 같은 결과를 얻을 수 있습니다.

- 지정된 이름의 데이터베이스가 있습니다.

- 지정된 이름의 사용자가 있습니다.
- 사용자가 지정된 암호를 사용하여 지정된 데이터베이스에 액세스할 수 있습니다.

프로세스

1. 데이터베이스 서비스를 생성합니다.

```
$ oc new-app postgresql -e POSTGRESQL_DATABASE=db_name -e
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password
```

데이터베이스 관리자의 암호도 설정하려면 다음 행을 이전 명령에 추가합니다.

```
-e POSTGRESQL_ADMIN_PASSWORD=admin_pw
```

2. 진행 상황을 확인합니다.

```
$ oc get pods --watch
```

11.4.2. 프런트 엔드 서비스 생성

애플리케이션을 OpenShift Container Platform으로 가져오려면 애플리케이션이 상주하는 리포지터리를 지정해야 합니다.

프로세스

1. 프런트 엔드 서비스를 생성하고 데이터베이스 서비스를 생성할 때 설정된 데이터베이스 관련 환경 변수를 지정합니다.

```
$ oc new-app path/to/source/code --name=rails-app -e
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password -e
POSTGRESQL_DATABASE=db_name -e DATABASE_SERVICE_NAME=postgresql
```

OpenShift Container Platform은 이 명령을 사용하여 소스 코드를 가져와서 빌더를 설정하고, 애플리케이션 이미지를 빌드하며, 새로 생성된 이미지를 지정된 환경 변수와 함께 배포합니다. 애플리케이션 이름은 **rails-app**으로 지정됩니다.

2. **rails-app** 배포 구성의 JSON 문서를 보고 환경 변수가 추가되었는지 확인합니다.

```
$ oc get dc rails-app -o json
```

다음 섹션이 표시되어야 합니다.

출력 예

```
env": [
    {
        "name": "POSTGRESQL_USER",
        "value": "username"
    },
    {
        "name": "POSTGRESQL_PASSWORD",
        "value": "password"
```

```
    },
    {
      "name": "POSTGRESQL_DATABASE",
      "value": "db_name"
    },
    {
      "name": "DATABASE_SERVICE_NAME",
      "value": "postgresql"
    }

  ],
```

3. 빌드 프로세스를 확인합니다.

```
$ oc logs -f build/rails-app-1
```

4. 빌드가 완료되면 OpenShift Container Platform에서 실행 중인 Pod를 확인합니다.

```
$ oc get pods
```

myapp-<number>-<hash>로 시작되는 행이 표시되어야 합니다. OpenShift Container Platform에서 실행 중인 애플리케이션을 나타냅니다.

5. 애플리케이션이 작동하려면 데이터베이스 마이그레이션 스크립트를 실행하여 데이터베이스를 초기화해야 합니다. 이 작업을 수행하는 방법은 다음 두 가지입니다.

- 실행 중인 프런트 엔드 컨테이너에서 수동으로 수행합니다.
  - **rsh** 명령을 사용하여 프런트 엔드 컨테이너에 대해 실행합니다.

    ```
    $ oc rsh <frontend_pod_id>
    ```

  - 컨테이너 내부에서 마이그레이션을 실행합니다.

    ```
    $ RAILS_ENV=production bundle exec rake db:migrate
    ```

    Rails 애플리케이션을 **development** 또는 **test** 환경에서 실행 중인 경우 **RAILS_ENV** 환경 변수를 지정할 필요가 없습니다.

- 템플릿에 사전 배포 라이프사이클 후크를 추가하여 수행합니다.

11.4.3. 애플리케이션 경로 생성

서비스를 공개하여 애플리케이션 경로를 생성할 수 있습니다.

프로세스

- **www.example.com**과 같이 외부에서 접근할 수 있는 호스트 이름을 지정하여 서비스를 공개하려면 OpenShift Container Platform 경로를 사용합니다. 이 경우 다음을 입력하여 프런트 엔드 서비스를 공개해야 합니다.

```
$ oc expose service rails-app --hostname=www.example.com
```

주의

지정한 호스트 이름이 라우터의 IP 주소로 해석되는지 확인하십시오.

12장. 이미지 사용

12.1. 이미지 사용 개요

다음 주제에서는 OpenShift Container Platform 사용자가 사용할 수 있는 다양한 S2I(Source-to-Image), 데이터베이스 및 기타 컨테이너 이미지를 알아봅니다.

Red Hat 공식 컨테이너 이미지는 registry.redhat.io의 Red Hat Registry에 제공되어 있습니다. OpenShift Container Platform의 지원되는 S2I, 데이터베이스 및 Jenkins 이미지는 Red Hat Quay Registry의 **openshift4** 리포지터리에 제공되어 있습니다. 예를 들어**quay.io/openshift-release-dev/ocp-v4.0-<address>**는 OpenShift Application Platform 이미지의 이름입니다.

xPaaS 미들웨어 이미지는 Red Hat Registry의 해당 제품 리포지터리에 제공되어 있으나 **-openshift** 접미사가 붙어 있습니다. 예를 들어 **registry.redhat.io/jboss-eap-6/eap64-openshift**는 JBoss EAP 이미지의 이름입니다.

이 섹션에서 설명하는 모든 Red Hat 지원 이미지는 Red Hat Ecosystem Catalog의 컨테이너 이미지 섹션에 설명되어 있습니다. 각 이미지의 모든 버전에 대한 콘텐츠와 사용법 세부 정보를 찾아볼 수 있습니다. 관심 있는 이미지를 찾아보거나 검색하십시오.

중요

최신 버전의 컨테이너 이미지는 이전 버전의 OpenShift Container Platform과 호환되지 않습니다. 사용 중인 OpenShift Container Platform 버전에 따라 올바른 버전의 컨테이너 이미지를 확인하고 사용하십시오.

12.2. S2I(SOURCE-TO-IMAGE)

Node.js, Perl 또는 Python과 같은 특정 런타임 환경에 종속된 애플리케이션의 기초로 Red Hat Software Collections 이미지를 사용할 수도 있습니다.Red Hat Java Source-to-Image for OpenShift문서를 Java를 사용하는 런타임 환경의 참조로 사용할 수 있습니다. 이러한 런타임 기본 이미지 중 일부 특수 버전은 S2I(Source-to-Image) 이미지라고 합니다. S2I 이미지를 사용하면 해당 코드를 실행할 준비가 된 기본 이미지 환경에 코드를 삽입할 수 있습니다.

S2I 이미지는 다음과 같습니다.

- .NET
- Java
- Go
- Node.js
- Perl
- PHP
- Python
- Ruby

다음 절차에 따라 OpenShift Container Platform 웹 콘솔에서 S2I 이미지를 직접 사용할 수 있습니다.

1. 로그인 인증 정보를 사용하여 OpenShift Container Platform 웹 콘솔에 로그인합니다. OpenShift Container Platform 웹 콘솔의 기본 보기는 Administrator 모드입니다.
2. 모드 전환 기능을 사용하여 Developer 모드로 전환하십시오.
3. +추가 보기에서 프로젝트 드롭다운 목록을 사용하여 기존 프로젝트를 선택하거나 새 프로젝트를 생성합니다.
4. 개발자 카탈로그 타일에서 모든 서비스를 클릭합니다.
5. 유형에서 빌더 이미지를 클릭하여 사용 가능한 S2I 이미지를 확인합니다.

Cluster Samples Operator 구성에도 S2I 이미지를 사용할 수 있습니다.

12.2.1. S2I(Source-to-Image) 빌드 프로세스 개요

S2I(Source-to-Image)는 소스 코드를 실행할 컨테이너에 소스 코드를 삽입하여 실행할 수 있는 이미지를 생성합니다. 다음과 같은 단계를 수행합니다.

1. **FROM <builder image>** 명령 실행
2. 소스 코드를 빌더 이미지의 정의된 위치에 복사
3. 빌더 이미지에서 assemble 스크립트 실행
4. 빌더 이미지에 run 스크립트를 기본 명령으로 설정

그런 다음 Buildah에서 컨테이너 이미지를 생성합니다.

12.2.2. 추가 리소스

- Cluster Samples Operator 구성
- 빌드 전략 사용
- Source-to-Image 프로세스 문제 해결
- S2I(Source-to-Image)를 사용하여 소스 코드에서 이미지 생성
- S2I(Source-to-Image) 이미지 테스트 정보
- S2I(Source-to-Image)를 사용하여 소스 코드에서 이미지 생성

12.3. S2I(SOURCE-TO-IMAGE) 이미지 사용자 정의

S2I(Source-to-Image) 빌더 이미지에는 assemble 및 run 스크립트가 포함되어 있지만 해당 스크립트의 기본 동작은 모든 사용자에게 적합하지 않습니다. 기본 스크립트가 포함된 S2I 빌더의 동작을 사용자 지정할 수 있습니다.

12.3.1. 이미지에 포함된 스크립트 호출

빌더 이미지는 가장 일반적인 사용 사례를 포함하는 자체 버전의 S2I(Source-to-Image) 스크립트를 제공합니다. 이러한 스크립트가 요구 사항을 충족하지 않으면 S2I는 **.s2i/bin** 디렉터리에 사용자 지정 설정을 추가하여 덮어쓰는 방법을 제공합니다. 하지만 이렇게 하면 표준 스크립트를 완전히 교체할 수 있습니다. 스크

립트 교체가 허용되는 경우도 있지만 다른 시나리오에서는 이미지에 제공된 스크립트의 논리를 유지하면서 스크립트 전 또는 후에 몇 명령을 실행할 수 있습니다. 표준 스크립트를 재사용하려면 사용자 정의 논리를 실행하고 이미지의 기본 스크립트로 추가 작업을 위임하는 래퍼 스크립트를 생성할 수 있습니다.

절차

1. **io.openshift.s2i.scripts-url** 레이블의 값을 보고 빌더 이미지 내부의 스크립트 위치를 확인합니다.

```
$ podman inspect --format='{{ index .Config.Labels "io.openshift.s2i.scripts-url" }}'
wildfly/wildfly-centos7
```

출력 예

```
image:///usr/libexec/s2i
```

wildfly/wildfly-centos7 빌더 이미지를 검사하고 스크립트가/**usr/libexec/s2i** 디렉터리에 있음을 확인합니다.

2. 다른 명령으로 래핑된 표준 스크립트 중 하나를 호출하는 스크립트를 생성합니다.

.s2i/bin/assemble 스크립트

```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
   echo "After successful assembling"
else
   echo "After failed assembling"
fi

exit $rc
```

이 예에서는 메시지를 출력하고, 이미지에서 표준 assemble 스크립트를 실행하는 사용자 정의 assemble 스크립트를 보여주고 assemble 스크립트의 종료 코드에 따라 다른 메시지를 출력합니다.

중요

run 스크립트를 래핑할 때 **exec**를 사용하여 신호가 올바르게 처리되는지 확인해야 합니다. **exec**를 사용하면 기본 이미지 실행 스크립트를 호출한 후 추가 명령을 실행할 수 없습니다.

.s2i/bin/run 스크립트

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```