OpenShift Container Platform 3.11

아키텍처

OpenShift Container Platform 3.11 아키텍처 정보

Last Updated: 2023-02-20

OpenShift Container Platform 3.11 아키텍처

OpenShift Container Platform 3.11 아키텍처 정보

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

법적 공지

초록

인프라 및 핵심 구성 요소를 포함하여 OpenShift Container Platform 3.11의 아키텍처를 학습합니다. 이러한 주제에서는 인증, 네트워킹 및 소스 코드 관리도 다룹니다.

차례

1장. 개요

OpenShift v3은 개발자가 애플리케이션을 쉽게 구성하는 데 중점을 두고 가능한 한 정확하게 기본 Docker 형식의 컨테이너 이미지와 Kubernetes 개념을 노출하도록 설계된 계층화된 시스템입니다. 예를 들어 Ruby, 내보내기 코드를 설치하고 MySQL을 추가합니다.

OpenShift v2와 달리, 모델의 모든 측면에 생성한 후 구성의 유연성이 향상됩니다. 별도의 개체로서의 애플리케이션 개념은 "서비스"를 보다 유연하게 구성하기 위해 제거되며, 두 웹 컨테이너에서 데이터베이스를 재사용하거나 네트워크 에지에 직접 데이터베이스를 노출할 수 있습니다.

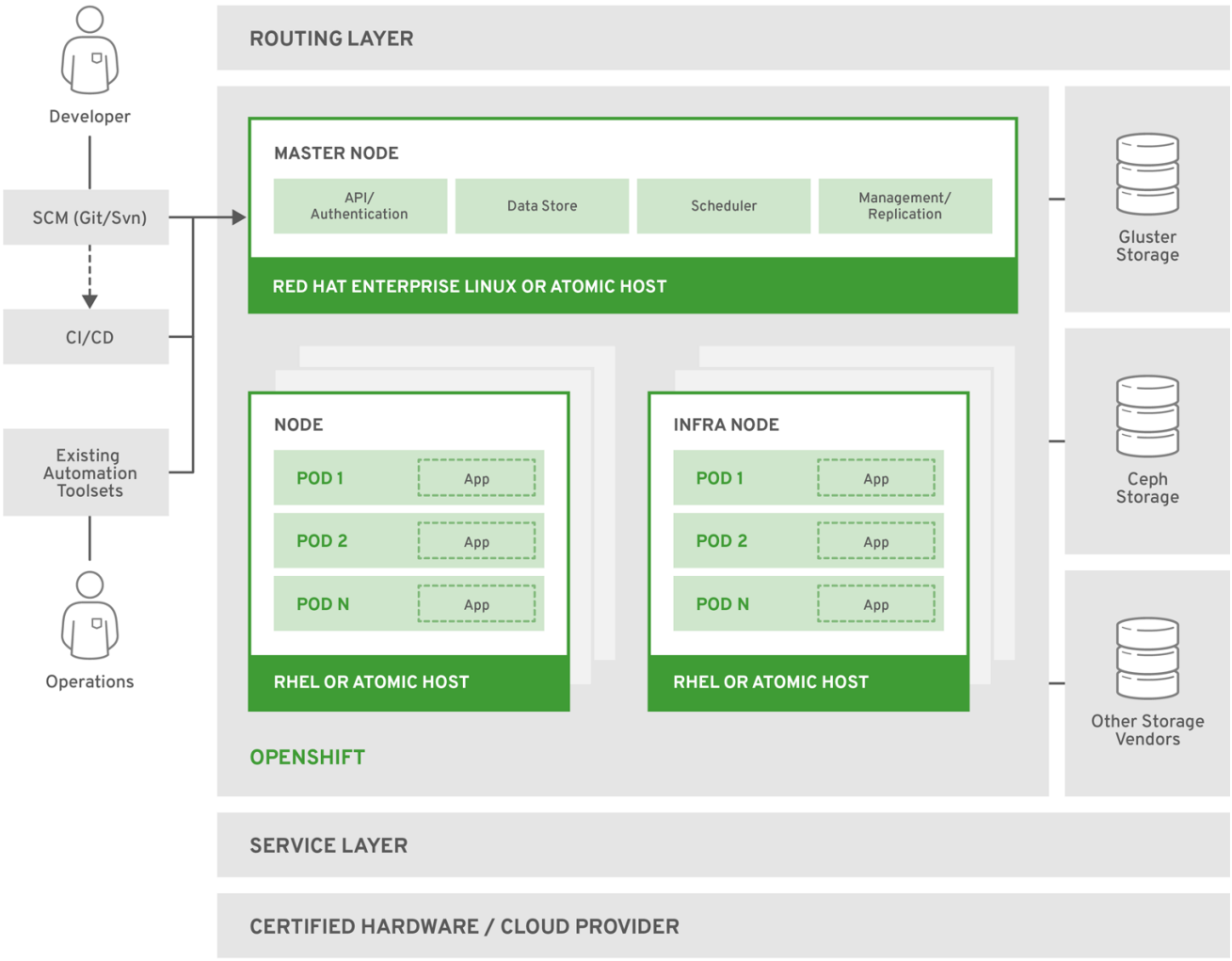
1.1. 계층은 무엇입니까?

Docker 서비스는 Linux 기반 경량 컨테이너 이미지를 패키징하고 생성하는 데 대한 추상화를 제공합니다. Kubernetes는 클러스터 관리를 제공하고 여러 호스트에서 컨테이너를 오케스트레이션합니다.

OpenShift Container Platform은 다음을 추가합니다.

- 개발자용 소스 코드 관리, 빌드 및 배포
- 시스템을 통과할 때 대규모로 이미지 관리 및 승격
- 대규모 애플리케이션 관리
- 대규모 개발자 조직 구성을 위한 팀 및 사용자 추적
- 클러스터를 지원하는 네트워킹 인프라

그림 1.1. OpenShift Container Platform 아키텍처 개요



아키텍처 개요의 노드 유형에 대한 자세한 내용은 Kubernetes 인프라를 참조하십시오.

1.2. OPENSHIFT CONTAINER PLATFORM 아키텍처란 무엇입니까?

OpenShift Container Platform에는 분리된 작은 단위의 마이크로 서비스 기반 아키텍처가 함께 작동합니다. 안정적인 클러스터형 키-값 저장소인 etcd 에 저장된 오브젝트에 대한 데이터를 사용하여 Kubernetes 클러스터에서 실행됩니다. 이러한 서비스는 다음과 같은 기능으로 분류됩니다.

- REST API - 각 핵심 오브젝트를 노출합니다.
- 해당 API를 읽고 변경 사항을 다른 오브젝트에 적용하고 상태를 보고하거나 오브젝트에 다시 쓰는 컨트롤러입니다.

사용자는 REST API를 호출하여 시스템 상태를 변경합니다. 컨트롤러는 REST API를 사용하여 사용자의 원하는 상태를 읽은 다음 시스템의 다른 부분을 동기화합니다. 예를 들어 사용자가 빌드를 요청하면 "build" 오브젝트를 생성합니다. 빌드 컨트롤러는 새 빌드가 생성되었음을 확인하고 클러스터에서 프로세스를 실행하여 해당 빌드를 수행합니다. 빌드가 완료되면 컨트롤러에서 REST API를 통해 빌드 오브젝트를 업데이트하고 사용자가 빌드가 완료되었음을 확인합니다.

컨트롤러 패턴은 OpenShift Container Platform의 대부분의 기능이 확장 가능함을 의미합니다. 이미지를 관리하는 방법 또는 배포의 발생 방식과 관계없이 빌드가 실행 및 시작되는 방식은 사용자 지정할 수 있습니다. 컨트롤러는 시스템의 "비즈니스 로직"을 수행하여 사용자 작업을 수행하고 현실로 변환합니다. 이러한 컨트롤러를 사용자 정의하거나 자체 논리로 교체하면 다양한 동작을 구현할 수 있습니다. 시스템 관리

관점에서는 API를 사용하여 반복 스케줄에 일반적인 관리 작업을 스크립팅할 수 있습니다. 이러한 스크립트는 변경 사항을 조사하고 조치를 취하는 컨트롤러이기도 합니다. OpenShift Container Platform을 사용하면 이러한 방식으로 최상위 동작으로 클러스터를 사용자 지정할 수 있습니다.

이를 가능하게 하기 위해 컨트롤러는 안정적인 시스템 변경 스트림을 활용하여 시스템 보기를 사용자가 수행하는 작업과 동기화합니다. 이 이벤트 스트림은 etcd의 변경 사항을 REST API로 푸시한 다음 변경 사항이 발생하는 즉시 컨트롤러에 푸시되므로 변경 사항이 시스템을 통해 빠르고 효율적으로 제거할 수 있습니다. 그러나 언제든지 오류가 발생할 수 있으므로 컨트롤러는 시작 시 시스템의 최신 상태를 가져오고 모든 항목이 올바른 상태에 있는지 확인할 수 있어야 합니다. 이러한 재동기화는 문제가 발생해도 Operator가 영향을 받는 구성 요소를 다시 시작할 수 있다는 것을 의미하기 때문에 시스템이 계속하기 전에 모든 것을 재동기화할 수 있다는 의미이기 때문에 중요합니다. 컨트롤러에서 항상 시스템을 동기화할 수 있으므로 시스템은 결국 사용자의 의도와 통합되어야 합니다.

1.3. OPENSHIFT CONTAINER PLATFORM은 어떻게 보호됩니까?

OpenShift Container Platform 및 Kubernetes API는 자격 증명을 제공하는 사용자를 인증한 다음 역할에 따라 권한을 부여합니다. 개발자와 관리자는 모두 주로 OAuth 토큰과 X.509 클라이언트 인증서 등 여러 가지 수단을 통해 인증할 수 있습니다. OAuth 토큰은 SHA-256을 사용한 RSA 서명 알고리즘인 PKCS#1 v1.5인 JSON Web Algorithm *RS256* 으로 서명됩니다.

개발자(시스템의 클라이언트)는 일반적으로 브라우저를 통해 **oc** 또는 웹 콘솔과 같은 클라이언트 프로그램에서 REST API 호출을 수행하고 대부분의 통신에 OAuth 전달자 토큰을 사용합니다. 인프라 구성 요소(노드와 같은)는 ID가 포함된 시스템에서 생성한 클라이언트 인증서를 사용합니다. 컨테이너에서 실행되는 인프라 구성 요소는 서비스 계정 과 연결된 토큰을 사용하여 API에 연결합니다.

권한 부여는 "Pod 생성" 또는 "list services"와 같은 작업을 정의하고 정책 문서의 역할에 그룹화하는 OpenShift Container Platform 정책 엔진에서 처리됩니다. 역할은 사용자 또는 그룹 식별자로 사용자 또는 그룹에 바인딩됩니다. 사용자 또는 서비스 계정이 작업을 시도하면 정책 엔진은 계속 허용하기 전에 사용자에게 할당된 하나 이상의 역할(예: 현재 프로젝트의 클러스터 관리자 또는 관리자)을 확인합니다.

클러스터에서 실행되는 모든 컨테이너는 서비스 계정과 연결되므로 시크릿 을 해당 서비스 계정에 연결하고 컨테이너에 자동으로 제공할 수도 있습니다. 이를 통해 인프라는 이미지, 빌드 및 배포 구성 요소를 가져오고 푸시하기 위한 시크릿을 관리할 수 있으며 애플리케이션 코드에서 이러한 시크릿을 쉽게 활용할 수 있습니다.

1.3.1. TLS 지원

REST API와의 모든 통신 채널은 물론 etcd 및 API 서버 등의 마스터 구성 요소 간 통신 채널은 TLS로 보호됩니다. TLS는 X.509 서버 인증서 및 공개 키 인프라를 사용하여 서버의 강력한 암호화, 데이터 무결성 및 인증을 제공합니다. 기본적으로 OpenShift Container Platform의 각 배포에 대해 새로운 내부 PKI가 생성됩니다. 내부 PKI는 2048비트 RSA 키와 SHA-256 서명을 사용합니다. 공용 호스트에 대한 사용자 지정 인증서 도 지원됩니다.

OpenShift Container Platform은 Golang의 **crypto/tls** 표준 라이브러리 구현을 사용하며 외부 암호화 및 TLS 라이브러리를 사용하지 않습니다. 또한 클라이언트는 GSSAPI 인증 및 OpenPGP 서명을 위한 외부 라이브러리에 의존합니다. GSSAPI는 일반적으로 MIT Kerberos 또는 Heimdal Kerberos에서 제공하며, 둘 다 OpenSSL의 libcrypto를 사용합니다. OpenPGP 서명 확인은 libgpgme 및 GnuPG에서 처리합니다.

안전하지 않은 버전의 SSL 2.0 및 SSL 3.0은 지원되지 않으며 사용할 수 없습니다. OpenShift Container Platform 서버 및 **oc** 클라이언트는 기본적으로 TLS 1.2만 제공합니다. TLS 1.0 및 TLS 1.1은 서버 구성에서 활성화할 수 있습니다. 서버와 클라이언트 모두 인증된 암호화 알고리즘과 완벽한 전달 보안을 갖춘 최신 암호화 제품군을 선호합니다. RC4, 3DES, MD5와 같이 더 이상 사용되지 않고 안전하지 않은 알고리즘이 있는 암호화 제품군이 비활성화됩니다. 일부 내부 클라이언트(예: LDAP 인증)는 TLS 1.0에서 1.2로 설정이 덜 제한되고 더 많은 암호 제품군이 활성화되어 있습니다.

표 1.1. 지원되는 TLS 버전

TLS 버전	OpenShift Container Platform Server	oc 클라이언트	기타 클라이언트
SSL 2.0	지원되지 않음	지원되지 않음	지원되지 않음
SSL 3.0	지원되지 않음	지원되지 않음	지원되지 않음
TLS 1.0	아니요 [1]	아니요 [1]	아마도 [2]
TLS 1.1	아니요 [1]	아니요 [1]	아마도 [2]
TLS 1.2	**있음**	**있음**	**있음**
TLS 1.3	N/A [3]	N/A [3]	N/A [3]

1. 기본적으로 비활성화되지만 서버 구성에서 활성화할 수 있습니다.
2. LDAP 클라이언트 등의 일부 내부 클라이언트.
3. TLS 1.3은 아직 개발 중입니다.

OpenShift Container Platform 서버 및 **oc** 클라이언트에서 활성화된 암호화 제품군의 다음 목록은 기본으로 정렬됩니다.

- **TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305**
- **TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305**
- **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**
- **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256**
- **TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**
- **TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384**
- **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256**
- **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256**
- **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA**
- **TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA**
- **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA**
- **TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA**
- **TLS_RSA_WITH_AES_128_GCM_SHA256**
- **TLS_RSA_WITH_AES_256_GCM_SHA384**
- **TLS_RSA_WITH_AES_128_CBC_SHA**

- **TLS_RSA_WITH_AES_256_CBC_SHA**

2장. 인프라 구성 요소

2.1. KUBERNETES 인프라

2.1.1. 개요

OpenShift Container Platform 내에서 Kubernetes는 컨테이너 또는 호스트 집합에서 컨테이너화된 애플리케이션을 관리하고 배포, 유지 관리 및 애플리케이션 확장에 대한 메커니즘을 제공합니다. 컨테이너화된 애플리케이션을 인스턴스화 및 실행합니다. Kubernetes 클러스터는 하나 이상의 마스터와 노드 집합으로 구성됩니다.

클러스터에 단일 장애 지점이 없는지 확인하기 위해 선택적으로 HA(고가용성)를 위해 마스터를 구성할 수 있습니다.

참고

OpenShift Container Platform은 Kubernetes 1.11 및 Docker 1.13.1을 사용합니다.

2.1.2. 마스터

마스터는 API 서버, 컨트롤러 관리자 서버 및 etcd를 포함하여 컨트롤 플레인 구성 요소가 포함된 호스트 또는 호스트입니다. 마스터는 Kubernetes 클러스터에서 노드를 관리하고 해당 노드에서 실행되도록 포드를 예약합니다.

표 2.1. 마스터 구성 요소

구성 요소	설명
API 서버	Kubernetes API 서버는 포드, 서비스 및 복제 컨트롤러의 데이터를 검증하고 구성합니다. 또한 노드에 포드를 할당하고 서비스 구성과 포드 정보를 동기화합니다.
etcd	etcd는 영구 마스터 상태를 저장하고 다른 구성 요소는 etcd에서 변경 사항을 원하는 상태로 표시합니다. 고가용성을 위해 etcd를 선택적으로 구성할 수 있으며 일반적으로 2n+1 피어 서비스로 배포할 수 있습니다.
컨트롤러 관리자 서버	컨트롤러 관리자 서버는 etcd에서 복제 컨트롤러 오브젝트의 변경 사항을 감시한 다음 API를 사용하여 원하는 상태를 적용합니다. 이러한 여러 프로세스는 한 번에 하나의 활성 리더가 있는 클러스터를 생성합니다.
HAProxy	API 마스터 엔드포인트 간에 부하를 분산하도록 **네이티브** 메서드로 고가용성 마스터를 구성할 때 사용되는 선택 사항입니다. 클러스터 설치 프로세스에서 는 **기본** 방법을 사용하여 HAProxy를 구성할 수 있습니다. 또는 **네이티브** 메서드를 사용할 수 있지만 선택한 자체 로드 밸런서를 사전 구성할 수 있습니다.

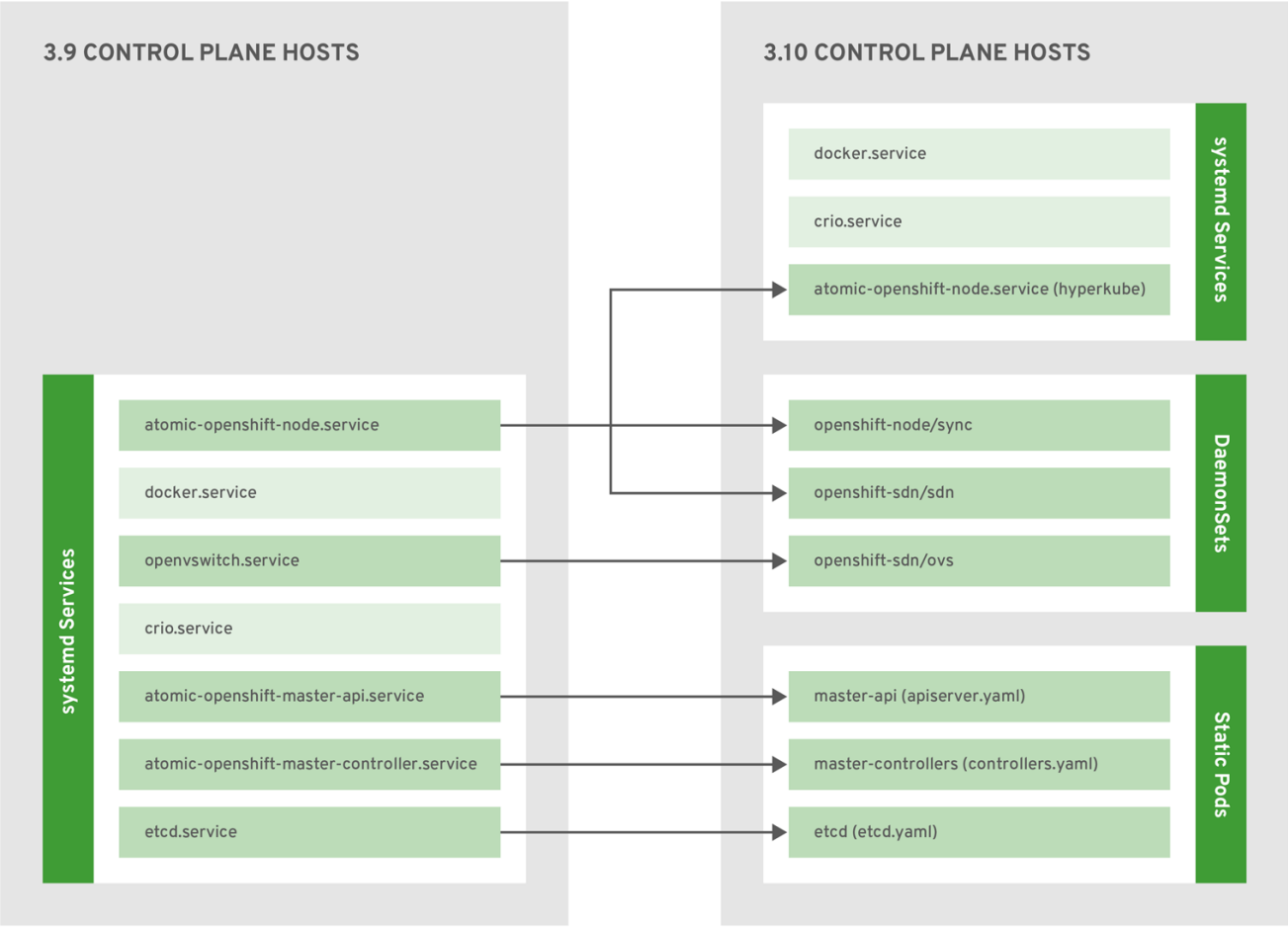
2.1.2.1. 컨트롤 플레인 정적 포드

핵심 컨트롤 플레인 구성 요소, API 서버 및 컨트롤러 관리자 구성 요소는 kubelet에서 작동하는 *정적 Pod*로 실행됩니다.

etcd가 동일한 호스트에 공동 배치된 마스터의 경우 etcd도 정적 포드로 이동합니다. RPM 기반 etcd는 여전히 마스터가 아닌 etcd 호스트에서 지원됩니다.

또한 노드 구성 요소 **openshift-sdn** 및 **openvswitch** 는 **systemd** 서비스 대신 DaemonSet을 사용하여 실행됩니다.

그림 2.1. 컨트롤 플레인 호스트 아키텍처 변경



OPENSHIFT_473421_0718

컨트롤 플레인 구성 요소가 정적 포드로 실행되는 경우에도 마스터 호스트는 마스터 및 노드 구성 항목에 설명된 대로 */etc/origin/master/master-config.yaml* 파일에서 구성을 계속 제공합니다.

시작 순서 개요

Hyperkube 는 모든 Kubernetes(kube**-apiserver,controller- manager,스케줄러,프록시**, **kubelet**)를 포함하는 바이너리입니다. 시작 시 **kubelet** 은 **kubepods.slice** 를 생성합니다. 다음으로 **kubelet** 은 **kubepods.slice 내에 QoS 수준 슬라이스burstable.slice 및 best-effort.slice** 를 생성합니다. Pod가 시작되면 **kubelet** 은 **pod<UUID-of-pod>.slice** 형식을 사용하여 Pod 수준 슬라이스를 생성하고 CRI(컨테이너 런타임 인터페이스)의 다른 쪽의 런타임에 해당 경로를 전달합니다. 그런 다음 Docker 또는 CRI-O가 Pod 수준 슬라이스 내에 컨테이너 수준 슬라이스를 생성합니다.

미러 Pod

마스터 노드의 kubelet은 **kube-system** 프로젝트의 클러스터에 표시되도록 각 컨트롤 플레인 정적 Pod에 대해 *API 서버에 미러* Pod를 자동으로 생성합니다. 이러한 정적 포드의 매니페스트는 기본적으로 마스터 호스트의 */etc/origin/node/pods* 디렉터리에 있는 **openshift-ansible** 설치 프로그램에서 설치합니다.

이러한 Pod에는 다음과 같은 **hostPath** 볼륨이 정의되어 있습니다.

/etc/origin/master	모든 인증서, 구성 파일 및 *admin.kubeconfig* 파일을 포함합니다.

/var/lib/origin	바이너리의 볼륨 및 잠재적인 코어 덤프를 포함합니다.
/etc/origin/cloudprovider	클라우드 공급자별 구성(AWS, Azure 등)을 포함합니다.
/usr/libexec/kubernetes/kubelet-plugins	추가 타사 볼륨 플러그인을 포함합니다.
/etc/origin/kubelet-plugins	시스템 컨테이너용 추가 타사 볼륨 플러그인을 포함합니다.

정적 Pod에서 수행할 수 있는 작업 세트는 제한됩니다. 예를 들면 다음과 같습니다.

```
$ oc logs master-api-<hostname> -n kube-system
```

API 서버에서 표준 출력을 반환합니다. 그러나 다음을 수행합니다.

```
$ oc delete pod master-api-<hostname> -n kube-system
```

포드를 실제로 삭제하지 않습니다.

또 다른 예로 클러스터 관리자는 문제가 발생하는 경우 더 자세한 데이터를 제공하기 위해 API **서버의 로그** 수준을 늘리는 등 일반적인 작업을 수행해야 할 수 있습니다. 이 값이 컨테이너 내부에서 실행되는 프로세스로 전달되므로 */etc/origin/master/master.env* 파일을 편집해야 합니다. 여기서 **OPTIONS** 변수의 **--loglevel** 매개 변수를 수정할 수 있습니다. 변경 사항을 적용하려면 컨테이너 내부에서 실행 중인 프로세스를 다시 시작해야 합니다.

마스터 서비스 다시 시작
컨트롤 플레인 정적 포드에서 실행 중인 컨트롤 플레인 서비스를 다시 시작하려면 마스터 호스트에서 **master-restart** 명령을 사용합니다.

마스터 API를 다시 시작하려면 다음을 수행합니다.

```
# master-restart api
```

컨트롤러를 다시 시작하려면 다음을 수행합니다.

```
# master-restart controllers
```

etcd를 다시 시작하려면 다음을 수행합니다.

```
# master-restart etcd
```

마스터 서비스 로그 보기
컨트롤 플레인 정적 Pod에서 실행되는 컨트롤 플레인 서비스의 로그를 보려면 해당 구성 요소에 대해 **master-logs** 명령을 사용합니다.

```
# master-logs api api
```

```
# master-logs controllers controllers
```

```
# master-logs etcd etcd
```

2.1.2.2. 고가용성 마스터

클러스터에 단일 장애 지점이 없는지 확인하기 위해 선택적으로 HA(고가용성)를 위해 마스터를 구성할 수 있습니다.

마스터 가용성에 대한 우려를 완화하려면 다음 두 가지 활동을 사용하는 것이 좋습니다.

1. 마스터를 다시 만들기 위해 런북 항목을 만들어야 합니다. 런북 항목은 고가용성 서비스에 필요한 백플레이스입니다. 추가 솔루션은 런북을 참조해야 하는 빈도만 제어합니다. 예를 들어 마스터 호스트의 콜드 대기 모드는 새 애플리케이션을 생성하거나 실패한 애플리케이션 구성 요소를 복구하기 위해 몇 분 이상의 다운타임이 필요한 SLA를 적절히 이행할 수 있습니다.

2. 고가용성 솔루션을 사용하여 마스터를 구성하고 클러스터에 단일 장애 지점이 없는지 확인합니다. 클러스터 설치 문서에서는 **기본** HA 방법을 사용하고 HAProxy를 구성하는 특정 예를 제공합니다. HAProxy 대신 **기본** 방법을 사용하여 개념을 사용하여 기존 HA 솔루션에 적용할 수도 있습니다.

참고

프로덕션 OpenShift Container Platform 클러스터에서는 API 서버 로드 밸런서의 고가용성을 유지 관리해야 합니다. API 서버 로드 밸런서를 사용할 수 없는 경우 노드에서 상태를 보고할 수 없으며, 모든 포드가 dead으로 표시되며 포드 엔드포인트가 서비스에서 제거됩니다.

OpenShift Container Platform용 HA를 구성하는 것 외에도 API 서버 로드 밸런서에 대해 HA를 별도로 구성해야 합니다. HA를 구성하려면 F5 Big-IP™ 또는 Citrix Netscaler™ 어플라이언스와 같은 LB(엔터프라이즈 로드 밸런서)를 통합하는 것이 좋습니다. 이러한 솔루션을 사용할 수 없는 경우 여러 HAProxy 로드 밸런서를 실행하고 Keepalived를 사용하여 HA에 유동 가상 IP 주소를 제공할 수 있습니다. 그러나 이 솔루션은 프로덕션 인스턴스에 권장되지 않습니다.

HAProxy와 함께 **기본** HA 방법을 사용하는 경우 마스터 구성 요소는 다음과 같은 가용성을 갖습니다.

표 2.2. HAProxy를 사용한 가용성 매트릭스

Role	스타일	참고
etcd	active-active	부하 분산을 통해 배포를 완전히 중복합니다. 별도의 호스트에 설치하거나 마스터 호스트에 배치할 수 있습니다.
API 서버	active-active	HAProxy에서 관리합니다.
컨트롤러 관리자 서버	active-passive	한 번에 하나의 인스턴스가 클러스터 리더로 선택됩니다.
HAProxy	active-passive	API 마스터 엔드포인트 간에 부하 분산.

Role	스타일	참고

클러스터형 etcd에는 쿼럼에 대한 홀수의 호스트가 필요하지만 마스터 서비스에는 홀수의 호스트가 있다는 쿼럼이나 요구 사항이 없습니다. 그러나 HA에 대해 두 개 이상의 마스터 서비스가 필요하므로 마스터 서비스 및 etcd를 함께 배치할 때 일관된 홀수의 호스트를 유지하는 것이 일반적입니다.

2.1.3. 노드

노드는 컨테이너를 위한 런타임 환경을 제공합니다. Kubernetes 클러스터의 각 노드에는 마스터에서 관리하는 데 필요한 서비스가 있습니다. 노드에는 컨테이너 런타임, kubelet 및 서비스 프록시를 포함하여 Pod를 실행하는 데 필요한 서비스도 있습니다.

OpenShift Container Platform은 클라우드 공급자, 물리적 시스템 또는 가상 시스템에서 노드를 생성합니다. Kubernetes는 해당 노드를 나타내는 노드 오브젝트 와 상호 작용합니다. 마스터는 노드 오브젝트의 정보를 사용하여 상태 점검에서 노드를 검증합니다. 노드는 상태 점검을 통과할 때까지 무시되며 마스터는 노드가 유효할 때까지 노드를 계속 확인합니다. Kubernetes 설명서 에는 노드 상태 및 관리에 대한 자세한 정보가 있습니다.

관리자는 CLI를 사용하여 OpenShift Container Platform 인스턴스에서 노드를 관리할 수 있습니다. 노드 서버를 시작할 때 전체 구성 및 보안 옵션을 정의하려면 전용 노드 구성 파일을 사용합니다.

중요

권장되는 최대 노드 수는 클러스터 제한 섹션을 참조하십시오.

2.1.3.1. kubelet

각 노드에는 Pod를 설명하는 YAML 파일인 컨테이너 매니페스트에서 지정한 대로 노드를 업데이트하는 kubelet이 있습니다. kubelet은 매니페스트 세트를 사용하여 해당 컨테이너가 시작되고 계속 실행되도록 합니다.

다음을 통해 kubelet에 컨테이너 매니페스트를 제공할 수 있습니다.

- 20초마다 확인되는 명령줄의 파일 경로입니다.
- 20초마다 확인되는 명령줄에서 HTTP 엔드포인트가 전달됩니다.
- kubelet은 */registry/hosts/$(hostname -f)* 와 같은 etcd 서버를 모니터링하고 변경 사항을 적용합니다.
- HTTP를 수신 대기하고 간단한 API에 응답하여 새 매니페스트를 제출하는 kubelet입니다.

2.1.3.2. 서비스 프록시

각 노드는 해당 노드의 API에 정의된 서비스를 반영하는 간단한 네트워크 프록시도 실행합니다. 이를 통해 노드는 백엔드 집합에서 간단한 TCP 및 UDP 스트림 전달을 수행할 수 있습니다.

2.1.3.3. 노드 오브젝트 정의

다음은 Kubernetes의 노드 오브젝트 정의의 예입니다.

```
apiVersion: v1 (1)
kind: Node (2)
metadata:
  creationTimestamp: null
  labels: (3)
    kubernetes.io/hostname: node1.example.com
  name: node1.example.com (4)
spec:
  externalID: node1.example.com (5)
status:
  nodeInfo:
    bootID: ""
    containerRuntimeVersion: ""
    kernelVersion: ""
    kubeProxyVersion: ""
    kubeletVersion: ""
    machineID: ""
    osImage: ""
    systemUUID: ""
```

(1) **apiVersion** 은 사용할 API 버전을 정의합니다.

(2) **Node** 로 설정된 **kind** 는 이를 노드 오브젝트의 정의로 식별합니다.

(3) **metadata.labels** 는 노드에 추가된 모든 레이블을 나열합니다.

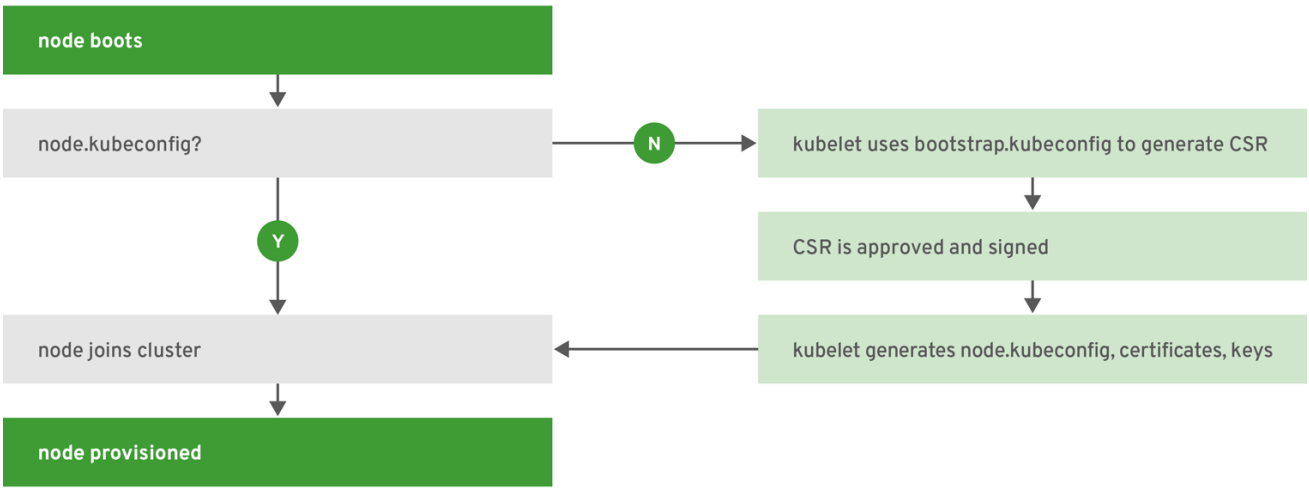
(4) **metadata.name** 은 노드 오브젝트의 이름을 정의하는 필수 값입니다. 이 값은 **oc get nodes** 명령을 실행할 때 **NAME** 열에 표시됩니다.

(5) **spec.externalID** 는 노드에 연결할 수 있는 정규화된 도메인 이름을 정의합니다. 비어 있는 경우 기본값은 **metadata.name** 값입니다.

2.1.3.4. 노드 부트스트랩

노드의 구성이 마스터에서 부트스트랩되므로 노드가 마스터에서 사전 정의된 구성과 클라이언트 및 서버 인증서를 가져옵니다. 따라서 노드 간 차이점을 줄이고 더 많은 구성을 중앙 집중화하고 원하는 상태에 클러스터가 병합되므로 더 빠른 노드 시작이 가능합니다. 인증서 교체 및 중앙 집중식 인증서 관리는 기본적으로 활성화되어 있습니다.

그림 2.2. 노드 부트스트랩 워크플로 개요



노드 서비스가 시작되면 노드는 클러스터에 참여하기 전에 */etc/origin/node/node.kubeconfig* 파일 및 기타 노드 구성 파일이 있는지 확인합니다. 노드가 없는 경우 노드는 마스터에서 구성을 가져온 다음 클러스터에 결합합니다.

ConfigMaps 는 */etc/origin/node/node-config.yaml* 의 노드 호스트에서 구성 파일을 채우는 클러스터에 노드 구성을 저장하는 데 사용됩니다. 기본 노드 그룹 및 해당 ConfigMap 세트의 정의는 클러스터 설치에서 노드 그룹 및 호스트 맵핑 정의를 참조하십시오 .

노드 부트스트랩 워크플로
자동 노드 부트스트랩 프로세스는 다음 워크플로우를 사용합니다.

1. 기본적으로 클러스터 설치 중에 노드 부트스트랩에 사용하기 위해 **clusterrole, clusterrolebinding** 및 **serviceaccount** 오브젝트 세트가 생성됩니다.

   - **system:node-bootstrapper** 클러스터 역할은 노드 부트스트랩 중에 CSR(인증서 서명 요청)을 생성하는 데 사용됩니다.

     ```
     $ oc describe clusterrole.authorization.openshift.io/system:node-bootstrapper
     ```

     출력 예

     ```
     Name:   system:node-bootstrapper
     Created:  17 hours ago
     Labels:   kubernetes.io/bootstrapping=rbac-defaults
     Annotations:  authorization.openshift.io/system-only=true
       openshift.io/reconcile-protect=false
     Verbs   Non-Resource URLs Resource Names API Groups  Resources
     [create get list watch] []   []  [certificates.k8s.io] [certificatesigningrequests]
     ```

   - 다음 **node-bootstrapper** 서비스 계정이 **openshift-infra** 프로젝트에 생성됩니다.

     ```
     $ oc describe sa node-bootstrapper -n openshift-infra
     ```

     출력 예

     ```
     Name:           node-bootstrapper
     ```

```
Namespace:          openshift-infra
Labels:             <none>
Annotations:        <none>
Image pull secrets:  node-bootstrapper-dockercfg-f2n8r
Mountable secrets:   node-bootstrapper-token-79htp
                    node-bootstrapper-dockercfg-f2n8r
Tokens:              node-bootstrapper-token-79htp
                    node-bootstrapper-token-mqn2q
Events:             <none>
```

- 다음 **system:node-bootstrapper** 클러스터 역할 바인딩은 노드 부트스트랩자 클러스터 역할 및 서비스 계정에 사용됩니다.

```
$ oc describe clusterrolebindings system:node-bootstrapper
```

출력 예

```
Name:   system:node-bootstrapper
Created:  17 hours ago
Labels:   <none>
Annotations:  openshift.io/reconcile-protect=false
Role:   /system:node-bootstrapper
Users:   <none>
Groups:   <none>
ServiceAccounts: openshift-infra/node-bootstrapper
Subjects:  <none>
Verbs   Non-Resource URLs Resource Names API Groups  Resources
[create get list watch] []   []  [certificates.k8s.io] [certificatesigningrequests]
```

2. 클러스터 설치 중에 **openshift-ansible** 설치 프로그램은 OpenShift Container Platform 인증 기관과 다양한 다른 인증서, 키 및 ***kubeconfig*** 파일을 ***/etc/origin/master*** 디렉터리에 생성합니다. 두 개의 참고 파일은 다음과 같습니다.

/etc/origin/master/admin.kubeconfig	**system:admin** 사용자를 사용합니다.
/etc/origin/master/bootstrap.kubeconfig	마스터 이외의 노드 부트스트랩 노드에 사용됩니다.

   a. 설치 프로그램이 다음과 같이 **node-*bootstrapper 서비스 계정을 사용하는 경우*** ***/etc/origin/master/bootstrap*.kubeconfig** 가 생성됩니다.

```
$ oc --config=/etc/origin/master/admin.kubeconfig \
    serviceaccounts create-kubeconfig node-bootstrapper \
    -n openshift-infra
```

   b. 마스터 노드에서 ***/etc/origin/master/admin.kubeconfig*** 는 부트스트랩 파일로 사용되며 ***/etc/origin/node/boostrap.kubeconfig*** 에 복사됩니다. 다른 비마스터 노드에서 ***/etc/origin/master/bootstrap.kubeconfig*** 파일이 각 노드 호스트의

/etc/origin/node/boostrap.kubeconfig 의 다른 모든 노드에 복사됩니다.

c. 그런 다음 */etc/origin/master/bootstrap.kubeconfig* 가 다음과 같이 **--bootstrap-kubeconfig** 플래그를 사용하여 kubelet에 전달됩니다.

```
--bootstrap-kubeconfig=/etc/origin/node/bootstrap.kubeconfig
```

3. kubelet은 제공된 */etc/origin/node/bootstrap.kubeconfig* 파일로 처음 시작됩니다. 내부적으로 초기 연결 후 kubelet은 CSR(인증서 서명 요청)을 생성하여 마스터로 보냅니다.

4. CSR은 컨트롤러 관리자(특히 인증서 서명 컨트롤러)를 통해 확인하고 승인합니다. 승인되면 kubelet 클라이언트 및 서버 인증서가 */etc/origin/node/ceritificates* 디렉터리에 생성됩니다. 예를 들면 다음과 같습니다.

```
# ls -al /etc/origin/node/certificates/
```

출력 예

```
total 12
drwxr-xr-x. 2 root root  212 Jun 18 21:56 .
drwx------. 4 root root  213 Jun 19 15:18 ..
-rw-------. 1 root root 2826 Jun 18 21:53 kubelet-client-2018-06-18-21-53-15.pem
-rw-------. 1 root root 1167 Jun 18 21:53 kubelet-client-2018-06-18-21-53-45.pem
lrwxrwxrwx. 1 root root   68 Jun 18 21:53 kubelet-client-current.pem ->
/etc/origin/node/certificates/kubelet-client-2018-06-18-21-53-45.pem
-rw-------. 1 root root 1447 Jun 18 21:56 kubelet-server-2018-06-18-21-56-52.pem
lrwxrwxrwx. 1 root root   68 Jun 18 21:56 kubelet-server-current.pem ->
/etc/origin/node/certificates/kubelet-server-2018-06-18-21-56-52.pem
```

5. CSR 승인 후 *node.kubeconfig* 파일이 */etc/origin/node/node.kubeconfig* 에 생성됩니다.

6. kubelet은 */etc/origin/node/node.kubeconfig* 파일과 */etc/origin/node/certificates/* 디렉터리의 인증서로 다시 시작됩니다. 이 인증서는 클러스터에 참여할 준비가 되었습니다.

노드 설정 워크플로

노드의 구성을 소싱하려면 다음 워크플로우를 사용합니다.

1. 처음에 노드의 kubelet은 노드 프로비저닝 시 생성된 */etc/origin/ node/ 디렉터리에 부트스트랩 구성 파일 bootstrap-node -config.yaml* 로 시작됩니다.

2. 각 노드에서 노드 서비스 파일은 */usr/local/bin/* 디렉터리에 있는 로컬 스크립트 *openshift-node* 를 사용하여 제공된 *bootstrap-node-config.yaml* 을 사용하여 kubelet을 시작합니다.

3. 각 마스터에서 */etc/origin/node/pods* 디렉터리에는 마스터에서 정적 포드로 생성되는 **apiserver,controller** 및 **etcd** 의 Pod 매니페스트가 포함되어 있습니다.

4. 클러스터 설치 중에 각 노드에 동기화 Pod를 생성하는 동기화 DaemonSet이 생성됩니다. 동기화 포드는 */etc/sysconfig/atomic-openshift-node* 파일의 변경 사항을 모니터링합니다.
BOOTSTRAP_CONFIG_NAME 이 설정되도록 특히 감시합니다.
BOOTSTRAP_CONFIG_NAME 은 **openshift-ansible** 설치 프로그램에서 설정하며 노드가 속하는 노드 구성 그룹에 따라 ConfigMap의 이름입니다.
기본적으로 설치 프로그램은 다음 노드 구성 그룹을 생성합니다.

- **node-config-master**

- **node-config-infra**
- **node-config-compute**
- **node-config-all-in-one**
- **node-config-master-infra**

각 그룹의 ConfigMap은 **openshift-node** 프로젝트에 생성됩니다.

5. 동기화 Pod는 **BOOTSTRAP_CONFIG_NAME** 에 설정된 값을 기반으로 적절한 ConfigMap을 추출합니다.
6. 동기화 Pod는 ConfigMap 데이터를 kubelet 구성으로 변환하고 해당 노드 호스트에 대한 */etc/origin/node/node-config.yaml* 을 생성합니다. 이 파일(또는 파일의 초기 생성)을 변경하면 kubelet이 다시 시작됩니다.

노드 구성 수정
노드의 구성은 **openshift-node** 프로젝트에서 적절한 ConfigMap을 편집하여 수정합니다. */etc/origin/node/node-config.yaml* 은 직접 수정해서는 안 됩니다.

예를 들어 node **-config-compute 그룹에 있는 노드**의 경우 다음을 사용하여 ConfigMap을 편집합니다.

```
$ oc edit cm node-config-compute -n openshift-node
```

2.2. 컨테이너 레지스트리

2.2.1. 개요

OpenShift Container Platform은 Docker Hub, 타사가 실행하는 프라이빗 레지스트리, 통합된 OpenShift Container Platform 레지스트리 등 컨테이너 이미지 레지스트리 API를 이미지의 소스로 구현하는 모든 서버를 활용할 수 있습니다.

2.2.2. 통합된 OpenShift Container Registry

OpenShift Container Platform은 필요에 따라 새 이미지 리포지토리를 자동으로 프로비저닝하는 기능을 추가하는 OCR(OpenShift *Container Registry*)이라는 통합 컨테이너 이미지 레지스트리를 제공합니다. 이를 통해 사용자는 애플리케이션 빌드에 결과 이미지를 푸시할 수 있는 기본 제공 위치를 제공합니다.

새 이미지를 OCR로 내보낼 때마다 레지스트리는 OpenShift Container Platform에 새 이미지에 대해 알리고 네임스페이스, 이름, 이미지 메타데이터와 같은 해당 이미지에 대한 모든 정보를 전달합니다. 다른 OpenShift Container Platform은 새 이미지에 반응하여 새 빌드 및 배포를 생성합니다.

OCR은 빌드 및 배포 통합 없이 컨테이너 이미지 레지스트리 역할을 하는 독립 실행형 구성 요소로도 배포할 수 있습니다. 자세한 내용은 OpenShift Container Registry 독립 실행형 배포 설치를 참조하십시오.

2.2.3. 타사 레지스트리

OpenShift Container Platform은 타사 레지스트리의 이미지를 사용하여 컨테이너를 생성할 수 있지만 이러한 레지스트리가 통합된 OpenShift Container Platform 레지스트리와 동일한 이미지 알림 지원을 제공하지는 않습니다. 이 경우 OpenShift Container Platform은 이미지 스트림 생성 시 원격 레지스트리에서 태그를 가져옵니다. **oc import-image <stream>**을 실행하여 가져온 태그를 간단하게 업데이트할 수 있습니다. 새 이미지가 감지되면 이전에 설명된 빌드 및 배포에 대한 응답이 발생합니다.

2.2.3.1. 인증

OpenShift Container Platform은 레지스트리와 통신하여 사용자가 지정한 인증 정보를 사용하여 개인 이미지 저장소에 액세스할 수 있습니다. 이를 통해 OpenShift Container Platform은 프라이빗 리포지토리에서 이미지 푸시 및 풀 작업을 수행할 수 있습니다. 인증 항목에는 자세한 정보가 있습니다.

2.2.4. Red Hat Quay 레지스트리

엔터프라이즈급 컨테이너 이미지 레지스트리가 필요한 경우 Red Hat Quay는 호스팅 서비스와 자체 데이터 센터 또는 클라우드 환경에 설치할 수있는 소프트웨어로 사용할 수 있습니다. Red Hat Quay의 고급 레지스트리에는 리전 복제, 이미지 스캔 및 이미지 롤백 기능이 포함되어 있습니다.

Quay.io 사이트를 방문하여 호스팅된 Quay 레지스트리 계정을 설정합니다. 그런 다음 Quay 튜토리얼 에 따라 Quay 레지스트리에 로그인하고 이미지 관리를 시작합니다. 또는 고유한 Red Hat Quay 레지스트리 설정에 대한 정보는 Red Hat Quay로 시작하기 를 참조하십시오.

원격 컨테이너 이미지 레지스트리와 마찬가지로 OpenShift Container Platform에서 Red Hat Quay 레지스트리에 액세스할 수 있습니다. Red Hat Quay에 보안 레지스트리로 액세스하기 위한 자격 증명을 설정하는 방법을 알아보려면 다른 보안 레지스트리의 참조 이미지에 포드 허용을 참조하십시오 .

2.2.5. 인증 활성화 Red Hat 레지스트리

Red Hat Container Catalog를 통해 사용할 수 있는 모든 컨테이너 이미지는 이미지 레지스트리인 **registry.access.redhat.com** 에 호스팅됩니다. OpenShift Container Platform 3.11 Red Hat Container Catalog를 **registry.access.redhat.com에서 registry. redhat.io** 로 이동했습니다.

새 레지스트리 **registry.redhat.io** 는 OpenShift Container Platform의 이미지 및 호스팅 콘텐츠에 액세스할 수 있는 인증이 필요합니다. 새 레지스트리로 마이그레이션한 후 기존 레지스트리를 일정 기간 동안 사용할 수 있습니다.

참고

OpenShift Container Platform은 **registry.redhat.io**에서 이미지를 가져 오므로 이를 사용할 수 있도록 클러스터를 설정해야합니다.

새 레지스트리는 다음과 같은 방법으로 인증에 표준 OAuth 메커니즘을 사용합니다.

- **인증 토큰**: 관리자가 생성한 토큰은 시스템에 컨테이너 이미지 레지스트리에 대한 인증 기능을 제공하는 서비스 계정입니다. 서비스 계정은 사용자 계정 변경의 영향을 받지 않으므로 인증에 토큰을 사용하는 것은 안정적이고 유연한 인증 방법입니다. 이는 프로덕션 클러스터에 대해 지원되는 유일한 인증 옵션입니다.

- **웹 사용자 이름 및 암호**: 이는 **access.redhat.com**과 같은 리소스에 로그인하는 데 사용하는 표준 인증 정보 집합입니다. OpenShift Container Platform에서 이 인증 방법을 사용할 수는 있지만 프로덕션 배포에는 지원되지 않습니다. 이 인증 방법은 OpenShift Container Platform 외부의 독립형 프로젝트에서만 사용해야합니다.

사용자 이름 및 암호 또는 인증 토큰 중 하나의 자격 **증명으로 Docker** 로그인을 사용하여 새 레지스트리의 콘텐츠에 액세스할 수 있습니다.

모든 이미지 스트림은 새 레지스트리를 가리킵니다. 새 레지스트리에는 액세스에 대한 인증이 필요하므로 OpenShift 네임스페이스에 **imagestreamsecret** 이라는 새 시크릿이 있습니다.

다음 두 위치에 인증 정보를 배치해야 합니다.

- **OpenShift 네임 스페이스**: OpenShift 네임스페이스의 이미지 스트림을 가져올 수 있도록 인증 정보가 OpenShift 네임스페이스에 있어야 합니다.
- **호스트**: Kubernetes에서 이미지를 가져올 때 호스트의 인증 정보를 사용하므로 호스트에 인증 정보가 있어야합니다.

새 레지스트리에 액세스하려면 다음을 수행합니다.

- 이미지 가져오기 시크릿 **imagestreamsecret** 이 OpenShift 네임스페이스에 있는지 확인합니다. 해당 시크릿에는 새 레지스트리에 액세스할 수 있는 인증 정보가 있습니다.
- 모든 클러스터 노드에 마스터에서 복사된 **/var/lib/origin/.docker/config.json** 이 Red Hat 레지스트리에 액세스할 수 있도록 합니다.

2.3. 웹 콘솔

2.3.1. 개요

OpenShift Container Platform 웹 콘솔은 웹 브라우저에서 액세스할 수 있는 사용자 인터페이스입니다. 개발자는 웹 콘솔을 사용하여 프로젝트의 콘텐츠를 시각화, 탐색 및 관리할 수 있습니다 .

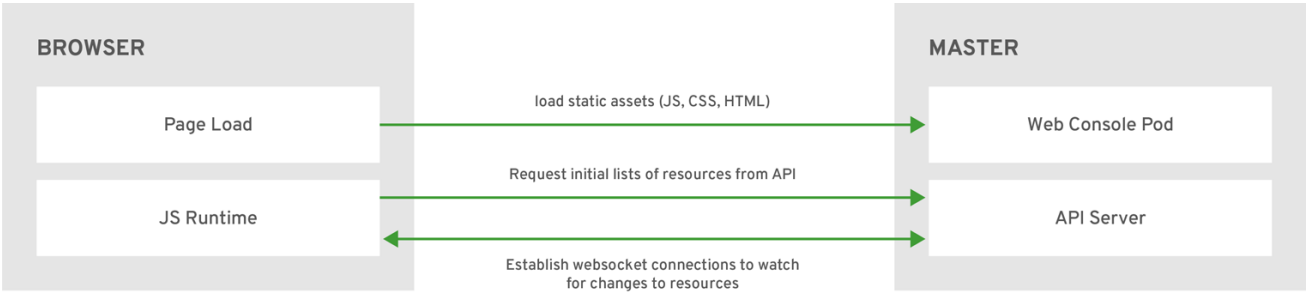
참고

웹 콘솔을 사용하려면 JavaScript가 활성화되어 있어야합니다. WebSockets을 지원하는 웹 브라우저를 사용하는 것이 좋습니다.

웹 콘솔은 마스터에서 포드로 실행됩니다. Pod에서는 웹 콘솔을 실행하는 데 필요한 정적 환경을 제공합니다. 관리자는 확장 기능을 사용하여 웹 콘솔을 사용자 지정할 수도 있습니다. 그러면 웹 콘솔이 로드될 때 스크립트를 실행하고 사용자 지정 스타일시트를 로드할 수 있습니다.

브라우저에서 웹 콘솔에 액세스하면 먼저 필요한 모든 정적 자산을 로드합니다. 그런 다음 **openshift start 옵션 --public-master에서 정의한 값을 사용하거나 openshift - web- console 네임스페이스에** 정의된 **webconsole-config** 구성 맵의 관련 매개변수 **masterPublicURL** 에서 OpenShift Container Platform API에 요청합니다. 웹 콘솔에서는 WebSockets를 사용하여 API 서버와의 지속적인 연결을 유지하고 사용 가능한 즉시 업데이트된 정보를 받습니다.

그림 2.3. 웹 콘솔 요청 아키텍처



OPENSHIFT_415489_0618

웹 콘솔에 대해 구성된 호스트 이름과 IP 주소는 브라우저에서 요청을 교차 원본으로 간주하는 경우에도 API 서버에 안전하게 액세스하도록 허용 목록에 표시됩니다. 다른 호스트 이름을 사용하여 웹 애플리케이션에서 API 서버에 액세스하려면 **openshift start** 또는 관련 마스터 구성 파일 **corsAllowedOrigin s에서 --cors-allowed-origin s 옵션을 지정하여 해당 호스트 이름을 허용해야 합니다.**

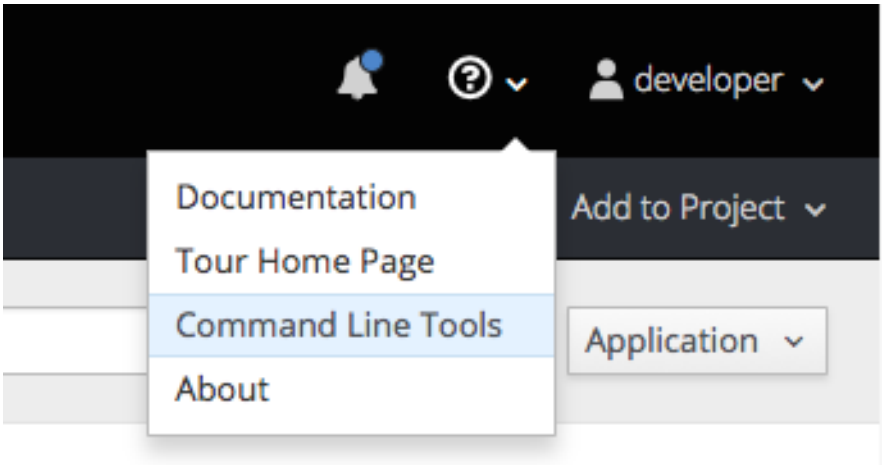
corsAllowedOrigins 매개 변수는 구성 필드에 의해 제어됩니다. 해당 값에 고정 또는 이스케이핑이 수행되지 않습니다. 다음은 호스트 이름 및 이스케이프 지점을 고정할 수 있는 방법의 예입니다.

```
corsAllowedOrigins:
- (?i)//my\.subdomain\.domain\.com(:|\z)
```

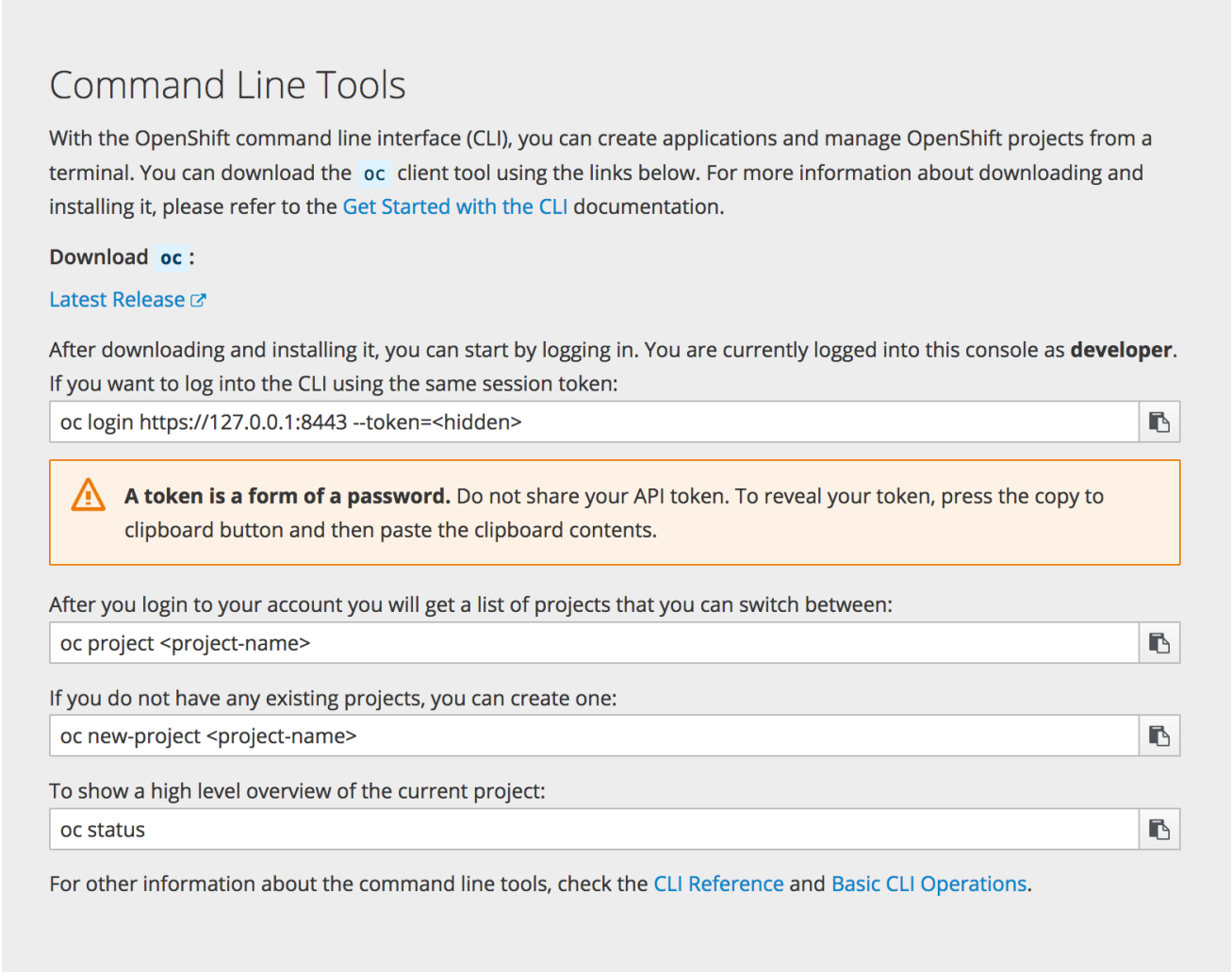
- **(?i)**는 대소문자를 구분하지 않습니다.
- **는** 도메인의 시작 부분에 고정되고 **http: 또는 https:** 뒤에 있는 이중 슬래시와 일치합니다.
- **\.**은 도메인 이름에서 점을 이스케이프합니다.
- **(:|\z)**는 도메인 이름 **(\z)**의 끝 또는 포트 구분 기호 **(:)**과 일치하는지 확인합니다.

2.3.2. CLI 다운로드

웹 콘솔의 도움말 아이콘에서 CLI 다운로드에 액세스할 수 있습니다.



클러스터 관리자는 이러한 링크를 추가로 사용자 지정할 수 있습니다 .



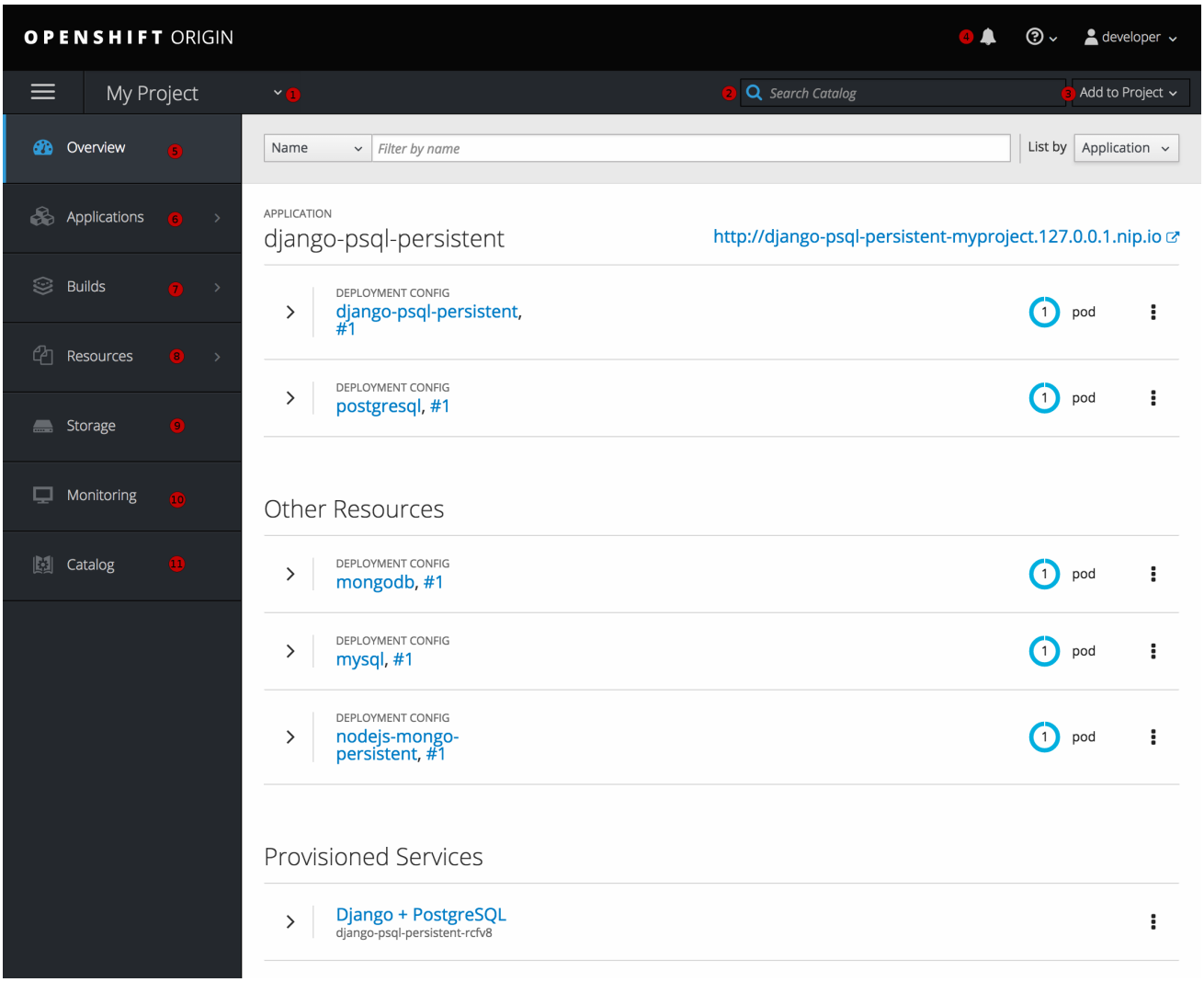
2.3.3. 브라우저 요구 사항

OpenShift Container Platform 에 대해 테스트된 통합을 검토합니다.

2.3.4. 프로젝트 개요

로그인 후 웹 콘솔에서는 현재 선택한 프로젝트에 대한 개요를 개발자에게 제공합니다.

그림 2.4. 웹 콘솔 프로젝트 개요



프로젝트 선택기를 사용하면 액세스할 수 있는 프로젝트를 전환 할 수 있습니다.

프로젝트 내에서 서비스를 빠르게 찾으려면 검색 기준을 입력하십시오.

서비스 카탈로그에서 소스 리포지토리 또는 서비스를 사용하여 새 애플리케이션을 만듭니다.

프로젝트 관련 알림.

Overview(개요) 탭(현재 선택됨)은 각 구성 요소에 대한 높은 수준의 보기로 프로젝트 내용을 시각화합니다.

애플리케이션 탭: 배포, 포드, 서비스 및 경로에 대한 작업을 검색하고 수행합니다.

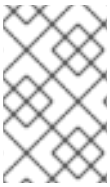
Build 탭: 빌드 및 이미지 스트림에서 작업을 검색하고 수행합니다.

Resources 탭: 현재 할당량 소비 및 기타 리소스를 확인합니다.

Storage 탭: 영구 볼륨 클레임을 보고 애플리케이션의 스토리지를 요청합니다.

모니터링 탭: 빌드, 포드 및 배포에 대한 로그와 프로젝트의 모든 오브젝트에 대한 이벤트 알림을 확인합니다.

Catalog 탭: 프로젝트 내에서 신속하게 카탈로그로 이동합니다.



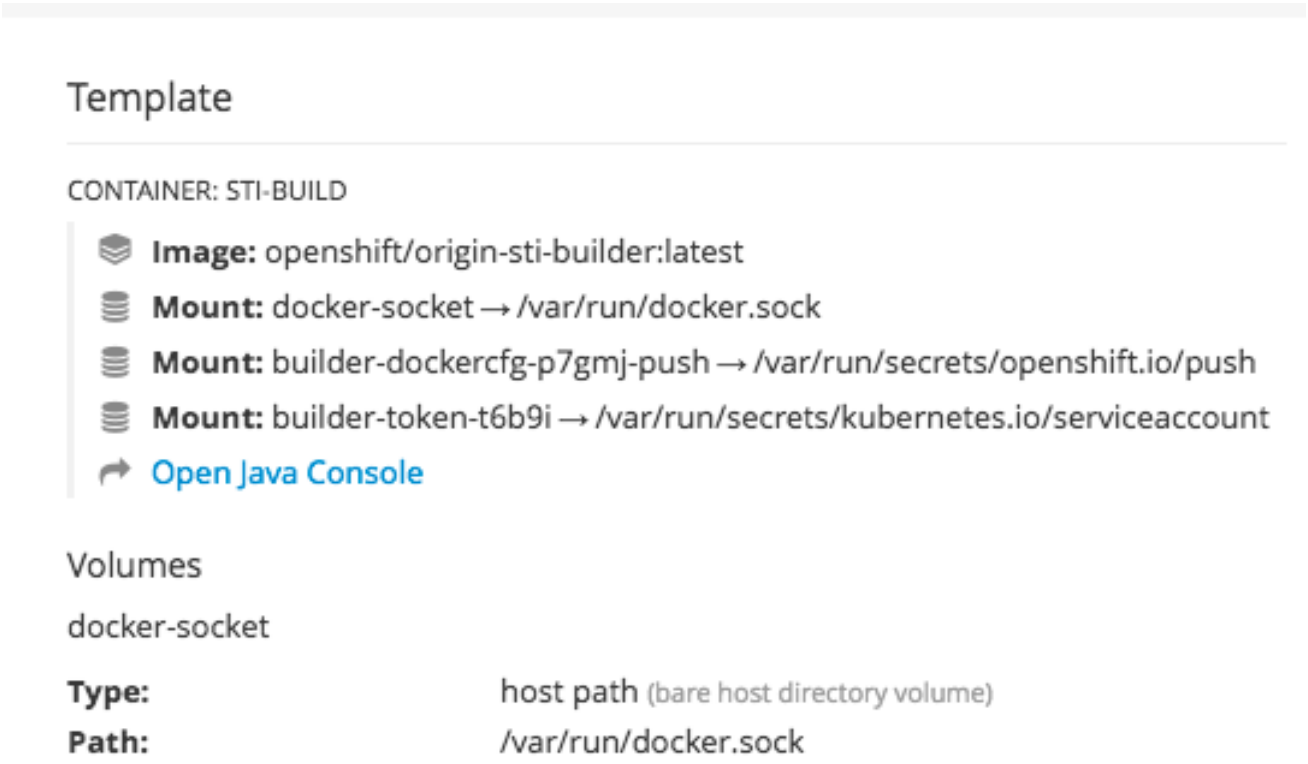
참고

Cockpit 은 개발 환경을 모니터링하는 데 도움이 되도록 에서 자동으로 설치 및 활성화됩니다. Red Hat Enterprise Linux Atomic Host: Cockpit 시작하기는 Cockpit 사용에 대한 자세한 정보를 제공합니다.

2.3.5. JVM 콘솔

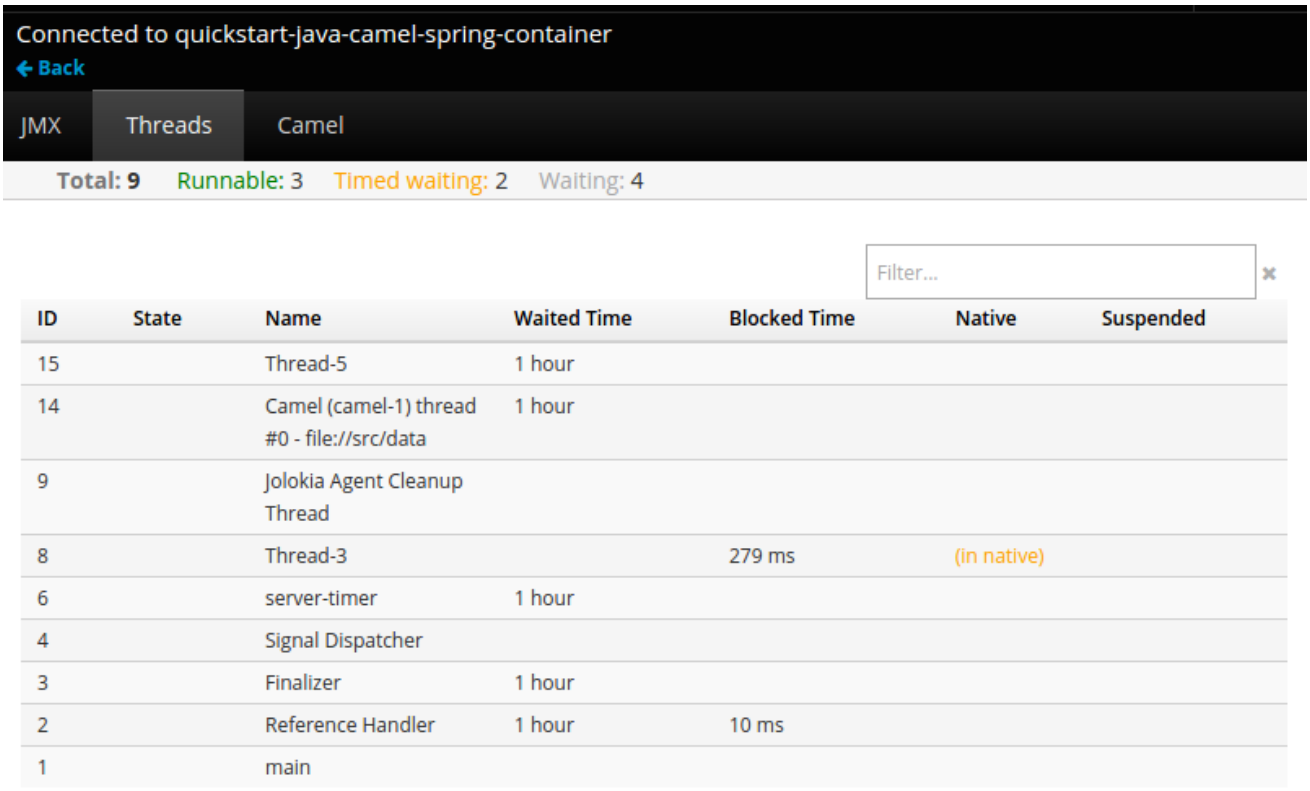
Java 이미지를 기반으로 하는 Pod의 경우 웹 콘솔은 관련된 통합 구성 요소를 보고 관리하기 위해 hawt.io기반 JVM 콘솔에 대한 액세스도 노출합니다. 컨테이너에 **jolokia** 라는 포트가 있는 경우 **연결** 링크가 *Browse → Pods 페이지의 Pod* 세부 정보에 표시됩니다.

그림 2.5. JVM 콘솔에 연결되는 Pod



JVM 콘솔에 연결하면 연결된 포드와 관련된 구성 요소에 따라 다른 페이지가 표시됩니다.

그림 2.6. JVM 콘솔



ID	State	Name	Waited Time	Blocked Time	Native	Suspended
15		Thread-5	1 hour			
14		Camel (camel-1) thread #0 - file://src/data	1 hour			
9		Jolokia Agent Cleanup Thread				
8		Thread-3		279 ms	(in native)	
6		server-timer	1 hour			
4		Signal Dispatcher				
3		Finalizer	1 hour			
2		Reference Handler	1 hour	10 ms		
1		main				

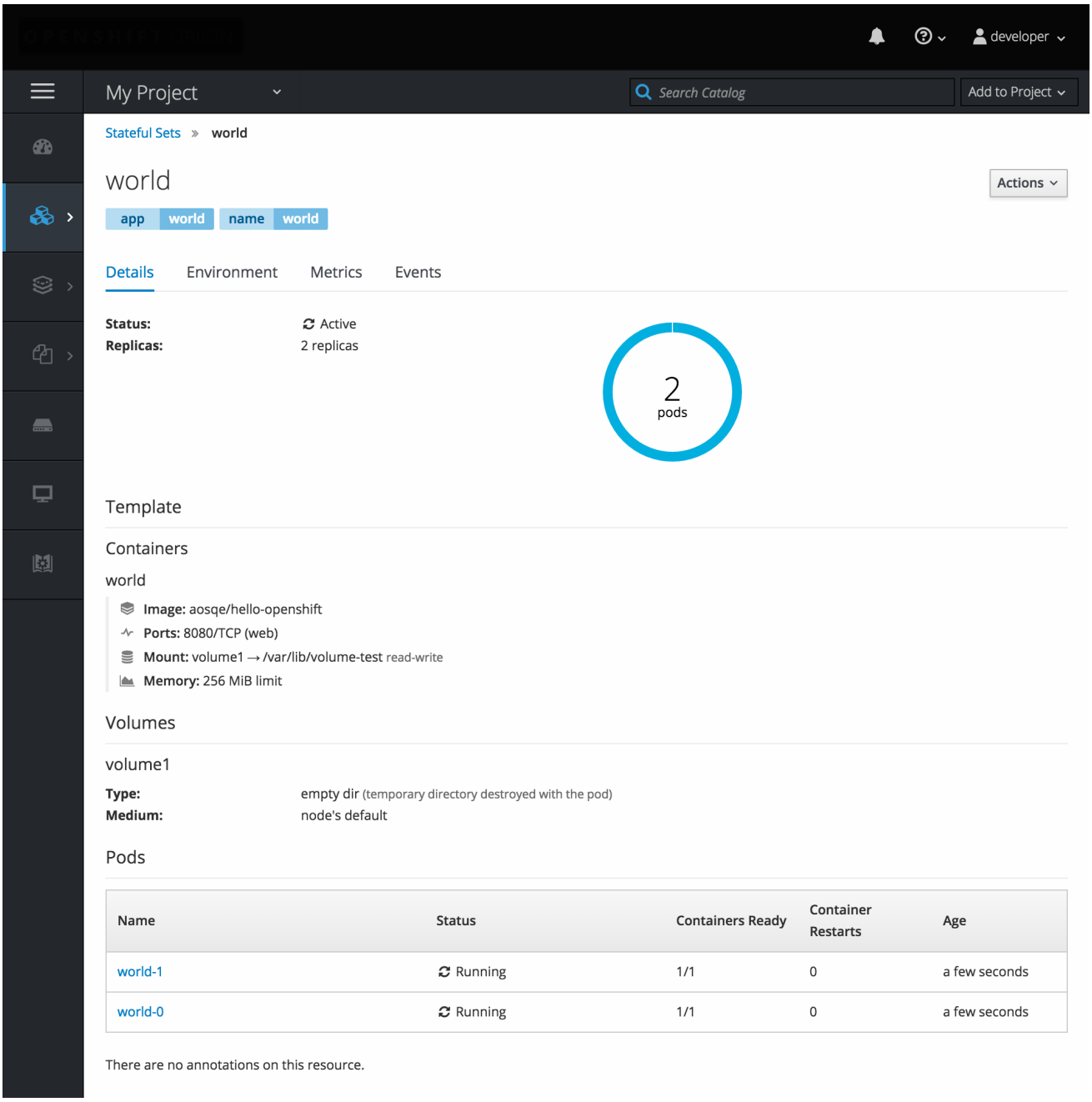
다음 페이지를 사용할 수 있습니다.

페이지	설명
JMX	JMX 도메인 및 mbean을 보고 관리합니다.
스레드	스레드 상태를 보고 모니터링합니다.
ActiveMQ	Apache ActiveMQ 브로커를 보고 관리합니다.
Camel	Apache Camel 경로 및 종속성을 보고 관리합니다.
OSGi	JBoss Fuse OSGi 환경 보기 및 관리.

2.3.6. StatefulSets

StatefulSet 컨트롤러는 Pod에 고유한 ID를 제공하고 배포 및 스케일링 순서를 결정합니다. **StatefulSet**는 고유한 네트워크 식별자, 영구저장장치, 정상 배포 및 확장, 정상적인 삭제 및 종료에 유용합니다.

그림 2.7. OpenShift Container Platform의 StatefulSet



3장. 핵심 개념

3.1. 개요

다음 주제는 OpenShift Container Platform을 사용할 때 발생하는 핵심 개념 및 오브젝트에 대한 고급 아키텍처 정보를 제공합니다. 이러한 오브젝트 중 상당수는 Kubernetes에서 제공되며 OpenShift Container Platform에서 확장하여 기능이 풍부한 개발 라이프사이클 플랫폼을 제공합니다.

- 컨테이너 및 이미지는 애플리케이션 배포를 위한 구성 요소입니다.
- 포드 및 서비스를 사용하면 컨테이너가 서로 통신하고 프록시 연결과 통신할 수 있습니다.
- 프로젝트와 사용자는 커뮤니티에서 콘텐츠를 함께 구성하고 관리할 수 있는 공간을 제공합니다.
- 빌드 및 이미지 스트림을 사용하면 작업 이미지를 빌드하고 새 이미지에 대응할 수 있습니다.
- 배포 시 소프트웨어 개발 및 배포 라이프사이클에 대한 확장된 지원이 추가되었습니다.
- 경로가 서비스를 전 세계에 알립니다.
- 템플릿 을 사용하면 사용자 지정 매개 변수를 기반으로 한 번에 많은 오브젝트를 생성할 수 있습니다.

3.2. 컨테이너 및 이미지

3.2.1. 컨테이너

OpenShift Container Platform 애플리케이션의 기본 단위는 *컨테이너*라고 합니다. Linux 컨테이너 기술 은 실행 중인 프로세스를 격리하는 데 필요한 간단한 메커니즘으로, 이를 통해 실행 중인 프로세스는 지정된 리소스하고만 상호 작용하도록 제한됩니다.

많은 애플리케이션 인스턴스는 서로의 프로세스, 파일, 네트워크 등을 보지 않으며 단일 호스트의 컨테이너에서 실행될 수 있습니다. 컨테이너를 임의의 워크로드에 사용할 수는 있지만 일반적으로 각 컨테이너는 웹 서버나 데이터베이스 같은 단일 서비스(흔히 "마이크로 서비스"라고 함)를 제공합니다.

Linux 커널은 수년간 컨테이너 기술을 위한 기능을 통합해 왔습니다. 최근에 Docker 프로젝트는 호스트의 Linux 컨테이너용 편리한 관리 인터페이스를 개발했습니다. OpenShift Container Platform 및 Kubernetes는 다중 호스트 설치에서 Docker 형식의 컨테이너를 오케스트레이션하는 기능을 추가합니다.

OpenShift Container Platform을 사용할 때 Docker CLI 또는 서비스와 직접 상호 작용하지는 않지만 OpenShift Container Platform에서 역할과 애플리케이션이 컨테이너 내에서 작동하는 방식을 이해하는 데 필요한 기능 및 용어를 이해하는 것이 중요합니다. **Docker** RPM은 RHEL 7과 CentOS 및 Fedora의 일부로 제공되므로 OpenShift Container Platform과 별도로 시험할 수 있습니다. 가이드 소개는 Red Hat Systems의 Docker Formatted Container Images 문서를 참조하십시오.

3.2.1.1. Init 컨테이너

포드에는 애플리케이션 컨테이너 외에도 init 컨테이너가 있을 수 있습니다. Init Container를 사용하면 설정 스크립트 및 바인딩 코드를 재구성할 수 있습니다. init 컨테이너는 항상 완료되기 때문에 일반 컨테이너와 다릅니다. 각 init 컨테이너는 다음 컨테이너를 시작하기 전에 성공적으로 완료해야 합니다.

자세한 내용은 Pod 및 Service를 참조하십시오.

3.2.2. 이미지

OpenShift Container Platform의 컨테이너는 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.

Docker CLI를 직접 사용하여 이미지를 빌드할 수 있지만 OpenShift Container Platform에서는 기존 이미지에 코드 또는 구성을 추가하여 새 이미지를 생성하는 데 도움이 되는 빌더 이미지도 제공합니다.

시간이 지남에 따라 애플리케이션이 개발되기 때문에 단일 이미지 이름은 실제로 "동일한" 이미지의 다양한 버전을 참조할 수 있습니다. 서로 다른 각 이미지는 해시 (예: **fd44297e2ddb050ec4f...**) 일반적으로 12자로 단축됩니다(예: **fd44297e2ddb**).

이미지 버전 태그 정책
버전 번호 대신 Docker 서비스를 사용하면 원하는 이미지를 추가로 지정하는 이미지 이름 외에도 태그 (예: **v1, v2.1,GA** 또는 기본 **최신**)를 적용할 수 있으므로 as **centos(가장 최신 태그 사용),centos :centos7** 또는 **fd44297e2ddb** 와 동일한 이미지를 볼 수 있습니다.

주의

공식 OpenShift Container Platform 이미지에 **latest** 태그를 사용하지 마십시오. **openshift3/** 로 시작하는 이미지입니다. **latest** 는 **3.10** 또는 **3.11** 과 같은 여러 버전을 참조할 수 있습니다.

이미지에 태그를 지정하는 방법에 따라 업데이트 정책이 지정됩니다. 세부적인 내용이 있을수록 이미지가 업데이트되는 빈도가 줄어듭니다. 다음을 사용하여 선택한 OpenShift Container Platform 이미지 정책을 확인합니다.

vX.Y

vX.Y 태그는 X.Y.Z-<number>를 가리킵니다. 예를 들어 **registry-console** 이미지가 v3.11로 업데이트되면 3.11.1-8과 같은 최신 3.11.Z-<number> 태그를 가리킵니다.

X.Y.Z

위의 vX.Y 예와 유사하게 X.Y.Z 태그는 최신 X.Y.Z-<number>를 가리킵니다. 예를 들어 3.11.1은 3.11.1-8을 가리킵니다.

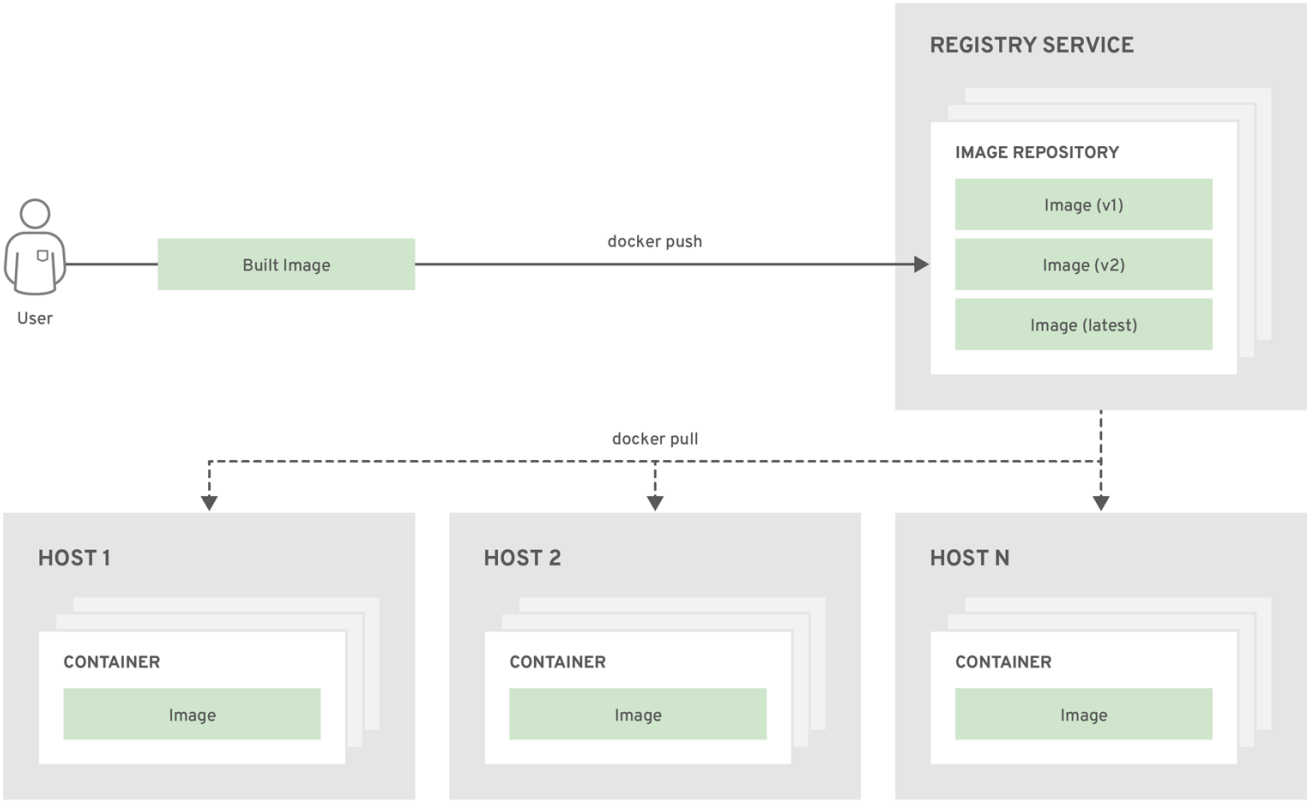
X.Y.Z-<number>

태그는 고유하며 변경되지 않습니다. 이 태그를 사용하면 이미지가 업데이트되면 이미지가 업데이트되지 않습니다. 예를 들어, 3.11.1-8은 이미지가 업데이트되는 경우에도 항상 3.11.1-8을 가리킵니다.

3.2.3. 컨테이너 이미지 레지스트리

컨테이너 이미지 레지스트리는 Docker 형식 컨테이너 이미지를 저장하고 검색하는 서비스입니다. 레지스트리에는 하나 이상의 이미지 리포지토리로 이루어진 컬렉션이 포함되어 있습니다. 각 이미지 리포지토리에는 하나 이상의 태그된 이미지가 포함되어 있습니다. Docker는 자체 레지스트리인 Docker Hub 를 제공하며 개인 또는 타사 레지스트리도 사용할 수 있습니다. Red Hat은 **registry.redhat.io**의 레지스트리를 구독자에게 제공합니다. OpenShift Container Platform은 사용자 정의 컨테이너 이미지 관리에 사용되는 자체 내부 레지스트리도 제공할 수 있습니다.

컨테이너, 이미지 및 레지스트리 간의 관계는 다음 다이어그램에 표시되어 있습니다.



3.3. POD 및 서비스

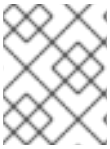
3.3.1. Pod

OpenShift Container Platform은 하나의 호스트에 함께 배포되는 하나 이상의 컨테이너 이자 정의, 배포 및 관리할 수 있는 최소 컴퓨팅 단위인 *Pod* 의 Kubernetes 개념을 활용합니다.

Pod는 컨테이너에 대한 머신 인스턴스(실제 또는 가상)와 대략적으로 동일합니다. 각 Pod에는 자체 내부 IP 주소가 할당되므로 해당 Pod가 전체 포트 공간을 소유하고 Pod 내의 컨테이너는 로컬 스토리지와 네트워킹을 공유할 수 있습니다.

Pod에는 라이프사이클이 정의되어 있으며 노드에서 실행되도록 할당된 다음 컨테이너가 종료되거나 기타 이유로 제거될 때까지 실행됩니다. Pod는 정책 및 종료 코드에 따라 종료 후 제거되거나 컨테이너 로그에 대한 액세스를 활성화하기 위해 유지될 수 있습니다.

OpenShift Container Platform에서는 대체로 Pod를 변경할 수 없는 것으로 취급합니다. 실행 중에는 Pod 정의를 변경할 수 없습니다. OpenShift Container Platform은 기존 Pod를 종료한 후 수정된 구성이나 기본 이미지 또는 둘 다 사용하여 Pod를 다시 생성하는 방식으로 변경 사항을 구현합니다. Pod를 다시 생성하면 확장 가능한 것으로 취급되고 상태가 유지되지 않습니다. 따라서 일반적으로 포드는 사용자가 직접 관리하는 대신 상위 수준 컨트롤러에서 관리해야 합니다.



참고

OpenShift Container Platform 노드 호스트당 최대 pod 수는 클러스터 최대값을 참조하십시오.

주의

복제 컨트롤러에서 관리하지 않는 베어 Pod는 노드 중단 시 다시 예약되지 않습니다.

다음은 실제로 OpenShift Container Platform 인프라(통합 컨테이너 이미지 레지스트리)의 일부인 장기 실행 서비스를 제공하는 Pod 정의의 예입니다. 이 예제에서는 Pod의 다양한 기능을 보여줍니다. 대부분 다른 주제에서 설명하므로 여기에서는 간단히 언급합니다.

Pod 오브젝트 정의(YAML)

```
apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }
  labels:                                    (1)
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1-           (2)
spec:
  containers:                                (3)
  - env:                                     (4)
    - name: OPENSHIFT_CA_DATA
      value: ...
    - name: OPENSHIFT_CERT_DATA
      value: ...
    - name: OPENSHIFT_INSECURE
      value: "false"
    - name: OPENSHIFT_KEY_DATA
      value: ...
    - name: OPENSHIFT_MASTER
      value: https://master.example.com:8443
    image: openshift/origin-docker-registry:v0.6.2 (5)
    imagePullPolicy: IfNotPresent
    name: registry
    ports:                                   (6)
    - containerPort: 5000
      protocol: TCP
    resources: {}
    securityContext: { ... }                 (7)
    volumeMounts:                            (8)
    - mountPath: /registry
      name: registry-storage
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-br6yz
      readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-at06w
  restartPolicy: Always                      (9)
```

```
serviceAccount: default                    10
volumes:                                   11
- emptyDir: {}
  name: registry-storage
- name: default-token-br6yz
  secret:
    secretName: default-token-br6yz
```

1. Pod는 단일 작업에서 Pod 그룹을 선택하고 관리하는 데 사용할 수 있는 하나 이상의 레이블을 사용하여 "태그"할 수 있습니다. 레이블은 **메타데이터** 해시의 키/값 형식으로 저장됩니다. 이 예에서 하나의 레이블은 **docker-registry=default** 입니다.

2. Pod에는 네임스페이스 내에 고유한 이름이 있어야 합니다. 포드 정의는 **generateName** 속성이 있는 이름의 기반을 지정할 수 있으며, 고유 이름을 생성하기 위해 임의 문자가 자동으로 추가됩니다.

3. **containers** 는 컨테이너 정의의 배열을 지정합니다. 이 경우 (대부분과 동일) 하나뿐입니다.

4. 각 컨테이너에 필요한 값을 전달하도록 환경 변수를 지정할 수 있습니다.

5. 포드의 각 컨테이너는 자체 Docker 형식 컨테이너 이미지에서 인스턴스화됩니다.

6. 컨테이너는 포드의 IP에서 사용할 수 있는 포트에 바인딩할 수 있습니다.

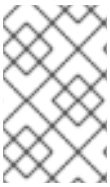
7. OpenShift Container Platform은 컨테이너에 대한 보안 컨텍스트를 정의합니다. 보안 컨텍스트는 권한 있는 컨테이너로 실행하거나 선택한 사용자로 실행할 수 있는지의 여부 등을 지정합니다. 기본 컨텍스트는 매우 제한적이지만 필요에 따라 관리자가 수정할 수 있습니다.

8. 컨테이너는 컨테이너 내에서 외부 스토리지 볼륨을 마운트해야 하는 위치를 지정합니다. 이 경우 레지스트리의 데이터를 저장하는 볼륨이 있으며 레지스트리에서 OpenShift Container Platform API에 요청하는 데 필요한 인증 정보에 액세스하기 위한 볼륨이 있습니다.

9. Pod는 가능한 값 **Always**,**OnFailure** 및 **Never** 로 정책을 다시 시작합니다. 기본값은 **Always**입니다.

10. OpenShift Container Platform API에 대해 요청하는 Pod는 요청 시 포드에서 인증해야 하는 서비스 계정 사용자를 지정하는 **serviceAccount** 필드가 있는 일반적인 패턴입니다. 따라서 사용자 정의 인프라 구성 요소에 대한 액세스 권한을 세부적으로 제어할 수 있습니다.

11. Pod는 사용할 컨테이너에서 사용할 수 있는 스토리지 볼륨을 정의합니다. 이 경우 레지스트리 스토리지의 임시 볼륨과 서비스 계정 자격 증명을 포함하는 **secret** 볼륨을 제공합니다.



참고

이 Pod 정의에는 Pod가 생성되고 해당 라이프사이클이 시작된 후 OpenShift Container Platform에 의해 자동으로 채워지는 특성은 포함되지 않습니다. Kubernetes Pod 설명서 에는 Pod의 기능 및 용도에 대한 세부 정보가 있습니다.

3.3.1.1. Pod 재시작 정책

Pod 재시작 정책에 따라 해당 Pod의 컨테이너가 종료될 때 OpenShift Container Platform에서 응답하는 방법이 결정됩니다. 정책은 해당 포드의 모든 컨테이너에 적용됩니다.

가능한 값은 다음과 같습니다.

- **Always** - Pod를 다시 시작할 때까지 급격한 백오프 지연(10초, 20초, 40초)을 사용하여 Pod에서 성공적으로 종료된 컨테이너를 지속적으로 재시작합니다. 기본값은 **Always**입니다.

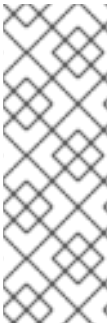
- **OnFailure** - 급격한 백오프 지연(10초, 20초, 40초)을 5분으로 제한하여 Pod에서 실패한 컨테이너를 재시작합니다.
- **Never** - Pod에서 종료되거나 실패한 컨테이너를 재시작하지 않습니다. Pod가 즉시 실패하고 종료됩니다.

노드에 바인딩되면 Pod가 다른 노드에 바인딩되지 않습니다. 따라서 노드 장애 시 Pod가 작동하려면 컨트롤러가 필요합니다.

상태	컨트롤러 유형	재시작 정책
종료할 것으로 예상되는 Pod(예: 일괄 계산)	Job	**OnFailure** 또는 **Never**
종료되지 않을 것으로 예상되는 Pod(예: 웹 서버)	복제 컨트롤러	**Always**
머신당 하나씩 실행해야 하는 Pod	DaemonSet	Any

Pod의 컨테이너가 실패하고 재시작 정책이 **OnFailure** 로 설정되면 Pod가 노드에 남아 있고 컨테이너가 다시 시작됩니다. 컨테이너를 재시작하지 않으려면 재시작 정책 **Never** 를 사용합니다.

전체 Pod가 실패하면 OpenShift Container Platform에서 새 Pod를 시작합니다. 개발자는 애플리케이션이 새 포드에서 다시 시작될 수 있는 가능성을 해결해야 합니다. 특히 애플리케이션은 이전 실행으로 발생한 임시 파일, 잠금, 불완전한 출력 등을 처리해야 합니다.



참고

Kubernetes 아키텍처에서는 클라우드 공급자의 끝점이 안정적인 것으로 예상합니다. 클라우드 공급자가 중단되면 kubelet에서 OpenShift Container Platform이 재시작되지 않습니다.

기본 클라우드 공급자 끝점이 안정적이지 않은 경우 클라우드 공급자 통합을 사용하여 클러스터를 설치하지 마십시오. 클라우드가 아닌 환경에서처럼 클러스터를 설치합니다. 설치된 클러스터에서 클라우드 공급자 통합을 설정하거나 해제하는 것은 권장되지 않습니다.

OpenShift Container Platform에서 실패한 컨테이너에 재시작 정책을 사용하는 방법에 대한 자세한 내용은 Kubernetes 설명서의 예제 상태를 참조하십시오.

3.3.2. Init 컨테이너

init 컨테이너 는 Pod 앱 컨테이너를 시작하기 전에 시작되는 Pod의 컨테이너입니다. Init 컨테이너는 나머지 컨테이너가 시작되기 전에 볼륨을 공유하고, 네트워크 작업을 수행하고, 계산을 수행할 수 있습니다. Init 컨테이너는 일부 사전 조건이 충족될 때까지 애플리케이션 컨테이너의 시작을 차단하거나 지연할 수도 있습니다.

Pod가 시작되면 네트워크와 볼륨을 초기화한 후 init 컨테이너가 순서대로 시작됩니다. 각 init 컨테이너는 다음이 호출되기 전에 성공적으로 종료해야 합니다. init 컨테이너가 시작되지 않거나(런타임) 실패로 종료되면 Pod 재시작 정책에 따라 다시 시작됩니다.

Pod는 모든 init 컨테이너가 성공할 때까지 준비할 수 없습니다.

일부 init 컨테이너 사용 예는 Kubernetes 설명서를 참조하십시오.

다음 예제에서는 두 개의 init 컨테이너가 있는 간단한 포드를 간략하게 설명합니다. 첫 번째 init 컨테이너는 **myservice** 를 대기하고 두 번째 컨테이너는 **mydb** 를 기다립니다. 두 컨테이너가 모두 성공하면 Pod가 시작됩니다.

샘플 Init Container Pod 오브젝트 정의(YAML)

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice 1
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb 2
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

1 **myservice** 컨테이너를 지정합니다.

2 **mydb** 컨테이너를 지정합니다.

각 init 컨테이너에는 **readinessProbe** 를 제외한 모든 app 컨테이너의 필드가 있습니다. Pod가 계속 시작되려면 Init 컨테이너가 종료되어야 하며 완료 이외의 준비는 정의할 수 없습니다.

Init 컨테이너는 컨테이너에 **activeDeadlineSeconds** 및 **livenessProbe** 를 포함하여 init 컨테이너가 영구적으로 실패하지 않도록 할 수 있습니다. 활성 데드라인에는 init 컨테이너가 포함됩니다.

3.3.3. 서비스

Kubernetes 서비스는 내부 로드 밸런서 역할을 합니다. 수신 연결을 프록시하기 위해 복제된 포드 집합을 식별합니다. 서비스에 의존하는 모든 항목이 일관된 주소에서 참조할 수 있도록 백업 포드를 서비스에 임의로 추가하거나 제거할 수 있습니다. 기본 서비스 clusterIP 주소는 OpenShift Container Platform 내부 네트워크에서 제공되며 포드가 서로 액세스할 수 있도록 허용하는 데 사용됩니다.

서비스에 대한 외부 액세스를 허용하려면 클러스터 외부에 있는 추가 **externalIP 및 ingressIP** 주소를 서비스에 할당할 수 있습니다. 이러한 **externalIP** 주소는 서비스에 대한 고가용성 액세스를 제공하는 가상 IP 주소일 수도 있습니다.

서비스에 액세스할 때 적절한 지원 포드로 프록시하는 IP 주소 및 포트 쌍이 할당됩니다. 서비스는 레이블 선택기를 사용하여 특정 포트에서 특정 네트워크 서비스를 제공하는 실행 중인 모든 컨테이너를 찾습니다.

포드와 마찬가지로 서비스는 REST 개체입니다. 다음 예는 위에서 정의한 Pod에 대한 서비스 정의를 보여줍니다.

서비스 오브젝트 정의(YAML)

-

```
apiVersion: v1
kind: Service
metadata:
  name: docker-registry      (1)
spec:
  selector:                  (2)
    docker-registry: default
  clusterIP: 172.30.136.123  (3)
  ports:
  - nodePort: 0
    port: 5000               (4)
    protocol: TCP
    targetPort: 5000         (5)
```

(1) 서비스 이름 **docker-registry** 는 동일한 네임스페이스의 다른 포드에 삽입된 서비스 IP를 사용하여 환경 변수를 구성하는 데도 사용됩니다. 최대 이름 길이는 63자입니다.

(2) 레이블 선택기는 **docker-registry=default** 레이블이 백업 포드로 연결된 모든 포드를 식별합니다.

(3) 내부 IP 풀에서 생성 시 자동으로 할당되는 서비스의 가상 IP IP입니다.

(4) 서비스가 수신 대기하는 포트입니다.

(5) 서비스가 연결을 전달하는 백업 포드의 포트입니다.

Kubernetes 설명서 에는 서비스에 대한 자세한 정보가 있습니다.

3.3.3.1. 서비스 externalIPs

클러스터의 내부 IP 주소 외에도 사용자는 클러스터 외부에 있는 IP 주소를 구성할 수 있습니다. 관리자는 트래픽이 이 IP를 사용하여 노드에 도착하는지 확인합니다.

master-config.yaml 파일에 구성된 **externalIPNetworkCIDRs** 범위에서 클러스터 관리자가 externalIP를 선택해야 합니다. ***master-config.yaml*** 이 변경되면 마스터 서비스를 다시 시작해야 합니다.

Sample externalIPNetworkCIDR /etc/origin/master/master-config.yaml

```
networkConfig:
  externalIPNetworkCIDRs:
  - 192.0.1.0.0/24
```

서비스 externalIPs 정의(JSON)

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "my-service"
    },
    "spec": {
        "selector": {
            "app": "MyApp"
        },
```

```
        "ports": [
            {
                "name": "http",
                "protocol": "TCP",
                "port": 80,
                "targetPort": 9376
            }
        ],
        "externalIPs" : [
            "192.0.1.1"          (1)
        ]
    }
}
```

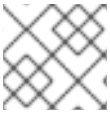
(1) **포트를** 노출하는 외부 IP 주소 목록입니다. 이 목록은 내부 IP 주소 목록에 추가되어 있습니다.

3.3.3.2. 서비스 ingressIPs

클라우드 이외의 클러스터에서 externalIP 주소는 주소 풀에서 자동으로 할당할 수 있습니다. 이렇게 하면 관리자가 수동으로 할당할 필요가 없습니다.

이 풀은 */etc/origin/master/master-config.yaml* 파일에 구성됩니다. 이 파일을 변경한 후 마스터 서비스를 다시 시작합니다.

ingressIPNetworkCIDR 은 기본적으로 **172.29.0.0/16** 으로 설정됩니다. 클러스터 환경에서 이 프라이빗 범위를 아직 사용하지 않는 경우 기본 범위를 사용하거나 사용자 지정 범위를 설정합니다.



참고

고가용성 을 사용하는 경우 이 범위는 256 주소 미만이어야 합니다.

Sample ingressIPNetworkCIDR /etc/origin/master/master-config.yaml

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

3.3.3.3. Service NodePort

서비스 **type=NodePort** 를 설정하면 플래그 구성 범위에서 포트가 할당됩니다(기본값: 30000~32767), 각 노드는 해당 포트(모든 노드에서 동일한 포트 번호)를 서비스에 프록시합니다.

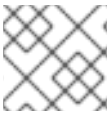
선택한 포트는 **spec.ports[*].nodePort** 의 서비스 구성에 보고됩니다.

사용자 지정 포트를 지정하려면 nodePort 필드에 포트 번호를 배치하면 됩니다. 사용자 지정 포트 번호는 nodePorts에 대해 구성된 범위에 있어야 합니다. '**master-config.yaml**'이 변경되면 마스터 서비스를 다시 시작해야 합니다.

Sample servicesNodePortRange /etc/origin/master/master-config.yaml

```
kubernetesMasterConfig:
  servicesNodePortRange: ""
```

서비스는 **<NodeIP>:spec.ports[].nodePort 및 spec.clusterIp:spec.ports[].port로 표시됩니다.**



참고

nodePort 설정은 권한 있는 작업입니다.

3.3.3.4. 서비스 프록시 모드

OpenShift Container Platform에는 서비스 라우팅 인프라에 대한 두 가지 다른 구현이 있습니다. 기본 구현은 전적으로 **iptables**기반이며 probabilistic **iptables** 재작성 규칙을 사용하여 수신되는 서비스 연결을 엔드포인트 포드에 배포합니다. 이전 구현에서는 사용자 공간 프로세스를 사용하여 들어오는 연결을 수락한 다음 클라이언트와 엔드포인트 포드 중 하나 간 트래픽을 프록시합니다.

iptables기반 구현은 훨씬 더 효율적이지만 모든 엔드포인트가 항상 연결을 허용할 수 있어야 합니다. 사용자 공간 구현이 느리지만 작동되는 엔드포인트를 찾을 때까지 여러 끝점을 차례로 시도할 수 있습니다. 준비 상태 검사 (또는 일반적으로 신뢰할 수 있는 노드 및 Pod)가 있으면 **iptables**기반 서비스 프록시가 최적의 선택입니다. 그렇지 않으면 노드 구성 파일을 편집하여 클러스터를 설치할 때 또는 클러스터를 배포한 후 사용자 공간 기반 프록시를 활성화할 수 있습니다.

3.3.3.5. 헤드리스 서비스

애플리케이션에 부하 분산 또는 단일 서비스 IP 주소가 필요하지 않은 경우 헤드리스 서비스를 생성할 수 있습니다. 헤드리스 서비스를 생성하면 부하 분산 또는 프록시가 수행되지 않으며 이 서비스에 대한 클러스터 IP가 할당되지 않습니다. 이러한 서비스의 경우 서비스에 선택기가 정의되어 있는지 여부에 따라 DNS가 자동으로 구성됩니다.

선택기가 있는 서비스: 선택기를 정의하는 헤드리스 서비스의 경우 엔드포인트 컨트롤러는 API에 **엔드포인트 레코드** 를 생성하고 DNS 구성을 수정하여 서비스를 지원하는 포드를 직접 가리키는 **A** 레코드(주소)를 반환합니다.

선택기 없는 서비스: 선택기를 정의하지 않는 헤드리스 서비스의 경우 엔드포인트 컨트롤러에서 엔드포인트 레코드 **를** 생성하지 않습니다. 그러나 DNS 시스템은 다음 레코드를 찾아 구성합니다.

- **ExternalName** 유형 서비스의 경우 **CNAME** 레코드.
- 다른 모든 서비스 유형의 경우 이름을 서비스와 공유하는 모든 엔드포인트에 대한 **A** 레코드입니다.

3.3.3.5.1. 헤드리스 서비스 생성

헤드리스 서비스 생성은 표준 서비스 생성과 비슷하지만 **ClusterIP** 주소를 선언하지 않습니다. 헤드리스 서비스를 생성하려면 **clusterIP를 추가합니다. none** 서비스 YAML 정의에 대한 매개변수 값입니다.

예를 들어 동일한 클러스터 또는 서비스의 일부가 되고자 하는 Pod 그룹의 경우입니다.

Pod 목록

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

출력 예

```
NAME             READY  STATUS    RESTARTS   AGE    IP          NODE
frontend-1-287hw  1/1    Running   0          7m     172.17.0.3  node_1
frontend-1-68km5  1/1    Running   0          7m     172.17.0.6  node_1
```

헤드리스 서비스를 다음과 같이 정의할 수 있습니다.

헤드리스 서비스 정의

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ruby-helloworld-sample
    template: application-template-stibuild
  name: frontend-headless (1)
spec:
  clusterIP: None (2)
  ports:
  - name: web
    port: 5432
    protocol: TCP
    targetPort: 8080
  selector:
    name: frontend (3)
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

(1) 헤드리스 서비스의 이름입니다.

(2) **clusterIP** 변수를 **None** 으로 설정하면 헤드리스 서비스를 선언합니다.

(3) **frontend** 레이블이 있는 모든 포드를 선택합니다.

또한 헤드리스 서비스에는 자체 IP 주소가 없습니다.

```
$ oc get svc
```

출력 예

```
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
frontend           ClusterIP   172.30.232.77   <none>        5432/TCP   12m
frontend-headless  ClusterIP   None            <none>        5432/TCP   10m
```

3.3.3.5.2. 헤드리스 서비스를 사용하여 끝점 검색

헤드리스 서비스를 사용하면 포드의 IP 주소를 직접 검색할 수 있다는 이점이 있습니다. 표준 서비스는 로드 밸런서 또는 프록시 역할을 하며 서비스 이름을 사용하여 워크로드 오브젝트에 대한 액세스 권한을 부여합니다. 헤드리스 서비스를 사용하면 서비스 이름이 서비스에서 그룹화한 포드의 IP 주소 집합으로 확인됩니다.

표준 서비스의 DNS **A** 레코드를 조회하면 서비스의 부하 분산된 IP가 표시됩니다.

```
$ dig frontend.test A +search +short
```

출력 예

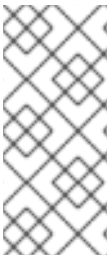
```
172.30.232.77
```

하지만 헤드리스 서비스의 경우 개별 포드의 IP 목록을 가져옵니다.

```
$ dig frontend-headless.test A +search +short
```

출력 예

```
172.17.0.3
172.17.0.6
```



참고

StatefulSet 및 초기화 및 종료 중에 Pod의 DNS를 확인해야 하는 관련 사용 사례와 함께 헤드리스 서비스를 사용하려면 **publishNotReadyAddresses** 를 **true** 로 설정합니다(기본값은 **false**임). **publishNotReadyAddresses** 를 **true** 로 설정하면 DNS 구현에서 서비스와 연결된 엔드포인트에 대한 **notReadyAddresses** 의 하위 집합을 게시해야 함을 나타냅니다.

3.3.4. 라벨

레이블은 API 오브젝트를 구성, 그룹화 또는 선택하는 데 사용됩니다. 예를 들어 Pod 는 레이블이 있는 "태그"에 해당된 다음 서비스에서 라벨 선택기를 사용하여 프록시할 포드를 식별합니다. 이렇게 하면 잠재적으로 다른 컨테이너가 있는 Pod를 관련 엔터티로 처리해도 서비스가 포드 그룹을 참조할 수 있습니다.

대부분의 오브젝트는 메타데이터에 레이블을 포함할 수 있습니다. 따라서 레이블을 사용하여 임의로 관련된 오브젝트를 그룹화할 수 있습니다. 예를 들어 특정 애플리케이션의 모든 포드,서비스,복제 컨트롤러 및 배포 구성을 그룹화할 수 있습니다.

레이블은 다음 예와 같이 간단한 키/값 쌍입니다.

```
labels:
  key1: value1
  key2: value2
```

고려 사항:

- 레이블이 **role=webserver** 인 **nginx** 컨테이너로 구성된 포드입니다.
- 동일한 레이블이 **role=webserver** 인 **Apache httpd** 컨테이너로 구성된 포드입니다.

role=webserver 레이블이 있는 포드를 사용하도록 정의된 서비스 또는 복제 컨트롤러는 이러한 포드를 모두 동일한 그룹의 일부로 취급합니다.

Kubernetes 설명서 에는 레이블에 대한 자세한 정보가 있습니다.

3.3.5. 엔드포인트

서비스를 지원하는 서버를 엔드포인트라고 하며, 서비스와 동일한 이름으로 **Endpoints** 유형의 오브젝트에 의해 지정됩니다. Pod에서 서비스를 지원하는 경우 일반적으로 해당 Pod는 서비스 사양의 라벨 선택기에 의해 지정되며 OpenShift Container Platform은 해당 Pod를 가리키는 Endpoints 오브젝트를 자동으

로 생성합니다.

경우에 따라 서비스를 생성하지만 OpenShift Container Platform 클러스터의 Pod가 아닌 외부 호스트에서 지원하도록 할 수 있습니다. 이 경우 서비스에서 **selector 필드를 종료하고** 끝점 오브젝트를 수동으로 생성할 수 있습니다.

OpenShift Container Platform에서는 대부분의 사용자가 Pod 및 서비스 IP용으로 예약된 네트워크 블록에서 IP 주소를 가리키는 엔드포인트 오브젝트를 수동으로 생성하지 않습니다. **엔드포인트/제한에서 리소스를 생성할 수 있는 권한이 있는 클러스터 관리자 또는 기타 사용자만 이러한 엔드포인트** 오브젝트를 생성할 수 있습니다.

3.4. 프로젝트 및 사용자

3.4.1. 사용자

OpenShift Container Platform과의 상호 작용은 사용자와 연결되어 있습니다. OpenShift Container Platform 사용자 오브젝트는 시스템에서 역할을 추가하여 권한을 부여받을 수 있는 행위자를 나타냅니다 .

다음과 같이 여러 유형의 사용자가 존재할 수 있습니다.

Regular users	대부분의 대화형 OpenShift Container Platform 사용자를 나타냅니다. 일반 사용자는 처음 로그인할 때 시스템에서 자동으로 생성되거나 API를 통해 생성할 수 있습니다. 일반 사용자는 **User** 오브젝트로 표시됩니다. 예:**joe alice**
System users	대부분의 시스템 사용자는 주로 인프라와 API 간의 안전한 상호 작용을 목적으로 인프라가 정의될 때 자동으로 생성됩니다. 여기에는 클러스터 관리자(전체 액세스 권한 보유), 노드별 사용자, 라우터 및 레지스트리용 사용자를 비롯하여 기타 다양한 사용자가 포함됩니다. 마지막으로 인증되지 않은 요청에 기본적으로 사용되는 **anonymous** 시스템 사용자가 있습니다. 예: **system:admin system:node:node1.example.com**
Service accounts	프로젝트와 관련된 특별한 시스템 사용자입니다. 일부는 프로젝트가 처음 생성될 때 자동으로 생성되는 반면 프로젝트 관리자는 각 프로젝트 내용에 대한 액세스를 정의하기 위해 추가로 생성할 수 있습니다. 서비스 계정은 **ServiceAccount** 오브젝트로 표시됩니다. 예: **system:serviceaccount:default:deployer system:serviceaccount:foo:builder**

모든 사용자가 OpenShift Container Platform에 액세스하려면 어떠한 방식으로든 인증 해야 합니다. 인증이 없거나 인증이 유효하지 않은 API 요청은 **anonymous** 시스템 사용자가 요청한 것으로 인증됩니다. 인증되고 나면 정책에 따라 사용자가 수행할 수 있는 작업이 결정됩니다.

3.4.2. 네임스페이스

Kubernetes 네임스페이스는 클러스터의 리소스 범위를 지정하는 메커니즘을 제공합니다. OpenShift Container Platform에서 프로젝트는 추가 주석이 있는 Kubernetes 네임스페이스입니다.

네임스페이스는 다음에 대한 고유 범위를 제공합니다.

- 기본 이름 지정 충돌을 피하기 위해 이름이 지정된 리소스
- 신뢰할 수 있는 사용자에게 위임된 관리 권한
- 커뮤니티 리소스 사용을 제한하는 기능

시스템에 있는 대부분의 오브젝트는 네임스페이스에 따라 범위가 지정되지만, 노드 및 사용자를 비롯한 일부는 여기에 해당하지 않으며 네임스페이스가 없습니다.

쿠버네티스 설명서에 네임스페이스에 대한 자세한 정보가 있습니다.

3.4.3. 프로젝트

프로젝트는 추가 주석이 있는 Kubernetes 네임스페이스이며 일반 사용자의 리소스에 대한 액세스를 관리하는 핵심 수단입니다. 사용자 커뮤니티는 프로젝트를 통해 다른 커뮤니티와 별도로 콘텐츠를 구성하고 관리할 수 있습니다. 사용자는 관리자로부터 프로젝트에 대한 액세스 권한을 부여받아야 합니다. 프로젝트를 생성하도록 허용된 경우 자신의 프로젝트에 액세스할 수 있는 권한이 자동으로 제공됩니다.

프로젝트에는 별도의 **이름**, **displayName** 및 **설명이** 있을 수 있습니다.

- 필수 **name** 은 프로젝트의 고유 식별자이며 CLI 툴 또는 API를 사용할 때 가장 잘 표시됩니다. 최대 이름 길이는 63자입니다.
- 선택적 **displayName** 은 프로젝트가 웹 콘솔에 표시되는 방식입니다(기본값: **name**).
- 선택적 **설명은** 프로젝트에 대한 자세한 설명이 될 수 있으며 웹 콘솔에도 표시됩니다.

각 프로젝트의 범위는 다음과 같습니다.

Objects	Pod, 서비스, 복제 컨트롤러 등입니다.
Policies	사용자는 오브젝트에서 이 규칙에 대해 작업을 수행할 수 있거나 수행할 수 없습니다.
Constraints	제한할 수 있는 각 종류의 오브젝트에 대한 할당량입니다.
Service accounts	서비스 계정은 프로젝트의 오브젝트에 지정된 액세스 권한으로 자동으로 작동합니다.

클러스터 관리자는 프로젝트를 생성하고 프로젝트에 대한 관리 권한을 사용자 커뮤니티의 모든 멤버에게 위임할 수 있습니다. 클러스터 관리자는 개발자가 자신의 프로젝트를 만들 수 있도록 허용할 수도 있습니다.

개발자와 관리자는 CLI 또는 웹 콘솔을 사용하여 프로젝트와 상호 작용할 수 있습니다 .

3.4.3.1. 설치 시 제공되는 프로젝트

OpenShift Container Platform에는 기본적으로 많은 프로젝트가 제공되며 **openshift-** 로 시작하는 프로젝트가 사용자에게 가장 중요합니다. 이러한 프로젝트는 Pod 및 기타 인프라 구성 요소로 실행되는 마스터 구성 요소를 호스팅합니다. 중요한 Pod 주석이 있는 네임스페이스에 생성된 Pod는 중요한 Pod로 간주되며 kubelet의 승인이 보장되었습니다. 이러한 네임스페이스에서 마스터 구성 요소용으로 생성된 Pod는 이미 중요로 표시되어 있습니다.

3.5. 빌드 및 이미지 스트림

3.5.1. 빌드

*빌드*는 입력 매개 변수를 결과 오브젝트로 변환하는 프로세스입니다. 대부분의 경우 프로세스는 입력 매개변수 또는 소스 코드를 실행 가능한 이미지로 변환하는 데 사용됩니다. **BuildConfig** 오브젝트는 전체 빌드 프로세스에 대한 정의입니다.

OpenShift Container Platform은 빌드 이미지에서 Docker 형식 컨테이너를 생성하고 컨테이너 이미지 레지스트리로 내보내 Kubernetes를 활용합니다.

빌드 오브젝트는 공통 특성을 공유합니다. 빌드에 대한 입력, 빌드 프로세스를 완료하고, 빌드 프로세스를 로깅하고, 빌드에 성공한 빌드의 리소스를 게시하고, 빌드의 최종 상태를 게시해야 합니다. 빌드에서는 CPU 사용량, 메모리 사용량, 빌드 또는 Pod 실행 시간과 같은 리소스 제한 사항을 활용합니다.

OpenShift Container Platform 빌드 시스템은 빌드 API에 지정된 선택 가능한 유형을 기반으로 하는 *빌드 전략에* 대한 확장 가능한 지원을 제공합니다. 다음은 세 가지 주요 빌드 전략입니다.

- Docker 빌드
- S2I(Source-to-Image) 빌드
- 사용자 정의 빌드

기본적으로 Docker 빌드 및 S2I 빌드가 지원됩니다.

빌드의 결과 오브젝트는 빌드를 생성하는 데 사용된 빌더에 따라 다릅니다. Docker 및 S2I 빌드의 경우 결과 오브젝트는 실행 가능한 이미지입니다. 사용자 정의 빌드의 경우 결과 오브젝트는 빌더 이미지 작성자가 지정한 모든 항목입니다.

또한 Pipeline 빌드 전략을 사용하여 정교한 워크플로를 구현할 수 있습니다.

- 연속 통합
- 연속 배포

빌드 명령 목록은 개발자 가이드를 참조하십시오.

OpenShift Container Platform이 빌드를 위해 Docker를 활용하는 방법에 대한 자세한 내용은 업스트림 문서를 참조하십시오.

3.5.1.1. Docker 빌드

Docker 빌드 전략에서는 Docker 빌드 명령을 호출하므로 실행 가능한 이미지를 생성하기 위해 ***Dockerfile*** 및 모든 필수 아티팩트가 있는 리포지토리가 필요합니다.

3.5.1.2. S2I(Source-to-Image) 빌드

S 2I(Source-to-Image) 는 재현 가능한 Docker 형식의 컨테이너 이미지를 빌드하는 툴입니다. 컨테이너 이미지에 애플리케이션 소스를 삽입하고 새 이미지를 어셈블하여 실행할 수 있는 이미지를 생성합니다. 새 이미지는 기본 이미지(빌더)와 빌드된 소스를 통합하며 **docker run** 명령과 함께 사용할 준비가 되었습니다. S2I는 이전에 다운로드한 종속성, 이전에 빌드한 아티팩트 등을 다시 사용하는 증분 빌드를 지원합니다.

S2I의 장점은 다음과 같습니다.

이미지 유연성	S2I 스크립트는 기존 에코시스템을 활용하여 거의 모든 Docker 형식의 컨테이너 이미지에 애플리케이션 코드를 삽입하도록 작성할 수 있습니다. 현재 S2I는 **tar** 을 사용하여 애플리케이션 소스를 삽입하므로 이미지가 tarred 콘텐츠를 처리할 수 있어야 합니다.

속도	S2I를 사용하면 각 단계에서 새 계층을 생성하지 않고도 assemble 프로세스에서 많은 수의 복잡한 작업을 수행할 수 있으므로 프로세스가 빨라집니다. 또한 S2I 스크립트는 빌드가 실행될 때마다 다운로드하거나 빌드할 필요 없이 이전 버전의 애플리케이션 이미지에 저장된 아티팩트를 다시 사용하도록 작성할 수 있습니다.
패치 가능성	보안 문제로 인해 기본 이미지에 패치가 필요한 경우 S2I를 사용하면 애플리케이션을 일관되게 다시 빌드할 수 있습니다.
운영 효율성	***Dockerfile*** 에서 허용하므로 임의의 작업을 허용하는 대신 빌드 작업을 제한하면 PaaS 운영자가 실수로 빌드 시스템의 오용을 방지할 수 있습니다.
운영 보안	임의의 ***Dockerfile*** 을 구축하면 호스트 시스템을 루트 권한 에스컬레이션에 노출합니다. 전체 Docker 빌드 프로세스가 Docker 권한이 있는 사용자로 실행되므로 악의적인 사용자가 악용할 수 있습니다. S2I는 root 사용자로 수행되는 작업을 제한하며 스크립트를 루트가 아닌 사용자로 실행할 수 있습니다.
사용자 효율성	S2I를 사용하면 개발자가 임의의 **yum install** 유형 작업을 수행하지 못하므로 애플리케이션 빌드 중에 개발 반복 속도가 느려질 수 있습니다.
에코시스템	S2I는 애플리케이션의 모범 사례를 활용할 수 있는 이미지의 공유 에코시스템을 권장합니다.
재현 가능성	생성된 이미지에는 특정 버전의 빌드 툴 및 종속성을 포함하여 모든 입력이 포함될 수 있습니다. 이렇게 하면 이미지를 정확하게 재현할 수 있습니다.

3.5.1.3. 사용자 정의 빌드

사용자 정의 빌드 전략을 사용하면 개발자가 전체 빌드 프로세스를 담당하는 특정 빌더 이미지를 정의할 수 있습니다. 자체 빌더 이미지를 사용하면 빌드 프로세스를 사용자 정의할 수 있습니다.

사용자 정의 빌더 이미지는 빌드 프로세스 논리에 포함된 일반 Docker 형식 컨테이너 이미지입니다(예: RPM 또는 기본 이미지 빌드). **openshift/origin-custom-docker-builder** 이미지는 사용자 지정 빌더 이미지의 예제 구현으로 Docker Hub 레지스트리에서 사용할 수 있습니다.

3.5.1.4. 파이프 라인 빌드

파이프라인 빌드 전략을 사용하면 개발자가 *Jenkins 파이프라인* 플러그인에 의해 실행할 Jenkins 파이프라인을 정의할 수 있습니다. 다른 빌드 유형과 동일한 방식으로 OpenShift Container Platform에서 빌드를 시작, 모니터링, 관리할 수 있습니다.

파이프라인 워크플로는 Jenkinsfile에 정의되거나 빌드 구성에 직접 포함되거나 Git 리포지토리에 제공되고 빌드 구성에서 참조됩니다.

프로젝트에서 Pipeline 전략을 사용하여 빌드 구성을 정의하면 OpenShift Container Platform에서 Jenkins 서버를 인스턴스화하여 파이프라인을 실행합니다. 프로젝트의 이후 파이프라인 빌드 구성은 이 Jenkins 서버를 공유합니다.

Jenkins 서버가 배포되는 방법과 자동 프로비저닝 동작을 구성하거나 비활성화하는 방법에 대한 자세한 내용은 파이프라인 실행 구성을 참조하십시오.

참고

모든 Pipeline 빌드 구성이 삭제되어도 Jenkins 서버가 자동으로 제거되지 않습니다. 사용자가 수동으로 삭제해야 합니다.

Jenkins 파이프라인에 대한 자세한 내용은 Jenkins 설명서 를 참조하십시오.

3.5.2. 이미지 스트림

이미지 스트림 및 관련 태그는 OpenShift Container Platform 내에서 컨테이너 이미지를 참조하기 위한 추상화를 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 스트림에서 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.

그러나 배포 또는 빌드에서 사용하는 이미지 스트림 태그가 업데이트되지 않으면 컨테이너 이미지 레지스트리의 컨테이너 이미지가 업데이트되어도 빌드 또는 배포에서 이전(알려진 정상) 이미지를 계속 사용합니다.

소스 이미지를 저장할 수 있는 위치는 다음과 같습니다.

- OpenShift Container Platform의 통합 레지스트리
- 외부 레지스트리(예: **registry.redhat.io** 또는 **hub.docker.com**)
- OpenShift Container Platform 클러스터의 기타 이미지 스트림

이미지 스트림 태그(예: 빌드 또는 배포 구성)를 참조하는 오브젝트를 정의하는 경우 Docker 리포지토리가 아닌 이미지 스트림 태그를 가리킵니다. 애플리케이션을 빌드하거나 배포할 때 OpenShift Container Platform은 이미지 스트림 태그를 사용하여 Docker 리포지토리를 쿼리하여 관련 이미지 ID를 찾고 바로 그 이미지를 사용합니다.

이미지 스트림 메타데이터는 다른 클러스터 정보와 함께 etcd 인스턴스에 저장됩니다.

다음 이미지 스트림에는 두 개의 태그가 포함되어 있습니다. **34** Python v3.4 이미지와 **35** 를 가리켜 Python v3.5 이미지를 가리킵니다.

```
$ oc describe is python
```

출력 예

```
Name:   python
Namespace:  imagestream
Created:  25 hours ago
Labels:   app=python
Annotations:  openshift.io/generated-by=OpenShiftWebConsole
  openshift.io/image.dockerRepositoryCheck=2017-10-03T19:48:00Z
Docker Pull Spec: docker-registry.default.svc:5000/imagestream/python
```

```
Image Lookup:  local=false
Unique Images:  2
Tags:   2

34
  tagged from centos/python-34-centos7

  * centos/python-34-
centos7@sha256:28178e2352d31f240de1af1370be855db33ae9782de737bb005247d8791a54d0
      14 seconds ago

35
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:2efb79ca3ac9c9145a63675fb0c09220ab3b8d4005d35e0644417ee552548b10
      7 seconds ago
```

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. 또한 OpenShift Container Platform에는 다른 리소스(예: Kubernetes 오브젝트)에 대한 일반 트리거가 있습니다.
- 정기적인 다시 가져오기를 위해 태그를 표시할 수 있습니다 . 소스 이미지가 변경되면 해당 변경 사항이 이미지 스트림에 반영되며, 이 변경 사항은 빌드 또는 배포 구성에 따라 빌드 및/또는 배포 흐름을 트리거합니다.
- 세분화된 액세스 제어를 사용하여 이미지를 공유하고 팀 전체에 이미지를 빠르게 배포할 수 있습니다.
- 소스 이미지가 변경되면 이미지 스트림 태그는 계속 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.

조정된 이미지 스트림 세트는 OpenShift Image Streams 및 Templates 라이브러리 를 참조하십시오.

이미지 스트림을 사용할 때는 이미지 스트림 태그가 가리키는 사항과 태그 및 이미지를 변경하는 방법을 이해해야 합니다. 예를 들면 다음과 같습니다.

- 이미지 스트림 태그가 컨테이너 이미지 태그를 가리키는 경우 컨테이너 이미지 태그 업데이트 방법을 이해해야 합니다. 예를 들어 컨테이너 이미지 태그 **docker.io/ruby:2.5** 는 v2.5 ruby 이미지를 가리키지만 컨테이너 이미지 태그 **docker.io/ruby:latest** 는 주요 버전에서 변경됩니다. 따라서 이미지 스트림 태그가 가리키는 컨테이너 이미지 태그는 이미지 스트림 태그의 안정성을 나타낼 수 있습니다.
- 이미지 스트림 태그가 컨테이너 이미지 태그를 직접 가리키는 대신 다른 이미지 스트림 태그를 따르는 경우 나중에 다른 이미지 스트림 태그를 따르도록 이미지 스트림 태그를 업데이트할 수 있습니다. 이러한 변경으로 인해 호환되지 않는 버전 변경이 발생할 수 있습니다.

3.5.2.1. 중요한 용어

Docker 리포지토리

관련 컨테이너 이미지와 이를 식별하는 태그의 컬렉션입니다. 예를 들어 OpenShift Jenkins 이미지는 Docker 리포지토리에 있습니다.

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

컨테이너 레지스트리

Docker 리포지토리에서 이미지를 저장하고 서비스할 수 있는 콘텐츠 서버입니다. 예를 들면 다음과 같습니다.

```
registry.redhat.io
```

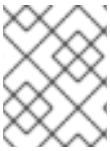
컨테이너 이미지

컨테이너로 실행할 수 있는 특정 콘텐츠 세트입니다. 일반적으로 Docker 리포지토리 내의 특정 태그와 연결됩니다.

컨테이너 이미지 태그

특정 이미지를 구분하는 리포지토리의 컨테이너 이미지에 적용되는 레이블입니다. 예를 들어, 여기에서 **3.6.0** 은 태그입니다.

```
docker.io/openshift/jenkins-2-centos7:3.6.0
```



참고

언제든지 새 컨테이너 이미지 콘텐츠를 가리키도록 컨테이너 이미지 태그를 업데이트할 수 있습니다.

컨테이너 이미지 ID

이미지를 가져오는 데 사용할 수 있는 SHA(Secure Hash Algorithm) 코드입니다. 예를 들면 다음과 같습니다.

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

참고

SHA 이미지 ID는 변경할 수 없습니다. 특정 SHA 식별자는 항상 정확히 동일한 컨테이너 이미지 콘텐츠를 참조합니다.

이미지 스트림

태그로 식별되는 Docker 형식 컨테이너 이미지의 포인터가 포함된 OpenShift Container Platform 오브젝트입니다. 이미지 스트림을 Docker 리포지토리에 해당하는 것으로 간주할 수 있습니다.

이미지 스트림 태그

이미지 스트림의 이미지에 대한 명명된 포인터. 이미지 스트림 태그는 컨테이너 이미지 태그와 유사합니다. 아래 이미지 스트림 태그를 참조하십시오.

이미지 스트림 이미지

태그된 특정 이미지 스트림에서 특정 컨테이너 이미지를 검색할 수 있는 이미지입니다. 이미지 스트림 이미지는 특정 이미지 SHA 식별자에 대한 일부 메타데이터를 함께 가져오는 API 리소스 오브젝트입니다. 아래 이미지 스트림 이미지를 참조하십시오.

이미지 스트림 트리거

이미지 스트림 태그가 변경되면 특정 작업을 수행하는 트리거입니다. 예를 들어 가져오기로 인해 태그 값이 변경되고 이어서 해당 태그를 수신하는 배포, 빌드 또는 기타 리소스에서 트리거가 실행될 수 있습니다. 아래 이미지 스트림 트리거 를 참조하십시오.

3.5.2.2. 이미지 스트림 구성

이미지 스트림 오브젝트 파일에는 다음 요소가 포함되어 있습니다.

참고

이미지 및 이미지 스트림 관리에 대한 자세한 내용은 개발자 가이드를 참조하십시오.

이미지 스트림 오브젝트 정의

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
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: 2017-09-29T13:40:11Z
      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/업데이트하기 위해 새 이미지를 푸시할 수 있는 Docker 리포지

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

이미지 스트림을 참조하는 샘플 빌드 구성은 구성의 **Strategy** 스탠자의 BuildConfig이란? 을 참조하십시오.

이미지 스트림을 참조하는 샘플 배포 구성은 구성의 **전략** 스탠자의 배포 구성 생성을 참조하십시오.

3.5.2.3. 이미지 스트림 이미지

*이미지 스트림 이미지*는 *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>
```

위의 이미지 스트림 오브젝트 예에서 이미지를 참조하려면 이미지 스트림 이미지가 다음과 같이 표시됩니다.

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

3.5.2.4. 이미지 스트림 태그

*이미지 스트림 태그*는 이미지 *스트림* 의 이미지에 대한 이름 지정된 포인터입니다. *istag*로 축약되는 경우가 많습니다. 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 있으며 이러한 방식으로 계속 배치됩니다. 이렇게 하면 태그가 다시 과거 이미지를 가리키도록 손쉽게 롤백할 수 있습니다.

다음 이미지 스트림 태그는 위의 이미지 스트림 오브젝트 예제에서 가져온 것입니다.

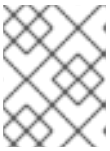
기록에 두 개의 이미지가 있는 이미지 스트림 태그

```
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: 2017-09-29T13:40:11Z
    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 태그*는 Python 3.5 같은 특정 버전의 이미지를 가리키는 버전 특정 태그입니다.
- *추적 태그*는 다른 이미지 스트림 태그를 따르는 참조 태그이며 symlink와 매우 유사하게 나중에 업데이트하여 후속 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 새 수준을 사용할 수 있게 되면 이미지에서 제공하는 프레임워크의 최신 수준으로 업데이트됩니다. **v3.10** 의 **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. 권장 태그 지정 규칙을 참조하십시오.

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

```
<image stream name>:<tag>
```

예를 들어 위의 이미지 스트림 오브젝트 예에서 **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** 이미지를 참조하려면 이미지 스트림 태그는 다음과 같습니다.

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

3.5.2.5. 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.

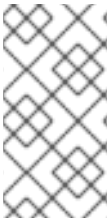
ImageChange 트리거를 사용하면 이미지 스트림 태그의 내용이 변경될 때마다(새 버전의 이미지를 푸시할 때) 새 복제 컨트롤러가 생성됩니다.

이미지 변경 트리거

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

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

위의 예에서 **origin-ruby-sample** 이미지 스트림의 **latest** 태그 값이 변경되고 새 이미지 값이 배포 구성의 helloworld 컨테이너에 지정된 현재 이미지와 다르면 **helloworld** 컨테이너의 새 이미지를 사용하여 새 복제 컨트롤러가 생성됩니다 .



참고

ImageChange 트리거가 배포 구성(Config **Change** 트리거 및 automatic **=false** 또는 **automatic=true**사용)에 정의되어 있고 **ImageChange** 트리거가 가리키는 **ImageStreamTag** 가 아직 존재하지 않는 경우 빌드에서 이미지를 가져오거나 **ImageStreamTag** 로 내보내는 즉시 초기 배포 프로세스가 자동으로 시작됩니다.

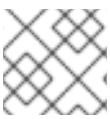
3.5.2.6. 이미지 스트림 매핑

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

참고

이미지 스트림 매핑 구성은 고급 기능입니다.

이 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.



참고

이미지 태그에 대한 자세한 내용은 개발자 가이드를 참조하십시오.

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

이미지 스트림 매핑 오브젝트 정의

```
apiVersion: 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
```

3.5.2.7. 이미지 스트림 작업

다음 섹션에서는 이미지 스트림 및 이미지 스트림 태그를 사용하는 방법에 대해 설명합니다. 이미지 스트림 사용에 대한 자세한 내용은 이미지 관리를 참조하십시오.

3.5.2.7.1. 이미지 스트림에 대한 정보 가져오기

이미지 스트림에 대한 일반 정보와 해당 이미지 스트림이 가리키는 모든 태그에 대한 자세한 정보를 얻으려면 다음 명령을 사용합니다.

```
$ 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 얻으려면 다음을 수행합니다.

```
$ 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.

3.5.2.7.2. 이미지 스트림에 태그 추가

기존 태그 중 하나를 가리키는 태그를 추가하려면 **oc tag** 명령을 사용할 수 있습니다.

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

예를 들면 다음과 같습니다.

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

출력 예

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

oc describe 명령을 사용하여 이미지 스트림에 외부 컨테이너 이미지를 가리키는 하나의 태그(3.5)와 첫 번째 태그를 기반으로 생성되었기 때문에 동일한 이미지를 가리키는 다른 태그(**최신**) 태그가 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
```

3.5.2.7.3. 외부 이미지의 태그 추가

내부 또는 외부 이미지를 가리키는 태그 추가와 같은 모든 태그 관련 작업에 **oc tag** 명령을 사용합니다.

```
$ oc tag <repositiory/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 이미지를 보호하는 경우 해당 레지스트리에 액세스하기 위한 자격 증명으로 시크릿을 생성해야 합니다. 자세한 내용은 프라이빗 레지스트리에서 이미지 가져오기를 참조하십시오.

3.5.2.7.4. 이미지 스트림 태그 업데이트

이미지 스트림의 다른 태그를 반영하도록 태그를 업데이트하려면 다음을 수행합니다.

```
$ 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.
```

3.5.2.7.5. 이미지 스트림에서 이미지 스트림 태그 제거

이미지 스트림에서 이전 태그를 제거하려면 다음을 수행합니다.

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

예를 들면 다음과 같습니다.

```
$ oc tag -d python:3.5
```

출력 예

```
Deleted tag default/python:3.5.
```

3.5.2.7.6. 태그 가져오기 주기 구성

외부 컨테이너 이미지 레지스트리로 작업하는 경우 이미지(예: 최신 보안 업데이트를 얻기 위해)를 정기적으로 다시 가져오려면 **--scheduled** 플래그를 사용합니다.

```
$ oc tag <repositiory/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.

정기적인 검사를 제거하려면 위의 명령을 다시 실행하되 **--scheduled** 플래그를 생략합니다. 이렇게 하면 동작이 기본값으로 재설정됩니다.

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

3.6. 배포

3.6.1. 복제 컨트롤러

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

복제 컨트롤러 구성은 다음과 같이 구성됩니다.

1. 원하는 복제본 수(런타임 시 조정할 수 있음).
2. 복제된 포드를 생성할 때 사용할 포드 정의입니다.
3. 관리형 Pod를 확인하는 선택기

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

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

복제 컨트롤러는 **ReplicationController** 라는 핵심 Kubernetes 오브젝트입니다.

다음은 **ReplicationController** 정의의 예입니다.

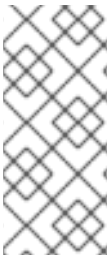
```
apiVersion: v1
kind: ReplicationController
metadata:
```

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

1. 실행할 Pod의 사본 수입니다.
2. 실행할 Pod의 라벨 선택기입니다.
3. 컨트롤러에서 생성하는 Pod용 템플릿입니다.
4. Pod의 라벨에는 라벨 선택기의 해당 항목이 포함되어야 합니다.
5. 매개변수를 확장한 후 최대 이름 길이는 63자입니다.

3.6.2. 복제 세트

복제 컨트롤러와 유사하게 복제본 세트는 지정된 수의 포드 복제본이 언제든지 실행되도록 합니다. 복제본 세트와 복제 컨트롤러의 차이점은 복제본 세트는 세트 기반 선택기 요구 사항을 지원하는 반면 복제 컨트롤러는 일치 기반 선택기 요구 사항만 지원한다는 점입니다.



참고

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

복제본 세트는 **ReplicaSet** 이라는 핵심 Kubernetes 오브젝트입니다.

다음은 **ReplicaSet** 정의의 예입니다.

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

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

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

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

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

3.6.3. Jobs

작업은 복제 컨트롤러와 비슷하며, 이는 지정된 이유로 포드를 생성하는 것입니다. 차이점은 복제 컨트롤러는 지속적으로 실행 중인 포드를 위해 설계된 반면 작업은 일회성 포드를 위한 것입니다. 작업은 완료를 추적하며 지정된 수의 완료에 도달하면 작업 자체가 완료됩니다.

다음 예제에서는 ✓~ 2000개의 위치를 계산하여 출력한 다음 다음과 같이 완료합니다.

```
apiVersion: extensions/v1
kind: Job
metadata:
  name: pi
spec:
  selector:
    matchLabels:
      app: pi
  template:
    metadata:
      name: pi
      labels:
        app: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

작업 사용 방법에 대한 자세한 내용은 Jobs (작업) 주제를 참조하십시오.

3.6.4. 배포 및 배포 구성

복제 컨트롤러를 기반으로 OpenShift Container Platform은 배포 개념을 사용하여 소프트웨어 개발 및 배포 라이프사이클에 대한 확장된 지원을 추가합니다. 가장 간단한 경우 배포는 새 복제 컨트롤러만 생성하고 포드를 시작할 수 있도록 합니다. 그러나 OpenShift Container Platform 배포에서는 이미지의 기존 배포에서 새 배포로 전환하고 복제 컨트롤러를 생성하기 전이나 후에 실행할 후크도 정의할 수 있습니다.

OpenShift Container Platform **DeploymentConfig** 오브젝트는 배포에 대한 다음 세부 정보를 정의합니다.

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

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

배포를 만들고 상호 작용하는 방법에 대한 자세한 지침은 배포를 참조하십시오.

다음은 일부 omissions 및 callouts가 포함된 예제 **DeploymentConfig** 정의입니다.

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

1. 복제 컨트롤러 템플릿이 변경될 때마다 **ConfigChange** 트리거를 통해 새 배포가 생성됩니다.
2. 명명된 이미지 스트림에서 새 버전의 백업 이미지를 사용할 수 있을 때마다 **ImageChange** 트리거를 통해 새 배포가 생성됩니다.

3 기본 **롤링** 전략을 사용하면 배포 간에 다운타임 없이 전환할 수 있습니다.

3.7. 템플릿

3.7.1. 개요

템플릿은 OpenShift Container Platform에서 생성할 오브젝트 목록을 생성하기 위해 매개 변수화 및 처리할 수 있는 오브젝트 세트를 설명합니다. 생성할 오브젝트에는 서비스, 빌드 구성 및 배포 구성과 같이 프로젝트 내에서 생성할 권한이 있는 모든 항목이 포함될 수 있습니다. 템플릿은 템플릿에 정의된 모든 오브젝트에 적용할 레이블 집합도 정의할 수 있습니다.

템플릿 생성 및 사용에 대한 자세한 내용은 템플릿 가이드를 참조하십시오.

4장. 추가 개념

4.1. 인증

4.1.1. 개요

그러면 인증 계층에서 OpenShift Container Platform API에 대한 요청과 관련된 사용자를 확인합니다. 그런 다음 권한 부여 계층에서는 요청하는 사용자에 대한 정보를 사용하여 요청을 허용해야 하는지 여부를 결정합니다.

관리자는 마스터 구성 파일을 사용하여 인증을 구성할 수 있습니다.

4.1.2. 사용자 및 그룹

OpenShift Container Platform에서 *사용자*는 OpenShift Container Platform API에 요청할 수 있는 엔티티입니다. 일반적으로 OpenShift Container Platform과 상호 작용하는 개발자 또는 관리자 계정을 의미합니다.

사용자는 하나 이상의 *그룹*에 할당될 수 있으며, 각 그룹은 특정 사용자 집합을 나타냅니다. 그룹은 사용자에게 개별적으로 부여하는 대신 권한 부여 정책을 관리하여 여러 사용자에게 한 번에 권한을 부여할 때 유용합니다(예: 프로젝트 내 오브젝트에 대한 액세스 허용).

명시적으로 정의된 그룹 외에도 OpenShift에서 자동으로 프로비저닝하는 시스템 그룹 또는 *가상 그룹이* 있습니다. 클러스터 바인딩을 볼 때 볼 수 있습니다.

가상 그룹의 기본 집합에서는 특히 다음 사항에 유의하십시오.

가상 그룹	설명
system:authenticated	인증된 모든 사용자와 자동으로 연결됩니다.
system:authenticated:oauth	OAuth 액세스 토큰을 사용하여 인증된 모든 사용자와 자동으로 연결됩니다.
system:unauthenticated	인증되지 않은 모든 사용자와 자동으로 연결됩니다.

4.1.3. API 인증

OpenShift Container Platform API에 대한 요청은 다음과 같은 방법으로 인증됩니다.

OAuth 액세스 토큰

- <master> **/oauth/authorize 및** ***<master> /oauth/*****token** 끝점을 사용하여 OpenShift Container Platform OAuth 서버에서 가져옵니다.
- **인증: 전달자...** 헤더.
- WebSocket 요청의 경우 **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** 형식의 WebSocket 하위 프로토콜 헤더로 전송됩니다.

X.509 클라이언트 인증서

- API 서버에 대한 HTTPS 연결이 필요합니다.
- 신뢰할 수 있는 인증 기관 번들과 대조하여 API 서버에서 확인합니다.
- API 서버는 인증서를 작성하고 컨트롤러에 분배하여 자체적으로 인증합니다.

유효하지 않은 액세스 토큰 또는 유효하지 않은 인증서가 있는 요청은 401 오류와 함께 인증 계층에서 거부됩니다.

액세스 토큰이나 인증서가 없는 경우 인증 계층은 **system:anonymous** 가상 사용자 및 **system:unauthenticated** 가상 그룹을 요청에 할당합니다. 그러면 권한 부여 계층에서 익명 사용자가 할 수 있는 요청(있는 경우)을 결정합니다.

4.1.3.1. 가장

OpenShift Container Platform API 요청에는 **Impersonate-User** 헤더가 포함될 수 있습니다. 이 헤더는 요청자가 지정된 사용자로부터 온 것처럼 요청을 처리하려고 함을 나타냅니다. 요청에 **--as=<user>** 플래그를 추가하여 사용자를 가장합니다.

사용자 A가 사용자 B를 가장할 수 있으려면 사용자 A가 인증됩니다. 그런 다음 권한 부여 확인이 발생하여 사용자 A가 User B라는 사용자를 가장할 수 있는지 확인합니다. 사용자 A가 서비스 계정을 가장하도록 요청하는 경우 **system:serviceaccount:namespace:name** 은 OpenShift Container Platform에서 User A가 **이름이** 인 **serviceaccount** 를 **네임스페이스** 에서 가장할 수 있는지 확인합니다. 검사에 실패하면 403(Forbidden) 오류 코드와 함께 요청이 실패합니다.

기본적으로 프로젝트 관리자 및 편집기는 해당 네임스페이스의 서비스 계정을 가장할 수 있습니다. **sudoers** 역할을 사용하면 **system:admin** 을 가장할 수 있으며, 이로 인해 클러스터 관리자 권한이 부여됩니다. **system:admin** 은 클러스터를 관리하는 사용자를 위해 보안이 아닌 오타에 대해 일부 보호 권한을 부여합니다. 예를 들어 **oc delete nodes --all** 을 실행하면 실패하지만 **oc delete nodes --all --as=system:admin** 을 실행하면 성공합니다. 다음 명령을 실행하여 권한을 사용자에게 부여할 수 있습니다.

```
$ oc create clusterrolebinding <any_valid_name> --clusterrole=sudoer --user=<username>
```

사용자를 대신하여 프로젝트 요청을 생성해야 하는 경우 명령에 **--as=<user> --as-group=<group1> --as-group=<group2>** 플래그를 포함합니다. **system:authenticated:oauth** 는 프로젝트 요청을 생성할 수 있는 유일한 부트스트랩 그룹이므로 다음 예와 같이 해당 그룹을 가장해야 합니다.

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

4.1.4. OAuth

OpenShift Container Platform 마스터에는 내장 OAuth 서버가 포함되어 있습니다. 사용자는 API 인증을 위해 OAuth 액세스 토큰을 가져옵니다.

사용자가 새 OAuth 토큰을 요청하면 OAuth 서버는 구성된 ID 공급자를 사용하여 요청한 사용자의 ID를 확인합니다.

그런 다음 해당 ID와 매핑되는 사용자를 결정하고 그 사용자를 위한 액세스 토큰을 만들어 제공합니다.

4.1.4.1. OAuth 클라이언트

OAuth 토큰을 요청할 때마다 토큰을 받고 사용할 OAuth 클라이언트를 지정해야 합니다. OpenShift Container Platform API를 시작하면 다음 OAuth 클라이언트가 자동으로 생성됩니다.

OAuth 클라이언트	사용법
openshift-web-console	웹 콘솔의 토큰을 요청합니다.
openshift-browser-client	대화형 로그인을 처리할 수 있는 사용자 에이전트를 사용하여 ***<master>*/oauth/token/request** 에서 토큰을 요청합니다.
openshift-challenging-client	**WWW-Authenticate** 챌린지를 처리할 수 있는 사용자 에이전트로 토큰을 요청합니다.

추가 클라이언트를 등록하려면 다음을 수행합니다.

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
 name: demo 1
secret: "..." 2
redirectURIs:
 - "http://www.example.com/" 3
grantMethod: prompt 4
')
```

1. OAuth 클라이언트의 **이름은** <master> **/oauth/authorize 및 *<master>* /*oauth*/token** 에 요청할 때 **client_id** 매개변수로 사용됩니다.
2. **secret** 은 ***<master>*/oauth/token** 에 요청할 때 **client_secret** 매개변수로 사용됩니다.
3. ***<master>* /*oauth/authorize* 및 < *master*>/oauth/token** 에 대한 요청에 지정된 **redirect_uri** 매개변수는 **redirectURIs** 의 URI 중 하나와 같아야 합니다.
4. **grantMethod**는 이 클라이언트에서 토큰을 요청할 때 사용자가 아직 액세스 권한을 부여받지 않은 경우 수행할 조치를 결정하는 데 사용됩니다. grant Options(제한 옵션)에 표시되는 동일한 값을 사용합니다.

4.1.4.2. OAuth 클라이언트로서의 서비스 계정

서비스 계정은 제한된 형태의 OAuth 클라이언트로 사용할 수 있습니다. 서비스 계정은 서비스 계정의 자체 네임스페이스 내에서 일부 기본 사용자 정보 및 역할 기반 권한에 액세스할 수 있는 부분만 요청할 수 있습니다.

- **user:info**
- **user:check-access**
- **role:<any_role>:<serviceaccount_namespace>**
- **role:<any_role>:<serviceaccount_namespace>:!**

서비스 계정을 OAuth 클라이언트로 사용하는 경우에는 다음과 같습니다.

- **client_id**는 **system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>**입니다.
- **client_secret**은 해당 서비스 계정의 API 토큰 중 하나일 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc sa get-token <serviceaccount_name>
```

- **WWW-Authenticate** 챌린지를 가져오려면 서비스 계정의 **serviceaccounts.openshift.io/oauth-want-challenges** 주석을 **true**로 설정합니다.
- **redirect_uri**는 서비스 계정의 주석과 일치해야 합니다. 서비스 계정의 URI를 OAuth 클라이언트에서 자세한 정보를 제공하므로 리디렉션합니다.

4.1.4.3. 서비스 계정의 URI를 OAuth 클라이언트로 리디렉션

주석 키에는 접두사 **serviceaccounts.openshift.io/oauth-redirecturi.** 또는 **serviceaccounts.openshift.io/oauth-redirectreference.**가 있어야 합니다. 예를 들면 다음과 같습니다.

```
serviceaccounts.openshift.io/oauth-redirecturi.<name>
```

가장 간단한 형식의 주석을 사용하여 유효한 리디렉션 URI를 직접 지정할 수 있습니다. 예를 들면 다음과 같습니다.

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "https://example.com"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

위 예에서 **first** 및 **second**라는 접미사는 두 개의 유효한 리디렉션 URI를 구분하는 데 사용됩니다.

복잡한 구성에서는 정적 리디렉션 URI로는 충분하지 않을 수 있습니다. 예를 들어 경로의 모든 인그레스가 유효한 것으로 간주될 수 있습니다. 이 경우 **serviceaccounts.openshift.io/oauth-redirectreference** 접두사를 통한 동적 리디렉션 URI가 작동합니다.

예를 들면 다음과 같습니다.

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

이 주석의 값은 직렬화된 JSON 데이터를 포함하므로 확장된 형식으로 쉽게 볼 수 있습니다.

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": "Route",
    "name": "jenkins"
  }
}
```

이제 **OAuthRedirectReference**를 통해 **jenkins**라는 경로를 참조할 수 있음을 확인할 수 있습니다. 따라서 해당 경로의 모든 인그레스가 이제 유효한 것으로 간주됩니다. **OAuthRedirectReference**의 전체 사양은 다음과 같습니다.

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": ..., 1
    "name": ..., 2
    "group": ... 3
  }
}
```

1 **kind**는 참조되는 오브젝트의 유형을 나타냅니다. 현재는 **route**만 지원됩니다.

2 **name**은 오브젝트의 이름을 나타냅니다. 오브젝트는 서비스 계정과 동일한 네임스페이스에 있어야 합니다.

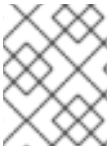
3 **group**은 오브젝트 그룹을 나타냅니다. 경로 그룹이 빈 문자열이므로 이 필드를 비워둡니다.

두 주석 접두사를 결합하여 참조 오브젝트에서 제공하는 데이터를 덮어쓸 수 있습니다. 예를 들면 다음과 같습니다.

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

first 접미사는 주석을 연결하는 데 사용됩니다. **jenkins** 경로에 *https://example.com* 의 인그레스가 있다고 가정하면 이제 *https://example.com/custompath* 은 유효한 것으로 간주되지만 *https://example.com* 은 유효한 것으로 간주되지 않습니다. 데이터 덮어쓰기를 부분적으로 제공하는 형식은 다음과 같습니다.

유형	구문
스키마	"https://"
호스트 이름	"//website.com"
포트	"//:8000"
경로	"examplepath"



참고

호스트 이름 덮어쓰기를 지정하면 참조된 오브젝트의 호스트 이름 데이터가 교체되며 이는 바람직한 동작이 아닙니다.

위 구문의 모든 조합은 다음과 같은 형식을 사용하여 결합할 수 있습니다.

<scheme:>//<hostname><:port>/<path>

유연성을 개선하기 위해 동일한 오브젝트를 두 번 이상 참조할 수 있습니다.

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":  "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second":  "//:8000"
"serviceaccounts.openshift.io/oauth-redirectreference.second": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

jenkins 라는 경로에 https://example.com 수신이 있다고 가정하면 ***https://example.com:8000*** 및 ***https://example.com/custompath*** *가 모두 유효한 것으로 간주됩니다.*

정적 및 동적 주석을 동시에 사용하여 원하는 동작을 수행할 수 있습니다.

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{\"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{\"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

4.1.4.3.1. OAuth에 대한 API 이벤트

경우에 따라 API 서버는 API 마스터 로그에 직접 액세스하지 않고 디버그하기 어려운 ***예기치 않은 조건*** *오류 메시지를 반환합니다. 근본적인 오류 원인은 인증되지 않은 사용자에게 서버 상태에 대한 정보를 제공하지 않기 위해 의도적으로 숨겨져 있습니다.*

이러한 오류 중 일부는 서비스 계정 OAuth 구성 문제와 관련이 있습니다. 이와 같은 문제는 관리자가 아닌 사용자가 볼 수 있는 이벤트에서 발견됩니다. OAuth 중에 ***unexpected condition*** *서버 오류가 발생하면* ***oc get events****를 실행하여* ***ServiceAccount****에서 해당 이벤트를 확인하십시오.*

다음 예는 적절한 OAuth 리디렉션 URI가 없는 서비스 계정에 대해 경고합니다.

```
$ oc get events | grep ServiceAccount
```

출력 예

```
1m        1m        1         proxy                   ServiceAccount                  Warning
NoSAOAuthRedirectURIs   service-account-oauth-client-getter
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

oc describe sa/<service-account-name>*을 실행하면 지정된 서비스 계정 이름과 관련된 OAuth 이벤트가 보고됩니다.*

```
$ oc describe sa/proxy | grep -A5 Events
```

출력 예

```
Events:
  FirstSeen     LastSeen        Count   From                            SubObjectPath   Type          Reason
Message
  ---------     --------        -----   ----                            -------------   --------      ------                  -------
  3m            3m              1       service-account-oauth-client-getter                             Warning
```

```
NoSAOAuthRedirectURIs   system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI
using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

다음은 가능한 이벤트 오류 목록입니다.

리디렉션 URI 주석이 없거나 유효하지 않은 URI가 지정되었습니다

```
Reason              Message
NoSAOAuthRedirectURIs   system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI
using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

잘못된 경로가 지정되었습니다

```
Reason              Message
NoSAOAuthRedirectURIs   [routes.route.openshift.io "<name>" not found,
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]
```

유효하지 않은 참조 유형이 지정되었습니다

```
Reason              Message
NoSAOAuthRedirectURIs   [no kind "<name>" is registered for version "v1",
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]
```

SA 토큰이 없습니다

```
Reason              Message
NoSAOAuthTokens         system:serviceaccount:myproject:proxy has no tokens
```

4.1.4.3.1.1. 잘못된 구성에서 사용되는 샘플 API 이벤트

다음 단계에서는 사용자가 손상된 상태로 전환하는 방법 한 가지와 문제를 디버그하거나 해결하는 방법을 보여줍니다.

1. 서비스 계정을 OAuth 클라이언트로 사용하여 프로젝트를 생성합니다.

   a. 프록시 서비스 계정 오브젝트에 대한 YAML을 생성하고 경로 **프록시** 를 사용하는지 확인합니다.

      ```
      $ vi serviceaccount.yaml
      ```

      다음 샘플 코드를 추가합니다.

      ```
      apiVersion: v1
      kind: ServiceAccount
      metadata:
        name: proxy
        annotations:
      ```

```
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
'{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"proxy"}}'
```

b. *경로 오브젝트에 대한 YAML을 생성하여 프록시에 대한 보안 연결을 생성합니다.*

```
$ vi route.yaml
```

다음 샘플 코드를 추가합니다.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: proxy
spec:
  to:
    name: proxy
  tls:
    termination: Reencrypt
apiVersion: v1
kind: Service
metadata:
  name: proxy
  annotations:
    service.alpha.openshift.io/serving-cert-secret-name: proxy-tls
spec:
  ports:
  - name: proxy
    port: 443
    targetPort: 8443
  selector:
    app: proxy
```

c. *배포 구성에 대한 YAML을 생성하여 프록시를 사이드카로 시작합니다.*

```
$ vi proxysidecar.yaml
```

다음 샘플 코드를 추가합니다.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: proxy
  template:
    metadata:
      labels:
        app: proxy
    spec:
      serviceAccountName: proxy
```

```
containers:
- name: oauth-proxy
  image: openshift3/oauth-proxy
  imagePullPolicy: IfNotPresent
  ports:
  - containerPort: 8443
    name: public
  args:
  - --https-address=:8443
  - --provider=openshift
  - --openshift-service-account=proxy
  - --upstream=http://localhost:8080
  - --tls-cert=/etc/tls/private/tls.crt
  - --tls-key=/etc/tls/private/tls.key
  - --cookie-secret=SECRET
  volumeMounts:
  - mountPath: /etc/tls/private
    name: proxy-tls

- name: app
  image: openshift/hello-openshift:latest
volumes:
- name: proxy-tls
  secret:
    secretName: proxy-tls
```

d. *세 개의 오브젝트를 생성합니다.*

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

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

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

2. ***oc edit sa/proxy*** *를 실행하여 서비스 계정을 편집하고* ***serviceaccounts.openshift.io/oauth-redirectreference*** *주석을 변경하여 존재하지 않는 경로를 가리킵니다.*

```
apiVersion: v1
imagePullSecrets:
- name: proxy-dockercfg-08d5n
kind: ServiceAccount
metadata:
  annotations:
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
'{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"notexist"}}'
...
```

3. *서비스의 OAuth 로그를 검토하여 서버 오류를 찾습니다.*

```
The authorization server encountered an unexpected condition that prevented it from fulfilling
the request.
```

4. ***oc get events*** *를 실행하여* ***ServiceAccount*** *이벤트를 확인합니다.*

```
$ oc get events | grep ServiceAccount
```

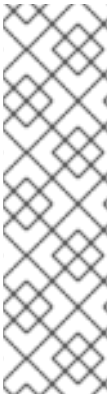
출력 예

```
23m 23m 1 proxy ServiceAccount Warning NoSAOAuthRedirectURIs service-account-oauth-client-getter [routes.route.openshift.io "notexist" not found, system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]
```

4.1.4.4. 통합

OAuth 토큰에 대한 모든 요청에는 **<master>/oauth/authorize** 에 대한 요청이 포함됩니다. 대부분의 인증 통합에서는 이 끝점 앞에 인증 프록시를 배치하거나 백업 ID 공급자에 대한 자격 증명의 유효성을 검사하도록 OpenShift Container Platform을 구성합니다. **<master>/oauth/authorize** 에 대한 요청은 CLI와 같은 대화형 로그인 페이지를 표시할 수 없는 사용자 에이전트에서 가져올 수 있습니다. 따라서 OpenShift Container Platform은 대화형 로그인 flows 외에도 **WWW-Authenticate** 챌린지를 사용한 인증을 지원합니다.

인증 프록시가 **<master>/oauth/authorize** 끝점 앞에 배치되면 대화형 로그인 페이지를 표시하거나 대화형 로그인 flow로 리디렉션하는 대신 인증되지 않은 브라우저가 아닌 사용자 에이전트 **WWW-Authenticate 챌린지** 를 보내야 합니다.



참고

브라우저 클라이언트에 대한 CSV(Cross-site Request Forgery) 공격을 방지하기 위해 **X-CSRF-Token** 헤더가 요청에 있는 경우에만 기본 인증 챌린지를 보내야 합니다. 기본 **WWW-Authenticate 챌린지**를 수신해야 하는 클라이언트는 이 헤더를 비어 있지 않은 값으로 설정해야 합니다.

인증 프록시가 **WWW-Authenticate 챌린지**를 지원할 수 없거나 OpenShift Container Platform이 WWW-Authenticate 챌린지를 지원하지 않는 ID 공급자를 사용하도록 구성된 경우 사용자는 **<master>/oauth/token/request** 를 통해 수동으로 액세스 토큰을 가져올 수 있습니다.

4.1.4.5. OAuth 서버 메타데이터

OpenShift Container Platform에서 실행되는 애플리케이션은 내장 OAuth 서버에 대한 정보를 검색해야 할 수 있습니다. 예를 들어, 수동 구성 없이 **<master>** 서버의 주소를 검색해야 할 수 있습니다. 이러한 작업을 지원하기 위해 OpenShift Container Platform에서는 IETF OAuth 2.0 인증 서버 메타데이터 초안 사양을 구현합니다.

따라서 클러스터 내에서 실행되는 모든 애플리케이션은 **https://openshift.default.svc/.well-known/oauth-authorization-server** 에 **GET** 요청을 발행하여 다음 정보를 가져올 수 있습니다.

```
{
  "issuer": "https://<master>", (1)
  "authorization_endpoint": "https://<master>/oauth/authorize", (2)
  "token_endpoint": "https://<master>/oauth/token", (3)
  "scopes_supported": [ (4)
    "user:full",
    "user:info",
    "user:check-access",
```

```
    "user:list-scoped-projects",
    "user:list-projects"
  ],
  "response_types_supported": [ (5)
    "code",
    "token"
  ],
  "grant_types_supported": [ (6)
    "authorization_code",
    "implicit"
  ],
  "code_challenge_methods_supported": [ (7)
    "plain",
    "S256"
  ]
}
```

(1) ***https*** 체계를 사용하고 쿼리 또는 조각 구성 요소가 없는 URL에 해당하는 권한 부여 서버의 발행자 식별자. 권한 부여 서버에 대한 정보가 포함된 ***.well-known*** RFC 5785 리소스가 게시되는 위치입니다.

(2) 권한 부여 서버의 권한 부여 끝점 URL. RFC 6749를 참조하십시오.

(3) 권한 서버의 토큰 Endpoint URL 입니다. RFC 6749를 참조하십시오.

(4) 이 권한 부여 서버에서 지원하는 OAuth 2.0 RFC 6749 범위 값 목록이 포함된 JSON 배열. 지원되는 모든 범위 값이 제공되는 것은 아닙니다.

(5) 이 권한 부여 서버에서 지원하는 OAuth 2.0 ***response_type*** 값 목록이 포함된 JSON 배열. 사용된 배열 값은 RFC 7591의 "OAuth 2.0 동적 클라이언트 등록 프로토콜"에서 정의한 ***response_types*** 매개변수에 사용된 것과 동일합니다.

(6) 이 권한 부여 서버에서 지원하는 OAuth 2.0 부여 유형 값 목록이 포함된 JSON 배열. 사용된 배열 값은 RFC 7591의 ***OAuth 2.0 동적 클라이언트 등록 프로토콜***에서 정의한 ***grant_types*** 매개변수에 사용된 것과 동일합니다.

(7) 이 권한 부여 서버에서 지원하는 PKCE RFC 7636 코드 챌린지 방법 목록이 포함된 JSON 배열. 코드 챌린지 방법 값은 RFC 7636의 4.3절에 정의된 ***code_challenge_method*** 매개변수에 사용됩니다. 유효한 코드 챌린지 방법 값은 IANA ***PKCE 코드 챌린지 방법*** 레지스트리에 등록된 값입니다. IANA OAuth 매개변수를 참조하십시오.

4.1.4.6. OAuth 토큰 가져오기

OAuth 서버는 표준 인증 코드 부여 및 암시적 부여 OAuth 인증 flows를 지원합니다.

인증 코드 부여 방법을 사용하여 OAuth 토큰을 요청하도록 다음 명령을 실행합니다.

```
$ curl -H "X-Remote-User: <username>" \
    --cacert /etc/origin/master/ca.crt \
    --cert /etc/origin/master/admin.crt \
    --key /etc/origin/master/admin.key \
    -I https://<master-address>/oauth/authorize?response_type=token\&client_id=openshift-challenging-client | grep -oP "access_token=\K[^&]*"
```

__WWW-Authenticate challenges__(예: __openshift-challenging-client__)를 요청하기 위해 구성된 client_id와 함께 암시적 부여 flow(__response_type=token__)를 사용하여 OAuth 토큰을 요청하는 경우, __/oauth/authorize__에서 제공 가능한 서버 응답 및 처리 방법은 다음과 같습니다.

상태	내용	클라이언트 응답
302	URL 조각에 **access_token** 매개변수를 포함하는 **Location** 헤더(RFC 4.2.2)	**access_token** 값을 OAuth 토큰으로 사용하십시오.
302	**error** 쿼리 매개변수를 포함하는 **Location** 헤더(RFC 4.1.2.1)	실패, **error** (및 선택적 **error_description**) 쿼리 값을 사용자에게 선택적으로 표시합니다.
302	기타 **Location** 헤더	리디렉션을 따르고 해당 규칙을 사용하여 결과를 처리하십시오.
401	**WWW-Authenticate** 헤더 있음	유형이 인식되면(예: **Basic**, **Negotiate**) 챌린지에 응답하고 요청을 다시 제출한 후 해당 규칙을 사용하여 결과를 처리하십시오.
401	**WWW-Authenticate** 헤더 없음	인증할 수 있는 챌린지가 없습니다. 실패했으며 응답 본문(OAuth 토큰을 가져올 수 있는 링크 및 대체 방법에 대한 세부 사항이 포함될 수 있음)을 표시합니다.
기타	기타	실패, 사용자에게 응답 본문을 선택적으로 표시합니다.

암시적 부여 흐름을 사용하여 OAuth 토큰을 요청하려면 다음을 수행합니다.

```
$ curl -u <username>:<password>
'https://<master-address>:8443/oauth/authorize?client_id=openshift-challenging-
client&response_type=token' -skv / (1)
/ -H "X-CSRF-Token: xxx" (2)
```

출력 예

```
*   Trying 10.64.33.43...
* Connected to 10.64.33.43 (10.64.33.43) port 8443 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 592 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_128_GCM_SHA256
*       server certificate verification SKIPPED
*       server certificate status verification SKIPPED
*       common name: 10.64.33.43 (matched)
*       server certificate expiration date OK
*       server certificate activation date OK
*       certificate public key: RSA
*       certificate version: #3
*       subject: CN=10.64.33.43
*       start date: Thu, 09 Aug 2018 04:00:39 GMT
```

```
*       expire date: Sat, 08 Aug 2020 04:00:40 GMT
*       issuer: CN=openshift-signer@1531109367
*       compression: NULL
* ALPN, server accepted to use http/1.1
* Server auth using Basic with user 'developer'
> GET /oauth/authorize?client_id=openshift-challenging-client&response_type=token HTTP/1.1
> Host: 10.64.33.43:8443
> Authorization: Basic ZGV2ZWxvcGVyOmRzc2Zkcw==
> User-Agent: curl/7.47.0
> Accept: */*
> X-CSRF-Token: xxx
>
< HTTP/1.1 302 Found
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Expires: Fri, 01 Jan 1990 00:00:00 GMT
< Location:
https://10.64.33.43:8443/oauth/token/implicit#access_token=gzTwOq_mVJ7ovHliHBTgRQEEXa1aCZD
9lnj7lSw3ekQ&expires_in=86400&scope=user%3Afull&token_type=Bearer (1)
< Pragma: no-cache
< Set-Cookie:
ssn=MTUzNTk0OTc1MnxIckVfNW5vNFlLSlF5MF9GWEF6Zm55Vl95bi1ZNE41S1NCbFJMYnN1TWV
wR1hwZmlLMzFQRklzVXRkc0RnUGEzdnBEa0NZZndXV2ZUVzN1dmFPM2dHSUlzUmVXakQ3Q09rV
XpxNlRoVmVkQU5DYmdLTE9SUWlyNkJJTm1mSDQ0N2pCV09La3gzMkMzckwxc1V1QXpybFlXT2ZY
SmI2R2FTVEZsdDBzRjJ8vk6zrQPjQUmoJCqb8Dt5j5s0b4wZlITgKlho9wlKAZI=; Path=/; HttpOnly;
Secure
< Date: Mon, 03 Sep 2018 04:42:32 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host 10.64.33.43 left intact
```

(1) ***client-id** 가 **openshift-challenging-client** 로 설정되고 **response-type** 이 **토큰** 으로 설정됩니다.*

(2) ***X-CSRF-Token** 헤더를 비어 있지 않은 값으로 설정합니다.*

(1) *토큰은 **access_token=gzTwOq_mVJ7ovHliHBTgRQEEXa1aCZD9lnj7lSw3ekQ** 로 **302** 응답의 **Location** 헤더에 반환됩니다.*

OAuth 토큰 값만 보려면 다음 명령을 실행합니다.

```
$ curl -u <username>:<password> /
'https://<master-address>:8443/oauth/authorize?client_id=openshift-challenging-
client&response_type=token' / (1)
-skv -H "X-CSRF-Token: xxx" --stderr - |  grep -oP "access_token=\K[^&]*" (2)
```

(1) ***client-id** 가 **openshift-challenging-client** 로 설정되고 **response-type** 이 **토큰** 으로 설정됩니다.*

(2) ***X-CSRF-Token** 헤더를 비어 있지 않은 값으로 설정합니다.*

출력 예

```
hvqxe5aMlAzvbqfM2WWw3D6tR0R2jCQGKx0viZBxwmc
```

코드 부여 방법을 사용하여 토큰을 요청할 수도 있습니다.

4.1.4.7. Prometheus의 인증 지표

OpenShift Container Platform은 인증 시도 중 다음과 같은 Prometheus 시스템 지표를 캡처합니다.

- ***openshift_auth_basic_password_count*는 *oc login* 사용자 이름 및 암호 시도 횟수를 계산합니다.**
- ***openshift_auth_basic_password_count_result* 는 *oc login* 사용자 이름 및 암호 시도 횟수(성공 또는 오류)를 계산합니다.**
- ***openshift_auth_form_password_count*는 웹 콘솔 로그인 시도 횟수를 계산합니다.**
- ***openshift_auth_form_password_count_result* 는 웹 콘솔 로그인 시도 횟수(성공 또는 오류)를 계산합니다.**
- ***openshift_auth_password_total*은 총 *oc login* 및 웹 콘솔 로그인 시도 횟수를 계산합니다.**

4.2. 권한 부여

4.2.1. 개요

RBAC(역할 기반 액세스 제어) 오브젝트는 사용자가 프로젝트 내에서 지정된 작업을 수행할 수 있는지 여부를 결정합니다.

이를 통해 플랫폼 관리자는 클러스터 역할 및 바인딩 을 사용하여 OpenShift Container Platform 플랫폼 자체와 모든 프로젝트에 대해 다양한 액세스 수준을 보유한 사용자를 제어할 수 있습니다.

이를 통해 개발자는 로컬 역할 및 바인딩 을 사용하여 프로젝트에 액세스할 수 있는 사용자를 제어할 수 있습니다. 권한 부여는 인증과 별도의 단계이며 , 이는 조치를 수행할 사용자의 ID를 결정하는 데 더 중요합니다.

권한 부여는 다음을 사용하여 관리합니다.

규칙	오브젝트 집합에 허용되는 동사 집합입니다. 예를 들어 포드를 **생성할** 수 있는지 여부입니다.
역할	규칙 모음입니다. 사용자와 그룹은 동시에 여러 역할과 연결하거나 *바인딩* 할 수 있습니다.
바인딩	역할이 있는 사용자 및/또는 그룹 간 연결 .

클러스터 관리자는 CLI를 사용하여 규칙, 역할 및 바인딩을 시각화할 수 있습니다.

*예를 들어 **admin** 및 **basic-user** 기본 클러스터 역할에 대한 규칙 세트를 보여주는 다음 발췌 사항을 고려하십시오.*

```
$ oc describe clusterrole.rbac admin basic-user
```

출력 예

```
Name:  admin
Labels:  <none>
```

```
Annotations: openshift.io/description=A user that has edit rights within the project and can change the
project's membership.
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources       Non-Resource URLs Resource Names Verbs
  ---------       ----------------- -------------- -----
  appliedclusterresourcequotas     []   []  [get list watch]
  appliedclusterresourcequotas.quota.openshift.io  []   []  [get list watch]
  bindings       []   []  [get list watch]
  buildconfigs       []   []  [create delete deletecollection get list patch update watch]
  buildconfigs.build.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  buildconfigs/instantiate     []   []  [create]
  buildconfigs.build.openshift.io/instantiate   []   []  [create]
  buildconfigs/instantiatebinary    []   []  [create]
  buildconfigs.build.openshift.io/instantiatebinary  []   []  [create]
  buildconfigs/webhooks      []   []  [create delete deletecollection get list patch update watch]
  buildconfigs.build.openshift.io/webhooks   []   []  [create delete deletecollection get list patch update
watch]
  buildlogs       []   []  [create delete deletecollection get list patch update watch]
  buildlogs.build.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  builds       []   []  [create delete deletecollection get list patch update watch]
  builds.build.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  builds/clone       []   []  [create]
  builds.build.openshift.io/clone    []   []  [create]
  builds/details      []   []  [update]
  builds.build.openshift.io/details    []   []  [update]
  builds/log       []   []  [get list watch]
  builds.build.openshift.io/log     []   []  [get list watch]
  configmaps       []   []  [create delete deletecollection get list patch update watch]
  cronjobs.batch      []   []  [create delete deletecollection get list patch update watch]
  daemonsets.extensions      []   []  [get list watch]
  deploymentconfigrollbacks     []   []  [create]
  deploymentconfigrollbacks.apps.openshift.io   []   []  [create]
  deploymentconfigs      []   []  [create delete deletecollection get list patch update watch]
  deploymentconfigs.apps.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  deploymentconfigs/instantiate     []   []  [create]
  deploymentconfigs.apps.openshift.io/instantiate  []   []  [create]
  deploymentconfigs/log      []   []  [get list watch]
  deploymentconfigs.apps.openshift.io/log   []   []  [get list watch]
  deploymentconfigs/rollback     []   []  [create]
  deploymentconfigs.apps.openshift.io/rollback   []   []  [create]
  deploymentconfigs/scale     []   []  [create delete deletecollection get list patch update watch]
  deploymentconfigs.apps.openshift.io/scale   []   []  [create delete deletecollection get list patch
update watch]
  deploymentconfigs/status     []   []  [get list watch]
  deploymentconfigs.apps.openshift.io/status   []   []  [get list watch]
  deployments.apps      []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions     []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions/rollback    []   []  [create delete deletecollection get list patch update watch]
  deployments.apps/scale     []   []  [create delete deletecollection get list patch update watch]
  deployments.extensions/scale     []   []  [create delete deletecollection get list patch update watch]
  deployments.apps/status     []   []  [create delete deletecollection get list patch update watch]
  endpoints       []   []  [create delete deletecollection get list patch update watch]
  events       []   []  [get list watch]
  horizontalpodautoscalers.autoscaling    []   []  [create delete deletecollection get list patch update
```

```
watch]
  horizontalpodautoscalers.extensions    []   []  [create delete deletecollection get list patch update
watch]
  imagestreamimages      []   []  [create delete deletecollection get list patch update watch]
  imagestreamimages.image.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  imagestreamimports      []   []  [create]
  imagestreamimports.image.openshift.io    []   []  [create]
  imagestreammappings      []   []  [create delete deletecollection get list patch update watch]
  imagestreammappings.image.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  imagestreams      []   []  [create delete deletecollection get list patch update watch]
  imagestreams.image.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  imagestreams/layers      []   []  [get update]
  imagestreams.image.openshift.io/layers   []   []  [get update]
  imagestreams/secrets      []   []  [create delete deletecollection get list patch update watch]
  imagestreams.image.openshift.io/secrets   []   []  [create delete deletecollection get list patch update
watch]
  imagestreams/status      []   []  [get list watch]
  imagestreams.image.openshift.io/status   []   []  [get list watch]
  imagestreamtags      []   []  [create delete deletecollection get list patch update watch]
  imagestreamtags.image.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  jenkins.build.openshift.io     []   []  [admin edit view]
  jobs.batch       []   []  [create delete deletecollection get list patch update watch]
  limitranges       []   []  [get list watch]
  localresourceaccessreviews     []   []  [create]
  localresourceaccessreviews.authorization.openshift.io  []   []  [create]
  localsubjectaccessreviews     []   []  [create]
  localsubjectaccessreviews.authorization.k8s.io  []   []  [create]
  localsubjectaccessreviews.authorization.openshift.io  []   []  [create]
  namespaces       []   []  [get list watch]
  namespaces/status      []   []  [get list watch]
  networkpolicies.extensions     []   []  [create delete deletecollection get list patch update watch]
  persistentvolumeclaims     []   []  [create delete deletecollection get list patch update watch]
  pods       []   []  [create delete deletecollection get list patch update watch]
  pods/attach       []   []  [create delete deletecollection get list patch update watch]
  pods/exec       []   []  [create delete deletecollection get list patch update watch]
  pods/log       []   []  [get list watch]
  pods/portforward      []   []  [create delete deletecollection get list patch update watch]
  pods/proxy       []   []  [create delete deletecollection get list patch update watch]
  pods/status       []   []  [get list watch]
  podsecuritypolicyreviews     []   []  [create]
  podsecuritypolicyreviews.security.openshift.io  []   []  [create]
  podsecuritypolicyselfsubjectreviews    []   []  [create]
  podsecuritypolicyselfsubjectreviews.security.openshift.io []   []  [create]
  podsecuritypolicysubjectreviews    []   []  [create]
  podsecuritypolicysubjectreviews.security.openshift.io  []   []  [create]
  processedtemplates      []   []  [create delete deletecollection get list patch update watch]
  processedtemplates.template.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  projects       []   []  [delete get patch update]
  projects.project.openshift.io     []   []  [delete get patch update]
  replicasets.extensions     []   []  [create delete deletecollection get list patch update watch]
  replicasets.extensions/scale     []   []  [create delete deletecollection get list patch update watch]
  replicationcontrollers     []   []  [create delete deletecollection get list patch update watch]
```

```
  replicationcontrollers/scale     []   []  [create delete deletecollection get list patch update watch]
  replicationcontrollers.extensions/scale   []   []  [create delete deletecollection get list patch update
watch]
  replicationcontrollers/status     []   []  [get list watch]
  resourceaccessreviews      []   []  [create]
  resourceaccessreviews.authorization.openshift.io  []   []  [create]
  resourcequotas      []   []  [get list watch]
  resourcequotas/status      []   []  [get list watch]
  resourcequotausages      []   []  [get list watch]
  rolebindingrestrictions     []   []  [get list watch]
  rolebindingrestrictions.authorization.openshift.io  []   []  [get list watch]
  rolebindings       []   []  [create delete deletecollection get list patch update watch]
  rolebindings.authorization.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  rolebindings.rbac.authorization.k8s.io   []   []  [create delete deletecollection get list patch update
watch]
  roles       []   []  [create delete deletecollection get list patch update watch]
  roles.authorization.openshift.io    []   []  [create delete deletecollection get list patch update watch]
  roles.rbac.authorization.k8s.io    []   []  [create delete deletecollection get list patch update watch]
  routes       []   []  [create delete deletecollection get list patch update watch]
  routes.route.openshift.io     []   []  [create delete deletecollection get list patch update watch]
  routes/custom-host      []   []  [create]
  routes.route.openshift.io/custom-host    []   []  [create]
  routes/status       []   []  [get list watch update]
  routes.route.openshift.io/status    []   []  [get list watch update]
  scheduledjobs.batch      []   []  [create delete deletecollection get list patch update watch]
  secrets       []   []  [create delete deletecollection get list patch update watch]
  serviceaccounts      []   []  [create delete deletecollection get list patch update watch impersonate]
  services       []   []  [create delete deletecollection get list patch update watch]
  services/proxy      []   []  [create delete deletecollection get list patch update watch]
  statefulsets.apps      []   []  [create delete deletecollection get list patch update watch]
  subjectaccessreviews      []   []  [create]
  subjectaccessreviews.authorization.openshift.io  []   []  [create]
  subjectrulesreviews      []   []  [create]
  subjectrulesreviews.authorization.openshift.io  []   []  [create]
  templateconfigs      []   []  [create delete deletecollection get list patch update watch]
  templateconfigs.template.openshift.io    []   []  [create delete deletecollection get list patch update
watch]
  templateinstances      []   []  [create delete deletecollection get list patch update watch]
  templateinstances.template.openshift.io   []   []  [create delete deletecollection get list patch update
watch]
  templates       []   []  [create delete deletecollection get list patch update watch]
  templates.template.openshift.io    []   []  [create delete deletecollection get list patch update watch]


Name:  basic-user
Labels:  <none>
Annotations: openshift.io/description=A user that can get basic information about projects.
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources      Non-Resource URLs Resource Names Verbs
  ---------      ----------------- -------------- -----
  clusterroles      []   []  [get list]
  clusterroles.authorization.openshift.io  []   []  [get list]
  clusterroles.rbac.authorization.k8s.io  []   []  [get list watch]
  projectrequests     []   []  [list]
```

```
projectrequests.project.openshift.io   []   []  [list]
projects      []   []  [list watch]
projects.project.openshift.io    []   []  [list watch]
selfsubjectaccessreviews.authorization.k8s.io  []   []  [create]
selfsubjectrulesreviews    []   []  [create]
selfsubjectrulesreviews.authorization.openshift.io []   []  [create]
storageclasses.storage.k8s.io    []   []  [get list]
users      []   [~]  [get]
users.user.openshift.io    []   [~]  [get]
```

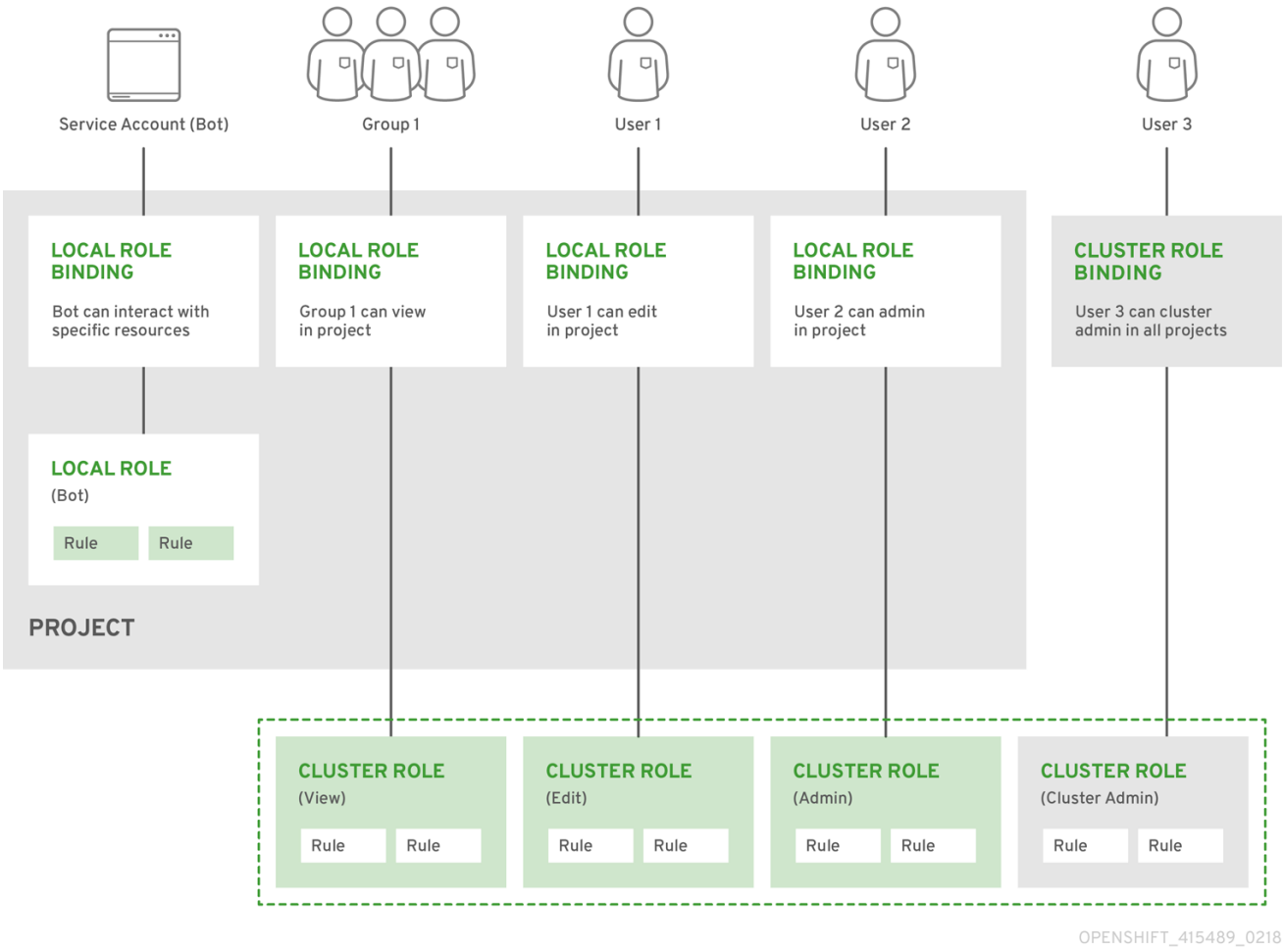
로컬 역할 바인딩 보기의 다음 발췌 내용은 다양한 사용자 및 그룹에 바인딩된 위의 역할을 보여줍니다.

```
$ oc describe rolebinding.rbac admin basic-user -n alice-project
```

출력 예

```
Name:  admin
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name  Namespace
  ---- ----  ---------
  User system:admin
  User alice


Name:  basic-user
Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind Name Namespace
  ---- ---- ---------
  User joe
  Group devel
```

아래에는 클러스터 역할, 로컬 역할, 클러스터 역할 바인딩, 로컬 역할 바인딩, 사용자, 그룹, 서비스 계정 간의 관계가 설명되어 있습니다.



4.2.2. 권한 부여 평가

OpenShift Container Platform에서 권한 부여를 평가할 때 몇 가지 요소가 결합되어 결정됩니다.

Identity	권한 부여 컨텍스트에서 사용자 이름 및 사용자가 속하는 그룹 목록.
작업	수행 중인 작업. 대부분의 경우 다음으로 구성됩니다. 프로젝트: 액세스 중인 프로젝트입니다. 동사: **get**,**list**,**create**,**update**,**patch**, **delete**, **delete collection** 또는 **watch** 가 될 수 있습니다. 리소스 이름: 액세스 중인 API 엔드포인트입니다.
바인딩	전체 바인딩 목록입니다.

OpenShift Container Platform은 다음 단계를 사용하여 권한 부여를 평가합니다.

1. *ID 및 프로젝트 범위 작업은 사용자 또는 해당 그룹에 적용되는 모든 바인딩을 찾는 데 사용됩니다.*
2. *바인딩은 적용되는 모든 역할을 찾는 데 사용됩니다.*

3. *역할은 적용되는 모든 규칙을 찾는 데 사용됩니다.*
4. *일치하는 규칙을 찾기 위해 작업을 각 규칙에 대해 확인합니다.*
5. *일치하는 규칙이 없으면 기본적으로 작업이 거부됩니다.*

4.2.3. 클러스터 및 로컬 RBAC

권한 부여를 제어하는 두 가지 수준의 RBAC 역할 및 바인딩이 있습니다.

클러스터 RBAC	모든 프로젝트에 적용할 수 있는 역할 및 바인딩입니다. 클러스터 전체에 존재하는 역할은 *클러스터 역할*로 간주됩니다. 클러스터 역할 바인딩은 클러스터 역할만 참조할 수 있습니다.
지역 RBAC	지정된 프로젝트에 적용되는 역할 및 바인딩입니다. 프로젝트에만 있는 역할은 *로컬 역할*로 간주됩니다. 로컬 역할 바인딩은 클러스터 및 로컬 역할을 모두 참조할 수 있습니다.

이 2단계 계층 구조에서는 클러스터 역할을 통해 여러 프로젝트에서 재사용할 수 있는 동시에 로컬 역할을 통해 개별 프로젝트 내에서 사용자 지정할 수 있습니다.

평가 중에는 클러스터 역할 바인딩과 로컬 역할 바인딩이 모두 사용됩니다. 예를 들면 다음과 같습니다.

1. *클러스터 전체의 "허용" 규칙을 확인합니다.*
2. *로컬 바인딩된 "허용" 규칙을 확인합니다.*
3. *기본적으로 거부합니다.*

4.2.4. 클러스터 역할 및 로컬 역할

역할은 일련의 리소스에서 수행할 수 있는 허용된 동사 집합인 정책 규칙 컬렉션입니다. OpenShift Container Platform에는 사용자 및 클러스터 전체 또는 로컬에 바인딩할 수 있는 기본 클러스터 역할 세트가 포함되어 있습니다.

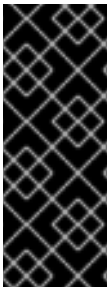
기본 클러스터 역할	설명
admin	프로젝트 관리자입니다. 로컬 바인딩 에서 사용되는 경우 **admin** 사용자는 프로젝트의 모든 리소스를 보고 할당량을 제외한 프로젝트의 모든 리소스를 수정할 수 있습니다.
basic-user	프로젝트 및 사용자에 대한 기본 정보를 가져올 수 있는 사용자입니다.
cluster-admin	모든 프로젝트에서 모든 작업을 수행할 수 있는 슈퍼 유저입니다. 로컬 바인딩을 사용하여 사용자에게 바인딩 하면 할당량과 프로젝트의 모든 리소스에 대한 모든 작업을 완전히 제어할 수 있습니다.
cluster-status	기본 클러스터 상태 정보를 가져올 수 있는 사용자입니다.
edit	프로젝트에서 대부분의 오브젝트를 수정할 수 있지만 역할 또는 바인딩을 보거나 수정할 권한은 없는 사용자입니다.
self-provisioner	자체 프로젝트를 만들 수 있는 사용자입니다.

기본 클러스터 역할	설명
view	수정할 수는 없지만 프로젝트의 오브젝트를 대부분 볼 수 있는 사용자입니다. 역할 또는 바인딩을 보거나 수정할 수 없습니다.
cluster-reader	클러스터의 개체를 읽을 수는 있지만 볼 수 없는 사용자입니다.

작은 정보

사용자와 그룹은 동시에 여러 역할과 연결되거나 바인딩 될 수 있습니다.

프로젝트 관리자는 각각 CLI를 사용하여 연결된 동사 및 리소스의 매트릭스를 포함하여 역할을 시각화하여 로컬 역할 및 바인딩을 볼 수 있습니다.



중요

프로젝트 관리자에게 바인딩된 클러스터 역할은 로컬 바인딩 을 통해 프로젝트에서 제한됩니다. cluster ***-admin 또는 system:admin 에 부여된 클러스터 역할과 같이 클러스터*** *전체에 바인딩되지 않습니다.*

클러스터 역할은 클러스터 수준에서 정의된 역할 이지만 클러스터 수준 또는 프로젝트 수준에서 바인딩할 수 있습니다.

프로젝트의 로컬 역할을 생성하는 방법을 알아봅니다.

4.2.4.1. 클러스터 역할 업데이트

OpenShift Container Platform 클러스터 업그레이드 후 서버가 시작될 때 기본 역할이 업데이트되고 자동으로 조정됩니다. 조정 중에 기본 역할에서 누락된 권한이 추가됩니다. 역할에 더 많은 권한을 추가하면 제거되지 않습니다.

기본 역할을 사용자 지정하고 자동 역할 조정을 방지하도록 구성한 경우 OpenShift Container Platform을 업그레이드할 때 정책 정의를 수동으로 업데이트해야 합니다.

4.2.4.2. 사용자 지정 역할 및 권한 적용

사용자 지정 역할 및 권한을 업데이트하려면 다음 명령을 사용하는 것이 좋습니다.

```
$ oc auth reconcile -f <file> 1
```

1 ***<file>*** *은 적용할 역할의 절대 경로와 권한입니다.*

이 명령을 사용하여 새 사용자 지정 역할 및 권한을 추가할 수도 있습니다. 새 사용자 지정 역할의 이름이 기존 역할과 동일한 경우 기존 역할이 업데이트됩니다. 클러스터 관리자는 동일한 이름의 사용자 지정 역할이 이미 있음을 알리지 않습니다.

이 명령은 다른 클라이언트를 중단하지 않는 방식으로 새 권한이 제대로 적용되었는지 확인합니다. 이는 컴퓨팅 논리에서 내부적으로 규칙 집합 간 작업을 다룹니다. 이 작업은 RBAC 리소스에서 JSON 병합을 통해 수행할 수 없습니다.

4.2.4.3. 클러스터 역할 집계

기본 **admin,edit,view** 및 **cluster-reader** 클러스터 역할은 새 규칙이 생성될 때 각 역할에 대한 클러스터 규칙이 동적으로 업데이트되는 클러스터 역할 *집계* 를 지원합니다. 이 기능은 *사용자 정의 리소스를 생성하여 Kubernetes API를 확장하는 경우에만 관련이 있습니다*.

클러스터 역할 집계 사용 방법을 알아봅니다.

4.2.5. 보안 컨텍스트 제약 조건

사용자가 수행할 수 있는 작업을 제어하는 *RBAC 리소스* 외에도 OpenShift Container Platform은 *Pod* 에서 수행할 수 있는 작업과 액세스할 수 있는 작업을 제어하는 SCC(보안 컨텍스트 제약 조건)를 제공합니다. 관리자는 CLI *를 사용하여 SCC를 관리할* 수 있습니다.

SCC는 영구 스토리지에 대한 액세스를 관리하는 데도 매우 유용합니다.

SCC는 시스템에 적용하기 위해 Pod를 실행해야 하는 조건 집합을 정의하는 오브젝트입니다. 이를 통해 관리자는 다음을 제어할 수 있습니다.

1. *권한 있는 컨테이너* 실행.
2. 컨테이너가 추가하도록 요청할 수 있는 기능.
3. 호스트 디렉터리를 볼륨으로 사용.
4. 컨테이너의 SELinux 컨텍스트
5. 사용자 ID입니다.
6. 호스트 네임스페이스 및 네트워킹 사용
7. Pod의 볼륨을 보유한 **FSGroup** 할당
8. 허용 가능한 추가 그룹 구성
9. 읽기 전용 루트 파일 시스템 사용 필요
10. 볼륨 유형의 사용 제어
11. 허용 가능한 seccomp 프로필 구성

기본적으로 7개의 SCC가 클러스터에 추가되며 CLI를 사용하여 클러스터 관리자가 볼 수 있습니다.

```
$ oc get scc
```

출력 예

```
NAME              PRIV    CAPS    SELINUX     RUNASUSER        FSGROUP     SUPGROUP
PRIORITY   READONLYROOTFS   VOLUMES
anyuid            false   []      MustRunAs   RunAsAny         RunAsAny    RunAsAny    10         false
[configMap downwardAPI emptyDir persistentVolumeClaim secret]
hostaccess        false   []      MustRunAs   MustRunAsRange   MustRunAs   RunAsAny    <none>
```

```
false           [configMap downwardAPI emptyDir hostPath persistentVolumeClaim secret]
hostmount-anyuid   false   []      MustRunAs   RunAsAny         RunAsAny   RunAsAny   <none>
false           [configMap downwardAPI emptyDir hostPath nfs persistentVolumeClaim secret]
hostnetwork        false   []      MustRunAs   MustRunAsRange   MustRunAs   MustRunAs   <none>
false           [configMap downwardAPI emptyDir persistentVolumeClaim secret]
nonroot            false   []      MustRunAs   MustRunAsNonRoot   RunAsAny   RunAsAny   <none>
false           [configMap downwardAPI emptyDir persistentVolumeClaim secret]
privileged         true    [*]     RunAsAny    RunAsAny         RunAsAny   RunAsAny   <none>
false           [*]
restricted         false   []      MustRunAs   MustRunAsRange   MustRunAs   RunAsAny   <none>
false           [configMap downwardAPI emptyDir persistentVolumeClaim secret]
```

중요

기본 SCC를 수정하지 마십시오. OpenShift Container Platform을 업그레이드할 때 기본 SCC를 사용자 정의하면 문제가 발생할 수 있습니다. 대신 새 SCC를 만듭니다.

CLI를 사용하여 클러스터 관리자가 각 SCC에 대한 정의도 볼 수 있습니다. 예를 들어 권한 있는 SCC의 경우 다음을 수행합니다.

```
$ oc get -o yaml --export scc/privileged
```

출력 예

```
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: 1
- '*'
apiVersion: v1
defaultAddCapabilities: [] 2
fsGroup: 3
  type: RunAsAny
groups: 4
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: 'privileged allows access to all privileged and host
      features and the ability to run as any user, any group, any fsGroup, and with
      any SELinux context.  WARNING: this is the most relaxed SCC and should be used
      only for cluster administration. Grant with caution.'
  creationTimestamp: null
  name: privileged
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: [] 5
runAsUser: 6
  type: RunAsAny
```

```
seLinuxContext: 7
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups: 8
  type: RunAsAny
users: 9
- system:serviceaccount:default:registry
- system:serviceaccount:default:router
- system:serviceaccount:openshift-infra:build-controller
volumes:
- '*'
```

1. Pod에서 요청할 수 있는 기능 목록입니다. 빈 목록은 기능을 요청할 수 없음을 나타내고, 특수 기호 *는 모든 기능을 요청할 수 있음을 나타냅니다.
2. 모든 포드에 추가할 추가 기능 목록입니다.
3. 보안 컨텍스트에 허용되는 값을 지시하는 **FSGroup** 전략입니다.
4. 이 SCC에 액세스할 수 있는 그룹입니다.
5. Pod에서 삭제될 기능 목록입니다.
6. 보안 컨텍스트에 허용되는 값을 지시하는 사용자 전략 유형으로 실행합니다.
7. 보안 컨텍스트에 허용되는 값을 지정하는 SELinux 컨텍스트 전략 유형입니다.
8. 보안 컨텍스트에 허용되는 추가 그룹을 지시하는 보충 그룹 전략입니다.
9. 이 SCC에 액세스할 수 있는 사용자입니다.

SCC의 **users** 및 **groups** 필드는 사용할 수 있는 SCC를 제어합니다. 기본적으로 클러스터 관리자, 노드 및 빌드 컨트롤러에는 권한 있는 SCC에 대한 액세스 권한이 부여됩니다. 인증된 모든 사용자에게는 제한된 SCC에 대한 액세스 권한이 부여됩니다.

Docker에는 포드의 각 컨테이너에 허용되는 기본 기능 목록이 있습니다. 컨테이너에서는 이 기본 목록의 기능을 사용하지만 Pod 매니페스트 작성자는 추가 기능을 요청하거나 일부 기본값을 삭제하여 이를 변경할 수 있습니다. **allowedCapabilities**,**defaultAddCapabilities** 및 **requiredDropCapabilities** 필드는 Pod에서 이러한 요청을 제어하고 요청 가능한 기능, 각 컨테이너에 추가해야 하는 기능 및 금지해야 하는 기능을 지정하는 데 사용됩니다.

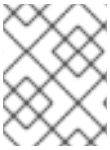
권한 있는 SCC:

- 권한 있는 Pod를 허용합니다.
- 호스트 디렉터리를 볼륨으로 마운트할 수 있습니다.
- 포드가 모든 사용자로 실행되도록 허용합니다.
- Pod가 모든 MCS 레이블로 실행되도록 허용합니다.
- 포드에서 호스트의 IPC 네임스페이스를 사용하도록 허용합니다.
- 포드에서 호스트의 PID 네임스페이스를 사용하도록 허용합니다.

- *Pod에서 FSGroup을 사용하도록 허용합니다.*
- *포드에서 추가 그룹을 사용하도록 허용합니다.*
- *포드에서 seccomp 프로필을 사용할 수 있습니다.*
- *포드에서 기능을 요청할 수 있습니다.*

restricted SCC:

- *Pod가 권한에 따라 실행되지 않도록 합니다.*
- *Pod에서 호스트 디렉터리 볼륨을 사용할 수 없는지 확인합니다.*
- *Pod를 미리 할당된 UID 범위에서 사용자로 실행해야 합니다.*
- *Pod를 사전 할당된 MCS 라벨로 실행해야 합니다.*
- *Pod에서 FSGroup을 사용하도록 허용합니다.*
- *포드에서 추가 그룹을 사용하도록 허용합니다.*



참고

*각 SCC에 대한 자세한 내용은 SCC에 제공되는 **kubernetes.io/description** 주석을 참조하십시오.*

SCC는 포드에서 액세스할 수 있는 보안 기능을 제어하는 설정 및 전략으로 구성됩니다. 이 설정은 세 가지 범주로 분류됩니다.

부울로 제어	이 유형의 필드는 기본적으로 가장 제한적인 값으로 설정됩니다. 예를 들어, **AllowPrivilegedContainer**는 값이 지정되지 않은 경우 항상 **false**로 설정됩니다.
허용 가능한 설정으로 제어	이 유형의 필드는 해당 값이 허용되는지 확인하기 위해 설정과 대조됩니다.
전략으로 제어	가치를 생성하는 전략이 있는 항목에서는 다음을 제공합니다. ● 가치를 생성하는 메커니즘 ● 지정된 값이 허용된 값 집합에 속하도록 하는 메커니즘

4.2.5.1. SCC 전략

4.2.5.1.1. RunAsUser

1. *MustRunAs - **runAsUser**를 구성해야 합니다. 구성된 **runAsUser**를 기본값으로 사용합니다. 구성된 **runAsUser**에 대해 검증합니다.*
2. ***MustRunAsRange** - 사전 할당된 값을 사용하지 않는 경우 최솟값과 최댓값을 정의해야 합니다. 최솟값을 기본값으로 사용합니다. 전체 허용 범위에 대해 검증합니다.*

3. ***MustRunAsNonRoot*** - *Pod를 0이 아닌 **runAsUser**를 사용하여 제출하거나 Pod의 이미지에 **USER** 지시문이 정의되어 있어야 합니다. 기본값이 제공되지 않습니다.*

4. ***RunAsAny*** - *기본값이 제공되지 않습니다. 모든 **runAsUser**를 지정할 수 있습니다.*

4.2.5.1.2. SELinuxContext

1. ***MustRunAs*** - *사전 할당된 값을 사용하지 않는 경우 **seLinuxOptions**를 구성해야 합니다. **seLinuxOptions**를 기본값으로 사용합니다. **seLinuxOptions** 에 대해 검증합니다.*

2. ***RunAsAny*** - *기본값이 제공되지 않습니다. 모든 **seLinuxOptions**를 지정할 수 있습니다.*

4.2.5.1.3. SupplementalGroups

1. ***MustRunAs*** - *사전 할당된 값을 사용하지 않는 경우 범위를 하나 이상 지정해야 합니다. 첫 번째 범위의 최솟값을 기본값으로 사용합니다. 모든 범위에 대해 검증합니다.*

2. ***RunAsAny*** - *기본값이 제공되지 않습니다. 임의의 **supplementalGroups**을 지정할 수 있습니다.*

4.2.5.1.4. FSGroup

1. ***MustRunAs*** - *사전 할당된 값을 사용하지 않는 경우 범위를 하나 이상 지정해야 합니다. 첫 번째 범위의 최솟값을 기본값으로 사용합니다. 첫 번째 범위의 첫 번째 ID에 대해 검증합니다.*

2. ***RunAsAny*** - *기본값이 제공되지 않습니다. **fsGroup** ID를 지정할 수 있습니다.*

4.2.5.2. 볼륨 제어

*SCC의 **volumes** 필드를 설정하여 특정 볼륨 유형의 사용을 제어할 수 있습니다. 이 필드에 허용되는 값은 볼륨 생성 시 정의되는 볼륨 소스에 해당합니다.*

- *azureFile*
- *azureDisk*
- *flocker*
- *flexVolume*
- *hostPath*
- *emptyDir*
- *gcePersistentDisk*
- *awsElasticBlockStore*
- *secret*
- *nfs*
- *iscsi*
- *glusterfs*
- *persistentVolumeClaim*

- *rbd*
- *cinder*
- *cephFS*
- *downwardAPI*
- *fc*
- *configMap*
- *vsphereVolume*
- *quobyte*
- *photonPersistentDisk*
- *projected*
- *portworxVolume*
- *scaleIO*
- *storageos*
- *** (모든 볼륨 유형을 사용할 수 있는 특수 값)**
- ***none*** *(모든 볼륨 유형의 사용을 허용하지 않는 특수 값입니다. 이전 버전과의 호환성을 위해서만 존재합니다.)*

새 SCC에 허용되는 볼륨의 최소 권장 집합은 ***configMap***, ***downwardAPI***, ***emptyDir***, ***persistentVolumeClaim***, ***secret***, ***projected*** *입니다.*

참고

OpenShift Container Platform의 각 릴리스마다 새로운 유형이 추가되므로 허용된 볼륨 유형 목록은 포괄적이지 않습니다.

참고

이전 버전과의 호환성을 위해 ***allowHostDirVolumePlugin****을 사용하면* ***volumes*** *필드의 설정을 덮어씁니다. 예를 들어,* ***allowHostDirVolumePlugin****이 false로 설정되어 있지만* ***volumes*** *필드에서 허용되는 경우* ***volumes****에서* ***hostPath*** *값이 제거됩니다.*

4.2.5.3. FlexVolume에 대한 액세스 제한

OpenShift Container Platform은 드라이버를 기반으로 FlexVolume을 추가로 제어합니다. SCC에서 FlexVolumes 사용을 허용하면 Pod에서 FlexVolumes를 요청할 수 있습니다. 그러나 클러스터 관리자가 ***AllowedFlexVolumes*** *필드에서 드라이버 이름을 지정하는 경우 Pod는 이러한 드라이버와 함께 FlexVolume 만 사용해야 합니다.*

두 개의 FlexVolume에만 대한 액세스 제한 예

```
volumes:
- flexVolume
```

```
allowedFlexVolumes:
- driver: example/lvm
- driver: example/cifs
```

4.2.5.4. seccomp

SeccompProfiles 는 Pod 또는 컨테이너의 seccomp 주석에 대해 설정할 수 있는 허용된 프로필을 나열합니다. 설정되지 않은(nil) 또는 빈 값은 포드 또는 컨테이너에서 프로필을 지정하지 않았음을 의미합니다. 와일드카드 * 를 사용하여 모든 프로필을 허용합니다. Pod 값을 생성하는 데 사용하는 경우 와일드카드가 아닌 첫 번째 프로필이 기본값으로 사용됩니다.

사용자 지정 프로필 구성 및 사용에 대한 자세한 내용은 seccomp 설명서 를 참조하십시오.

4.2.5.5. 허용

SCC를 통한 허용 제어를 사용하면 사용자에게 부여된 기능을 기반으로 리소스 생성을 제어할 수 있습니다.

SCC 측면에서는 허용 컨트롤러가 컨텍스트에서 사용 가능한 사용자 정보를 검사하여 적절한 SCC 집합을 검색할 수 있음을 의미합니다. 이 경우 Pod에 운영 환경에 대한 요청을 하거나 Pod에 적용할 일련의 제약 조건을 생성할 수 있는 권한이 부여됩니다.

허용 작업에서 Pod를 승인하는 데 사용하는 SCC 집합은 사용자 ID 및 사용자가 속하는 그룹에 따라 결정됩니다. 또한 Pod에서 서비스 계정을 지정하는 경우, 허용된 SCC 집합에 서비스 계정에 액세스할 수 있는 모든 제약 조건이 포함됩니다.

허용 작업에서는 다음 방법을 사용하여 Pod에 대한 최종 보안 컨텍스트를 생성합니다.

1. 사용 가능한 모든 SCC를 검색합니다.
2. 요청에 지정되지 않은 보안 컨텍스트 설정에 대한 필드 값을 생성합니다.
3. 사용 가능한 제약 조건에 대해 최종 설정을 검증합니다.

일치하는 제약 조건 집합이 있는 경우 Pod가 승인됩니다. 요청을 SCC와 일치시킬 수 없는 경우 Pod가 거부됩니다.

Pod는 SCC에 대해 모든 필드를 검증해야 합니다. 다음은 검증이 필요한 필드 중 단 2개의 예입니다.

참고

이러한 예는 사전 할당된 값을 사용하는 전략과 관련이 있습니다.

MustRunAs의 FSGroup SCC 전략

Pod에서 **fsGroup** ID를 정의하는 경우 해당 ID는 기본 **fsGroup** ID와 같아야 합니다. 그렇지 않으면 해당 SCC에서 Pod를 검증하지 않고 다음 SCC를 평가합니다.

SecurityContextConstraints.fsGroup 필드에 값 **RunAsAny**가 있고 Pod 사양에서 **Pod.spec.securityContext.fsGroup**을 생략하는 경우, 이 필드는 유효한 것으로 간주됩니다. 유효성을 확인하는 동안 다른 SCC 설정에서 다른 Pod 필드를 거부하여 Pod가 실패할 수 있습니다.

MustRunAs의 SupplementalGroups SCC 전략

*Pod 사양에서 하나 이상의 **supplementalGroups** ID를 정의하는 경우, Pod의 ID는 네임스페이스의 **openshift.io/sa.scc.supplemental-groups** 주석에 있는 ID 중 하나와 같아야 합니다. 그렇지 않으면 해당 SCC에서 Pod를 검증하지 않고 다음 SCC를 평가합니다.*

***SecurityContextConstraints.supplementalGroups** 필드에 값 **RunAsAny**가 있고 Pod 사양에서 **Pod.spec.securityContext.supplementalGroups**를 생략하는 경우, 이 필드는 유효한 것으로 간주됩니다. 유효성을 확인하는 동안 다른 SCC 설정에서 다른 Pod 필드를 거부하여 Pod가 실패할 수 있습니다.*

4.2.5.5.1. SCC 우선순위 지정

SCC에는 허용 컨트롤러의 요청을 검증할 때 순서에 영향을 미치는 우선순위 필드가 있습니다. 정렬 시 우선순위가 높은 SCC가 집합의 앞쪽으로 이동합니다. 사용 가능한 SCC 전체 집합이 결정되면 다음 순서에 따라 정렬됩니다.

1. *우선순위가 가장 높은 SCC가 맨 앞에 오고, NIL은 우선순위 0으로 간주됩니다*
2. *우선순위가 동일한 경우, 가장 제한적인 SCC에서 가장 덜 제한적인 SCC 순서대로 정렬됩니다.*
3. *우선순위와 제한이 모두 같은 경우 SCC는 이름순으로 정렬됩니다.*

*기본적으로 클러스터 관리자에게 부여된 anyuid SCC에는 SCC 세트에서 우선 순위가 부여됩니다. 이를 통해 클러스터 관리자는 Pod의 **SecurityContext**에 **RunAsUser**를 지정하지 않고도 모든 사용자로 Pod를 실행할 수 있습니다. 원하는 경우 관리자는 **RunAsUser**를 계속 지정할 수도 있습니다.*

4.2.5.5.2. SCC에 대한 역할 기반 액세스

OpenShift Container Platform 3.11부터 RBAC에서 처리하는 리소스로 SCC를 지정할 수 있습니다. 그러면 SCC에 대한 액세스 권한의 범위를 특정 프로젝트 또는 전체 클러스터로 지정할 수 있습니다. SCC에 사용자, 그룹 또는 서비스 계정을 직접 할당하면 클러스터 전체 범위가 유지됩니다.

역할에 대한 SCC에 대한 액세스를 포함하려면 역할 정의에 다음 규칙을 지정합니다. .Role-Based SCC에 대한 액세스

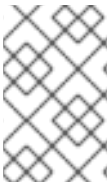
```
rules:
- apiGroups:
  - security.openshift.io (1)
  resources:
  - securitycontextconstraints (2)
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
  resourceNames:
  - myPermittingSCC (3)
```

(1) ***securitycontextconstraints** 리소스를 포함하는 API 그룹*

(2) *사용자가 resource **Names 필드에 SCC 이름을 지정할 수 있는 리소스** 그룹의 이름*

(3) *액세스 권한을 부여하려는 SCC의 이름 예*

*이러한 규칙이 있는 로컬 또는 클러스터 역할을 사용하면 역할 바인딩 또는 clusterrolebinding으로 바인딩된 주체가 **myPermittingSCC** 라는 사용자 정의 SCC를 사용할 수 있습니다.*



참고

*RBAC는 에스컬레이션되지 않도록 설계되었으므로 프로젝트 관리자도 기본적으로 **restricted** SCC를 포함하여 SCC 리소스에 동사 **사용을** 사용할 수 없으므로 SCC에 대한 액세스 권한을 부여할 수 없습니다.*

4.2.5.5.3. 사전 할당된 값 및 보안 컨텍스트 제약 조건 이해

허용 컨트롤러는 보안 컨텍스트 제약 조건의 특정 조건을 인식하여 네임스페이스에서 미리 할당된 값을 조회하고 Pod를 처리하기 전에 보안 컨텍스트 제약 조건을 채울 수 있습니다. 각 SCC 전략은 실행 중인 Pod에 정의된 다양한 ID에 대한 최종 값을 만들기 위해 포드 사양 값으로 집계된 각 정책에 대해 사전 할당된 값(허용 위치)을 사용하여 다른 전략과 독립적으로 평가됩니다.

다음 SCC를 사용하면 Pod 사양에 범위가 정의되지 않은 경우 허용 컨트롤러에서 사전 할당된 값을 찾습니다.

1. *최소 또는 최대 집합이 없는 **MustRunAsRange**의 **RunAsUser** 전략. 허용 작업에서는 범위 필드를 채우기 위해 **openshift.io/sa.scc.uid-range** 주석을 찾습니다.*

2. *수준이 설정되지 않은 **MustRunAs**의 **SELinuxContext** 전략. 허용 작업에서는 수준을 채울 **openshift.io/sa.scc.mcs** 주석을 찾습니다.*

3. ***MustRunAs**의 **FSGroup** 전략. 허용 작업에서는 **openshift.io/sa.scc.supplemental-group** 주석을 찾습니다.*

4. ***MustRunAs**의 **SupplementalGroups** 전략. 허용 작업에서는 **openshift.io/sa.scc.supplemental-group** 주석을 찾습니다.*

생성 단계 중에 보안 컨텍스트 프로바이더는 포드에 구체적으로 설정되지 않은 값을 기본값으로 설정합니다. 기본값은 사용되는 전략을 기반으로 합니다.

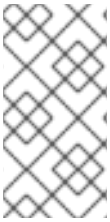
1. ***RunAsAny** 및 **MustRunAsNonRoot** 전략에서는 기본값을 제공하지 않습니다. 따라서 Pod에 정의된 필드(예: 그룹 ID)가 필요한 경우 Pod 사양 내에 이 필드를 정의해야 합니다.*

2. ***MustRunAs** (단일 값) 전략에서는 항상 사용되는 기본값을 제공합니다. 예를 들어 그룹 ID의 경우: Pod 사양에서 자체 ID 값을 정의하더라도 네임스페이스의 기본 필드도 Pod의 그룹에 표시됩니다.*

3. ***MustRunAsRange** 및 **MustRunAs** (범위 기반) 전략에서는 최소 범위 값을 제공합니다. 단일 값 **MustRunAs** 전략과 마찬가지로 네임스페이스의 기본값이 실행 중인 Pod에 표시됩니다. 범위 기반 전략을 여러 범위로 구성할 수 있는 경우 처음 구성된 최소 범위 값을 제공합니다.*

참고

***openshift.io/sa.scc.supplemental-groups** 주석이 네임스페이스에 존재하지 않는 경우, **FSGroup** 및 **SupplementalGroups** 전략이 **openshift.io/sa.scc.uid-range** 주석으로 변경됩니다. 둘 다 존재하지 않으면 SCC를 만들지 못합니다.*



참고

*기본적으로 주석 기반 **FSGroup** 전략은 주석의 최솟값에 따라 단일 범위로 자체 구성됩니다. 예를 들어 주석이 **1/3** 이면 **FSGroup** 전략은 최솟값 및 최대 **1** 로 구성됩니다. **FSGroup** 필드에 더 많은 그룹을 허용하려면 주석을 사용하지 않는 사용자 정의 SCC 를 구성하면 됩니다.*

참고

***openshift.io/sa.scc.supplemental-groups** 주석은 **<start>/<length** 또는 **<start>-<end>** 형식으로 쉼표로 구분된 블록 목록을 허용합니다. **openshift.io/sa.scc.uid-range** 주석에는 단일 블록만 사용할 수 있습니다.*

4.2.6. 인증된 사용자로 수행할 수 있는 항목 확인

OpenShift Container Platform 프로젝트 내에서 모든 네임스페이스 범위 리소스(타사 리소스 포함)에 대해 수행할 수 있는 동사를 확인할 수 있습니다. 다음을 실행합니다.

```
$ oc policy can-i --list --loglevel=8
```

출력은 정보를 수집하기 위해 수행할 API 요청을 결정하는 데 도움이 됩니다.

사용자가 읽을 수 있는 형식으로 정보를 다시 받으려면 다음을 실행합니다.

```
$ oc policy can-i --list
```

출력은 전체 목록을 제공합니다.

특정 동사 를 수행할 수 있는지 확인하려면 다음을 실행합니다.

```
$ oc policy can-i <verb> <resource>
```

사용자 범위는 지정된 범위에 대한 자세한 정보를 제공할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc policy can-i <verb> <resource> --scopes=user:info
```

4.3. 영구 스토리지

4.3.1. 개요

스토리지 관리는 컴퓨팅 리소스 관리와 다릅니다. OpenShift Container Platform 에서는 Kubernetes PV(영구 볼륨) 프레임워크를 사용하여 클러스터 관리자가 클러스터의 영구 스토리지를 프로비저닝할 수 있습니다. 개발자는 PVC(영구 볼륨 클레임)를 사용하여 기본 스토리지 인프라를 구체적으로 잘 몰라도 PV 리소스를 요청할 수 있습니다.

PVC 는 프로젝트에 고유하며 PV 를 사용하는 수단으로 개발자가 생성 및 사용합니다. 자체 PV 리소스는 단일 프로젝트로 범위가 지정되지 않으며, 전체 OpenShift Container Platform 클러스터에서 공유되고 모든 프로젝트에서 요청할 수 있습니다. PV 가 PVC 에 바인딩된 후에는 해당 PV 가 추가 PVC 에 바인딩될 수 없습니다. 이는 바인딩 프로젝트(바인딩 프로젝트)에 바인딩된 PV 의 범위를 단일 네임스페이스 로 지정하는 효과가 있습니다.

*PV 는 a **PersistentVolume** API 오브젝트로 정의되며, 이는 클러스터 관리자가 프로비저닝한 클러스터에서 기존 네트워크 스토리지 부분을 나타냅니다. 그리고 노드가 클러스터 리소스인 것과 마찬가지로 클러*

스터의 리소스입니다. PV는 볼륨과 같은 **볼륨** 플러그인이지만 PV를 사용하는 개별 Pod 와 관련된 라이프사이클이 있습니다. PV 오브젝트는 NFS, iSCSI 또는 클라우드 공급자별 스토리지 시스템에서 스토리지 구현의 세부 정보를 캡처합니다.

중요

인프라의 스토리지의 고가용성은 기본 스토리지 공급자가 담당합니다.

PVC는 **PersistentVolumeClaim** API 오브젝트에 의해 정의되며, 개발자의 스토리지 요청을 나타냅니다. Pod는 노드 리소스를 사용하고 PVC는 PV 리소스를 사용하는 점에서 Pod와 유사합니다. 예를 들어 Pod는 특정 수준의 리소스(예: CPU 및 메모리)를 요청할 수 있지만 PVC는 특정 스토리지 용량 및 액세스 모드(예: 읽기/쓰기 또는 여러 번 읽기 전용)를 요청할 수 있습니다.

4.3.2. 볼륨 및 클레임의 라이프사이클

PV는 클러스터의 리소스입니다. PVC는 그러한 리소스에 대한 요청이며, 리소스에 대한 클레임을 검사하는 역할을 합니다. PV와 PVC 간의 상호 작용에는 다음과 같은 라이프사이클이 있습니다.

4.3.2.1. 스토리지 프로비저닝

PVC에 정의된 개발자의 요청에 대한 응답으로 클러스터 관리자는 스토리지 및 일치하는 PV를 프로비저닝하는 하나 이상의 동적 프로비저너를 구성합니다.

다른 방법으로 클러스터 관리자는 사용할 수 있는 실제 스토리지의 세부 정보를 전달하는 여러 PV를 사전에 생성할 수 있습니다. PV는 API에 위치하며 사용할 수 있습니다.

4.3.2.2. 클레임 바인딩

PVC를 생성할 때 스토리지의 특정 용량을 요청하고, 필요한 액세스 모드를 지정하며, 스토리지를 설명 및 분류하는 스토리지 클래스를 만듭니다. 마스터의 제어 루프는 새 PVC를 감시하고 새 PVC를 적절한 PV에 바인딩합니다. 적절한 PV가 없으면 스토리지 클래스를 위한 프로비저너가 PV를 1개 생성합니다.

PV 볼륨이 요청된 볼륨을 초과할 수 있습니다. 이는 특히 수동으로 프로비저닝된 PV의 경우 더욱 그러합니다. 초과를 최소화하기 위해 OpenShift Container Platform은 기타 모든 조건과 일치하는 최소 PV로 바인딩됩니다.

일치하는 볼륨이 없거나 스토리지 클래스에 서비스를 제공하는 사용 가능한 프로비저너로 생성할 수 없는 경우 클레임은 무제한으로 유지됩니다. 일치하는 볼륨을 사용할 수 있을 때 클레임이 바인딩됩니다. 예를 들어, 수동으로 프로비저닝된 50Gi 볼륨이 있는 클러스터는 100Gi 요청하는 PVC와 일치하지 않습니다. 100Gi PV가 클러스터에 추가되면 PVC를 바인딩할 수 있습니다.

4.3.2.3. Pod 및 클레임된 PV 사용

Pod는 클레임을 볼륨으로 사용합니다. 클러스터는 클레임을 검사하여 바인딩된 볼륨을 찾고 Pod에 해당 볼륨을 마운트합니다. 여러 액세스 모드를 지원하는 그러한 볼륨의 경우 Pod에서 클레임을 볼륨으로 사용할 때 적용되는 모드를 지정해야 합니다.

클레임이 있고 해당 클레임이 바인딩되면 바인딩된 PV는 필요한 동안 사용자에게 속합니다. Pod의 볼륨 블록에 **persistentVolumeClaim**을 포함하여 Pod를 예약하고 클레임된 PV에 액세스할 수 있습니다. 자세한 구문 정보는 아래를 참조하십시오.

4.3.2.4. PVC 보호

PVC 보호는 기본적으로 활성화되어 있습니다.

4.3.2.5. 릴리스 볼륨

볼륨 사용 작업이 끝나면 API에서 PVC 오브젝트를 삭제하여 리소스를 회수할 수 있습니다. 클레임이 삭제되면 볼륨은 "릴리스"로 간주되지만 다른 클레임에서는 아직 사용할 수 없습니다. 이전 클레임의 데이터는 볼륨에 남아 있으며 정책에 따라 처리되어야 합니다.

4.3.2.6. 볼륨 회수

a **PersistentVolume** 의 회수 정책은 해제된 볼륨으로 수행할 작업을 클러스터에 지시합니다. PV의 회수 정책은 **Retain** 또는 **Delete** 일 수 있습니다.

- **Retain** 회수 정책을 사용하면 이를 지원하는 해당 볼륨 플러그인에 대한 리소스를 수동으로 회수할 수 있습니다.
- **Delete** 회수 정책은 OpenShift Container Platform에서 **PersistentVolume** 오브젝트 및 AWS EBS, GCE PD 또는 Cinder 볼륨과 같은 외부 인프라의 관련 스토리지 자산을 모두 삭제합니다.

참고

동적으로 프로비저닝된 볼륨의 기본 **ReclaimPolicy** 값은 **Delete** 입니다. 수동으로 프로비저닝된 볼륨의 기본 **ReclaimPolicy** 값은 **Retain** 입니다.

4.3.2.7. 수동으로 PersistentVolume 회수

PersistentVolumeClaim이 삭제되면 PersistentVolume이 계속 존재하며 "릴리스됨"으로 간주됩니다. 그러나 이전 클레임의 데이터가 볼륨에 남아 있으므로 다른 클레임에서 PV를 아직 사용할 수 없습니다.

클러스터 관리자로 PV를 수동으로 회수하려면 다음을 수행합니다.

1. PV를 삭제합니다.

```
$ oc delete <pv-name>
```

AWS EBS, GCE PD, Azure Disk 또는 Cinder 볼륨과 같은 외부 인프라의 연결된 스토리지 자산은 PV가 삭제된 후에도 계속 존재합니다.

2. 연결된 스토리지 자산에서 데이터를 정리합니다.
3. 연결된 스토리지 자산을 삭제합니다. 대안으로, 동일한 스토리지 자산을 재사용하려면, 스토리지 자산 정의를 사용하여 새 PV를 생성합니다.

이제 회수된 PV를 다른 PVC에서 사용할 수 있습니다.

4.3.2.8. 회수 정책 변경

PV의 회수 정책을 변경하려면 다음을 수행합니다.

1. 클러스터의 PV를 나열합니다.

```
$ oc get pv
```

출력 예

```
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM         STORAGECLASS    REASON    AGE
```

```
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim1    manual                  10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim2    manual                  6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim3    manual                  3s
```

2. *PV 중 하나를 선택하고 회수 정책을 변경합니다.*

```
$ oc patch pv <your-pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

3. *선택한 PV에 올바른 정책이 있는지 확인합니다.*

```
$ oc get pv
```

출력 예

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM             STORAGECLASS     REASON    AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim1    manual                  10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim2    manual                  6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Retain          Bound
default/claim3    manual                  3s
```

*이전 출력에서 **default/claim3 클레임에** 바인딩된 PV에 이제 **Retain** 회수 정책이 적용됩니다. 사용자가 **default/claim3** 클레임을 삭제하면 PV가 자동으로 삭제되지 않습니다.*

4.3.3. PV(영구 볼륨)

*각 PV에는 **사양** 및 **상태**가 포함됩니다. 이는 볼륨의 사양과 상태이고 예는 다음과 같습니다.*

PV 오브젝트 정의 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /tmp
    server: 172.17.0.2
```

4.3.3.1. PV 유형

*OpenShift Container Platform에서는 다음 **PersistentVolume** 플러그인을 지원합니다.*

- NFS
- hostPath
- GlusterFS
- gluster-block
- OpenShift Container Storage (OCS) 파일
- OpenShift Container Storage (OCS) 블록
- Ceph RBD
- OpenStack Cinder
- AWS Elastic Block Store (EBS)
- GCE 영구 디스크
- iSCSI
- 파이버 채널
- Azure Disk
- Azure File
- VMWare vSphere
- 로컬

4.3.3.2. 용량

일반적으로 PV에는 특정 스토리지 용량이 있습니다. PV의 **용량** 특성을 사용하여 설정됩니다.

현재는 스토리지 용량이 설정 또는 요청할 수 있는 유일한 리소스뿐입니다. 향후 속성에는 IOPS, 처리량 등이 포함될 수 있습니다.

4.3.3.3. 액세스 모드

A **PersistentVolume** 은 리소스 프로바이더가 지원하는 방식으로 호스트에 마운트할 수 있습니다. 프로바이더에는 다양한 기능이 있으며 각 PV의 액세스 모드는 해당 볼륨에서 지원하는 특정 모드로 설정됩니다. 예를 들어 NFS에서는 여러 읽기/쓰기 클라이언트를 지원할 수 있지만 특정 NFS PV는 서버에서 읽기 전용으로 내보낼 수 있습니다. 각 PV는 특정 PV의 기능을 설명하는 자체 액세스 모드 세트를 가져옵니다.

클레임은 액세스 모드가 유사한 볼륨과 매칭됩니다. 유일하게 일치하는 두 가지 기준은 액세스 모드와 크기입니다. 클레임의 액세스 모드는 요청을 나타냅니다. 따라서 더 많이 부여될 수 있지만 절대로 부족하게는 부여되지 않습니다. 예를 들어, 클레임 요청이 RWO이지만 사용 가능한 유일한 볼륨이 NFS PV(RWO+ROX+RWX)인 경우, RWO를 지원하므로 클레임이 NFS와 일치하게 됩니다.

항상 직접 일치가 먼저 시도됩니다. 볼륨의 모드는 사용자의 요청과 일치하거나 더 많은 모드를 포함해야 합니다. 크기는 예상되는 크기보다 크거나 같아야 합니다. 두 가지 유형의 볼륨(예: NFS 및 iSCSI)에 동일한 액세스 모드 집합이 있는 경우 둘 중 하나를 해당 모드와 클레임과 일치시킬 수 있습니다. 볼륨 유형과 특정 유형을 선택할 수 있는 순서는 없습니다.

모드가 같은 모든 볼륨이 그룹화된 다음 크기별로 정렬됩니다(최소 볼륨에서 최대 볼륨 순으로). 바인더는 모드가 일치하는 그룹을 가져오고 크기가 일치할 때까지(크기 순서로) 각 그룹에 대해 반복합니다.

다음 표에는 액세스 모드가 나열되어 있습니다.

표 4.1. 액세스 모드

액세스 모드	CLI 약어	설명
ReadWriteOnce	**RWO**	볼륨은 단일 노드에서 읽기-쓰기로 마운트할 수 있습니다.
ReadOnlyMany	**ROX**	볼륨은 여러 노드에서 읽기 전용으로 마운트할 수 있습니다.
ReadWriteMany	**RWX**	볼륨은 여러 노드에서 읽기-쓰기로 마운트할 수 있습니다.

중요

*볼륨의 **AccessModes** 는 볼륨의 기능에 대한 설명자입니다. 제한 조건이 적용되지 않습니다. 리소스를 잘못된 사용으로 인한 런타임 오류는 스토리지 공급자가 처리합니다.*

*예를 들어 Ceph는 **ReadWriteOnce** 액세스 모드를 제공합니다. 볼륨의 ROX 기능을 사용하려면 클레임을 **읽기 전용**으로 표시해야 합니다. 공급자에서의 오류는 런타임 시 마운트 오류로 표시됩니다.*

iSCSI 및 파이버 채널 볼륨은 현재 펜싱 메커니즘을 지원하지 않습니다. 볼륨을 한 번에 하나씩만 사용하는지 확인해야 합니다. 노드 드레이닝과 같은 특정 상황에서는 두 개의 노드에서 볼륨을 동시에 사용할 수 있습니다. 노드를 드레인하기 전에 먼저 이러한 볼륨을 사용하는 Pod가 삭제되었는지 확인합니다.

다음 표에는 다른 PV에서 지원하는 액세스 모드가 나열되어 있습니다.

표 4.2. PV에서 지원되는 액세스 모드

볼륨 플러그인	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWS EBS	▮	-	-
Azure File	▮	▮	▮
Azure Disk	▮	-	-
Ceph RBD	▮	▮	-
파이버 채널	▮	▮	-
GCE 영구 디스크	▮	-	-
GlusterFS	▮	▮	▮
gluster-block	▮	-	-

볼륨 플러그인	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
HostPath	▮	-	-
iSCSI	▮	▮	-
NFS	▮	▮	▮
OpenStack Cinder	▮	-	-
VMWare vSphere	▮	-	-
로컬	▮	-	-

참고

AWS EBS, GCE 영구 디스크 또는 Openstack Cinder PV를 사용하는 Pod에 재생성 배포 전략을 사용합니다.

4.3.3.4. 회수 정책

다음 표에는 현재 회수 정책이 나열되어 있습니다.

표 4.3. 현재 회수 정책

회수 정책	설명
유지	수동 재요청 허용.
delete	PV 및 연결된 외부 스토리지 자산을 모두 삭제합니다.

주의

모든 Pod를 유지하지 않으려면 동적 프로비저닝을 사용합니다.

4.3.3.5. 단계

볼륨은 다음 단계 중 하나에서 찾을 수 있습니다.

표 4.4. 볼륨 단계

단계	설명
Available	아직 클레임에 바인딩되지 않은 여유 리소스입니다.

단계	설명
Bound	볼륨이 클레임에 바인딩됩니다.
해제됨	클레임이 삭제되었지만, 리소스가 아직 클러스터에 의해 회수되지 않았습니다.
실패	볼륨에서 자동 회수가 실패했습니다.

CLI에는 PV에 바인딩된 PVC의 이름이 표시됩니다.

4.3.3.6. 마운트 옵션

volume. ***beta.kubernetes.io/mount-options*** *주석을 사용하여 영구 볼륨을 마운트하는 동안 마운트 옵션을 지정할 수 있습니다.*

예를 들면 다음과 같습니다.

마운트 옵션 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec 1
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

1 *지정된 마운트 옵션이 PV를 디스크에 마운트하는 동안 사용됩니다.*

다음 영구 볼륨 유형은 마운트 옵션을 지원합니다.

- *NFS*
- *GlusterFS*
- *Ceph RBD*
- *OpenStack Cinder*
- *AWS Elastic Block Store (EBS)*

- GCE 영구 디스크
- iSCSI
- Azure Disk
- Azure File
- VMWare vSphere

참고

파이버 채널 및 HostPath 영구 볼륨은 마운트 옵션을 지원하지 않습니다.

4.3.3.7. 재귀 chown

PV가 Pod에 마운트되거나 Pod 재시작이 발생하면 해당 Pod와 일치하도록 볼륨의 소유권 및 권한이 재귀적으로 변경됩니다. PV의 모든 파일 및 디렉터리에서 **chown** 을 수행할 때 볼륨에 읽기/쓰기 액세스 권한이 추가됩니다. 이를 통해 Pod 내부에서 실행되는 프로세스가 PV 파일 시스템에 액세스할 수 있습니다. 사용자는 Pod 및 SCC(보안 컨텍스트 제약 조건)에 **fsGroup** 을 지정하지 않으면 소유권에 대한 재귀적 변경을 방지할 수 있습니다.

SELinux 레이블 도 반복적으로 변경됩니다. 사용자는 SELinux 레이블을 재귀적으로 변경하지 못하게 할 수 없습니다.

참고

사용자에게 많은 수의 파일(예: > 100,000)이 있는 경우 PV를 Pod에 마운트하기 전에 상당한 지연이 발생할 수 있습니다.

4.3.4. 영구 볼륨 클레임

각 PVC에는 **사양** 및 **상태가** 포함됩니다. 이는 클레임의 사양 및 상태입니다. 예를 들면 다음과 같습니다.

PVC 오브젝트 정의 예

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: gold
```

4.3.4.1. 스토리지 클래스

선택 사항으로 클레임은 **storageClassName** 속성에 스토리지 클래스의 이름을 지정하여 특정 스토리지 클래스를 요청할 수 있습니다. PVC와 **storageClassName**이 동일하고 요청된 클래스의 PV만 PVC에 바인딩할 수 있습니다. 클러스터 관리자는 동적 프로비저너를 구성하여 하나 이상의 스토리지 클래스에 서비스를 제공할 수 있습니다. 클러스터 관리자는 PVC의 사양과 일치하는 PV를 생성할 수 있습니다.

클러스터 관리자는 모든 PVC의 기본 스토리지 클래스도 설정할 수도 있습니다. 기본 스토리지 클래스가 구성된 경우 PVC는 ""로 설정된 **StorageClass** 또는 **storageClassName** 주석이 스토리지 클래스를 제외하고 PV에 바인딩되도록 명시적으로 요청해야 합니다.

4.3.4.2. 액세스 모드

클레임은 특정 액세스 모드로 스토리지를 요청할 때 볼륨과 동일한 규칙을 사용합니다.

4.3.4.3. 리소스

Pod와 같은 클레임은 특정 리소스 수량을 요청할 수 있습니다. 이 경우 요청은 스토리지에 대한 요청입니다. 동일한 리소스 모델이 볼륨 및 클레임에 적용됩니다.

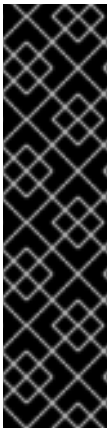
4.3.4.4. 클레임을 볼륨으로

클레임을 볼륨으로 사용하여 Pod 액세스 스토리지 클레임을 사용하는 경우 클레임은 Pod와 동일한 네임스페이스에 있어야 합니다. 클러스터는 Pod의 네임스페이스에서 클레임을 검색하고 이를 사용하여 클레임을 지원하는 **PersistentVolume**을 가져옵니다. 볼륨은 호스트에 마운트되며, 예를 들면 다음과 같습니다.

호스트 및 Pod에 볼륨 마운트 예

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

4.3.5. 블록 볼륨 지원



중요

블록 볼륨 지원은 기술 프리뷰 기능이며 수동으로 프로비저닝된 PV에서만 사용할 수 있습니다.

기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원하지 않으며, 기능상 완전하지 않을 수 있어 프로덕션에 사용하지 않는 것이 좋습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능 지원 범위에 대한 자세한 내용은 https://access.redhat.com/support/offerings/techpreview/ 을 참조하십시오.

PV 및 PVC 사양에 API 필드를 포함하여 원시 블록 볼륨을 정적으로 프로비저닝할 수 있습니다.

*블록 볼륨을 사용하려면 먼저 **BlockVolume** 기능 게이트를 활성화해야 합니다. 마스터의 기능 게이트를 활성화하려면 **apiServerArguments 및 controllerArgument s에 feature-gate s 를 추가합니다.** 노드의 기능 게이트를 활성화하려면 **kubeletArguments에 feature-gates 를 추가합니다.** 예를 들면 다음과 같습니다.*

```
kubeletArguments:
  feature-gates:
    - BlockVolume=true
```

PV 예

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block (1)
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

(1) *이 PV가 원시 블록 볼륨임을 나타내는 **volumeMode** 필드.*

PVC 예

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block (1)
  resources:
    requests:
      storage: 10Gi
```

(1) *원시 블록 영구 볼륨이 요청되었음을 나타내는 **volumeMode** 필드.*

Pod 사양 예

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices:  1
        - name: data
          devicePath: /dev/xvda  2
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc  3
```

1 ***volumeDevices*** *(volume* ***Mounts****와 유사함)는 블록 장치에 사용되며* ***PersistentVolumeClaim*** *소스에서만 사용할 수 있습니다.*

2 ***DevicePath*** *(mount* ***Path****와 유사함)는 물리적 장치의 경로를 나타냅니다.*

3 *볼륨 소스는* ***persistentVolumeClaim*** *유형이어야 하며 예상되는 PVC의 이름과 일치해야 합니다.*

표 4.5. ***VolumeMode****에 대해 허용되는 값*

현재의	기본
파일 시스템	예
블록	아니요

표 4.6. 블록 볼륨에 대한 바인딩 시나리오

PV VolumeMode	PVC VolumeMode	바인딩 결과
파일 시스템	파일 시스템	바인딩
지정되지 않음	지정되지 않음	바인딩
파일 시스템	지정되지 않음	바인딩
지정되지 않음	파일 시스템	바인딩
블록	블록	바인딩
지정되지 않음	블록	바인딩되지 않음
블록	지정되지 않음	바인딩되지 않음

PV VolumeMode	PVC VolumeMode	바인딩 결과
파일 시스템	블록	바인딩되지 않음
블록	파일 시스템	바인딩되지 않음

중요

*값을 지정하지 않으면 **Filesystem**의 기본값이 사용됩니다.*

4.4. 임시 로컬 스토리지

4.4.1. 개요

참고

이 항목은 임시 스토리지 기술 프리뷰가 활성화된 경우에만 적용됩니다. 이 기능은 기본적으로 비활성화되어 있습니다. 활성화된 경우 OpenShift Container Platform 클러스터는 임시 스토리지를 사용하여 클러스터를 삭제한 후 유지하지 않아도 되는 정보를 저장합니다. 이 기능을 활성화하려면 임시 스토리지 구성을 참조하십시오.

영구 스토리지 외에도 포드 및 컨테이너에는 작업을 위해 임시 또는 임시 로컬 스토리지가 필요할 수 있습니다. 이러한 임시 스토리지의 수명은 개별 Pod의 수명 이상으로 연장되지 않으며 이 임시 스토리지는 여러 Pod 사이에서 공유할 수 없습니다.

OpenShift Container Platform 3.10 이전에는 임시 로컬 스토리지가 컨테이너의 쓰기 가능한 계층, 로그 디렉터리, EmptyDir 볼륨을 사용하여 Pod에 노출되었습니다. Pod는 스크래치 공간, 캐싱 및 로그를 위해 임시 로컬 스토리지를 사용합니다. 로컬 스토리지 회계 및 격리 부족과 관련한 문제는 다음과 같습니다.

- *Pod는 사용할 수 있는 로컬 스토리지의 용량을 알 수 없습니다.*
- *Pod는 보장되는 로컬 스토리지를 요청할 수 없습니다.*
- *로컬 스토리지는 최상의 노력 리소스입니다.*
- *Pod는 로컬 스토리지를 채우는 다른 Pod로 인해 제거할 수 있으며 충분한 스토리지를 회수할 때까지 새 Pod가 허용되지 않습니다.*

영구 볼륨과 달리 임시 스토리지는 비정형 및 공유되지만 시스템, 컨테이너 런타임, OpenShift Container Platform의 다른 용도 외에도 노드에서 실행되는 모든 포드 간에 실제 데이터가 아닌 공간을 구성합니다. 임시 스토리지 프레임워크를 사용하면 Pod에서 임시 로컬 스토리지 요구 사항을 지정하고 OpenShift Container Platform에서 적절한 Pod를 예약하고 로컬 스토리지를 과도하게 사용하지 않도록 노드를 보호할 수 있습니다.

임시 스토리지 프레임워크를 사용하면 관리자와 개발자가 이 로컬 스토리지를 더욱 더 쉽게 관리할 수 있지만 I/O 처리량 및 대기 시간과 관련해서는 보장을 하지 않습니다.

4.4.2. 임시 스토리지 유형

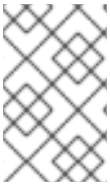
임시 로컬 스토리지는 항상 기본 파티션에서 사용할 수 있습니다. 기본 파티션을 생성하는 두 가지 기본 방법, 루트 및 런타임이 있습니다.

4.4.2.1. 루트

이 파티션에는 기본적으로 kubelet의 루트 디렉터리인 **/var/lib/origin/** 및 **/var/log/** 디렉터리가 있습니다. 이 파티션은 사용자 포드, OS 및 Kubernetes 시스템 데몬 간에 공유할 수 있습니다. 이 파티션은 EmptyDir 볼륨, 컨테이너 로그, 이미지 계층 및 쓰기 가능한 컨테이너를 통해 Pod에서 사용할 수 있습니다. kubelet은 이 파티션의 공유 액세스 및 격리를 관리합니다. 이 파티션은 임시이며 애플리케이션은 이 파티션에서 성능 SLA, 디스크 IOPS를 기대할 수 없습니다.

4.4.2.2. 런타임

런타임에서 오버레이 파일 시스템에 사용할 수 있는 선택적 파티션입니다. OpenShift Container Platform에서는 이 파티션에 대한 격리와 함께 공유 액세스를 식별하고 제공합니다. 컨테이너 이미지 계층 및 쓰기 가능한 계층이 여기에 저장됩니다. 런타임 파티션이 있는 경우 **루트** 파티션은 이미지 계층 또는 기타 쓰기 가능한 스토리지를 유지하지 않습니다.



참고

DeviceMapper를 사용하여 런타임 스토리지를 제공할 때 컨테이너의 COW(Copy-On-Write) 계층은 임시 스토리지 관리에서 고려되지 않습니다. 오버레이 스토리지를 사용하여 이 임시 스토리지를 모니터링합니다.

4.4.3. 임시 스토리지 관리

클러스터 관리자는 터미널이 아닌 상태에서 모든 Pod에서 임시 스토리지에 대한 제한 범위 및 요청 수를 정의하는 할당량을 설정하여 프로젝트 내에서 임시 스토리지를 관리할 수 있습니다. 개발자는 Pod 및 Container 수준에서 이 컴퓨팅 리소스에 대한 요청 및 제한을 설정 할 수도 있습니다.

4.4.4. 임시 스토리지 모니터링

/bin/df 를 임시 컨테이너 데이터가 있는 볼륨에서 임시 스토리지 사용을 모니터링하는 도구로, /var/lib/**origin 및 /var/lib/ docker** 를 사용할 수 있습니다. 클러스터 관리자가 **/var/lib/ docker를 별도의 디스크에 배치하는 경우 df 명령을 사용할 때 /var/lib/ origin** 에만 사용 가능한 공간이 표시됩니다.

df -h 명령을 사용하여 **/var/lib** 에서 사용된 공간 및 사용 가능한 공간의 사람이 읽을 수 있는 값을 표시합니다.

```
$ df -h /var/lib
```

출력 예

```
Filesystem  Size  Used Avail Use% Mounted on
/dev/sda1    69G   32G   34G  49% /
```

4.5. 소스 제어 관리

OpenShift Container Platform은 내부(예: 사내 Git 서버) 또는 외부(예: GitHub,Bitbucket 등) 호스팅된 기존 소스 제어 관리(SCM) 시스템을 활용합니다. 현재 OpenShift Container Platform은 Git 솔루션만 지원합니다.

SCM 통합은 빌드 와 긴밀하게 결합되며, 다음 두 가지 사항은 다음과 같습니다.

- OpenShift Container Platform 내부에서 애플리케이션을 빌드할 수 있는 리포지토리를 사용하여 **BuildConfig** 를 생성합니다. 리포지토리를 검사하여 수동으로 **BuildConfig**를 생성하거나 OpenShift Container Platform에서 자동으로 생성하도록 할 수 있습니다.
- 리포지토리 변경 시 빌드 트리거.

4.6. 승인 컨트롤러

4.6.1. 개요

승인 제어 플러그인은 리소스가 유지되기 전에 마스터 API에 대한 요청을 가로채지만 요청이 인증 및 승인된 후입니다.

각 승인 제어 플러그인은 클러스터에 요청을 수락하기 전에 순서대로 실행됩니다. 시퀀스의 플러그인이 요청을 거부하면 전체 요청이 즉시 거부되고 최종 사용자에게 오류가 반환됩니다.

승인 제어 플러그인은 경우에 따라 시스템 구성 기본값을 적용하도록 들어오는 오브젝트를 수정할 수 있습니다. 또한 승인 제어 플러그인은 할당량 사용량 증가와 같은 작업을 수행하기 위해 요청 처리의 일부로 관련 리소스를 수정할 수 있습니다.

주의

OpenShift Container Platform 마스터에는 각 리소스 유형(Kubernetes 및 OpenShift Container Platform)에 대해 기본적으로 활성화된 기본 플러그인 목록이 있습니다. 마스터가 올바르게 작동하는 데 필요합니다. 작업을 엄격하게 알고 있지 않으면 이 목록을 수정하는 것은 권장되지 않습니다. 향후 버전의 제품에서는 다른 플러그인 세트를 사용할 수 있으며 순서를 변경할 수 있습니다. 마스터 구성 파일의 기본 플러그인 목록을 재정의하는 경우 최신 버전의 OpenShift Container Platform 마스터 요구 사항을 반영하도록 업데이트해야 합니다.

4.6.2. 일반 승인 규칙

OpenShift Container Platform은 Kubernetes 및 OpenShift Container Platform 리소스에 단일 승인 체인을 사용합니다. 즉, 최상위 **admissionConfig.pluginConfig.pluginConfig** 요소에는 **kubernetesMasterConfig.admissionConfig.pluginConfig** 에 포함된 데 사용되는 승인 플러그인 구성이 포함될 수 있습니다.

kubernetesMasterConfig.admissionConfig.pluginConfig 를 이동하여 **admissionConfig.pluginConfig** 로 병합해야 합니다.

지원되는 모든 승인 플러그인은 단일 체인으로 정렬됩니다. **admissionConfig.pluginOrderOverride** 또는 **kubernetesMasterConfig.admissionConfig.pluginOrderOverride** 를 설정하지 않습니다. 대신 플러그인 특정 구성을 추가하거나 다음과 같이 **DefaultAdmissionConfig** 스탠자를 추가하여 기본적으로 꺼진 플러그인을 활성화합니다.

```
admissionConfig:
  pluginConfig:
    AlwaysPullImages: 1
      configuration:
```

```
kind: DefaultAdmissionConfig
apiVersion: v1
disable: false (2)
```

(1) 승인 플러그인 이름.

(2) 플러그인을 활성화해야 함을 나타냅니다. 이는 선택 사항이며 참조용으로만 여기에 표시됩니다.

disable를 true 로 설정하면 기본값이 on인 승인 플러그인을 비활성화합니다.

주의

승인 플러그인은 일반적으로 API 서버에 대한 보안을 적용하는 데 사용됩니다. 비활성화할 때는 주의하십시오.

참고

*이전에 단일 승인 체인에 안전하게 결합할 수 없는 **admissionConfig** 요소를 사용 중인 경우 API 서버 로그에 경고가 표시되고 API 서버는 기존 호환성을 위해 두 개의 별도의 승인 체인으로 시작합니다. **admissionConfig** 를 업데이트하여 경고를 해결합니다.*

4.6.3. 사용자 정의 가능한 승인 플러그인

클러스터 관리자는 다음과 같은 특정 동작을 제어하도록 일부 승인 제어 플러그인을 구성할 수 있습니다.

- *사용자당 자체 프로비저닝 프로젝트 수 제한*
- *글로벌 빌드 기본값 및 덮어쓰기 구성*
- *Pod 배치 제어*
- *역할 바인딩 관리*

4.6.4. 컨테이너를 사용하는 승인 컨트롤러

컨테이너를 사용하는 승인 컨트롤러도 init 컨테이너를 지원합니다.

4.7. 사용자 정의 ADMISSION CONTROLLERS

4.7.1. 개요

기본 승인 컨트롤러 외에도 승인 체인의 일부로 승인 Webhook 를 사용할 수 있습니다.

승인 Webhook는 레이블을 삽입하기 위해 생성할 때 Pod를 변경하거나 승인 프로세스 중 Pod 구성의 특정 측면을 검증하도록 웹 후크 서버를 호출합니다.

승인 Webhook는 리소스를 지속성하기 전에 마스터 API로 요청을 가로채지만 요청이 인증 및 승인된 후입니다.

4.7.2. Admission Webhooks

OpenShift Container Platform에서는 API 승인 체인 중에 웹 후크 서버를 호출하는 승인 Webhook 오브젝트를 사용할 수 있습니다.

다음 두 가지 유형의 승인 Webhook 오브젝트를 구성할 수 있습니다.

- 승인 Webhook 변경을 사용하면 리소스 콘텐츠가 지속되기 전에 변경 웹 후크를 사용하여 리소스 콘텐츠를 수정할 수 있습니다.
- 승인 Webhook를 검증 하면 웹 후크를 검증하여 사용자 정의 승인 정책을 적용할 수 있습니다.

웹 후크 및 외부 웹 후크 서버를 구성하는 것은 이 문서의 범위를 벗어납니다. 그러나 OpenShift Container Platform에서 제대로 작동하려면 Webhook가 인터페이스를 준수해야 합니다.

중요

승인 Webhook 는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원하지 않으며, 기능상 완전하지 않을 수 있어 프로덕션에 사용하지 않는 것이 좋습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능 지원 범위에 대한 자세한 내용은 https://access.redhat.com/support/offerings/techpreview/를 참조하십시오.

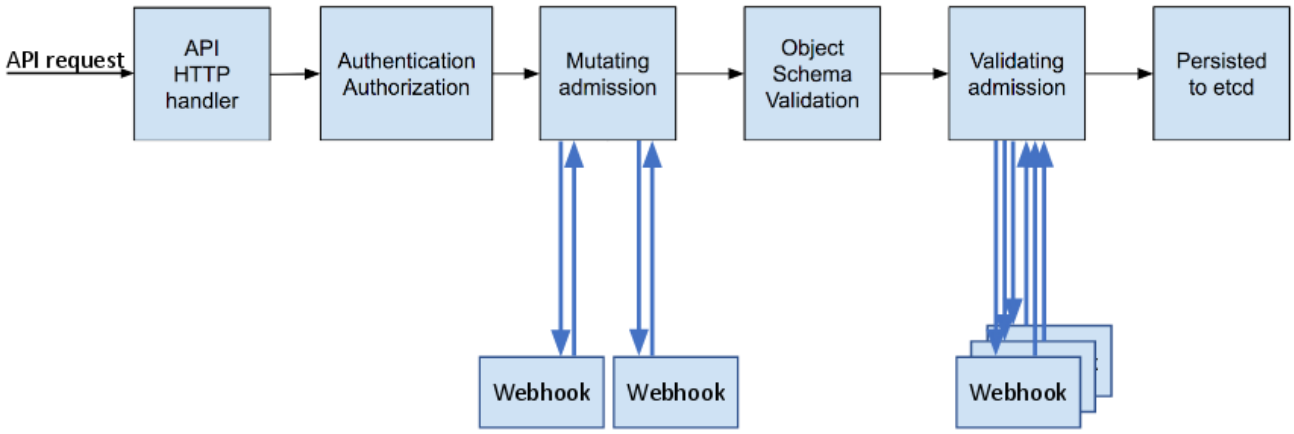
오브젝트가 인스턴스화되면 OpenShift Container Platform은 API 호출을 수행하여 오브젝트를 허용합니다. 승인 프로세스 중에 승인 컨트롤러는 Webhook 를 호출하여 선호도 레이블 삽입과 같은 작업을 수행할 수 있습니다. 승인 프로세스가 끝나면 검증 승인 컨트롤러에서 Webhook를 호출하여 선호도 레이블 확인과 같은 오브젝트가 올바르게 구성되었는지 확인할 수 있습니다. 검증이 통과되면 OpenShift Container Platform에서 구성된 대로 오브젝트를 스케줄링합니다.

API 요청이 도착하면 변경 또는 검증 승인 컨트롤러는 구성의 외부 Webhook 목록을 사용하여 병렬로 호출합니다.

- **모든** 웹 후크에서 요청을 승인하면 승인 체인이 계속됩니다.
- 웹 후크 중 하나 **라도** 요청을 거부하면 승인 요청이 거부되고 이를 수행하는 이유는 첫 번째 웹 후크 거부 이유를 기반으로 합니다.
  둘 이상의 웹 후크가 승인 요청을 거부하면 첫 번째 웹 후크만 사용자에게 반환됩니다.
- Webhook를 호출할 때 오류가 발생하면 해당 요청이 거부되거나 웹 후크가 무시됩니다.

승인 컨트롤러와 웹 후크 서버 간의 통신은 TLS를 사용하여 보안을 설정해야 합니다. CA 인증서를 생성하고 인증서를 사용하여 웹 후크 서버에서 사용하는 서버 인증서에 서명합니다. PEM 형식의 CA 인증서는 Service Serving 인증서 비밀 과 같은 메커니즘을 사용하여 승인 컨트롤러에 제공됩니다.

다음 다이어그램에서는 여러 웹 후크를 호출하는 두 개의 승인 Webhook가 있는 이 프로세스를 보여줍니다.



승인 Webhook의 간단한 사용 사례는 리소스의 구문 검증입니다. 예를 들어 모든 Pod에 공통 레이블 세트가 있어야 하는 인프라가 있으며, Pod에 해당 라벨이 없는 경우 Pod가 유지되지 않도록 합니다. 이러한 라벨과 다른 웹 후크를 삽입하도록 웹 후크를 작성하여 레이블이 있는지 확인할 수 있습니다. 그런 다음 OpenShift Container Platform은 라벨이 있는 Pod를 예약하고 유효성 검사를 통과하며 누락된 라벨으로 인해 전달되지 않는 Pod를 거부합니다.

일반적인 사용 사례는 다음과 같습니다.

- *사이드카 컨테이너를 포드에 삽입하기 위한 리소스 변경.*
- *프로젝트에서 일부 리소스를 차단하도록 프로젝트를 제한합니다.*
- *종속 필드에 복잡한 검증을 수행하는 사용자 정의 리소스 검증.*

4.7.2.1. Admission Webhook의 유형

클러스터 관리자는 API 서버의 승인 체인에 승인 Webhook 또는 검증 승인 Webhook를 포함할 수 있습니다.

*승인 프로세스 **변경 단계에서 승인 Webhook** 변경이 호출되므로 리소스 콘텐츠가 지속되기 전에 수정할 수 있습니다. 변경 승인 Webhook의 한 예로는 네임스페이스의 주석을 사용하여 라벨 선택기를 찾아 Pod 사양에 추가하는 Pod 노드 선택기 기능이 있습니다.*

샘플 상호 승인 Webhook 구성

```
apiVersion: admissionregistration.k8s.io/v1beta1
 kind: MutatingWebhookConfiguration (1)
 metadata:
   name: <controller_name> (2)
 webhooks:
 - name: <webhook_name> (3)
   clientConfig: (4)
     service:
       namespace: (5)
       name: (6)
     path: <webhook_url> (7)
    caBundle: <cert> (8)
   rules: (9)
   - operations: (10)
```

```
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - <resource>
  failurePolicy: <policy> 11
```

1 변경 승인 Webhook 구성을 지정합니다.

2 승인 Webhook 오브젝트의 이름입니다.

3 호출할 웹 후크의 이름입니다.

4 웹 후크 서버에 연결하고 신뢰하고 데이터를 전송하는 방법에 대한 정보입니다.

5 프론트엔드 서비스가 생성되는 프로젝트입니다.

6 프론트 엔드 서비스의 이름입니다.

7 승인 요청에 사용되는 웹 후크 URL 입니다.

8 웹 후크 서버에서 사용하는 서버 인증서에 서명하는 PEM 인코딩 CA 인증서입니다.

9 API 서버가 이 컨트롤러를 사용해야 하는 시기를 정의하는 규칙입니다.

10 API 서버를 트리거하여 이 컨트롤러를 호출하는 작업은 다음과 같습니다.

- create
- update
- delete
- 연결

11 웹 후크 승인 서버를 사용할 수 없는 경우 정책 진행 방법을 지정합니다. **무시** (허용/실패) 또는 **실패** (블록/실패 마감).

***승인 프로세스의 검증 단계 중에 승인 Webhook** 검증이 호출됩니다. 이 단계에서는 특정 API 리소스에 대한 변형을 적용하여 리소스가 다시 변경되지 않게 합니다. Pod 노드 선택기는 프로젝트의 노드 선택기 제한으로 인해 모든 **nodeSelector** 필드가 제한되도록 하여 검증 승인의 예이기도 합니다.*

Admission Webhook 구성 검증 샘플

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration 1
metadata:
  name: <controller_name> 2
webhooks:
- name: <webhook_name> 3
  clientConfig: 4
    service:
      namespace: default 5
```

```
    name: kubernetes (6)
    path: <webhook_url> (7)
   caBundle: <cert> (8)
 rules: (9)
 - operations: (10)
   - <operation>
   apiGroups:
   - ""
   apiVersions:
   - "*"
   resources:
   - <resource>
 failurePolicy: <policy> (11)
```

(1) 검증 승인 Webhook 구성을 지정합니다.

(2) 웹 후크 승인 오브젝트의 이름입니다.

(3) 호출할 웹 후크의 이름입니다.

(4) 웹 후크 서버에 연결하고 신뢰하고 데이터를 전송하는 방법에 대한 정보입니다.

(5) 프론트엔드 서비스가 생성되는 프로젝트입니다.

(6) 프론트 엔드 서비스의 이름입니다.

(7) 승인 요청에 사용되는 웹 후크 URL 입니다.

(8) 웹 후크 서버에서 사용하는 서버 인증서에 서명하는 PEM 인코딩 CA 인증서입니다.

(9) API 서버가 이 컨트롤러를 사용해야 하는 시기를 정의하는 규칙입니다.

(10) API 서버를 트리거하여 이 컨트롤러를 호출하는 작업입니다.

- create
- update
- delete
- 연결

(11) 웹 후크 승인 서버를 사용할 수 없는 경우 정책 진행 방법을 지정합니다. **무시** (허용/실패) 또는 **실패** (블록/실패 마감).

참고

fail open으로 인해 모든 클라이언트에 대해 예기치 않은 동작이 발생할 수 있습니다.

4.7.2.2. Admission Webhook 만들기

먼저 외부 웹 후크 서버를 배포하고 제대로 작동하는지 확인합니다. 그렇지 않으면 웹 후크가 **열린 실패로** 구성되었는지 여부에 따라 **작업이** 무조건 수락되거나 거부됩니다.

1. YAML 파일에서 승인 Webhook 오브젝트를 교체하거나 검증 하도록 구성합니다.

2. 다음 명령을 실행하여 오브젝트를 생성합니다.

```
$ oc create -f <file-name>.yaml
```

승인 Webhook 오브젝트를 생성한 후 OpenShift Container Platform은 새 구성을 수행하는 데 몇 초가 걸립니다.

3. 승인 Webhook에 대한 프론트엔드 서비스를 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    role: webhook (1)
  name: <name>
spec:
  selector:
    role: webhook (2)
```

(1) (2) free-form 라벨을 사용하여 Webhook를 트리거합니다.

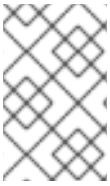
4. 다음 명령을 실행하여 오브젝트를 생성합니다.

```
$ oc create -f <file-name>.yaml
```

5. Webhook에서 제어하려는 Pod에 승인 Webhook 이름을 추가합니다.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: webhook (1)
  name: <name>
spec:
  containers:
    - name: <name>
      image: myrepo/myimage:latest
      imagePullPolicy: <policy>
      ports:
        - containerPort: 8000
```

(1) Webhook를 트리거하는 레이블입니다.



참고

라이브러리의 보안 및 이식 가능한 웹 후크 승인 서버 및 일반-admission-apiserver를 빌드하는 방법에 대한 엔드 투 엔드 예는 kubernetes-namespace- retention 프로젝트를 참조하십시오.

4.7.2.3. Admission Webhook Example

다음은 네임스페이스가 예약된 경우 네임스페이스 생성을 허용하지 않는 승인 Webhook의 예입니다.

```
apiVersion: admissionregistration.k8s.io/v1beta1
 kind: ValidatingWebhookConfiguration
 metadata:
  name: namespacereservations.admission.online.openshift.io
 webhooks:
 - name: namespacereservations.admission.online.openshift.io
  clientConfig:
   service:
    namespace: default
    name: webhooks
   path: /apis/admission.online.openshift.io/v1beta1/namespacereservations
   caBundle: KUBE_CA_HERE
  rules:
  - operations:
   - CREATE
   apiGroups:
   - ""
   apiVersions:
   - "b1"
   resources:
   - namespaces
  failurePolicy: Ignore
```

다음은 webhook라는 승인 Webhook에 의해 평가되는 Pod의 예입니다.

```
apiVersion: v1
kind: Pod
metadata:
 labels:
  role: webhook
 name: webhook
spec:
 containers:
  - name: webhook
   image: myrepo/myimage:latest
   imagePullPolicy: IfNotPresent
   ports:
- containerPort: 8000
```

다음은 웹 후크의 프런트엔드 서비스입니다.

```
apiVersion: v1
kind: Service
metadata:
 labels:
  role: webhook
 name: webhook
spec:
 ports:
  - port: 443
   targetPort: 8000
 selector:
role: webhook
```

4.8. 기타 API 오브젝트

4.8.1. LimitRange

제한 범위는 Kubernetes 네임스페이스 의 리소스에 배치된 최소/최대 제한을 적용하는 메커니즘을 제공합니다.

제한 범위를 네임스페이스에 추가하여 개별 Pod 또는 컨테이너에서 사용하는 최소 및 최대 CPU 및 메모리 양을 적용할 수 있습니다.

CPU 및 메모리 제한의 경우 **max** 값을 지정하지만 LimitRange 오브젝트에 **min** 제한을 지정하지 않으면 리소스에서 **max** 값보다 큰 CPU/메모리 리소스를 사용할 수 있습니다.

4.8.2. 리소스 쿼터

Kubernetes는 네임스페이스에 생성된 오브젝트 수와 네임스페이스 의 오브젝트에서 요청된 총 리소스 양을 모두 제한할 수 있습니다. 따라서 한 팀이 다른 클러스터 리소스를 사용하지 못하게 하는 메커니즘으로 각각 네임스페이스에서 여러 팀이 단일 Kubernetes 클러스터를 공유할 수 있습니다.

ResourceQuota 에 대한자세한 내용은 클러스터 관리를 참조하십시오.

4.8.3. 리소스

Kubernetes **리소스는** 포드 또는 컨테이너에서 요청하거나, 할당하거나, 사용할 수 있는 항목입니다. 예를 들어 메모리(RAM), CPU, 디스크 시간 및 네트워크 대역폭이 있습니다.

자세한 내용은 개발자 가이드를 참조하십시오.

4.8.4. Secret

시크릿 은 키, 암호 및 인증서와 같은 중요한 정보를 위한 스토리지입니다. 원하는 포드에서 액세스할 수 있지만 정의와 별도로 포함됩니다.

4.8.5. 영구 볼륨

영구 볼륨은 클러스터 관리자가 프로비저닝하는 인프라의 오브젝트(**PersistentVolume**) 입니다. 영구 볼륨은 상태 저장 애플리케이션을 위한 지속적 스토리지를 제공합니다.

4.8.6. PersistentVolumeClaim

PersistentVolumeClaim 오브젝트는 Pod 작성자의 스토리지 요청입니다. Kubernetes는 사용 가능한 볼륨 풀에 대한 클레임과 일치하고 함께 바인딩합니다. 그런 다음 Pod에서 클레임을 볼륨으로 사용합니다. Kubernetes는 볼륨을 필요한 Pod와 동일한 노드에서 사용할 수 있는지 확인합니다.

4.8.6.1. 사용자 정의 리소스

사용자 정의 리소스는 API를 확장하거나 자체 API를 프로젝트 또는 클러스터에 도입할 수 있는 Kubernetes API의 확장입니다.

사용자 지정 리소스로 Kubernetes API 확장을 참조하십시오 .

4.8.7. OAuth 개체

4.8.7.1. OAuthClient

OAuthClient 는 RFC 6749 섹션 2 에 설명된 대로 OAuth 클라이언트를 나타냅니다.

다음 **OAuthClient** 오브젝트가 자동으로 생성됩니다.

openshift-web-console	웹 콘솔의 토큰을 요청하는 데 사용되는 클라이언트
openshift-browser-client	대화형 로그인을 처리할 수 있는 사용자 에이전트를 사용하여 /oauth/token/request에서 토큰을 요청하는 데 사용되는 클라이언트
openshift-challenging-client	WWW-Authenticate 챌린지를 처리할 수 있는 사용자 에이전트로 토큰을 요청하는 데 사용되는 클라이언트

OAuthClient 오브젝트 정의

```
kind: "OAuthClient"
accessTokenMaxAgeSeconds: null (1)
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "openshift-web-console" (2)
  selflink: "/oapi/v1/oAuthClients/openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01Z"
respondWithChallenges: false (3)
secret: "45e27750-a8aa-11e4-b2ea-3c970e4b7ffe" (4)
redirectURIs:
  - "https://localhost:8443" (5)
```

(1) 액세스 토큰의 수명(초) 입니다(아래 설명 참조).

(2) 이름은 OAuth 요청에서 **client_id** 매개 변수로 사용됩니다.

(3) **respondWithChallenges** 가 **true** 로 설정되면 구성된 인증 방법으로 지원하는 경우 **/oauth/authorize** 에 대해 인증되지 않은 요청이 **WWW-Authenticate 챌린지** 가 됩니다.

(4) **secret** 매개 변수의 값은 권한 부여 코드 흐름에서 **client_secret** 매개 변수로 사용됩니다.

(5) 하나 이상의 절대 URI를 **redirectURIs** 섹션에 둘 수 있습니다. 권한 부여 요청과 함께 전송된 **redirect_uri** 매개 변수 앞에 지정된 **redirectURIs** 중 하나가 추가되어야 합니다.

accessTokenMaxAgeSeconds 값은 개별 OAuth 클라이언트의 마스터 구성 파일의 기본 **accessTokenMaxAgeSeconds** 값을 덮어씁니다. 클라이언트에 이 값을 설정하면 다른 클라이언트의 수명에 영향을 주지 않고 해당 클라이언트의 장기 액세스 토큰을 사용할 수 있습니다.

- **null** 인 경우 마스터 구성 파일의 기본값이 사용됩니다.
- **0** 으로 설정하면 토큰이 만료되지 않습니다.

- *0 보다 큰 값으로 설정하면 해당 클라이언트에 대해 발행된 토큰에 지정된 만료 시간이 부여됩니다. 예를 들면 **accessTokenMaxAgeSeconds입니다. 172800** 토큰이 발행된 후 48시간 후에 만료됩니다.*

4.8.7.2. OAuthClientAuthorization

***OAuthClientAuthorization** 은 특정 범위의 OAuth **AccessToken을 제공할 특정 OAuth Client** 에 대한 **사용자의** 승인을 나타냅니다.*

***OAuthClientAuthorization** 오브젝트는 **OAuth** 서버에 대한 권한 부여 요청 중에 수행됩니다.*

***OAuthClientAuthorization** Object Definition*

```
kind: "OAuthClientAuthorization"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "bob:openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
clientName: "openshift-web-console"
userName: "bob"
userUID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
scopes: []
```

4.8.7.3. OAuthAuthorizeToken

***OAuthAuthorizeToken** 은 RFC 6749 섹션 1.3.1 에 설명된 대로 **OAuth** 인증 코드를 나타냅니다.*

***OAuthAuthorizeToken** 은 RFC 6749, 4.1.1절에 설명된 대로 /oauth/authorize 엔드포인트에 대한 요청에 의해 생성됩니다.*

*그런 다음 **OAuthAuthorizeToken을** 사용하여 RFC 6749, 4.1.3 섹션에 설명된 대로 **/oauth/token** 엔드포인트에 대한 요청을 사용하여 **OAuthAccessToken** 을 가져올 수 있습니다.*

***OAuthAuthorizeToken** Object Definition*

```
kind: "OAuthAuthorizeToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" (1)
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
clientName: "openshift-web-console" (2)
expiresIn: 300 (3)
scopes: []
redirectURI: "https://localhost:8443/console/oauth" (4)
userName: "bob" (5)
userUID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" (6)
```

(1) ***name** 은 OAuthAccessToken을 교환하기 위해 권한 부여 코드로 사용되는 토큰 이름을 나타냅니다.*

(2) ***clientName** 값은 이 토큰을 요청한 OAuthClient 입니다.*

(3) ***expiresIn** 값은 creationTimestamp 에서 만료(초) 입니다.*

4 **redirectURI** 값은 이 토큰으로 인한 권한 부여 흐름 중에 사용자가 리디렉션된 위치입니다.

5 **username은** 에 대한 OAuthAccessToken을 가져올 수 있는 사용자 이 토큰의 이름을 나타냅니다.

6 **userUID** 는 이 토큰을 통해 OAuthAccessToken을 가져올 수 있는 사용자 UID를 나타냅니다.

4.8.7.4. OAuthAccessToken

OAuthAccessToken 은 RFC 6749 섹션 1.4 에 설명된 대로 **OAuth** 액세스 토큰을 나타냅니다.

OAuthAccessToken 은 RFC 6749, 4.1.3 섹션에 설명된 대로 **/oauth/token** 엔드포인트에 대한 요청에 의해 생성됩니다.

액세스 토큰은 API 인증을 위해 전달자 토큰으로 사용됩니다.

OAuthAccessToken Object Definition

```
kind: "OAuthAccessToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "ODliOGE5ZmMtYzczYi00Nzk1LTg4MGEtNzQyZmUxZmUwY2Vh" 1
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:02-00:00"
clientName: "openshift-web-console" 2
expiresIn: 86400 3
scopes: []
redirectURI: "https://localhost:8443/console/oauth" 4
userName: "bob" 5
userUID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" 6
authorizeToken: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" 7
```

1 **name** 은 API 인증에 전달자 토큰으로 사용되는 토큰 이름입니다.

2 **clientName** 값은 이 토큰을 요청한 OAuthClient 입니다.

3 **expiresIn** 값은 creationTimestamp 에서 만료(초) 입니다.

4 **redirectURI** 는 이 토큰이 생성되는 동안 사용자가 권한 부여 흐름 중에 로 리디렉션된 위치입니다.

5 username ***은 이 토큰에서 인증을 허용하는 사용자를*** 나타냅니다.

6 **userUID** 는 이 토큰이 인증을 으로 허용하는 User를 나타냅니다.

7 **authorizeToken** 은 이 토큰을 가져오는 데 사용되는 OAuthAuthorizationToken의 이름입니다(있는 경우).

4.8.8. 사용자 객체

4.8.8.1. ID

사용자가 OpenShift Container Platform에 로그인하면 구성된 ID 공급자를 사용하여 이를 수행합니다. 이렇게 하면 사용자의 ID가 결정되고 해당 정보를 OpenShift Container Platform에 제공합니다.

그런 다음 OpenShift Container Platform은 해당 **ID** 에 대한 **UserIdentityMapping** 을 찾습니다.

참고

*예를 들어 외부 LDAP 시스템을 사용하는 경우와 같이 **조회** 매핑 방법으로 ID 프로바이더가 구성된 경우 이 자동 매핑은 수행되지 않습니다. 매핑을 수동으로 생성해야 합니다. 자세한 내용은 Lookup Mapping Method 를 참조하십시오.*

- ***ID가** 이미 있지만 사용자에게 매핑되지 않은 경우 **로그인** 에 실패합니다.*
- ***ID가** 이미 존재하고 사용자에게 매핑되는 **경우** 사용자에게 매핑된 사용자에 대한 **OAuthAccessToken** 이 **제공됩니다**.*
- ***ID** 가 존재하지 않으면 **ID**, **사용자** 및 **UserIdentityMapping** 이 생성되고 사용자에게 매핑된 사용자에 대한 **OAuthAccessToken** 이 제공됩니다 **.***

ID 오브젝트 정의

```
kind: "Identity"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" (1)
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
providerName: "anypassword" (2)
providerUserName: "bob" (3)
user:
  name: "bob" (4)
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" (5)
```

(1) *ID 이름은 providerName:providerUserName 형식이어야 합니다.*

(2) ***providerName** 은 ID 공급자의 이름입니다.*

(3) ***providerUserName** 은 ID 공급자의 범위에서 이 ID를 고유하게 표시하는 이름입니다.*

(4) ***사용자 매개 변수의 이름은 이 ID가 매핑되는 사용자의 이름입니다.***

(5) ***uid 는 이 ID가 매핑되는 사용자의 UID를 나타냅니다.***

4.8.8.2. 사용자

사용자는 시스템의 행위자를 나타냅니다. 사용자에게는 사용자 또는 그룹에 역할을 추가하여 권한이 부여됩니다.

사용자 오브젝트는 처음 로그인할 때 자동으로 생성되거나 API를 통해 생성할 수 있습니다.

참고

/, :, %를 포함하는 OpenShift Container Platform 사용자 이름은 지원되지 않습니다.

사용자 오브젝트 정의

```
kind: "User"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "bob" 1
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
identities:
  - "anypassword:bob" 2
fullName: "Bob User" 3
```

1

name 은 사용자에게 역할을 추가할 때 사용되는 사용자 이름입니다.

2

ID 의 값은 이 사용자에게 매핑되는 ID 오브젝트입니다. 로그인할 수 없는 사용자의 경우 null 이거나 비어 있을 수 있습니다.

3

fullName 값은 사용자의 선택적 표시 이름입니다.

4.8.8.3. UserIdentityMapping

UserIdentityMapping 은 ID 를 사용자에게 매핑합니다 .

UserIdentityMapping 생성, 업데이트 또는 삭제는 Identity 및 User 오브젝트의 해당 필드를 수정합

니다.

ID 는 단일 사용자만 매핑할 수 있으므로 특정 ID로 로그인하면 사용자를 모호하지 않게 확인할 수 있습니다.

사용자는 여러 ID를 매핑할 수 있습니다. 이를 통해 여러 로그인 방법을 통해 동일한 사용자를 식별할 수 있습니다.

useridentitymapping 오브젝트 정의

```
kind: "UserIdentityMapping"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" (1)
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
identity:
  name: "anypassword:bob"
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
user:
  name: "bob"
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
```

(1) useridentitymapping 이름은 매핑된 ID 이름과 일치합니다.

4.8.8.4. Group

그룹은 시스템의 사용자 목록을 나타냅니다. 그룹에는 사용자 또는 그룹에 역할을 추가하여 권한이 부여됩니다.

그룹 오브젝트 정의

```
kind: "Group"
apiVersion: "user.openshift.io/v1"
metadata:
```

```
  name: "developers" (1)
  creationTimestamp: "2015-01-01T01:01:01-00:00"
users:
  - "bob" (2)
```

(1) **name** 은 그룹에 역할을 추가할 때 사용되는 그룹 이름입니다.

(2) 사용자 값은 이 그룹의 멤버인 **User** 오브젝트의 이름입니다.

5장. 네트워킹

5.1. 네트워킹

5.1.1. 개요

Kubernetes는 pod가 네트워크를 통해 서로 연결할 수 있도록 하며 각 pod에 내부 네트워크의 IP 주소를 할당합니다. 이렇게 하면 pod 내의 모든 컨테이너가 마치 동일한 호스트에 있는 것처럼 작동합니다. 각 pod에 고유 IP 주소를 부여하면 포트 할당, 네트워킹, 이름 지정, 서비스 검색, 로드 밸런싱, 애플리케이션 구성 및 마이그레이션 등 다양한 업무를 할 때 pod를 물리적 호스트 또는 가상 머신처럼 취급할 수 있습니다.

포드 간 링크를 생성할 필요가 없으며 포드가 IP 주소를 사용하여 직접 서로 통신하지 않는 것이 좋습니다. 대신 서비스를 생성한 다음 서비스와 상호 작용하는 것이 좋습니다.

5.1.2. OpenShift Container Platform DNS

frontend 포드가 백엔드 서비스와 통신하기 위해 프론트엔드 및 백엔드 서비스와 같은 여러 서비스를 실행하는 경우 사용자 이름, 서비스 IP 등에 대한 환경 변수가 생성됩니다. 서비스를 삭제하고 다시 생성하면 새 IP 주소를 서비스에 할당할 수 있으며 서비스 IP 환경 변수의 업데이트된 값을 가져오기 위해 프론트엔드 포드를 다시 생성해야 합니다. 또한 서비스 IP가 올바르게 생성되고 프론트엔드 포드에 환경 변수로 제공할 수 있도록 백엔드 서비스를 생성한 후 frontend 포드에 생성해야 합니다.

이러한 이유로 서비스 DNS는 물론 서비스 IP/포트를 통해서도 서비스를 이용할 수 있도록 OpenShift Container Platform에 DNS를 내장했습니다. OpenShift Container Platform은 서비스의 DNS 쿼리에 응답하는 마스터에서 SkyDNS 를 실행하여 분할된 DNS를 지원합니다. 마스터는 기본적으로 포트 53을 수신 대기합니다.

노드가 시작되면 다음 메시지는 Kubelet이 마스터로 올바르게 확인되었음을 나타냅니다.

```
0308 19:51:03.118430    4484 node.go:197] Started Kubelet for node
openshiftdev.local, server at 0.0.0.0:10250
I0308 19:51:03.118459    4484 node.go:199]   Kubelet is setting 10.0.2.15 as a
DNS nameserver for domain "local"
```

두 번째 메시지가 표시되지 않으면 Kubernetes 서비스를 사용할 수 없습니다.

노드 호스트에서 각 컨테이너의 이름 서버에는 앞에 마스터 이름이 추가되고 컨테이너의 기본 검색 도메인은 <pod_namespace>.cluster.local 이 됩니다. 그런 다음 컨테이너는 Docker 형식 컨테이너의 기

*본 동작인 노드의 다른 이름 서버 앞에 이름 서버 쿼리를 마스터로 보냅니다. 마스터는 다음 양식이 있는 **.cluster.local** 도메인의 쿼리에 응답합니다.*

*표 **5.1. DNS** 예제 이름*

개체 유형	예제
Default	<pod_namespace>.cluster.local
서비스	<service>.<pod_namespace>.svc.cluster.local
엔드포인트	<name>.<namespace>.endpoints.cluster.local

*이렇게 하면 서비스에 대한 새 **IP**를 생성하는 새 서비스를 가져오기 위해 프론트엔드 포드를 다시 시작하지 않아도 됩니다. 또한 포드에서 서비스 **DNS**를 사용할 수 있기 때문에 환경 변수를 사용할 필요가 없습니다. 또한 **DNS**는 변경되지 않으므로 구성 파일에서 데이터베이스 서비스를 **db.local** 로 참조할 수 있습니다. 조회가 서비스 **IP**로 확인되고, 프론트엔드 포드보다 먼저 백엔드 서비스를 만들 필요가 없기 때문에 와일드카드 조회도 지원됩니다. 따라서 서비스 이름(따라서 **DNS**)이 초기 상태로 설정되어 있기 때문입니다.*

*이 **DNS** 구조는 포틸 **IP**가 서비스에 할당되지 않고 **kube-proxy**가 해당 엔드포인트에 라우팅을 로드 밸런싱하지 않거나 라우팅을 제공하는 헤드리스 서비스도 다룹니다. 서비스 **DNS**를 계속 사용하고 서비스의 각 포드에 하나씩 여러 **A** 레코드로 응답할 수 있으므로 클라이언트가 각 포드 간에 라운드 로빈할 수 있습니다.*

5.2. OPENSHIFT SDN

5.2.1. 개요

***OpenShift Container Platform**에서는 소프트웨어 정의 네트워킹(SDN) 접근법을 사용하여 **OpenShift Container Platform** 클러스터 전체의 **pod** 간 통신이 가능한 통합 클러스터 네트워크를 제공합니다. 이 **pod** 네트워크는 **OVS(Open vSwitch)**를 사용하여 오버레이 네트워크를 구성하는 **OpenShift SDN**에 의해 설정 및 유지 관리됩니다.*

***OpenShift SDN**은 **Pod** 네트워크를 구성하는 데 필요한 세 가지 **SDN** 플러그인을 제공합니다.*

- ***ovs-subnet** 플러그인은 원래 플러그인으로, 모든 포드가 다른 모든 포드 및 서비스와 통신할 수 있는 "플랫" 포드 네트워크를 제공합니다.*

- ***ovs-multitenant** 플러그인은 포드 및 서비스에 대한 프로젝트 수준 격리를 제공합니다. 각 프로젝트에는 프로젝트에 할당된 포드의 트래픽을 확인하는 고유한 **VNID**(가상 네트워크 **ID**)가 수*

신됩니다. 다른 프로젝트의 **Pod**는 다른 프로젝트의 **Pod** 및 **Service**에서 패킷을 보내거나 받을 수 없습니다.

그러나 **VNID 0**을 수신하는 프로젝트는 다른 모든 포드와 통신할 수 있다는 점에서 더 권한이 부여되며 다른 모든 포드는 사용자와 통신할 수 있습니다. **OpenShift Container Platform** 클러스터에서 **default** 프로젝트에는 **VNID 0**이 있습니다. 이렇게 하면 로드 밸런서와 같은 특정 서비스가 클러스터의 다른 모든 포드와 통신할 수 있으며 그 반대의 경우도 마찬가지입니다.

- **ovs-networkpolicy** 플러그인을 사용하면 프로젝트 관리자가 **NetworkPolicy** 오브젝트를 사용하여 자체 격리 정책을 구성할 수 있습니다.

참고

마스터 및 노드에서 **SDN**을 구성하는 방법에 대한 정보는 **SDN** 구성에서 확인할 수 있습니다.

5.2.2. 마스터에 대한 설계

OpenShift Container Platform 마스터에서 **OpenShift SDN**은 **etcd** 에 저장된 노드 레지스트리를 유지 관리합니다. 시스템 관리자가 노드를 등록하면 **OpenShift SDN**에서 클러스터 네트워크에서 사용되지 않은 서브넷을 할당하고 이 서브넷을 레지스트리에 저장합니다. 노드가 삭제되면 **OpenShift SDN**은 레지스트리에서 서브넷을 삭제하고 사용 가능한 서브넷을 다시 할당할 수 있다고 간주합니다.

기본 구성에서 클러스터 네트워크는 **10.128.0.0/14** 네트워크(예: **10.128.0.0 - 10.131.255.255**)이며 노드에는 **/23** 서브넷(예: **10.128.0.0/23,10.128.2.0/23,10.128.4.0/23** 등)이 할당됩니다. 즉, 클러스터 네트워크에는 노드에 할당할 수 있는 **512**개의 서브넷이 있으며 지정된 노드에는 실행 중인 컨테이너에 할당할 수 있는 주소가 할당됩니다. 는 호스트 서브넷 크기이므로 클러스터 네트워크의 크기 및 주소 범위를 구성할 수 있습니다.

참고

서브넷이 더 높은 옥텟으로 확장되면 공유 옥텟에서 **0**이 있는 서브넷 비트가 먼저 할당되도록 순환됩니다. 예를 들어 네트워크가 **hostsubnetlength=6** 인 **10.1.0.0/16**인 경우 **10.1.0.0/26** 및 **10.1.1.0/26**의 서브넷이 **10.1.0.64/26** 이전에 할당됩니다. 이렇게 하면 서브넷을 더 쉽게 추적할 수 있습니다.

마스터의 **OpenShift SDN**은 클러스터 네트워크에 액세스할 수 있도록 로컬(마스터) 호스트를 구성하지 않습니다. 결과적으로 마스터 호스트는 노드로 실행되지 않는 한 클러스터 네트워크를 통해 포드에 액세스할 수 없습니다.

ovs-multitenant 플러그인을 사용하는 경우 OpenShift SDN 마스터는 프로젝트의 생성 및 삭제를 감시하고 나중에 노드에서 트래픽을 격리하는 데 사용하는 VXLAN VNID를 할당합니다.

5.2.3. 노드에서 설계

노드에서 OpenShift SDN은 먼저 마스터가 노드에 서브넷을 할당하도록 앞에서 설명한 레지스트리에서 SDN 마스터에 로컬 호스트를 등록합니다.

그런 다음 OpenShift SDN은 세 개의 네트워크 장치를 만들고 구성합니다.

- *br0: 포드 컨테이너가 연결될 OVS 브리지 장치입니다. OpenShift SDN은 이 브리지에서 서브넷이 아닌 흐름 규칙 집합도 구성합니다.*
- *tun0: OVS 내부 포트(br 0의포트 2). 이 네트워크에는 클러스터 서브넷 게이트웨이 주소가 할당되며 외부 네트워크 액세스에 사용됩니다. OpenShift SDN에서는 NAT를 통해 클러스터 서브넷에서 외부 네트워크로 액세스할 수 있도록 netfilter 및 라우팅 규칙을 구성합니다.*
- *vxlan_sys_4789: 원격 노드의 컨테이너에 대한 액세스를 제공하는 OVS VXLAN 장치(br 0의 포트 1)입니다. OVS 규칙에서 vxlan0 을 참조합니다.*

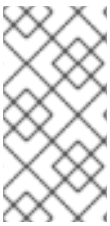
호스트에서 포드를 시작할 때마다 OpenShift SDN은 다음과 같습니다.

1. *Pod에 노드의 클러스터 서브넷에서 사용 가능한 IP 주소를 할당합니다.*
2. *포드 veth 인터페이스 쌍의 호스트 측면을 OVS 브리지 br0 에 연결합니다.*
3. *OpenFlow 규칙을 OVS 데이터베이스에 추가하여 새 포드로 주소가 지정된 트래픽을 올바른 OVS 포트로 라우팅합니다.*
4. *ovs-multitenant 플러그인의 경우 OpenFlow 규칙을 추가하여 포드에서 들어오는 트래픽을 포드의 VNID로 태그하고 트래픽의 VNID가 Pod의 VNID(또는 권한 있는 VNID 0)와 일치하는 경우 Pod로 트래픽을 허용합니다. 일치하지 않는 트래픽은 일반 규칙에 따라 필터링됩니다.*

OpenShift SDN 노드는 SDN 마스터의 서브넷 업데이트도 감시합니다. 새 서브넷이 추가되면 노드는 br0 에 OpenFlow 규칙을 추가하여 원격 서브넷의 대상 IP 주소가 있는 패킷이 vxlan0 (br 0의 포트 1)으로 이동하므로 네트워크로 나갑니다. ovs-subnet 플러그인은 VNID 0을 사용하여 VXLAN의 모든 패킷을 전송하지만 ovs-multitenant 플러그인은 소스 컨테이너에 적절한 VNID를 사용합니다.

5.2.4. 패킷 흐름

컨테이너 A의 eth0에 대한 피어 가상 이더넷 장치의 이름은 vethA이고 컨테이너 B의 eth0 에 대한 피어 가상 이더넷 장치의 이름은 veth B 인 두 개의 컨테이너 A와 B가 있다고 가정합니다.



참고

Docker 서비스의 피어 가상 이더넷 장치를 아직 사용하지 않는 경우 Docker의 고급 네트워킹 설명서를 참조하십시오.

이제 먼저 컨테이너 A가 로컬 호스트에 있고 컨테이너 B도 로컬 호스트에 있다고 가정합니다. 컨테이너 A에서 컨테이너 B로의 패킷 흐름은 다음과 같습니다.

eth0 (A의 netns에서) → vethA → br0 → vethB → eth0 (B의 netns에서)

다음으로 컨테이너 A가 로컬 호스트에 있고 컨테이너 B가 클러스터 네트워크의 원격 호스트에 있다고 가정합니다. 컨테이너 A에서 컨테이너 B로의 패킷 흐름은 다음과 같습니다.

eth0 (A의 netns에서) → vethA → br0 → vxlan0 → 네트워크 [1] → vxlan0 → br0 → vethB → eth0 (B의 netns에서)

마지막으로 컨테이너 A가 외부 호스트에 연결하는 경우 트래픽은 다음과 같습니다.

eth0 (A의 netns에서) → vethA → br0 → tun0 → (NAT) → eth0 (물리 장치) → 인터넷

대부분의 패킷 전달 결정은 플러그인 네트워크 아키텍처를 간소화하고 유연한 라우팅을 제공하는 OVS 브리지 br0 에서 OpenFlow 규칙을 사용하여 수행됩니다. ovs-multitenant 플러그인의 경우 적용 가능한 네트워크 격리 도 제공합니다.

5.2.5. 네트워크 격리

ovs-multitenant 플러그인을 사용하여 네트워크 격리를 수행할 수 있습니다. 패킷이 기본이 아닌 프로젝트에 할당된 포드를 종료하면 OVS 브리지 br0 은 프로젝트의 할당된 VNID와 패킷하는 태그입니다. 패킷이 노드의 클러스터 서브넷에 있는 다른 IP 주소로 전달되는 경우 OVS 브리지는 VNID가 일치하는 경우에만 패킷이 대상 포드로 전달되도록 허용합니다.

VXLAN 터널을 통해 다른 노드에서 패킷을 수신하면 터널 ID가 VNID로 사용되며 OVS 브리지는 터널 ID가 대상 포드의 VNID와 일치하는 경우에만 패킷이 로컬 포드로 전달되도록 허용합니다.

다른 클러스터 서브넷으로 향하는 패킷은 VNID로 태그가 지정되며 클러스터 서브넷을 소유하는 노드의 터널 대상 주소가 있는 VXLAN 터널로 전달됩니다.

앞에서 설명한 대로 VNID 0은 할당된 모든 Pod에 VNID 0을 입력할 수 있는 해당 트래픽에서 권한이 있으며 VNID 0을 사용하는 트래픽은 모든 Pod를 입력할 수 있습니다. 기본 OpenShift Container Platform 프로젝트만 VNID 0으로 할당됩니다. 다른 모든 프로젝트에는 격리 가능 고유 VNID가 할당됩니다. 클러스터 관리자는 관리자 CLI 를 사용하여 프로젝트의 포드 네트워크를 선택적으로 제어할 수 있습니다.

5.3. 사용 가능한 SDN 플러그인

OpenShift Container Platform은 OpenShift Container Platform과 Kubernetes 간의 인터페이스로 Kubernetes CNI(Container Network Interface) 를 지원합니다. 소프트웨어 정의 네트워크(SDN) 플러그인은 네트워킹 요구 사항에 대한 네트워크 기능과 일치합니다. CNI 인터페이스를 지원하는 추가 플러그인을 필요에 따라 추가할 수 있습니다.

5.3.1. OpenShift SDN

OpenShift SDN은 Ansible 기반 설치 절차의 일부로 기본적으로 설치 및 구성됩니다. 자세한 내용은 OpenShift SDN 섹션을 참조하십시오.

5.3.2. 타사 SDN 플러그인

5.3.2.1. Cisco ACI SDN

OpenShift Container Platform용 Cisco ACI CNI 플러그인은 Cisco ACI 패브릭에 연결된 Cisco ACI(Application Policy Infrastructure Controller) 컨트롤러와 하나 이상의 OpenShift Container Platform 클러스터 간의 통합을 제공합니다.

이 통합은 두 가지 주요 기능 영역에 걸쳐 구현됩니다.

1. *Cisco ACI CNI 플러그인은 OpenShift Container Platform 워크로드를 위한 IP 주소 관리, 네트워킹, 로드 밸런싱 및 보안 기능을 제공하기 위해 ACI 패브릭 기능을 OpenShift Container Platform 클러스터로 확장합니다. Cisco ACI CNI 플러그인은 모든 OpenShift Container Platform Pod를 Cisco ACI에서 제공하는 통합 VXLAN 오버레이에 연결합니다.*

2. *Cisco ACI CNI 플러그인은 전체 OpenShift Container Platform 클러스터를 Cisco APIC에서 VMM 도메인으로 모델링합니다. 이를 통해 APIC는 OpenShift Container Platform 노드, OpenShift Container Platform 네임스페이스, 서비스, 배포, Pod, IP 및 MAC 주소, 사용 인터페이스 등을 포함하여 OpenShift Container Platform 클러스터의 리소스 인벤토리에 액세스할 수 있습니다. APIC는 이 정보를 사용하여 작업을 간소화하기 위해 물리적 리소스와 가상 리소스를 자동으로 상호 연결합니다.*

Cisco ACI CNI 플러그인은 OpenShift Container Platform 개발자 및 관리자를 위해 투명하고 운영 관점에서 원활하게 통합되도록 설계되었습니다.

자세한 내용은 Cisco ACI CNI 플러그인 for Red Hat OpenShift Container Platform Architecture and Design Guide를 참조하십시오.

5.3.2.2. Flannel SDN

Flannel 은 컨테이너를 위해 특별히 설계된 가상 네트워킹 계층입니다. OpenShift Container Platform은 기본 소프트웨어 정의 네트워킹(SDN) 구성 요소가 아닌 네트워킹 컨테이너에 사용할 수 있습니다. 이 기능은 OpenStack과 같은 SDN에도 의존하는 클라우드 공급자 플랫폼에서 OpenShift Container Platform을 실행하고 두 플랫폼을 통해 패킷을 두 번 캡슐화하지 않으려는 경우 유용합니다.

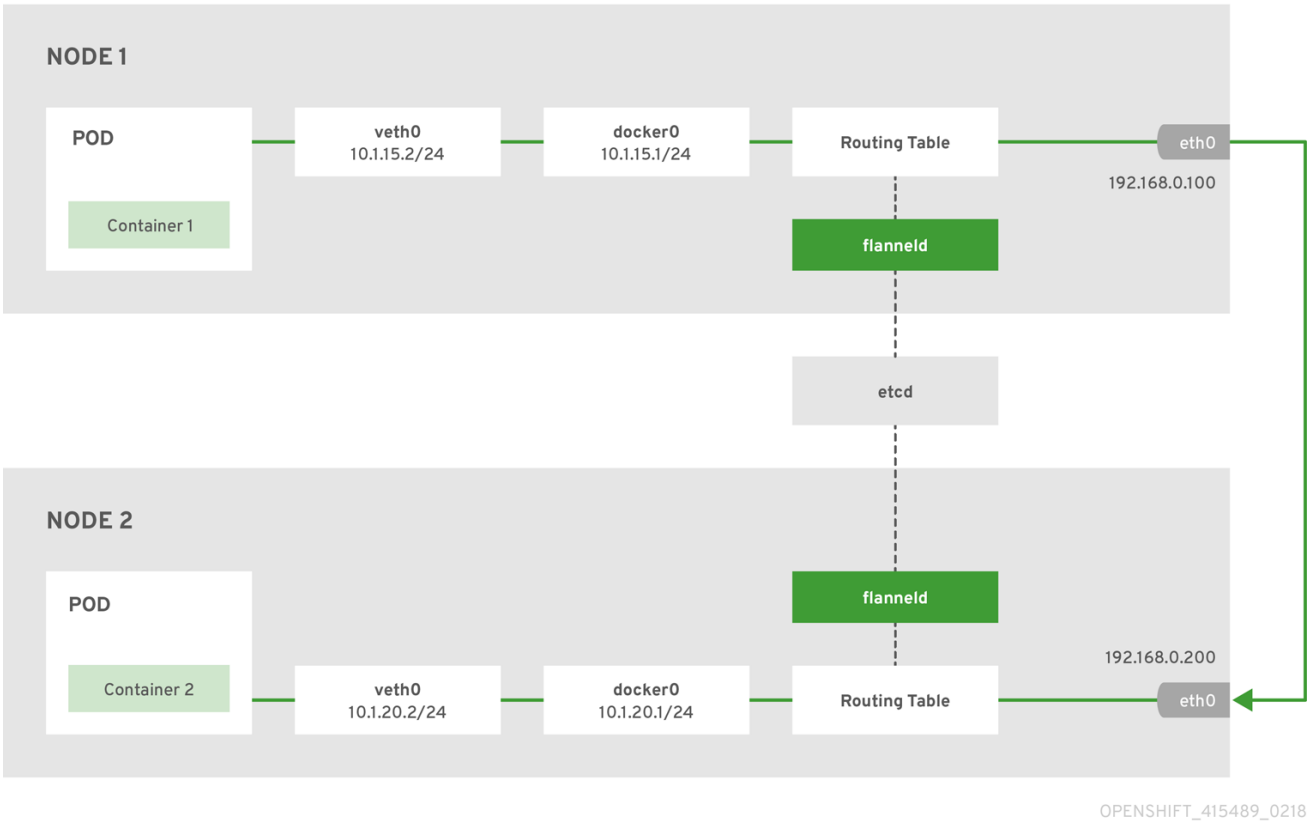
아키텍처

OpenShift Container Platform은 host-gw 모드에서 flannel 을 실행하여 컨테이너에서 컨테이너로 경로를 매핑합니다. 네트워크 내의 각 호스트는 flanneld 라는 에이전트를 실행합니다. 이 에이전트는 다음을 담당합니다.

- *각 호스트에서 고유한 서브넷 관리*
- *호스트의 각 컨테이너에 IP 주소 배포*
- *다른 호스트의 경우에도 한 컨테이너의 경로 매핑*

각 **flanneld** 에이전트는 중앙 집중식 **etcd** 저장소에 이 정보를 제공하므로 호스트의 다른 에이전트가 **flannel** 네트워크 내의 다른 컨테이너로 패킷을 라우팅할 수 있습니다.

다음 다이어그램에서는 **flannel** 네트워크를 사용하여 한 컨테이너에서 다른 컨테이너의 아키텍처 및 데이터 흐름을 보여줍니다.



노드 **1** 에는 다음 경로가 포함됩니다.

```
default via 192.168.0.100 dev eth0 proto static metric 100
10.1.15.0/24 dev docker0 proto kernel scope link src 10.1.15.1
10.1.20.0/24 via 192.168.0.200 dev eth0
```

노드 **2** 에는 다음 경로가 포함됩니다.

```
default via 192.168.0.200 dev eth0 proto static metric 100
10.1.20.0/24 dev docker0 proto kernel scope link src 10.1.20.1
10.1.15.0/24 via 192.168.0.100 dev eth0
```

5.3.2.3. NSX-T SDN

VMware NSX-T ™ Data Center는 기본 **OpenShift Container Platform** 네트워킹 기능을 위한 소프트웨어로 전환, 라우팅, 액세스 제어, 방화벽 및 **QoS**와 같은 레이어 **2** 네트워킹 서비스(예: 스위칭, 라우

팅, 액세스 제어, 방화벽, QoS)를 통해 전체 계층 2를 재현하는 정책 기반 오버레이 네트워크를 제공합니다.

NSX-T 구성 요소는 베어 메탈, 가상 머신 및 OpenShift Container Platform 포드를 연결하는 데이터 센터 전체 NSX-T 가상화 네트워크에 OpenShift Container Platform SDN을 통합하는 Ansible 설치 절차의 일부로 설치하고 구성할 수 있습니다. VMware NSX-T를 사용하여 OpenShift Container Platform을 설치하고 배포하는 방법에 대한 자세한 내용은 설치 섹션을 참조하십시오.

NSX-T 컨테이너 플러그인(NCP)은 일반적으로 전체 데이터 센터에 구성된 NSX-T Manager에 OpenShift Container Platform을 통합합니다.

NSX-T 데이터 센터 아키텍처 및 관리에 대한 자세한 내용은 VMware NSX-T Data Center v2.4 설명서 및 NSX-T NCP 구성 가이드를 참조하십시오.

5.3.2.4. Nuage SDN

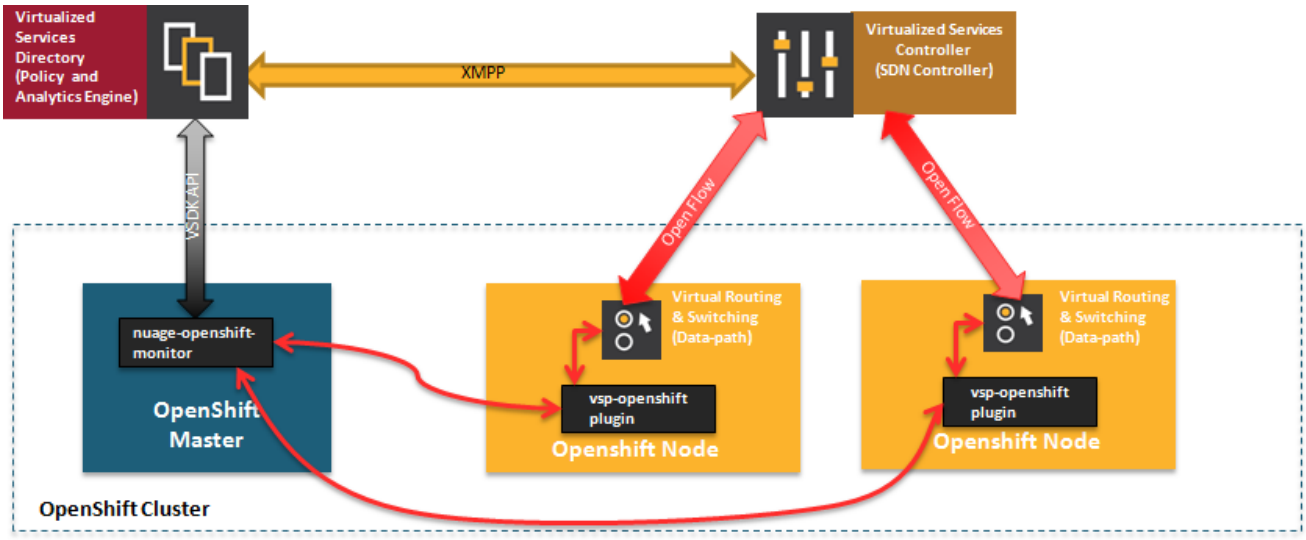
Nuage Networks의 SDN 솔루션은 OpenShift Container Platform 클러스터의 Pod에 확장성이 뛰어난 정책 기반 오버레이 네트워킹을 제공합니다. Nuage SDN은 Ansible 기반 설치 절차의 일부로 설치 및 구성할 수 있습니다. Nuage SDN을 사용하여 OpenShift Container Platform을 설치하고 배포하는 방법에 대한 자세한 내용은 고급 설치 섹션을 참조하십시오.

Nuage Networks 는 VSP(가상화된 서비스 플랫폼)라는 확장성이 뛰어난 정책 기반 SDN 플랫폼을 제공합니다. Nuage VSP는 데이터 플레인용 오픈 소스 Open vSwitch와 함께 SDN 컨트롤러를 사용합니다.

Nuage는 오버레이를 사용하여 OpenShift Container Platform과 VM 및 베어 메탈 서버로 구성된 다른 환경 간에 정책 기반 네트워킹을 제공합니다. 플랫폼의 실시간 분석 엔진을 사용하면 OpenShift Container Platform 애플리케이션에 대한 가시성과 보안 모니터링이 가능합니다.

Nuage VSP는 OpenShift Container Platform과 통합되어 DevOps 팀이 직면한 네트워크 지연을 제거하여 비즈니스 애플리케이션을 신속하게 켜고 업데이트할 수 있습니다.

그림 5.1. OpenShift Container Platform과 Nuage VSP 통합



통합을 담당하는 두 가지 특정 구성 요소가 있습니다.

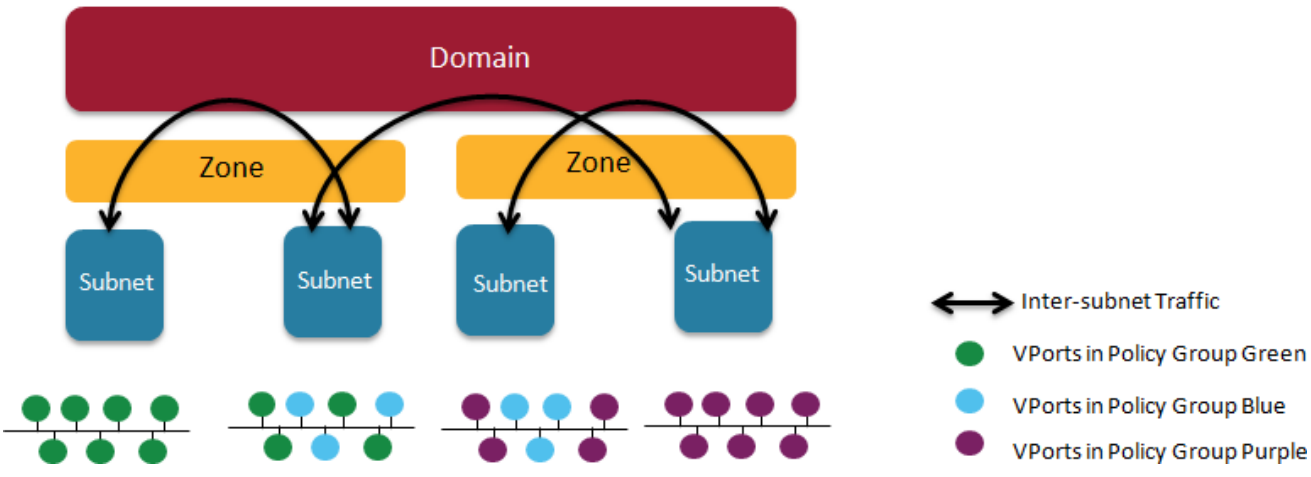
1. ***OpenShift Container Platform** 마스터 노드에서 별도의 서비스로 실행되는 **nuage-openshift-monitor** 서비스입니다.*

2. *클러스터의 각 노드에서 **OpenShift Container Platform** 런타임에 의해 호출되는 **vsp-openshift** 플러그인입니다.*

***Nuage Virtual Routing and Switching software (VRS)**는 오픈 소스 **Open vSwitch**를 기반으로 하며 데이터 경로 전달을 담당합니다. **VRS**는 각 노드에서 실행되며 컨트롤러에서 정책 구성을 가져옵니다.*

***Nuage VSP** 용어*

그림 5.2. Nuage VSP 빌드 블록



1. *도메인: 조직에는 하나 이상의 도메인이 포함되어 있습니다. 도메인은 단일 "계층 **3**" 공간입니다. 표준 네트워킹 용어에서 도메인은 **VRF** 인스턴스에 매핑됩니다.*

2. *영역: 영역은 도메인 아래에 정의됩니다. 영역은 네트워크의 어떤 항목에도 매핑되지 않지만, 대신 정책이 연결된 개체 역할을 하며 영역의 모든 엔드포인트가 동일한 정책 집합을 준수하도록 합니다.*

3. *서브넷: 서브넷은 영역에 정의되어 있습니다. 서브넷은 도메인 인스턴스 내의 특정 계층 **2** 서브넷입니다. 서브넷은 도메인 내에서 고유하며 고유합니다. 즉, 도메인 내의 서브넷은 표준 **IP** 서브넷 정의에 따라 중복되거나 다른 서브넷을 포함할 수 없습니다.*

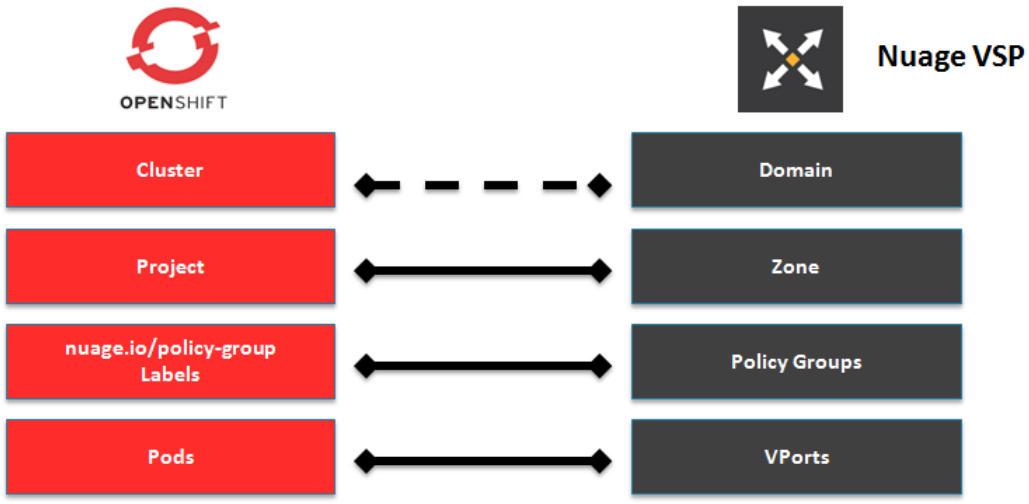
4. ***VPort: VPort**는 더 세분화된 구성을 제공하기 위한 도메인 계층 구조의 새로운 수준입니다. 컨테이너 및 **VM** 외에도 **VPort**는 베어 메탈 서버, 어플라이언스 및 레거시 **VLAN**에 연결을 제공하는 호스트 및 브리지 인터페이스를 연결하는 데도 사용됩니다.*

5. *정책 그룹: 정책 그룹은 **VPort** 컬렉션입니다.*

구조 매핑

*많은 OpenShift Container Platform 개념에 는 **Nuage VSP** 구성 요소와 직접 매핑됩니다.*

*그림 **5.3. Nuage VSP** 및 **OpenShift Container Platform** 매핑*



Nuage 서브넷은 OpenShift Container Platform 노드에 매핑되지 않지만 특정 프로젝트의 서브넷은 OpenShift Container Platform의 여러 노드를 확장할 수 있습니다.

OpenShift Container Platform에서 생성된 Pod는 VSP에서 생성되는 가상 포트로 변환됩니다. vsp-openshift 플러그인은 VRS와 상호 작용하고 VSC를 통해 VSD에서 해당 가상 포트에 대한 정책을 가져옵니다. 정책 그룹은 동일한 정책 집합을 적용해야 하는 여러 포드를 함께 그룹화하도록 지원됩니다. 현재 포드는 VSD의 관리 사용자가 정책 그룹을 생성하는 작업 워크플로 를 사용하여 정책 그룹에만 할당할 수 있습니다. 정책 그룹의 일부인 포드는 포드 사양의 nuage.io/policy-group 라벨을 통해 지정합니다.

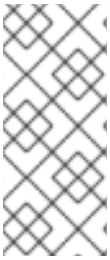
통합 구성 요소

Nuage VSP는 다음 두 가지 주요 구성 요소를 사용하여 OpenShift Container Platform과 통합됩니다.

1. nuage-openshift-monitor
2. vsp-openshift plugin

nuage-openshift-monitor

Nuage-openshift-monitor 는 OpenShift Container Platform API 서버를 모니터링하여 프로젝트, 서비스, 사용자, 사용자 그룹 등을 생성하는 서비스입니다.



참고

여러 마스터가 있는 HA(고가용성) OpenShift Container Platform 클러스터의 경우 nuage-openshift-monitor 프로세스는 모든 마스터에서 기능을 변경하지 않고 독립적으로 실행됩니다.

개발자 워크플로의 경우 nuage-openshift-monitor 는 OpenShift Container Platform 구문을 VSP 구성에 매핑하도록 VSD REST API를 실행하여 VSD 오브젝트를 자동으로 생성합니다. 각 클러스터 인스턴스는 Nuage VSP의 단일 도메인에 매핑됩니다. 따라서 특정 기업은 잠재적으로 여러 개의 클러스터 설치(숫자 내 해당 엔터프라이즈에 대한 도메인 인스턴스당 하나)를 설치할 수 있습니다. 각 OpenShift Container Platform 프로젝트는 Nuage VSP에 있는 클러스터 도메인의 영역에 매핑됩니다. nuage-openshift-monitor 는 프로젝트의 추가 또는 삭제가 표시될 때마다 해당 프로젝트에 해당하는 VSDK API 를 사용하여 영역을 인스턴스화하고 해당 영역에 서브넷 블록을 할당합니다. 또한 nuage-openshift-monitor 는 이 프로젝트에 대한 네트워크 매크로 그룹도 만듭니다. 마찬가지로 nuage-openshift-

monitor 는 서비스의 추가 ordeletion을 볼 때마다 서비스 IP에 해당하는 네트워크 매크로를 생성하고 해당 프로젝트의 네트워크 매크로 그룹에 해당 네트워크 매크로를 할당합니다(레이블을 사용하여 사용자 제공 네트워크 매크로 그룹도 지원됨) 해당 서비스에 대한 통신을 활성화합니다.

개발자 워크플로의 경우 영역 내에 생성된 모든 포드는 해당 서브넷 풀에서 IP를 가져옵니다. 서브넷 풀 할당 및 관리는 master -config 파일의 몇 가지 플러그인 특정 매개 변수를 기반으로 nuage-openshift-monitor 를 통해 수행됩니다. 그러나 실제 IP 주소 확인 및 vport 정책 확인은 프로젝트가 생성될 때 인스턴스화되는 도메인/영역을 기반으로 VSD에서 계속 수행됩니다. 초기 서브넷 풀이 소모되면 nuage-openshift-monitor 가 클러스터 CIDR에서 추가 서브넷을 만들어 지정된 프로젝트에 할당합니다.

작업 워크플로의 경우 사용자는 애플리케이션 또는 Pod 사양에서 Nuage recognized 라벨을 지정하여 Pod를 특정 사용자 정의 영역 및 서브넷으로 확인합니다. 그러나 nuage-openshift-monitor 를 통해 개발자 워크플로를 통해 생성된 영역 또는 서브넷의 Pod를 확인하는 데 사용할 수 없습니다.

참고

운영 워크플로에서 관리자는 포드를 특정 영역/서브넷에 매핑하고 OpenShift 엔터티(ACL 규칙, 정책 그룹, 네트워크 매크로 및 네트워크 매크로 그룹) 간 통신을 허용하는 VSD 구성을 사전 생성합니다. Nuage 라벨 사용 방법에 대한 자세한 내용은 Nuage VSP Openshift Integration Guide를 참조하십시오.

vsp-openshift Plug-in

vsp-openshift 네트워킹 플러그인은 각 OpenShift Container Platform 노드의 OpenShift Container Platform 런타임에 의해 호출됩니다. 네트워크 플러그인 init 및 Pod 설정, 해제 및 상태 후크를 구현합니다. vsp-openshift 플러그인은 포드의 IP 주소 할당도 담당합니다. 특히 VRS(전달 엔진)와 통신하고 Pod에 IP 정보를 구성합니다.

5.3.3. OpenShift Container Platform용 Kuryr SDN

Kuryr (또는 구체적으로 Kuryr-Kubernetes)는 CNI 및 OpenStack Neutron 을 사용하여 빌드된 SDN 솔루션입니다. 이 솔루션의 장점은 광범위한 Neutron SDN 백엔드를 사용하고 Kubernetes 포드와 OpenStack VM(가상 시스템) 간에 상호 연결을 제공할 수 있다는 것입니다.

Kuryr-Kubernetes 및 OpenShift Container Platform 통합은 주로 OpenStack VM에서 실행되는 OpenShift Container Platform 클러스터를 위해 설계되었습니다.

5.3.3.1. OpenStack 배포 요구 사항

Kuryr SDN에는 사용할 OpenStack 구성과 관련된 몇 가지 요구 사항이 있습니다. 특히 중요한 요인은 다음과 같습니다.

- 최소 서비스 세트는 Keystone 및 Neutron입니다.
- Octavia 와 호환됩니다.
- 트렁크 포트 확장을 활성화해야 합니다.
- Neutron에서는 Open vSwitch 방화벽 드라이버를 사용해야 합니다.

5.3.3.2. kuryr-controller

Kuryr-controller는 OpenShift Container Platform API가 생성되는 새 Pod를 조사하고 Neutron 리소스를 생성하는 서비스입니다. 예를 들어 포드가 생성되면 kuryr-controller는 를 확인하고 OpenStack Neutron을 호출하여 새 포트를 만듭니다. 그런 다음 해당 포트(또는 VIF)에 대한 정보가 Pod 주석에 저장됩니다. kuryr-controller는 미리 생성된 포트 풀을 사용하여 더 빠른 Pod 생성을 수행할 수 있습니다.

현재 kuryr-controller는 단일 서비스 인스턴스로 실행되어야 하므로 OpenShift Container Platform에서 replicas=1 이 있는 Deployment 로 모델링됩니다. 기본 OpenStack 서비스 API에 액세스해야 합니다.

5.3.3.3. kuryr-cni

Kuryr-cni 컨테이너는 Kuryr-Kubernetes 배포에서 두 가지 역할을 제공합니다. OpenShift Container Platform 노드에 Kuryr CNI 스크립트를 설치하고 구성하고 호스트의 Pod 네트워킹인 kuryr-daemon 서비스를 실행합니다. 즉 kuryr-cni 컨테이너는 모든 OpenShift Container Platform 노드에서 실행해야 하므로 DaemonSet 으로 모델링됩니다.

OpenShift Container Platform CNI는 OpenShift Container Platform 호스트에서 새 Pod를 생성하거나 삭제할 때마다 Kuryr CNI 스크립트를 호출합니다. 이 스크립트는 Docker API에서 로컬 kuryr-cni의 컨테이너 ID를 가져와서 모든 CNI 호출 인수를 전달한 docker exec를 통해 Kuryr CNI 플러그인 바이너리를 실행합니다. 그런 다음 플러그인은 로컬 HTTP 소켓을 통해 kuryr-daemon을 호출하여 다시 모든 매개 변수를 전달합니다.

Kuryr-daemon 서비스는 생성된 Neutron VIF에 대한 Pod 주석 감시를 담당합니다. 지정된 Pod 에 대한 CNI 요청이 수신되면 메모리에 VIF 정보가 이미 있거나 Pod 정의에 주석이 표시될 때까지 기다립니

다. 알려진 모든 네트워킹 작업의 VIF 정보.

5.4. 사용 가능한 라우터 플러그인

OpenShift Container Platform 클러스터의 트래픽을 제어하기 위해 노드에 라우터를 할당할 수 있습니다. OpenShift Container Platform에서는 HAProxy를 기본 라우터로 사용하지만 옵션을 사용할 수 있습니다.

5.4.1. HAProxy 템플릿 라우터

HAProxy 템플릿 라우터 구현은 템플릿 라우터 플러그인의 참조 구현입니다. openshift3/ose-haproxy-router 리포지토리를 사용하여 템플릿 라우터 플러그인과 함께 HAProxy 인스턴스를 실행합니다.

템플릿 라우터에는 다음 두 가지 구성 요소가 있습니다.

- 엔드 포인트 및 경로를 감시하고 변경 사항을 기반으로 HAProxy를 다시 로드하는 래퍼
- 경로 및 엔드포인트를 기반으로 HAProxy 구성 파일을 빌드하는 컨트롤러

참고

HAProxy 라우터 는 버전 1.8.1을 사용합니다.

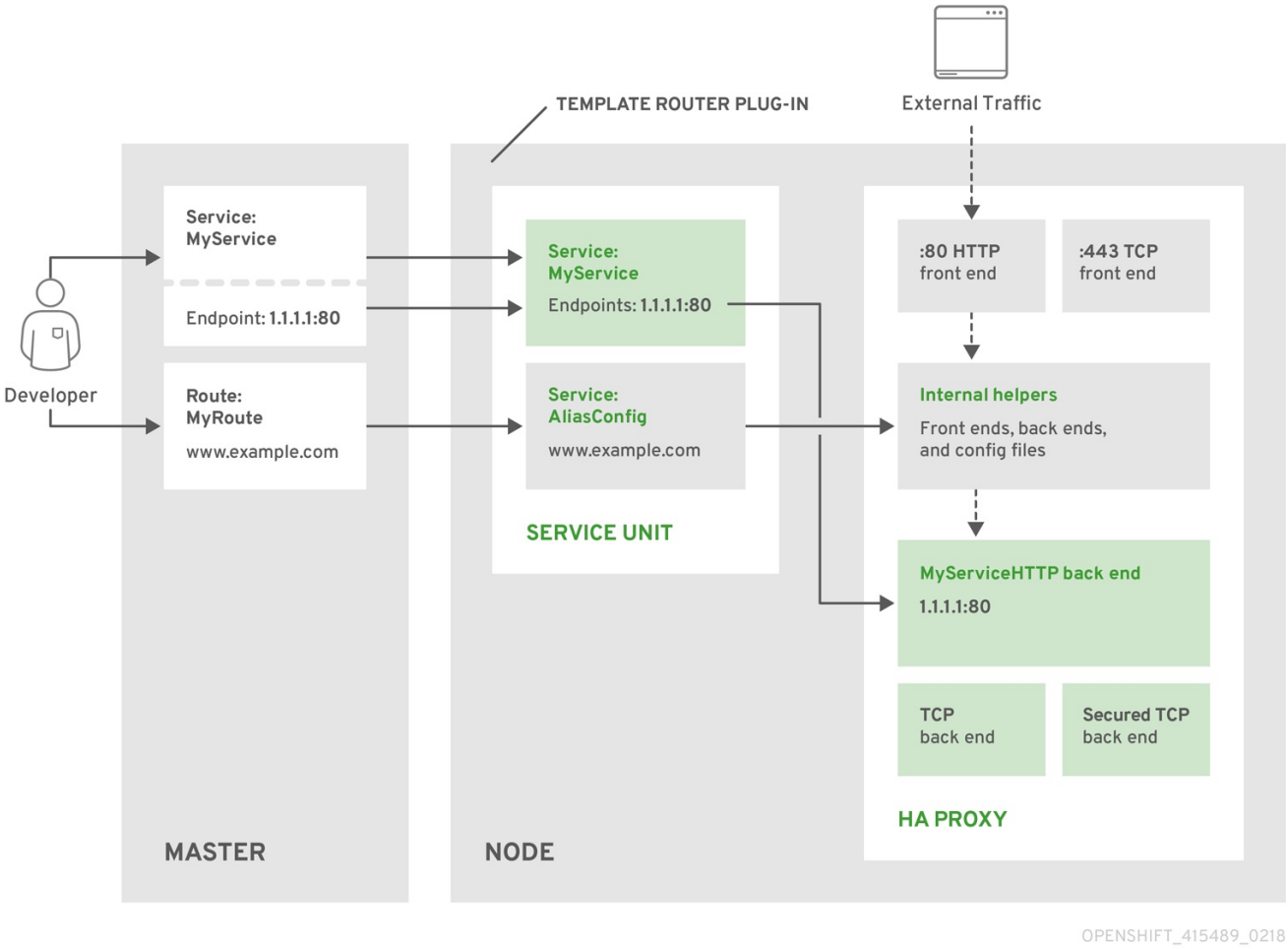
컨트롤러 및 HAProxy는 포드 내부에 있으며 배포 구성에서 관리합니다. 라우터를 설정하는 프로세스는 oc adm router 명령으로 자동화됩니다.

컨트롤러는 HAProxy 상태를 비롯하여 변경 사항이 있는지 경로 및 엔드포인트를 감시합니다. 변경 사항이 감지되면 새 haproxy-config 파일을 빌드하고 HAProxy를 다시 시작합니다. haproxy-config 파일은 라우터의 템플릿 파일과 OpenShift Container Platform의 정보를 기반으로 구성됩니다.

HAProxy 템플릿 파일은 OpenShift Container Platform에서 현재 지원하지 않는 기능을 지원하기 위해 필요에 따라 사용자 지정할 수 있습니다. HAProxy 설명서는 HAProxy 에서 지원하는 모든 기능을 설명합니다.

다음 다이어그램에서는 데이터가 마스터에서 플러그인을 거쳐 **HAProxy** 구성으로 이동하는 방법을 보여줍니다.

그림 **5.4. HAProxy** 라우터 데이터 흐름



HAProxy 템플릿 라우터 지표

HAProxy 라우터는 외부 지표 수집 및 집계 시스템(예: **Prometheus, statsd**)에서 사용할 수 있도록 **Prometheus** 형식으로 지표를 노출하거나 게시합니다. 라우터는 **HAProxy CSV** 형식 지표를 제공하거나 라우터 지표를 전혀 제공하지 않도록 구성할 수 있습니다.

지표는 라우터 컨트롤러와 **5**초마다 **HAProxy**에서 수집됩니다. 라우터 지표 카운터는 라우터가 배포될 때 **0**으로 시작되고 시간이 지남에 따라 증가합니다. **haproxy**가 다시 로드될 때마다 **HAProxy** 지표 카운터가 **0**으로 재설정됩니다. 라우터는 각 프론트엔드, 백엔드 및 서버에 대한 **HAProxy** 통계를 수집합니다. **500**대 이상의 서버가 있는 경우 리소스 사용량을 줄이기 위해 백엔드에는 여러 서버가 있을 수 있으므로 서버 대신 백엔드가 보고됩니다.

통계는 사용 가능한 **HAProxy** 통계의 하위 집합입니다.

*다음 **HAProxy** 지표는 주기적으로 수집되어 **Prometheus** 형식으로 변환됩니다. 프런트 엔드마다 **"F"** 카운터가 수집됩니다. 각 백엔드에 대해 카운터를 수집하고 각 서버에 대해 **"S"** 서버 카운터를 수집하면 됩니다. 그렇지 않으면 각 백엔드에 대해 **"B"** 카운터가 수집되어 서버 카운터가 수집되지 않습니다.*

자세한 내용은 라우터 환경 변수를 참조하십시오.

다음 표에서 다음을 수행합니다.

열 1 - HAProxy CSV 통계의 인덱스

2열

F	프론트 엔드 지표
b	서버 임계값으로 인해 서버 메트릭이 표시되지 않는 경우 백엔드 메트릭은 다음과 같습니다.
B	서버 메트릭을 표시할 때 백엔드 지표
S	서버 지표.

열 3 - 카운터

열 4 - 카운터 설명

인덱스	사용법	카운터	설명
2	bBS	current_queue	서버에 할당되지 않은 현재 대기 중인 요청 수입니다.
4	FbS	current_sessions	현재 활성 세션 수.
5	FbS	max_sessions	확인된 최대 활성 세션 수.
7	FbBS	connections_total	총 연결 수.

8	FbS	bytes_in_total	들어오는 바이트의 현재 합계.
9	FbS	bytes_out_total	현재 보내는 바이트 합계입니다.
13	bS	connection_errors_total	총 연결 오류.
14	bS	response_errors_total	응답 오류 합계.
17	bBS	up	백엔드의 현재 상태(1 = UP, 0 = DOWN).
21	S	check_failures_total	총 실패한 상태 점검 수.
24	S	downtime_seconds_total	총 가동 중지 시간(초)입니다.", nil,
33	FbS	current_session_rate	최근 경과 초당 초당 현재 세션 수입니다.
35	FbS	max_session_rate	초당 최대 관찰 세션 수입니다.
40	FbS	http_responses_total	총 HTTP 응답, 코드 2xx
43	FbS	http_responses_total	총 HTTP 응답, 코드 5xx
60	bS	http_average_response_latency_milliseconds	(밀리초 단위) 마지막 1024개 요청 중.

라우터 컨트롤러는 다음 항목을 스크랩합니다. Prometheus 형식 지표만 사용할 수 있습니다.

이름	설명
template_router_reload_seconds	라우터를 다시 로드하는 데 소요되는 시간을 초 단위로 측정합니다.
template_router_write_config_seconds	라우터 구성을 디스크에 쓰는 데 소요되는 시간을 초 단위로 측정합니다.
haproxy_exporter_up	haproxy의 마지막 스크랩이 성공했습니까.
haproxy_exporter_csv_parse_failures	CSV를 구문 분석하는 동안 오류 수입니다.

haproxy_exporter_scrape_interval	다른 스크랩이 허용되기 전의 시간(초)은 데이터 크기에 비례합니다.
haproxy_exporter_server_threshold	추적된 서버 수 및 현재 임계값.
haproxy_exporter_total_scrapes	현재 전체 HAProxy 스크랩.
http_request_duration_microseconds	마이크로초 단위의 HTTP 요청 대기 시간입니다.
http_request_size_bytes	HTTP 요청 크기(바이트)입니다.
http_response_size_bytes	HTTP 응답 크기(바이트)입니다.
openshift_build_info	OpenShift가 빌드된 major, minor, git commit 및 git 버전으로 레이블이 지정된 상수 '1' 값이 있는 메트릭입니다.
ssh_tunnel_open_count	SSH 터널 총 열린 시도 카운터
ssh_tunnel_open_fail_count	SSH 터널의 카운터가 열린 시도에 실패했습니다

5.5. 포트 전달

5.5.1. 개요

***OpenShift Container Platform**은 **Kubernetes**에 내장된 기능을 활용하여 포드로의 포트 전달을 지원합니다. 이는 **SPDY** 또는 **HTTP /2**와 같은 멀티플렉싱된 스트리밍 프로토콜과 함께 **HTTP** 를 사용하여 구현됩니다.*

*개발자는 **CLI**를 사용하여 포드에 포트 전달을 수행할 수 있습니다. **CLI**는 사용자가 지정한 각 로컬 포트에서 수신 대기하여 설명된 프로토콜을 통해 전달합니다.*

5.5.2. 서버 작업

***Kubelet**은 클라이언트의 포트 전달 요청을 처리합니다. 요청을 수신하면 응답을 업그레이드하고 클라이언트가 포트 전달 스트림을 생성할 때까지 기다립니다. 새 스트림을 수신하면 스트림과 **Pod**의 포트 간에 데이터를 복사합니다.*

*구조적으로 **Pod**의 포트로 전달할 수 있는 옵션이 있습니다. 현재 **OpenShift Container Platform**에서 지원되는 구현은 노드 호스트에서 직접 **nsenter** 를 호출하여 **Pod**의 네트워크 네임스페이스를 입력한 다음 **socat** 을 호출하여 스트림과 **Pod** 포트 간 데이터를 복사합니다. 그러나 사용자 정의 구현에는 **nsenter***

및 socat 을 실행하는 "helper" Pod를 실행할 수 있으므로 이러한 바이너리를 호스트에 설치할 필요가 없습니다.

5.6. 원격 명령

5.6.1. 개요

OpenShift Container Platform은 Kubernetes에 내장된 기능을 활용하여 컨테이너에서 명령 실행을 지원합니다. 이는 SPDY 또는 HTTP /2와 같은 멀티플렉싱된 스트리밍 프로토콜과 함께 HTTP 를 사용하여 구현됩니다.

개발자는 CLI를 사용하여 컨테이너에서 원격 명령을 실행할 수 있습니다.

5.6.2. 서버 작업

Kubelet은 클라이언트의 원격 실행 요청을 처리합니다. 요청을 수신하면 응답이 업그레이드되고 요청 헤더를 평가하여 수신할 스트림(stdin, stdout 및/또는 stderr)을결정하고 클라이언트가 스트림을 생성할 때까지 기다립니다.

Kubelet은 모든 스트림을 수신한 후 컨테이너에서 명령을 실행하여 스트림과 명령의 stdin, stdout 및 stderr 간에 적절하게 복사합니다. 명령이 종료되면 Kubelet은 업그레이드된 연결과 기본 연결을 종료합니다.

구조적으로는 컨테이너에서 명령을 실행할 수 있는 옵션이 있습니다. 현재 OpenShift Container Platform에서 지원되는 구현은 명령을 실행하기 전에 노드 호스트에서 직접 nsenter 를 호출하여 컨테이너의 네임스페이스를 입력합니다. 그러나 사용자 정의 구현에는 docker exec 를 사용하거나 nsenter 를 실행하는 "helper" 컨테이너를 실행하여 nsenter 가 호스트에 설치해야 하는 필수 바이너리가 아닙니다.

5.7. 라우트

5.7.1. 개요

OpenShift Container Platform 경로는 호스트 이름(예: www.example.com) 에 서비스를 노출하므로 외부 클라이언트가 이름으로 연결할 수 있습니다.

호스트 이름의 DNS 확인은 라우팅과 별도로 처리됩니다. 관리자는 OpenShift Container Platform 라우터를 실행 중인 OpenShift Container Platform 노드로 확인되는 DNS 와일드카드 항목을 구성했을 수

있습니다. 다른 호스트 이름을 사용하는 경우 라우터를 실행 중인 노드를 확인하기 위해 해당 DNS 레코드를 독립적으로 수정해야 할 수 있습니다.

각 경로는 이름(63자), 서비스 선택기 및 선택적 보안 구성으로 구성됩니다.

5.7.2. 라우터

OpenShift Container Platform 관리자는 OpenShift Container Platform 클러스터의 노드에 라우터를 배포할 수 있으므로 개발자가 생성한 경로를 외부 클라이언트에서 사용할 수 있습니다. OpenShift Container Platform의 라우팅 계층은 플러그형이며, 기본적으로 여러 라우터 플러그인이 제공되고 지원됩니다.

참고

라우터 구성에 대한 자세한 내용은 클러스터 구성 가이드를 참조하십시오.

라우터는 서비스 선택기를 사용하여 서비스를 지원하는 서비스와 엔드포인트를 찾습니다. 라우터와 서비스 모두 로드 밸런싱을 제공하는 경우 OpenShift Container Platform은 라우터 로드 밸런싱을 사용합니다. 라우터는 서비스의 IP 주소에서 관련 변경 사항을 감지하고 그에 따라 구성을 조정합니다. 이 기능은 사용자 지정 라우터를 통해 API 오브젝트를 외부 라우팅 솔루션으로 변경하는 데 유용합니다.

요청 경로는 호스트 이름의 DNS를 하나 이상의 라우터로 확인하는 것으로 시작됩니다. 제안된 방법은 여러 라우터 인스턴스에서 지원하는 하나 이상의 가상 IP(VIP) 주소를 가리키는 와일드카드 DNS 항목을 사용하여 클라우드 도메인을 정의하는 것입니다. 클라우드 도메인 외부의 이름과 주소를 사용하는 경로에는 개별 DNS 항목을 구성해야 합니다.

라우터보다 VIP 주소가 적으면 주소 수에 해당하는 라우터가 활성 상태이며 나머지는 패시브 입니다. 패시브 라우터는 핫-대기 라우터라고도 합니다. 예를 들어 VIP 주소 2개와 라우터 3개가 있는 경우 "active-active-passive" 구성이 있습니다. 라우터 VIP 구성에 대한 자세한 내용은 High Availability 를 참조하십시오.

경로는 라우터 집합 간에 분할 할 수 있습니다. 관리자는 클러스터 전체에서 분할을 설정할 수 있으며 사용자는 프로젝트에서 네임스페이스에 대한 분할을 설정할 수 있습니다. 운영자가 분할을 통해 여러 라우터 그룹을 정의할 수 있습니다. 그룹의 각 라우터는 트래픽의 하위 집합만 제공합니다.

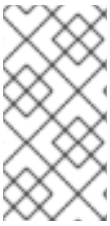
OpenShift Container Platform 라우터는 라우터에 직접 구분 정보를 전달하는 프로토콜을 통해 외부 호스트 이름 매핑 및 서비스 엔드포인트의 부하 분산을 제공합니다. 라우터에서 보낼 위치를 결정하려면 호스트 이름이 프로토콜에 있어야 합니다.

라우터 플러그인은 기본적으로 호스트 포트 **80(HTTP)** 및 **443(HTTPS)**에 바인딩할 수 있다고 가정합니다. 즉, 라우터를 해당 포트가 사용하지 않는 노드에 배치해야 합니다. 또는 **ROUTER_SERVICE_HTTP_PORT** 및 **ROUTER_SERVICE_HTTPS_PORT** 환경 변수를 설정하여 다른 포트에서 수신 대기하도록 라우터를 구성할 수 있습니다.

라우터가 호스트 노드의 포트에 바인딩되므로 라우터가 호스트 네트워킹(기본값)을 사용하는 경우 해당 포트에서 수신 대기하는 하나의 라우터만 각 노드에 있을 수 있습니다. 클러스터 네트워킹은 모든 라우터가 클러스터의 모든 포드에 액세스할 수 있도록 구성됩니다.

라우터는 다음과 같은 프로토콜을 지원합니다.

- **HTTP**
- **HTTPS(SNI 포함)**
- **WebSockets**
- **SNI 포함 TLS**



참고

WebSocket 트래픽은 동일한 경로 규칙을 사용하며 다른 트래픽과 동일한 **TLS** 종료 유형을 지원합니다.

보안 연결을 설정하려면 클라이언트와 서버에 공통된 암호를 협상해야 합니다. 시간이 지남에 따라 더욱 새롭고 안전한 암호를 사용할 수 있게 되며 클라이언트 소프트웨어로 통합됩니다. 이전 클라이언트가 사용되지 않게 되면 더 오래되고 안전하지 않은 암호를 삭제할 수 있습니다. 기본적으로 라우터는 사용 가능한 광범위한 클라이언트를 지원합니다. 원하는 클라이언트를 지원하고 덜 안전한 암호를 포함하지 않는 선택한 암호 집합을 사용하도록 라우터를 구성할 수 있습니다.

5.7.2.1. 템플릿 라우터

템플릿 라우터 는 다음과 같은 기본 라우터 구현에 특정 인프라 정보를 제공하는 라우터 유형입니다.

- 엔드포인트 및 경로를 감시하는 래퍼입니다.
- 사용 가능한 형식으로 저장되는 엔드포인트 및 경로 데이터.
- 내부 상태를 구성 가능 템플릿에 전달하고 템플릿 실행.
- 다시 로드 스크립트 호출.

5.7.3. 사용 가능한 라우터 플러그인

사용 가능한 라우터 플러그인은 사용 가능한 라우터 플러그인 섹션을 참조하십시오.

이러한 라우터를 배포하는 방법은 라우터 배포에서 확인할 수 있습니다.

5.7.4. 고정 세션

고정 세션 구현은 기본 라우터 구성에 따라 달라집니다. 기본 **HAProxy** 템플릿은 소스 **IP**를 기반으로 균형을 조정하는 **balance source** 지시문을 사용하여 고정 세션을 구현합니다. 또한 템플릿 라우터 플러그인에서는 서비스 이름과 네임스페이스를 기본 구현에 제공합니다. 이 명령은 피어 집합 간에 동기화되는 **stick-tables** 구현과 같은 고급 구성에 사용할 수 있습니다.

고정 세션을 사용하면 사용자 세션의 모든 트래픽이 동일한 포드로 이동하여 더 나은 사용자 환경을 만들 수 있습니다. 사용자의 요청을 충족하는 동안 **Pod**는 후속 요청에 사용할 수 있는 데이터를 캐시합니다. 예를 들어 **5**개의 백엔드 포드와 부하 분산 라우터가 있는 클러스터의 경우 동일한 포드가 이를 처리하는 라우터와 관계없이 동일한 웹 브라우저에서 웹 트래픽을 수신하는지 확인할 수 있습니다.

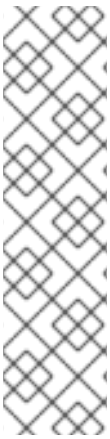
동일한 **Pod**로 라우팅 트래픽을 반환하는 동안은 보장할 수 없습니다. 그러나 **HTTP** 헤더를 사용하여 마지막 연결에서 사용되는 **Pod**를 판별하기 위해 쿠키를 설정할 수 있습니다. 사용자가 애플리케이션에 다른 요청을 보내면 브라우저가 쿠키를 다시 전송하고 라우터는 트래픽을 보낼 위치를 알고 있습니다.

클러스터 관리자는 다른 연결과 별도로 패스스루 경로의 고정을 해제하거나 고정을 완전히 끌 수 있습니다.

기본적으로 패스스루 경로에 대한 고정 세션은 소스 로드 밸런싱 전략을 사용하여 구현됩니다. **ROUTER_TCP_BALANCE_SCHEME**환경 변수를 사용하고 **haproxy.router.openshift.io/balance** 경로

특정 주석을 사용하여 개별 경로에 대해 모든 통과 경로에 대해 기본값을 변경할 수 있습니다.

다른 유형의 경로는 기본적으로 **leastconn** 부하 분산 전략을 사용하며, **ROUTER_LOAD_BALANCE_ALGORITHM** 환경 변수를 사용하여 변경할 수 있습니다. **haproxy.router.openshift.io/balance** 경로 특정 주석을 사용하여 개별 경로에 대해 변경할 수 있습니다.



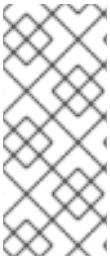
참고

HTTP 트래픽을 볼 수 없기 때문에 통과 경로에서는 쿠키를 설정할 수 없습니다. 대신, 번호는 백엔드를 결정하는 소스 **IP** 주소를 기반으로 계산됩니다.

백엔드가 변경되면 트래픽이 잘못된 서버로 전환되어 스티키를 줄이면서 부하 분산기(소스 **IP** 숨기기)를 사용하는 경우 모든 연결에 대해 동일한 번호가 설정되고 트래픽이 동일한 포드로 전송됩니다.

또한 템플릿 라우터 플러그인에서는 서비스 이름과 네임스페이스를 기본 구현에 제공합니다. 이는 피어 집합 간에 동기화되는 **stick-tables** 구현과 같은 고급 구성에 사용할 수 있습니다.

이 라우터 구현에 대한 특정 구성은 라우터 컨테이너의 **/var/lib/haproxy/conf** 디렉터리에 있는 **haproxy-config.template** 파일에 저장됩니다. 파일을 사용자 지정할 수 있습니다.



참고

소스 부하 분산 전략은 외부 클라이언트 **IP** 주소를 구분하지 않습니다. **NAT** 구성으로 인해 원래 **IP** 주소(**HAProxy** 원격)는 동일합니다. **HAProxy** 라우터가 **hostNetwork: true**로 실행되지 않는 한 모든 외부 클라이언트는 단일 포드로 라우팅됩니다.

5.7.5. 라우터 환경 변수

이 섹션에 설명된 모든 항목에 대해 라우터의 배포 구성에 환경 변수를 설정하여 구성을 변경하거나 **oc set env** 명령을 사용할 수 있습니다.

```
$ oc set env <object_type>/<object_name> KEY1=VALUE1 KEY2=VALUE2
```

예를 들면 다음과 같습니다.

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1 ROUTER_LOG_LEVEL=debug
```

표 5.2. 라우터 환경 변수

Variable	Default	설명
DEFAULT_CERTIFICATE		PEM 형식의 TLS 서버 인증서를 노출하지 않는 경로에 사용할 기본 인증서의 내용입니다.
DEFAULT_CERTIFICATE_DIR		***tls.crt*** 라는 파일이 포함된 디렉터리의 경로입니다.***tls.crt*** 가 개인 키가 포함된 PEM 파일이 아닌 경우 먼저 동일한 디렉터리에 tls.key라는 파일과 결합됩니다. 그런 다음 PEM-format 콘텐츠가 기본 인증서로 사용됩니다. **DEFAULT_CERTIFICATE** 또는 **DEFAULT_CERTIFICATE_PATH** 가 지정되지 않은 경우에만 사용됩니다.
DEFAULT_CERTIFICATE_PATH		PEM 형식의 TLS 서버 인증서를 노출하지 않는 경로에 사용할 기본 인증서 경로입니다. **DEFAULT_CERTIFICATE** 가 지정되지 않은 경우에만 사용됩니다.
EXTENDED_VALIDATION	**true**	**true인** 경우 라우터는 인증서가 구조적으로 올바른지 확인합니다. CA에 대한 인증서를 확인하지 않습니다. 테스트를 끄려면 **false** 를 설정합니다.
NAMESPACE_LABELS		감시할 네임스페이스에 적용할 레이블 선택기는 모두 있음을 의미합니다.
PROJECT_LABELS		조사할 프로젝트에 적용할 레이블 선택기이며, emtpy는 모두 를 의미합니다.
RELOAD_SCRIPT		라우터를 다시 로드하는 데 사용할 다시 로드 스크립트의 경로입니다.
ROUTER_ALLOWED_DOMAINS		경로의 호스트 이름만 포함할 수 있는 쉼표로 구분된 도메인 목록입니다. 도메인의 모든 하위 도메인을 사용할 수 있습니다. **ROUTER_DENIED_DOMAINS** 옵션은 이 옵션에 제공된 값을 재정의합니다. 설정된 경우 허용된 도메인 외부의 모든 항목이 거부됩니다.
ROUTER_BACKEND_PROCESS_ENDPOINTS		템플릿 기능 processEndpointsForAlias를 사용하는 동안 끝점을 처리하는 방법을 지정하는 문자열입니다. 유효한 값은 ["shuffle", ""]입니다. "shuffle"은 모든 호출 시 요소를 임의로 지정합니다. 기본 동작은 사전 결정된 순서로 반환됩니다.
ROUTER_BIND_PORTS_AFTER_SYNC	**false**	**true** 또는 **TRUE** 로 설정하면 라우터가 완전히 동기화된 상태가 될 때까지 모든 포트에 바인딩되지 않습니다. 'true' 또는 'TRUE'로 설정되지 않은 경우 라우터가 포트에 바인딩되고 즉시 요청을 처리하지만 로드되지 않은 경로가 있을 수 있습니다.
ROUTER_COOKIE_NAME		내부적으로 생성된 기본 이름을 재정의하는 쿠키 이름을 지정합니다. 이름은 대문자와 소문자, 숫자, '_', '-'의 조합으로 구성해야 합니다. 기본값은 경로의 해시된 내부 키 이름입니다.

Variable	Default	설명
ROUTER_COMPRESSION_MIME	"text/html text/plain text/css"	압축할 공백으로 구분된 MIME 유형 목록입니다.
ROUTER_DENIED_DOMAINS		경로의 호스트 이름을 포함할 수 없는 쉼표로 구분된 도메인 목록입니다. 도메인의 하위 도메인도 사용할 수 없습니다. **ROUTER_ALLOWED_DOMAINS** 옵션을 재정의합니다.
ROUTER_ENABLE_COMPRESSION		**true** 또는 **TRUE인 경우** 가능한 경우 응답을 압축합니다.
ROUTER_LISTEN_ADDR	0.0.0.0:1936	라우터 지표에 대한 수신 대기 주소를 설정합니다.
ROUTER_LOG_LEVEL	경고	syslog 서버로 전송할 로그 수준입니다.
ROUTER_MAX_CONNECTIONS	20000	최대 동시 연결 수입니다.
ROUTER_METRICS_HAPROXY_SERVER_THRESHOLD	500	
ROUTER_METRICS_HAPROXY_EXPORTED		CSV 형식으로 수집된 지표입니다. 예를 들어 **defaultSelectedMetrics** = []int{2, 4, 5, 7, 8, 9, 13, 14, 17, 21, 24, 33, 35, 40, 43, 60}
ROUTER_METRICS_HAPROXY_BASE_SCRAPE_INTERVAL	5s	
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	
ROUTER_METRICS_TYPE	haproxy	HAProxy 라우터에 대한 지표를 생성합니다(haproxy가 지원되는 유일한 값임)
ROUTER_OVERRIDE_DOMAINS		쉼표로 구분된 도메인 이름 목록입니다. 경로의 도메인 이름이 경로의 호스트와 일치하는 경우 호스트 이름은 무시되고 **ROUTER_SUBDOMAIN** 에 정의된 패턴이 사용됩니다.

Variable	Default	설명
ROUTER_OVERRIDE_HOSTNAME		**true** 를 설정하면 **ROUTER_SUBDOMAIN** 에서 템플릿을 사용하여 경로의 spec.host 값을 재정의합니다.
ROUTER_SERVICE_HTTPS_PORT	443	HTTPS 요청을 수신 대기할 포트입니다.
ROUTER_SERVICE_HTTP_PORT	80	HTTP 요청을 수신 대기할 포트입니다.
ROUTER_SERVICE_NAME	공용	라우터가 경로 상태에서 식별하는 이름입니다.
ROUTER_CANONICAL_HOSTNAME		경로 상태에 표시된 라우터의 (선택 사항) 호스트 이름입니다.
ROUTER_SERVICE_NAMESPACE		라우터가 경로 상태에서 식별하는 네임스페이스입니다. **ROUTER_SERVICE_NAME** 이 사용되는 경우 필수 항목입니다.
ROUTER_SERVICE_NO_SNI_PORT	10443	백엔드 통신에 대한 일부 프런트 엔드의 내부 포트(아래 참조).
ROUTER_SERVICE_SNI_PORT	10444	백엔드 통신에 대한 일부 프런트 엔드의 내부 포트(아래 참조).
ROUTER_SUBDOMAIN		spec.host 없이 경로에 대한 호스트 이름을 생성하는 데 사용해야 하는 템플릿(예: ${name}-${namespace}.myapps.mycompany.com).
ROUTER_SYSLOG_ADDRESS		로그 메시지를 보낼 주소입니다. 비어 있는 경우 비활성화됨.
ROUTER_SYSLOG_FORMAT		설정된 경우 기본 라우터 구현에서 사용하는 기본 로그 형식을 재정의합니다. 해당 값은 기본 라우터 구현의 사양을 준수해야 합니다.
ROUTER_TCP_BALANCE_SCHEME	소스	부하 분산 전략. 패스쓰루(Pass-through) 경로를 위한 여러 엔드포인트의 경우. 사용 가능한 옵션은 **source**,**roundrobin** 또는 **leastconn** 입니다.

Variable	Default	설명
ROUTER_THREADS		haproxy 라우터의 스레드 수를 지정합니다.
ROUTER_LOAD_BALANCE_ALGORITHM	leastconn	여러 엔드포인트가 있는 경로의 경우 로드 밸런싱 전략입니다. 사용 가능한 옵션은 **source**, **roundrobin**, **leastconn**입니다.
ROUTE_LABELS		감시할 경로에 적용할 레이블 선택기는 모두 있음을 의미합니다.
STATS_PASSWORD		라우터 구현에서 지원하는 경우 라우터 통계에 액세스하는 데 필요한 암호입니다.
STATS_PORT		통계를 노출하는 포트(라우터 구현에서 지원하는 경우). 설정되지 않으면 통계가 노출되지 않습니다.
STATS_USERNAME		라우터 구현에서 지원하는 경우 라우터 통계에 액세스해야 하는 사용자 이름.
TEMPLATE_FILE	**/var/lib/haproxy/conf/custom/haproxy-config-custom.template**	HAProxy 템플릿 파일(컨테이너 이미지)의 경로입니다.
ROUTER_USE_PROXY_PROTOCOL		**true** 또는 **TRUE** 로 설정하면 HAProxy는 포트 80 또는 포트 443에서 **PROXY** 프로토콜을 사용하도록 들어오는 연결을 예상합니다. 로드 밸런서에서 프로토콜을 지원하는 경우 소스 IP 주소는 로드 밸런서를 통과할 수 있습니다(예: Amazon ELB).
ROUTER_ALLOW_WILDCARD_ROUTES		**true** 또는 **TRUE** 로 설정하면 라우터 승인 검사를 통과하는 와일드카드 정책이 **Subdomain** 인 모든 경로가 HAProxy 라우터에서 서비스를 제공합니다.
ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK		네임스페이스 소유권 정책을 완화하려면 **true** 로 설정합니다.
ROUTER_STRICT_SNI		strict-sni
ROUTER_CIPHERS	중간	바인딩에서 지원하는 암호 집합을 지정합니다.

Variable	Default	설명
ROUTER_HAPROXY_CONFIG_MANAGER		**true** 또는 **TRUE** 로 설정하면 특정 유형의 경로를 관리하고 HAproxy 라우터 다시 로드 양을 줄일 수 있는 HAproxy를 사용하여 동적 구성 관리자를 활성화합니다. 자세한 내용은 동적 구성 관리자 사용을 참조하십시오.
COMMIT_INTERVAL	3600	동적 구성 관리자의 변경 사항을 커밋하는 빈도를 지정합니다. 이로 인해 기본 템플릿 라우터 구현이 구성을 다시 로드합니다.
ROUTER_BLUEPRINT_ROUTE_NAMESPACE		동적 구성 관리자의 청사진 역할을 하는 경로가 포함된 네임스페이스로 설정합니다. 이를 통해 동적 구성 관리자는 사용자 지정 주석, 인증서 또는 구성 파일을 사용하여 사용자 지정 경로를 지원할 수 있습니다.
ROUTER_BLUEPRINT_ROUTE_LABELS		블루프린트 경로 네임스페이스의 경로에 적용하려면 레이블 선택기로 설정합니다. 이를 통해 동적 구성 관리자의 청사진 역할을 할 수 있는 네임스페이스에서 경로를 지정할 수 있습니다.
ROUTER_BLUEPRINT_ROUTE_POOL_SIZE	10	동적 구성 관리자가 관리하는 각 경로 청사진에 대해 사전 할당된 풀의 크기를 지정합니다. 이는 청사진 경로에서 **router.openshift.io/pool-size** 주석을 사용하여 개별 경로를 기반으로 재정의할 수 있습니다.
ROUTER_MAX_DYNAMIC_SERVERS	5	동적 구성 관리자가 사용할 각 경로에 추가된 최대 동적 서버 수를 지정합니다.

참고

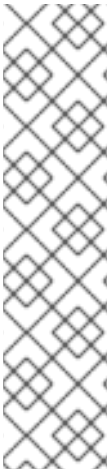
동일한 시스템에서 여러 라우터를 실행하려면 라우터가 수신 대기 중인 포트, ROUTER_SERVICE_ SNI_PORT 및 ROUTER_SERVICE_ NO_SNI_PORT 를 변경해야 합니다. 이 포트는 시스템에서 고유한 포트만 사용할 수 있습니다. 이러한 포트는 외부에 노출되지 않습니다.

라우터 시간 제한 변수

***TimeUnits**는 다음과 같이 표시됩니다. **us** *(마이크로초), **ms** (밀리초, 기본값), **s** (초), **m** (분), **h** *(시간), **d** (일).*

*정규 표현식은 **[1-9][0-9]*(us\|ms\|s\|m\|h\|d)**입니다.*

ROUTER_BACKEND_CHECK_INTERVAL	5000ms	백엔드의 후속 활성 검사 사이의 시간입니다.
ROUTER_CLIENT_FIN_TIMEOUT	1s	경로에 연결된 클라이언트의 TCP FIN 시간 제한 기간을 제어합니다. 지정된 시간 내에 연결을 닫기 위해 전송된 FIN이 응답하지 않으면 HAProxy가 연결을 종료합니다. 낮은 값으로 설정하면 문제가 없으며 라우터에서 더 적은 리소스를 사용합니다.
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	클라이언트가 데이터를 승인하거나 보내야 하는 시간입니다.
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	최대 연결 시간.
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	라우터에서 경로를 지원하는 pod로의 TCP FIN 시간 초과를 제어합니다.
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	서버에서 데이터를 승인하거나 보내야 하는 시간입니다.
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP 또는 WebSocket 연결이 열린 상태로 유지되는 동안의 시간입니다. websockets/tcp 연결이 있고 HAProxy가 다시 로드될 때마다 이전 HAProxy 프로세스는 해당 기간 동안 유지됩니다.
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	새 HTTP 요청이 표시될 때까지 대기할 최대 시간을 설정합니다. 이 값을 너무 낮게 설정하면 작은 **keepalive** 값을 예상하지 못하는 브라우저 및 애플리케이션에 문제가 발생할 수 있습니다. 더하기. 자세한 내용은 아래 참고 상자를 참조하십시오.
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 요청 전송에 걸리는 시간입니다.
RELOAD_INTERVAL	5s	새 변경 사항을 수락하도록 라우터를 다시 로드할 수 있는 최소 빈도입니다.
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy 메트릭 수집에 대한 시간 제한입니다.



참고

일부 유효한 시간 제한 값은 예상되는 특정 시간 초과가 아니라 특정 변수의 합계일 수 있습니다.

예를 들면 다음과 같습니다. **ROUTER_SLOWLORIS_HTTP_KEEPALIVE** 는 시간 초과 **http-keep-alive** 를 조정하며 기본적으로 **300s** 로 설정되어 있지만 **haproxy**는 **5s** 로 설정된 **tcp-request inspect-delay** 도 대기합니다. 이 경우 전체 시간 초과는 **300s + 5s**입니다.

5.7.6. 로드 밸런싱 전략

경로에 여러 엔드포인트가 있는 경우 **HAProxy**는 선택한 로드 밸런싱 전략을 기반으로 엔드포인트에 요청을 배포합니다. 이는 세션의 첫 번째 요청과 같이 지속성 정보를 사용할 수 없는 경우에 적용됩니다.

전략은 다음 중 하나일 수 있습니다.

- **roundrobin:** 각 엔드포인트는 가중치에 따라 차례로 사용됩니다. 이 알고리즘은 서버의 처리 시간이 동일하게 분산되어 있을 때 가장 원활하고 공정하게 알고리즘됩니다.

- **leastconn:** 연결 수가 가장 적은 엔드포인트에서 요청을 수신합니다. 라운드 로빈은 여러 엔드포인트에 동일한 개수의 커넥션 수가 같으면 수행됩니다. **LDAP, SQL, TSE** 등과 같이 매우 긴 세션이 예상되는 경우 이 알고리즘을 사용합니다. 일반적으로 **HTTP**와 같은 짧은 세션을 사용하는 프로토콜에는 사용되지 않습니다.

- **출처:** 소스 **IP** 주소는 해시되고 실행 중인 서버의 총 가중치로 나누어 요청을 받을 서버를 지정합니다. 이렇게 하면 서버가 다운되거나 가동되지 않는 한 동일한 클라이언트 **IP** 주소가 항상 동일한 서버에 도달할 수 있습니다. 실행 중인 서버 수로 인해 해시 결과가 변경되면 많은 클라이언트가 다른 서버로 전달됩니다. 이 알고리즘은 일반적으로 **passthrough** 경로와 함께 사용됩니다.

ROUTER_TCP_BALANCE_SCHEME 환경 변수는 통과 경로의 기본 전략을 설정합니다. **ROUTER_LOAD_BALANCE_ALGORITHM** 환경 변수는 나머지 경로의 라우터에 대한 기본 전략을 설정합니다. 경로별 주석 인**haproxy.router.openshift.io/balance** 를 사용하여 특정 경로를 제어할 수 있습니다.

5.7.7. HAProxy Strict SNI

기본적으로 호스트가 **HTTPS** 또는 **TLS SNI** 요청의 경로를 확인하지 않으면 기본 인증서가 **503** 응답

의 일부로 호출자로 반환됩니다. 이렇게 하면 기본 인증서가 노출되고 잘못된 인증서가 사이트에 제공되므로 보안 문제가 발생할 수 있습니다. **HAProxy**의 기본 인증서 사용을 억제하는 **HAProxy strict-sni** 옵션입니다.

ROUTER_STRICT_SNI 환경 변수는 바인드 처리를 제어합니다. **true** 또는 **TRUE** 로 설정하면 **HAProxy** 바인딩에**strict-sni** 가 추가됩니다. 기본 설정은 **false** 입니다.

옵션은 나중에 라우터를 만들거나 추가할 때 설정할 수 있습니다.

```
$ oc adm router --strict-sni
```

그러면 **ROUTER_STRICT_SNI=true**가 설정됩니다.

5.7.8. 라우터 암호 제품군

각 클라이언트(예: **Chrome 30** 또는 **Java8**)에는 라우터에 안전하게 연결하는 데 사용되는 암호 집합이 포함되어 있습니다. 연결이 완료되려면 라우터에 암호가 하나 이상 있어야 합니다.

표 **5.3.** 라우터 암호 프로필

Profile	가장 오래된 호환 클라이언트
Modern	Firefox 27, Chrome 30, IE 11 on Windows 7, Edge, Opera 17, Safari 9, Android 5.0, Java 8
중간	Firefox 1, Chrome 1, IE 7, Opera 5, Safari 1, Windows XP IE8, Android 2.3, Java 7
Old	Windows XP IE6, Java 6

자세한 내용은 보안/서버 사이드 **TLS** 참조 가이드를 참조하십시오.

기본적으로 라우터는 중간 프로필을 선택하고 이 프로필을 기반으로 암호를 설정합니다. 프로필을 선택하면 암호만 설정됩니다. **TLS** 버전은 프로필에서 관리하지 않습니다.

라우터를 만들 때 **--ciphers** 옵션을 사용하거나 기존 라우터의 **modern,intermediate** 또는 **old** 값으로 **ROUTER_CIPHERS** 환경 변수를 변경하여 다른 프로필을 선택할 수 있습니다. 또는 **":"**로 구분된 암호 집합을 제공할 수 있습니다. 암호는 다음을 통해 표시되는 세트의 암호여야 합니다.

```
openssl ciphers
```

5.7.9. 경로 호스트 이름

서비스를 외부에서 노출하기 위해 **OpenShift Container Platform** 경로를 사용하면 외부에서 연결할 수 있는 호스트 이름과 서비스를 연결할 수 있습니다. 이 에지 호스트 이름은 트래픽을 서비스로 라우팅하는 데 사용됩니다.

다른 네임스페이스의 여러 경로가 동일한 호스트를 클레임하면 가장 오래된 경로가 성공하고 네임스페이스에 대해 클레임합니다. 경로 필드가 다른 추가 경로가 동일한 네임스페이스에 정의된 경우 해당 경로가 추가됩니다. 동일한 경로가 있는 여러 경로가 사용되는 경우 가장 오래된 경로가 우선합니다.

이 동작의 결과는 호스트 이름에 대한 두 개의 경로, 즉 이전 경로와 최신 경로가 있는 경우입니다. 다른 두 경로를 만들 때 생성된 동일한 호스트 이름에 대한 경로가 있는 경우 이전 경로를 삭제하면 호스트 이름에 대한 클레임이 더 이상 적용되지 않습니다. 이제 다른 네임스페이스에서 호스트 이름을 클레임하고 클레임이 손실됩니다.

지정된 호스트가 있는 경로:

```
apiVersion: v1
kind: Route
metadata:
  name: host-route
spec:
  host: www.example.com 1
  to:
    kind: Service
    name: service-name
```

1

서비스를 노출하는 데 사용되는 외부에서 연결할 수 있는 호스트 이름을 지정합니다.

호스트 없는 경로:

```
apiVersion: v1
kind: Route
```

```
metadata:
  name: no-route-hostname
spec:
  to:
    kind: Service
    name: service-name
```

호스트 이름이 경로 정의의 일부로 제공되지 않으면 **OpenShift Container Platform**에서 자동으로 이름을 생성합니다. 생성된 호스트 이름은 다음과 같은 형식입니다.

```
<route-name>[-<namespace>].<suffix>
```

다음 예는 **mynamespace** 네임스페이스에 호스트를 추가하지 않고 위의 경로 구성에 대한 **OpenShift Container Platform** 생성 호스트 이름을 보여줍니다.

생성된 호스트 이름

```
no-route-hostname-mynamespace.router.default.svc.cluster.local 1
```

1

생성된 호스트 이름 접미사는 기본 라우팅 하위 도메인 **router.default.svc.cluster.local** 입니다.

클러스터 관리자는 해당 환경의 기본 라우팅 하위 도메인으로 사용된 접미사를 사용자 지정할 수도 있습니다.

5.7.10. 경로 유형

경로는 보안 또는 비보안일 수 있습니다. 보안 경로는 여러 유형의 **TLS** 종료를 사용하여 클라이언트에 인증서를 제공하는 기능을 제공합니다. 라우터는 에지,패스스루 및 재암호화 종료를 지원합니다.

보안되지 않은 경로 오브젝트 **YAML** 정의

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name
```

보안되지 않은 경로는 키 또는 인증서가 필요하지 않으므로 구성하는 가장 간단한 경로이지만 보안 경로는 개인용 연결을 위한 보안을 제공합니다.

보안 경로는 경로의 **TLS** 종료를 지정하는 경로입니다. 사용 가능한 종료 유형은 다음과 같습니다.

5.7.10.1. 경로 기반 경로

경로 기반 경로는 동일한 호스트 이름을 사용하여 여러 경로를 각각 다른 경로로 제공할 수 있도록 **URL**과 비교할 수 있는 경로 구성 요소를 지정합니다(경로에 대한 트래픽이 **HTTP** 기반이어야 함). 라우터는 가장 구체적인 경로를 기반으로 하는 경로와 일치해야 합니다. 그러나 라우터 구현에 따라 달라집니다. 호스트 이름과 경로는 백엔드 서버로 전달되므로 요청에 성공적으로 응답할 수 있습니다. 예를 들어 라우터로 이동하는 **http://example.com/foo/** 에 대한 요청으로 인해 **Pod**에 **http://example.com/foo/** 에 대한 요청이 표시됩니다.

다음 표에서는 경로 및 액세스 가능성을 보여줍니다.

표 **5.4.** 경로 가용성

경로	비교 대상	액세스 가능
www.example.com/test	www.example.com/test	있음
	www.example.com	없음
www.example.com/test 및 www.example.com	www.example.com/test	있음
	www.example.com	있음

경로	비교 대상	액세스 가능
www.example.com	*www.example.com/test*	예 (경로가 아닌 호스트에 의해 결정됨)
	www.example.com	있음

경로가 있는 보안되지 않은 경로:

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1

경로는 경로 기반 라우터에 대해 추가된 유일한 속성입니다.

참고

*라우터가 해당 경우 **TLS**를 종료하지 않고 요청 콘텐츠를 읽을 수 없기 때문에 패스스루 **TLS**를 사용할 때 경로 기반 라우팅을 사용할 수 없습니다.*

5.7.10.2. 보안 경로

*보안 경로는 경로의 **TLS** 종료를 지정하고 선택적으로 키와 인증서를 제공합니다.*

참고

OpenShift Container Platform의 **TLS** 종료는 사용자 정의 인증서를 제공하기 위해 **SNI** 를 사용합니다. 포트 **443**에서 수신된 **SNI** 이외의 트래픽은 **TLS** 종료 및 기본 인증서 (요청된 호스트 이름과 일치하지 않을 수 있으므로 유효성 검사 오류가 발생할 수 있음)를 사용하여 처리됩니다.

보안 경로는 다음 세 가지 유형의 보안 **TLS** 종료를 사용할 수 있습니다.

에지 종료

에지 종료를 사용하면 대상에 트래픽을 프록시하기 전에 라우터에서 **TLS**가 종료됩니다. **TLS** 인증서는 라우터의 프런트 엔드에서 제공하므로 경로에 구성해야 합니다. 그렇지 않으면 **TLS** 종료에 라우터의 기본 인증서를 사용합니다.

에지 종료를 사용하여 보안 경로

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination: edge (3)
    key: |- (4)
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |- (5)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |- (6)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

1 2

63자로 제한되는 개체의 이름입니다.

3

termination 필드는 에지 종료를 위한 에지입니다.

4

key 필드는 PEM 형식 키 파일의 내용입니다.

5

certificate 필드는 PEM 형식 인증서 파일의 내용입니다.

6

검증을 위해 인증서 체인을 설정하는 데 선택적 CA 인증서가 필요할 수 있습니다.

TLS가 라우터에서 종료되므로 내부 네트워크를 통한 라우터에서 엔드포인트로의 연결은 암호화되지 않습니다.

에지 종료 경로는 비보안 체계(HTTP)의 트래픽을 비활성화, 허용 또는 리디렉션할 수 있는 비보안 EdgeTerminationPolicy 를 지정할 수 있습니다. insecureEdgeTerminationPolicy 에 허용되는 값은 다음과 같습니다. 없음 또는 비어 있음(비활성화된 경우), 허용 또는 리디렉션. 기본 insecureEdgeTerminationPolicy 는 비보안 스키마에서 트래픽을 비활성화하는 것입니다. 일반적인 사용 사례는 콘텐츠를 보안 체계를 통해 제공할 수 있지만 비보안 스키마를 통해 자산(예: 이미지, 스타일시트 및 javascript)을 제공하는 것입니다.

HTTP 트래픽을 허용하는 에지 종료를 사용하여 보안 경로

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-allow-insecure 1
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name 2
  tls:
```

```
    termination:                edge  (3)
    insecureEdgeTerminationPolicy: Allow  (4)
    [ ... ]
```

(1) (2)

63자로 제한되는 개체의 이름입니다.

(3)

termination 필드는 에지 종료를 위한 에지입니다.

(4)

비보안 스키마 HTTP 에서 보낸 요청을 허용하는 비보안 정책입니다.

HTTP 트래픽을 HTTPS로 리디렉션하는 에지 종료를 사용하여 보안 경로

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-redirect-insecure  (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name  (2)
  tls:
    termination:                edge  (3)
    insecureEdgeTerminationPolicy: Redirect  (4)
    [ ... ]
```

(1) (2)

63자로 제한되는 개체의 이름입니다.

(3)

termination 필드는 에지 종료를 위한 에지입니다.

4

비보안 스키마 HTTP 에서 전송된 요청을 보안 스키마 HTTPS 로 리디렉션하는 비보안 정책입니다.

패스스루 종료

패스스루 종료를 사용하면 암호화된 트래픽이 라우터에서 TLS 종료를 제공하지 않고 바로 대상으로 전송됩니다. 따라서 키 또는 인증서가 필요하지 않습니다.

패스스루 종료를 사용하는 보안 경로

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name 2
  tls:
    termination: passthrough 3
```

1 2

63자로 제한되는 개체의 이름입니다.

3

termination 필드는 passthrough로 설정됩니다. 다른 암호화 필드는 필요하지 않습니다.

대상 Pod는 끝점의 트래픽에 대한 인증서를 제공해야 합니다. 현재 이 방법은 클라이언트 인증서(양방향 인증이라고도 함)를 지원할 수 있는 유일한 방법입니다.

참고

passthrough 경로에는 **insecureEdgeTerminationPolicy** 도 있을 수 있습니다. 유일한 유효한 값은 **None** (또는 비어 있거나 비활성화된 경우) 또는 리디렉션 입니다.

재암호화 종료

재암호화는 에지 종료의 변형으로, 라우터에서 인증서를 사용하여 **TLS**를 종료한 다음 다른 인증서가 있을 수 있는 엔드포인트에 대한 연결을 다시 암호화합니다. 따라서 내부 네트워크를 통해서도 전체 연결 경로가 암호화됩니다. 라우터는 상태 점검을 사용하여 호스트의 신뢰성을 결정합니다.

재암호화 종료를 사용하여 보안 경로

```
apiVersion: v1
kind: Route
metadata:
  name: route-pt-secured (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1) (2)

63자로 제한되는 개체의 이름입니다.

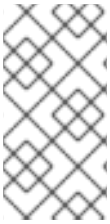
(3)

termination 필드는 **reencrypt**로 설정됩니다. 다른 필드는 에지 종료와 같습니다.

4

재암호화에 필요합니다. **destinationCACertificate** 는 엔드포인트 인증서의 유효성을 검사하고 라우터에서 대상 포드로의 연결을 보호하는 **CA** 인증서를 지정합니다. 서비스에서 서비스 서명 인증서를 사용 중이거나 관리자가 라우터의 기본 **CA** 인증서를 지정하고 서비스에 해당 **CA**에서 서명한 인증서가 있는 경우 이 필드를 생략할 수 있습니다.

destinationCACertificate 필드를 비워 두면 라우터는 서비스 제공 인증서에 대해 생성된 인증 기관을 자동으로 활용하고 모든 **Pod**에 **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** 로 삽입됩니다. 이를 통해 경로에 대한 인증서를 생성하지 않고도 엔드 투 엔드 암호화를 활용하는 새 경로가 허용됩니다. 이는 관리자가 허용하지 않는 한 **destinationCACertificate** 를 허용하지 않을 수 있는 사용자 지정 라우터 또는 **F5** 라우터에 유용합니다.



참고

재암호화 경로에는 **edge** 종료 경로와 동일한 값이 있는 **insecureEdgeTerminationPolicy** 가 있을 수 있습니다.

5.7.11. 라우터 공유

OpenShift Container Platform에서 각 경로에는 **metadata** 필드에 여러 라벨이 있을 수 있습니다. 라우터는 선택기 (선택식이라고도 함)를 사용하여 제공할 전체 경로 풀에서 경로의 하위 집합을 선택합니다. 선택 표현식에는 경로의 네임스페이스에 라벨이 포함될 수도 있습니다. 선택한 경로는 라우터 **shard** 를 형성합니다. 경로와 독립적으로 라우터 **shard** 를 생성 및 수정할 수 있습니다.

이 설계는 기존의 샤딩뿐만 아니라 겹치는 샤딩 을 지원합니다. 기존의 분할에서는 선택으로 인해 중복 세트가 발생하지 않으며 경로는 정확히 하나의 **shard**에 속합니다. 중복된 샤딩에서 선택 결과 겹치는 세트와 경로가 여러 다른 **shard**에 속할 수 있습니다. 예를 들어 단일 경로는 **SLA=high shard(SLA=** 중간 또는 **SLA=** 하위 **shard**가 아님)와 **geo= west shard(geo= east shard**가 아님)에 속할 수 있습니다.

중복된 분할의 또 다른 예는 경로의 네임스페이스를 기반으로 선택하는 라우터 세트입니다.

라우터	선택	네임스페이스
router-1	**A* – J***	**A*, B*, C*, D*, E*, F*, G*, H*, I*, J***
라우터-2	**K* – T***	**K*, L*, M*, N*, O*, P*, Q*, R*, S*, T***
router-3	**Q* – Z***	**Q*, R*, S*, T*, U*, V*, W*, X*, Y*, Z***

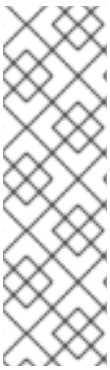
네임스페이스 **Q*, R*, S *, T*** 에 있는 **router-2** 및 **router -3** 경로 모두. 이 예제를 기존 샤딩으로 변경하기 위해 라우터**-2** 를 **K*cd - P *** 로 변경하여 중복을 제거할 수 있습니다.

라우터가 분할되면 지정된 경로가 그룹의 **0**개 이상의 라우터에 바인딩됩니다. 경로 바인딩은 **shard**에서 경로의 고유성을 보장합니다. 고유성을 사용하면 동일한 경로의 보안 및 비보안 버전이 단일 **shard** 내에 존재할 수 있습니다. 이는 이제 경로에 생성된 수명 주기가 생성되어 활성으로 바인드됨을 의미합니다.

분할된 환경에서 **shard**를 도달하는 첫 번째 경로는 재시작 시에도 무기한 존재할 권리가 있습니다.

녹색/파란색 배포 중에 여러 라우터에서 경로를 선택할 수 있습니다. **OpenShift Container Platform** 애플리케이션 관리자는 한 버전의 애플리케이션에서 다른 버전으로 트래픽을 복사한 다음 이전 버전을 종료할 수 있습니다.

관리자는 클러스터 수준 및 프로젝트/네임스페이스 수준에서 사용자가 수행할 수 있습니다. 네임스페이스 레이블을 사용하는 경우 라우터의 서비스 계정에는 라우터의 라벨에 액세스할 수 있는 **cluster-reader** 권한이 있어야 합니다.



참고

동일한 호스트 이름을 클레임하는 두 개 이상의 경로의 경우 확인 순서는 경로의 사용 기간을 기반으로 하며 가장 오래된 경로는 해당 호스트에 대한 클레임을 얻습니다. 분할된 라우터의 경우 라우터의 선택 기준에 일치하는 레이블을 기반으로 경로를 선택합니다. 경로에 라벨이 추가되는 시기를 결정할 수 있는 일관된 방법은 없습니다. 따라서 라우터의 선택 기준에 맞게 기존 호스트 이름을 클레임하는 이전 경로가 **"re-label"**인 경우 위에서 언급한 확인 순서(오래된 경로가 성공)에 따라 기존 경로를 대체합니다.

5.7.12. 대체 백엔드 및 가중치

경로는 일반적으로 를 통해 하나의 서비스와 **kind**가 있는 토큰과 연결됩니다. **service.** 경로에 대한 모든 요청은 부하 분산 전략을 기반으로 서비스의 엔드포인트에서 처리합니다.

경로를 지원하는 최대 **4**개의 서비스가 있을 수 있습니다. 각 서비스에서 처리하는 요청의 일부는 서비스 가중치에 의해 관리됩니다.

첫 번째 서비스는 **to:** 토큰을 사용하여 이전과 같이 입력하며, **alternateBackend:** 토큰을 사용하여 최대 **3**개의 추가 서비스를 입력할 수 있습니다. 각 서비스는 **kind**여야 합니다. **service** 기본값입니다.

각 서비스에는 연결된 가중치 가 있습니다. 서비스에서 처리하는 요청의 일부는 **weight /**

sum_of_all_weights입니다. 서비스에 둘 이상의 엔드포인트가 있는 경우 서비스의 weight는 각 엔드포인트에 최소 1을 갖는 엔드포인트에 분산됩니다. 서비스 weight 가 0이면 각 서비스 엔드포인트에 0이 부여됩니다.

가중치 는 범위 0-256에 있어야합니다. 기본값은 100입니다. weight 가 0이면 서비스는 부하 분산에 참여하지 않지만 기존 영구 연결을 계속 제공합니다.

alternateBackends 를 사용하는 경우 roundrobin 부하 분산 전략을 사용하여 요청이 weight 를 기반으로 서비스에 예상대로 배포되도록 합니다. 경로 주석을 사용하여 경로에 대해roundrobin 을 설정하거나 일반적으로 환경 변수를 사용하여 라우터에 대해 설정할 수 있습니다.

다음은 A/B 배포에 대체 백엔드를 사용하는 경로 구성의 예입니다.

alternateBackends 및 weights가 있는 경로:

```
apiVersion: v1
kind: Route
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin 1
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name 2
    weight: 20 3
  alternateBackends:
  - kind: Service
    name: service-name2 4
    weight: 10 5
  - kind: Service
    name: service-name3 6
    weight: 10 7
```

1

이 경로는 roundrobin부하 분산 전략을 사용합니다.

2

4 6

alternateBackend 서비스에 0개 이상의 Pod도 있을 수 있습니다.

3 5 7

총 가중치 는 40입니다. service-name 은 요청의 20/40 또는 1/2을 얻습니다. service-name2 및 service-name3 은 각 서비스에 1개 이상의 엔드포인트가 있다고 가정하면 각각 요청의 1/4을 얻습니다.

5.7.13. 경로별 주석

라우터는 환경 변수를 사용하여 노출하는 모든 경로에 대한 기본 옵션을 설정할 수 있습니다. 개별 경로는 주석에 특정 구성을 제공하는 방식으로 이러한 기본값 중 일부를 덮어쓸 수 있습니다.

경로 주석

이 섹션에 설명된 모든 항목에 대해 경로의 경로 정의에 주석을 설정하여 설정을 변경할 수 있습니다.

표 5.5. 경로 주석

Variable	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/balance	로드 밸런싱 알고리즘을 설정합니다. 사용 가능한 옵션은 **source**, **roundrobin**, **leastconn**입니다.	경유 경로의 경우 **ROUTER_TCP_BALANCE_SCHEME**입니다. 그 외에는 **ROUTER_LOAD_BALANCE_ALGORITHM**을 사용하십시오.
haproxy.router.openshift.io/disable_cookies	쿠키를 사용하여 관련 연결을 추적하지 않도록 설정합니다. **true** 또는 **TRUE**로 설정하면 들어오는 각 HTTP 요청에 사용할 백엔드 연결을 밸런스 알고리즘으로 선택합니다.	
router.openshift.io/cookie_name	이 경로에 사용할 선택적 쿠키를 지정합니다. 이름은 대문자와 소문자, 숫자, '_', '-'의 조합으로 구성해야 합니다. 기본값은 경로의 해시된 내부 키 이름입니다.	

Variable	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/pod-concurrent-connections	라우터에서 백업 Pod로 허용되는 최대 연결 수를 설정합니다. 참고: Pod가 여러 개인 경우 각각 이 수만큼의 연결이 있을 수 있습니다. 그러나 라우터가 여러 개 있고 조정이 이루어지지 않는 경우에는 각각 이 횟수만큼 연결할 수 있습니다. 설정하지 않거나 0으로 설정하면 제한이 없습니다.	
haproxy.router.openshift.io/rate-limit-connections	속도 제한 기능을 사용하려면 **true** 또는 **TRUE**로 설정합니다.	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	한 IP 주소에서 공유하는 동시 TCP 연결 수를 제한합니다.	
haproxy.router.openshift.io/rate-limit-connections.rate-http	IP 주소에서 HTTP 요청을 생성할 수 있는 속도를 제한합니다.	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	IP 주소의 TCP 연결 속도를 제한합니다.	
haproxy.router.openshift.io/timeout	경로에 대한 서버 쪽 타임아웃을 설정합니다. (TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
router.openshift.io/haproxy.health.check.interval	백엔드 상태 점검 간격을 설정합니다. (TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	경로에 대한 허용 목록을 설정합니다.	
haproxy.router.openshift.io/hsts_header	엣지 종단 경로 또는 재암호화 경로에 Strict-Transport-Security 헤더를 설정합니다.	

Variable	설명	기본값으로 사용되는 환경 변수
router.openshift.io/cookie-same-site	쿠키를 제한하는 값을 설정합니다. 값은 다음과 같습니다. **Lax**: 방문한 사이트와 타사 사이트 간에 쿠키가 전송됩니다. **Strict**: 쿠키가 방문한 사이트로 제한됩니다. **None**: 쿠키가 방문한 사이트로 제한됩니다. 이 값은 재암호화 및 엣지 경로에만 적용됩니다. 자세한 내용은 SameSite 쿠키 설명서를 참조하십시오.	

사용자 정의 시간 제한 설정 경로

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
[...]
```

(1)

HAProxy 지원 단위(us, ms, s, m, h, d)를 사용하여 새 시간 제한을 지정합니다. 단위가 제공되지 않는 경우 ms가 기본값입니다.

참고

패스스루(passthrough) 경로에 대한 서버 쪽 타임아웃 값을 너무 낮게 설정하면 해당 경로에서 WebSocket 연결이 자주 시간 초과될 수 있습니다.

5.7.14. 경로별 IP 화이트리스트

경로에 haproxy.router.openshift.io/ip_whitelist 주석을 추가하여 경로에 대한 액세스를 선택한 IP

주소 집합으로 제한할 수 있습니다. 화이트리스트는 승인된 소스 주소에 대한 IP 주소 및/또는 CIDR의 공백으로 구분된 목록입니다. 화이트리스트에 없는 IP 주소의 요청은 삭제됩니다.

몇 가지 예:

경로를 편집할 때 다음 주석을 추가하여 원하는 소스 IP를 정의합니다. 또는 oc annotate route <name> 을 사용합니다.

하나의 특정 IP 주소만 허용합니다.

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

여러 IP 주소를 허용합니다.

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP CIDR 네트워크를 허용합니다.

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

혼합 IP 주소 및 IP CIDR 네트워크를 허용합니다.

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

5.7.15. 와일드카드 하위 도메인 정책을 지정하는 경로 생성

와일드카드 정책을 사용하면 사용자가 도메인 내의 모든 호스트를 대상으로 하는 경로를 정의할 수 있습니다(라우터가 이를 허용하도록 구성된 경우). 경로는 wildcard Policy 필드를 사용하여 와일드카드 정책을 구성의 일부로 지정할 수 있습니다. 와일드카드 경로를 허용하는 정책과 함께 실행되는 모든 라우터는 와일드카드 정책에 따라 경로를 적절하게 노출합니다.

와일드카드 경로를 허용하도록 HAProxy 라우터를 구성하는 방법에 대해 알아봅니다.

하위 도메인 WildcardPolicy를 지정하는 경로

```
apiVersion: v1
kind: Route
spec:
  host: wildcard.example.com 1
  wildcardPolicy: Subdomain 2
  to:
    kind: Service
    name: service-name
```

1

서비스를 노출하는 데 사용되는 외부에서 연결할 수 있는 호스트 이름을 지정합니다.

2

외부에서 연결할 수 있는 호스트 이름이 하위 도메인 example.com 의 모든 호스트를 허용하도록 지정합니다..example.com 은 노출된 서비스에 도달하기 위해 호스트 이름 wildcard.example.com 의 하위 도메인입니다.*

5.7.16. 경로 상태

경로 상태 필드는 라우터에서만 설정합니다. 라우터가 더 이상 특정 경로를 제공하지 않도록 경로가 변경되면 상태가 오래됩니다. 라우터는 경로 상태 필드를 지우지 않습니다. 경로 상태의 부실 항목을 제거하려면 clear-route-status 스크립트를 사용합니다.

5.7.17. 경로에서 특정 도메인 거부 또는 허용

ROUTER_DENIED_DOMAINS 및 ROUTER_ALLOWED_DOMAINS 환경 변수를 사용하여 경로의 호스트 이름에서 특정 도메인 하위 집합을 거부하거나 허용하도록 라우터를 구성할 수 있습니다.

ROUTER_DENIED_DOMAINS	나열된 도메인은 표시된 경로에 허용되지 않습니다.
ROUTER_ALLOWED_DOMAINS	나열된 도메인만 표시된 경로에 허용됩니다.

*거부된 도메인 목록에 있는 도메인이 허용되는 도메인 목록보다 우선합니다. 즉, **OpenShift Container Platform**을 통해 거부 목록(해당되는 경우)을 먼저 확인하고 호스트 이름이 거부 도메인 목록에 없는 경우 허용된 도메인 목록을 확인합니다. 그러나 허용되는 도메인 목록은 더 제한적이며 라우터에서 해당 목록에 속하는 호스트의 경로만 허용합니다.*

*예를 들어 **myrouter** 경로의 **[*.]open.header.test,[*.]openshift .org** 및 **[*.]block.it** 경로를 거부하려면 다음 두 명령을 실행합니다.*

```
$ oc adm router myrouter ...
```

```
$ oc set env dc/myrouter ROUTER_DENIED_DOMAINS="open.header.test, openshift.org,
block.it"
```

*즉, **myrouter** 는 경로의 이름을 기반으로 다음을 허용합니다.*

```
$ oc expose service/<name> --hostname="foo.header.test"
```

```
$ oc expose service/<name> --hostname="www.allow.it"
```

```
$ oc expose service/<name> --hostname="www.openshift.test"
```

*그러나 **myrouter** 는 다음을 거부합니다.*

```
$ oc expose service/<name> --hostname="open.header.test"
```

```
$ oc expose service/<name> --hostname="www.open.header.test"
```

```
$ oc expose service/<name> --hostname="block.it"
```

```
$ oc expose service/<name> --hostname="franco.baresi.block.it"
```

```
$ oc expose service/<name> --hostname="openshift.org"
```

```
$ oc expose service/<name> --hostname="api.openshift.org"
```

*또는 호스트 이름이 **[*.]stickshift.org** 또는 **[*.] kates.net** 으로 설정되지 않은 경로를 차단하려면 다음 두 명령을 실행합니다.*

```
$ oc adm router myrouter ...
```

```
$ oc set env dc/myrouter ROUTER_ALLOWED_DOMAINS="stickshift.org, kates.net"
```

즉, myrouter 라우터에서 다음을 허용합니다.

```
$ oc expose service/<name> --hostname="stickshift.org"
```

```
$ oc expose service/<name> --hostname="www.stickshift.org"
```

```
$ oc expose service/<name> --hostname="kates.net"
```

```
$ oc expose service/<name> --hostname="api.kates.net"
```

```
$ oc expose service/<name> --hostname="erno.r.kube.kates.net"
```

그러나 myrouter 는 다음을 거부합니다.

```
$ oc expose service/<name> --hostname="www.open.header.test"
```

```
$ oc expose service/<name> --hostname="drive.ottomatic.org"
```

```
$ oc expose service/<name> --hostname="www.wayless.com"
```

```
$ oc expose service/<name> --hostname="www.deny.it"
```

두 시나리오를 모두 구현하려면 다음 두 명령을 실행합니다.

```
$ oc adm router adrouter ...
```

```
$ oc set env dc/adrouter ROUTER_ALLOWED_DOMAINS="okd.io, kates.net" \
    ROUTER_DENIED_DOMAINS="ops.openshift.org, metrics.kates.net"
```

그러면 호스트 이름이 [*.]openshift.org 또는 [*.] kates.net 으로 설정된 모든 경로가 허용되며 호스트 이름이 [*.]ops.openshift.org 또는 [*.]metrics. kates.net 으로 설정된 경로를 허용하지 않습니다.

따라서 다음이 거부됩니다.

```
$ oc expose service/<name> --hostname="www.open.header.test"
```

```
$ oc expose service/<name> --hostname="ops.openshift.org"
```

```
$ oc expose service/<name> --hostname="log.ops.openshift.org"

$ oc expose service/<name> --hostname="www.block.it"

$ oc expose service/<name> --hostname="metrics.kates.net"

$ oc expose service/<name> --hostname="int.metrics.kates.net"
```

그러나 다음이 허용됩니다.

```
$ oc expose service/<name> --hostname="openshift.org"

$ oc expose service/<name> --hostname="api.openshift.org"

$ oc expose service/<name> --hostname="m.api.openshift.org"

$ oc expose service/<name> --hostname="kates.net"

$ oc expose service/<name> --hostname="api.kates.net"
```

5.7.18. Kubernetes 인그레스 오브젝트 지원

Kubernetes 인그레스 오브젝트는 인바운드 연결이 내부 서비스에 연결하는 방법을 결정하는 구성 오브젝트입니다. **OpenShift Container Platform**은 수신 컨트롤러 구성 파일을 사용하여 이러한 오브젝트를 지원합니다.

이 컨트롤러는 인그레스 오브젝트를 감시하고 하나 이상의 경로를 생성하여 인그레스 오브젝트의 조건을 충족합니다. 컨트롤러는 **Ingress** 오브젝트와 생성된 경로 오브젝트도 동기화된 상태로 유지합니다. 여기에는 **Ingress** 오브젝트와 연결된 보안에 대한 생성된 경로 권한을 제공합니다.

예를 들어 다음과 같이 구성된 **Ingress** 오브젝트는 다음과 같습니다.

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: test
spec:
  rules:
  - host: test.com
    http:
     paths:
     - path: /test
```

```
    backend:
     serviceName: test-1
     servicePort: 80
```

다음 경로 오브젝트를 생성합니다.

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: test-a34th 1
  ownerReferences:
  - apiVersion: extensions/v1beta1
    kind: Ingress
    name: test
    controller: true
spec:
  host: test.com
  path: /test
  to:
    name: test-1
  port:
     targetPort: 80
```

1

이름은 인그레스 이름을 접두사로 사용하여 경로 오브젝트에서 생성합니다.

참고

경로를 생성하려면 **ingress** 오브젝트에 호스트, 서비스 및 경로가 있어야 합니다.

5.7.19. 네임스페이스 소유권 확인 비활성화

호스트 및 하위 도메인은 먼저 클레임을 수행하는 경로의 네임스페이스에서 소유합니다. 네임스페이스에 생성된 다른 경로는 하위 도메인에 클레임을 만들 수 있습니다. 다른 모든 네임스페이스는 클레임된 호스트 및 하위 도메인에 대한 클레임을 만들 수 없습니다. 호스트를 소유하는 네임스페이스는 호스트와 연결된 모든 경로(예: **www.abc.xyz/path1**) 도 소유합니다.

예를 들어 호스트 **www.abc.xyz** 가 경로에 의해 클레임되지 않은 경우. 네임스페이스 **ns 1**에서 호스트 **www.abc.xyz** 를 사용하여 **r 1** 경로를 생성하면 네임스페이스 **ns1** 이 호스트 **www.abc.xyz** 의 소유자이고 와일드카드 경로의 경우 **abc.xyz** 하위 도메인이 생성됩니다. 다른 네임스페이스 **ns2** 가 다른 경로 **www.abc.xyz/path1/path2** 인 경로를 생성하려고 하면 다른 네임스페이스의 경로(이 경우 **ns 1**)가 해당 호스트를 소유하고 있기 때문에 실패합니다.

*와일드카드를 사용하면 하위 도메인을 소유한 네임스페이스가 하위 도메인의 모든 호스트를 소유합니다. 위 예제와 같이 네임스페이스에서 하위 도메인 **abc.xyz** 를 소유하는 경우 다른 네임스페이스는 **z.abc.xyz** 를 클레임할 수 없습니다.*

네임스페이스 소유권 규칙을 비활성화하면 이러한 제한 사항을 비활성화하고 여러 네임스페이스에서 호스트(및 하위 도메인)를 클레임할 수 있습니다.

주의

라우터에서 네임스페이스 소유권을 비활성화하기로 결정한 경우 최종 사용자가 네임스페이스에서 호스트의 소유권을 요청할 수 있다는 점에 유의하십시오. 이러한 변경은 특정 개발 환경에서 바람직할 수 있지만, 이 기능을 프로덕션 환경에서 주의해서 사용하고 클러스터 정책이 신뢰할 수 없는 최종 사용자가 경로를 생성하지 못하도록 잠그고 있는지 확인합니다.

*예를 들어 **ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** 의 경우 네임스페이스 **ns1**이 가장 오래된 경로 **r1 www.abc.xyz** 을 생성하면 호스트 이름(+ 경로)만 소유합니다. 다른 네임스페이스는 해당 하위 도메인(**abc.xyz**)에서 가장 오래된 경로가 없어도 와일드카드 경로를 만들 수 있으며, 다른 네임스페이스에서 와일드카드가 겹치지 않는 다른 네임스페이스(예: **foo.abc.xyz,bar.abc.xyz,baz.abc.xyz**)를 클레임할 수 있습니다.*

*다른 모든 네임 스페이스 (예 : **ns2**)는 이제 **r2 www.abc.xyz/p1/p2** 경로를 만들 수 있으며 허용됩니다. 마찬가지로 다른 네임스페이스(**ns3**)는 하위 도메인 와일드카드 정책을 사용하여 **wildthing.abc.xyz** 경로를 만들 수있으며 와일드카드를 소유할 수 있습니다.*

*이 예제에서 볼 수 있듯이 정책 **ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** 는 더 많은 **lax**이며 네임스페이스 간에 클레임을 허용합니다. 라우터에서 네임스페이스 소유권이 비활성화된 경로를 거부하는 유일한 방법은 **host+path**가 이미 클레임된 경우입니다.*

*예를 들어 새로운 경로 **rx** 가 **www.abc.xyz/p1/p2** 를 클레임하려고 하면 **r2** 경로가 해당 **host+path** 조합을 소유하므로 거부됩니다. 정확한 **host+path**가 이미 주장되었으므로 경로 **rx** 가 동일한 네임 스페이스에 있는지 또는 다른 네임스페이스에 있는지 여부는 마찬가지입니다.*

이 기능은 라우터를 생성하는 동안 또는 라우터의 배포 구성에서 환경 변수를 설정하여 설정할 수 있습니다.

라우터 생성 중 설정

```
$ oc adm router ... --disable-namespace-ownership-check=true
```

라우터 배포 구성에서 환경 변수 설정

```
$ oc set env dc/router ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true
```

---

[1]
*이 지점 이후 장치 이름은 컨테이너 **B**의 호스트의 장치를 나타냅니다.*

6장. 서비스 카탈로그 구성 요소

6.1. SERVICE CATALOG

6.1.1. 개요

클라우드 네이티브 플랫폼에서 실행할 마이크로 서비스 기반 애플리케이션을 개발할 때 서비스 프로바이더 및 플랫폼에 따라 다양한 리소스를 프로비저닝하고 조정, 자격 증명 및 구성을 공유하는 여러 가지 방법이 있습니다.

개발자에게 보다 원활한 환경을 제공하기 위해 **OpenShift Container Platform**에는 **Kubernetes**용 **OSB API(Open Service Broker API)** 구현인 서비스 카탈로그 가 포함되어 있습니다. 이를 통해 사용자는 **OpenShift Container Platform**에 배포된 애플리케이션을 다양한 서비스 브로커에 연결할 수 있습니다.

서비스 카탈로그를 사용하면 클러스터 관리자가 단일 **API** 사양을 사용하여 여러 플랫폼을 통합할 수 있습니다. **OpenShift Container Platform** 웹 콘솔은 서비스 카탈로그에 서비스 브로커가 제공하는 클러스터 서비스 클래스를 표시하여 사용자가 애플리케이션에 사용할 서비스를 검색하고 인스턴스화할 수 있습니다.

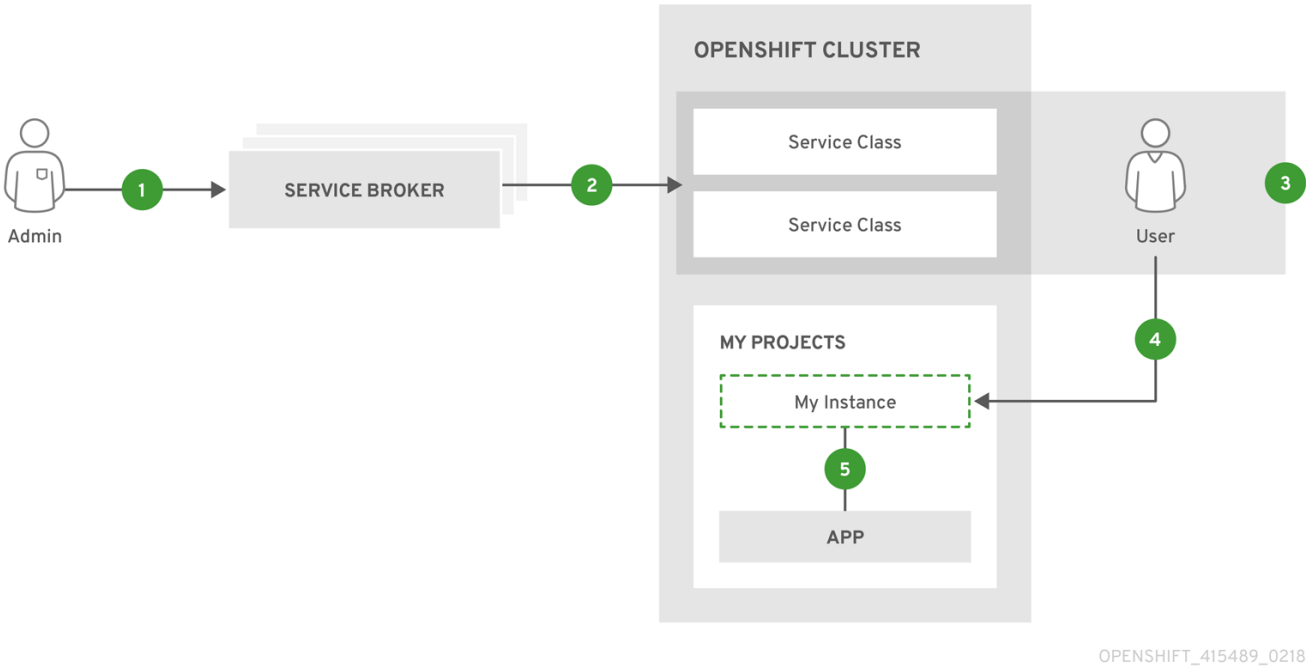
따라서 서비스 사용자는 다양한 공급자의 다양한 유형의 서비스에 대해 쉽고 일관성 있게 사용할 수 있는 반면, 서비스 공급자는 여러 플랫폼에 대한 액세스를 제공하는 하나의 통합 지점을 갖게 됩니다.

6.1.2. 설계

서비스 카탈로그 설계는 이 기본 워크플로를 따릅니다.

참고

다음 용어의 새로운 용어는 개념 및 용어에 더 정의되어 있습니다.



*클러스터 관리자는 하나 이상의 클러스터 서비스 브로커 를 **OpenShift Container Platform** 클러스터에 등록합니다. 이 작업은 일부 기본 제공 서비스 브로커에 대해 설치하거나 수동으로 설치하는 동안 자동으로 수행할 수 있습니다.*

*각 서비스 브로커는 사용자가 사용할 수 있어야 하는 **OpenShift Container Platform**에 대한 클러스터서비스 클래스집합과 해당 서비스(서비스 계획)의 변형을 지정합니다.*

***OpenShift Container Platform** 웹 콘솔 또는 **CLI**를 사용하여 사용자는 사용 가능한 서비스를 검색합니다. 예를 들어, **BestDataBase**라는 **database-as-a-service**인 클러스터 서비스 클래스를 사용할 수 있습니다.*

*사용자는 클러스터 서비스 클래스를 선택하고 자체 인스턴스를 요청합니다. 예를 들어 서비스 인스턴스는 **my_db** 라는 **BestDataBase** 인스턴스일 수 있습니다.*

*사용자 링크 또는 바인딩 은 해당 서비스 인스턴스를 포드 집합(애플리케이션)에 연결합니다. 예를 들어 **my_db** 서비스 인스턴스는 **my_app** 이라는 사용자의 애플리케이션에 바인딩될 수 있습니다.*

*사용자가 리소스를 배포하거나 프로비저닝 해제하도록 요청하면 서비스 카탈로그에 요청한 후 요청을 적절한 클러스터 서비스 브로커로 보냅니다. 일부 서비스에서 **provision,deprovision, update** 와 같은 일부 작업은 이행하는 데 다소 시간이 걸릴 것으로 예상됩니다. 클러스터 서비스 브로커를 사용할 수 없는 경우 서비스 카탈로그에서 작업을 계속 시도합니다.*

이 인프라를 사용하면 **OpenShift Container Platform**에서 실행되는 애플리케이션과 사용자가 사용하는 서비스 간에 느슨하게 결합할 수 있습니다. 이를 통해 이러한 서비스를 사용하여 자체 비즈니스 로직에 초점을 맞출 수 있으며 이러한 서비스 관리를 공급업체에게 맡기십시오.

6.1.2.1. 리소스 삭제

사용자가 서비스를 수행하거나 더 이상 청구되지 않으려는 경우 서비스 인스턴스를 삭제할 수 있습니다. 서비스 인스턴스를 삭제하려면 먼저 서비스 바인딩을 제거해야 합니다. 서비스 바인딩 삭제는 바인딩 해제라고 합니다. 삭제 프로세스의 일부에는 삭제되는 서비스 바인딩을 참조하는 시크릿 삭제가 포함됩니다.

모든 서비스 바인딩이 제거되면 서비스 인스턴스가 삭제될 수 있습니다. 서비스 인스턴스를 삭제하면 프로비저닝 해제 라고 합니다.

서비스 바인딩 및 서비스 인스턴스가 포함된 프로젝트 또는 네임스페이스가 삭제되면 서비스 카탈로그에서 먼저 클러스터 서비스 브로커를 요청하여 연결된 인스턴스 및 바인딩을 삭제해야 합니다. 서비스 카탈로그가 클러스터 서비스 브로커와 통신해야 하므로 프로젝트 또는 네임스페이스의 실제 삭제를 지연하고 프로비저닝 해제 작업을 수행할 때까지 기다립니다. 정상적인 상황에서 서비스에 따라 몇 분 이상 걸릴 수 있습니다.

참고

배포에서 사용하는 서비스 바인딩을 삭제하는 경우 배포에서 바인딩 보안에 대한 참조도 제거해야 합니다. 그렇지 않으면 다음 롤아웃이 실패합니다.

6.1.3. 개념 및 용어

클러스터 서비스 브로커

클러스터 서비스 브로커는 **OSB API** 사양을 준수하고 하나 이상의 서비스 세트를 관리하는 서버입니다. 소프트웨어는 자체 **OpenShift Container Platform** 클러스터 또는 다른 곳에서 호스팅할 수 있습니다.

클러스터 관리자는 클러스터 서비스 브로커를 나타내는 **ClusterServiceBroker API** 리소스를 생성하고 **OpenShift Container Platform** 클러스터에 등록할 수 있습니다. 이를 통해 클러스터 관리자는 클러스터 내에서 사용 가능한 클러스터 서비스 브로커를 사용하여 새로운 유형의 관리 서비스를 만들 수 있습니다.

ClusterServiceBroker 리소스는 사용자가 사용할 수 있어야 하는 **OpenShift Container Platform**에 대한 클러스터 서비스 브로커 및 서비스 세트(및 해당 서비스의 변형)에 대한 연결 세부 정보를 지정합니다. 특별한 유의는 클러스터 서비스 브로커로 인증하는 데 사용되는 데이터가 포함된 **authInfo** 섹션입니다.

ClusterServiceBroker 리소스의 예

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceBroker
metadata:
  name: BestCompanySaaS
spec:
  url: http://bestdatabase.example.com
  authInfo:
    basic:
      secretRef:
        namespace: test-ns
        name: secret-name
```

클러스터 서비스 클래스

또한 서비스 카탈로그 컨텍스트에서 "서비스"와 유사하게 클러스터 서비스 클래스 는 특정 클러스터 서비스 브로커에서 제공하는 관리 서비스 유형입니다. 클러스터에 새 클러스터 서비스 브로커 리소스가 추가될 때마다 서비스 카탈로그 컨트롤러가 해당 클러스터 서비스 브로커에 연결하여 서비스 제품 목록을 가져옵니다. 각각에 대해 새 **ClusterServiceClass** 리소스가 자동으로 생성됩니다.

참고

OpenShift Container Platform에는 내부 로드 밸런싱과 관련된 별도의 **Kubernetes** 리소스인 **services** 라는 핵심 개념도 있습니다. 이러한 리소스는 서비스 카탈로그 및 **OSB API** 컨텍스트에서 용어를 사용하는 방식과 혼동하지 않아야 합니다.

ClusterServiceClass 리소스의 예

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceClass
metadata:
  name: smallDB
  brokerName: BestDataBase
  plans: [...]
```

클러스터 서비스 계획

클러스터 서비스 계획은 클러스터 서비스 클래스의 계층을 나타냅니다. 예를 들어 클러스터 서비스 클래스는 서로 다른 비용으로 **QoS(Quality-of-service)**를 제공하는 일련의 플랜을 노출할 수 있습니다.

서비스 인스턴스

서비스 인스턴스는 클러스터 서비스 클래스의 프로비저닝된 인스턴스입니다. 사용자가 서비스 클래스에서 제공하는 기능을 사용하려는 경우 새 서비스 인스턴스를 생성할 수 있습니다.

새 **ServiceInstance** 리소스가 생성되면 서비스 카탈로그 컨트롤러가 적절한 클러스터 서비스 브로커에 연결하여 서비스 인스턴스를 프로비저닝하도록 지시합니다.

ServiceInstance 리소스의 예

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: my_db
  namespace: test-ns
spec:
  externalClusterServiceClassName: smallDB
  externalClusterServicePlanName: default
```

애플리케이션

애플리케이션 이라는 용어는 **OpenShift Container Platform** 배포 아티팩트(예: 서비스 인스턴스를 사용하는 사용자의 프로젝트에서 실행 중인 포드)를 나타냅니다.

인증 정보

자격 증명 은 애플리케이션에서 서비스 인스턴스와 통신하는 데 필요한 정보입니다.

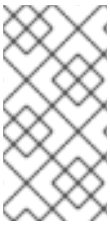
서비스 바인딩

서비스 바인딩 은 서비스 인스턴스와 애플리케이션 간의 링크입니다. 해당 애플리케이션은 서비스 인스턴스를 참조하고 사용하기를 원하는 클러스터 사용자가 생성합니다.

생성 시 서비스 카탈로그 컨트롤러는 서비스 인스턴스에 대한 연결 세부 정보 및 자격 증명이 포함된 **Kubernetes** 시크릿을 생성합니다. 이러한 시크릿은 일반적인 포드에 마운트할 수 있습니다.

서비스 바인딩 리소스 예

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parameters:
    securityLevel: confidential
  secretName: mySecret
```



참고

사용자는 애플리케이션 경로에 액세스할 수 없게 될 수 있으므로 인스턴스화된 인스턴스의 환경 변수 접두사를 변경하는 데 웹 콘솔을 사용하지 않아야 합니다.

매개 변수

*매개 변수는 서비스 바인딩 또는 서비스 인스턴스를 사용할 때 추가 데이터를 클러스터 서비스 브로커에 전달하는 데 사용할 수 있는 특수 필드입니다. 유일한 포맷 요구 사항은 매개 변수가 유효한 **YAML** 또는 **JSON**이 되도록 하는 것입니다. 위 예제에서 **security level** 매개 변수는 서비스 바인딩 요청의 클러스터 서비스 브로커에 전달됩니다. 더 많은 보안이 필요한 매개변수의 경우 시크릿에 배치하고 **parametersFrom** 을 사용하여 참조합니다.*

시크릿을 참조하는 서비스 바인딩 리소스의 예

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parametersFrom:
    - secretKeyRef:
        name: securityLevel
        key: myKey
  secretName: mySecret
```

6.1.4. 제공되는 클러스터 서비스 브로커

OpenShift Container Platform은 서비스 카탈로그와 함께 사용할 클러스터 서비스 브로커를 제공합니다.

- Template Service Broker
- OpenShift Ansible Broker

6.2. 서비스 카탈로그 CLI(명령줄 인터페이스)

6.2.1. 개요

서비스 카탈로그와 상호 작용하는 기본 워크플로 는 다음과 같습니다.

- 클러스터 관리자는 서비스를 사용할 수 있도록 브로커 서버를 설치하고 등록합니다.
- 사용자는 OpenShift 프로젝트에서 인스턴스화하고 해당 서비스 인스턴스를 포드에 연결하여 이러한 서비스를 사용합니다.

svcat 라는 서비스 카탈로그 명령줄 인터페이스(CLI) 유틸리티를 사용하여 이러한 사용자 관련 작업을 처리할 수 있습니다. oc 명령은 동일한 작업을 수행할 수 있지만 svcat 를 사용하여 서비스 카탈로그 리소스와의 더 쉽게 상호 작용할 수 있습니다. svcat 는 OpenShift 클러스터에서 집계된 API 엔드포인트를 사용하여 Service Catalog API와 통신합니다.

6.2.2. svcat설치

Red Hat 계정에 활성 OpenShift Enterprise 서브스크립션이 있는 경우 RHSM(Red Hat Subscription Management)을 사용하여 svcat 을 RPM으로 설치할 수 있습니다.

```
# yum install atomic-enterprise-service-catalog-svcat
```

6.2.2.1. 클라우드 공급자 고려 사항

Google Compute Engine Google Cloud Platform의 경우 다음 명령을 실행하여 들어오는 트레픽을 허용하도록 방화벽 규칙을 설정합니다.

```
$ gcloud compute firewall-rules create allow-service-catalog-secure --allow tcp:30443 --
description "Allow incoming traffic on 30443 port."
```

6.2.3. svcat사용

이 섹션에는 서비스 카탈로그 워크플로 에 나열된 사용자와 관련된 작업을 처리하는 일반적인 명령이 포함되어 있습니다. **svcat --help** 명령을 사용하여 자세한 정보를 가져오고 사용 가능한 다른 명령줄 옵션을 확인합니다. 이 섹션의 샘플 출력은 Ansible Service Broker가 이미 클러스터에 설치되어 있다고 가정합니다.

6.2.3.1. 브로커 세부 정보 얻기

사용 가능한 브로커 목록을 보고 브로커 카탈로그를 동기화하며 서비스 카탈로그에 배포된 브로커에 대한 세부 정보를 가져올 수 있습니다.

6.2.3.1.1. 브로커 찾기

클러스터에 설치된 모든 브로커를 보려면 다음을 수행합니다.

```
$ svcat get brokers
```

출력 예

```
         NAME                                    URL                                    STATUS
+-------------------------+-------------------------------------------------------------------------------------------+--
------+
  ansible-service-broker    https://asb.openshift-ansible-service-broker.svc:1338/ansible-
service-broker              Ready
  template-service-broker   https://apiserver.openshift-template-service-
broker.svc:443/brokers/template.openshift.io    Ready
```

6.2.3.1.2. 브로커 카탈로그 동기화

브로커에서 카탈로그 메타데이터를 새로 고치려면 다음을 수행합니다.

```
$ svcat sync broker ansible-service-broker
```

출력 예

```
Synchronization requested for broker: ansible-service-broker
```

6.2.3.1.3. 브로커 세부 정보보기

브로커의 세부 정보를 보려면 다음을 수행합니다.

```
$ svcat describe broker ansible-service-broker
```

출력 예

```
  Name:     ansible-service-broker
  URL:      https://openshift-automation-service-broker.openshift-automation-service-broker.svc:1338/openshift-automation-service-broker/
  Status:   Ready - Successfully fetched catalog entries from broker @ 2018-06-07 00:32:59 +0000 UTC
```

6.2.3.2. 서비스 클래스 및 서비스 계획 보기

ClusterServiceBroker 리소스를 생성할 때 서비스 카탈로그 컨트롤러는 브로커 서버를 쿼리하여 각 서비스에 대해 제공하는 모든 서비스를 찾고 서비스 클래스(**ClusterServiceClass**)를 생성합니다. 또한 브로커의 각 서비스에 대한 서비스 계획(**ClusterServicePlan**)도 생성합니다.

6.2.3.2.1. 서비스 클래스 보기

사용 가능한 **ClusterServiceClass** 리소스를 보려면 다음을 수행합니다.

```
$ svcat get classes
```

출력 예

```
      NAME                DESCRIPTION
+-------------------+-------------------------------+
  rh-mediawiki-apb    Mediawiki apb implementation

  ...

  rh-mariadb-apb      Mariadb apb implementation
  rh-mysql-apb        Software Collections MySQL APB
  rh-postgresql-apb   SCL PostgreSQL apb
                      implementation
```

서비스 클래스의 세부 정보를 보려면 다음을 수행합니다.

```
$ svcat describe class rh-postgresql-apb
```

출력 예

```
  Name:         rh-postgresql-apb
  Description:  SCL PostgreSQL apb implementation
  UUID:         d5915e05b253df421efe6e41fb6a66ba
  Status:       Active
  Tags:         database, postgresql
  Broker:       ansible-service-broker

Plans:
  NAME          DESCRIPTION
+------+---------------------------------+
  prod   A single DB server with
         persistent storage
  dev    A single DB server with no
         storage
```

6.2.3.2.2. 서비스 계획 보기

*클러스터에서 사용 가능한 **ClusterServicePlan** 리소스를 보려면 다음을 수행합니다.*

```
$ svcat get plans
```

출력 예

```
  NAME         CLASS                DESCRIPTION
+---------+------------------+-----------------------------+
  default   rh-mediawiki-apb    An APB that deploys MediaWiki

  ...

  prod      rh-mariadb-apb      This plan deploys a single
                                MariaDB instance with 10 GiB
                                of persistent storage
  dev       rh-mariadb-apb      This plan deploys a single
                                MariaDB instance with
                                ephemeral storage
  prod      rh-mysql-apb        A MySQL server with persistent
                                storage
  dev       rh-mysql-apb        A MySQL server with ephemeral
                                storage
  prod      rh-postgresql-apb   A single DB server with
                                persistent storage
  dev       rh-postgresql-apb   A single DB server with no
                                storage
```

계획 세부 정보 보기:

```
$ svcat describe plan rh-postgresql-apb/dev
```

출력 예

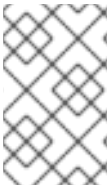
```
  Name:         dev
  Description:   A single DB server with no storage
  UUID:         9783fc2e859f9179833a7dd003baa841
  Status:       Active
  Free:         true
  Class:        rh-postgresql-apb

 Instances:
```

```
No instances defined

Instance Create Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  properties:
    postgresql_database:
      default: admin
      pattern: ^[a-zA-Z_][a-zA-Z0-9_]*$
      title: PostgreSQL Database Name
      type: string
    postgresql_password:
      pattern: ^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$
      title: PostgreSQL Password
      type: string
    postgresql_user:
      default: admin
      maxLength: 63
      pattern: ^[a-zA-Z_][a-zA-Z0-9_]*$
      title: PostgreSQL User
      type: string
    postgresql_version:
      default: "9.6"
      enum:
      - "9.6"
      - "9.5"
      - "9.4"
      title: PostgreSQL Version
      type: string
  required:
  - postgresql_database
  - postgresql_user
  - postgresql_password
  - postgresql_version
  type: object

Instance Update Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  properties:
    postgresql_version:
      default: "9.6"
      enum:
      - "9.6"
      - "9.5"
      - "9.4"
      title: PostgreSQL Version
      type: string
  required:
  - postgresql_version
  type: object

Binding Create Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  type: object
```

6.2.3.3. 서비스 프로비저닝

프로비저닝은 서비스를 사용할 수 있도록 하는 것을 의미합니다. 서비스를 프로비저닝하려면 서비스 인스턴스를 만든 다음 바인딩해야 합니다.

6.2.3.3.1. ServiceInstance 만들기



참고

서비스 인스턴스는 **OpenShift** 네임스페이스 내부에서 생성해야 합니다.

1. 새 프로젝트를 생성합니다.

```
$ oc new-project <project-name> 1
```

1 <project-name> 을 프로젝트 이름으로 바꿉니다.

2. 명령을 사용하여 서비스 인스턴스를 생성합니다.

```
$ svcat provision postgresql-instance --class rh-postgresql-apb --plan dev --params-
json
'{"postgresql_database":"admin","postgresql_password":"admin","postgresql_user":
"admin","postgresql_version":"9.6"}' -n szh-project
```

출력 예

```
  Name:       postgresql-instance
  Namespace:  szh-project
  Status:
  Class:      rh-postgresql-apb
  Plan:       dev

Parameters:
  postgresql_database: admin
```

```
postgresql_password: admin
postgresql_user: admin
postgresql_version: "9.6"
```

6.2.3.3.1.1. 서비스 인스턴스 세부 정보 보기

서비스 인스턴스 세부 정보를 보려면 다음을 수행합니다.

```
$ svcat get instance
```

출력 예

```
      NAME          NAMESPACE        CLASS        PLAN   STATUS
+--------------------+-------------+------------------+------+--------+
  postgresql-instance   szh-project   rh-postgresql-apb   dev    Ready
```

6.2.3.3.2. 서비스 바인딩 만들기

ServiceBinding 리소스를 생성할 때 다음을 수행합니다.

1. 서비스 카탈로그 컨트롤러는 브로커 서버와 통신하여 바인딩을 시작합니다.
2. 브로커 서버에서 자격 증명을 생성하고 서비스 카탈로그 컨트롤러에 발행합니다.
3. 서비스 카탈로그 컨트롤러는 해당 자격 증명을 프로젝트에 시크릿으로 추가합니다.

명령을 사용하여 서비스 바인딩을 생성합니다.

```
$ svcat bind postgresql-instance --name mediawiki-postgresql-binding
```

출력 예

```
Name:       mediawiki-postgresql-binding
Namespace:  szh-project
Status:
Instance:   postgresql-instance

Parameters:
{}
```

6.2.3.3.2.1. 서비스 바인딩 세부 정보보기

1. 서비스 바인딩 세부 정보를 보려면 다음을 수행합니다.

```
$ svcat get bindings
```

출력 예

```
          NAME              NAMESPACE       INSTANCE        STATUS
+-----------------------------+-------------+--------------------+--------+
 mediawiki-postgresql-binding   szh-project   postgresql-instance   Ready
```

2. 서비스를 바인딩한 후 인스턴스 세부 정보를 확인합니다.

```
$ svcat describe instance postgresql-instance
```

출력 예

```
 Name:       postgresql-instance
 Namespace:  szh-project
 Status:     Ready - The instance was provisioned successfully @ 2018-06-05 08:42:55
+0000 UTC
 Class:      rh-postgresql-apb
 Plan:       dev

Parameters:
```

```
  postgresql_database: admin
  postgresql_password: admin
  postgresql_user: admin
  postgresql_version: "9.6"

Bindings:
          NAME            STATUS
+-----------------------------+--------+
  mediawiki-postgresql-binding   Ready
```

6.2.4. 리소스 삭제

서비스 카탈로그 관련 리소스를 삭제하려면 서비스 바인딩을 해제하고 서비스 인스턴스를 프로비저닝 해제해야 합니다.

6.2.4.1. 서비스 바인딩 삭제

1. 서비스 인스턴스와 연결된 모든 서비스 바인딩을 삭제하려면 다음을 수행합니다.

```
$ svcat unbind -n <project-name> (1)
  \ <instance-name> (2)
```

(1) 서비스 인스턴스가 포함된 프로젝트의 이름입니다.

(2) 바인딩과 연결된 서비스 인스턴스의 이름입니다.

예를 들면 다음과 같습니다.

```
$ svcat unbind -n szh-project postgresql-instance
```

출력 예

```
deleted mediawiki-postgresql-binding
```

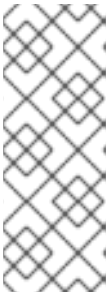
2.

모든 서비스 바인딩이 삭제되었는지 확인합니다.

```
$ svcat get bindings
```

출력 예

```
  NAME   NAMESPACE   INSTANCE   STATUS
+------+-----------+----------+--------+
```



참고

이 명령을 실행하면 인스턴스의 모든 서비스 바인딩이 삭제됩니다. 인스턴스 내에서 개별 바인딩을 삭제하려면 ***svcat unbind -n <project-name> --name <binding-name>*** *명령을 실행합니다. 예를 들어* ***svcat unbind -n szh-project --name mediawiki-postgresql-binding.***

3.

연결된 보안이 삭제되었는지 확인합니다.

```
$ oc get secret -n szh-project
```

출력 예

```
NAME                       TYPE                                  DATA      AGE
builder-dockercfg-jxk48    kubernetes.io/dockercfg               1         9m
builder-token-92jrf        kubernetes.io/service-account-token   4         9m
builder-token-b4sm6        kubernetes.io/service-account-token   4         9m
default-dockercfg-cggcr    kubernetes.io/dockercfg               1         9m
default-token-g4sg7        kubernetes.io/service-account-token   4         9m
default-token-hvdpq        kubernetes.io/service-account-token   4         9m
deployer-dockercfg-wm8th   kubernetes.io/dockercfg               1         9m
deployer-token-hnk5w       kubernetes.io/service-account-token   4         9m
deployer-token-xfr7c       kubernetes.io/service-account-token   4         9m
```

6.2.4.2. 서비스 인스턴스 삭제

1. *서비스 인스턴스를 프로비저닝 해제하려면 다음을 수행합니다.*

```
$ svcat deprovision postgresql-instance
```

출력 예

```
deleted postgresql-instance
```

2. *인스턴스가 삭제되었는지 확인합니다.*

```
$ svcat get instance
```

출력 예

```
  NAME   NAMESPACE   CLASS   PLAN   STATUS
+------+-----------+-------+------+--------+
```

6.2.4.3. 서비스 브로커 삭제

1. *서비스 카탈로그의 브로커 서비스를 제거하려면 **ClusterServiceBroker** 리소스를 삭제합니다.*

```
$ oc delete clusterservicebrokers template-service-broker
```

출력 예

-

```
clusterservicebroker "template-service-broker" deleted
```

2. 클러스터에 설치된 모든 브로커를 보려면 다음을 수행합니다.

```
$ svcat get brokers
```

출력 예

```
        NAME                                    URL                                STATUS
+-----------------------+-----------------------------------------------------------------------------
------+--------+
  ansible-service-broker    https://asb.openshift-ansible-service-
broker.svc:1338/ansible-service-broker              Ready
```

3. 브로커의 **ClusterServiceClass** 리소스를 보고 브로커가 제거되었는지 확인합니다.

```
$ svcat get classes
```

출력 예

```
  NAME   DESCRIPTION
+------+-------------+
```

6.3. TEMPLATE SERVICE BROKER

템플릿 서비스 브로커 (TSB)는 초기 릴리스 이후 OpenShift Container Platform과 함께 제공되는 기

본 Instant App 및 빠른 시작 템플릿에 대한 서비스 카탈로그 가시성을 제공합니다. TSB는 Red Hat에서 제공했는지, 클러스터 관리자 또는 사용자 또는 타사 벤더가 제공했는지 여부에 관계없이 OpenShift Container Platform 템플릿이 작성된 모든 서비스로 사용할 수도 있습니다.

기본적으로 TSB는 openshift 프로젝트에서 전역적으로 사용 가능한 오브젝트를 표시합니다. 클러스터 관리자가 선택하는 다른 프로젝트를 조사하도록 구성할 수도 있습니다.

6.4. OPENSHIFT ANSIBLE 브로커

6.4.1. 개요

OAB(Open Service Broker)는 Ansible 플레이북 번들(APB)에서 정의한 애플리케이션을 관리하는 OSB(Open Service Broker) API 구현입니다. APB는 OpenShift Container Platform에서 컨테이너 애플리케이션을 정의하고 배포하는 새로운 방법을 제공합니다. 이 방법은 Ansible 런타임을 사용하여 컨테이너 이미지에 빌드된 Ansible 플레이북 번들로 구성됩니다. APB는 Ansible을 활용하여 복잡한 배포를 자동화하는 표준 메커니즘을 생성합니다.

OAB 설계는 이 기본 워크플로우를 따릅니다.

1. OpenShift Container Platform 웹 콘솔을 사용하여 서비스 카탈로그에서 사용 가능한 애플리케이션 목록을 사용자 요청합니다.
2. 서비스 카탈로그는 사용 가능한 애플리케이션에 대해 OAB를 요청합니다.
3. OAB는 정의된 컨테이너 이미지 레지스트리와 통신하여 사용할 수 있는 APB를 확인합니다.
4. 사용자는 특정 APB를 배포하기 위해 요청을 발행합니다.
5. 배포 요청은 OAB로 이동하여 APB에서 배포 방법을 호출하여 사용자의 요청을 이행합니다.

6.4.2. Ansible Playbook 번들

Ansible 플레이북 번들(APB)은 Ansible 역할 및 플레이북에 대한 기존 투자를 활용할 수 있는 경량 애플리케이션 정의입니다.

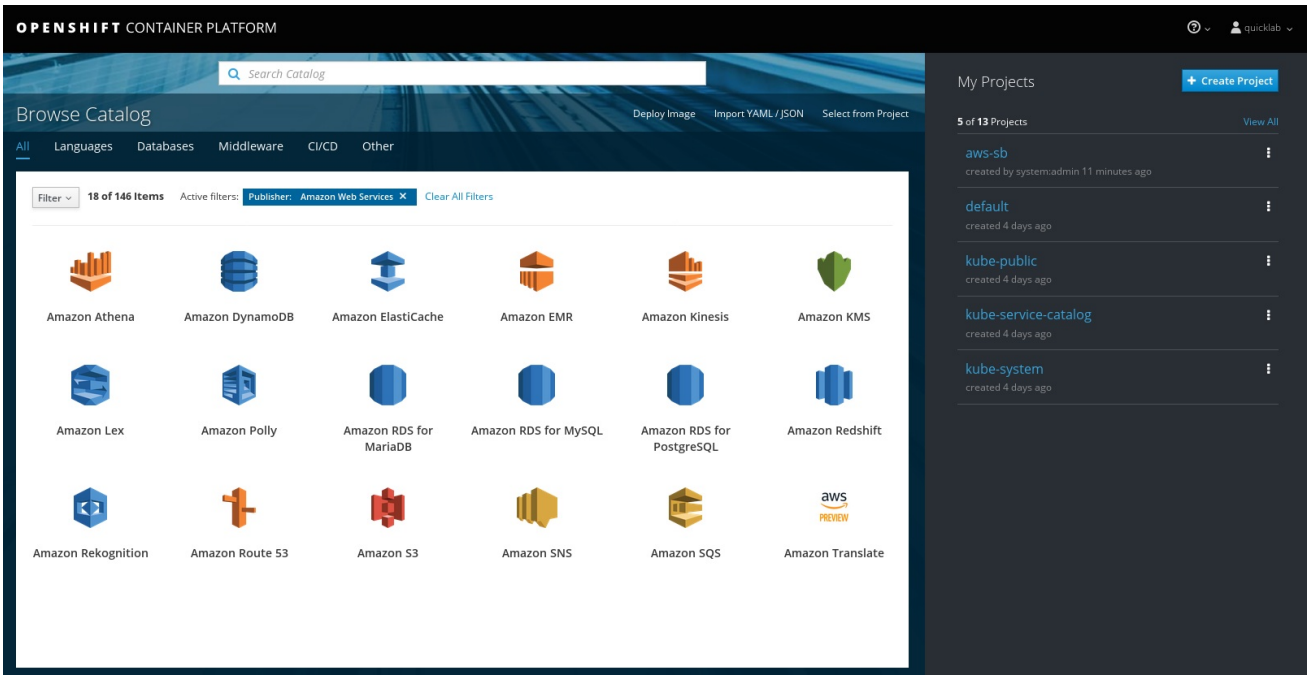
APB는 명명된 플레이북과 함께 간단한 디렉터리를 사용하여 프로비저닝 및 바인딩과 같은 OSB API 작업을 수행합니다. apb.yml 사양 파일에 정의된 메타데이터에는 배포 중에 사용할 필수 매개변수 및 선택적 매개변수 목록이 포함되어 있습니다.

전체 설계 및 APB 작성 방법에 대한 자세한 내용은 APB 개발 가이드를 참조하십시오.

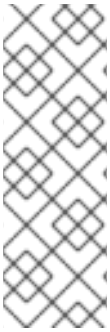
6.5. AWS SERVICE BROKER

AWS 서비스 브로커는 OpenShift Container Platform 서비스 카탈로그를 통해 AWS(Amazon Web Services)에 대한 액세스를 제공합니다. AWS 서비스 및 구성 요소는 OpenShift Container Platform 웹 콘솔 및 AWS 대시보드 모두에서 구성하고 볼 수 있습니다.

그림 6.1. OpenShift Container Platform 서비스 카탈로그의 AWS 서비스 예



AWS 서비스 브로커 설치에 대한 자세한 내용은 Amazon Web Services - Labs 문서 리포지토리의 AWS 서비스 브로커 설명서 를 참조하십시오.



참고

***OpenShift Container Platform**에서 **AWS** 서비스 브로커가 지원되고 검증됩니다. **Amazon**에서 직접 다운로드로 제공됩니다. 따라서 이 서비스 브로커 솔루션의 많은 구성 요소는 새 **OpenShift Container Platform** 릴리스 후 **2** 개월의 지연을 가진 가장 최근 두 가지 버전에 대해 **Amazon**에서 직접 지원됩니다. **Red Hat**은 **OpenShift** 클러스터 및 서비스 카탈로그 문제의 설치 및 문제 해결을 지원합니다.*