



Red Hat Training and Certification

수강생 워크북 (ROLE)

OCP 4.5 DO285

**Containers, Kubernetes, and Red Hat
OpenShift Administration II**

여orum 1

Containers, Kubernetes, and Red Hat OpenShift Administration II



OCP 4.5 DO285
Containers, Kubernetes, and Red Hat OpenShift
Administration II
엮음 1 20201202
Publication date 20201202

Authors: Richard Allred, Joel Birchler, Christopher Caillouet, Ivan Chavero, Zach Guterman, Andres Hernandez, Michael Jarrett, Dan Kolepp, Fernando Lozano, Razique Mahroua, James Mighion, Michael Phillips, Eduardo Ramirez Ronco, Jordi Sola Alaball
Editor: David O'Brien, Nicole Muller, Dave Sacco

Copyright © 2020 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2020 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, Hibernate, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

The OpenStack® word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation in the United States and other countries, and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

기여자: Forrest Taylor, Manuel Aude Morales, Jim Rigsbee

문서 규칙	ix
소개	xi
Containers, Kubernetes, and Red Hat OpenShift Administration II	xi
강의실 환경 오리엔테이션	xii
국제화	xix
1. 컨테이너 기술 소개	1
컨테이너 기술 개요	2
퀴즈: 컨테이너 기술 개요	5
컨테이너 아키텍처 개요	9
퀴즈: 컨테이너 아키텍처 개요	12
Kubernetes 및 OpenShift 개요	14
퀴즈: Kubernetes 및 OpenShift 설명	17
연습 가이드: 강의실 환경 구성	19
요약	23
2. 컨테이너화된 서비스 생성	25
컨테이너화된 서비스 프로비저닝	26
연습 가이드: MySQL 데이터베이스 인스턴스 생성	32
요약	35
3. 컨테이너 관리	37
컨테이너 라이프사이클 관리	38
연습 가이드: MySQL 컨테이너 관리	46
컨테이너에 영구저장장치 연결	49
연습 가이드: MySQL 데이터베이스 유지	52
컨테이너 액세스	55
연습 가이드: 데이터베이스 로드	59
랩: 컨테이너 관리	62
요약	71
4. 컨테이너 이미지 관리	73
레지스트리 액세스	74
퀴즈: 레지스트리 사용	80
컨테이너 이미지 관리	84
연습 가이드: 사용자 지정 Apache 컨테이너 이미지 생성	89
랩: 이미지 관리	93
요약	101
5. 사용자 지정 컨테이너 이미지 생성	103
사용자 지정 컨테이너 이미지 설계	104
퀴즈: 컨테이너 이미지 설계에 대한 접근법	108
Dockerfiles를 사용하여 사용자 지정 컨테이너 이미지 빌드	110
연습 가이드: 기본 Apache 컨테이너 이미지 생성	115
랩: 사용자 지정 컨테이너 이미지 생성	119
요약	126
6. OpenShift에 컨테이너화된 애플리케이션 배포	127
Kubernetes 리소스 생성	128
연습 가이드: OpenShift에 데이터베이스 서버 배포	138
경로 만들기	143
연습 가이드: 서비스를 경로로 노출	147
S2I(Source-to-Image)로 애플리케이션 생성	152
연습 가이드: S2I(Source-to-Image)로 컨테이너화된 애플리케이션 생성	162
랩: OpenShift에 컨테이너화된 애플리케이션 배포	168
요약	173
7. 멀티컨테이너 애플리케이션 배포	175

OpenShift에 멀티컨테이너 애플리케이션 배포	176
연습 가이드: 템플릿으로 애플리케이션 생성	186
랩: 소프트웨어 애플리케이션 컨테이너화 및 배포	192
요약	202
8. Red Hat OpenShift Container Platform 설명	203
OpenShift Container Platform 기능 설명	204
퀴즈: OpenShift Container Platform 기능 설명	209
OpenShift 아키텍처 설명	213
퀴즈: OpenShift 아키텍처 설명	216
클러스터 운영자 설명	218
퀴즈: 클러스터 운영자 설명	221
요약	223
9. 클러스터 확인	225
설치 방법 설명	226
퀴즈: 설치 방법 설명	228
OpenShift 클러스터 및 애플리케이션의 문제 해결	230
연습 가이드: OpenShift 클러스터 및 애플리케이션의 문제 해결	238
OpenShift 동적 스토리지 소개	245
연습 가이드: OpenShift 동적 스토리지 소개	249
요약	254
10. 인증 및 권한 부여 구성	255
ID 프로바이더 구성	256
연습 가이드: ID 프로바이더 구성	263
RBAC를 사용하여 권한 정의 및 적용	272
연습 가이드: RBAC를 사용하여 권한 정의 및 적용	276
랩: 인증 및 권한 부여 구성	281
요약	289
11. 애플리케이션 보안 구성	291
시크릿을 사용하여 중요 정보 관리	292
연습 가이드: 시크릿을 사용하여 중요 정보 관리	297
보안 컨텍스트 제약 조건으로 애플리케이션 권한 제어	303
연습 가이드: 보안 컨텍스트 제약 조건으로 애플리케이션 권한 제어	306
랩: 애플리케이션 보안 구성	310
요약	317
12. OpenShift 네트워킹 구성 요소 구성	319
OpenShift 소프트웨어 정의 네트워킹 문제 해결	320
연습 가이드: OpenShift 소프트웨어 정의 네트워킹 문제 해결	326
외부 액세스를 위해 애플리케이션 노출	335
연습 가이드: 외부 액세스를 위해 애플리케이션 노출	341
네트워크 정책 구성	351
연습 가이드: 네트워크 정책 구성	355
랩: 애플리케이션을 위한 OpenShift 네트워킹 구성	363
요약	375
13. 포드 스케줄링 제어	377
포드 스케줄링 동작 제어	378
연습 가이드: 포드 스케줄링 동작 제어	385
애플리케이션의 리소스 사용 제한	391
연습 가이드: 애플리케이션의 리소스 사용 제한	401
애플리케이션 크기 조정	411
연습 가이드: 애플리케이션 크기 조정	415
랩: 포드 스케줄링 제어	421
요약	429

14. 클러스터 업데이트 설명	431
클러스터 업데이트 프로세스 설명	432
퀴즈: 클러스터 업데이트 프로세스 설명	442
요약	446
15. 웹 콘솔로 클러스터 관리	447
클러스터 관리 수행	448
연습 가이드: 클러스터 관리 수행	451
워크로드 및 운영자 관리	458
연습 가이드: 워크로드 및 운영자 관리	463
클러스터 지표 검사	472
연습 가이드: 클러스터 지표 검사	476
랩: 웹 콘솔로 클러스터 관리	481
요약	492
16. 종합 검토	493
종합 검토	494
랩: OpenShift 클러스터 및 애플리케이션의 문제 해결	496
랩: 리소스 및 네트워크 제한 사항을 사용하여 프로젝트 템플릿 구성	510
A. GitHub 계정 만들기	523
GitHub 계정 만들기	524
B. Quay 계정 만들기	527
Quay 계정 만들기	528
리포지토리 가시성	531
C. 유용한 Git 명령	535
GIT 명령	536
D. OpenShift 클러스터 확장	539
OpenShift 클러스터 수동 확장	540
OpenShift 클러스터 자동 확장	544
요약	547

문서 규칙



참조

"참조"는 주제와 관련된 외부 설명서를 찾을 수 있는 위치를 설명합니다.



참고

"참고 사항"은 수행 중인 작업에 대한 팁, 바로 가기 또는 대체 접근법입니다. 참고 사항을 무시하더라도 큰 문제가 발생하지는 않지만, 더 쉬운 메서드를 알려주는 도움말을 놓칠 수도 있습니다.



중요

"중요" 박스는 현재 세션에만 적용되는 구성 변경 사항 또는 업데이트를 적용하기 전에 다시 시작해야 하는 서비스 등 놓치기 쉬운 내용을 자세히 정의합니다. "중요" 레이블이 지정된 박스를 무시하는 경우 데이터 손실이 발생하지는 않지만 불편하고 당황스러운 경우가 발생할 수 있습니다.



경고

"경고"는 무시해서는 안 됩니다. 경고를 무시할 경우 데이터 손실이 발생할 수 있습니다.

소개

Containers, Kubernetes, and Red Hat OpenShift Administration II

Container, Kubernetes, Red Hat OpenShift Administration II(DO285)를 이수하면 Linux 컨테이너와 Red Hat® OpenShift® Container Platform을 구축하고 관리하기 위한 핵심 지식을 쌓을 수 있습니다. 이 핸즈온 랙 기반 교육 과정에서는 로컬 컨테이너 런타임 또는 OpenShift 클러스터에 샘플 애플리케이션을 배포하는 방법과 OpenShift 클러스터를 구성하고 관리하는 방법에 대한 설명을 통해 개발자들이 플랫폼을 사용하는 방법을 자세히 이해할 수 있습니다. 이러한 기술은 개발자, 관리자, 사이트 안정성 엔지니어를 비롯한 여러 역할에 필요합니다.

교육 과정 목표

- 이 교육 과정을 마치면 Podman을 사용하여 로컬 컨테이너를 생성하고 관리하며, 새로운 OpenShift 클러스터를 구축하고, 클러스터의 초기 구성과 실행을 수행하며, 매일 클러스터를 관리하는 기술을 보여줄 수 있습니다. 이 교육 과정에서 중점을 두는 주요 사항 중 하나는 첫날 이후 발생하는 일반적인 문제를 해결하는 것입니다.

대상

- OpenShift 클러스터의 특징과 기능을 이해하는 데 관심이 있는 시스템 설계자 및 소프트웨어 설계자
- 클러스터의 초기 구축에 관심이 있는 시스템 관리자
- 클러스터의 지속적인 유지 관리에 관심이 있는 클러스터 운영자
- 클러스터의 지속적인 유지 관리 및 문제 해결에 관심이 있는 사이트 안정성 엔지니어

사전 요구 사항

- RHCSA(Red Hat Certified System Administrator) 인증 취득 또는 이에 상응하는 지식 보유

강의실 환경 오리엔테이션

워크스테이션 시스템

이 교육 과정에서 실습 활동(연습)에 사용되는 주요 컴퓨터 시스템은 **workstation**입니다.

workstation 시스템은 표준 사용자 계정 **student**를 사용하며 암호는 **student**입니다. 이 교육 과정의 연습을 수행하는 경우 **root**로 로그인하지 않아도 됩니다. 그러나 필요한 경우 **workstation**에서 **root**의 암호는 **redhat**입니다.

이 암호는 강의실 환경의 일부로 사전 설치되는 OpenShift 클러스터를 관리하기 위해 **oc** 명령을 입력하는 **workstation** 시스템의 암호입니다.

이 교육 과정의 연습을 완료하는 데 필요한 쉘 스크립트 및 Ansible 플레이북을 실행하는 **workstation** 시스템의 암호이기도 합니다.

연습에서 웹 브라우저를 열어 애플리케이션 또는 웹사이트에 액세스해야 하는 경우, **workstation** 시스템의 그래픽 콘솔에서 Firefox 웹 브라우저를 사용해야 합니다.



참고

강의실 환경을 처음 시작하는 경우 OpenShift 클러스터를 완전히 사용할 수 있을 때까지 시간이 조금 더 걸립니다. **lab** 명령은 연습을 시작할 때마다 필요에 따라 확인하고 대기합니다.

lab 명령을 먼저 실행하지 않고 **oc** 명령 또는 웹 콘솔을 사용하여 클러스터에 액세스하려고 하면 클러스터를 아직 사용하지 못할 수 있습니다. 이 경우 몇 분 후에 다시 시도합니다.

강의실 환경

이 Containers, Kubernetes, and Red Hat OpenShift Administration II(DO285) 교육 과정에는 Red Hat OpenShift I의 Containers & Kubernetes(DO180) 및 OpenShift Container Platform Administration II(DO280)의 내용 및 랩이 포함됩니다. 모든 수강생에게 완벽한 원격 강의실 환경이 제공됩니다. 모든 수강생에게는 해당 환경의 일부로 관리 작업을 수행할 수 있는 전용 OpenShift 클러스터가 제공됩니다.

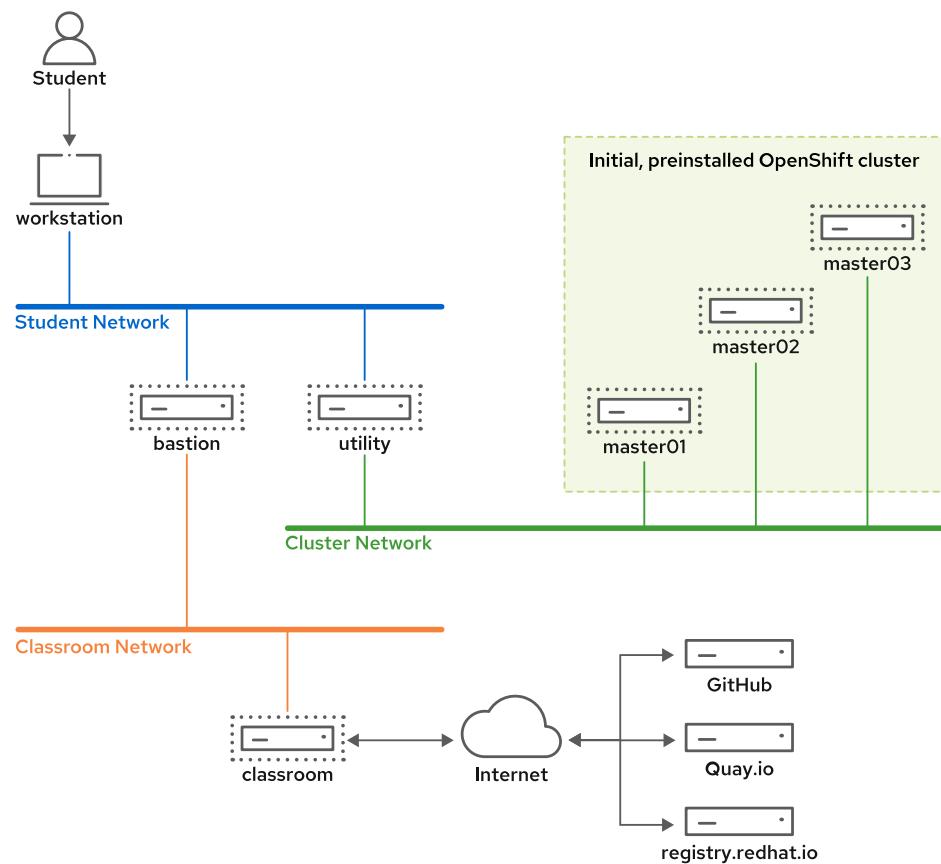
교육 과정의 DO180 부분에 있는 많은 연습을 수행하려면 연습 가이드: 강의실 환경 구성의 작업을 완료해야 합니다. 여기에는 **lab-configure** 명령을 실행하고 <https://github.com/RedHatTraining/D0180-apps> Git 리포지토리의 분기를 `/home/student/D0180-apps`에 복제하는 작업이 포함됩니다. **lab-configure** 명령은 GitHub 및 Quay.io 사용자 이름을 입력하도록 표시하고 변경 사항을 `/usr/local/etc/ocp4.config` 파일에 저장합니다.

기본적으로 강의실 환경의 OpenShift 클러스터는 HTPasswd ID 프로바이더를 사용하고 **developer** 사용자가 암호 **developer**를 사용하여 액세스할 수 있습니다. HTPasswd ID 프로바이더를 사용하도록 OpenShift를 구성하는 것이 교육 과정에 있는 DO280 부분의 목표이기 때문에 일부 랩 스크립트에서는 ID 프로바이더 구성을 제거합니다. 교육 과정의 DO180 부분으로 돌아가면 **lab-configure** 명령을 실행하여 OpenShift 클러스터에 암호 **developer**를 사용하여 **developer** 사용자로 액세스할 수 있는지 확인하십시오.

강의실 환경은 다수의 수강생들이 공유하는 대규모 Red Hat OpenStack Platform 클러스터에서 전부 가상 시스템으로 실행됩니다.

소개

Red Hat 교육은 많은 국가의 수강생들에게 단축된 대기 시간을 제공하기 위해 전 세계 다수의 데이터 센터에서 수많은 OpenStack 클러스터를 유지 관리합니다.



수강생 네트워크, 강의실 네트워크, 클러스터 네트워크의 모든 시스템에서는 Red Hat Enterprise Linux 8(RHEL 8)을 실행합니다. 단, OpenShift 클러스터의 노드에 해당하는 시스템은 제외됩니다. 해당 시스템에서는 RHEL CoreOS를 실행합니다.

bastion, **utility**, **classroom**이라는 시스템은 항상 실행 중이어야 합니다. 이러한 시스템은 강의실 환경과 OpenShift 클러스터에 필요한 인프라 서비스를 제공합니다. 해당 시스템과 직접 상호 작용할 필요는 없습니다.

일반적으로 환경을 연습용으로 설정하라는 요구 사항이 있는 경우 연습에서 **lab** 명령을 실행하면 이러한 시스템에 액세스하여, 사용자의 추가 작업은 필요하지 않습니다.

수강생 네트워크의 모든 시스템은 **lab.example.com** DNS 도메인에 있고, 강의실 네트워크의 모든 시스템은 **example.com** DNS 도메인에 있습니다.

masterXX라는 시스템은 강의실 환경의 일부인 OpenShift 4 클러스터의 노드입니다.

클러스터 네트워크의 모든 시스템은 **ocp4.example.com** DNS 도메인에 있습니다.

Classroom Machines

시스템 이름	IP 주소	역할
workstation.lab.example.com	172.25.250.9	시스템 관리에 사용되는 그래픽 워크스테이션
classroom.example.com	172.25.254.254	강의실 네트워크를 인터넷에 연결하는 라우터
bastion.lab.example.com	172.25.250.254	수강생 네트워크를 강의실 네트워크에 연결하는 라우터
utility.lab.example.com	172.25.250.253	수강생 네트워크를 클러스터 네트워크 및 스토리지 서버에 연결하는 라우터
master01.ocp4.example.com	192.168.50.10	컨트롤 플레인 및 계산 노드
master02.ocp4.example.com	192.168.50.11	컨트롤 플레인 및 계산 노드
master03.ocp4.example.com	192.168.50.12	컨트롤 플레인 및 계산 노드

인터넷 서비스에 대한 종속성

Red Hat OpenShift Container Platform 4를 사용하려면 두 개의 컨테이너 레지스트리에 액세스하여 운영자, S2I 빌더 및 기타 클러스터 서비스에 대한 컨테이너 이미지를 다운로드해야 합니다. 해당 레지스트리는 다음과 같습니다.

- `registry.redhat.io`
- `quay.io`

강의실 환경을 시작할 때 두 레지스트리 중 하나를 사용할 수 없는 경우 OpenShift 클러스터가 시작되지 않았거나 성능이 저하된 상태일 수 있습니다.

강의실 환경이 시작되어 실행되고 있는 동안 이러한 컨테이너 레지스트리가 중단되는 경우, 중단이 해결될 때까지 연습을 완료하지 못할 수 있습니다.

전용 OpenShift 클러스터

강의실 환경 내의 Red Hat OpenShift Container Platform 4 클러스터는 기존 인프라 설치 방법을 사용하여 사전 설치됩니다. 모든 노드는 실제로 OpenStack 클러스터의 가상 시스템인 경우에도 베어 메탈 서버로 취급됩니다.

OpenShift 클라우드-프로바이더 통합 기능은 활성화되어 있지 않으며, 이러한 통합 기능을 사용하는 일부 기능(예: 시스템 설정 및 클러스터 노드 자동 확장)을 사용할 수 없습니다.

OpenShift 클러스터에 대한 액세스 복원

대부분의 랩 스크립트 시작 기능을 통해 연습과 관련된 적절한 사전 요구 사항이 완료되었는지 확인할 수 있습니다. DO285에는 두 개의 개별 교육 과정에 있는 랩 스크립트가 합쳐져 있으므로 연습을 순차적으로 수행하지 않으면 문제가 발생할 수 있습니다. 교육 과정의 DO180 부분에서 문제가 발생하면 `lab-configure` 명령을 실행하여 OpenShift 클러스터에 암호 `developer`를 사용하여 `developer` 사용자로 액세스할 수 있는지 확인하십시오. `lab-configure -d`를 실행하면 새 GitHub 및 Quay.io 사용자 이름을 입력할 수 있습니다.

소개

교육 과정의 DO280 부분에 있는 대부분의 연습에서는 **admin** 및 **developer** OpenShift 사용자에게 액세스 권한을 제공합니다. 클러스터 인증 설정을 잘못 변경하여 OpenShift 클러스터에 **admin** 사용자로 로그인 할 수 없는 것으로 의심되는 경우, 현재 연습에서 **lab finish** 명령을 실행한 후 **lab start** 명령을 실행하여 연습을 재시작합니다.

admin과 **developer** 사용자가 필요한 랩에서는 **lab** 명령을 실행하면 클러스터 인증 설정이 재설정되고, 암호가 복원되어 **admin** 사용자의 암호는 **redhat**으로, **developer** 사용자의 암호는 **developer**로 설정됩니다.

lab 명령을 실행해도 문제가 해결되지 않는 경우에는 다음 섹션의 지침에 따라 **utility** 시스템을 사용하여 OpenShift 클러스터에 액세스할 수 있습니다.

OpenShift 클러스터에 대한 액세스 문제 해결

utility 시스템은 강의실 환경 내부에서 OpenShift 설치 프로그램을 실행하는 데 사용되었으며, 클러스터 문제를 해결하는 데 유용한 리소스입니다. 설치 프로그램 매니페스트는 **utility** 시스템의 `/home/lab/ocp4` 폴더에서 확인할 수 있습니다.

연습을 수행하기 위해 **utility** 서버에 로그인할 필요는 없습니다. OpenShift 클러스터를 시작하는 데 시간이 너무 오래 걸리거나 성능이 저하된 상태인 경우, 강의실 환경의 문제를 해결하기 위해 **utility** 시스템에 **lab** 사용자로 로그인할 수 있습니다.

workstation 시스템의 **student** 사용자는 암호 없이 **utility** 시스템에 로그인할 수 있도록 SSH 키를 사용하여 이미 구성되어 있습니다.

```
[student@workstation ~]$ ssh lab@utility
```

utility 시스템에서 **lab** 사용자는 먼저 **oc login**을 요구하지 않고 액세스 권한 **system:admin**을 부여하는 `.kube/config` 파일을 사용하여 사전 구성됩니다.

따라서 **workstation** 시스템에서 **oc get node**와 같은 문제 해결 명령을 실행하지 못하는 경우 이 시스템에서 실행할 수 있습니다.

OpenShift 4에서는 **oc debug** 명령을 제공하므로 일반 관리 작업을 위해 OpenShift 클러스터 노드에 SSH 액세스를 요구해서는 안 됩니다. 필요한 경우 **utility** 서버의 **lab** 사용자는 모든 클러스터 노드에 액세스 할 수 있도록 SSH 키를 사용하여 사전 구성됩니다. 예를 들면 다음과 같습니다.

```
[lab@utility ~]$ ssh -i ~/.ssh/lab_rsa core@master01.ocp4.example.com
```

위 예에서는 **master01**을 원하는 클러스터 노드 이름으로 교체합니다.

OpenShift 클러스터의 노드 인증서 승인

Red Hat OpenShift Container Platform 클러스터는 해제될 때까지 지속적으로 실행되도록 설계되었습니다. 프로덕션 클러스터와 달리 강의실 환경에는 설치 후 중지된 클러스터가 포함되어 있으며, 이 교육 과정을 마치기 전에 몇 번 중지되고 재시작됩니다. 이는 특별한 취급이 필요한 시나리오로, 프로덕션 클러스터에서는 필요하지 않습니다.

OpenShift 클러스터의 컨트롤 플레인과 계산 노드는 서로 자주 통신합니다. 모든 클러스터 노드 간 통신은 노드별 TLS 인증서를 기반으로 상호 인증을 통해 보호됩니다.

OpenShift 설치 프로그램은 풀스택 자동화 설치 방법에 필요한 TLS CSR(인증서 서명 요청)의 생성 및 승인을 처리합니다. 기존 인프라 설치 방법을 사용하려면 시스템 관리자가 이러한 CSR을 수동으로 승인해야 합니다.

소개

모든 노드별 TLS 인증서의 만료 기간은 24시간(최초) 및 30일(갱신 후)로 짧습니다. 만료일이 다가오면 영향을 받는 클러스터 노드에서 새 CSR을 생성하고 컨트롤 플레인에서 자동으로 승인합니다. 노드의 TLS 인증서가 만료될 때 컨트롤 플레인이나 오프라인 상태인 경우에는 클러스터 관리자가 보류 중인 CSR을 승인해야 합니다.

utility 시스템에는 강의실을 시작할 때 클러스터의 CSR을 승인하여 연습을 시작할 때 클러스터가 준비되었는지 확인하는 시스템 서비스가 포함되어 있습니다. 강의실을 만들거나 시작하고 연습을 너무 빨리 시작하는 경우 클러스터가 아직 준비되지 않을 수 있습니다. 이 경우 **utility** 시스템에서 CSR을 처리하는 동안 몇 분 정도 기다렸다가 다시 시도하십시오.

예를 들면 클러스터에서 필요한 모든 CSR 요청을 생성하는 데 시간이 너무 오래 걸리고 시스템 서비스에서 오랫동안 대기하지 않아 **utility** 시스템에서 필요한 모든 CSR을 승인하지 못하는 경우가 있습니다. 또한 일부 OpenShift 클러스터 노드에서 CSR이 승인될 때까지 충분히 기다리지 않고 이전 CSR을 대체하는 새 CSR을 발행할 수 있습니다.

이러한 문제가 발생하면 클러스터가 표시되는 데 너무 오래 걸려 **oc login** 또는 **lab** 명령이 계속 실패합니다. 이 문제를 해결하려면 위에서 설명한 대로 **utility** 시스템에 로그인한 후 **sign.sh** 스크립트를 실행하여 추가 및 보류 중인 CSR을 승인하면 됩니다.

```
[lab@utility ~]$ ./sign.sh
```

sign.sh 스크립트는 해당 클러스터 노드에서 승인한 CSR을 대체하는 새 CSR을 발행하는 경우를 대비하여 몇 번 반복됩니다.

직접 승인하거나, **utility** 시스템의 시스템 서비스에서 모든 CSR을 승인한 후에는 OpenShift에서 일부 클러스터 운영자를 재시작해야 합니다. OpenShift 클러스터가 클라이언트의 요청에 응답할 준비가 될 때까지 몇 분 정도 걸립니다. **utility** 시스템에서는 이러한 시나리오를 처리할 수 있도록 OpenShift 클러스터가 원격 클라이언트의 인증 및 API 요청을 수락할 준비가 될 때까지 기다리도록 하는 **wait.sh** 스크립트를 제공합니다.

```
[lab@utility ~]$ ./wait.sh
```

드문 경우지만 **utility** 시스템의 서비스를 사용하거나 **sigh.sh** 및 **wait.sh** 스크립트를 실행해도 OpenShift 클러스터를 통해 연습을 시작할 수 없는 경우에는 고객 지원 티켓을 여십시오.



참고

준비되지 않은 컨트롤 플레인 노드가 있는 경우에도 언제든 **utility** 시스템에서 문제 해결 명령을 실행할 수 있습니다. 다음은 몇 가지 유용한 명령입니다.

- **oc get node**: 모든 클러스터 노드가 준비되었는지 확인합니다.
- **oc get csr**: 클러스터에 승인되지 않고 보류 중인 CSR이 있는지 확인합니다.
- **oc get co**: 사용할 수 없거나 성능이 저하된 상태이거나 구성이 진행하여 포드를 롤아웃 중인 클러스터 운영자가 있는지 확인합니다.

해당 명령이 실패하는 경우 고객 지원 티켓을 생성하기 전에 마지막 수단으로 강의실을 삭제한 후 다시 만들어 볼 수 있습니다.

시스템 관리

Red Hat 온라인 학습 강의실의 원격 컴퓨터가 할당됩니다. 해당 컴퓨터는 rol.redhat.com [<http://rol.redhat.com>]. Red Hat Customer Portal 사용자 자격 증명을 사용하여 이 사이트에 로그인해야 합니다.

소개

가상 시스템 제어

강의실 환경의 가상 컴퓨터는 웹 페이지를 통해 제어됩니다. 강의실의 각 가상 시스템 상태는 **Online Lab(온라인 랩)** 탭에 있는 페이지에 표시됩니다.

시스템 상태

가상 시스템 상태	설명
STARTING	가상 시스템이 부팅 중입니다.
STARTED	가상 시스템이 실행 중이고 사용 가능합니다(또는 부팅 중인 경우 곧 사용 가능 상태가 됨).
STOPPING	가상 시스템이 종료되는 중입니다.
STOPPED	가상 시스템이 완전히 종료되었습니다. 가상 시스템을 시작하면 종료했을 때와 동일한 상태로 부팅됩니다(디스크가 유지됨).
PUBLISHING	가상 시스템의 초기 생성이 수행 중입니다.
WAITING_TO_START	다른 가상 시스템이 시작될 때까지 가상 시스템이 기다리는 중입니다.

시스템 상태에 따라 다음 작업을 선택할 수 있습니다.

강의실/시스템 작업

버튼 또는 작업	설명
PROVISION LAB(랩 프로비저닝)	ROL 강의실을 생성합니다. 강의실에 필요한 모든 가상 시스템이 생성 및 시작되며, 완료되는 데 몇 분이 소요됩니다.
DELETE LAB(랩 삭제)	ROL 강의실을 삭제합니다. 강의실의 모든 가상 컴퓨터를 제거합니다. 주의: 디스크에 생성된 모든 작업이 제거됩니다.
START LAB(랩 시작)	강의실의 모든 가상 시스템을 시작합니다.
SHUTDOWN LAB(랩 종료)	강의실의 모든 가상 시스템을 중지합니다.
OPEN CONSOLE(콘솔 열기)	브라우저에서 새 탭을 열고 가상 시스템의 콘솔에 연결합니다. 가상 시스템에 직접 로그인하고 명령을 실행할 수 있습니다. 대부분의 경우 workstation 가상 시스템에 로그인하고 ssh 를 사용하여 다른 가상 시스템에 연결합니다.
ACTION(작업) → Start(시작)	가상 시스템을 시작합니다(전원 켜기).
ACTION(작업) → Shutdown(종료)	가상 시스템을 정상적으로 종료하고 디스크 컨텐츠를 유지합니다.
ACTION(작업) → Power Off(전원 끄기)	가상 시스템을 강제로 종료하고 디스크 컨텐츠를 유지합니다. 이 경우 실제 시스템에서 전원을 분리하는 것과 동일합니다.
ACTION(작업) → Reset(재설정)	가상 시스템을 강제로 종료하고 디스크를 초기 상태로 재설정합니다. 주의: 디스크에 생성된 모든 작업이 제거됩니다.

연습을 시작할 때 단일 가상 시스템 노드를 재설정하라는 지침이 나타나면 특정 가상 시스템에 대해서만 ACTION(작업) → Reset(재설정)을 클릭합니다.

연습을 시작할 때 가상 시스템을 모두 재설정하라는 지침이 나타나면 ACTION(작업) → Reset(재설정)을 클릭합니다.

교육 과정 시작 시 강의실 환경을 원래 상태로 되돌리려면 Delete LAB(LAB 삭제)을 클릭하여 전체 강의실 환경을 제거할 수 있습니다. 랩이 삭제되면 PROVISION LAB(랩 프로비전)을 클릭하여 새 강의실 시스템을 프로비전할 수 있습니다.



경고

DELETE LAB(LAB 삭제) 작업은 실행 취소할 수 없습니다. 해당 시점까지 강의실 환경에서 완료한 모든 작업이 사라집니다.

자동 중지 타이머

Red Hat 온라인 학습에 등록할 경우 일정 시간 동안 컴퓨터를 사용할 수 있습니다. ROL 강의실에는 할당된 컴퓨터 사용 시간을 보존하기 위해 타이머 만료 시 강의실 환경을 종료하는 관련 카운트다운 타이머가 있습니다.

타이머를 조정하려면 +를 클릭하여 타이머에 1시간을 추가합니다. classroom 시스템이 자동으로 중지되어야 하는 시간 값을 설정하십시오. 최대 시간은 10시간입니다.

국제화

사용자별 언어 선택

사용자가 시스템 전체 기본 언어가 아닌 다른 언어를 해당 데스크탑 환경에 사용하려고 할 수 있습니다. 다른 키보드 레이아웃이나 입력 방법을 해당 계정에 사용하려고 할 수도 있습니다.

언어 설정

GNOME 데스크탑 환경에서는 사용자가 처음 로그인할 때 선호하는 언어와 입력 방법을 설정하라는 화면이 표시될 수 있습니다. 그렇지 않은 경우 개별 사용자가 선호하는 언어와 입력 방법 설정을 가장 쉽게 조절하는 방법은 Region & Language(국가별 설정 및 언어) 애플리케이션을 사용하는 것입니다.

이 애플리케이션은 두 가지 방법으로 시작할 수 있습니다. 터미널 창에서 **gnome-control-center** 명령을 실행하거나, 상단 표시줄의 오른쪽에 있는 시스템 메뉴의 왼쪽 아래에서 설정 버튼(십자형 드라이버와 랜치 아이콘)을 선택합니다.

열리는 창에서 Region & Language(국가별 설정 및 언어)를 선택합니다. 사용자가 **Language(언어)** 상자를 클릭하고 표시되는 목록에서 선호하는 언어를 선택합니다. 그러면 **Formats(포맷)** 설정을 해당 언어의 기본 설정으로 업데이트합니다. 다음에 로그인하면 이러한 변경 사항이 완전히 적용됩니다.

이러한 설정은 GNOME 데스크탑 환경과 그 안에서 시작되는 **gnome-terminal** 등의 모든 애플리케이션에 영향을 미칩니다. 그러나 원격 시스템에서의 **ssh** 로그인 또는 가상 콘솔에서의 텍스트 기반 로그인(예: **tty5**)을 통해 액세스한 경우에는 해당 계정에 기본적으로 적용되지 않습니다.



참고

텍스트 기반 가상 콘솔 또는 **ssh**를 통해 로그인할 때에도 쉘 환경이 그래픽 환경과 동일한 **LANG** 설정을 사용하도록 만들 수 있습니다. 이처럼 하는 방법 중 하나는 다음과 유사한 코드를 **~/.bashrc** 파일에 추가하는 것입니다. 이 예제 코드는 텍스트 로그인에 사용되는 언어를 사용자의 GNOME 데스크탑 환경에 현재 설정된 언어와 일치하도록 설정합니다.

```
i=$(grep 'Language=' /var/lib/AccountsService/users/${USER} \
| sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

일본어, 한국어, 중국어 및 라틴어 이외의 문자 집합을 사용하는 기타 언어는 텍스트 기반 가상 콘솔에서 올바르게 표시되지 않을 수 있습니다.

명령줄에서 **LANG** 변수를 설정하여 개별 명령이 다른 언어를 사용하도록 만들 수 있습니다.

```
[user@host ~]$ LANG=fr_FR.utf8 date
jeu. avril 25 17:55:01 CET 2019
```

후속 명령은 시스템의 출력 기본 언어 사용으로 되돌립니다. **locale** 명령을 사용하여 현재 **LANG** 값과 다른 관련 환경 변수를 판별할 수 있습니다.

소개

입력 방법 설정

Red Hat Enterprise Linux 7 이상의 GNOME 3은 IBus 입력 방법 선택 시스템을 자동으로 사용하므로 키보드 레이아웃 및 입력 방법을 빠르고 쉽게 변경할 수 있습니다.

Region & Language(국가별 설정 및 언어) 애플리케이션도 대체 입력 방법을 활성화하는데 사용할 수 있습니다. Region & Language(국가별 설정 및 언어) 애플리케이션 창의 **Input Sources(입력 소스)** 상자에는 현재 사용 가능한 입력 방법이 표시됩니다. 기본적으로 **English (US)(영어(미국))**만 사용 가능한 방법입니다. **English (US)(영어(미국))**를 강조 표시한 다음 키보드 아이콘을 클릭하여 현재 **키보드** 레이아웃을 확인합니다.

다른 입력 방법을 추가하려면 **Input Sources(입력 소스)** 창의 왼쪽 하단에서 **+** 버튼을 클릭합니다. **Add an Input Source(입력 소스 추가)** 창이 열립니다. 언어를 선택한 다음 선호하는 입력 방법 또는 키보드 레이아웃을 선택합니다.

입력 방법을 두 가지 이상 구성한 경우에는 **Super+Space(Windows+Space라고도 함)**를 입력하여 빠르게 전환할 수 있습니다. GNOME 상단 표시줄에 나타나는 상태 표시등에는 두 가지 기능이 있습니다. 하나는 활성화된 입력 방법이 무엇인지 표시하는 것이고, 다른 하나는 입력 방법 사이를 전환하거나 입력 방법이 더 복잡한 고급 기능을 선택하는 데 사용할 수 있는 메뉴로 사용되는 것입니다.

기어 표시가 있는 일부 방법은 고급 구성 옵션 및 기능이 있음을 나타냅니다. 예를 들어 **Japanese (Kana Kanji)** 입력 방법에서는 사용자가 라틴으로 텍스트를 사전 편집하고 아래쪽 화살표와 위쪽 화살표 키를 사용하여 사용할 올바른 문자를 선택할 수 있습니다.

미국 영어 사용자도 이 기능이 유용할 수 있습니다. 예를 들어 **English (United States)(영어(US))**에는 **English (international AltGr dead keys)(영어(국제 AltGr 데드 키))** 키보드 레이아웃이 있습니다. 이것은 PC 104/105키 키보드의 **AltGr**(또는 오른쪽 **Alt**)을 추가 문자를 입력하기 위한 "보조 Shift" 수식어 키 및 데드 키 활성화 키로 취급합니다. 또한 드보락 및 다른 레이아웃도 사용할 수 있습니다.



참고

문자의 유니코드 코드 포인트를 아는 경우 GNOME 데스크탑 환경에서 유니코드 문자를 입력할 수 있습니다. **Ctrl+Shift+U**와 코드 포인트를 차례로 입력하면 됩니다.

Ctrl+Shift+U를 입력하면 **u** 아래에 밑줄이 표시되는데, 이는 시스템이 Unicode 코드 포인트 입력을 기다리고 있음을 의미합니다.

예를 들어 그리스어 소문자 람다의 코드 포인트는 U+03BB이며, **Ctrl+Shift+U**와 **03BB**를 차례로 입력한 다음 **Enter**를 눌러 입력할 수 있습니다.

시스템 전체 기본 언어 설정

시스템의 기본 언어는 Unicode의 UTF-8 인코딩을 문자 집합으로 사용하는 미국 영어로 설정되어 있으나 (**en_US.utf8**) 설치하는 동안 또는 설치 후에 변경할 수 있습니다.

명령줄에서 **root** 사용자는 **localectl** 명령으로 시스템 전체 로케일 설정을 변경할 수 있습니다. **localectl**을 인수 없이 실행한 경우 현재 시스템 전체 로케일 설정이 표시됩니다.

시스템 전체 기본 언어를 설정하려면 **localectl set-locale LANG=locale** 명령을 실행합니다. 여기서 **locale**은 이 장의 "언어 코드 참조" 테이블에 있는 **LANG** 환경 변수의 해당 값입니다. 변경 사항은 사용자가 다음에 로그인할 때 적용되며 **/etc/locale.conf**에 저장됩니다.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

소개

GNOME의 관리자는 Region & Language(국가별 설정 및 언어)에서 창의 오른쪽 위에 있는 **Login Screen**(로그인 화면) 버튼을 클릭하여 이 설정을 변경할 수 있습니다. 그래픽 로그인 화면의 **Language**(언어)를 변경할 경우 **/etc/locale.conf** 구성 파일에 저장된 시스템 전체 기본 언어 설정도 조정됩니다.



중요

텍스트 기반 가상 콘솔(예: **tty4**)은 그래픽 환경을 실행하는 가상 콘솔의 터미널이나 **ssh** 세션을 위한 의사 터미널보다 표시할 수 있는 글꼴이 더 제한적입니다. 예를 들어 일본어, 한국어, 중국어 문자는 텍스트 기반 가상 콘솔에서 제대로 표시되지 않을 수 있습니다. 이러한 이유로 영어 또는 라틴 문자 집합을 포함하는 다른 언어를 시스템 전체 기본 언어로 사용하는 것 이 좋습니다.

마찬가지로, 텍스트 기반 가상 콘솔은 지원하는 입력 방법이 더 제한적이며 그래픽 데스크톱 환경과 별도로 관리됩니다. 텍스트 기반 가상 콘솔과 그래픽 환경 모두, **localectl**을 통해 사용 가능한 글로벌 입력 설정을 구성할 수 있습니다. 자세한 내용은 **localectl(1)** 및 **vconsole.conf(5)** 도움말 페이지를 참조하십시오.

언어 패키지

langpack이라는 특수 RPM 패키지는 특정 언어에 대한 지원을 추가하는 언어 패키지를 설치합니다. 이러한 **langpack**은 종속성을 사용하여 시스템의 다른 소프트웨어 패키지에 대한 지역화, 사전 및 번역이 포함된 추가 RPM 패키지를 자동으로 설치합니다.

설치된 **langpack**과 설치할 수 있는 **langpack**을 나열하려면 **yum list langpacks-***를 사용합니다.

```
[root@host ~]# yum list langpacks-*
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Installed Packages
langpacks-en.noarch      1.0-12.el8          @AppStream
Available Packages
langpacks-af.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-am.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-ar.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-as.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-ast.noarch      1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
...output omitted...
```

언어 지원을 추가하려면 해당 **langpack** 패키지를 설치합니다. 예를 들어 다음 명령은 프랑스어 지원을 추가합니다.

```
[root@host ~]# yum install langpacks-fr
```

yum repoquery --whatsonplements를 사용하여 **langpack**에서 설치할 수 있는 RPM 패키지를 확인합니다.

```
[root@host ~]# yum repoquery --whatsonplements langpacks-fr
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Last metadata expiration check: 0:01:33 ago on Wed 06 Feb 2019 10:47:24 AM CST.
glibc-langpack-fr-0:2.28-18.el8.x86_64
gnome-getting-started-docs-fr-0:3.28.2-1.el8.noarch
```

소개

```
hunspell-fr-0:6.2-1.el8.noarch
hyphen-fr-0:3.0-1.el8.noarch
libreoffice-langpack-fr-1:6.0.6.1-9.el8.x86_64
man-pages-fr-0:3.70-16.el8.noarch
mythes-fr-0:2.3-10.el8.noarch
```

중요

langpack 패키지는 필요로 하는 핵심 패키지도 설치된 경우에만 보충 패키지를 설치하기 위해 RPM 약한 종속성을 사용합니다.

예를 들어, 앞의 예제와 같이 langpacks-fr을 설치하는 경우 mythes-fr 패키지는 mythes 등 의어 사전도 시스템에 설치된 경우에만 설치됩니다.

나중에 mythes를 해당 시스템에 설치하는 경우 이미 설치된 langpacks-fr 패키지와의 약한 종속성으로 인해 mythes-fr 패키지도 자동으로 설치됩니다.

참조

`locale(7)`, `localectl(1)`, `locale.conf(5)`, `vconsole.conf(5)`, `unicode(7)` 및 `utf-8(7)` 도움말 페이지

그래픽 데스크탑 환경의 X11 레이아웃과 `localectl`의 해당 이름 간 변환은 `/usr/share/X11/xkb/rules/base.lst` 파일에 있습니다.

언어 코드 참조

참고

시스템에서 사용 가능한 일부 langpack이 테이블에 반영되지 않을 수도 있습니다. 특정 langpack 패키지에 대한 자세한 내용을 보려면 `yum info langpacks-SUFFIX`를 사용 합니다.

언어 코드

언어	langpack 접미사	\$LANG 값
영어(미국)	en	en_US.utf8
아랍어	as	ar_IN.utf8
벵골어	bn	bn_IN.utf8
중국어(간체)	zh_CN	zh_CN.utf8
중국어(번체)	zh_TW	zh_TW.utf8
프랑스어	fr	fr_FR.utf8
독일어	de	de_DE.utf8
구자라트어	gu	gu_IN.utf8

언어	langpack 접미사	\$LANG 값
힌디어	hi	hi_IN.utf8
이탈리아어	IT	it_IT.utf8
일본어	ja	ja_JP.utf8
칸나다어	kn	kn_IN.utf8
한국어	ko	ko_KR.utf8
말라얄람어	ml	ml_IN.utf8
마라티어	mr	mr_IN.utf8
오디아어	또는	or_IN.utf8
포르투갈어(브라질)	pt_BR	pt_BR.utf8
펀자브어	pa	pa_IN.utf8
러시아어	ru	ru_RU.utf8
스페인어	es	es_ES.utf8
타밀어	ta	ta_IN.utf8
텔루구어	te	te_IN.utf8

1장

컨테이너 기술 소개

목적

Red Hat OpenShift Container Platform이 오케스트레이션한 컨테이너에서 애플리케이션이 실행되는 방법을 설명합니다.

목표

- 컨테이너 애플리케이션과 기존 배포의 차이점을 설명합니다.
- 컨테이너 아키텍처의 기본 사항을 설명합니다.
- 애플리케이션과 OpenShift Container Platform을 오케스트레이션하는 이점을 설명합니다.

섹션

- 컨테이너 기술 개요 및 퀴즈
- 컨테이너 아키텍처 개요 및 퀴즈
- Kubernetes 및 OpenShift 개요 및 퀴즈
- 안내에 따른 연습: 강의실 환경 구성

컨테이너 기술 개요

목표

이 섹션을 완료한 수강생은 컨테이너 애플리케이션과 기존 배포의 차이점을 설명할 수 있게 됩니다.

컨테이너화된 애플리케이션

소프트웨어 애플리케이션은 일반적으로 런타임 환경에서 제공하는 다른 라이브러리, 구성 파일 또는 서비스를 사용합니다. 소프트웨어 애플리케이션의 기존 런타임 환경은 물리적 호스트 또는 가상 시스템이며, 애플리케이션 종속성이 호스트의 일부로서 설치됩니다.

예를 들어 TLS 프로토콜을 구현하는 공통 공유 라이브러리에 액세스해야 하는 Python 애플리케이션을 생각해보십시오. 일반적으로 시스템 관리자는 Python 애플리케이션을 설치하기 전에 공유 라이브러리를 제공하는 필수 패키지를 설치합니다.

기존에 배포된 소프트웨어 애플리케이션의 주요 단점은 애플리케이션의 종속성이 런타임 환경과 얹혀 있다는 점입니다. 업데이트 또는 패치가 기본 OS(운영 체제)에 적용되면 애플리케이션이 중단될 수 있습니다.

예를 들어 TLS 공유 라이브러리에 대한 업데이트를 OS에 적용하면 지원되는 TLS 프로토콜이 제거됩니다. 이 경우 배포된 Python 애플리케이션은 네트워크 요청에 TLS 1.0 프로토콜을 사용하도록 작성되었기 때문에 중단됩니다. 이로 인해 시스템 관리자가 애플리케이션을 계속 실행하기 위해 OS 업데이트를 롤백해야 하므로, 다른 애플리케이션이 업데이트된 패키지의 이점을 사용할 수 없게 됩니다. 따라서 기존의 소프트웨어 애플리케이션을 개발하는 회사는 OS 업데이트가 호스트에서 실행 중인 애플리케이션에 영향을 미치지 않도록 보장하기 위해 전체 테스트를 수행해야 할 수 있습니다.

또한, 기존에 배포된 애플리케이션은 관련 종속성을 업데이트하기 전에 중지해야 합니다. 애플리케이션 다운 타임을 최소화하기 위해 조직은 복잡한 시스템을 설계 및 구현하여 애플리케이션의 고가용성을 제공합니다. 단일 호스트에서 여러 애플리케이션을 유지 관리하는 것은 번거로운 경우가 많으며, 배포 또는 업데이트로 인해 조직의 애플리케이션 중 하나가 중단될 수 있습니다.

그림 1.1에서는 컨테이너로 실행 중인 애플리케이션과 호스트 운영 체제에서 실행 중인 애플리케이션 간의 차이점을 설명합니다.

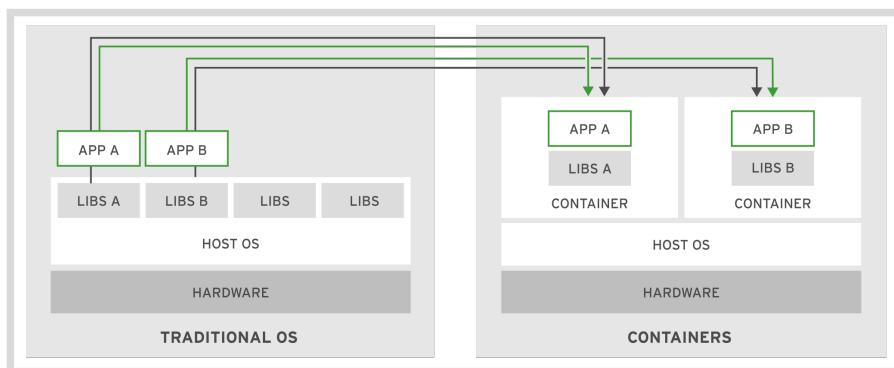


그림 1.1: 컨테이너 및 운영 체제의 차이

또는 소프트웨어 애플리케이션을 컨테이너를 사용하여 배포할 수 있습니다. 컨테이너는 나머지 시스템에서 격리된 하나 이상의 프로세스로 이뤄진 세트입니다. 컨테이너에는 보안, 스토리지, 네트워크 격리 등 가상 시스템과 동일한 많은 이점이 있습니다. 컨테이너는 하드웨어 리소스가 훨씬 적게 필요하며, 신속하게 시작하

1장 | 컨테이너 기술 소개

고종료할 수 있습니다. 또한 그림 1.1에 설명된 대로, 애플리케이션을 위한 라이브러리 및 런타임 리소스(예: CPU 및 스토리지)를 격리하여 호스트 OS에 대한 OS 업데이트의 영향을 최소화합니다.

컨테이너를 사용하면 호스팅된 애플리케이션의 효율성, 탄력성 및 재사용성뿐 아니라 애플리케이션 이식성에도 도움이 됩니다. Open Container Initiative는 컨테이너 런타임 사양 및 컨테이너 이미지 사양을 정의하는 일련의 산업 표준을 제공합니다. 이미지 사양은 컨테이너 이미지를 형성하는 파일 및 메타데이터로 이루어진 번들의 포맷을 정의합니다. OCI 표준을 준수하는 컨테이너 이미지로 애플리케이션을 빌드하는 경우 OCI 규격 컨테이너 엔진을 사용하여 애플리케이션을 실행할 수 있습니다.

Rocket, Drawbridge, LXC, Docker 및 Podman을 포함하여 개별 컨테이너를 관리하고 실행할 수 있는 많은 컨테이너 엔진이 있습니다. Podman은 Red Hat Enterprise Linux 7.6 이상 버전에서 제공되며, 이 교육 과정에서는 개별 컨테이너를 시작, 관리 및 종료하는 데 사용됩니다.

다음은 컨테이너를 사용하여 얻을 수 있는 다른 주요 이점입니다.

적은 하드웨어 풋프린트

컨테이너는 OS 내부 기능을 사용하여 네임스페이스, cGroups 등의 OS 기능을 통해 리소스가 관리되는 격리된 환경을 생성합니다. 이 접근법은 가상 시스템 하이퍼바이저와 비교하여 CPU와 메모리 오버헤드의 양을 최소화합니다. VM에서 애플리케이션을 실행하는 것은 실행 중인 환경에서 격리를 생성하는 방법이지만 컨테이너에서 제공하는 것과 동일한 낮은 하드웨어 풋프린트 격리를 지원하려면 무거운 서비스 계층이 필요합니다.

환경 격리

컨테이너는 호스트 OS 또는 기타 애플리케이션의 변경 사항이 컨테이너에 영향을 미치지 않는 폐쇄된 환경에서 작업합니다. 컨테이너에서 필요로 하는 라이브러리가 자체 포함되어 있으므로 애플리케이션은 중단 없이 실행할 수 있습니다. 예를 들어 각 애플리케이션은 자체 라이브러리 집합이 포함된 자체 컨테이너에 있을 수 있습니다. 한 컨테이너의 업데이트는 다른 컨테이너에 영향을 미치지 않습니다.

빠른 배포

전체 기본 운영 체제를 설치할 필요가 없으므로 컨테이너가 빠르게 배포됩니다. 일반적으로 격리를 지원하기 위해 새로운 OS 설치가 물리적 호스트 또는 VM에 필요하며 간단한 업데이트 때문에 전체 OS 재시작이 필요할 수 있습니다. 컨테이너를 재시작할 때 호스트 OS의 서비스를 중지하지 않아도 됩니다.

다중 환경 배포

단일 호스트를 사용하는 기존의 배포 시나리오에서는 환경의 차이로 인해 애플리케이션이 중단될 수 있습니다. 그러나 컨테이너를 사용하면 모든 애플리케이션 종속성 및 환경 설정이 컨테이너 이미지에 캡슐화됩니다.

재사용성

전체 OS를 설치할 필요 없이 동일한 컨테이너를 재사용할 수 있습니다. 예를 들어 프로덕션 데이터베이스 서비스를 제공하는 동일한 데이터베이스 컨테이너를 각 개발자가 사용하여 애플리케이션 개발 중에 개발 데이터베이스를 생성할 수 있습니다. 컨테이너를 사용하면 더 이상 별도의 프로덕션 및 개발 데이터베이스 서버를 유지 관리할 필요가 없습니다. 단일 컨테이너 이미지가 데이터베이스 서비스 인스턴스를 생성하는 데 사용됩니다.

일반적으로 소프트웨어 애플리케이션과 모든 해당 종속 서비스(데이터베이스, 메시징, 파일 시스템)는 단일 컨테이너에서 실행되도록 개발되었습니다. 이 경우 가상 시스템 또는 물리적 호스트에 대한 기존의 소프트웨어 배포와 관련된 동일한 문제가 발생할 수 있습니다. 이러한 경우에는 멀티 컨테이너 배포가 더 적합할 수 있습니다.

또한, 컨테이너는 애플리케이션 개발에 마이크로서비스를 사용할 때 이상적인 접근법입니다. 각 서비스는 프로덕션 또는 개발 환경에 배포될 수 있는 안정적인 경량 컨테이너 환경에 캡슐화됩니다. 애플리케이션에 필요한 컨테이너화된 서비스 컬렉션을 단일 시스템에서 호스팅할 수 있으므로 각 서비스에 대한 시스템을 관리할 필요가 없습니다.

반면, 컨테이너화된 환경에 적합하지 않은 애플리케이션이 많습니다. 예를 들어 메모리, 파일 시스템, 장치 등 낮은 수준의 하드웨어 정보에 액세스하는 애플리케이션은 컨테이너 제한 사항으로 인해 불안정할 수 있습니다.



참조

홈 - Open Containers Initiative

<https://www.opencontainers.org/>

▶ 퀴즈

컨테이너 기술 개요

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너에서 실행할 수 있는 소프트웨어 애플리케이션의 예에 대한 두 가지 옵션은 무엇입니까? (두 개를 선택하십시오.)

- a. MySQL 데이터베이스, 파일 전송 프로토콜(FTP) 서버 및 단일 물리적 호스트의 웹 서버 등의 서비스에 액세스하는 데이터베이스 주도의 Python 애플리케이션.
- b. Oracle 데이터베이스가 포함된 Java Enterprise Edition 애플리케이션 및 단일 VM에서 실행 중인 메시지 브로커.
- c. 트래픽 및 블록 데이터 전송을 분석하는 I/O 모니터링 툴.
- d. 디버깅을 목적으로 모든 메모리 CPU 캐시에서 스냅샷을 찍을 수 있는 메모리 덤프 애플리케이션 툴.

▶ 2. 다음 중 컨테이너에 가장 적합한 두 가지 사용 사례는 무엇입니까? (두 개를 선택하십시오.)

- a. 소프트웨어 프로바이더는 오류가 없으며 빠른 방식으로 다른 회사에서 재사용할 수 있는 소프트웨어를 배포해야 합니다.
- b. 회사는 물리적 호스트에서 애플리케이션을 배포하고 있으며 컨테이너를 사용하여 성능을 개선하고 싶어 합니다.
- c. 회사의 개발자는 개발한 코드를 신속하게 테스트할 수 있도록 프로덕션 환경을 모방한 일회용 환경이 필요합니다.
- d. 금융 회사는 필요한 프로세서의 수를 최소화하도록 자체 컨테이너에서 CPU 사용량이 많은 위험 분석 툴을 구현하고 있습니다.

▶ 3. 회사는 새 아키텍처에 동일한 호스트에서 실행 중인 PHP 및 Python 애플리케이션을 마이그레이션하고 있습니다. 내부 정책으로 인해 둘 다 OS의 사용자 개발 공유 라이브러리 집합을 사용하고 있지만, Python 개발 팀 요청의 결과로 적용된 최신 업데이트가 PHP 애플리케이션을 중단했습니다. 다음 중 두 애플리케이션 모두에 대한 최적의 지원을 제공하는 아키텍처 두 개는 무엇입니까? (두 개를 선택하십시오.)

- a. 각 애플리케이션을 다른 VM에 배포하고 사용자 개발 공유 라이브러리를 개별적으로 각 VM 호스트에 적용합니다.
- b. 각 애플리케이션을 다른 컨테이너에 배포하고 사용자 개발 공유 라이브러리를 개별적으로 각 컨테이너에 적용합니다.
- c. 각 애플리케이션을 다른 VM에 배포하고 사용자 개발 공유 라이브러리를 모든 VM 호스트에 적용합니다.
- d. 각 애플리케이션을 다른 컨테이너에 배포하고 사용자 개발 공유 라이브러리를 모든 컨테이너에 적용합니다.

▶ 4. 다음 중 즉각적인 사용을 위해 컨테이너로 패키징할 수 있는 애플리케이션 종류 세 가지는 무엇입니까? (세 개를 선택하십시오.)

- a. 가상 시스템 하이퍼바이저
- b. 블로그 소프트웨어(예: WordPress)
- c. 데이터베이스
- d. 로컬 파일 시스템 복구 툴
- e. 웹 서버

▶ 솔루션

컨테이너 기술 개요

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너에서 실행할 수 있는 소프트웨어 애플리케이션의 예에 대한 두 가지 옵션은 무엇입니까? (두 개를 선택하십시오.)

- a. MySQL 데이터베이스, 파일 전송 프로토콜(FTP) 서버 및 단일 물리적 호스트의 웹 서버 등의 서비스에 액세스하는 데이터베이스 주도의 Python 애플리케이션.
- b. Oracle 데이터베이스가 포함된 Java Enterprise Edition 애플리케이션 및 단일 VM에서 실행 중인 메시지 브로커.
- c. 트래픽 및 블록 데이터 전송을 분석하는 I/O 모니터링 툴.
- d. 디버깅을 목적으로 모든 메모리 CPU 캐시에서 스냅샷을 찍을 수 있는 메모리 덤프 애플리케이션 툴.

▶ 2. 다음 중 컨테이너에 가장 적합한 두 가지 사용 사례는 무엇입니까? (두 개를 선택하십시오.)

- a. 소프트웨어 프로바이더는 오류가 없으며 빠른 방식으로 다른 회사에서 재사용할 수 있는 소프트웨어를 배포해야 합니다.
- b. 회사는 물리적 호스트에서 애플리케이션을 배포하고 있으며 컨테이너를 사용하여 성능을 개선하고 싶어 합니다.
- c. 회사의 개발자는 개발한 코드를 신속하게 테스트할 수 있도록 프로덕션 환경을 모방한 일회용 환경이 필요합니다.
- d. 금융 회사는 필요한 프로세서의 수를 최소화하도록 자체 컨테이너에서 CPU 사용량이 많은 위험 분석 툴을 구현하고 있습니다.

▶ 3. 회사는 새 아키텍처에 동일한 호스트에서 실행 중인 PHP 및 Python 애플리케이션을 마이그레이션하고 있습니다. 내부 정책으로 인해 둘 다 OS의 사용자 개발 공유 라이브러리 집합을 사용하고 있지만, Python 개발 팀 요청의 결과로 적용된 최신 업데이트가 PHP 애플리케이션을 중단했습니다. 다음 중 두 애플리케이션 모두에 대한 최적의 지원을 제공하는 아키텍처 두 개는 무엇입니까? (두 개를 선택하십시오.)

- a. 각 애플리케이션을 다른 VM에 배포하고 사용자 개발 공유 라이브러리를 개별적으로 각 VM 호스트에 적용합니다.
- b. 각 애플리케이션을 다른 컨테이너에 배포하고 사용자 개발 공유 라이브러리를 개별적으로 각 컨테이너에 적용합니다.
- c. 각 애플리케이션을 다른 VM에 배포하고 사용자 개발 공유 라이브러리를 모든 VM 호스트에 적용합니다.
- d. 각 애플리케이션을 다른 컨테이너에 배포하고 사용자 개발 공유 라이브러리를 모든 컨테이너에 적용합니다.

▶ 4. 다음 중 즉각적인 사용을 위해 컨테이너로 패키징할 수 있는 애플리케이션 종류 세 가지는 무엇입니까? (세 개를 선택하십시오.)

- a. 가상 시스템 하이퍼바이저
- b. 블로그 소프트웨어(예: WordPress)
- c. 데이터베이스
- d. 로컬 파일 시스템 복구 툴
- e. 웹 서버

컨테이너 아키텍처 개요

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- Linux 컨테이너의 아키텍처를 설명합니다.
- `podman` 유틸리티를 설치하여 컨테이너를 관리합니다.

컨테이너 역사 소개

컨테이너는 최근 몇 년 동안 빠르게 인기를 얻었습니다. 그러나 컨테이너의 배경 기술은 상대적으로 오랜 시간 동안 사용되었습니다. 2001년에 Linux는 VServer라는 프로젝트를 시작했습니다. VServer는 고도의 격리 수준을 갖춘 단일 서버 내에서 전체 프로세스 세트를 실행하려는 첫 번째 시도였습니다.

'격리된 프로세스'라는 아이디어는 VServer 이후로 더욱 발전했으며, Linux 커널의 다음 기능을 중심으로 공식화되었습니다.

네임스페이스

커널은 대체로 모든 프로세스에서 볼 수 있는 특정 시스템 리소스를 네임스페이스 내에 배치하여 격리할 수 있습니다. 네임스페이스 내에서는 해당 네임스페이스의 멤버인 프로세스만 이러한 리소스를 확인할 수 있습니다. 네임스페이스에는 네트워크 인터페이스, 프로세스 ID 목록, 마운트 지점, IPC 리소스, 시스템의 호스트 이름 정보 등의 리소스가 포함될 수 있습니다.

컨트롤 그룹(cgroup)

컨트롤 그룹은 사용하는 리소스를 관리하고 제한하기 위해 프로세스 집합 및 하위 집합을 그룹으로 분할 합니다. 컨트롤 그룹은 프로세스가 사용할 수 있는 시스템 리소스의 양을 제한합니다. 이러한 제한으로 인해 하나의 프로세스가 호스트에서 너무 많은 리소스를 사용할 수 없게 됩니다.

Seccomp

2005년에 개발되어 2014년경에 컨테이너에 도입된 Seccomp는 프로세스에서 시스템 호출을 사용할 수 있는 방법을 제한합니다. Seccomp는 프로세스가 사용할 수 있는 시스템 호출, 매개 변수 및 파일 설명자를 허용 목록에 추가하여 프로세스에 대한 보안 프로필을 정의합니다.

SELinux

SELinux(Security-Enhanced Linux)는 프로세스에 대한 필수 액세스 제어 시스템입니다. Linux 커널은 SELinux를 사용하여 프로세스 간에 보호하고 실행 중인 프로세스에서 호스트 시스템을 보호합니다. 프로세스는 호스트 시스템 리소스에 대해 제한된 액세스가 있는 한정된 SELinux 유형으로 실행됩니다.

이러한 모든 혁신과 기능은 프로세스가 시스템 리소스에 액세스하는 동시에 격리된 상태로 실행될 수 있도록 하는 기본 개념에 초점을 맞춰져 있습니다. 이 개념은 컨테이너 기술의 기초이며 모든 컨테이너 구현의 기초입니다. 현재 컨테이너는 이러한 보안 기능을 사용하여 격리된 환경을 생성하는 Linux 커널의 프로세스입니다. 이 환경은 격리된 프로세스가 시스템이나 다른 컨테이너 리소스를 남용하는 것을 방지합니다.

컨테이너의 일반적인 사용 사례는 동일한 서비스(예: 데이터베이스 서버)의 여러 복제본을 동일한 호스트에서 사용하는 것입니다. 각 복제본에 격리된 리소스(파일 시스템, 포트, 메모리)가 있으므로 서비스에서 리소스 공유를 처리하지 않아도 됩니다. 격리는 오작동 서비스 또는 해로운 서비스가 동일한 호스트 또는 기본 시스템의 다른 서비스나 컨테이너에 영향을 미치지 않도록 보장합니다.

Linux 컨테이너 아키텍처 설명

Linux 커널 관점에서 컨테이너는 제한이 있는 프로세스입니다. 그러나 단일 바이너리 파일을 실행하는 대신, 컨테이너는 이미지를 실행합니다. 이미지는 파일 시스템의 파일, 설치된 패키지, 사용 가능한 리소스, 실행 중인 프로세스, 커널 모듈 등 프로세스를 실행하는데 필요한 모든 종속성이 포함된 파일 시스템 번들입니다.

실행 가능한 파일이 프로세스 실행의 기초인 것처럼, 이미지는 컨테이너 실행의 기초입니다. 실행 중인 컨테이너는 변경할 수 없는 이미지 보기 사용으로 여러 컨테이너가 동일한 이미지를 동시에 재사용할 수 있습니다. 이미지는 파일이므로 버전 관리 시스템에서 관리할 수 있으며, 컨테이너 및 이미지 프로비저닝의 자동화를 향상합니다.

컨테이너 이미지는 컨테이너 런타임에서 실행하기 위해 로컬로 사용할 수 있어야 하지만, 이미지는 일반적으로 이미지 리포지토리에 저장되고 유지 관리됩니다. 이미지 리포지토리는 이미지를 저장 및 검색할 수 있는 서비스(퍼블릭 또는 프라이빗)일 뿐입니다. 이미지 리포지토리가 제공하는 다른 기능으로는 원격 액세스, 이미지 메타데이터, 권한 부여, 이미지 버전 제어 등이 있습니다.

각각 다른 기능을 제공하는 아래의 이미지 리포지토리를 사용할 수 있습니다.

- Red Hat Container Catalog [<https://registry.redhat.io>]
- Docker Hub [<https://hub.docker.com>]
- Red Hat Quay [<https://quay.io/>]
- Google Container Registry [<https://cloud.google.com/container-registry/>]
- Amazon Elastic Container Registry [<https://aws.amazon.com/ecr/>]

이 교육 과정에서는 공용 이미지 레지스트리인 Quay를 사용하므로 여러 수강생이 서로 방해하는 일 없이 이미지로 작업할 수 있습니다.

Podman을 사용하여 컨테이너 관리

컨테이너, 이미지 및 이미지 레지스트리는 상호 작용할 수 있어야 합니다. 예를 들어 이미지를 빌드하여 이미지 레지스트리에 넣을 수 있어야 합니다. 또한 이미지 레지스트리에서 이미지를 검색하고 해당 이미지에서 컨테이너를 빌드할 수 있어야 합니다.

Podman은 컨테이너 및 컨테이너 이미지를 관리하고 이미지 레지스트리와 상호 작용할 수 있는 오픈소스 툴입니다. Podman은 다음과 같은 주요 기능을 제공합니다.

- Open Container Initiative [<https://www.opencontainers.org>] (OCI)에서 지정한 이미지 포맷을 사용합니다. 이러한 사양은 커뮤니티 기반의 표준 비소유 이미지 포맷을 정의합니다.
- Podman은 로컬 이미지를 로컬 파일 시스템에 저장합니다. 이렇게 하면 불필요한 클라이언트/서버 아키텍처 이용 또는 로컬 시스템에서 데몬 실행을 방지할 수 있습니다.
- Podman이 동일한 명령 패턴을 Docker CLI로 따르므로 새로운 툴 세트를 배울 필요가 없습니다.
- Podman은 Kubernetes와 호환됩니다. Kubernetes는 Podman을 사용하여 컨테이너를 관리할 수 있습니다.

현재 Podman은 Linux 시스템에서만 사용할 수 있습니다. Red Hat Enterprise Linux, Fedora 또는 유사한 RPM 기반 시스템에 Podman을 설치하려면 `sudo yum install podman` 또는 `sudo dnf install podman`을 실행합니다.



참조

Red Hat Quay Container Registry

<https://quay.io>

Podman 사이트

<https://podman.io/>

Open Container Initiative

<https://www.opencontainers.org>

▶ 퀴즈

컨테이너 아키텍처 개요

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너를 실행하는 데 사용되는 세 가지 Linux 기능은 무엇입니까? (세 개를 선택하십시오.)

- a. 네임스페이스
- b. 무결성 관리
- c. Security-Enhanced Linux
- d. 컨트롤 그룹

▶ 2. 다음 중 컨테이너 이미지를 가장 잘 설명하는 것은 무엇입니까?

- a. 컨테이너가 생성될 가상 시스템 이미지
- b. 컨테이너가 생성될 컨테이너 청사진
- c. 애플리케이션이 실행될 런타임 환경
- d. 레지스트리에서 사용하는 컨테이너의 색인 파일

▶ 3. 다음 중 컨테이너 아키텍처 구현 간에 공통적인 세 가지 구성 요소는 무엇입니까? (세 개를 선택하십시오.)

- a. 컨테이너 런타임
- b. 컨테이너 권한
- c. 컨테이너 이미지
- d. 컨테이너 레지스트리

▶ 4. Linux 커널과 관련된 컨테이너는 무엇입니까?

- a. 가상 시스템.
- b. 제한된 리소스 액세스가 있는 격리된 프로세스.
- c. UnionFS에 의해 노출되는 일련의 파일 시스템 계층.
- d. 컨테이너 이미지를 제공하는 외부 서비스.

▶ 솔루션

컨테이너 아키텍처 개요

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너를 실행하는 데 사용되는 세 가지 Linux 기능은 무엇입니까? (세 개를 선택하십시오.)

- a. 네임스페이스
- b. 무결성 관리
- c. Security-Enhanced Linux
- d. 컨트롤 그룹

▶ 2. 다음 중 컨테이너 이미지를 가장 잘 설명하는 것은 무엇입니까?

- a. 컨테이너가 생성될 가상 시스템 이미지
- b. 컨테이너가 생성될 컨테이너 청사진
- c. 애플리케이션이 실행될 런타임 환경
- d. 레지스트리에서 사용하는 컨테이너의 색인 파일

▶ 3. 다음 중 컨테이너 아키텍처 구현 간에 공통적인 세 가지 구성 요소는 무엇입니까? (세 개를 선택하십시오.)

- a. 컨테이너 런타임
- b. 컨테이너 권한
- c. 컨테이너 이미지
- d. 컨테이너 레지스트리

▶ 4. Linux 커널과 관련된 컨테이너는 무엇입니까?

- a. 가상 시스템.
- b. 제한된 리소스 액세스가 있는 격리된 프로세스.
- c. UnionFS에 의해 노출되는 일련의 파일 시스템 계층.
- d. 컨테이너 이미지를 제공하는 외부 서비스.

Kubernetes 및 OpenShift 개요

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- Linux 컨테이너의 제한 사항과 컨테이너 오케스트레이션의 필요성을 확인합니다.
- Kubernetes 컨테이너 오케스트레이션 도구를 설명합니다.
- Red Hat OpenShift Container Platform(RHOCP)을 설명합니다.

컨테이너의 제한 사항

컨테이너는 서비스를 패키징하고 실행하는 쉬운 방법을 제공합니다. 조직이 관리하는 컨테이너 수가 증가하면 외부 요구에 신속하게 응답해야 하는 필요성과 함께 컨테이너를 수동으로 시작하는 작업도 기하급수적으로 증가합니다.

프로덕션 환경에서 컨테이너를 사용할 때 회사에는 대체로 다음이 필요합니다.

- 다수의 서비스 간에 편리한 통신
- 애플리케이션을 실행하는 컨테이너 수와 관계없이 애플리케이션에 대한 리소스 제한
- 애플리케이션 사용량 급증에 응답하여 실행 중인 컨테이너 증가 또는 감소
- 서비스 저하에 대응
- 일련의 사용자에게 새 릴리스를 점차 출시

컨테이너 런타임(예: Podman)에서 위의 요구 사항을 적절하게 처리하지 못해 회사에 컨테이너 오케스트레이션 기술이 필요한 경우가 많습니다.

Kubernetes 개요

Kubernetes는 컨테이너화된 애플리케이션의 배포, 관리 및 스케일링을 간소화하는 오케스트레이션 서비스입니다.

Kubernetes에서 관리할 수 있는 가장 작은 단위는 포드입니다. 포드는 단일 애플리케이션을 나타내는 스토리지 리소스 및 IP 주소를 공유하는 하나 이상의 컨테이너로 구성됩니다. 또한 Kubernetes는 Pod를 사용하여 내부 컨테이너를 오케스트레이션하고 해당 리소스를 단일 단위로 제한합니다.

Kubernetes 기능

Kubernetes는 컨테이너 인프라를 토대로 다음과 같은 기능을 제공합니다.

서비스 검색 및 부하 분산

Kubernetes는 각 컨테이너 세트에 단일 DNS 항목을 할당하여 서비스 간에 통신할 수 있도록 합니다. 이렇게 하면 요청하는 서비스가 대상의 DNS 이름만 알면 되기 때문에 클러스터가 서비스에 영향을 주지 않고 컨테이너의 위치와 IP 주소를 변경할 수 있습니다. 이 경우 서비스를 제공하는 컨테이너 풀 간에 요청을 부하 분산할 수 있습니다. 예를 들어 Kubernetes는 Pod의 가용성을 고려하여 들어오는 요청을 MySQL 서비스에 균등하게 분할할 수 있습니다.

수평 스케일링

Kubernetes 명령줄 인터페이스 또는 웹 UI로 설정한 구성을 사용하여 자동으로 또는 수동으로 애플리케이션을 확장하고 축소할 수 있습니다.

자체 복구

Kubernetes는 사용자 정의 상태 점검을 통해 컨테이너를 모니터링하여, 실패할 경우 다시 시작하고 스케줄을 조정할 수 있습니다.

자동 출시

Kubernetes는 상태를 점검하면서 애플리케이션 컨테이너에 업데이트를 점차 출시할 수 있습니다. 출시 중에 문제가 발생하면 Kubernetes를 배포의 이전 반복으로 롤백할 수 있습니다.

비밀 및 구성 관리

컨테이너를 다시 빌드하지 않고도 애플리케이션의 구성 설정과 비밀을 관리할 수 있습니다. 애플리케이션 비밀은 사용자 이름, 암호, 서비스 엔드포인트 등 비공개로 유지해야 하는 모든 구성 설정일 수 있습니다.

운영자

운영자는 애플리케이션 라이프사이클에 대한 지식도 Kubernetes 클러스터로 가져오는 패키징된 Kubernetes 애플리케이션입니다. 운영자로 패키징된 애플리케이션은 Kubernetes API를 사용하여 애플리케이션 상태 변경에 따라 클러스터 상태를 업데이트합니다.

OpenShift 개요

RHOCP(Red Hat OpenShift Container Platform)는 Kubernetes 컨테이너 인프라를 토대로 빌드된 모듈식 구성 요소 및 서비스 세트입니다. RHOCP에서는 개발자를 위해 원격 관리, 멀티 테넌시, 보안 향상, 모니터링 및 감사, 애플리케이션 라이프사이클 관리, 셀프 서비스 인터페이스 등 프로덕션 PaaS 플랫폼을 제공하는 기능을 추가합니다.

Red Hat OpenShift v4부터 OpenShift 클러스터의 호스트는 모두 Red Hat Enterprise Linux CoreOS를 기본 운영 체제로 사용합니다.

이 교육 과정 전반에서 RHOCP 및 OpenShift 용어는 Red Hat OpenShift Container Platform을 참조하는데 사용됩니다.

OpenShift 기능

OpenShift는 Kubernetes 클러스터에 다음과 같은 기능을 추가합니다.

통합된 개발자 워크플로

RHOCP는 기본 제공 컨테이너 레지스트리, CI/CD 파이프라인 및 소스 리포지토리에서 컨테이너 이미지로 아티팩트를 빌드하는 도구인 S2I를 통합합니다.

경로

서비스를 외부 세계에 쉽게 공개합니다.

메트릭 및 로깅

기본 제공 및 자체 분석 메트릭 서비스와 집계된 로깅을 포함합니다.

통합 UI

OpenShift는 다른 모든 기능을 관리할 수 있는 통합 도구와 UI를 제공합니다.



참조

프로덕션 등급 컨테이너 오케스트레이션 - **Kubernetes**

<https://kubernetes.io/>

OpenShift: Docker 및 Kubernetes에 빌드된 Red Hat의 컨테이너 애플리케이션 플랫폼

<https://www.openshift.com/>

▶ 퀴즈

Kubernetes 및 OpenShift 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너 제한 사항과 관련해서 세 가지 올바른 설명은 무엇입니까? (세 개를 선택하십시오.)

- a. 다수의 컨테이너가 쉽게 오케스트레이션됩니다.
- b. 자동화가 없으면 문제에 대한 대응 시간이 늘어납니다.
- c. 컨테이너는 내부 애플리케이션 오류를 관리하지 않습니다.
- d. 컨테이너는 부하가 분산되지 않습니다.
- e. 컨테이너는 극도로 격리된 패키지 애플리케이션입니다.

▶ 2. 다음 중 Kubernetes와 관련해서 두 가지 올바른 설명은 무엇입니까? (두 개를 선택하십시오.)

- a. Kubernetes는 컨테이너입니다.
- b. Kubernetes는 Docker 컨테이너만 사용할 수 있습니다.
- c. Kubernetes는 컨테이너 오케스트레이션 시스템입니다.
- d. Kubernetes는 컨테이너화된 애플리케이션의 배포, 관리 및 스케일링을 간소화합니다.
- e. Kubernetes 클러스터에서 관리되는 애플리케이션은 유지 관리하기가 더 어렵습니다.

▶ 3. 다음 중 Red Hat OpenShift v4와 관련해서 세 가지 올바른 설명은 무엇입니까? (세 개를 선택하십시오.)

- a. OpenShift는 Kubernetes 인프라에 추가 기능을 제공합니다.
- b. Kubernetes와 OpenShift는 함께 사용할 수 없습니다.
- c. OpenShift 호스트는 Red Hat Enterprise Linux를 기본 운영 체제로 사용합니다.
- d. OpenShift는 Source-to-Image 기술과 CI/CD 파이프라인을 통합하여 개발을 간소화합니다.
- e. OpenShift는 라우팅 및 부하 분산을 간소화합니다.

▶ 4. Kubernetes 기능을 확장하고 OpenShift가 제공하는 기능은 무엇입니까? (두 개를 선택하십시오.)

- a. 운영자 및 운영자 프레임워크.
- b. 서비스를 외부 세계에 공개하는 경로.
- c. 통합된 개발 워크플로.
- d. 자동 복구 및 상태 점검.

▶ 솔루션

Kubernetes 및 OpenShift 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너 제한 사항과 관련해서 세 가지 올바른 설명은 무엇입니까? (세 개를 선택하십시오.)

- a. 다수의 컨테이너가 쉽게 오케스트레이션됩니다.
- b. 자동화가 없으면 문제에 대한 대응 시간이 늘어납니다.
- c. 컨테이너는 내부 애플리케이션 오류를 관리하지 않습니다.
- d. 컨테이너는 부하가 분산되지 않습니다.
- e. 컨테이너는 극도로 격리된 패키지 애플리케이션입니다.

▶ 2. 다음 중 Kubernetes와 관련해서 두 가지 올바른 설명은 무엇입니까? (두 개를 선택하십시오.)

- a. Kubernetes는 컨테이너입니다.
- b. Kubernetes는 Docker 컨테이너만 사용할 수 있습니다.
- c. Kubernetes는 컨테이너 오케스트레이션 시스템입니다.
- d. Kubernetes는 컨테이너화된 애플리케이션의 배포, 관리 및 스케일링을 간소화합니다.
- e. Kubernetes 클러스터에서 관리되는 애플리케이션은 유지 관리하기가 더 어렵습니다.

▶ 3. 다음 중 Red Hat OpenShift v4와 관련해서 세 가지 올바른 설명은 무엇입니까? (세 개를 선택하십시오.)

- a. OpenShift는 Kubernetes 인프라에 추가 기능을 제공합니다.
- b. Kubernetes와 OpenShift는 함께 사용할 수 없습니다.
- c. OpenShift 호스트는 Red Hat Enterprise Linux를 기본 운영 체제로 사용합니다.
- d. OpenShift는 Source-to-Image 기술과 CI/CD 파이프라인을 통합하여 개발을 간소화합니다.
- e. OpenShift는 라우팅 및 부하 분산을 간소화합니다.

▶ 4. Kubernetes 기능을 확장하고 OpenShift가 제공하는 기능은 무엇입니까? (두 개를 선택하십시오.)

- a. 운영자 및 운영자 프레임워크.
- b. 서비스를 외부 세계에 공개하는 경로.
- c. 통합된 개발 워크플로.
- d. 자동 복구 및 상태 점검.

▶ 연습 가이드

강의실 환경 구성

이 연습에서는 교육 과정의 DO180 부분에 사용할 **workstation** 시스템을 구성합니다.

결과

다음을 수행할 수 있습니다.

- 교육 과정 전반에서 사용되는 OpenShift 클러스터, 컨테이너 이미지 레지스트리, Git 리포지토리에 액세스하도록 **workstation** 시스템을 구성합니다.
- Red Hat DO180 샘플 애플리케이션 리포지토리를 개인 GitHub 계정으로 분기합니다.
- 개인 GitHub 계정에 있는 DO180 샘플 애플리케이션 리포지토리를 **workstation** 시스템으로 복제합니다.

시작하기 전에

이 연습을 수행하려면 다음이 있어야 합니다.

- 개인적인 무료 GitHub 계정. GitHub에 등록해야 하는 경우 부록 A. GitHub 계정 만들기의 지침을 참조하십시오.
- 개인적인 무료 Quay.io 계정. Quay.io에 등록해야 하는 경우 부록 B. Quay 계정 만들기의 지침을 참조하십시오.

- ▶ 1. **workstation** 시스템에서 터미널을 열고 **lab-configure** 명령을 실행합니다. 대화형 프롬프트에 답변합니다. **Ctrl+C**를 사용하여 언제든지 명령을 중단하고 처음부터 다시 시작할 수 있습니다.

```
[student@workstation ~]$ lab-configure
```

lab-configure 명령에서 GitHub 및 Quay.io 계정 이름을 입력하라는 메시지를 표시합니다. 입력한 계정 이름 중 하나가 존재하지 않으면 명령이 종료되고 오류 메시지가 표시됩니다.

- . Enter the GitHub account name: **your_github_username**
Verifying GitHub account name: **your_github_username**
- . Enter the Quay.io account name: **your_quay_username**
Verifying Quay.io account name: **your_quay_username**

두 계정 이름이 모두 있는 경우 **lab-configure** 명령은 일부 연습에 필요한 변수를 사용하여 /usr/local/etc/ocp4.config 파일을 구성합니다.

- . Configuring RHT_OCP4_GITHUB_USER variable: SUCCESS
- . Configuring RHT_OCP4_QUAY_USER variable: SUCCESS

이전에 **lab-configure** 명령을 완료한 경우 명령을 다시 실행하면 GitHub 및 Quay.io 계정 이름이 표시됩니다. 또한 이 명령을 사용하면 **developer** 사용자가 OpenShift 클러스터에 로그인할 수

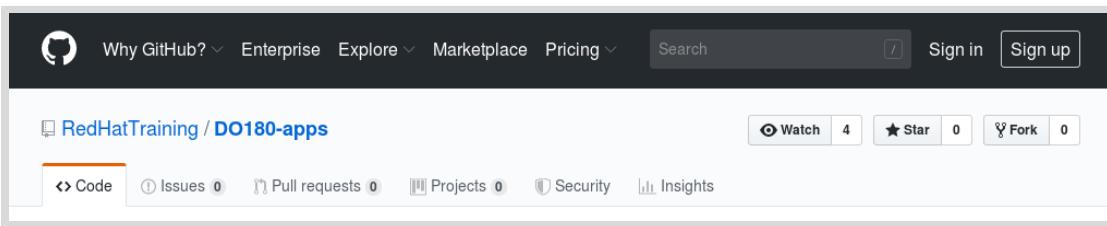
1장 | 컨테이너 기술 소개

있습니다. 필요한 경우 이 명령은 HTPasswd ID 프로바이더를 사용하도록 OpenShift 클러스터를 구성합니다. 이는 일부 DO280 연습을 완료한 후 DO180 연습으로 돌아가는 경우에 유용합니다.

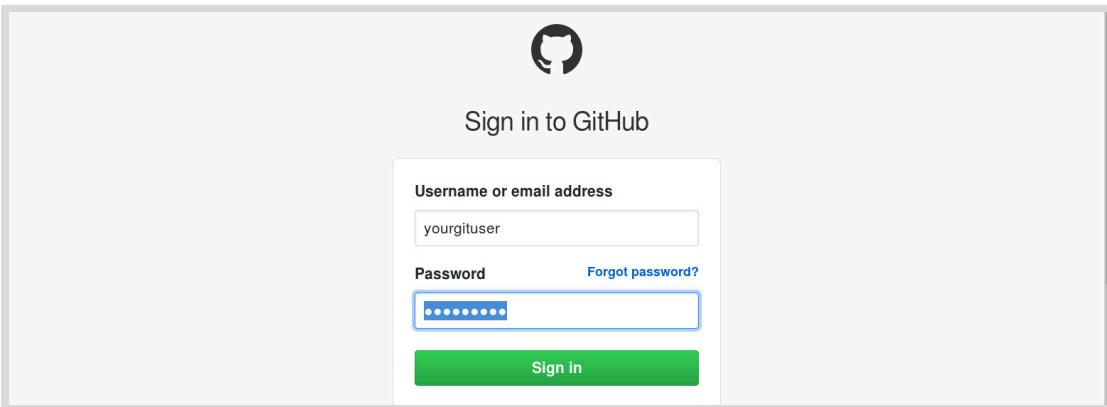
- . GitHub account name: your_github_username
- . Quay.io account name: your_quay_username
- . To reconfigure, run: lab-configure -d
- . Ensuring user 'developer' can log in to the OpenShift cluster.

- ▶ 2. 일부 DO180 연습에는 개인 GitHub 계정에 저장된 DO180 샘플 애플리케이션의 분기된 버전이 필요합니다. 다음 단계를 수행하십시오.

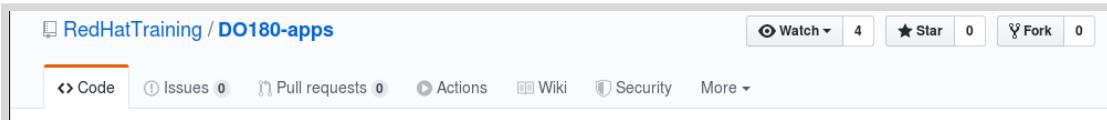
- 2.1. 웹 브라우저를 열고 <https://github.com/RedHatTraining/DO180-apps>로 이동합니다. GitHub에 로그인하지 않은 경우 오른쪽 상단에 있는 **Sign in(로그인)**을 클릭합니다.



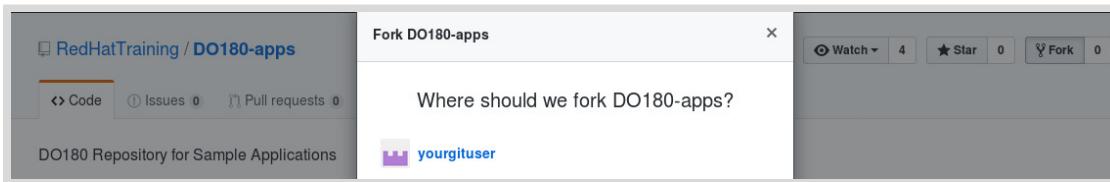
- 2.2. 개인 사용자 이름 및 암호를 사용하여 GitHub에 로그인합니다.



- 2.3. [RedHatTraining/DO180-apps](#) 리포지토리로 돌아가서 오른쪽 상단 모서리에 있는 **Fork(분기)**를 클릭합니다.

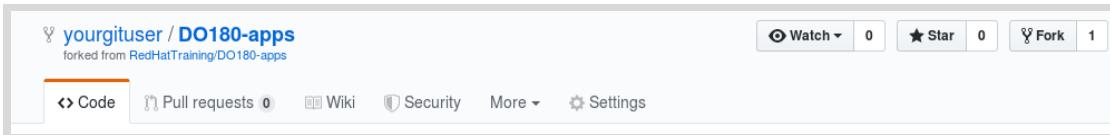


- 2.4. Fork DO180-apps(DO180-apps 분기) 창에서 **yourgituser**를 클릭하여 개인 GitHub 프로젝트를 선택합니다.

**중요**

<https://github.com/RedHatTraining/DO180-apps> 리포지토리의 개인 포크 이름을 바꿀 수는 있지만 본 교육 과정의 평가 스크립트, 도우미 스크립트 및 예제 출력에서는 리포지토리를 포크할 때 **DO180-apps**라는 이름을 유지한다고 가정합니다.

2.5. 몇 분 후 GitHub 웹 인터페이스에 새 리포지토리 **yourgituser/DO180-apps**가 표시됩니다.



- ▶ 3. 일부 DO180 연습에는 **/home/student/DO180-apps** 디렉터리에 복제된 개인 GitHub 계정의 DO180 샘플 애플리케이션 사본이 필요합니다. 다음 단계를 수행하십시오.

3.1. **/usr/local/etc/ocp4.config**에서 액세스할 수 있는 강의실 구성 파일을 찾습니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

3.2. 다음 명령을 실행하여 DO180 샘플 애플리케이션 리포지토리를 복제합니다.

```
[student@workstation ~]$ git clone \
> https://github.com/${RHT_OCP4_GITHUB_USER}/DO180-apps
Cloning into 'DO180-apps'...
...output omitted...
```

3.3. **/home/student/DO180-apps** 디렉터리가 Git 리포지토리인지 확인합니다.

```
[student@workstation ~]$ cd ~/DO180-apps
[student@workstation DO180-apps]$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

3.4. **/home/student/DO180-apps** 디렉터리에 DO180 샘플 애플리케이션이 포함되어 있는지 확인합니다.

```
[student@workstation DO180-apps]$ head README.md
# DO180-apps
DO180 Repository for Sample Applications
```

3.5. **/home/student** 디렉터리로 다시 변경합니다.

```
[student@workstation DO180-apps]$ cd  
[student@workstation ~]$
```

- ▶ 4. 이제 **workstation** 시스템에 **DO180-apps** 리포지토리의 로컬 복제본이 있고 **lab-configure** 명령을 성공적으로 실행했으므로 교육 과정의 DO180 부분에 있는 연습을 시작할 준비가 되었습니다.

이 교육 과정 중 소스에서 애플리케이션을 빌드하는 모든 연습은 **DO180-apps** Git 리포지토리의 **master** 분기에서 시작됩니다. 소스 코드를 변경하는 연습을 수행하려면 변경 내용을 호스팅하는 새 분기를 만들어 **master** 분기에 항상 알려진 양호한 시작점이 포함되도록 해야 합니다. 어떠한 이유로 연습을 일시 중지하거나 다시 시작해야 하고 Git 분기에 대한 변경 사항을 저장하거나 무시해야 하는 경우에는 부록 C. 유용한 Git 명령을 참조하십시오.

이로써 안내에 따른 연습이 완료됩니다.

요약

이 장에서 학습한 내용:

- 컨테이너는 매우 적은 오버헤드로 생성된 격리된 애플리케이션 런타임입니다.
- 컨테이너 이미지는 모든 종속성과 함께 애플리케이션을 패키징하여 여러 다른 환경에서 애플리케이션을 더 쉽게 실행할 수 있게 합니다.
- Podman 등의 애플리케이션은 표준 Linux 커널의 기능을 사용하여 컨테이너를 생성합니다.
- 컨테이너 이미지 레지스트리는 컨테이너 이미지를 여러 사용자와 호스트에 배포하는 데 선호되는 메커니즘입니다.
- OpenShift는 Kubernetes를 사용하여 다중 컨테이너로 구성된 애플리케이션을 조정합니다.
- Kubernetes는 컨테이너화된 애플리케이션에 대한 부하 분산, 고가용성 및 영구저장장치를 관리합니다.
- OpenShift는 Kubernetes에 멀티 테넌시, 보안, 사용 편의성 및 지속적 통합 및 지속적 개발 기능을 추가합니다.
- OpenShift 경로를 사용하면 관리 가능한 방식으로 컨테이너화된 애플리케이션을 외부에서 액세스할 수 있습니다.

2장

컨테이너화된 서비스 생성

목적

컨테이너 기술을 사용하여 서비스를 프로비저닝합니다.

목표

- 컨테이너 이미지에서 데이터베이스 서버를 생성합니다.

섹션

- 컨테이너화된 서비스 프로비저닝(안내에 따른 연습)

컨테이너화된 서비스 프로비저닝

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- Podman을 사용하여 컨테이너 이미지를 검색하고 가져옵니다.
- 컨테이너를 로컬에서 실행하고 구성합니다.
- Red Hat Container Catalog을 사용합니다.

Podman을 사용하여 컨테이너 이미지 가져오기

격리되고 제어된 실행 환경에서 제공하는 방법으로 컨테이너 내부에서 애플리케이션을 실행할 수 있습니다. 컨테이너화된 애플리케이션을 실행하려면(즉, 컨테이너 내부에서 애플리케이션을 실행하려면) 모든 애플리케이션 파일, 라이브러리 및 애플리케이션에서 실행할 종속성을 제공하는 파일 시스템 번들 컨테이너 이미지가 필요합니다. 컨테이너 이미지는 이미지 레지스트리에서 찾을 수 있습니다. 이미지 레지스트리는 사용자가 컨테이너 이미지를 검색하고 가져올 수 있는 서비스입니다. Podman 사용자는 **search** 하위 명령을 사용하여 원격 또는 로컬 레지스트리에서 사용 가능한 이미지를 찾을 수 있습니다.

```
[student@workstation ~]$ sudo podman search rhel
INDEX      NAME          DESCRIPTION   STARS OFFICIAL AUTOMATED
redhat.com registry.access.redhat.com/rhel This plat... 0
...output omitted...
```

이미지를 찾으면 Podman을 사용하여 이미지를 다운로드할 수 있습니다. **pull** 하위 명령을 사용하는 경우 Podman은 이미지를 가져와서 나중에 사용할 수 있도록 로컬에 저장합니다.

```
[student@workstation ~]$ sudo podman pull rhel
Trying to pull registry.access.redhat.com/rhel...Getting image source signatures
Copying blob sha256: ...output omitted...
  72.25 MB / 72.25 MB [=====] 8s
Copying blob sha256: ...output omitted...
  1.20 KB / 1.20 KB [=====] 0s
Copying config sha256: ...output omitted...
  6.30 KB / 6.30 KB [=====] 0s
Writing manifest to image destination
Storing signatures
699d44bc6ea2b9fb23e7899bd4023d3c83894d3be64b12e65a3fe63e2c70f0ef
```

컨테이너 이미지의 이름은 다음 구문에 따라 지정됩니다.

`registry_name/user_name/image_name:tag`

- 먼저 **registry_name**은 이미지를 저장하는 레지스트리의 이름입니다. 일반적으로 레지스트리의 FQDN입니다.
- **user_name** 이미지가 속한 사용자 또는 조직을 나타냅니다.
- **image_name**은 사용자 네임스페이스에서 고유해야 합니다.

- **tag**는 이미지 버전을 식별합니다. 이미지 이름에 이미지 태그가 없으면 **latest**로 가정됩니다.

**참고**

이 강의실의 Podman 설치에서는 공개적으로 사용 가능한 여러 레지스트리(예: [Quay.io](#) 및 Red Hat Container Catalog)를 사용합니다.

Podman은 이미지를 가져온 후 로컬에 저장하고 **images** 하위 명령을 사용하여 나열할 수 있습니다.

```
[student@workstation ~]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
registry.access.redhat.com/rhel   latest   699d44bc6ea2  4 days ago   214MB
...output omitted...
```

컨테이너 실행

podman run 명령은 이미지를 기반으로 하여 컨테이너를 로컬에서 실행합니다. 최소한, 이 명령에는 컨테이너에서 실행할 이미지의 이름이 필요합니다.

컨테이너 이미지는 컨테이너 내부에서 시작되는, 진입점이라는 프로세스를 지정합니다. **podman run** 명령은 이미지 이름 뒤의 모든 매개 변수를 컨테이너의 진입점 명령으로 사용합니다. 다음 예제에서는 Red Hat Enterprise Linux 이미지에서 컨테이너를 시작합니다. 이 컨테이너의 진입점을 **echo "Hello world"** 명령으로 설정합니다.

```
[student@workstation ~]$ sudo podman run ubi7/ubi:7.7 echo 'Hello!'
Hello world
```

컨테이너 이미지를 백그라운드 프로세스로 시작하려면 **-d** 옵션을 **podman run** 명령에 전달합니다.

```
[student@workstation ~]$ sudo podman run -d rhsc1/httpd-24-rhel7:2.4-36.8
ff4ec6d74e9b2a7b55c49f138e56f8bc46fe2a09c23093664fea7febc3dfa1b2
[student@workstation ~]$ sudo podman inspect -l \
> -f "{{.NetworkSettings.IPAddress}}"
10.88.0.68
[student@workstation ~]$ curl http://10.88.0.68:8080
...output omitted...
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...output omitted...
<title>
Test Page for the Apache HTTP Server on Red Hat Enterprise Linux
</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style type="text/css">
...output omitted...
```

이전 예제에서는 컨테이너화된 Apache HTTP 서버를 백그라운드에서 실행했습니다. 그런 다음, 예제에서는 **podman inspect** 명령을 사용하여 컨테이너 메타데이터에서 컨테이너의 내부 IP 주소를 검색합니다. 마지막으로, IP 주소를 사용하여 Apache HTTP 서버에서 루트 페이지를 가져옵니다. 이 응답은 **podman run** 명령 이후 컨테이너가 시작되어 여전히 실행 중임을 증명합니다.

**참고**

대부분의 Podman 하위 명령은 **-1** 플래그(가장 최근 항목의 경우 **1**)를 컨테이너 ID 대신 허용합니다. 이 플래그는 모든 Podman 명령에서 가장 최근에 사용한 컨테이너에 명령을 적용합니다.

**참고**

podman run 명령을 사용할 때 실행할 이미지를 로컬에서 사용할 수 없는 경우 Podman은 자동으로 **pull**을 사용하여 이미지를 다운로드합니다.

컨테이너를 참조하는 경우 Podman은 컨테이너 이름 또는 생성된 컨테이너 ID로 컨테이너를 인식합니다. Podman으로 컨테이너를 실행할 때 **--name** 옵션을 사용하여 컨테이너 이름을 설정할 수 있습니다. 컨테이너 이름은 고유해야 합니다. **podman run** 명령에 컨테이너 이름이 없으면 Podman은 고유한 무작위 이름을 생성합니다.

이미지가 콘솔 입력으로 사용자와 상호 작용해야 하는 경우 Podman은 컨테이너 입력 및 출력 스트림을 콘솔로 리디렉션할 수 있습니다. 상호 작용이 활성화되려면 **run** 하위 명령에 **-t** 및 **-i** 플래그(또는 간단하게 **-it** 플래그)가 필요합니다.

**참고**

많은 Podman 플래그에는 긴 대체 형식도 있습니다. 아래에 몇 가지 플래그가 설명되어 있습니다.

- **-t**는 **--tty**와 같으며, **pseudo-tty**(pseudo-terminal)가 컨테이너에 대해 할당되어야 함을 의미합니다.
- **-i**는 **--interactive**와 같습니다. 이 플래그를 사용하면 표준 입력이 컨테이너에 열린 채로 유지됩니다.
- **-d** 또는 긴 형식인 **--detach**는 컨테이너가 백그라운드에서 실행됨을 의미합니다(분리 됨). 그런 다음, Podman은 컨테이너 ID를 인쇄합니다.

전체 플래그 목록은 Podman 문서를 참조하십시오.

다음 예제에서는 컨테이너 내부에서 Bash 터미널을 시작하고, 터미널에서 대화형으로 명령을 실행합니다.

```
[student@workstation ~]$ sudo podman run -it ubi7/ubi:7.7 /bin/bash
bash-4.2# ls
...output omitted...
bash-4.2# whoami
root
bash-4.2# exit
exit
[student@workstation ~]$
```

일부 컨테이너는 시작할 때 제공되는 외부 매개 변수가 필요하거나 사용할 수 있습니다. 이러한 매개 변수를 제공하고 사용하는 가장 일반적인 방법은 환경 변수를 이용하는 것입니다. Podman은 **run** 하위 명령에 **-e** 플래그를 추가하여 시작할 때 컨테이너에 환경 변수를 삽입할 수 있습니다.

```
[student@workstation ~]$ sudo podman run -e GREET=Hello -e NAME=RedHat \
> > rhel7:7.5 printenv GREET NAME
Hello
RedHat
[student@workstation ~]$
```

앞의 예제에서는 매개 변수로 제공된 환경 변수 두 개를 인쇄하는 RHEL 이미지 컨테이너를 시작합니다. 환경 변수에 대한 또 다른 사용 사례는 MySQL 데이터베이스 서버에 자격 증명을 설정하는 것입니다.

```
[root@workstation ~]# sudo podman run --name mysql-custom \
> -e MYSQL_USER=redhat -e MYSQL_PASSWORD=r3dh4t \
> -d rhmap47/mysql:5.5
```

Red Hat Container Catalog 사용

Red Hat은 미세 조정된 컨테이너 이미지의 리포지토리를 유지 관리합니다. 이 리포지토리를 사용하면 테스트되지 않은 이미지로 인해 유발될 수 있는 알려진 취약성에 대한 보호 및 안정성 계층이 고객에게 제공됩니다. 표준 **podman** 명령은 Red Hat Container Catalog와 호환됩니다. Red Hat Container Catalog는 Red Hat 리포지토리에서 컨테이너 이미지를 검색하고 살펴보기 위한, 사용자에게 친숙한 인터페이스를 제공합니다.

Container Catalog는 단일 인터페이스 역할도 하며, 리포지토리에서 사용할 수 있는 모든 컨테이너 이미지의 다양한 측면에 액세스할 수 있습니다. 상태 색인 등급을 기준으로 여러 버전의 컨테이너 이미지 중에서 최적의 이미지를 결정하는 데 유용합니다. 상태 색인 평가는 이미지가 얼마나 최신인지 및 최신 보안 업데이트를 포함하는지를 나타냅니다.

Container Catalog는 이미지의 에라타(Errata) 문서에 대한 액세스도 제공합니다. 각 업데이트에서 최신 버그 수정 및 개선 사항을 설명합니다. 또한 각 운영 체제에서 이미지를 가져오는데 최고인 기술을 제안합니다.

다음 이미지는 Red Hat Container Catalog의 일부 기능을 강조 표시합니다.

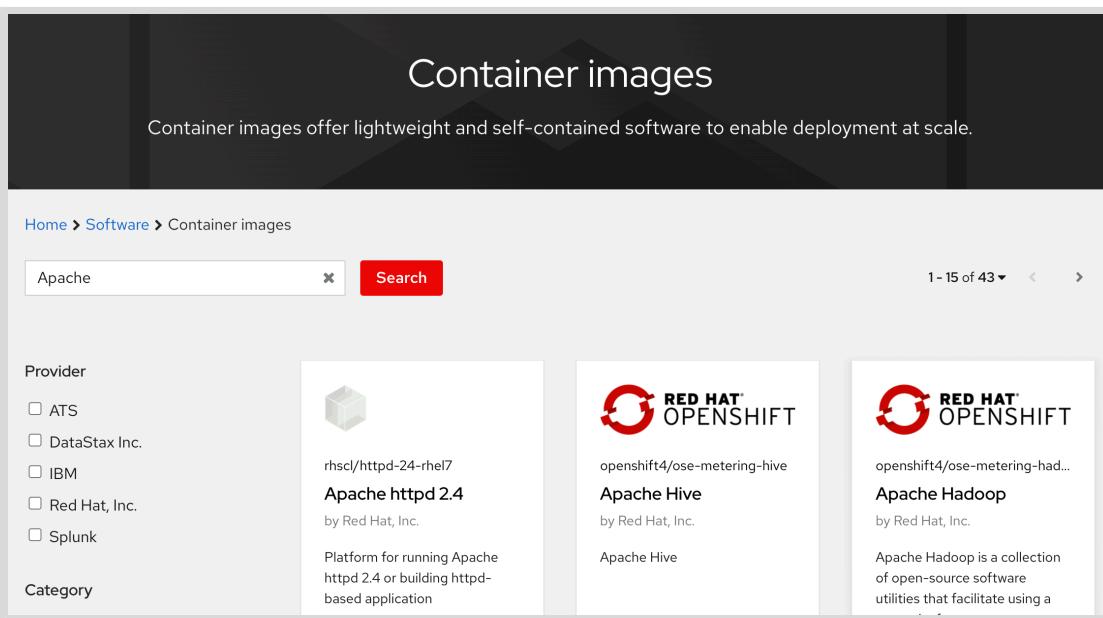


그림 2.1: Red Hat Container Catalog 검색 페이지

위에 표시된 대로 Container Catalog 검색 상자에 **Apache**를 검색하면 검색 패턴과 일치하는 제품 및 이미지 리포지토리를 제안하는 목록이 표시됩니다. **Apache httpd 2.4** 이미지 페이지에 액세스하려면 제안된 목록에서 **rhscl/httpd-24-rhel7**을 선택합니다.

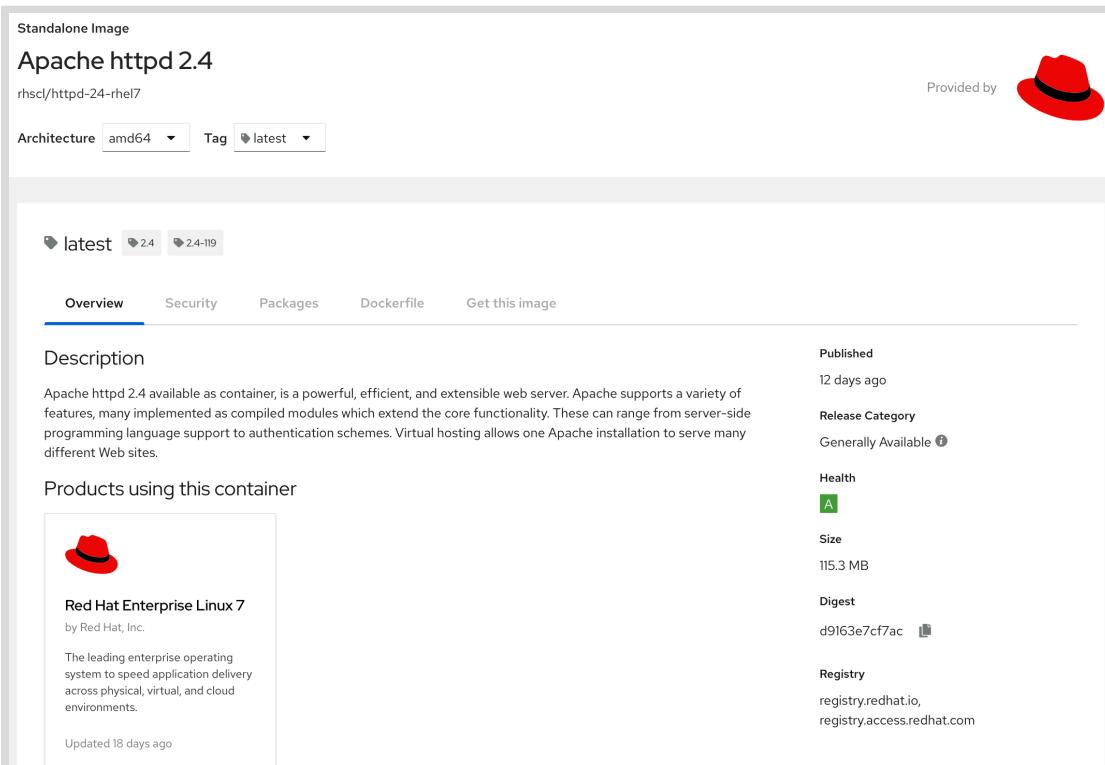


그림 2.2: Apache httpd 2.4(rhscl/httpd-24-rhel7) 개요 이미지 페이지

Apache httpd 2.4 패널에는 이미지 세부 정보와 여러 개의 탭이 표시됩니다. 이 페이지에는 Red Hat이 이미지 리포지토리를 유지 관리한다고 설명되어 있습니다. Overview(개요) 탭에는 다른 세부 정보가 있습니다.

- Description(설명): 이미지의 기능에 대한 요약
- 이 컨테이너를 사용하는 제품: Red Hat Enterprise Linux가 이 이미지 리포지토리를 사용함을 나타냅니다.
- Most Recent Tag(최신 태그): 이미지가 최신 업데이트, 이미지에 적용된 최신 태그, 이미지 상태 등을 받은 시점

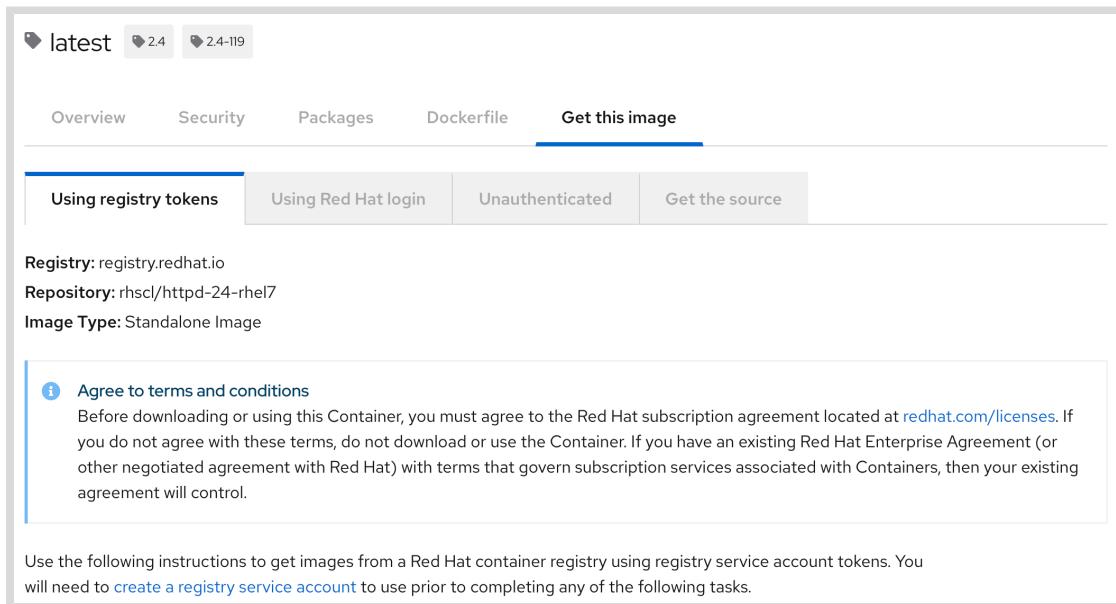


그림 2.3: Apache httpd 2.4(rhscl/httpd-24-rhel7) 최신 이미지 페이지

Get this image(이 이미지 받기) 탭에서는 최신 버전의 이미지를 받을 수 있는 절차를 안내합니다. 페이지에는 이미지를 검색하는 데 사용되는 다양한 옵션이 제공되어 있습니다. 탭에서 선호하는 절차를 선택하면 페이지에서 해당 이미지를 검색하는 데 적합한 지침을 제공합니다.



참조

Red Hat Container Catalog

<https://registry.redhat.io>

Quay.io 웹사이트

<https://quay.io>

▶ 연습 가이드

MySQL 데이터베이스 인스턴스 생성

이 연습에서는 컨테이너 내부에서 MySQL 데이터베이스를 시작한 다음 데이터베이스를 생성하고 채웁니다.

결과

컨테이너 이미지에서 데이터베이스를 시작하고, 데이터베이스 내부에 정보를 저장할 수 있습니다.

시작하기 전에

workstation에서 student 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab container-create start
```

▶ 1. MySQL 컨테이너 인스턴스를 생성합니다.

1.1. Red Hat Software Collections Library MySQL 이미지에서 컨테이너를 시작합니다.

```
[student@workstation ~]$ sudo podman run --name mysql-basic \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> -d rh scl/mysql-57-rhel7:5.7-3.14
Trying to pull ...output omitted...
Copying blob sha256:e373541...output omitted...
69.66 MB / 69.66 MB [=====] 8s
Copying blob sha256:c5d2e94...output omitted...
1.20 KB / 1.20 KB [=====] 0s
Copying blob sha256:b3949ae...output omitted...
62.03 MB / 62.03 MB [=====] 8s
Writing manifest to image destination
Storing signatures
92eaa6b67da0475745b2beffa7e0895391ab34ab3bf1ded99363bb09279a24a0
```

이 명령은 **5.7-3.14** 태그가 지정된 MySQL 컨테이너 이미지를 다운로드한 다음, 컨테이너 기반 이미지를 시작합니다. 그러면 사용자 **user1**이 소유하고 암호가 **mypa55**인 **items**라는 데이터베이스가 생성됩니다. 데이터베이스 관리자 암호는 **r00tpa55**로 설정되어 컨테이너는 백그라운드에서 실행됩니다.

1.2. 컨테이너가 오류 없이 시작되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman ps --format "{{.ID}} {{.Image}} {{.Names}}"
92eaa6b67da0 registry.access.redhat.com/rh scl/mysql-57-rhel7:5.7-3.14 mysql-basic
```

▶ 2. 다음 명령을 실행하여 컨테이너 샌드박스에 액세스합니다.

```
[student@workstation ~]$ sudo podman exec -it mysql-basic /bin/bash
bash-4.2$
```

이 명령은 MySQL 컨테이너 내부에서 mysql 사용자로 실행되는 Bash 쉘을 시작합니다.

▶ 3. 데이터베이스에 데이터를 추가합니다.

- 3.1. 데이터베이스 관리자(root)로 MySQL에 연결합니다.

컨테이너 터미널에서 다음 명령을 실행하여 데이터베이스에 연결합니다.

```
bash-4.2$ mysql -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
...output omitted...
mysql>
```

mysql 명령으로 MySQL 데이터베이스 대화식 프롬프트를 엽니다. 다음 명령을 실행하여 데이터베이스 가용성을 확인합니다.

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.01 sec)
```

- 3.2. items 데이터베이스에 새 테이블을 생성합니다. 다음 명령을 실행하여 데이터베이스에 액세스합니다.

```
mysql> use items;
Database changed
```

- 3.3. items 데이터베이스에 Projects라는 테이블을 생성합니다.

```
mysql> CREATE TABLE Projects (id int(11) NOT NULL,
   -> name varchar(255) DEFAULT NULL,
   -> code varchar(255) DEFAULT NULL,
   -> PRIMARY KEY (id));
Query OK, 0 rows affected (0.01 sec)
```

필요에 따라 ~/D0180/solutions/container-create/create_table.txt 파일을 사용하여 위에 제공된 CREATE TABLE MySQL 문을 복사하여 붙여넣을 수 있습니다.

- 3.4. show tables 명령을 사용하여 테이블이 생성되었는지 확인합니다.

```
mysql> show tables;
+-----+
| Tables_in_items |
+-----+
| Projects        |
+-----+
1 row in set (0.00 sec)
```

3.5. `insert` 명령을 사용하여 행을 테이블에 삽입합니다.

```
mysql> insert into Projects (id, name, code) values (1, 'DevOps', 'DO180');
Query OK, 1 row affected (0.02 sec)
```

3.6. `select` 명령을 사용하여 프로젝트 정보가 테이블에 추가되었는지 확인합니다.

```
mysql> select * from Projects;
+---+-----+---+
| id | name      | code   |
+---+-----+---+
| 1  | DevOps    | DO180  |
+---+-----+---+
1 row in set (0.00 sec)
```

3.7. MySQL 프롬프트 및 MySQL 컨테이너를 종료합니다.

```
mysql> exit
Bye
bash-4.2$ exit
exit
```

완료

workstation에서 `lab container-create finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab container-create finish
```

이로써 연습이 완료됩니다.

요약

이 장에서 학습한 내용:

- Podman을 사용하면 사용자가 로컬 또는 원격 레지스트리에서 이미지를 검색하고 다운로드할 수 있습니다.
- **podman run** 명령은 컨테이너 이미지에서 컨테이너를 생성하고 시작합니다.
- 컨테이너는 **-d** 플래그를 사용하거나 대화형으로 **-it** 플래그를 사용하여 백그라운드에서 실행됩니다.
- 일부 컨테이너 이미지에는 **podman run** 명령의 **-e** 옵션을 사용하여 설정하는 환경 변수가 필요합니다.
- Red Hat Container Catalog는 Red Hat의 공식 컨테이너 이미지 리포지토리에서 컨테이너 이미지 검색, 탐색 및 분석을 지원합니다.

3장

컨테이너 관리

목적

사전 빌드된 컨테이너 이미지를 수정하여 컨테이너화된 서비스를 생성하고 관리합니다.

목표

- 생성에서 삭제에 이르기까지 컨테이너의 라이프사이클을 관리합니다.
- 영구저장장치를 사용하여 컨테이너 애플리케이션 데이터를 저장합니다.
- 포트 전달을 사용하여 컨테이너에 액세스하는 방법을 설명합니다.

섹션

- 컨테이너 라이프사이클 관리(안내에 따른 연습)
- 컨테이너에 영구저장장치 연결(안내에 따른 연습)
- 컨테이너 액세스(안내에 따른 연습)

랩

컨테이너 관리

컨테이너 라이프사이클 관리

목표

이 섹션을 마치면 생성에서 종료까지 컨테이너의 라이프사이클을 관리할 수 있습니다.

Podman을 사용한 컨테이너 라이프사이클 관리

이전 장에서는 Podman을 사용하여 컨테이너 서비스를 생성하는 방법을 알아보았습니다. 이제 컨테이너의 라이프사이클을 관리하는 데 사용할 수 있는 명령과 전략을 자세히 살펴보겠습니다. Podman을 사용하면 컨테이너를 실행할 수 있을 뿐 아니라 컨테이너가 백그라운드에서 실행되도록 하고, 컨테이너 내부에서 새 프로세스를 실행하고, 파일 시스템 볼륨이나 네트워크와 같은 리소스를 제공할 수도 있습니다.

podman 명령으로 구현하는 Podman은 컨테이너를 생성 및 관리하는 일련의 하위 명령을 제공합니다. 개발자는 이러한 하위 명령을 사용하여 컨테이너 및 컨테이너 이미지 라이프사이클을 관리합니다. 다음 그림은 컨테이너 및 이미지 상태를 변경하는데 가장 많이 사용되는 하위 명령을 요약해서 보여줍니다.

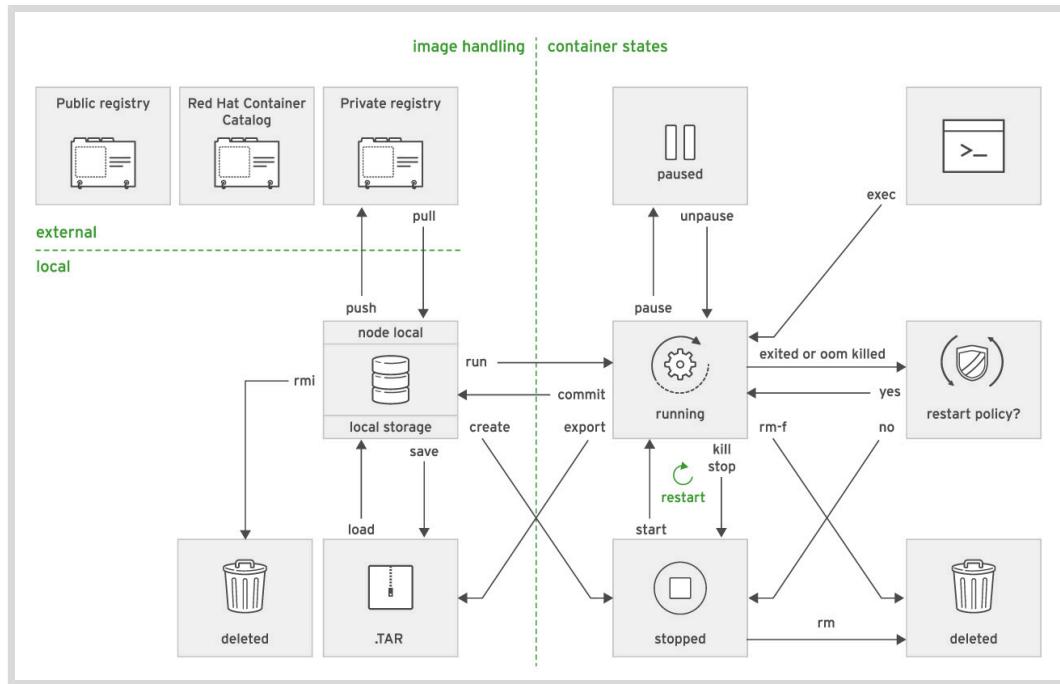


그림 3.1: Podman 관리 하위 명령

Podman은 실행 중인 컨테이너와 중지된 컨테이너에 대한 정보를 가져오는 일련의 유용한 하위 명령도 제공합니다. 이 하위 명령을 사용하여 디버깅, 업데이트 또는 보고 목적으로 컨테이너 및 이미지에서 정보를 추출 할 수 있습니다. 다음 그림은 컨테이너 및 이미지에서 정보를 쿼리하는 데 가장 많이 사용되는 하위 명령을 요약해서 보여줍니다.

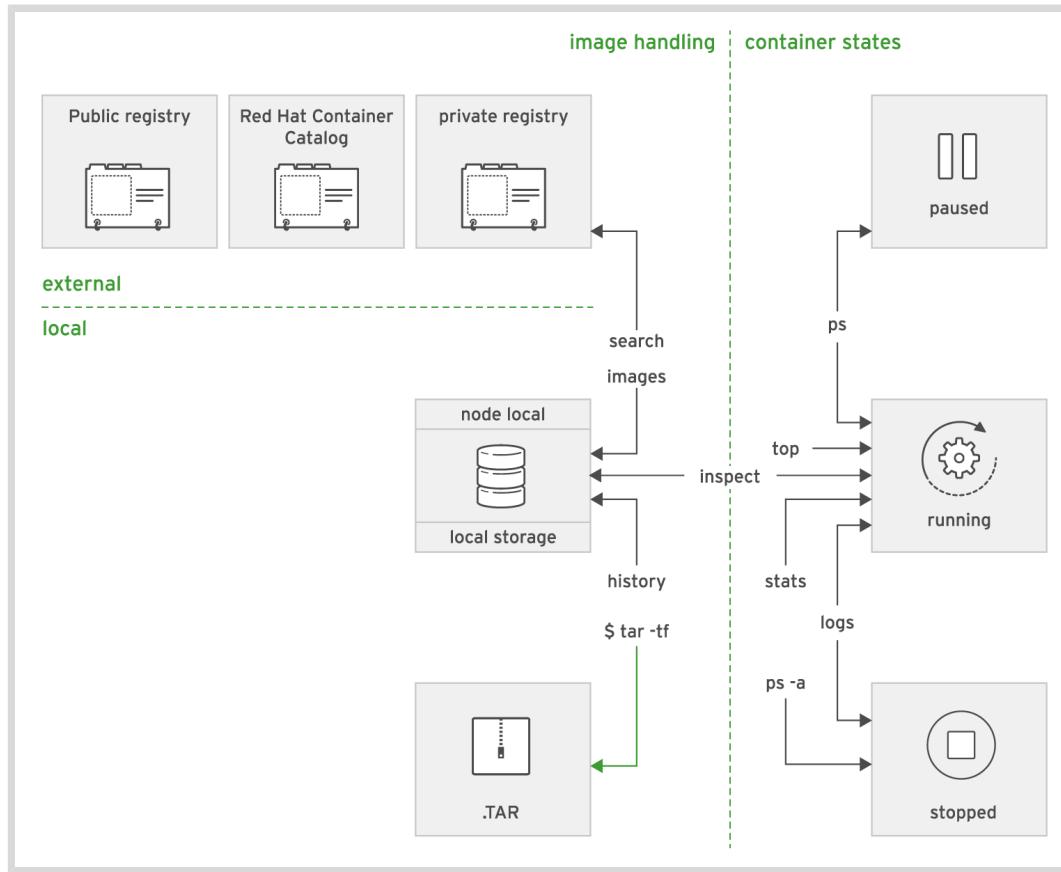


그림 3.2: Podman 쿼리 하위 명령

이 교육 과정에서 Podman 하위 명령을 알아보는 동안 위 두 그림을 참조로 사용하십시오.

컨테이너 생성

`podman run` 명령은 이미지에서 새 컨테이너를 만들고 새 컨테이너 내부에서 프로세스를 시작합니다. 컨테이너 이미지를 로컬에서 사용할 수 없는 경우 이 명령은 구성된 이미지 리포지토리를 사용하여 이미지를 다운로드하려고 시도합니다.

```
[student@workstation ~]$ sudo podman run rhsc1/httpd-24-rhel7
Trying to pull registry...httpd-24-rhel7:latest...Getting image source signatures
Copying blob sha256:23113...b0be82
72.21 MB / 72.21 MB [=====] 7s
...output omitted...AH00094: Command line: 'httpd -D FOREGROUND'
^C
```

이전 출력 샘플에서는 컨테이너가 대화형이 아닌 프로세스로 시작되었으며(`-it` 옵션 제외), `-d` 옵션으로 시작되지 않았기 때문에 포그라운드에서 실행 중입니다. 따라서 **Ctrl+C(SIGINT)**를 사용하여 결과 프로세스를 중지하면 컨테이너 프로세스와 컨테이너 자체도 중지됩니다.

Podman은 고유한 컨테이너 ID 또는 컨테이너 이름으로 컨테이너를 식별합니다. `podman ps` 명령은 활성으로 실행 중인 모든 컨테이너의 컨테이너 ID와 이름을 표시합니다.

3장 | 컨테이너 관리

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID      IMAGE               COMMAND                  ... NAMES
47c9aad6049①    rhsc1/httpd-24-rhel7 "httpd -D FOREGROUND" ... focused_fermat②
```

- ① 컨테이너 ID는 고유하며 자동으로 생성됩니다.
- ② 컨테이너 이름은 수동으로 지정할 수 있지만, 지정하지 않을 경우 자동으로 생성됩니다. 이 이름은 고유해야 하며, 그렇지 않으면 `run` 명령이 실패합니다.

`podman run` 명령은 고유한 임의 ID를 자동으로 생성합니다. 또한 임의의 컨테이너 이름을 생성합니다. 컨테이너 이름을 명시적으로 정의하려면 컨테이너를 실행할 때 `--name` 옵션을 사용합니다.

```
[student@workstation ~]$ sudo podman run --name my-httpd-container
rhsc1/httpd-24-rhel7
...output omitted...AH00094: Command line: 'httpd -D FOREGROUND'
```

**참고**

이름은 고유해야 합니다. 중지된 컨테이너를 포함하여 이름이 이미 사용 중이면 Podman에서 오류가 발생합니다.

또 다른 중요한 기능은 컨테이너를 백그라운드에서 데몬 프로세스로 실행하는 기능입니다. `-d` 옵션은 분리 모드로 실행합니다. 이 옵션을 사용하면 Podman은 화면에 컨테이너 ID를 반환하므로, 컨테이너가 백그라운드에서 실행되는 동안 동일한 터미널에서 명령을 계속 실행할 수 있습니다.

```
[student@workstation ~]$ sudo podman run --name my-httpd-container -d
rhsc1/httpd-24-rhel7
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

컨테이너 이미지는 진입점이라는 컨테이너화된 프로세스를 시작하기 위해 실행할 명령을 지정합니다. `podman run` 명령은 컨테이너 이미지 다음에 명령을 포함하여 이 진입점을 재정의할 수 있습니다.

```
[student@workstation ~]$ sudo podman run rhsc1/httpd-24-rhel7 ls /tmp
anaconda-post.log
ks-script-1j4CXN
yum.log
```

지정한 명령은 컨테이너 이미지 내부에서 실행할 수 있어야 합니다.

**참고**

이전 예제에서는 지정한 명령이 표시되기 때문에 컨테이너가 `httpd` 이미지에 대한 진입점을 건너뜁니다. 따라서 `httpd` 서비스가 시작되지 않습니다.

일부 컨테이너는 대화형 쉘 또는 프로세스로 실행해야 합니다. 여기에는 사용자 입력(예: 명령 입력)이 필요한 프로세스와 표준 출력을 통해 출력을 생성하는 프로세스를 실행하는 컨테이너가 포함됩니다. 다음 예제에서는 `rhsc1/httpd-24-rhel7` 컨테이너의 대화형 `bash` 쉘을 시작합니다.

```
[student@workstation ~]$ sudo podman run -it rhsc1/httpd-24-rhel7 /bin/bash
bash-4.2#
```

-t 및 **-i** 옵션은 대화형 텍스트 기반 프로그램에 대해 터미널 리디렉션을 활성화합니다. **-t** 옵션은 **pseudo-tty**(터미널)를 할당하고 컨테이너의 표준 입력에 연결합니다. **-i** 옵션은 분리된 경우에도 컨테이너의 표준 입력을 열린 상태로 유지하므로, 주 프로세스가 계속 입력을 기다릴 수 있습니다.

컨테이너에서 명령 실행

컨테이너가 시작되면 진입점 명령을 실행합니다. 그러나 실행 중인 컨테이너를 관리하기 위해 다른 명령을 실행해야 할 수도 있습니다. 몇 가지 일반 사용 사례는 다음과 같습니다.

- 이미 실행 중인 컨테이너에서 대화형 쉘 실행
- 컨테이너의 파일을 업데이트하거나 표시하는 프로세스 실행
- 컨테이너 내부에서 새 백그라운드 프로세스 시작

podman exec 명령은 이미 실행 중인 컨테이너 내부에서 추가 프로세스를 시작합니다.

```
[student@workstation ~]$ sudo podman exec 7ed6e671a600 cat /etc/hostname
7ed6e671a600
```

이전 예제에서는 컨테이너 ID를 사용하여 명령을 실행합니다.

Podman은 모든 명령에서 마지막으로 사용된 컨테이너를 기억합니다. 개발자는 **-l** 옵션으로 컨테이너 ID를 대체하여 후속 Podman 명령에서 이 컨테이너의 ID 또는 이름 쓰기를 건너뛸 수 있습니다.

```
[student@workstation ~]$ sudo podman exec my-httdp-container cat /etc/hostname
7ed6e671a600
[student@workstation ~]$ sudo podman exec -l cat /etc/hostname
7ed6e671a600
```

컨테이너 관리

컨테이너 생성 및 시작은 컨테이너 라이프사이클의 첫 번째 단계일 뿐입니다. 컨테이너 라이프사이클에는 컨테이너 중지, 다시 시작 또는 마지막으로 제거도 포함됩니다. 사용자는 디버깅, 업데이트 또는 보고 목적으로 컨테이너 상태와 메타데이터를 검사할 수도 있습니다.

Podman은 컨테이너를 관리하기 위해 다음과 같은 명령을 제공합니다.

- **podman ps**: 이 명령은 실행 중인 컨테이너를 나열합니다.

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
77d4b7b8ed1f① rhscl/httpd-24-rhel7② "httpd..."③ ...ago④ Up...⑤ 80/tcp⑥ my-
htt...⑦
```

- ❶ 각 컨테이너가 생성되면 16진수인 **컨테이너 ID**를 받습니다. 이 ID는 이미지 ID와 유사하지만, 관련이 없습니다.
- ❷ 컨테이너를 시작하는 데 사용한 컨테이너 이미지입니다.
- ❸ 컨테이너를 시작할 때 실행한 명령입니다.
- ❹ 컨테이너를 시작한 날짜 및 시간입니다.
- ❺ 계속 실행 중인 경우 전체 컨테이너 가동 시간 또는 종료된 이후의 시간입니다.
- ❻ 컨테이너나 구성되었을 수 있는 포트 전달에 의해 공개된 포트입니다.
- ❼ 컨테이너 이름입니다.

3장 | 컨테이너 관리

Podman은 중지된 컨테이너를 즉시 삭제하지 않습니다. Podman은 사후 분석이 가능하도록 로컬 파일 시스템 및 다른 상태를 보존합니다. **-a** 옵션은 중지된 컨테이너를 포함하여 모든 컨테이너를 나열합니다.

```
[student@workstation ~]$ sudo podman ps -a
CONTAINER ID  IMAGE          COMMAND       CREATED      STATUS        PORTS     NAMES
4829d82fbbff  rhscl/httpd-24-rhel7  "httpd..."  ...ago      Exited (0)...  my-
httpd...
```

**참고**

컨테이너를 만드는 동안, 컨테이너 이름이 이미 사용 중이면 컨테이너가 “중지됨” 상태인 경우에도 Podman이 중단됩니다. 이 옵션은 중복된 컨테이너 이름을 방지하는 데 도움이 될 수 있습니다.

- **podman inspect**: 이 명령은 실행 중이거나 중지된 컨테이너에 대한 메타데이터를 나열합니다. 이 명령은 **JSON** 출력을 생성합니다.

```
[student@workstation ~]$ sudo podman inspect my-httpd-container
[
{
  "Id": "980e45...76c8be",
  ...output omitted...
  "NetworkSettings": {
    "Bridge": "",
    "EndpointID": "483fc9...5d801a",
    "Gateway": "172.17.42.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "HairpinMode": false,
    "IPAddress": "172.17.0.9",
    ...output omitted...
}
```

-f 옵션에 지정된 Go 템플릿을 사용하여 이 명령을 실행하면 출력 문자열의 형식을 지정할 수 있습니다. 예를 들어 IP 주소만 검색하려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' my-httpd-container
172.17.0.9
```

- **podman stop**: 이 명령은 실행 중인 컨테이너를 정상적으로 중지합니다.

```
[student@workstation ~]$ sudo podman stop my-httpd-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

podman stop을 사용하는 것이 호스트 OS에서 컨테이너 시작 프로세스를 찾아 강제 종료하는 것보다 쉽습니다.

- **podman kill**: 이 명령은 Unix 신호를 컨테이너의 주 프로세스로 전송합니다. 신호를 지정하지 않으면 **SIGKILL** 신호를 전송하여 주 프로세스와 컨테이너를 종료합니다.

```
[student@workstation ~]$ sudo podman kill my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

-s 옵션을 사용하여 신호를 지정할 수 있습니다.

```
[student@workstation ~]$ sudo podman kill -s SIGKILL my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

모든 Unix 신호를 주 프로세스로 전송할 수 있습니다. Podman은 신호 이름과 번호 중 하나를 허용합니다. 다음 테이블에는 몇 가지 유용한 신호가 나와 있습니다.

신호	값	기본 작업	주석
SIGHUP	1	용어	제어 터미널에서 종단 검색됨 또는 제어 프로세스 종료
SIGINT	2	용어	키보드에서 인터럽트
SIGQUIT	3	코어	키보드에서 종료
SIGILL	4	코어	잘못된 명령
SIGABRT	6	코어	abort(3)의 종단 신호
SIGFPE	8	코어	부동 소수점 예외
SIGKILL	9	용어	종료(kill) 신호
SIGSEGV	11	코어	잘못된 메모리 참조
SIGPIPE	13	용어	손상된 파이프: 판독기가 없는 파이프에 쓰기
SIGALRM	14	용어	alarm(2)의 타이머 신호
SIGTERM	15	용어	종료 신호
SIGUSR1	30,10,16	용어	사용자 정의 신호 1
SIGUSR2	31,12,17	용어	사용자 정의 신호 2
SIGCHLD	20,17,18	무시	하위 프로세스가 종지 또는 종료됨
SIGCONT	19,18,25	Cont	중지된 경우 계속
SIGSTOP	17,19,23	중지	프로세스 종지
SIGTSTP	18,20,24	중지	tty에서 종지 입력됨
SIGTTIN	21,21,26	중지	백그라운드 프로세스에 대한 tty 입력

신호	값	기본 작업	주석
SIGTTOU	22,22,27	중지	백그라운드 프로세스에 대한 tty 출력



참고 용어

프로세스를 종료합니다.

코어

프로세스를 종료하고 코어 덤프를 생성합니다.

무시

신호가 무시됩니다.

중지

프로세스를 중지합니다.

- **podman restart**: 이 명령은 중지된 컨테이너를 다시 시작합니다.

```
[student@workstation ~]$ sudo podman restart my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

podman restart 명령은 동일한 컨테이너 ID로 새 컨테이너를 생성하여 중지된 컨테이너 상태 및 파일 시스템을 재사용합니다.

- **podman rm**: 이 명령은 컨테이너를 삭제하고 컨테이너 상태 및 파일 시스템을 삭제합니다.

```
[student@workstation ~]$ sudo podman rm my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

rm 하위 명령의 **-f** 옵션은 중지되지 않은 경우에도 컨테이너를 제거하도록 Podman에 지시합니다. 이 옵션은 컨테이너를 강제 종료하고 제거합니다. **-f** 옵션 사용은 **podman kill** 및 **podman rm** 명령을 함께 사용하는 것과 같습니다.

모든 컨테이너를 동시에 삭제할 수 있습니다. 많은 **podman** 하위 명령은 **-a** 옵션을 허용합니다. 이 옵션은 사용 가능한 모든 컨테이너 또는 이미지에서 하위 명령을 사용함을 나타냅니다. 다음 예제에서는 모든 컨테이너를 제거합니다.

```
[student@workstation ~]$ sudo podman rm -a
5fd8e98ec7eab567eabe84943fe82e99fdfc91d12c65d99ec760d5a55b8470d6
716fd687f65b0957edac73b84b3253760e915166d3bc620c4aec8e5f4eadfe8e
86162c906b44f4cb63ba2e3386554030dcba6abedbce9e9fcad60aa9f8b2d5d4
```

모든 컨테이너를 삭제하기 전에 실행 중인 컨테이너가 모두 “중지됨” 상태여야 합니다. 다음 명령을 사용하여 모든 컨테이너를 중지할 수 있습니다.

```
[student@workstation ~]$ sudo podman stop -a
5fd8e98ec7eab567eabe84943fe82e99fdfc91d12c65d99ec760d5a55b8470d6
716fd687f65b0957edac73b84b3253760e915166d3bc620c4aec8e5f4eadfe8e
86162c906b44f4cb63ba2e3386554030dcba6abedbce9e9fcad60aa9f8b2d5d4
```



참고

`inspect`, `stop`, `kill`, `restart`, `rm` 하위 명령은 컨테이너 이름 대신 컨테이너 ID를 사용할 수 있습니다.



참조

Unix Posix 신호 도움말 페이지

<http://man7.org/linux/man-pages/man7/signal.7.html>

▶ 연습 가이드

MySQL 컨테이너 관리

이 연습에서는 MySQL® 데이터베이스 컨테이너를 만들고 관리합니다.

결과

MySQL 데이터베이스 컨테이너를 만들고 관리할 수 있습니다.

시작하기 전에

터미널 창에서 다음 명령을 실행하여 workstation에서 podman 명령을 사용할 수 있고 제대로 설정되었는지 확인합니다.

```
[student@workstation ~]$ lab manage-lifecycle start
```

- ▶ 1. MySQL 데이터베이스 컨테이너 이미지를 다운로드하고 시작해 보십시오. 이미지에 여러 환경 변수를 제공해야 하기 때문에 컨테이너가 시작되지 않습니다.

```
[student@workstation ~]$ sudo podman run --name mysql-db rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
You must either specify the following environment variables:
  MYSQL_USER (regex: '^[_a-zA-Z0-9_-]+$')
  MYSQL_PASSWORD (regex: '^[_a-zA-Z0-9_-!@#$%^&*()-=<>,_.?;:_]+$')
  MYSQL_DATABASE (regex: '^[_a-zA-Z0-9_-]+$')
Or the following environment variable:
  MYSQL_ROOT_PASSWORD (regex: '^[_a-zA-Z0-9_-!@#$%^&*()-=<>,_.?;:_]+$')
Or both.
Optional Settings:
...output omitted...

For more information, see https://github.com/sclorg/mysql-container
```



참고

컨테이너를 데몬(-d)으로 실행하면 필수 변수에 대한 오류 메시지가 표시됩니다. 이 메시지는 컨테이너 로그의 일부로 포함되며, 다음 명령을 사용하여 볼 수 있습니다.

```
[student@workstation ~]$ sudo podman logs mysql-db
```

- ▶ 2. mysql이라는 새 컨테이너를 만들고 -e 매개변수를 사용하여 각 필수 변수를 지정합니다.

**참고**

올바른 이름으로 새 컨테이너를 시작해야 합니다.

```
[student@workstation ~]$ sudo podman run --name mysql \
> -d -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r0otpa55 \
> rhsc1/mysql-57-rhel7
```

명령은 `mysql` 컨테이너의 컨테이너 ID를 표시합니다. 아래에 출력 예가 나와 있습니다.

```
a49dba9ff17f2b5876001725b581fdd331c9ab8b9eda21cc2a2899c23f078509
```

▶ 3. `mysql` 컨테이너가 제대로 시작되었는지 확인합니다. 다음 명령을 실행합니다.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
a49dba9ff17f    mysql    Up About a minute ago
```

명령은 이전 명령에 표시된 컨테이너 ID의 처음 12자만 표시합니다.

▶ 4. 컨테이너 메타데이터를 검사하여 MySQL 데이터베이스의 IP 주소를 가져옵니다.

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' mysql
10.88.0.6
```

컨테이너의 IP 주소는 위에 표시된 주소(**10.88.0.6**)와 다를 수 있습니다.

**참고**

`podman inspect` 명령은 중요한 추가 정보를 제공합니다. 예를 들어, `Env` 섹션에는 환경 변수(예: `MYSQL_ROOT_PASSWORD` 변수)가 표시됩니다.

▶ 5. `items` 데이터베이스를 `Projects` 테이블로 채웁니다.

```
[student@workstation ~]$ mysql -uuser1 -h 10.88.0.6 \
> -pmypa55 items < /home/student/D0180/labs/manage-lifecycle/db.sql
```

▶ 6. 이전 컨테이너와 동일한 컨테이너 이미지를 사용하여 다른 컨테이너를 생성합니다. 컨테이너 이미지에 기본 명령을 사용하는 대신 대화식으로 `/bin/bash` 쉘을 입력합니다.

```
[student@workstation ~]$ sudo podman run --name mysql-2 \
> -it rhsc1/mysql-57-rhel7 /bin/bash
bash-4.2$
```

- ▶ 7. 새 컨테이너의 MySQL 데이터베이스에 연결을 시도합니다.

```
bash-4.2$ mysql -uroot
```

다음과 같은 오류가 표시됩니다.

```
ERROR 2002 (HY000): Can't connect to local MySQL ...output omitted...
```

컨테이너는 MySQL 서버를 시작하는 대신 `/bin/bash` 명령을 실행하므로 MySQL 데이터베이스 서버가 실행 중이지 않습니다.

- ▶ 8. 컨테이너를 종료합니다.

```
bash-4.2$ exit
```

- ▶ 9. `mysql-2` 컨테이너가 실행 중이 아님을 확인합니다.

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
2871e392af02 mysql-2 Exited (1) 19 seconds ago
a49dba9ff17f mysql Up 10 minutes ago
c053c7e09c21 mysql-db Exited (1) 44 minutes ago
```

- ▶ 10. `mysql` 컨테이너를 쿼리하여 `Projects` 테이블의 모든 행을 나열합니다. 명령에서는 `mysql` 명령을 사용하여 `items` 데이터베이스를 쿼리하도록 `bash` 쉘에 지시합니다.

```
[student@workstation ~]$ sudo podman exec mysql /bin/bash \
> -c 'mysql -uuser1 -pmypa55 -e "select * from items.Projects;"'
mysql: [Warning] Using a password on the command line interface can be insecure.
id      name      code
1       DevOps    D0180
```

완료

`workstation`에서 `lab manage-lifecycle finish` 스크립트를 실행하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab manage-lifecycle finish
```

이로써 연습이 완료됩니다.

컨테이너에 영구저장장치 연결

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- 컨테이너를 다시 시작할 때마다 영구 스토리지를 사용하여 애플리케이션 데이터를 저장합니다.
- 컨테이너 볼륨으로 사용할 호스트 디렉터리를 구성합니다.
- 컨테이너 내부에 볼륨을 마운트합니다.

영구 스토리지 위치 준비

컨테이너 스토리지는 임시 저장소이므로, 컨테이너를 제거한 후에는 콘텐츠가 보존되지 않습니다. 컨테이너화된 애플리케이션은 항상 빈 스토리지로 시작한다는 가정하에 작업하므로 상대적으로 저렴하게 컨테이너를 생성하고 폐기할 수 있습니다.

이 교육 과정의 앞부분에서 컨테이너 이미지는 변경할 수 없으며 계층화된 것으로 설명했습니다. 즉, 컨테이너 이미지는 변경되지 않고 아래 계층의 콘텐츠를 추가하거나 재정의하는 계층으로 구성됩니다.

실행 중인 컨테이너는 기본 컨테이너 이미지 위에 새 계층을 가져오는데, 이 계층이 컨테이너 스토리지입니다. 처음에 이 계층은 컨테이너에 사용할 수 있는 유일한 읽기/쓰기 스토리지이며, 작업 파일, 임시 파일, 로그 파일을 생성하는 데 사용됩니다. 이러한 파일은 일시적인 것으로 간주됩니다. 이러한 파일들이 손실되더라도 애플리케이션의 실행은 중단되지 않습니다. 컨테이너 스토리지 계층은 실행 중인 컨테이너 전용이므로, 동일한 기본 이미지에서 다른 컨테이너를 생성하면 다른 읽기/쓰기 계층을 가져옵니다. 이렇게 하면 각 컨테이너의 리소스가 다른 유사한 컨테이너로부터 격리됩니다.

임시 컨테이너 스토리지는 데이터베이스와 같이 다시 시작하더라도 데이터가 유지되어야 하는 애플리케이션에는 충분하지 않습니다. 이러한 애플리케이션을 지원하려면 관리자가 컨테이너에 영구 스토리지를 제공해야 합니다.

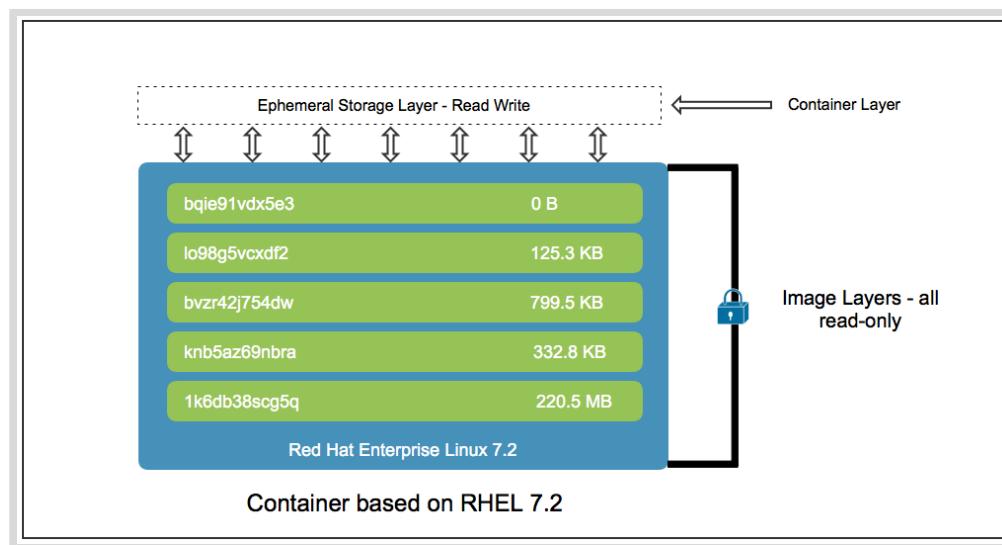


그림 3.3: 컨테이너 계층

컨테이너화된 애플리케이션에서는 컨테이너 스토리지를 사용하여 영구적 데이터를 저장하지 않아야 합니다. 콘텐츠 유지 기간을 제어할 수 없기 때문입니다. 컨테이너 스토리지를 무기한 유지할 수 있더라도 계층화된 파일 시스템은 I/O 사용량이 많은 워크로드에 대해 제대로 작동하지 않으며, 영구저장장치가 필요한 대부분의 애플리케이션에 적합하지 않습니다.

스토리지 리클레임

Podman은 실패한 컨테이너 로그에서 오류 메시지를 검토하는 등 문제 해결 작업에 사용할 수 있도록 기존의 중지된 컨테이너 스토리지를 유지합니다.

관리자가 기존 컨테이너 스토리지를 회수해야 하는 경우 **podman rm container_id**를 사용하여 컨테이너를 삭제할 수 있습니다. 이 명령은 컨테이너 스토리지도 삭제합니다. 중지된 컨테이너 ID는 **podman ps -a** 명령을 사용하여 찾을 수 있습니다.

호스트 디렉터리 준비

Podman은 실행 중인 컨테이너 내부에 호스트 디렉터리를 마운트할 수 있습니다. 일반 애플리케이션에서 원격 네트워크 볼륨이 호스트 파일 시스템의 일부로 표시되는 것처럼, 컨테이너화된 애플리케이션에서 이러한 호스트 디렉터리는 컨테이너 스토리지의 일부로 표시됩니다. 그러나 해당 호스트 디렉터리의 내용은 컨테이너를 중지한 후에 회수되지 않으며, 필요한 경우 새 컨테이너에 마운트할 수 있습니다.

예를 들어 데이터베이스 컨테이너에서 호스트 디렉터리를 사용하여 데이터베이스 파일을 저장할 수 있습니다. 이 데이터베이스 컨테이너가 실패할 경우 Podman은 동일한 호스트 디렉터리를 사용하여 새 컨테이너를 생성할 수 있습니다. 이렇게 하면 클라이언트 애플리케이션에서 데이터베이스 데이터를 계속 사용할 수 있습니다. 호스트 관점에서 이 호스트 디렉터리가 저장되는 위치는 데이터베이스 컨테이너에 중요하지 않습니다. 로컬 하드 디스크 파티션부터 원격 네트워크 파일 시스템에 이르기까지 다양합니다.

컨테이너는 호스트 운영 체제 사용자 및 그룹 ID를 사용하여 호스트 운영 체제 프로세스로 실행되므로, 컨테이너에 대한 액세스를 허용하는 소유권 및 권한을 사용하여 호스트 디렉터리를 구성해야 합니다. RHEL에서는 호스트 디렉터리도 적절한 SELinux 컨텍스트, 즉 **container_file_t**를 사용하여 구성해야 합니다. Podman은 **container_file_t** SELinux 컨텍스트를 사용하여 컨테이너가 액세스할 수 있는 호스트 시스템의 파일을 제한합니다. 이렇게 하면 호스트 시스템과 컨테이너 내부에서 실행 중인 애플리케이션 간의 정보 유출을 방지할 수 있습니다.

아래에서는 호스트 디렉터리를 설정하는 한 가지 방법을 설명합니다.

1. 소유자 및 그룹 **root**를 사용하여 디렉터리를 생성합니다.

```
[student@workstation ~]$ sudo mkdir /var/dbfiles
```

2. 컨테이너에서 프로세스를 실행하는 사용자가 디렉터리에 파일을 작성할 수 있어야 합니다. 호스트 시스템에 정확히 동일한 사용자가 정의되어 있지 않으면 컨테이너의 숫자 UID(사용자 ID)를 사용하여 권한을 정의해야 합니다. Red Hat 제공 MySQL 서비스의 경우 UID는 27입니다.

```
[student@workstation ~]$ sudo chown -R 27:27 /var/dbfiles
```

3. **container_file_t** 컨텍스트를 디렉터리(및 모든 하위 디렉터리)에 적용하여 컨테이너가 모든 디렉터리 내용에 액세스할 수 있게 합니다.

```
[student@workstation ~]$ sudo semanage fcontext -a -t container_file_t '/var/dbfiles(/.*)?'
```

4. 첫 번째 단계에서 설정한 SELinux 컨테이너 정책을 새로 생성한 디렉터리에 적용합니다.

```
[student@workstation ~]$ sudo restorecon -Rv /var/dbfiles
```

호스트 디렉터리를 사용하는 컨테이너를 시작하기 전에 해당 디렉터리를 구성해야 합니다.

볼륨 마운트

호스트 디렉터리를 생성 및 구성한 후 다음 단계는 이 디렉터리를 컨테이너에 마운트하는 것입니다. 호스트 디렉터리를 하나의 컨테이너에 바인딩하여 마운트하려면 **podman run** 명령에 **-v** 옵션을 추가하여 호스트 디렉터리 경로 및 컨테이너 스토리지 경로를 콜론(:)으로 구분하여 지정합니다.

예를 들어, MySQL 서버 데이터베이스 파일에 **mysql**이라는 MySQL 컨테이너 이미지 내부의 **/var/lib/mysql**에 있을 것으로 예상되는 **/var/dbfiles** 호스트 디렉터리를 사용하려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman run -v /var/dbfiles:/var/lib/mysql  
rhmap47/mysql
```

위 명령에서 **/var/lib/mysql**이 **mysql** 컨테이너 이미지 내부에 이미 있는 경우 **/var/dbfiles**가 오버레이를 마운트하지만 컨테이너 이미지의 컨텐츠는 제거하지 않습니다. 마운트가 제거되면 원래 컨텐츠에 다시 액세스할 수 있습니다.

▶ 연습 가이드

MySQL 데이터베이스 유지

이 연습에서는 MySQL 데이터베이스 데이터를 호스트 디렉터리에 저장하는 컨테이너를 생성합니다.

결과

영구적 데이터베이스를 사용하여 컨테이너를 배포할 수 있게 됩니다.

시작하기 전에

`workstation`에 실행 중인 컨테이너 이미지가 없어야 합니다. `workstation`에서 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab manage-storage start
```

- ▶ 1. 올바른 SELinux 컨텍스트와 권한으로 `/var/local/mysql` 디렉터리를 생성합니다.

- 1.1. `/var/local/mysql` 디렉터리를 생성합니다.

```
[student@workstation ~]$ sudo mkdir -pv /var/local/mysql
mkdir: created directory '/var/local/mysql'
```

- 1.2. `/var/local/mysql` 디렉터리 및 디렉터리 내용에 대한 적절한 SELinux 컨텍스트를 추가합니다.

```
[student@workstation ~]$ sudo semanage fcontext -a \
> -t container_file_t '/var/local/mysql(/.*)?'
```

- 1.3. 새로 생성한 디렉터리에 SELinux 정책을 적용합니다.

```
[student@workstation ~]$ sudo restorecon -R /var/local/mysql
```

- 1.4. `/var/local/mysql` 디렉터리에 대한 SELinux 컨텍스트 유형이 `container_file_t`인지 확인합니다.

```
[student@workstation ~]$ ls -ldZ /var/local/mysql
drwxr-xr-x. root root unconfined_u:object_r:container_file_t:s0 /var/local/mysql
```

- 1.5. `/var/local/mysql` 디렉터리의 소유자를 `mysql` 사용자 및 `mysql` 그룹으로 변경합니다.

```
[student@workstation ~]$ sudo chown -Rv 27:27 /var/local/mysql
changed ownership of '/var/local/mysql' from root:root to 27:27
```

**참고**

컨테이너에서 프로세스를 실행하는 사용자가 디렉터리에 파일을 작성할 수 있어야 합니다. 호스트 시스템에 정확히 동일한 사용자가 정의되어 있지 않으면 컨테이너의 숫자 UID(사용자 ID)를 사용하여 권한을 정의해야 합니다. Red Hat에서 제공하는 MySQL 서비스의 경우 UID는 27입니다.

▶ 2. 영구 스토리지를 사용하여 MySQL 컨테이너 인스턴스를 만듭니다.

2.1. MySQL 컨테이너 이미지를 가져옵니다.

```
[student@workstation ~]$ sudo podman pull rhscl/mysql-57-rhel7
Trying to pull ...output omitted...rhscl/mysql-57-rhel7...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
4ae3a3f4f409a8912cab9fbf71d3564d011ed2e68f926d50f88f2a3a72c809c5
```

2.2. MySQL 데이터베이스 데이터를 저장할 마운트 지점을 지정하는 새 컨테이너를 만듭니다.

```
[student@workstation ~]$ sudo podman run --name persist-db \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
6e0ef134315b510042ca757faf869f2ba19df27790c601f95ec2fd9d3c44b95d
```

이 명령은 호스트의 `/var/local/mysql` 디렉터리를 컨테이너의 `/var/lib/mysql/data` 디렉터리에 마운트합니다. 기본적으로 MySQL 데이터베이스는 `/var/lib/mysql/data` 디렉터리에 데이터를 저장합니다.

2.3. 컨테이너가 제대로 시작되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
6e0ef134315b persist-db Up 3 minutes ago
```

▶ 3. `/var/local/mysql` 디렉터리에 `items` 디렉터리가 포함되어 있는지 확인합니다.

```
[student@workstation ~]$ ls -ld /var/local/mysql/items
drwxr-x--- 2 27 27 20 Nov 13 12:55 /var/local/mysql/items
```

`items` 디렉터리는 이 컨테이너에서 생성한 `items` 데이터베이스와 관련된 데이터를 저장합니다. `items` 디렉터리가 없는 경우 컨테이너를 생성할 때 마운트 지점을 올바르게 정의하지 않은 것입니다.

완료

workstation에서 `lab manage-storage finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab manage-storage finish
```

이로써 연습이 완료됩니다.

컨테이너 액세스

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- 컨테이너 네트워킹의 기본을 설명합니다.
- 컨테이너 내부의 서비스에 원격으로 연결합니다.

컨테이너 네트워킹 소개

CNCF(Cloud Native Computing Foundation)는 CNI(Container Networking Interface) 오픈소스 프로젝트를 후원합니다. CNI 프로젝트는 Kubernetes, Red Hat OpenShift Container Platform 등의 클라우드 네이티브 환경에서 컨테이너에 대한 네트워크 인터페이스를 표준화하는 것을 목표로 합니다.

Podman은 CNI 프로젝트를 사용하여 각 호스트에서 컨테이너에 대한 SDN(소프트웨어 정의 네트워크)을 구현합니다. Podman은 각 컨테이너를 가상 브리지에 연결하고 각 컨테이너에 개인 IP 주소를 할당합니다. Podman의 CNI 설정을 지정하는 구성 파일은 `/etc/cni/net.d/87-podman-bridge.conflist`입니다.

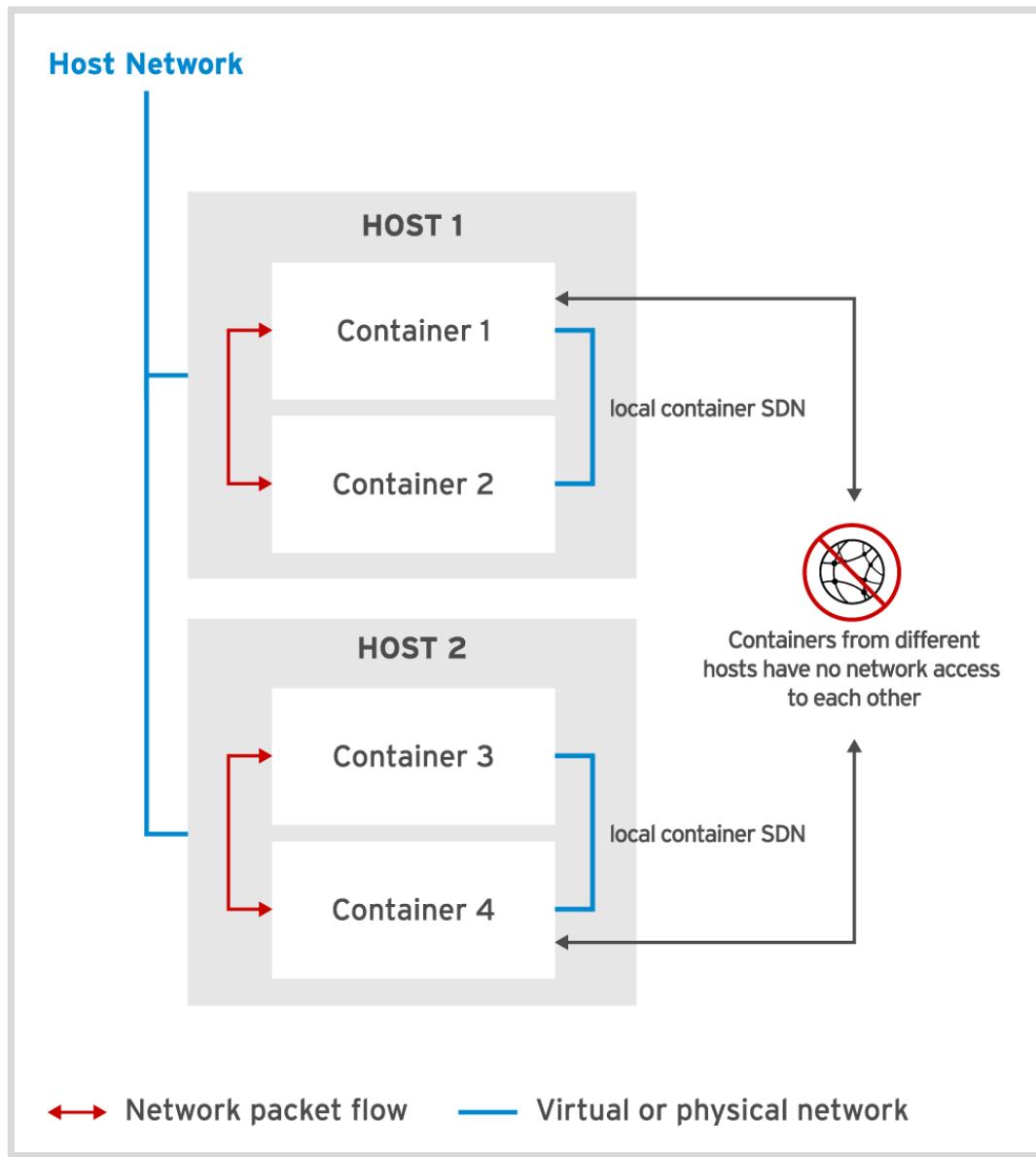


그림 3.4: 기본 Linux 컨테이너 네트워킹

Podman은 동일한 호스트에 컨테이너를 생성할 때 각 컨테이너에 고유한 IP 주소를 할당하고, 모두 동일한 소프트웨어 정의 네트워크에 연결합니다. 이러한 컨테이너는 IP 주소를 통해 서로 자유롭게 통신할 수 있습니다.

다른 호스트에서 실행 중인 Podman을 사용하여 생성한 컨테이너는 다른 소프트웨어 정의 네트워크에 속합니다. 각 SDN이 격리되므로 한 네트워크의 컨테이너가 다른 네트워크의 컨테이너와 통신할 수 없습니다. 네트워크 격리 때문에 한 SDN에 있는 컨테이너는 다른 SDN에 있는 컨테이너와 동일한 IP 주소를 가질 수 있습니다.

또한 기본적으로 모든 컨테이너 네트워크는 호스트 네트워크에서 숨겨진다는 것에 유의해야 합니다. 즉, 컨테이너는 일반적으로 호스트 네트워크에 액세스할 수 있지만 명시적 구성이 없으면 컨테이너 네트워크에 다시 액세스할 수 없습니다.

네트워크 포트 매팅

호스트 네트워크에서 컨테이너에 액세스하는 것이 어려울 수 있습니다. 컨테이너에는 사용 가능한 주소 풀의 IP 주소가 할당됩니다. 컨테이너가 폐기되면 컨테이너 주소가 다시 사용 가능한 주소 풀로 해제됩니다. 또 다른 문제는 컨테이너 호스트에서만 컨테이너 소프트웨어 정의 네트워크에 액세스할 수 있다는 점입니다.

이러한 문제를 해결하려면 컨테이너 서비스에 대한 외부 액세스를 허용하는 포트 전달 규칙을 정의합니다. `podman run` 명령에 `-p [<IP address>:][:<host port>:<container port>]` 옵션을 사용하여 외부에서 액세스 가능한 컨테이너를 생성합니다. 다음 예제를 고려해 보십시오.

```
[student@workstation ~]$ sudo podman run -d --name apache1 -p 8080:80
rhsc1/httpd-24-rhel7:2.4
```

값 `8080:80`은 호스트의 포트 8080에 대한 모든 요청이 컨테이너 내의 포트 80으로 전달되도록 지정합니다.

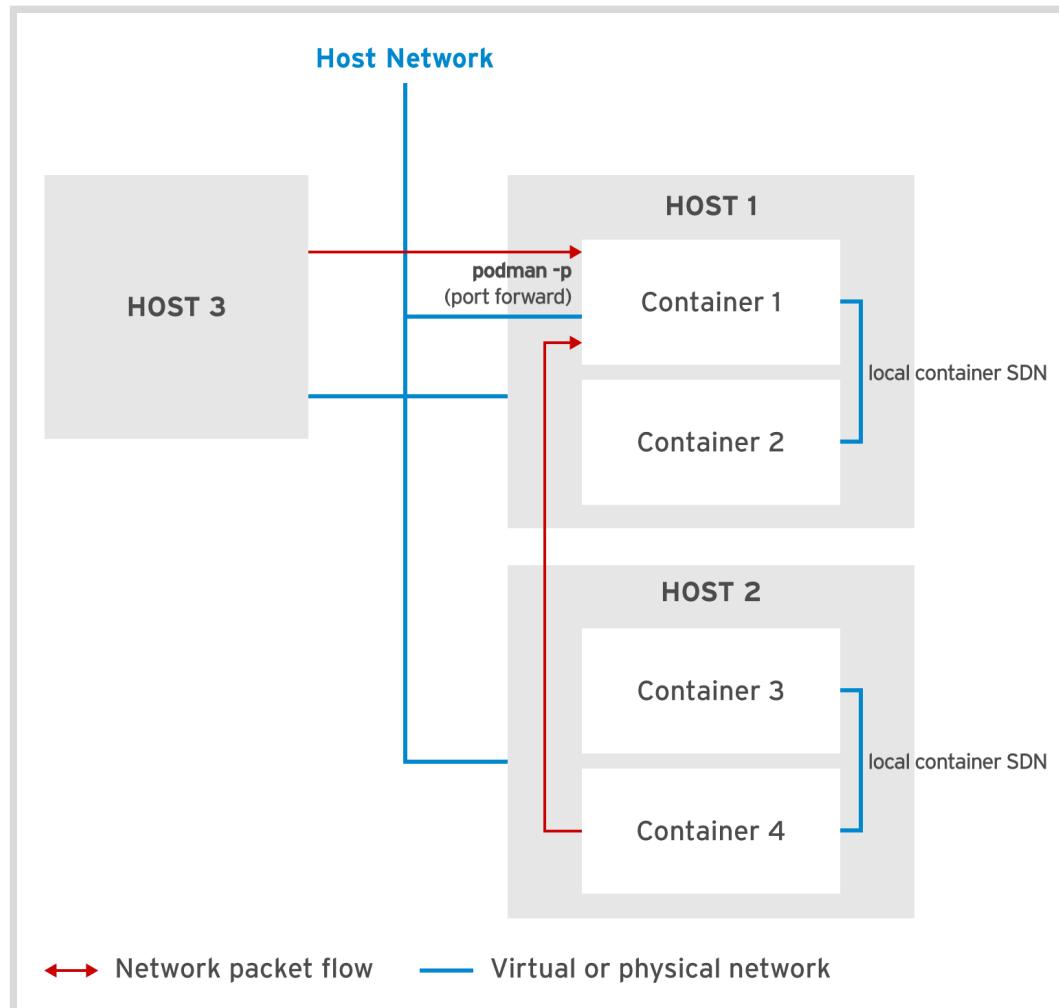


그림 3.5: Linux 컨테이너에 대한 외부 액세스 허용

`-p` 옵션을 사용하여 해당 요청이 지정한 IP 주소에서 시작된 경우에만 요청을 컨테이너로 전달할 수도 있습니다.

```
[student@workstation ~]$ sudo podman run -d --name apache2 \
> -p 127.0.0.1:8081:80 rhsc1/httpd-24-rhel7:2.4
```

3장 | 컨테이너 관리

위의 예제에서는 **apache2** 컨테이너에 대한 외부 액세스를 호스트 포트 8081에 대한 **로컬 호스트**의 요청으로 제한합니다. 이러한 요청은 **apache2** 컨테이너의 포트 80으로 전달됩니다.

호스트 포트에 대해 포트를 지정하지 않으면 Podman은 컨테이너에 사용 가능한 임의의 호스트 포트를 할당합니다.

```
[student@workstation ~]$ sudo podman run -d --name apache3 -p 127.0.0.1::80  
rhsc1/httpd-24-rhel7:2.4
```

Podman에서 할당한 포트를 확인하려면 **podman port <container name>** 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman port apache3  
80/tcp -> 127.0.0.1:35134  
[student@workstation ~]$ curl 127.0.0.1:35134  
<html><body><h1>It works!</h1></body></html>
```

-p 옵션을 사용하여 컨테이너 포트를 지정한 경우에만 사용 가능한 임의의 호스트 포트가 컨테이너에 할당됩니다. 이 할당된 호스트 포트에 대한 모든 IP 주소의 요청이 컨테이너 포트로 전달됩니다.

```
[student@workstation ~]$ sudo podman run -d --name apache4 -p 80  
rhsc1/httpd-24-rhel7:2.4  
[student@workstation ~]$ sudo podman port apache4  
80/tcp -> 0.0.0.0:37068
```

위의 예제에서 호스트 포트 **37068**에 대한 모든 라우팅 가능 요청은 컨테이너의 포트 80으로 전달됩니다.



참조

Container Network Interface - Linux 컨테이너용 네트워킹

<https://github.com/containernetworking/cni>

Cloud Native Computing Foundation

<https://www.cncf.io/>

▶ 연습 가이드

데이터베이스 로드

이 연습에서는 포트 전달을 활성화하지 않고 MySQL 데이터베이스 컨테이너를 생성합니다. SQL 스크립트를 사용하여 데이터베이스를 채운 다음 세 가지 다른 방법을 사용하여 데이터베이스 콘텐츠를 확인합니다.

결과

데이터베이스 컨테이너를 배포하고 SQL 스크립트를 로드할 수 있습니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab manage-networking start
```

이렇게 하면 `/var/local/mysql` 디렉터리가 존재하며, MySQL 컨테이너에 대해 영구저장장치를 활성화하도록 올바른 권한으로 구성됩니다.

- ▶ 1. 영구저장장치 및 포트 전달을 사용하여 MySQL 컨테이너 인스턴스를 생성합니다.

```
[student@workstation ~]$ sudo podman run --name mysqladb-port \
> -d -v /var/local/mysql:/var/lib/mysql/data -p 13306:3306 \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
Copying blob sha256:1c9f515...output omitted...
  72.70 MB / ? [=====] 5s
Copying blob sha256:1d2c4ce...output omitted...
  1.54 KB / ? [=====] 0s
Copying blob sha256:f1e961f...output omitted...
  6.85 MB / ? [=====] 0s
Copying blob sha256:9f1840c...output omitted...
  62.31 MB / ? [=====] 7s
Copying config sha256:60726...output omitted...
```

3장 | 컨테이너 관리

```
6.85 KB / 6.85 KB [=====] 0s
Writing manifest to image destination
Storing signatures
066630d45cb902ab533d503c83b834aa6a9f9cf88755cb68eedb8a3e8edbc5aa
```

각 이미지 계층을 다운로드하는 데 필요한 시간뿐 아니라 출력의 마지막 줄은 위에 표시된 것과 다릅니다.

-p 옵션은 로컬 호스트의 포트 13306가 컨테이너 포트 3306으로 전달되도록 포트 전달을 구성합니다.

**참고**

시작 스크립트는 컨테이너화된 데이터베이스에 필요한 적절한 소유권 및 SELinux 컨텍스트를 사용하여 `/var/local/mysql` 디렉터리를 생성합니다.

- ▶ 2. `mysqldb-port` 컨테이너가 제대로 시작되었는지 확인하고 포트 전달을 활성화합니다.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Ports}}"
9941da2936a5  mysql ldb-port  0.0.0.0:13306->3306/tcp
```

- ▶ 3. 제공된 파일을 사용하여 데이터베이스를 채웁니다. 오류가 없으면 명령은 출력을 반환하지 않습니다.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypassword \
> -P13306 items < /home/student/D0180/labs/manage-networking/db.sql
```

데이터베이스가 제대로 로드되었는지 확인하는 방법에는 여러 가지가 있습니다. 다음 단계에서는 세 가지 다른 방법을 보여줍니다. 이러한 방법 중 하나를 완료하기만 하면 됩니다.

- ▶ 4. 컨테이너에서 비대화형 명령을 실행하여 데이터베이스가 제대로 로드되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman exec -it mysql ldb-port \
> /opt/rh/rh-mysql57/root/usr/bin/mysql -uroot items -e "SELECT * FROM Item"
+-----+
| id | description      | done |
+-----+
| 1  | Pick up newspaper |  0  |
| 2  | Buy groceries     |  1  |
+-----+
```

**참고**

`mysql` 명령이 컨테이너 PATH 변수에 정의된 디렉터리에 없으므로 절대 경로를 사용해야 합니다.

- ▶ 5. 로컬 호스트에서 포트 전달을 사용하여 데이터베이스가 제대로 로드되었는지 확인합니다. 이 대체 방법은 선택사항입니다.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
> -P13306 items -e "SELECT * FROM Item"
+---+-----+---+
| id | description | done |
+---+-----+---+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+---+-----+---+
```

- ▶ 6. 컨테이너에서 대화형 터미널 세션을 열어 데이터베이스가 제대로 로드되었는지 확인합니다. 이 대체 방법은 선택사항입니다.

- 6.1. 컨테이너 내부에서 Bash 쉘을 엽니다.

```
[student@workstation ~]$ sudo podman exec -it mysql5db-port /bin/bash
bash-4.2$
```

- 6.2. 데이터베이스에 데이터가 포함되어 있는지 확인합니다.

```
bash-4.2$ mysql -uroot items -e "SELECT * FROM Item"
+---+-----+---+
| id | description | done |
+---+-----+---+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+---+-----+---+
```

- 6.3. 컨테이너를 종료합니다.

```
bash-4.2$ exit
```

완료

workstation에서 `lab manage-networking finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab manage-networking finish
```

이로써 연습이 완료됩니다.

▶ 랩

컨테이너 관리

성능 체크리스트

이 랩에서는 MySQL 데이터베이스 데이터를 호스트에 있는 폴더에 저장하는 컨테이너를 배포한 다음, 또 다른 컨테이너의 데이터베이스를 로드합니다.

결과

공유 볼륨을 사용하여 데이터베이스를 영구적으로 배포 및 관리할 수 있습니다. 또한 동일한 공유 볼륨을 사용하여 두 번째 데이터베이스를 시작하고, 호스트에서 동일한 디렉터리를 사용하여 MySQL 데이터를 저장하기 때문에 두 컨테이너 간에 데이터 일관성이 유지되는 것을 관찰할 수 있게 됩니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab manage-review start
```

1. 올바른 SELinux 컨텍스트와 권한으로 `/var/local/mysql` 디렉터리를 생성합니다.
 - 1.1. `/var/local/mysql` 디렉터리를 생성합니다.
 - 1.2. `/var/local/mysql` 디렉터리 및 디렉터리 내용에 대한 적절한 SELinux 컨텍스트를 추가합니다. 올바른 컨텍스트를 사용하여 이 디렉터리를 실행 중인 컨테이너에 마운트할 수 있습니다.
 - 1.3. 새로 생성한 디렉터리에 SELinux 정책을 적용합니다.
 - 1.4. `rhsc1/mysql-57-rhel7` 컨테이너 이미지의 `mysql` 사용자 및 `mysql` 그룹과 일치하도록 `/var/local/mysql` 디렉터리의 소유자를 변경합니다.
2. 다음 특성을 사용하여 MySQL 컨테이너 인스턴스를 배포합니다.
 - **이름:** mysql-1
 - **데몬으로 실행:** yes
 - **볼륨:** from `/var/local/mysql` host folder to `/var/lib/mysql/data` container folder
 - **컨테이너 이미지:** `rhsc1/mysql-57-rhel7`
 - **포트 전달:** no;
 - **환경 변수:**
 - `MYSQL_USER: user1`
 - `MYSQL_PASSWORD: mypa55`
 - `MYSQL_DATABASE: items`
 - `MYSQL_ROOT_PASSWORD: r00tpa55`

3. /home/student/D0180/labs/manage-review/db.sql 스크립트를 사용하여 **items** 데이터베이스를 로드합니다.

- 3.1. 컨테이너 IP 주소를 가져옵니다.
- 3.2. /home/student/D0180/labs/manage-review/db.sql에서 SQL 명령을 사용하여 데이터베이스를 로드합니다. 이전 단계에서 찾은 IP 주소를 데이터베이스 서버의 호스트 IP로 사용합니다.

**참고**

직접 입력하는 대신, mysql 명령 뒤에 보다 작은 연산자(<)를 사용하여 위 파일의 모든 명령을 가져올 수 있습니다. 또한 mysql 명령에 -h CONTAINER_IP 매개 변수를 추가하여 올바른 컨테이너에 연결해야 합니다.

- 3.3. SQL **SELECT** 문을 통해 Item 테이블의 모든 행을 출력하여 Items 데이터베이스가 로드되었는지 확인합니다.

**참고**

mysql 명령에 -e SQL 매개 변수를 추가하여 SQL 명령을 실행할 수 있습니다.

4. 컨테이너를 정상적으로 중지합니다.

**중요**

데이터베이스 데이터에 대해 동일한 볼륨을 공유하여 새 컨테이너가 생성되기 때문에 이 단계는 매우 중요합니다. 동일한 볼륨을 사용하는 두 개의 컨테이너가 있으면 데이터베이스가 손상될 수 있습니다. mysql-1 컨테이너를 다시 시작하지 마십시오.

5. 다음 특성을 사용하여 새 컨테이너를 만듭니다.

- **이름:** mysql-2
- **데몬으로 실행:** yes
- **볼륨:** from /var/local/mysql host folder to /var/lib/mysql/data container folder
- **컨테이너 이미지:** rhsc1/mysql-57-rhel7
- **포트 전달:** yes, from host port 13306 to container port 3306
- **환경 변수:**
 - MySQL_USER: user1
 - MySQL_PASSWORD: mypa55
 - MySQL_DATABASE: items
 - MySQL_ROOT_PASSWORD: r00tpa55

6. 중지된 컨테이너를 포함하여 모든 컨테이너 목록을 /tmp/my-containers 파일에 저장합니다.

7. 컨테이너 내부의 Bash 쉘에 액세스하고 **items** 데이터베이스 및 **Item** 테이블을 여전히 사용할 수 있는지 확인합니다. 또한 테이블에 데이터가 포함되어 있는지 확인합니다.

- 7.1. 컨테이너 내부의 Bash 쉘에 액세스합니다.
 - 7.2. MySQL 서버에 연결합니다.
 - 7.3. 모든 데이터베이스를 나열하고 **items** 데이터베이스가 사용 가능한지 확인합니다.
 - 7.4. **items** 데이터베이스의 모든 테이블을 나열하고 **Item** 테이블이 사용 가능한지 확인합니다.
 - 7.5. 테이블의 데이터를 확인합니다.
 - 7.6. MySQL 클라이언트 및 컨테이너 웰을 종료합니다.
- 8.** 포트 전달 기능을 사용하여 **Item** 테이블에 새 행을 삽입합니다. 행은 **Finished** **lab**의 **description** 값과 **1**의 **done** 값을 가집니다.
- 8.1. MySQL 데이터베이스에 연결합니다.
 - 8.2. 새 행을 삽입합니다.
 - 8.3. MySQL 클라이언트를 종료합니다.
- 9.** 첫 번째 컨테이너는 더 이상 필요하지 않으므로 이 컨테이너를 제거하여 리소스를 해제합니다.

평가

workstation 시스템에서 **lab manage-review grade** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab manage-review grade
```

완료

workstation에서 **lab manage-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab manage-review finish
```

이로써 랩이 완료됩니다.

▶ 솔루션

컨테이너 관리

성능 체크리스트

이 랩에서는 MySQL 데이터베이스 데이터를 호스트에 있는 폴더에 저장하는 컨테이너를 배포한 다음, 또 다른 컨테이너의 데이터베이스를 로드합니다.

결과

공유 볼륨을 사용하여 데이터베이스를 영구적으로 배포 및 관리할 수 있습니다. 또한 동일한 공유 볼륨을 사용하여 두 번째 데이터베이스를 시작하고, 호스트에서 동일한 디렉터리를 사용하여 MySQL 데이터를 저장하기 때문에 두 컨테이너 간에 데이터 일관성이 유지되는 것을 관찰할 수 있게 됩니다.

시작하기 전에

workstation에서 student 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab manage-review start
```

1. 올바른 SELinux 컨텍스트와 권한으로 /var/local/mysql 디렉터리를 생성합니다.

- 1.1. /var/local/mysql 디렉터리를 생성합니다.

```
[student@workstation ~]$ sudo mkdir -pv /var/local/mysql
mkdir: created directory '/var/local/mysql'
```

- 1.2. /var/local/mysql 디렉터리 및 디렉터리 내용에 대한 적절한 SELinux 컨텍스트를 추가합니다. 올바른 컨텍스트를 사용하여 이 디렉터리를 실행 중인 컨테이너에 마운트할 수 있습니다.

```
[student@workstation ~]$ sudo semanage fcontext -a \
> -t container_file_t '/var/local/mysql(/.*)?'
```

- 1.3. 새로 생성한 디렉터리에 SELinux 정책을 적용합니다.

```
[student@workstation ~]$ sudo restorecon -R /var/local/mysql
```

- 1.4. rhel7/mysql-5.7-rhel7 컨테이너 이미지의 mysql 사용자 및 mysql 그룹과 일치하도록 /var/local/mysql 디렉터리의 소유자를 변경합니다.

```
[student@workstation ~]$ sudo chown -Rv 27:27 /var/local/mysql
changed ownership of '/var/local/mysql' from root:root to 27:27
```

2. 다음 특성을 사용하여 MySQL 컨테이너 인스턴스를 배포합니다.

- **이름:** mysql-1
- **데몬으로 실행:** yes

3장 | 컨테이너 관리

- **볼륨**: from /var/local/mysql host folder to /var/lib/mysql/data container folder
- **컨테이너 이미지**: rhscl/mysql-57-rhel7
- **포트 전달**: no;
- **환경 변수**:
 - MYSQL_USER: user1
 - MYSQL_PASSWORD: mypa55
 - MYSQL_DATABASE: items
 - MYSQL_ROOT_PASSWORD: r00tpa55

2.1. 컨테이너를 만들고 시작합니다.

```
[student@workstation ~]$ sudo podman run --name mysql-1 \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mpa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
616azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

2.2. 컨테이너가 제대로 시작되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}}"
616azfaa55x8    mysql-1
```

3. /home/student/D0180/labs/manage-review/db.sql 스크립트를 사용하여 items 데이터베이스를 로드합니다.

3.1. 컨테이너 IP 주소를 가져옵니다.

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' mysql-1
10.88.0.6
```

- 3.2. /home/student/D0180/labs/manage-review/db.sql에서 SQL 명령을 사용하여 데이터베이스를 로드합니다. 이전 단계에서 찾은 IP 주소를 데이터베이스 서버의 호스트 IP로 사용합니다.

**참고**

직접 입력하는 대신, mysql 명령 뒤에 보다 작은 연산자(<)를 사용하여 위 파일의 모든 명령을 가져올 수 있습니다. 또한 mysql 명령에 -h CONTAINER_IP 매개 변수를 추가하여 올바른 컨테이너에 연결해야 합니다.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP \
> -pmypa55 items < /home/student/D0180/labs/manage-review/db.sql
```

- 3.3. SQL **SELECT** 문을 통해 Item 테이블의 모든 행을 출력하여 Items 데이터베이스가 로드되었는지 확인합니다.



참고

`mysql` 명령에 `-e` SQL 매개 변수를 추가하여 SQL 명령을 실행할 수 있습니다.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP -pmypa55 items \
> -e "SELECT * FROM Item"
+----+-----+----+
| id | description | done |
+----+-----+----+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+----+-----+----+
```

4. 컨테이너를 정상적으로 종지합니다.



중요

데이터베이스 데이터에 대해 동일한 볼륨을 공유하여 새 컨테이너가 생성되기 때문에 이 단계는 매우 중요합니다. 동일한 볼륨을 사용하는 두 개의 컨테이너가 있으면 데이터베이스가 손상될 수 있습니다. `mysql-1` 컨테이너를 다시 시작하지 마십시오.

다음 명령을 사용하여 컨테이너를 종지합니다.

```
[student@workstation ~]$ sudo podman stop mysql-1
616azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

5. 다음 특성을 사용하여 새 컨테이너를 만듭니다.

- **이름:** `mysql-2`
- **데몬으로 실행:** `yes`
- **볼륨:** from `/var/local/mysql` host folder to `/var/lib/mysql/data` container folder
- **컨테이너 이미지:** `rhscl/mysql-57-rhel7`
- **포트 전달:** `yes`, from host port 13306 to container port 3306
- **환경 변수:**
 - `MYSQL_USER: user1`
 - `MYSQL_PASSWORD: mypa55`

3장 | 컨테이너 관리

- MYSQL_DATABASE: items
- MYSQL_ROOT_PASSWORD: r00tpa55

5.1. 컨테이너를 만들고 시작합니다.

```
[student@workstation ~]$ sudo podman run --name mysql-2 \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -p 13306:3306 \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhsc1/mysql-57-rhel7
281c0e2790e54cd5a0b8e2a8cb6e3969981b85cde8ac611bf7ea98ff78bdffbb
```

5.2. 컨테이너가 제대로 시작되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}}"
281c0e2790e5    mysql-2
```

6. 중지된 컨테이너를 포함하여 모든 컨테이너 목록을 `/tmp/my-containers` 파일에 저장합니다.

다음 명령을 사용하여 정보를 저장합니다.

```
[student@workstation ~]$ sudo podman ps -a > /tmp/my-containers
```

7. 컨테이너 내부의 Bash 쉘에 액세스하고 `items` 데이터베이스 및 `Item` 테이블을 여전히 사용할 수 있는지 확인합니다. 또한 테이블에 데이터가 포함되어 있는지 확인합니다.

7.1. 컨테이너 내부의 Bash 쉘에 액세스합니다.

```
[student@workstation ~]$ sudo podman exec -it mysql-2 /bin/bash
```

7.2. MySQL 서버에 연결합니다.

```
bash-4.2$ mysql -uroot
```

7.3. 모든 데이터베이스를 나열하고 `items` 데이터베이스가 사용 가능한지 확인합니다.

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.03 sec)
```

7.4. `items` 데이터베이스의 모든 테이블을 나열하고 `Item` 테이블이 사용 가능한지 확인합니다.

3장 | 컨테이너 관리

```
mysql> use items;
Database changed
mysql> show tables;
+-----+
| Tables_in_items |
+-----+
| Item           |
+-----+
1 row in set (0.01 sec)
```

7.5. 테이블의 데이터를 확인합니다.

```
mysql> SELECT * FROM Item;
+----+-----+----+
| id | description | done |
+----+-----+----+
| 1  | Pick up newspaper | 0 |
| 2  | Buy groceries | 1 |
+----+-----+----+
```

7.6. MySQL 클라이언트 및 컨테이너 쉘을 종료합니다.

```
mysql> exit
Bye
bash-4.2$ exit
```

8. 포트 전달 기능을 사용하여 **Item** 테이블에 새 행을 삽입합니다. 행은 **Finished lab**의 **description** 값과 1의 **done** 값을 가집니다.

8.1. MySQL 데이터베이스에 연결합니다.

```
[student@workstation ~]$ mysql -uuser1 -h workstation.lab.example.com \
> -pmypa55 -P13306 items
...output omitted...

Welcome to the MariaDB monitor. Commands end with ; or \g.
...output omitted...

MySQL [items]>
```

8.2. 새 행을 삽입합니다.

```
MySQL[items]> insert into Item (description, done) values ('Finished lab', 1);
Query OK, 1 row affected (0.00 sec)
```

8.3. MySQL 클라이언트를 종료합니다.

```
MySQL[items]> exit
Bye
```

9. 첫 번째 컨테이너는 더 이상 필요하지 않으므로 이 컨테이너를 제거하여 리소스를 해제합니다.

다음 명령을 사용하여 컨테이너를 제거합니다.

```
[student@workstation ~]$ sudo podman rm mysql-1  
616azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

평가

workstation 시스템에서 **lab manage-review grade** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab manage-review grade
```

완료

workstation에서 **lab manage-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab manage-review finish
```

이로써 랩이 완료됩니다.

요약

이 장에서 학습한 내용:

- Podman에는 새 컨테이너 생성(**run**), 컨테이너 삭제(**rm**), 컨테이너 나열(**ps**), 컨테이너 중지(**stop**), 컨테이너에서 프로세스 시작(**exec**) 등의 하위 명령이 있습니다.
- 기본 컨테이너 스토리지는 임시 저장소이므로, 컨테이너를 다시 시작하거나 제거한 후에는 콘텐츠가 유지되지 않습니다.
 - 컨테이너는 호스트 파일 시스템의 폴더를 사용하여 영구적 데이터로 작업할 수 있습니다.
 - Podman은 **podman run** 명령에 **-v** 옵션을 사용하여 컨테이너에 볼륨을 마운트합니다.
- **podman exec** 명령은 실행 중인 컨테이너 내부에서 추가 프로세스를 시작합니다.
- Podman은 **run** 하위 명령에 **-p** 옵션을 사용하여 로컬 포트를 컨테이너 포트에 매핑합니다.

4장

컨테이너 이미지 관리

목적

생성에서 삭제에 이르기까지 컨테이너 이미지의 라이프 사이클을 관리합니다.

목표

- 원격 레지스트리에서 이미지를 검색하고 가져옵니다.
- 레지스트리에서 컨테이너 이미지를 로컬로 내보내고, 가져오고, 관리합니다.

섹션

- 레지스트리 액세스 및 퀴즈
- 컨테이너 이미지 조작(안내에 따른 연습)

랩

이미지 관리

레지스트리 액세스

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- Podman 명령과 레지스트리 REST API를 사용하여 원격 레지스트리에서 이미지를 검색하고 가져옵니다.
- 인증된 공개 레지스트리를 사용하여 보안 이미지를 다운로드하는 경우의 장점을 나열합니다.
- Podman의 구성을 사용자 지정하여 대체 컨테이너 이미지 레지스트리에 액세스합니다.
- 레지스트리에서 로컬 파일 시스템으로 다운로드한 이미지를 나열합니다.
- 태그를 관리하여 태그가 지정된 이미지를 가져옵니다.

공개 레지스트리

이미지 레지스트리는 컨테이너 이미지를 다운로드 할 수 있는 서비스입니다. 이미지 레지스트리를 사용하면 생성자와 유지 관리자가 컨테이너 이미지를 저장하고 퍼블릭 또는 프라이빗 대상자에게 배포할 수 있습니다.

Podman은 퍼블릭 또는 프라이빗 레지스트리에서 컨테이너 이미지를 검색하고 다운로드합니다. Red Hat Container Catalog는 Red Hat이 관리하는 공개 이미지 레지스트리입니다. Red Hat Container Catalog는 Apache, MySQL, Jenkins와 같은 주요 오픈소스 프로젝트에서 제공하는 이미지를 비롯하여 수많은 컨테이너 이미지를 호스팅합니다. Container Catalog의 모든 이미지는 Red Hat 내부 보안 팀이 검사하므로 신뢰할 수 있으며 보안 결함으로부터 보호됩니다.

Red Hat 컨테이너 이미지는 다음과 같은 혜택을 제공합니다.

- 신뢰할 수 있는 소스: 모든 컨테이너 이미지는 Red Hat에서 신뢰하는 알려진 소스로 구성됩니다.
- 원본 종속성: 컨테이너 패키지가 변조되지 않았으며, 알려진 라이브러리만 포함합니다.
- 취약성 없음: 컨테이너 이미지에는 플랫폼 구성 요소 또는 계층의 알려진 취약성이 없습니다.
- 런타임 보호: 컨테이너 이미지의 모든 애플리케이션은 root가 아닌 사용자로 실행되므로 악의적이거나 결함이 있는 애플리케이션에 대한 노출 표면이 최소화됩니다.
- Red Hat Enterprise Linux(RHEL) 호환 가능: 컨테이너 이미지는 베어 메탈에서 클라우드까지 모든 RHEL 플랫폼과 호환됩니다.
- Red Hat 지원: Red Hat은 전체 스택을 상업적으로 지원합니다.

Quay.io는 Red Hat에서 후원하는 또 다른 퍼블릭 이미지 리포지토리입니다. Quay.io는 서버 쪽 이미지 빌드, 세분화된 액세스 제어, 알려진 취약성에 대한 이미지 자동 검사 등 몇 가지 흥미로운 기능을 도입합니다.

Red Hat Container Catalog 이미지는 신뢰할 수 있고 검증되었으며 Quay.io는 작성자가 정기적으로 업데이트하는 라이브 이미지를 제공합니다. Quay.io 사용자는 세분된 액세스 제어를 사용하여 네임스페이스를 만들고 생성한 이미지를 해당 네임스페이스에 게시할 수 있습니다. Container Catalog 사용자는 새 이미지를 거의 혹은 아예 내보내지 않지만 Red Hat 팀에서 생성한 신뢰할 수 있는 이미지를 사용합니다.

프라이빗 레지스트리

일부 이미지 작성자 또는 관리자는 자신의 이미지를 공개 할 수 있습니다. 그러나 다음과 같은 이유로 이미지를 비공개로 유지하려는 이미지 생성자도 있습니다.

- 회사의 개인 정보 보호 및 비밀 보호
- 법적 제한 및 법률
- 개발 중인 이미지 게시 방지

프라이빗 레지스트리를 사용하면 이미지 생성자가 이미지 배치, 배포 및 사용을 제어할 수 있습니다.

Podman에서 레지스트리 구성

`podman` 명령에 대해 레지스트리를 구성하려면 `/etc/containers/registries.conf` 파일을 업데이트해야 합니다. `[registries.search]` 섹션의 `registries` 항목을 편집하고 값 목록에 항목을 추가합니다.

```
[registries.search]
registries = ["registry.access.redhat.com", "quay.io"]
```



참고

FQDN과 포트 번호를 사용하여 레지스트리를 확인합니다. 포트 번호를 포함하지 않는 레지스트리는 기본 포트 번호 5000을 사용합니다. 레지스트리가 다른 포트를 사용하는 경우 포트를 지정해야 합니다. FQDN 뒤에 콜론(:)과 포트 번호를 추가하여 포트 번호를 지정합니다.

레지스트리에 대한 보안 연결에는 신뢰할 수 있는 인증서가 필요합니다. 안전하지 않은 연결을 지원하려면 레지스트리 이름을 `/etc/containers/registries.conf` 파일의 `[registries.insecure]` 섹션에 있는 `registries` 항목에 추가합니다.

```
[registries.insecure]
registries = ['localhost:5000']
```

레지스트리 액세스

레지스트리의 이미지 검색

`podman search` 명령은 `/etc/containers/registries.conf` 구성 파일에 나열된 모든 레지스트리에서 이미지 이름, 사용자 이름 또는 설명으로 이미지를 찾습니다. `podman search` 명령의 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman search [OPTIONS] <term>
```

다음 테이블에는 `search` 하위 명령에 사용할 수 있는 몇 가지 유용한 옵션이 나와 있습니다.

옵션	설명
<code>--limit <number></code>	레지스트리당 나열되는 이미지 수를 제한합니다.

옵션	설명
--filter <filter=value>	제공된 조건에 따라 출력을 필터링합니다. 지원되는 필터는 다음과 같습니다. <ul style="list-style-type: none"> . stars=<number>: 별이 이 개수 이상인 이미지만 표시합니다. . is-automated=<true false>: 자동으로 빌드된 이미지만 표시합니다. . is-official=<true false>: 공식으로 플래그가 지정된 이미지만 표시합니다.
--tls-verify <true false>	사용된 모든 레지스트리에 대해 HTTPS 인증서 유효성 검사를 활성화하거나 비활성화합니다. true

레지스트리 HTTP API

원격 레지스트리는 레지스트리에 대한 API(애플리케이션 프로그래밍 인터페이스)를 제공하는 웹 서비스를 공개합니다. Podman은 이러한 인터페이스를 사용하여 원격 리포지토리에 액세스하고 상호 작용합니다. 많은 레지스트리가 레지스트리 상호 작용을 위한 표준화된 REST 인터페이스를 공개하는 **Docker Registry HTTP API v2** 사양을 준수합니다. Podman을 사용하는 대신, 이 REST 인터페이스를 사용하여 레지스트리를 직접 조작할 수 있습니다.

curl 명령과 함께 이 API를 사용하는 몇 가지 샘플은 다음과 같습니다.

레지스트리에서 사용 가능한 모든 리포지토리를 나열하려면 **/v2/_catalog** 엔드포인트를 사용합니다. **n** 매개 변수는 반환할 리포지토리 수를 제한하는 데 사용됩니다.

```
[student@workstation ~]$ curl -Ls https://myserver/v2/_catalog?n=3
{"repositories": ["centos/httpd", "do180/custom-httpd", "hello-openshift"]}
```



참고

Python을 사용할 수 있는 경우 Python을 사용하여 JSON 응답의 형식을 지정합니다.

```
[student@workstation ~]$ curl -Ls https://myserver/v2/_catalog?n=3 \
> | python -m json.tool
{
  "repositories": [
    "centos/httpd",
    "do180/custom-httpd",
    "hello-openshift"
  ]
}
```

/v2/<name>/tags/list 엔드포인트는 단일 이미지에 사용할 수 있는 태그 목록을 제공합니다.

```
[student@workstation ~]$ curl -Ls \
> https://quay.io/v2/redhattraining/httpd-parent/tags/list \
> | python -m json.tool
```

```
{
  "name": "redhattraining/httpd-parent",
  "tags": [
    "latest",
    "2.4"
  ]
}
```



참고

Quay.io는 Docker 리포지토리 API에 지정된 것보다는 리포지토리와 상호 작용하는 전용 API를 제공합니다. 자세한 내용은 <https://docs.quay.io/api/>를 참조하십시오.

레지스트리 인증

일부 컨테이너 이미지 레지스트리에는 액세스 권한이 필요합니다. `podman login` 명령은 레지스트리에 대한 사용자 이름 및 암호 인증을 허용합니다.

```
[student@workstation ~]$ sudo podman login -u username \
> -p password registry.access.redhat.com
Login Succeeded!
```

레지스트리 HTTP API에는 인증 자격 증명이 필요합니다. 먼저 Red Hat SSO(Single Sign On) 서비스를 사용하여 액세스 토큰을 얻습니다.

```
[student@workstation ~]$ curl -u username:password -Ls \
> "https://sso.redhat.com/auth/realms/rhcc/protocol/redhat-docker-v2/auth?
service=docker-registry"
{"token":"eyJh...o5G8",
"access_token":"eyJh...mgL4",
"expires_in":...output omitted...}[student@workstation ~]$
```

그런 다음, 이 토큰을 후속 요청의 `Bearer` 인증 헤더에 포함합니다.

```
[student@workstation ~]$ curl -H "Authorization: Bearer eyJh...mgL4" \
> -Ls https://registry.redhat.io/v2/rhscl/mysql-57-rhel7/tags/list \
> | python -mjson.tool
{
  "name": "rhscl/mysql-57-rhel7",
  "tags": [
    "5.7-3.9",
    "5.7-3.8",
    "5.7-3.4",
    "5.7-3.7",
    ...output omitted...
```



참고

다른 레지스트리에는 자격 증명을 제공하는 다른 단계가 필요할 수 있습니다. 레지스트리가 `Docker Registry HTTP v2 API`를 준수하는 경우 인증은 RFC7235 체계를 따릅니다.

이미지 가져오기

레지스트리에서 컨테이너 이미지를 가져오려면 `podman pull` 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman pull [OPTIONS] [REGISTRY[:PORT]/]NAME[:TAG]
```

`podman pull` 명령은 `search` 하위 명령을 통해 얻은 이미지 이름을 사용하여 레지스트리에서 이미지를 가져옵니다. `pull` 하위 명령을 사용하면 이미지에 레지스트리 이름을 추가할 수 있습니다. 이 변형은 여러 레지스트리에서 동일한 이미지를 사용할 수 있도록 지원합니다.

예를 들어 `quay.io` 레지스트리에서 NGINX 컨테이너를 가져오려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman pull quay.io/bitnami/nginx
```



참고

이미지 이름에 레지스트리 이름이 없으면 Podman은 `/etc/containers/registries.conf` 구성 파일에 나열된 레지스트리를 사용하여 일치하는 컨테이너 이미지를 검색합니다. Podman은 구성 파일에 표시된 순서대로 레지스트리에서 이미지를 검색합니다.

로컬 이미지 사본 나열

레지스트리에서 다운로드한 컨테이너 이미지는 `podman` 명령이 실행되는 동일한 호스트에 로컬로 저장됩니다. 이 동작은 이미지 다운로드 반복을 방지하고 컨테이너에 대한 배포 시간을 최소화합니다. Podman은 사용자가 빌드한 사용자 지정 컨테이너 이미지도 동일한 로컬 스토리지에 저장합니다.



참고

기본적으로 Podman은 컨테이너 이미지를 `/var/lib/containers/storage/overlay-images` 디렉터리에 저장합니다.

Podman은 로컬에 저장된 모든 컨테이너 이미지를 나열하는 `images` 하위 명령을 제공합니다.

```
[student@workstation ~]$ sudo podman images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
registry.redhat.io/rhscl/mysql-57-rhel7  latest   c07bf25398f4  13 days ago  444MB
```

이미지 태그

이미지 태그는 동일한 이미지의 여러 릴리스를 지원하는 메커니즘입니다. 이 기능은 동일한 소프트웨어의 여러 버전(예: 프로덕션 지원 컨테이너 또는 커뮤니티 평가용으로 개발된 동일한 소프트웨어의 최신 업데이트)이 제공되는 경우에 유용합니다. 컨테이너 이미지 이름이 필요한 Podman 하위 명령은 여러 태그를 구별하는 태그 매개 변수를 허용합니다. 이미지 이름에 태그가 없으면 태그값은 기본적으로 `latest`로 지정됩니다. 예를 들어, `rhscl/mysql-57-rhel7`에서 태그 `5.7`이 지정된 이미지를 가져오려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman pull rhscl/mysql-57-rhel7:5.7
```

`rhscl/mysql-57-rhel7:5.7` 이미지 기반의 새 컨테이너를 시작하려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman run rhscl/mysql-57-rhel7:5.7
```



참조

Red Hat Container Catalog

<https://registry.redhat.io>

Quay.io

<https://quay.io>

Docker Registry HTTP API V2

<https://github.com/docker/distribution/blob/master/docs/spec/api.md>

RFC7235 - HTTP/1.1: 인증

<https://tools.ietf.org/html/rfc7235>

▶ 퀴즈

레지스트리 사용

아래 정보를 기반으로 다음 질문에 대한 올바른 답안을 선택하십시오.

`/etc/containers/registries.conf` 파일의 다음 항목을 사용하여 RHEL 호스트에서 Podman을 사용할 수 있습니다.

```
[registries.search]
registries = ["registry.redhat.io", "quay.io"]
```

`registry.redhat.io` 및 `quay.io` 호스트에는 레지스트리가 실행 중이고, 둘 다 유효한 인증서가 있으며, 버전 1 레지스트리를 사용합니다. 다음 이미지를 각 호스트에 사용할 수 있습니다.

레지스트리별 이미지 이름/태그

레지스트리	이미지
registry.redhat.io	<ul style="list-style-type: none"> nginx/1.0 mysql/5.6 httpd/2.2
quay.io	<ul style="list-style-type: none"> mysql/5.5 httpd/2.4

로컬에서 사용할 수 있는 이미지가 없습니다.

▶ 1. 다음 중 `registry.redhat.io`에서 다운로드할 수 있는 `mysql` 이미지를 표시하는 두 가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- `podman search registry.redhat.io/mysql`
- `podman images`
- `podman pull mysql`
- `podman search mysql`

▶ 2. 다음 중 `httpd` 컨테이너 이미지에 사용할 수 있는 모든 이미지 태그를 나열하는 데 사용되는 명령은 무엇입니까?

- `podman search httpd`
- `podman images httpd`
- `podman pull --all-tags=true httpd`
- 태그를 검색하는 데 사용할 수 있는 Podman 명령이 없습니다.

▶ 3. 2.2 태그가 있는 httpd 이미지를 가져오는 2가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- a. podman pull httpd:2.2
- b. podman pull httpd:latest
- c. podman pull quay.io/httpd
- d. podman pull registry.redhat.io/httpd:2.2

▶ 4. 다음 명령을 실행할 때 다운로드되는 컨테이너 이미지는 무엇입니까?

```
podman pull registry.redhat.io/httpd:2.2  
podman pull quay.io/mysql:5.6
```

- a. quay.io/httpd:2.2
registry.redhat.io/mysql:5.6
- b. registry.redhat.io/httpd:2.2
registry.redhat.io/mysql:5.6
- c. registry.redhat.io/httpd:2.2
mysql에 대해 다운로드되는 이미지는 없습니다.
- d. quay.io/httpd:2.2
mysql에 대해 다운로드되는 이미지는 없습니다.

▶ 솔루션

레지스트리 사용

아래 정보를 기반으로 다음 질문에 대한 올바른 답안을 선택하십시오.

`/etc/containers/registries.conf` 파일의 다음 항목을 사용하여 RHEL 호스트에서 Podman을 사용할 수 있습니다.

```
[registries.search]
registries = ["registry.redhat.io", "quay.io"]
```

`registry.redhat.io` 및 `quay.io` 호스트에는 레지스트리가 실행 중이고, 둘 다 유효한 인증서가 있으며, 버전 1 레지스트리를 사용합니다. 다음 이미지를 각 호스트에 사용할 수 있습니다.

레지스트리별 이미지 이름/태그

레지스트리	이미지
registry.redhat.io	<ul style="list-style-type: none"> nginx/1.0 mysql/5.6 httpd/2.2
quay.io	<ul style="list-style-type: none"> mysql/5.5 httpd/2.4

로컬에서 사용할 수 있는 이미지가 없습니다.

▶ 1. 다음 중 `registry.redhat.io`에서 다운로드할 수 있는 `mysql` 이미지를 표시하는 두 가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- `podman search registry.redhat.io/mysql`
- `podman images`
- `podman pull mysql`
- `podman search mysql`

▶ 2. 다음 중 `httpd` 컨테이너 이미지에 사용할 수 있는 모든 이미지 태그를 나열하는 데 사용되는 명령은 무엇입니까?

- `podman search httpd`
- `podman images httpd`
- `podman pull --all-tags=true httpd`
- 태그를 검색하는 데 사용할 수 있는 Podman 명령이 없습니다.

▶ 3. 2.2 태그가 있는 httpd 이미지를 가져오는 2가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- a. podman pull httpd:2.2
- b. podman pull httpd:latest
- c. podman pull quay.io/httpd
- d. podman pull registry.redhat.io/httpd:2.2

▶ 4. 다음 명령을 실행할 때 다운로드되는 컨테이너 이미지는 무엇입니까?

```
podman pull registry.redhat.io/httpd:2.2  
podman pull quay.io/mysql:5.6
```

- a. quay.io/httpd:2.2
registry.redhat.io/mysql:5.6
- b. registry.redhat.io/httpd:2.2
registry.redhat.io/mysql:5.6
- c. registry.redhat.io/httpd:2.2
mysql에 대해 다운로드되는 이미지는 없습니다.
- d. quay.io/httpd:2.2
mysql에 대해 다운로드되는 이미지는 없습니다.

컨테이너 이미지 관리

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- 컨테이너 이미지를 로컬 파일에 저장하고 로드합니다.
- 로컬 스토리지에서 이미지를 삭제합니다.
- 컨테이너에서 새 컨테이너 이미지를 생성하고 이미지 메타데이터를 업데이트합니다.
- 배포를 위해 이미지 태그를 관리합니다.

소개

DevOps 원칙을 준수하는 동시에 다양한 방법으로 이미지 컨테이너를 관리할 수 있습니다. 예를 들어 개발자가 시스템에서 사용자 지정 컨테이너의 테스트를 마친 후 다른 개발자를 위해 이 컨테이너 이미지를 다른 호스트로 전송하거나, 프로덕션 서버로 전송해야 합니다. 이 작업을 수행하는 방법에는 다음 두 가지가 있습니다.

- 컨테이너 이미지를 `.tar` 파일에 저장합니다.
- 컨테이너 이미지를 이미지 레지스트리에 게시(내보내기)합니다.



참고

한 가지 방법은 개발자가 이 장(`podman commit`)에서 나중에 다른 사용자 지정 컨테이너를 만들 수 있습니다. 그러나 다음 장에서는 `Dockerfiles`를 사용한 권장 방법을 설명합니다.

이미지 저장 및 로드

`podman save` 명령을 사용하여 Podman 로컬 스토리지의 기존 이미지를 `.tar` 파일에 저장할 수 있습니다. 생성된 파일은 일반적인 TAR 아카이브가 아닙니다. 이미지 메타데이터를 포함하고 원본 이미지 계층을 유지합니다. 이 파일을 사용하여 Podman은 원본 이미지를 그대로 재생성할 수 있습니다.

`save` 하위 명령의 일반적인 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman save [-o FILE_NAME] IMAGE_NAME[:TAG]
```

Podman은 생성된 이미지를 표준 출력에 바이너리 데이터로 전송합니다. 전송을 방지하려면 `-o` 옵션을 사용합니다.

다음 예제에서는 Red Hat Container Catalog에서 이전에 다운로드한 MySQL 컨테이너 이미지를 `mysql.tar` 파일에 저장합니다.

```
[student@workstation ~]$ sudo podman save \
> -o mysql.tar registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
```

4장 | 컨테이너 이미지 관리

save 하위 명령을 통해 백업 목적으로 생성된 **.tar** 파일을 사용합니다. 컨테이너 이미지를 복원하려면 **podman load** 명령을 사용합니다. 일반적인 명령 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman load [-i FILE_NAME]
```

예를 들어 이 명령은 **mysql.tar** 파일에 저장된 이미지를 로드합니다.

```
[student@workstation ~]$ sudo podman load -i mysql.tar
```

인수로 제공된 **.tar** 파일이 메타데이터가 포함된 컨테이너 이미지가 아닌 경우 **podman load** 명령이 실패합니다.



참고

디스크 공간을 절약하려면 Gzip에 **--compress** 매개 변수를 사용하여 **save** 하위 명령을 통해 생성된 파일을 압축합니다. **load** 하위 명령은 파일을 로컬 스토리지로 가져오기 전에 **gunzip** 명령을 사용합니다.

이미지 삭제

Podman은 현재 컨테이너에서 사용하지 않는 이미지를 포함하여 로컬 스토리지에 다운로드한 모든 이미지를 유지합니다. 그러나 이미지가 오래되면 이후 교체할 수 있습니다.



참고

레지스트리의 이미지가 업데이트되어도 자동으로 업데이트되지 않습니다. 로컬 스토리지에 이미지의 최신 버전이 있도록 하려면 이미지를 제거한 다음 다시 가져와야 합니다.

로컬 스토리지에서 이미지를 삭제하려면 **podman rmi** 명령을 실행합니다. 이 명령의 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman rmi [OPTIONS] IMAGE [IMAGE...]
```

제거를 위해 이미지 이름이나 ID를 사용하여 이미지를 참조할 수 있습니다. Podman은 컨테이너가 사용 중인 이미지를 삭제할 수 없습니다. 이미지를 삭제하기 전에 해당 이미지를 사용하는 컨테이너를 모두 종지하고 제거해야 합니다.

이 문제를 방지하기 위해 **rmi** 하위 명령에는 **--force** 옵션이 있습니다. 이 옵션은 여러 컨테이너에서 이미지를 사용하거나 이러한 컨테이너를 실행 중인 경우에도 이미지를 강제로 제거합니다. Podman은 실제로 제거하기 전에 강제로 제거되는 이미지를 사용하는 컨테이너를 모두 종지하고 제거합니다.

이미지 모두 삭제

컨테이너에서 사용하지 않는 이미지를 모두 삭제하려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman rmi -a
```

이 명령은 로컬 스토리지에서 사용할 수 있는 모든 이미지 ID를 반환하고 제거를 위해 **podman rmi** 명령에 매개 변수로 전달합니다. 사용 중인 이미지는 삭제되지 않습니다. 그러나 사용하지 않는 이미지가 제거되지 않도록 방지하지는 않습니다.

이미지 수정

로그 파일, 임시 파일 또는 컨테이너 사용자 지정을 통해 생성된 기타 아티팩트 없이 일련의 이미지 계층을 깔끔하고 가볍게 생성하려면 **Dockerfile**을 사용하여 모든 컨테이너 이미지를 빌드하는 것이 가장 좋습니다. 하지만 일부 사용자는 **Dockerfile** 없이 컨테이너 이미지를 있는 그대로 제공할 수도 있습니다. 새 이미지를 생성하는 대체 방법으로, 실행 중인 컨테이너를 현재 위치에서 변경하고 해당 계층을 저장하여 새 컨테이너 이미지를 생성합니다. 이 기능은 **podman commit** 명령이 제공합니다.



경고

podman commit 명령이 새 이미지를 생성하는 가장 간단한 방법이지만 이미지 크기 (**commit**는 로그 및 프로세스 ID 파일을 캡처된 계층에 유지함) 및 변경 추적 기능 부족으로 인해 권장되지는 않습니다. **Dockerfile**에서는 운영 체제에서 생성된 일련의 파일 없이 사람이 읽을 수 있는 명령 세트를 사용하여 컨테이너 변경 사항을 사용자 지정하고 구현하는 강력한 메커니즘을 제공합니다.

podman commit 명령의 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman commit [OPTIONS] CONTAINER \
> [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]
```

다음 테이블에는 **podman commit** 명령에 사용할 수 있는 중요한 옵션이 나와 있습니다.

옵션	설명
--author ""	컨테이너 이미지를 만든 사람을 식별합니다.
--message ""	레지스트리에 커밋 메시지를 포함합니다.
--format	이미지의 형식을 선택합니다. 유효한 옵션은 oci 및 docker 입니다.



참고

--message 옵션은 기본 OCI 컨테이너 형식에서 사용할 수 없습니다.

Podman에서 실행 중인 컨테이너의 ID를 찾으려면 **podman ps** 명령을 실행합니다.

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID IMAGE ... NAMES
87bdfcc7c656 mysql ...output omitted... mysql-basic
```

결국 관리자는 이미지를 사용자 지정하고 컨테이너를 원하는 상태로 설정할 수 있습니다. 컨테이너가 시작된 이후 변경, 생성 또는 삭제된 파일을 확인하려면 **diff** 하위 명령을 사용합니다. 이 하위 명령에는 컨테이너 이름 또는 컨테이너 ID만 필요합니다.

```
[student@workstation ~]$ sudo podman diff mysql-basic
C /run
C /run/mysql
A /run/mysql/mysql.pid
A /run/mysql/mysql.sock
A /run/mysql/mysql.sock.lock
A /run/secrets
```

diff 하위 명령은 추가된 파일에는 **A** 태그, 변경된 파일에는 **C** 태그, 삭제된 파일에는 **D** 태그를 지정합니다.



참고

diff 명령은 추가, 변경 또는 삭제된 파일만 컨테이너 파일 시스템에 보고합니다. 실행 중인 컨테이너에 마운트된 파일은 컨테이너 파일 시스템의 일부로 간주되지 않습니다.

실행 중인 컨테이너에 대한 마운트된 파일 및 디렉터리 목록을 검색하려면 **podman inspect** 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman inspect \
> -f "{{range .Mounts}}{{println .Destination}}{{end}}" CONTAINER_NAME/ID
```

이 목록에 있는 파일 또는 디렉터리의 파일은 **podman diff** 명령의 출력에 표시되지 않습니다.

다른 이미지에 변경 사항을 커밋하려면 다음 명령을 실행합니다.

```
[student@workstation ~]$ sudo podman commit mysql-basic mysql-custom
```

태깅 이미지

동일한 소프트웨어 기반의 여러 이미지가 있는 프로젝트를 배포하여 각 이미지에 해당하는 개별 프로젝트를 생성할 수 있지만, 이 방법을 사용하려면 이미지를 관리하고 올바른 위치에 배포하기 위한 추가 유지 관리가 필요합니다.

컨테이너 이미지 레지스트리는 동일한 프로젝트의 여러 릴리스를 구분할 수 있도록 태그를 지원합니다. 예를 들어, 고객은 MySQL 또는 PostgreSQL 데이터베이스에서 실행할 컨테이너 이미지를 사용하고, 이때 태그를 사용하여 컨테이너 이미지에서 사용할 데이터베이스를 구분할 수 있습니다.



참고

일반적으로 태그는 컨테이너 개발자가 동일한 소프트웨어의 여러 버전을 구분하는데 사용됩니다. 릴리스를 쉽게 확인할 수 있도록 여러 개의 태그가 제공됩니다. 공식 MySQL 컨테이너 이미지 웹사이트에서는 버전을 태그 이름(**5.5.16**)으로 사용합니다. 또한 동일한 이미지에 부 버전(예: 5.5)의 두 번째 태그가 있어 특정 버전의 최신 릴리스를 가져와야 하는 필요성을 최소화합니다.

이미지에 태그를 지정하려면 **podman tag** 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman tag [OPTIONS] IMAGE[:TAG] \
> [REGISTRYHOST/]USERNAME/]NAME[:TAG]
```

4장 | 컨테이너 이미지 관리

IMAGE 인수는 Podman에서 관리하는 옵션 태그가 포함된 이미지 이름입니다. 다음 인수는 이미지의 새 대체 이름을 나타냅니다. 태그값이 없는 경우 Podman은 **latest** 태그가 나타내는 최신 버전을 가정합니다. 예를 들어, 다음 명령을 사용하여 이미지에 태그를 지정할 수 있습니다.

```
[student@workstation ~]$ sudo podman tag mysql-custom devops/mysql
```

mysql-custom 옵션은 컨테이너 레지스트리의 이미지 이름에 해당합니다.

다른 태그 이름을 사용하려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman tag mysql-custom devops/mysql:snapshot
```

이미지의 태그 제거

단일 이미지에 **podman tag** 명령을 사용하여 할당된 여러 태그가 포함될 수 있습니다. 태그를 제거하려면 앞에서 설명한 대로 **podman rmi** 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman rmi devops/mysql:snapshot
```



참고

여러 태그가 동일한 이미지를 가리킬 수 있으므로 여러 태그에서 참조하는 이미지를 제거하려면 먼저 각 태그를 개별적으로 제거합니다.

이미지 태그 지정의 모범 사례

태그를 지정하지 않으면 Podman에서 이미지를 최신 빌드로 간주하므로 **latest** 태그를 자동으로 추가합니다. 그러나 각 프로젝트에서 태그를 사용하는 방법에 따라 이렇게 동작하지 않을 수도 있습니다. 예를 들어, 많은 오픈소스 프로젝트에서는 최신 빌드가 아닌 최신 릴리스와 일치하도록 **최신** 태그를 간주합니다.

더욱이, 특정 프로젝트 버전의 최신 릴리스를 기억해야 하는 필요성을 최소화하기 위해 여러 개의 태그가 제공됩니다. 따라서 프로젝트 버전 릴리스(예: **2.1.10**)가 있는 경우, **2.1**이라는 또 다른 태그를 생성하여 **2.1.10** 릴리스의 동일한 이미지를 가리킬 수 있습니다. 이렇게 하면 레지스트리에서 이미지를 가져오는 작업이 간소화됩니다.

레지스트리에 이미지 게시

이미지를 레지스트리에 게시하려면 Podman의 로컬 스토리지에 있어야 하며, 식별하기 위해 태그를 지정해야 합니다. 이미지를 레지스트리로 내보내기 위한 **push** 하위 명령의 구문은 다음과 같습니다.

```
[student@workstation ~]$ sudo podman push [OPTIONS] IMAGE [DESTINATION]
```

예를 들어, **bitnami/nginx** 이미지를 리포지토리에 내보내려면 다음 명령을 사용합니다.

```
[student@workstation ~]$ sudo podman push quay.io/bitnami/nginx
```



참조

Podman 사이트

<https://podman.io/>

▶ 연습 가이드

사용자 지정 Apache 컨테이너 이미지 생성

이 안내에 따른 연습에서는 `podman commit` 명령을 사용하여 사용자 지정 Apache 컨테이너 이미지를 만듭니다.

결과

사용자 지정 컨테이너 이미지를 만들 수 있습니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab image-operations start
```

- ▶ 1. Quay.io 계정에 로그인하고 `quay.io/redhattraining/httpd-parent`에서 사용할 수 있는 이미지를 사용하여 컨테이너를 시작합니다. `-p` 옵션을 통해 리디렉션 포트를 지정할 수 있습니다. 이 경우, Podman은 호스트의 TCP 포트 8180에 수신되는 요청을 컨테이너의 TCP 포트 80으로 전달합니다.

```
[student@workstation ~]$ sudo podman login quay.io
Username: your_quay_username
Password: your_quay_password
Login Succeeded!
[student@workstation ~]$ sudo podman run -d --name official-httpd \
> -p 8180:80 redhattraining/httpd-parent
...output omitted...
Writing manifest to image destination
Storing signatures
3a6baecaff2b4e8c53b026e04847dda5976b773ade1a3a712b1431d60ac5915d
```

마지막 출력 행은 위에 표시된 마지막 행과 다릅니다. 처음 12개 문자를 확인합니다.

- ▶ 2. `official-httpd` 컨테이너에 HTML 페이지를 만듭니다.

2.1. `podman exec` 명령을 사용하여 컨테이너의 쉘에 액세스하고 HTML 페이지를 생성합니다.

```
[student@workstation ~]$ sudo podman exec -it official-httpd /bin/bash
bash-4.4# echo "DO180 Page" > /var/www/html/do180.html
```

2.2. 컨테이너를 종료합니다.

```
bash-4.4# exit
```

2.3. `curl` 명령을 사용하여 `workstation` VM에서 HTML 파일에 연결할 수 있는지 확인합니다.

```
[student@workstation ~]$ curl 127.0.0.1:8180/do180.html
```

다음 출력이 표시되어야 합니다.

D0180 Page

- ▶ 3. `podman diff` 명령을 사용하여 컨테이너에서 이미지와 컨테이너가 생성한 새 계층의 차이점을 검토합니다.

```
[student@workstation ~]$ sudo podman diff official-httdp
C /etc
C /root
A /root/.bash_history
...output omitted...
C /tmp
C /var
C /var/log
C /var/log/httdp
A /var/log/httdp/access_log
A /var/log/httdp/error_log
C /var/www
C /var/www/html
A /var/www/html/do180.html
```



참고

종종, 웹 서버 컨테이너 이미지는 `/var/www/html` 디렉터리를 볼륨으로 레이블 지정합니다. 이 경우, 이 디렉터리에 추가된 모든 파일은 컨테이너 파일 시스템의 일부로 간주되지 않으며 파일은 `git diff` 명령의 출력에 표시되지 않습니다.

`redhattraining/httdp-parent` 컨테이너 이미지는 `/var/www/html` 디렉터리를 볼륨으로 레이블 지정하지 않습니다. 결과적으로, `/var/www/html/do180.html` 파일은 기본 컨테이너 파일 시스템에 대한 변경 사항으로 간주됩니다.

- ▶ 4. 컨테이너를 실행하여 만든 변경 사항으로 새 이미지를 만듭니다.

- 4.1. `official-httdp` 컨테이너를 중지합니다.

```
[student@workstation ~]$ sudo podman stop official-httdp
3a6baecaff2b4e8c53b026e04847ddda5976b773ade1a3a712b1431d60ac5915d
```

- 4.2. 새 컨테이너 이미지에 대한 변경 사항을 새 이름으로 커밋합니다. 귀하의 이름을 변경 사항의 작성자로 사용합니다.

```
[student@workstation ~]$ sudo podman commit \
> -a 'Your Name' official-httdp do180-custom-httdp
Getting image source signatures
Skipping fetch of repeat blob sha256:071d8bd765171080d01682844524be57ac9883e...
...output omitted...
Copying blob sha256:1e19be875ce6f5b9dece378755eb9df96ee205abfb4f165c797f59a9...
15.00 KB / 15.00 KB [=====] 0s
```

4장 | 컨테이너 이미지 관리

```
Copying config sha256:8049dc2e7d0a0b1a70fc0268ad236399d9f5fb686ad4e31c7482cc...
2.99 KB / 2.99 KB [=====] 0s
Writing manifest to image destination
Storing signatures
31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb399bef2a23
```

4.3. 사용 가능한 컨테이너 이미지를 나열합니다.

```
[student@workstation ~]$ sudo podman images
```

예상되는 출력은 다음과 유사합니다.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/do180-custom-httdp	latest	31c3ac78e9d4
quay.io/redhattraining/httpd-parent	latest	2cc07fbb5000

이미지 ID는 해시의 첫 12자와 일치합니다. 최신 이미지가 맨 위에 나열됩니다.

▶ 5. 저장된 컨테이너 이미지를 컨테이너 레지스트리에 게시합니다.

5.1. 강의실 환경 구성 파일을 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

5.2. 레지스트리 호스트 이름 및 태그를 사용하여 이미지에 태그를 지정하려면 다음 명령을 실행합니다.

```
[student@workstation ~]$ sudo podman tag do180-custom-httdp \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httdp:v1.0
```

5.3. `podman images` 명령을 실행하여 새 이름이 캐시에 추가되었는지 확인합니다.

```
[student@workstation ~]$ sudo podman images
```

REPOSITORY	TAG	IMAGE ID	...
localhost/do180-custom-httdp	latest	31c3ac78e9d4	...
quay.io/\${RHT_OCP4_QUAY_USER}/do180-custom-httdp	v1.0	31c3ac78e9d4	...
quay.io/redhattraining/httpd-parent	latest	2cc07fbb5000	...

5.4. 이미지를 Quay.io 레지스트리에 게시합니다.

```
[student@workstation ~]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httdp:v1.0
Getting image source signatures
Copying blob sha256:071d8bd765171080d01682844524be57ac9883e53079b6ac66707e19...
200.44 MB / 200.44 MB [=====] 1m38s
...output omitted...
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3...
2.99 KB / 2.99 KB [=====] 0s
Writing manifest to image destination
```

4장 | 컨테이너 이미지 관리

```
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3...
  0 B / 2.99 KB [-----] 0s
Writing manifest to image destination
Storing signatures
```

**참고**

do180-custom-httd 이미지를 내보내면 Quay.io 계정에 개인 리포지토리가 생성됩니다. 현재 Quay.io 무료 플랜에서는 개인 리포지토리가 허용되지 않습니다. 이미지를 내보내기 전에 공용 리포지토리를 만들거나 나중에 리포지토리를 공용 리포지토리로 변경할 수 있습니다.

- Quay.io에서 이미지를 사용할 수 있는지 확인합니다. **podman search** 명령을 실행하려면 이미지를 Quay.io로 인덱싱해야 합니다. 이 작업을 수행하는 데 몇 시간이 걸릴 수 있으므로 **podman pull** 명령을 사용하여 이미지를 가져옵니다. 이를 통해 이미지에 대한 가용성을 증명합니다.

```
[student@workstation ~]$ sudo podman pull \
> -q quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httd:v1.0
31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3
```

▶ 6. 새로 게시된 이미지에서 컨테이너를 만듭니다.

podman run 명령을 사용하여 새 컨테이너를 시작합니다. 기본 이미지로 `your_quay_username/do180-custom-httd:v1.0`을 사용합니다.

```
[student@workstation ~]$ sudo podman run -d --name test-httd -p 8280:80 \
> ${RHT_OCP4_QUAY_USER}/do180-custom-httd:v1.0
c0f04e906bb12bd0e514cbd0e581d2746e04e44a468dfbc85bc29ffcc5acd16c
```

▶ 7. **curl** 명령을 사용하여 HTML 페이지에 액세스합니다. 포트 8280을 사용해야 합니다.

이전 단계에서 생성한 HTML 페이지가 표시됩니다.

```
[student@workstation ~]$ curl http://localhost:8280/do180.html
D0180 Page
```

완료

`workstation`에서 `lab image-operations finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab image-operations finish
```

이로써 안내에 따른 연습이 완료됩니다.

▶ 랩

이미지 관리

수행 체크리스트

이 랩에서는 컨테이너 이미지를 만들고 관리합니다.

결과

사용자 지정 컨테이너 이미지를 만들고 컨테이너 이미지를 관리할 수 있습니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab image-review start
```

1. `podman pull` 명령을 사용하여 `quay.io/redhattraining/nginx:1.17` 컨테이너 이미지를 다운로드합니다. 이 이미지는 `docker.io/library/nginx:1.17`에서 사용할 수 있는 컨테이너 이미지의 사본입니다.
이미지를 제대로 다운로드했는지 확인합니다.
2. 다음 목록에 나열된 사양에 따라 Nginx 이미지를 사용하여 새 컨테이너를 시작합니다.
 - **이름:** `official-nginx`
 - **데몬으로 실행:** `yes`
 - **컨테이너 이미지:** `nginx`
 - **포트 전달:** from host port 8080 to container port 80.
3. `podman exec` 명령을 사용하여 컨테이너에 로그인합니다. `index.html` 파일의 내용을 `D0180`으로 바꿉니다. 웹 서버 디렉터리는 `/usr/share/nginx/html`에 있습니다.
파일이 업데이트되면 컨테이너를 종료하고, `curl` 명령을 사용하여 웹 페이지에 액세스합니다.
4. 실행 중인 컨테이너를 중지하고 변경을 커밋하여 새 컨테이너 이미지를 만듭니다. 새 이미지에 이름 `do180/mynginx`, 태그 `v1.0-SNAPSHOT`을 지정합니다. 다음 사양을 사용합니다.
 - 이미지 이름: `do180/mynginx`
 - 이미지 태그: `v1.0-SNAPSHOT`
 - 작성자 이름: 귀하의 이름
5. 다음 목록에 나열된 사양에 따라 업데이트된 Nginx 이미지를 사용하여 새 컨테이너를 시작합니다.
 - **이름:** `official-nginx-dev`
 - **데몬으로 실행:** `yes`
 - **컨테이너 이미지:** `do180/mynginx:v1.0-SNAPSHOT`

4장 | 컨테이너 이미지 관리

- **포트 전달**: from host port 8080 to container port 80.
6. `podman exec` 명령을 통해 컨테이너에 로그인하여 최종 변경 사항을 도입합니다. `/usr/share/nginx/html/index.html` 파일의 내용을 `DO180 Page`로 바꿉니다.
파일이 업데이트되면 컨테이너를 종료하고, `curl` 명령을 사용하여 변경 사항을 확인합니다.
7. 실행 중인 컨테이너를 중지하고 변경 사항을 커밋하여 최종 컨테이너 이미지를 생성합니다. 새 이미지의 이름을 `do180/mynginx`로, 태그의 이름을 `v1.0`으로 지정합니다. 다음 사양을 사용합니다.
- 이미지 이름: `do180/mynginx`
 - 이미지 태그: `v1.0`
 - 작성자 이름: 귀하의 이름
8. 로컬 이미지 스토리지에서 개발 이미지 `do180/mynginx:v1.0-SNAPSHOT`을 제거합니다.
9. `do180/mynginx:v1.0` 태그가 지정된 이미지를 사용하여 다음 사양으로 새 컨테이너를 만듭니다.
- 컨테이너 이름: `my-nginx`
 - 데몬으로 실행: yes
 - 컨테이너 이미지: `do180/mynginx:v1.0`
 - 포트 전달: from host port 8280 to container port 80

`workstation`에서 `curl` 명령을 사용하여 포트 8280에서 액세스할 수 있는 웹 서버에 액세스합니다.

평가

`workstation` 시스템에서 `lab image-review grade` 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab image-review grade
```

완료

`workstation`에서 `lab image-review finish` 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab image-review finish
```

이로써 랩이 완료됩니다.

▶ 솔루션

이미지 관리

수행 체크리스트

이 랩에서는 컨테이너 이미지를 만들고 관리합니다.

결과

사용자 지정 컨테이너 이미지를 만들고 컨테이너 이미지를 관리할 수 있습니다.

시작하기 전에

workstation에서 student 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab image-review start
```

- podman pull 명령을 사용하여 `quay.io/redhattraining/nginx:1.17` 컨테이너 이미지를 다운로드합니다. 이 이미지는 `docker.io/library/nginx:1.17`에서 사용할 수 있는 컨테이너 이미지의 사본입니다.

이미지를 제대로 다운로드했는지 확인합니다.

- podman pull 명령을 사용하여 Nginx 컨테이너 이미지를 가져옵니다.

```
[student@workstation ~]$ sudo podman pull quay.io/redhattraining/nginx:1.17
Trying to pull Trying to pull quay.io/redhattraining/nginx:1.17...
...output omitted...
Storing signatures
9beeba249f3ee158d3e495a6ac25c5667ae2de8a43ac2a8bfd2bf687a58c06c9
```

- podman images 명령을 실행하여 컨테이너 이미지가 로컬 시스템에 있는지 확인합니다.

```
[student@workstation ~]$ sudo podman images
```

이 명령은 다음과 유사한 출력을 생성합니다.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
quay.io/redhattraining/nginx	1.17	9beeba249f3e	6 months ago	131MB

- 다음 목록에 나열된 사양에 따라 Nginx 이미지를 사용하여 새 컨테이너를 시작합니다.

- 이름:** official-nginx
- 데몬으로 실행:** yes
- 컨테이너 이미지:** nginx
- 포트 전달:** from host port 8080 to container port 80.

4장 | 컨테이너 이미지 관리

- 2.1. `workstation`에서 `podman run` 명령을 사용하여 `official-nginx`라는 컨테이너를 생성합니다.

```
[student@workstation ~]$ sudo podman run --name official-nginx \
> -d -p 8080:80 quay.io/redhattraining/nginx:1.17
b9d5739af239914b371025c38340352ac1657358561e7ebbd5472dfd5ff97788
```

3. `podman exec` 명령을 사용하여 컨테이너에 로그인합니다. `index.html` 파일의 내용을 `D0180`으로 바꿉니다. 웹 서버 디렉터리는 `/usr/share/nginx/html`에 있습니다.

파일이 업데이트되면 컨테이너를 종료하고, `curl` 명령을 사용하여 웹 페이지에 액세스합니다.

- 3.1. `podman exec` 명령을 사용하여 컨테이너에 로그인합니다.

```
[student@workstation ~]$ sudo podman exec -it official-nginx /bin/bash
root@b9d5739af239:/#
```

- 3.2. `/usr/share/nginx/html`에 있는 `index.html` 파일을 업데이트합니다. 파일은 `D0180`이어야 합니다.

```
root@b9d5739af239:/# echo 'D0180' > /usr/share/nginx/html/index.html
```

- 3.3. 컨테이너를 종료합니다.

```
root@b9d5739af239:/# exit
```

- 3.4. `curl` 명령을 사용하여 `index.html` 파일이 업데이트되었는지 확인합니다.

```
[student@workstation ~]$ curl 127.0.0.1:8080
D0180
```

4. 실행 중인 컨테이너를 중지하고 변경을 커밋하여 새 컨테이너 이미지를 만듭니다. 새 이미지에 이름 `do180/mynginx`, 태그 `v1.0-SNAPSHOT`을 지정합니다. 다음 사양을 사용합니다.

- 이미지 이름: `do180/mynginx`
- 이미지 태그: `v1.0-SNAPSHOT`
- 작성자 이름: 귀하의 이름

- 4.1. `sudo podman stop` 명령을 사용하여 `official-nginx` 컨테이너를 중지합니다.

```
[student@workstation ~]$ sudo podman stop official-nginx
b9d5739af239914b371025c38340352ac1657358561e7ebbd5472dfd5ff97788
```

- 4.2. 새 컨테이너 이미지에 변경을 커밋합니다. 귀하의 이름을 변경 사항의 작성자로 사용합니다.

```
[student@workstation ~]$ sudo podman commit -a 'Your Name' \
> official-nginx do180/mynginx:v1.0-SNAPSHOT
Getting image source signatures
...output omitted...
Storing signatures
d6d10f52e258e4e88c181a56c51637789424e9261b208338404e82a26c960751
```

4.3. 새로 생성한 이미지를 찾는데 사용할 수 있는 컨테이너 이미지를 나열합니다.

```
[student@workstation ~]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED     ...
localhost/do180/mynginx    v1.0-SNAPSHOT  d6d10f52e258  30 seconds ago  ...
quay.io/redhattraining/nginx  1.17      9beeba249f3e  6 months ago   ...
```

5. 다음 목록에 나열된 사양에 따라 업데이트된 Nginx 이미지를 사용하여 새 컨테이너를 시작합니다.

- **이름:** official-nginx-dev
- **데몬으로 실행:** yes
- **컨테이너 이미지:** do180/mynginx:v1.0-SNAPSHOT
- **포트 전달:** from host port 8080 to container port 80.

5.1. workstation에서 podman run 명령을 사용하여 official-nginx-dev라는 컨테이너를 생성합니다.

```
[student@workstation ~]$ sudo podman run --name official-nginx-dev \
> -d -p 8080:80 do180/mynginx:v1.0-SNAPSHOT
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

6. podman exec 명령을 통해 컨테이너에 로그인하여 최종 변경 사항을 도입합니다. /usr/share/nginx/html/index.html 파일의 내용을 D0180 Page로 바꿉니다.

파일이 업데이트되면 컨테이너를 종료하고, curl 명령을 사용하여 변경 사항을 확인합니다.

6.1. podman exec 명령을 사용하여 컨테이너에 로그인합니다.

```
[student@workstation ~]$ sudo podman exec -it official-nginx-dev /bin/bash
root@cfa21f02a77d:/#
```

6.2. /usr/share/nginx/html에 있는 index.html 파일을 업데이트합니다. 파일은 D0180 Page이어야 합니다.

```
root@cfa21f02a77d:/# echo 'D0180 Page' > /usr/share/nginx/html/index.html
```

6.3. 컨테이너를 종료합니다.

```
root@cfa21f02a77d:/# exit
```

6.4. curl 명령을 사용하여 index.html 파일이 업데이트되었는지 확인합니다.

```
[student@workstation ~]$ curl 127.0.0.1:8080
D0180 Page
```

7. 실행 중인 컨테이너를 중지하고 변경 사항을 커밋하여 최종 컨테이너 이미지를 생성합니다. 새 이미지의 이름을 **do180/mynginx**로, 태그의 이름을 **v1.0**으로 지정합니다. 다음 사양을 사용합니다.

- 이미지 이름: **do180/mynginx**
- 이미지 태그: **v1.0**
- 작성자 이름: 귀하의 이름

7.1. **sudo podman stop** 명령을 사용하여 **official-nginx-dev** 컨테이너를 중지합니다.

```
[student@workstation ~]$ sudo podman stop official-nginx-dev
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

7.2. 새 컨테이너 이미지에 변경을 커밋합니다. 귀하의 이름을 변경 사항의 작성자로 사용합니다.

```
[student@workstation ~]$ sudo podman commit -a 'Your Name' \
> official-nginx-dev do180/mynginx:v1.0
Getting image source signatures
...output omitted...
Storing signatures
90915976c33de534e06778a74d2a8969c25ef5f8f58c0c1ab7aeaac19abd18af
```

7.3. 새로 만든 이미지를 찾으려면 사용 가능한 컨테이너 이미지를 나열합니다.

```
[student@workstation ~]$ sudo podman images
REPOSITORY          TAG      IMAGE ID   CREATED     ...
localhost/do180/mynginx    v1.0    90915976c33d  6 seconds ago ...
localhost/do180/mynginx    v1.0-SNAPSHOT  d6d10f52e258  8 minutes ago ...
quay.io/redhattraining/nginx  1.17    9beeba249f3e  6 months ago ...
```

8. 로컬 이미지 스토리지에서 개발 이미지 **do180/mynginx:v1.0-SNAPSHOT**을 제거합니다.

8.1. 중지되었지만 **official-nginx-dev**가 여전히 있습니다. **podman ps** 명령에 **-a** 플래그를 사용하여 컨테이너를 표시합니다.

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
cfa21f02a77d  official-nginx-dev  Exited (0) 9 minutes ago
b9d5739af239  official-nginx      Exited (0) 12 minutes ago
```

8.2. **podman rm** 명령을 사용하여 컨테이너를 제거합니다.

```
[student@workstation ~]$ sudo podman rm official-nginx-dev
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

8.3. 동일한 **podman ps** 명령을 다시 제출하여 컨테이너가 삭제되었는지 확인합니다.

4장 | 컨테이너 이미지 관리

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
b9d5739af239    official-nginx      Exited (0) 12 minutes ago
```

- 8.4. `sudo podman rmi` 명령을 사용하여 `do180/mynginx:v1.0-SNAPSHOT` 이미지를 제거합니다.

```
[student@workstation ~]$ sudo podman rmi do180/mynginx:v1.0-SNAPSHOT
Untagged: localhost/do180/mynginx:v1.0-SNAPSHOT
```

- 8.5. `podman images` 명령을 통해 모든 이미지를 나열하여 이미지가 더 이상 없음을 확인합니다.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/do180/mynginx	v1.0	90915976c33d	5 minutes ago	131MB
quay.io/redhattraining/nginx	1.17	9beeba249f3e	6 months ago	131MB

9. `do180/mynginx:v1.0` 태그가 지정된 이미지를 사용하여 다음 사양으로 새 컨테이너를 만듭니다.

- 컨테이너 이름: `my-nginx`
- 데몬으로 실행: `yes`
- 컨테이너 이미지: `do180/mynginx:v1.0`
- 포트 전달: from host port 8280 to container port 80

`workstation`에서 `curl` 명령을 사용하여 포트 8280에서 액세스할 수 있는 웹 서버에 액세스합니다.

- 9.1. `sudo podman run` 명령을 사용하여 사양에 따라 `my-nginx` 컨테이너를 만듭니다.

```
[student@workstation ~]$ sudo podman run -d --name my-nginx \
> -p 8280:80 do180/mynginx:v1.0
51958c8ec8d2613bd26f85194c66ca96c95d23b82c43b23b0f0fb9fded74da20
```

- 9.2. `curl` 명령을 사용하여 `index.html` 페이지를 사용할 수 있으며 사용자 지정 컨텐츠를 반환하는지 확인합니다.

```
[student@workstation ~]$ curl 127.0.0.1:8280
DO180 Page
```

평가

`workstation` 시스템에서 `lab image-review grade` 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab image-review grade
```

완료

workstation에서 **lab image-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab image-review finish
```

이로써 랩이 완료됩니다.

요약

이 장에서 학습한 내용:

- Red Hat Container Catalog는 `registry.redhat.io`에서 테스트 및 인증을 거친 이미지를 제공합니다.
- Podman은 원격 컨테이너 레지스트리와 상호 작용하여 컨테이너 이미지를 검색하고, 가져오고, 내보낼 수 있습니다.
- 이미지 태그는 컨테이너 이미지의 여러 릴리스를 지원하는 메커니즘입니다.
- Podman은 로컬 스토리지에서 및 압축 파일로 컨테이너 이미지를 관리하는 명령을 제공합니다.
- `podman commit`을 사용하여 컨테이너에서 이미지를 생성합니다.

5장

사용자 지정 컨테이너 이미지 생성

목적

Dockerfile을 설계 및 코딩하여 사용자 지정 컨테이너 이미지를 빌드합니다.

목표

- 사용자 지정 컨테이너 이미지를 만드는 접근법을 설명합니다.
- 공통 Dockerfile 명령을 사용하여 컨테이너 이미지를 만듭니다.

섹션

- 사용자 지정 컨테이너 이미지 설계 및 퀴즈
- Dockerfiles를 사용하여 사용자 지정 컨테이너 이미지 빌드(안내에 따른 연습)

랩

사용자 지정 컨테이너 이미지 생성

사용자 지정 컨테이너 이미지 설계

목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- 사용자 지정 컨테이너 이미지를 만드는 접근법을 설명합니다.
- 기존 Dockerfile을 찾아 사용자 지정 컨테이너 이미지를 만드는 시작 지점으로 사용합니다.
- Red Hat 레지스트리에서 컨테이너 이미지를 설계할 때 RHSCL(Red Hat Software Collections Library)에서 수행하는 역할을 정의합니다.
- Dockerfile에 대한 S2I(Source-to-Image) 대안을 설명합니다.

기존 Dockerfile 재사용

지금까지 컨테이너 이미지를 생성하는 한 가지 방법을 설명했습니다. 컨테이너를 생성하고 컨테이너에서 실행할 애플리케이션의 요구 사항에 맞게 수정한 다음, 이미지에 대한 변경 사항을 커밋하면 됩니다. 이 옵션은 간단하지만, 매우 구체적인 변경 사항을 사용하거나 테스트하는 경우에만 적합합니다. 유지 관리, 빌드 자동화, 반복성 등의 소프트웨어 모범 사례를 따르지 않습니다.

Dockerfile은 컨테이너 이미지를 생성하고 이러한 제한 사항을 해결하는 또 다른 옵션입니다. Dockerfile은 공유, 버전 제어, 재사용 및 확장이 용이합니다.

Dockerfile을 사용하면 다른 이미지(상위 이미지)에서 이미지(하위 이미지)를 쉽게 확장할 수도 있습니다. 하위 이미지는 상위 이미지의 모든 내용과 모든 변경 사항 및 추가 정보를 통합하여 생성합니다.

이미지를 공유하고 재사용하기 위해 **Quay.io** 등의 퍼블릭 이미지 레지스트리에서 많은 인기 애플리케이션, 언어 및 프레임워크를 이미 사용할 수 있습니다. 컨테이너 권장 사례를 따르도록 애플리케이션 구성과 사용자 지정하는 것은 쉽지 않으므로 입증된 상위 이미지부터 시작하면 일반적으로 수행할 작업이 많이 줄어듭니다.

고품질 상위 이미지를 사용하면, 특히 작성자가 버그 수정 및 보안 문제를 처리하기 위해 상위 이미지를 계속 업데이트하는 경우 유지 관리가 향상됩니다.

기존 컨테이너 이미지에서 하위 이미지를 빌드하기 위한 Dockerfile을 생성하는 일반적인 시나리오에는 다음 단계가 포함됩니다.

- 데이터베이스 커넥터와 같은 새 런타임 라이브러리를 추가합니다.
- SSL 인증서, 인증 프로바이더 등 조직 전체의 사용자 지정을 포함합니다.
- 여러 컨테이너 이미지에서 다양한 애플리케이션에 대해 단일 이미지 계층으로 공유할 내부 라이브러리를 추가합니다.

기존 Dockerfile을 변경하여 새 이미지를 만들어도 다른 시나리오에서 합리적인 접근법이 될 수 있습니다. 예를 들면 다음과 같습니다.

- 사용하지 않는 자료(예: `/usr/share/doc`에 있는 도움말 페이지 또는 문서)를 제거하여 컨테이너 이미지를 자릅니다.
- 상위 이미지 또는 일부 포함된 소프트웨어 패키지를 특정 릴리스로 잠금을 수행하여 향후 소프트웨어 업데이트와 관련된 위험을 줄입니다.

5장 | 사용자 지정 컨테이너 이미지 생성

상위 이미지로 사용하거나 Dockerfile을 변경하는 데 사용할 컨테이너 이미지의 두 가지 소스는 Docker Hub와 RHSC(Red Hat Software Collections Library)입니다.

Red Hat Software Collections Library 사용

RHSC(Red Hat Software Collections Library) 또는 간단히 Software Collections는 표준 RHEL 릴리스 스케줄에 맞지 않는 최신 개발 툴을 사용해야 하는 개발자를 위한 Red Hat 솔루션입니다.

RHEL(Red Hat Enterprise Linux)은 회사 애플리케이션을 위한 안정적인 환경을 제공합니다. 이를 위해 RHEL에서는 API 및 구성 파일의 형식이 변경되지 않도록 업스트림 패키지의 주요 릴리스를 동일한 수준으로 유지해야 합니다. 보안 및 성능 수정 사항은 이후의 업스트림 릴리스에서 백포팅되지만, 이전 버전과 호환되지 않는 새 기능은 백포팅되지 않습니다.

RHSC를 사용하면 소프트웨어 개발자가 RHEL에 영향을 주지 않고 최신 버전을 사용할 수 있습니다. RHSC 패키지가 기본 RHEL 패키지를 교체하거나 해당 패키지와 충돌하지 않기 때문입니다. 기본 RHEL 패키지 및 RHSC 패키지는 나란히 설치됩니다.



참고

모든 RHEL 구독자는 RHSC에 액세스할 수 있습니다. 특정 사용자 또는 애플리케이션 환경(예: 이름이 `rh-mysql157`로 지정된 MySQL 5.7)에 대한 특정 소프트웨어 컬렉션을 활성화하려면 RHSC 소프트웨어 Yum 리포지토리를 활성화하고 몇 가지 간단한 단계를 따릅니다.

Red Hat Software Collections Library에서 Dockerfile 찾기

RHSC은 RHEL Atomic Host 및 OpenShift Container Platform 고객이 사용할 수 있도록 Red Hat 이미지 레지스터리에서 제공하는 컨테이너 이미지의 주요 소스입니다.

Red Hat은 RHSC 리포지토리에서 사용할 수 있는 `rhscl-dockerfiles` 패키지에 RHSC Dockerfile 및 관련 소스를 제공합니다. 커뮤니티 사용자는 <https://github.com/sclorg?q=-container>에서 CentOS 기반의 동급 컨테이너 이미지에 대한 Dockerfile을 가져올 수 있습니다.



참고

많은 RHSC 컨테이너 이미지에는 OpenShift Container Platform 기능으로 가장 잘 알려진 S2I(Source-to-Image) 지원이 포함됩니다. S2I를 지원해도 Docker에서 이러한 컨테이너 이미지를 사용하는 기능에는 영향을 주지 않습니다.

RHCC(Red Hat Container Catalog)의 컨테이너 이미지

미션 크리티컬 애플리케이션에는 신뢰할 수 있는 컨테이너가 필요합니다. Red Hat Container Catalog은 RHEL(Red Hat Enterprise Linux) 버전 및 관련 시스템을 토대로 빌드된 신뢰할 수 있고 테스트, 인증 및 조정된 컨테이너 이미지 컬렉션의 리포지토리입니다. RHCC를 통해 사용할 수 있는 컨테이너 이미지는 품질 보증 프로세스를 거쳤습니다. 모든 구성 요소는 알려진 보안 취약성을 피하기 위해 Red Hat에서 다시 빌드되었습니다. 새 이미지를 아직 사용할 수 없을 때라도 필요한 소프트웨어 버전을 포함하도록 정기적으로 업그레이드됩니다. RHCC를 사용하면 이미지를 찾아보고 검색할 수 있으며 버전, 콘텐츠, 사용법 등 각 이미지에 대한 정보에 액세스할 수 있습니다.

Quay.io를 사용하여 이미지 검색

Quay.io는 팀 공동 작업에 최적화된 CoreOS의 고급 컨테이너 리포지토리입니다. <https://quay.io/search>를 사용하여 컨테이너 이미지를 검색할 수 있습니다.

이미지 이름을 클릭하면 이미지의 모든 기존 태그 액세스, 이미지 가져오기 명령을 비롯하여 이미지 정보 페이지에 액세스할 수 있습니다.

Docker Hub에서 Dockerfile 찾기

누구나 Docker Hub 계정을 만들고 여기에 컨테이너 이미지를 게시할 수 있습니다. 품질 및 보안에 대한 일반적인 보증은 없습니다. Docker Hub의 이미지는 전문적으로 지원되는 실험에서 일회성 실험까지 다양합니다. 각 이미지는 개별적으로 평가해야 합니다.

이미지를 검색한 후 설명서 페이지에서 Dockerfile에 대한 링크를 제공할 수 있습니다. 예를 들어, **mysql** 검색 시 첫 번째 결과는 MySQL official(MySQL 공식) 이미지에 대한 설명서 페이지(https://hub.docker.com/_/mysql)입니다.

해당 페이지에서 **5.6/Dockerfile** 이미지의 링크는 Docker 커뮤니티 자동 빌드 시스템에서 빌드한 이미지에 대해 **Dockerfiles**를 호스팅하는 **docker-library** GitHub 프로젝트를 가리킵니다.

Docker Hub MySQL 5.6 **Dockerfile** 트리의 직접 URL은 <https://github.com/docker-library/mysql/blob/master/5.6>입니다.

OpenShift Source-to-Image 툴을 사용하는 방법 설명

S2I(Source-to-Image)는 Dockerfile을 사용하여 새 컨테이너 이미지를 만드는 방법에 대한 대안을 제공하며, OpenShift의 기능 또는 독립 실행형 **s2i** 유틸리티로 사용할 수 있습니다. S2I를 사용하면 개발자는 Dockerfile 구문을 익히고 **yum**과 같은 운영 체제 명령을 사용하는 대신, 일반 툴을 사용하여 작업할 수 있으며, 일반적으로 S2I는 계층이 적은 가벼운 이미지를 만듭니다.

S2I는 다음 프로세스를 사용하여 애플리케이션에 대한 사용자 지정 컨테이너 이미지를 빌드합니다.

1. 프로그래밍 언어 런타임 및 필수 개발 툴(예: 컴파일러, 패키지 관리자)이 포함된 빌더 이미라는 기본 컨테이너 이미지에서 컨테이너를 시작합니다.
2. 일반적으로 Git 서버에서 애플리케이션 소스 코드를 가져와서 컨테이너로 보냅니다.
3. 컨테이너 내부에 애플리케이션 바이너리 파일을 빌드합니다.
4. 일부를 정리한 후 컨테이너를 프로그래밍 언어 런타임 및 애플리케이션 바이너리가 포함된 새 컨테이너 이미지로 저장합니다.

빌더 이미지는 일반 컨테이너 이미지로, 표준 디렉터리 구조를 따르고 S2I 프로세스 중에 호출되는 스크립트를 제공합니다. 이러한 빌더 이미지는 대부분 S2I 프로세스 외부에서 Dockerfile의 기본 이미지로 사용할 수도 있습니다.

s2i 명령은 Docker 전용 환경의 OpenShift 외부에서 S2I 프로세스를 실행하는 데 사용됩니다. source-to-image RPM 패키지를 통해 RHEL 시스템에 설치할 수 있으며, GitHub의 S2I 프로젝트에서 사용할 수 있는 설치 프로그램을 통해 Windows, Mac OS 등의 기타 플랫폼에 설치할 수 있습니다.



참조

Red Hat Software Collections Library(RHSCl)

<https://access.redhat.com/documentation/en/red-hat-software-collections/>

Red Hat Container Catalog(RHCC)

<https://access.redhat.com/containers/>

GitHub의 RHSCl Dockerfile

<https://github.com/sclorg?q=-container>

Red Hat Software Collections 컨테이너 이미지 사용

<https://access.redhat.com/articles/1752723>

Quay.io

<https://quay.io/search>

Docker Hub

<https://hub.docker.com/>

Docker 라이브러리 GitHub 프로젝트

<https://github.com/docker-library>

S2I GitHub 프로젝트

<https://github.com/openshift/source-to-image>

▶ 퀴즈

컨테이너 이미지 설계에 대한 접근법

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너 커뮤니티에서 권장하는 컨테이너 이미지 생성 방법은 무엇입니까? (한 개를 선택하십시오.)

- a. 기본 OS 컨테이너 내부에서 명령을 실행하고 컨테이너를 커밋한 다음, 새 컨테이너 이미지로 저장하거나 내보냅니다.
- b. Dockerfile의 명령을 실행하고 생성된 컨테이너 이미지를 이미지 레지스트리로 내보냅니다.
- c. tar 파일에서 컨테이너 이미지 계층을 수동으로 만듭니다.
- d. `podman build` 명령을 실행하여 컨테이너 이미지 설명을 YAML 형식으로 처리합니다.

▶ 2. 독립 실행형 S2I 프로세스를 Dockerfile의 대안으로 사용하여 얻을 수 있는 두 가지 이점은 무엇입니까? (두 개를 선택하십시오.)

- a. 기본 Podman 설치 이외의 추가 툴이 필요하지 않습니다.
- b. 계층이 줄어들어 더 작은 컨테이너 이미지를 만듭니다.
- c. 고품질 빌더 이미지를 재사용합니다.
- d. 상위 이미지가 변경되면 하위 이미지를 자동으로 업데이트합니다(예: 보안 수정 사항 포함).
- e. Docker 툴에서 생성하는 컨테이너 이미지와 달리 OpenShift와 호환되는 이미지를 만듭니다.

▶ 3. 다음 중 기존 이미지에서 하위 이미지를 빌드하기 위한 Dockerfile을 생성하는 두 가지 일반적인 시나리오는 무엇입니까? (두 개를 선택하십시오.)

- a. 새 런타임 라이브러리 추가.
- b. 호스트 시스템의 CPU에 대한 컨테이너 액세스에 제약 요건 설정
- c. 여러 컨테이너 이미지에서 다양한 애플리케이션에 대해 단일 이미지 계층으로 공유할 내부 라이브러리 추가

▶ 솔루션

컨테이너 이미지 설계에 대한 접근법

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 컨테이너 커뮤니티에서 권장하는 컨테이너 이미지 생성 방법은 무엇입니까? (한 개를 선택하십시오.)

- a. 기본 OS 컨테이너 내부에서 명령을 실행하고 컨테이너를 커밋한 다음, 새 컨테이너 이미지로 저장하거나 내보냅니다.
- b. Dockerfile의 명령을 실행하고 생성된 컨테이너 이미지를 이미지 레지스트리로 내보냅니다.
- c. tar 파일에서 컨테이너 이미지 계층을 수동으로 만듭니다.
- d. podman build 명령을 실행하여 컨테이너 이미지 설명을 YAML 형식으로 처리합니다.

▶ 2. 독립 실행형 S2I 프로세스를 Dockerfile의 대안으로 사용하여 얻을 수 있는 두 가지 이점은 무엇입니까? (두 개를 선택하십시오.)

- a. 기본 Podman 설치 이외의 추가 툴이 필요하지 않습니다.
- b. 계층이 줄어들어 더 작은 컨테이너 이미지를 만듭니다.
- c. 고품질 빌더 이미지를 재사용합니다.
- d. 상위 이미지가 변경되면 하위 이미지를 자동으로 업데이트합니다(예: 보안 수정 사항 포함).
- e. Docker 툴에서 생성하는 컨테이너 이미지와 달리 OpenShift와 호환되는 이미지를 만듭니다.

▶ 3. 다음 중 기존 이미지에서 하위 이미지를 빌드하기 위한 Dockerfile을 생성하는 두 가지 일반적인 시나리오는 무엇입니까? (두 개를 선택하십시오.)

- a. 새 런타임 라이브러리 추가.
- b. 호스트 시스템의 CPU에 대한 컨테이너 액세스에 제약 요건 설정
- c. 여러 컨테이너 이미지에서 다양한 애플리케이션에 대해 단일 이미지 계층으로 공유할 내부 라이브러리 추가

Dockerfiles를 사용하여 사용자 지정 컨테이너 이미지 빌드

목표

이 섹션을 마치면 공통 Dockerfile 명령을 사용하여 컨테이너 이미지를 만들 수 있습니다.

기본 컨테이너 빌드

Dockerfile은 컨테이너 이미지 빌드를 자동화하는 메커니즘입니다. Dockerfile에서 이미지를 빌드하는 작업은 다음과 같은 3단계 프로세스입니다.

1. 작업 디렉터리 만들기
2. **Dockerfile** 작성
3. Podman을 사용하여 이미지 빌드

작업 디렉터리 만들기

작업 디렉터리는 이미지를 빌드하는 데 필요한 모든 파일이 들어 있는 디렉터리입니다. 불필요한 파일을 이미지에 통합하지 않으려면 빈 작업 디렉터리를 생성하는 것이 좋습니다. 보안상의 이유로 루트 디렉터리 /를 이미지 빌드용 작업 디렉터리로 사용하면 안 됩니다.

Dockerfile 사양 작성

Dockerfile은 작업 디렉터리에 있어야 하는 텍스트 파일입니다. 이 파일에는 이미지 빌드에 필요한 명령이 포함되어 있습니다. **Dockerfile**의 기본 구문은 다음과 같습니다.

```
# Comment
INSTRUCTION arguments
```

해시 또는 파운드 기호(#)로 시작하는 줄은 주석입니다. INSTRUCTION은 Dockerfile 명령 키워드를 나타냅니다. 명령은 대/소문자를 구분하지 않지만, 알아보기 쉽도록 명령을 모두 대문자로 표시하는 것이 일반적입니다.

주석으로 처리되지 않은 첫 번째 명령은 기본 이미지를 지정하는 **FROM** 명령이어야 합니다. Dockerfile 명령은 이 이미지를 사용하는 새 컨테이너로 실행된 다음 새 이미지로 커밋됩니다. 다음 명령(있는 경우)은 새 이미지로 실행됩니다. 명령의 실행 순서는 Dockerfile에 표시된 순서를 따릅니다.



참고

ARG 명령은 FROM 명령 앞에 올 수 있지만, ARG 명령은 이 섹션의 목표를 벗어납니다.

각 Dockerfile 명령은 이전의 모든 명령에서 빌드된 중간 이미지를 사용하는 독립 컨테이너에서 실행됩니다. 따라서 각 명령은 Dockerfile의 다른 명령과 독립적입니다.

다음은 간단한 Apache 웹 서버 컨테이너를 빌드하는 Dockerfile의 예입니다.

```

# This is a comment line ①
FROM ubi7/ubi:7.7 ②
LABEL description="This is a custom httpd container image" ③
MAINTAINER John Doe <jdoe@xyz.com> ④
RUN yum install -y httpd ⑤
EXPOSE 80 ⑥
ENV LogLevel "info" ⑦
ADD http://someserver.com/filename.pdf /var/www/html ⑧
COPY ./src/ /var/www/html/ ⑨
USER apache ⑩
ENTRYPOINT ["/usr/sbin/httpd"] ⑪
CMD ["-D", "FOREGROUND"] ⑫

```

- ① 해시 또는 파운드 기호(#)로 시작하는 줄은 주석입니다.
- ② **FROM** 명령은 새 컨테이너 이미지가 **ubi7/ubi:7.7** 컨테이너 기본 이미지를 확장함을 선언합니다. Dockerfile은 운영 체제 배포판에 있는 이미지뿐만 아니라 다른 컨테이너 이미지를 기본 이미지로 사용할 수 있습니다. Red Hat은 인증 및 테스트된 컨테이너 이미지 세트를 제공하며, 이러한 컨테이너 이미지를 기본 이미지로 사용하도록 권장합니다.
- ③ **LABEL**은 이미지에 일반 메타데이터를 추가합니다. **LABEL**은 단순한 키-값 쌍입니다.
- ④ **MAINTAINER**는 생성된 컨테이너 이미지 메타데이터의 **Author** 필드를 나타냅니다. **podman inspect** 명령을 사용하여 이미지 메타데이터를 볼 수 있습니다.
- ⑤ **RUN**은 현재 이미지 맨 위의 새 계층에 있는 명령을 실행합니다. 명령을 실행하는 데 사용되는 쉘은 **/bin/sh**입니다.
- ⑥ **EXPOSE**는 컨테이너가 런타임에 지정된 네트워크 포트에서 수신함을 나타냅니다. **EXPOSE** 명령은 메타데이터만 정의합니다. 호스트에서 포트에 액세스하도록 설정할 수 없습니다. **podman run** 명령의 **-p** 옵션은 호스트의 컨테이너 포트를 공개합니다.
- ⑦ **ENV**는 컨테이너에서 사용할 수 있는 환경 변수를 정의합니다. Dockerfile 내에 다양한 **ENV** 명령을 선언할 수 있습니다. 컨테이너 내에 **env** 명령을 사용하여 각 환경 변수를 볼 수 있습니다.
- ⑧ **ADD** 명령은 로컬 또는 원격 소스에서 파일이나 폴더를 복사하여 컨테이너의 파일 시스템에 추가합니다. 이 명령이 로컬 파일을 복사하는 데 사용되는 경우 작업 디렉터리에 있어야 합니다. **ADD** 명령은 대상 이미지 디렉터리에 로컬 **.tar** 파일의 압축을 풁니다.
- ⑨ **COPY**는 작업 디렉터리에서 파일을 복사하여 컨테이너의 파일 시스템에 추가합니다. 이 Dockerfile 명령으로 URL을 사용하여 원격 파일을 복사할 수 없습니다.
- ⑩ **USER**는 **RUN**, **CMD**, **ENTRYPOINT** 명령에 대해 컨테이너 이미지를 실행할 때 사용할 사용자 이름 또는 UID를 지정합니다. 보안상의 이유로 **root**가 아닌 다른 사용자를 정의하는 것이 좋습니다.
- ⑪ **ENTRYPOINT**는 컨테이너에서 이미지가 실행될 때 실행할 기본 명령을 지정합니다. 생략할 경우 기본 **ENTRYPOINT**는 **/bin/sh -c**입니다.
- ⑫ **CMD**는 **ENTRYPOINT** 명령의 기본 인수를 제공합니다. 기본 **ENTRYPOINT**가 **(/bin/sh -c)**를 적용하는 경우 **CMD**는 컨테이너를 시작할 때 실행되는 실행 가능한 명령과 매개 변수를 형성합니다.

CMD 및 ENTRYPOINT

ENTRYPOINT 및 **CMD** 명령은 다음 두 가지 형식을 사용합니다.

- Exec 형식(JSON 배열 사용):

```

ENTRYPOINT ["command", "param1", "param2"]
CMD ["param1", "param2"]

```

- 쉘 형식:

5장 | 사용자 지정 컨테이너 이미지 생성

```
ENTRYPOINT command param1 param2
CMD param1 param2
```

Exec 형식이 기본 형식입니다. 쉘 형식은 `/bin/sh -c` 쉘에 명령을 래핑하므로 불필요한 쉘 프로세스가 생성되는 경우가 있습니다. 또한 일부 조합은 허용되지 않거나 예상대로 작동하지 않을 수 있습니다. 예를 들어 `ENTRYPOINT`가 `["ping"]`(exec 형식)이고 `CMD`가 `localhost`(쉘 형식)이면 예상 실행 명령은 `ping localhost`이지만 컨테이너가 잘못된 형식의 명령인 `ping /bin/sh -c localhost`를 시도합니다.

`Dockerfile`에 하나 이하의 `ENTRYPOINT` 및 하나의 `CMD` 명령이 있어야 합니다. 각 명령이 두 개 이상 있으면 마지막 명령만 적용됩니다. `ENTRYPOINT`를 지정하지 않고 `CMD`가 존재할 수 있습니다. 이 경우 기본 이미지의 `ENTRYPOINT`가 적용되거나, 정의되어 있지 않으면 기본 `ENTRYPOINT`가 적용됩니다.

Podman은 컨테이너를 시작할 때 `CMD` 명령을 재정의할 수 있습니다. 있는 경우, 이미지 이름 뒤에 오는 `podman run` 명령의 모든 매개 변수는 `CMD` 명령을 형성합니다. 예를 들어 다음 명령을 수행하면 실행 중인 컨테이너에서 현재 시간을 표시합니다.

```
ENTRYPOINT ["/bin/date", "+%H:%M"]
```

`ENTRYPOINT`가 실행할 명령과 매개 변수를 둘 다 정의합니다. 따라서 `CMD` 명령을 사용할 수 없습니다. 다음 예제는 동일한 기능을 제공하며, 컨테이너를 시작할 때 `CMD` 명령을 덮어쓸 수 있는 추가적인 이점이 있습니다.

```
ENTRYPOINT ["/bin/date"]
CMD ["+%H:%M"]
```

두 경우 모두, 매개 변수를 제공하지 않고 컨테이너를 시작하면 현재 시간이 표시됩니다.

```
[student@workstation ~]$ sudo podman run -it do180/rhel
11:41
```

두 번째 경우에는 `podman run` 명령에서 이미지 이름 뒤에 매개 변수가 오면 `CMD` 명령을 덮어씁니다. 다음 명령은 시간 대신 요일을 표시합니다.

```
[student@workstation demo-basic]$ sudo podman run -it do180/rhel +%A
Tuesday
```

또 다른 방법은 기본 `ENTRYPOINT` 및 `CMD` 명령을 사용하여 초기 명령을 정의하는 것입니다. 다음 명령은 현재 시간을 표시하며, 런타임에 재정의할 수 있는 추가적인 이점이 있습니다.

```
CMD ["date", "+%H:%M"]
```

ADD 및 COPY

`ADD` 및 `COPY` 명령은 다음 두 가지 형식을 취합니다.

- 쉘 형식:

```
ADD <source>... <destination>
COPY <source>... <destination>
```

- Exec 형식:

```
ADD ["<source>", ... "<destination>"]
COPY ["<source>", ... "<destination>"]
```

source가 파일 시스템 경로이면 작업 디렉터리 안에 있어야 합니다.

또한 **ADD** 명령을 사용하면 URL을 사용하여 리소스를 지정할 수 있습니다.

```
ADD http://someserver.com/filename.pdf /var/www/html
```

source가 압축 파일이면 **ADD** 명령은 destination 폴더에 파일 압축을 풉니다. **COPY** 명령에는 이 기능이 없습니다.



경고

ADD 및 **COPY** 명령은 둘 다 권한을 유지하고, **USER** 명령이 지정된 경우에도 **root**를 소유자로 사용하여 파일을 복사합니다. 복사 후에 **RUN** 명령을 사용해서 소유자를 변경하여 “permission denied” 오류를 방지하는 것이 좋습니다.

이미지 계층화

Dockerfile의 각 명령은 새 이미지 계층을 생성합니다. **Dockerfile**에 명령이 너무 많으면 계층이 과도하게 생성되어 이미지가 커집니다. 예를 들면, **Dockerfile**에서 다음 **RUN** 명령을 살펴보겠습니다.

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms"
RUN yum update -y
RUN yum install -y httpd
```

위 예제는 컨테이너 이미지를 생성할 때 좋은 방법이 아닙니다. 마지막 계층만 적용되는데 3개의 계층(**RUN** 명령당 하나씩)을 생성합니다. 계층 수를 최소화하는 것이 좋습니다. 단일 계층을 생성하는 동안 **&&** 접속사를 사용하면 동일한 목표를 달성할 수 있습니다.

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && yum update -y && yum
install -y httpd
```

이 방법의 문제점은 **Dockerfile**의 가독성이 저하된다는 것입니다. \ 이스케이프 코드를 사용하여 줄 바꿈을 삽입하면 가독성이 향상됩니다. 줄을 들여쓰기하여 명령을 정렬할 수도 있습니다.

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" &&
yum update -y && \
yum install -y httpd
```

이 예제에서는 하나의 계층만 생성하며 가독성이 향상됩니다. **RUN**, **COPY** 및 **ADD** 명령이 새 이미지 계층을 생성하지만, 이런 방식으로 **RUN**을 향상할 수 있습니다.

LABEL 및 **ENV**와 같이 여러 매개 변수를 허용하는 다른 명령에도 유사한 형식 지정 규칙을 적용하는 것이 좋습니다.

```
LABEL version="2.0" \
      description="This is an example container image" \
      creationDate="01-09-2017"
```

```
ENV MYSQL_ROOT_PASSWORD="my_password" \
      MYSQL_DATABASE "my_database"
```

Podman을 사용하여 이미지 빌드

`podman build` 명령은 **Dockerfile**을 처리하고 포함된 명령에 따라 새 이미지를 빌드합니다. 이 명령의 구문은 다음과 같습니다.

```
$ podman build -t NAME:TAG DIR
```

DIR은 작업 디렉터리의 경로로, **Dockerfile**을 포함해야 합니다. 작업 디렉터리가 현재 디렉터리인 경우 점(.)으로 지정된 현재 디렉터리일 수 있습니다. NAME:TAG는 새 이미지에 지정된 태그가 있는 이름입니다. TAG를 지정하지 않으면 이미지에 **latest** 태그가 자동으로 지정됩니다.



참조

Dockerfile 참조 가이드

<https://docs.docker.com/engine/reference/builder/>

기본 이미지 생성

<https://docs.docker.com/engine/userguide/eng-image/baseimages/>

▶ 연습 가이드

기본 Apache 컨테이너 이미지 생성

이 연습에서는 기본 Apache 컨테이너 이미지를 만듭니다.

결과

Red Hat Enterprise Linux 7.5 이미지에 빌드된 사용자 지정 Apache 컨테이너 이미지를 생성할 수 있게 됩니다.

시작하기 전에

다음 명령을 실행하여 관련 랩 파일을 다운로드하고, Docker가 실행되고 있는지 확인합니다.

```
[student@workstation ~]$ lab dockerfile-create start
```

▶ 1. Apache Dockerfile 생성

- 1.1. **workstation**에서 터미널을 엽니다. 선호하는 편집기를 사용하고 새 Dockerfile을 만듭니다.

```
[student@workstation ~]$ vim /home/student/D0180/labs/dockerfile-create/Dockerfile
```

- 1.2. 새 Dockfile의 맨 위에 다음 **FROM** 명령을 추가하여 UBI 7.7을 기본 이미지로 사용합니다.

```
FROM ubi7/ubi:7.7
```

- 1.3. **FROM** 명령 아래에 **MAINTAINER** 명령을 포함하여 새 이미지의 **Author** 필드를 설정합니다. 이름 및 이메일 주소를 포함하도록 값을 바꿉니다.

```
MAINTAINER Your Name <youremail>
```

- 1.4. **MAINTAINER** 명령 아래에 다음 **LABEL** 명령을 추가하여 새 이미지에 설명 메타데이터를 추가합니다.

```
LABEL description="A custom Apache container based on UBI 7"
```

- 1.5. **yum install** 명령과 함께 **RUN** 명령을 추가하여 새 컨테이너에 Apache를 설치합니다.

```
RUN yum install -y httpd && \
    yum clean all
```

- 1.6. **RUN** 명령을 추가하여 기본 HTTPD 홈페이지의 내용을 바꿉니다.

```
RUN echo "Hello from Dockerfile" > /var/www/html/index.html
```

5장 | 사용자 지정 컨테이너 이미지 생성

- 1.7. **RUN** 명령 아래에 **EXPOSE** 명령을 사용하여 런타임에 컨테이너가 수신 대기하는 포트를 기록합니다. 이 경우 포트를 Apache 서버의 기본값인 80으로 설정합니다.

```
EXPOSE 80
```

**참고**

EXPOSE 명령을 통해 실제로 호스트에서 지정된 포트를 사용할 수 있게 되는 것은 아닙니다. 대신, 이 명령은 컨테이너가 수신 대기하는 포트에 대한 메타데이터 역할을 합니다.

- 1.8. 파일 끝에 다음 **CMD** 명령을 사용하여 **httpd**를 기본 진입점으로 설정합니다.

```
CMD ["httpd", "-D", "FOREGROUND"]
```

- 1.9. 저장하고 다음 단계를 진행하기 전에 Dockerfile이 다음과 일치하는지 확인합니다.

```
FROM ubi7/ubi:7.7

MAINTAINER Your Name <youremail>

LABEL description="A custom Apache container based on UBI 7"

RUN yum install -y httpd && \
    yum clean all

RUN echo "Hello from Dockerfile" > /var/www/html/index.html

EXPOSE 80

CMD ["httpd", "-D", "FOREGROUND"]
```

▶ 2. Apache 컨테이너 이미지를 빌드하고 확인합니다.

- 2.1. 다음 명령을 사용하여 새로 만든 Dockerfile로 기본 Apache 컨테이너 이미지를 만듭니다.

```
[student@workstation ~]$ cd /home/student/D0180/labs/dockerfile-create
[student@workstation dockerfile-create]$ sudo podman build --layers=false \
> -t do180/apache .
STEP 1: FROM ubi7/ubi:7.7
Getting image source signatures ①
Copying blob sha256:...output omitted...
71.46 MB / 71.46 MB [=====] 18s
...output omitted...
Storing signatures
STEP 2: MAINTAINER Your Name <youremail>
STEP 3: LABEL description="A custom Apache container based on UBI 7"
STEP 4: RUN yum install -y httpd &&      yum clean all
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
...output omitted...
STEP 5: RUN echo "Hello from Dockerfile" > /var/www/html/index.html
STEP 6: EXPOSE 80
STEP 7: CMD ["httpd", "-D", "FOREGROUND"]
```

5장 | 사용자 지정 컨테이너 이미지 생성

```
ERRO[0109] HOSTNAME is not supported for OCI image format...output omitted... ②
STEP 8: COMMIT ...output omitted... localhost/do180/apache:latest
Getting image source signatures
...output omitted...
Storing signatures
--> b49375fa8ee1e0ad5b4a82bb088e5befbe59377bcf
```

- ❶ 로컬 스토리지에 아직 없는 경우에만 **FROM** 명령에 나열된 컨테이너 이미지가 다운로드 됩니다.
- ❷ 이 오류는 무해하며, 무시해도 됩니다. Podman의 알려진 문제이며(버그 1634806 [https://bugzilla.redhat.com/show_bug.cgi?id=1634806] 참조), 이후 버전에서 해결 될 예정입니다.

**참고**

Podman은 빌드 프로세스 중에 익명의 많은 중간 이미지를 생성합니다. **-a**를 사용하지 않으면 해당 이미지는 나열되지 않습니다. **build** 하위 명령의 **--layers=false** 옵션을 사용하여 중간 이미지를 삭제하도록 Podman에 지시합니다.

- 2.2. 빌드 프로세스가 완료되면 **podman images**를 실행하여 이미지 리포지토리의 새 이미지를 확인합니다.

```
[student@workstation dockerfile-create]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
localhost/do180/apache    latest   b49375fa8ee1  11 seconds ago  247MB
registry.access.redhat.com/ubi7/ubi    7.7     0355cd652bd1  8 months ago   215MB
```

- ▶ 3. Apache 컨테이너를 실행합니다.

- 3.1. 다음 명령을 사용하여 Apache 이미지로 컨테이너를 실행합니다.

```
[student@workstation dockerfile-create]$ sudo podman run --name lab-apache \
> -d -p 10080:80 do180/apache
fa1d1c450e8892ae085dd8bbf763edac92c41e6ffaa7ad6ec6388466809bb391
```

- 3.2. **podman ps** 명령을 실행하여 실행 중인 컨테이너를 확인합니다.

```
[student@workstation dockerfile-create]$ sudo podman ps
CONTAINER ID IMAGE                  COMMAND            ...output omitted...
fa1d1c450e88 localhost/do180/apache:latest httpd -D FOREGROU...output omitted...
```

- 3.3. **curl** 명령을 사용하여 서버가 실행되고 있는지 확인합니다.

```
[student@workstation dockerfile-create]$ curl 127.0.0.1:10080
Hello from Dockerfile
```

완료

workstation에서 **lab dockerfile-create finish** 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab dockerfile-create finish
```

이로써 안내에 따른 연습이 완료됩니다.

▶ 랩

사용자 지정 컨테이너 이미지 생성

수행 체크리스트

이 랩에서는 사용자 지정 Apache 웹 서버 컨테이너 이미지를 빌드하기 위한 Dockerfile을 생성합니다. 사용자 지정 이미지는 RHEL 7.7 UBI 이미지를 기반으로 하며, 사용자 지정 `index.html` 페이지를 제공합니다.

결과

정적 HTML 파일을 호스팅하는 사용자 지정 Apache 웹 서버 컨테이너를 만들 수 있습니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab dockerfile-review start
```

1. /home/student/D0180/labs/dockerfile-review/ 폴더에서 제공된 Dockerfile stub를 검토합니다. Dockerfile을 편집하고 다음 사양을 충족하는지 확인합니다.
 - 기본 이미지는 `ubi7/ubi:7.7`입니다.
 - MAINTAINER 명령을 사용하여 원하는 작성자 이름 및 이메일 ID를 설정합니다.
 - 환경 변수 `PORT`를 8080으로 설정합니다.
 - Apache(`httpd` 패키지)를 설치합니다.
 - 기본 포트 80 대신 포트 8080을 수신 대기하도록 Apache 구성 파일 `/etc/httpd/conf/httpd.conf`를 변경합니다.
 - `/etc/httpd/logs` 및 `/run/httpd` 폴더의 소유권을 사용자 및 그룹 `apache`(UID 및 GID: 48)로 변경합니다.
 - 컨테이너 사용자가 Apache 웹 서버에 액세스하는 방법을 알 수 있도록 `PORT` 환경 변수에 설정된 값을 노출합니다.
 - 랩 디렉터리의 `src/` 폴더 내용을 컨테이너 내부의 Apache `DocumentRoot` 파일(`/var/www/html/`)에 복사합니다.

`src` 폴더에는 `Hello World!` 메시지를 인쇄하는 하나의 `index.html` 파일이 있습니다.

 - 다음 명령을 사용하여 포그라운드에서 Apache `httpd` 데몬을 시작합니다.

```
httpd -D FOREGROUND
```

2. `do180/custom-apache`를 이름으로 사용하여 사용자 지정 Apache 이미지를 빌드합니다.
3. 다음 특성을 사용하여 분리된 모드로 새 컨테이너를 만듭니다.

- 이름: **dockerfile**
- 컨테이너 이미지: **do180/custom-apache**
- 포트 전달: from host port 20080 to container port 8080
- 데몬으로 실행: yes

컨테이너가 준비되고 실행 중인지 확인합니다.

4. 서버가 HTML 파일을 제공하고 있는지 확인합니다.

평가

workstation 시스템에서 **lab dockerfile-review grade** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab dockerfile-review grade
```

완료

workstation에서 **lab dockerfile-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab dockerfile-review finish
```

이로써 랩이 완료됩니다.

▶ 솔루션

사용자 지정 컨테이너 이미지 생성

수행 체크리스트

이 랩에서는 사용자 지정 Apache 웹 서버 컨테이너 이미지를 빌드하기 위한 Dockerfile을 생성합니다. 사용자 지정 이미지는 RHEL 7.7 UBI 이미지를 기반으로 하며, 사용자 지정 `index.html` 페이지를 제공합니다.

결과

정적 HTML 파일을 호스팅하는 사용자 지정 Apache 웹 서버 컨테이너를 만들 수 있습니다.

시작하기 전에

`workstation`에서 `student` 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab dockerfile-review start
```

1. /home/student/D0180/labs/dockerfile-review/ 폴더에서 제공된 Dockerfile stub를 검토합니다. Dockerfile을 편집하고 다음 사양을 충족하는지 확인합니다.
 - 기본 이미지는 `ubi7/ubi:7.7`입니다.
 - MAINTAINER 명령을 사용하여 원하는 작성자 이름 및 이메일 ID를 설정합니다.
 - 환경 변수 `PORT`를 8080으로 설정합니다.
 - Apache(`httpd` 패키지)를 설치합니다.
 - 기본 포트 80 대신 포트 8080을 수신 대기하도록 Apache 구성 파일 `/etc/httpd/conf/httpd.conf`를 변경합니다.
 - `/etc/httpd/logs` 및 `/run/httpd` 폴더의 소유권을 사용자 및 그룹 `apache`(UID 및 GID: 48)로 변경합니다.
 - 컨테이너 사용자가 Apache 웹 서버에 액세스하는 방법을 알 수 있도록 `PORT` 환경 변수에 설정된 값을 노출합니다.
 - 랩 디렉터리의 `src/` 폴더 내용을 컨테이너 내부의 Apache `DocumentRoot` 파일(`/var/www/html/`)에 복사합니다.

`src` 폴더에는 `Hello World!` 메시지를 인쇄하는 하나의 `index.html` 파일이 있습니다.

 - 다음 명령을 사용하여 포그라운드에서 Apache `httpd` 데몬을 시작합니다.

```
httpd -D FOREGROUND
```

1. 선호하는 편집기를 사용하여 /home/student/D0180/labs/dockerfile-review/ 폴더에 있는 Dockerfile을 수정합니다.

5장 | 사용자 지정 컨테이너 이미지 생성

```
[student@workstation ~]$ cd /home/student/D0180/labs/dockerfile-review/  
[student@workstation dockerfile-review]$ vim Dockerfile
```

- 1.2. Dockerfile의 기본 이미지를 **ubi7/ubi:7.7**로 설정합니다.

```
FROM ubi7/ubi:7.7
```

- 1.3. **MAINTAINER** 명령으로 이름 및 이메일을 설정합니다.

```
MAINTAINER Your Name <youremail>
```

- 1.4. **PORT**라는 환경 변수를 만들고 8080으로 설정합니다.

```
ENV PORT 8080
```

- 1.5. Apache 서버를 설치합니다.

```
RUN yum install -y httpd && \  
    yum clean all
```

- 1.6. 단일 **RUN** 명령을 사용하여 포트 8080을 수신 대기하도록 Apache HTTP 서버 구성 파일을 변경하고, 서버 작업 폴더의 소유권을 변경합니다.

```
RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf && \  
    chown -R apache:apache /etc/httpd/logs/ && \  
    chown -R apache:apache /run/httpd/
```

- 1.7. **USER** 명령을 사용하여 **apache** 사용자로 컨테이너를 실행합니다. **EXPOSE** 명령을 사용하여 런타임에 컨테이너가 수신하는 포트를 기록합니다. 이 경우 포트를 Apache 서버의 기본값인 **PORT** 환경 변수로 설정합니다.

```
USER apache  
  
# Expose the custom port that you provided in the ENV var  
EXPOSE ${PORT}
```

- 1.8. **src** 폴더의 모든 파일을 **/var/www/html**의 Apache **DocumentRoot** 경로에 복사합니다.

```
# Copy all files under src/ folder to Apache DocumentRoot (/var/www/html)  
COPY ./src/ /var/www/html/
```

- 1.9. 마지막으로, **CMD** 명령을 삽입하여 포그라운드에서 **httpd**를 실행하고 Dockerfile을 저장합니다.

```
# Start Apache in the foreground  
CMD ["httpd", "-D", "FOREGROUND"]
```

2. **do180/custom-apache**를 이름으로 사용하여 사용자 지정 Apache 이미지를 빌드합니다.

5장 | 사용자 지정 컨테이너 이미지 생성

2.1. 사용자 지정 Apache 이미지의 Dockerfile을 확인합니다.

사용자 지정 Apache 이미지의 Dockerfile은 다음과 같아야 합니다.

```
FROM ubi7/ubi:7.7

MAINTAINER Your Name <youremail>

ENV PORT 8080

RUN yum install -y httpd && \
    yum clean all

RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf && \
    chown -R apache:apache /etc/httpd/logs/ && \
    chown -R apache:apache /run/httpd/

USER apache

# Expose the custom port that you provided in the ENV var
EXPOSE ${PORT}

# Copy all files under src/ folder to Apache DocumentRoot (/var/www/html)
COPY ./src/ /var/www/html/

# Start Apache in the foreground
CMD ["httpd", "-D", "FOREGROUND"]
```

2.2. `sudo podman build` 명령을 실행하여 사용자 지정 Apache 이미지를 빌드하고, 이름을 `do180/custom-apache`로 지정합니다.

```
[student@workstation dockerfile-review]$ sudo podman build --layers=false \
> -t do180/custom-apache .
STEP 1: FROM ubi7/ubi:7.7
...output omitted...
STEP 2: MAINTAINER username <username@example.com>
STEP 3: ENV PORT 8080
STEP 4: RUN yum install -y httpd &&      yum clean all
...output omitted...
STEP 5: RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf
&&      chown -R apache:apache /etc/httpd/logs/ &&      chown -R apache:apache /
run/httpd/
STEP 6: USER apache
STEP 7: EXPOSE ${PORT}
STEP 8: COPY ./src/ /var/www/html/
STEP 9: CMD ["httpd", "-D", "FOREGROUND"]
ERRO[0109] HOSTNAME is not supported for OCI image format...output omitted...
STEP 10: COMMIT ...output omitted... localhost/do180/custom-apache:latest
...output omitted...
```

2.3. `podman images` 명령을 실행하여 사용자 지정 이미지가 제대로 빌드되었는지 확인합니다.

5장 | 사용자 지정 컨테이너 이미지 생성

```
[student@workstation dockerfile-review]$ sudo podman images
REPOSITORY                                     TAG      IMAGE ID      ...
localhost/do180/custom-apache                 latest   da92b9426325 ...
registry.access.redhat.com/ubi7/ubi           7.7     6fecccc91c83 ...
```

3. 다음 특성을 사용하여 분리된 모드로 새 컨테이너를 만듭니다.

- 이름: **dockerfile**
- 컨테이너 이미지: **do180/custom-apache**
- 포트 전달: from host port 20080 to container port 8080
- 데몬으로 실행: yes

컨테이너가 준비되고 실행 중인지 확인합니다.

- 3.1. 컨테이너를 만들고 실행합니다.

```
[student@workstation dockerfile-review]$ sudo podman run -d \
> --name dockerfile -p 20080:8080 do180/custom-apache
367823e35c4a...
```

- 3.2. 컨테이너가 준비되고 실행 중인지 확인합니다.

```
[student@workstation dockerfile-review]$ sudo podman ps
... IMAGE          COMMAND          ... PORTS          NAMES
... do180/custom... "httpd -D ..." ... 0.0.0.0:20080->8080/tcp dockerfile
```

4. 서버가 HTML 파일을 제공하고 있는지 확인합니다.

127.0.0.1:20080에서 curl 명령 실행

```
[student@workstation dockerfile-review]$ curl 127.0.0.1:20080
```

출력은 다음과 같아야 합니다.

```
<html>
<header><title>D0180 Hello!</title></header>
<body>
  Hello World! The dockerfile-review lab works!
</body>
</html>
```

평가

workstation 시스템에서 **lab dockerfile-review grade** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab dockerfile-review grade
```

완료

workstation에서 **lab dockerfile-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab dockerfile-review finish
```

이로써 랩이 완료됩니다.

요약

이 장에서 학습한 내용:

- **Dockerfile**에는 컨테이너 이미지를 생성하는 방법을 지정하는 명령이 포함되어 있습니다.
- Red Hat Container Catalog 또는 Quay.io에서 제공하는 컨테이너 이미지는 특정 언어나 기술에 대한 사용자 지정 이미지를 생성하는 데 효과적인 시작 지점입니다.
- Dockerfile에서 이미지를 빌드하는 작업은 다음과 같은 3단계 프로세스입니다.
 1. 작업 디렉터리를 만듭니다.
 2. **Dockerfile** 파일에서 빌드 명령을 지정합니다.
 3. **podman build** 명령을 사용하여 이미지를 빌드합니다.
- S2I(Source-to-Image) 프로세스는 Dockerfile에 대한 대안을 제공합니다. S2I는 애플리케이션 소스 코드에서 일반 기술에 대한 표준화된 컨테이너 이미지 빌드 프로세스를 구현합니다. 따라서 개발자가 Dockerfile 개발이 아닌 애플리케이션 개발에 집중할 수 있습니다.

6장

OpenShift에 컨테이너화된 애플리케이션 배포

목적

OpenShift Container Platform에 단일 컨테이너 애플리케이션 배포

목표

- Kubernetes 및 Red Hat OpenShift Container Platform의 아키텍처를 설명합니다.
- 표준 Kubernetes 리소스를 만듭니다.
- 서비스 경로를 생성합니다.
- OpenShift Container Platform의 Source-to-Image 기능을 사용하여 애플리케이션을 빌드합니다.
- OpenShift 웹 콘솔을 사용하여 애플리케이션을 만듭니다.

섹션

- Kubernetes 리소스 생성(안내에 따른 연습)
- 경로 생성(안내에 따른 연습)
- S2I(Source-to-Image)로 애플리케이션 생성(안내에 따른 연습)

랩

OpenShift에 컨테이너화된 애플리케이션 배포

Kubernetes 리소스 생성

목표

이 섹션을 마친 수강생은 표준 Kubernetes 리소스를 생성할 수 있습니다.

Red Hat OpenShift Container Platform(RHOCP) 명령 줄 툴

RHOCP 클러스터와 상호 작용하는 주요 방법은 **oc** 명령을 사용하는 것입니다. 기본적으로 다음 구문의 하위 명령을 통해 명령을 사용합니다.

```
$> oc <command>
```

대부분 작업은 클러스터와 상호 작용하기 전에 사용자 로그인이 필요합니다. 로그인 구문은 다음과 같습니다.

```
$> oc login <clusterUrl>
```

포드 리소스 정의 구문 설명

RHOCP는 Kubernetes 포드 내에서 컨테이너를 실행하며, 컨테이너 이미지에서 포드를 생성하려면 OpenShift에 포드 리소스 정의가 필요합니다. 이 정의는 JSON 또는 YAML 텍스트 파일로 제공하거나 **oc new-app** 명령 또는 OpenShift 웹 콘솔을 통해 기본값에서 생성할 수 있습니다.

포드는 컨테이너 및 기타 리소스의 컬렉션입니다. YAML 형식으로 된 WildFly 애플리케이션 서버 포드 정의의 예는 다음과 같습니다.

```
apiVersion: v1
kind: Pod①
metadata:
  name: wildfly②
  labels:
    name: wildfly③
spec:
  containers:
    - resources:
        limits :
          cpu: 0.5
      image: do276/todojee
      name: wildfly
      ports:
        - containerPort: 8080④
          name: wildfly
    env:⑤
      - name: MYSQL_ENV_MYSQL_DATABASE
        value: items
      - name: MYSQL_ENV_MYSQL_USER
```

```

    value: user1
  - name: MYSQL_ENV_MYSQL_PASSWORD
    value: mypa55
  
```

- ❶ Kubernetes 포드의 리소스 유형을 선언합니다.
- ❷ 관리자가 명령을 실행할 수 있는 Kubernetes 내 포드의 고유한 이름입니다.
- ❸ Kubernetes의 다른 리소스(일반적으로 서비스)에서 찾는데 사용할 수 있는 **name**이라는 키로 레이블을 생성합니다.
- ❹ 컨테이너의 노출되는 포트를 식별하는 컨테이너 종속 속성입니다.
- ❺ 환경 변수의 컬렉션을 정의합니다.

일부 포드에는 컨테이너에서 읽을 수 있는 환경 변수가 필요할 수 있습니다. Kubernetes는 모든 **name** 및 **value** 쌍을 환경 변수로 전환합니다. 인스턴스의 경우 **MYSQL_ENV_MYSQL_USER** 변수는 **user1**이라는 값을 사용하여 Kubernetes 런타임에서 내부적으로 선언되며 컨테이너 이미지 정의로 전달됩니다. 컨테이너에서 동일한 변수 이름을 사용하여 사용자의 로그인을 가져오므로 MySQL 데이터베이스 인스턴스에 액세스하는 사용자 이름을 설정하도록 WildFly 컨테이너 인스턴스에서 값을 사용합니다.

서비스 리소스 정의 구문 설명

Kubernetes에서는 다른 작업자의 포드가 연결할 수 있는 가상 네트워크가 제공되지만, 포드에서 다른 포드의 IP 주소를 간편하게 검색할 수 없습니다.

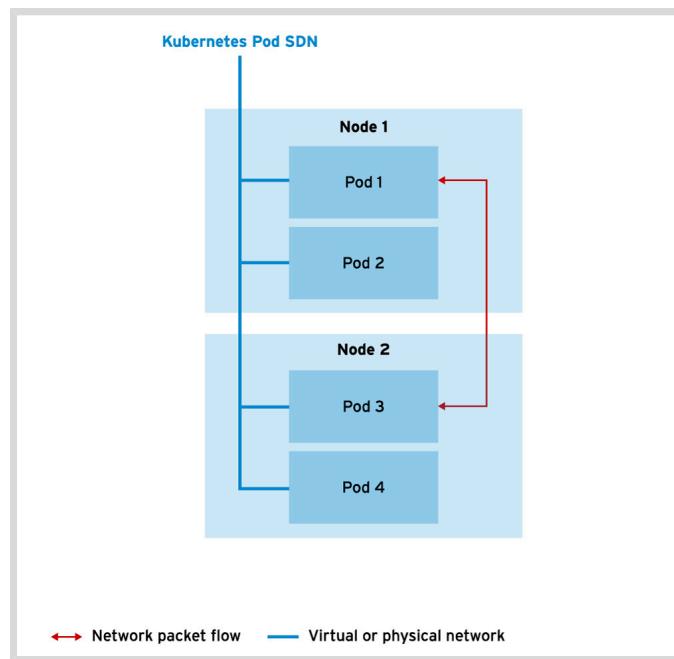


그림 6.1: 기본 Kubernetes 네트워킹

서비스는 모든 OpenShift 애플리케이션의 필수 리소스입니다. 이 서비스를 사용하면 한 포드의 컨테이너에서 다른 포드에 있는 컨테이너의 네트워크 연결을 열 수 있습니다. 포드는 여러 이유로 인해 다시 시작할 수 있으며 매번 다른 내부 IP 주소를 받습니다. 다시 시작한 후 매번 포드가 다른 포드의 IP 주소를 검색할 필요 없이, 서비스에서는 다시 시작한 후 포드를 실행하는 작업자 노드와 상관없이 다른 포드가 사용할 안정된 IP 주소를 제공합니다.

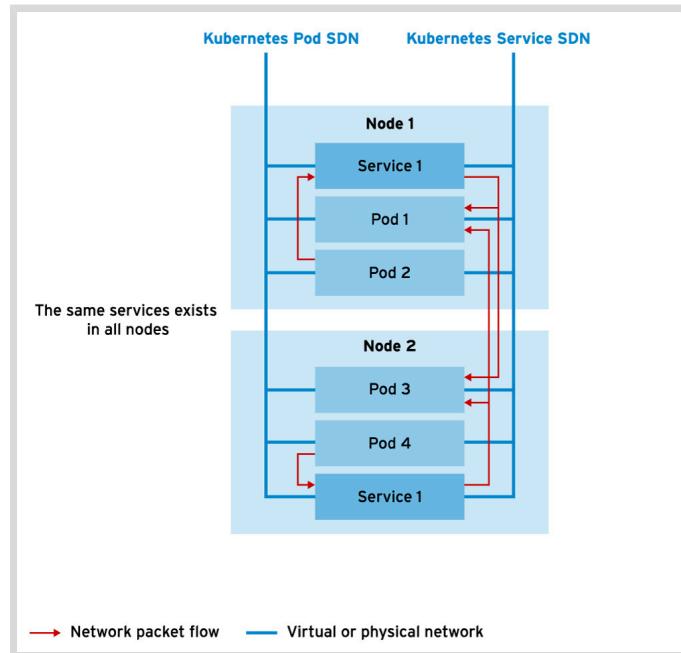


그림 6.2: Kubernetes 서비스 네트워킹

대부분의 실제 애플리케이션은 단일 포드로 실행되지 않습니다. 애플리케이션은 수평으로 확장되어야 하므로, 증가하는 사용자 요구를 만족시키기 위해 많은 포드가 동일한 포드 리소스 정의에서 동일한 컨테이너를 실행합니다. 서비스는 포드 집합과 연결되어 있으며, 전체 집합에 단일 IP 주소를 제공하고 멤버 포드 간에 클라이언트 요청의 부하를 분산합니다.

서비스 뒤에서 실행 중인 포드 집합은 DeploymentConfig 리소스가 관리합니다. DeploymentConfig 리소스는 생성해야 할 포드 복사본(복제본) 수를 관리하고 실패하는 경우 새 복사본을 생성하는 ReplicationController를 포함합니다. DeploymentConfig 및 ReplicationController 리소스는 이 장의 후반부에서 설명합니다.

다음 예는 최소 서비스 정의를 JSON 구문으로 보여줍니다.

```
{
  "kind": "Service", ①
  "apiVersion": "v1",
  "metadata": {
    "name": "quotedb" ②
  },
  "spec": {
    "ports": [ ③
      {
        "port": 3306,
        "targetPort": 3306
      }
    ],
    "selector": {
      "name": "mysqldb" ④
    }
  }
}
```

① Kubernetes 리소스의 종류입니다. 이 경우는 서비스입니다.

- ❷ 서비스의 고유 이름입니다.
- ❸ **ports**는 서비스에서 공개한 네트워크 포트를 설명하는 오브젝트의 배열입니다. **targetPort** 속성은 포드 컨테이너 정의의 **containerPort**와 일치해야 하며 **port** 속성은 서비스에서 노출하는 포트입니다. 클라이언트에서 서비스 포트에 연결하고 서비스에서 포드 **targetPort**에 패킷을 전달합니다.
- ❹ **selector**는 서비스가 패킷을 전달하기 위해 포드를 찾는 방법입니다. 대상 포드에는 메타데이터 특성에 일치하는 레이블이 있어야 합니다. 서비스에서 레이블이 일치하는 포드를 여러 개 발견하면 해당 포드에 네트워크 연결 부하를 분산합니다.

각 서비스에는 클라이언트가 연결할 고유한 IP 주소가 할당됩니다. 이 IP 주소는 포드 내부 네트워크와 구분된 다른 내부 OpenShift SDN에서 제공되며 포드에만 표시됩니다. **selector**와 일치하는 각 포드는 엔드포인트로 서비스 리소스에 추가됩니다.

서비스 검색

일반적으로 애플리케이션에서는 환경 변수를 사용하여 서비스 IP 주소와 포트를 찾습니다. OpenShift 프로젝트에 있는 각 서비스에서 동일한 프로젝트에 있는 모든 포드에 대해 환경 변수가 자동으로 정의되며 컨테이너에 삽입됩니다.

- **SVC_NAME_SERVICE_HOST**는 서비스 IP 주소입니다.
- **SVC_NAME_SERVICE_PORT**는 서비스 TCP 포트입니다.



참고

DNS 이름 지정 제한사항에 맞게 변수의 SVC_NAME 부분이 변경됩니다. 문자는 대문자로 변경되고 밑줄(_)은 대시(-)로 대체됩니다.

포드에서 서비스를 검색하는 또 다른 방법은 포드에만 표시되는 OpenShift 내부 DNS 서버를 사용하는 것입니다. 각 서비스에는 FQDN 형식의 SRV 레코드가 자동으로 할당됩니다.

SVC_NAME.PROJECT_NAME.svc.cluster.local

환경 변수를 사용하여 서비스를 검색할 때 서비스를 생성한 후에만 포드를 생성 및 시작해야 합니다. DNS 쿼리를 사용하여 서비스를 검색하도록 애플리케이션을 작성한 경우에도 포드를 시작한 후 생성된 서비스를 찾을 수 있습니다.

애플리케이션이 OpenShift 클러스터 외부에서 서비스에 액세스하는 방법은 두 가지가 있습니다.

1. **NodePort** 유형: 이전 Kubernetes 기반 접근법입니다. 이 접근법에서는 서비스가 작업자 노드 호스트의 사용 가능한 포트에 바인드되어 외부 클라이언트에 노출된 후에 서비스 IP 주소에 대한 연결을 프록시합니다. **oc edit svc** 명령을 사용하여 서비스 속성을 편집하고, **NodePort**를 **type**에 대한 값으로 지정한 후 **nodePort** 속성에 포트값을 제공합니다. 그러면 OpenShift에서 작업자 노드 호스트의 공용 IP 주소 및 **nodePort**에 설정된 포트값을 통해 서비스에 대한 연결을 프록시합니다.
2. OpenShift 경로: OpenShift에서 고유한 URL을 사용하여 서비스를 노출하는 선호되는 접근법입니다. 외부에서 액세스할 수 있도록 **oc expose** 명령을 사용하여 서비스를 노출하거나 OpenShift 웹 콘솔에서 서비스를 노출합니다.

그림 6.3에서는 **NodePort** 서비스를 통해 Kubernetes 서비스에 대한 외부 액세스를 허용하는 방법을 보여줍니다. OpenShift 경로는 이 교육 과정의 뒷부분에서 더 자세히 설명합니다.

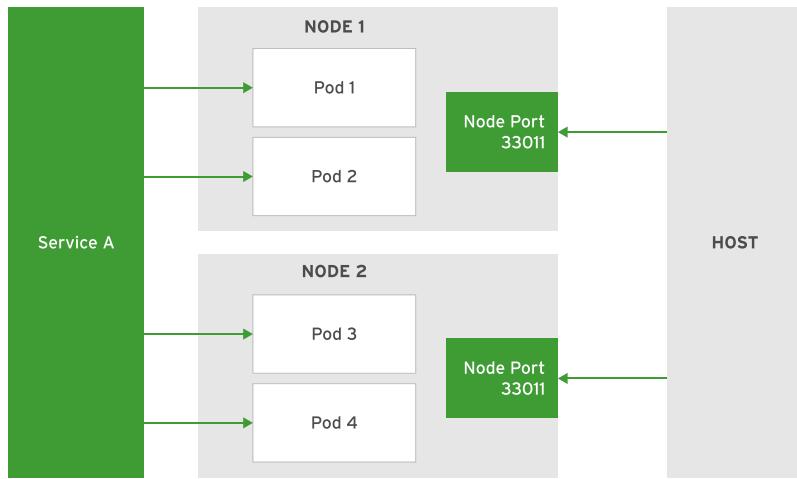


그림 6.3: Kubernetes 서비스에 대한 외부 액세스 권한 대체 방법

OpenShift는 포드 포트에 로컬 포트를 전달하는 **oc port-forward** 명령을 제공합니다. 서비스 리소스를 통해 포드에 액세스하는 것과는 다릅니다.

- port-forwarding 매핑은 **oc** 클라이언트가 실행하는 워크스테이션에서만 있으며 서비스는 모든 네트워크 사용자를 위해 포트를 매핑합니다.
- 서비스는 잠재적으로 여러 포드에 연결을 부하 분산을 수행하지만 port-forwarding 매핑은 단일 포드에 연결을 전달합니다.



참고

Red Hat은 서비스를 직접 연결에 노출하지 않도록 **NodePort** 접근법 사용을 권장하지 않습니다. OpenShift에서 port-forwarding을 통한 매핑은 더 안전한 대안으로 간주됩니다.

다음 예에서는 **oc port-forward** 명령을 사용하는 방법을 보여줍니다.

```
[student@workstation ~]$ oc port-forward mysql-openshift-1-g1qrp 3306:3306
```

이전 명령은 개발자 시스템에서 **db** 포드에 있는 포트 3306에 포트 3306을 전달하며 여기에서 MySQL 서버(컨테이너 내)는 네트워크 연결을 수락합니다.



참고

이 명령을 실행할 때 터미널 창은 계속 실행되게 두어야 합니다. 창을 닫거나 프로세스를 취소하면 포트 매핑이 중지됩니다.

새 애플리케이션 생성

간단한 애플리케이션, 복잡한 다계층 애플리케이션 및 마이크로 서비스 애플리케이션은 단일 리소스 정의 파일을 통해 설명할 수 있습니다. 이 단일 파일에는 포드에 연결하기 위한 서비스 정의, 애플리케이션 데이터를 수평으로 확장할 복제 컨트롤러 또는 DeploymentConfigs, 애플리케이션 데이터를 유지할 PersistentVolumeClaims 및 OpenShift에서 관리하는데 필요한 기타 모든 사항과 함께 여러 포드 정의가 포함되어 있습니다.

**중요**

OpenShift 4.5에서 **oc new-app** 명령은 이제 기본적으로 DeploymentConfig 리소스 대신 배포 리소스를 생성합니다. DO180의 이 버전은 배포 리소스를 다루지 않으며 DeploymentConfigs만 다룹니다. DeploymentConfig 리소스를 생성하려면 **oc new-app**을 호출할 때 **--as-deployment-config** 플래그를 전달할 수 있으며 이는 이 과정의 전체에 걸쳐 수행됩니다. 자세한 내용은 배포 및 DeploymentConfigs 이해 [https://docs.openshift.com/container-platform/4.5/applications/deployments/what-deployments-are.html#what-deployments-are]을 참조하십시오.

oc new-app 명령은 JSON 또는 YAML 형식에서 각각 기본 리소스 정의 파일을 생성하도록 **-o json** 또는 **-o yaml** 옵션과 함께 사용할 수 있습니다. 이 파일은 사용자 지정할 수 있으며 **oc create -f <filename>** 명령을 사용하여 애플리케이션을 생성하는데 사용하거나 복합 애플리케이션을 생성하도록 다른 리소스 정의 파일과 함께 병합할 수 있습니다.

oc new-app 명령은 애플리케이션 포드를 생성하여 다양한 방법으로 OpenShift에서 실행할 수 있습니다. 기존 Docker 이미지, Dockerfile 및 S2I(Source-to-Image) 프로세스를 사용하여 원시 소스 코드에서 포드를 생성할 수 있습니다.

oc new-app -h 명령을 실행하여 OpenShift에서 새 애플리케이션을 생성하는데 사용할 수 있는 다양한 모든 옵션을 파악합니다.

다음 명령은 **db=mysql**로 레이블이 설정된 Docker Hub에서 이미지 **mysql**을 기반으로 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc new-app mysql --as-deployment-config \
> MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -l db=mysql
```

다음 그림은 인수가 컨테이너 이미지인 경우 **oc new-app** 명령에서 생성한 Kubernetes 및 OpenShift 리소스를 보여줍니다.

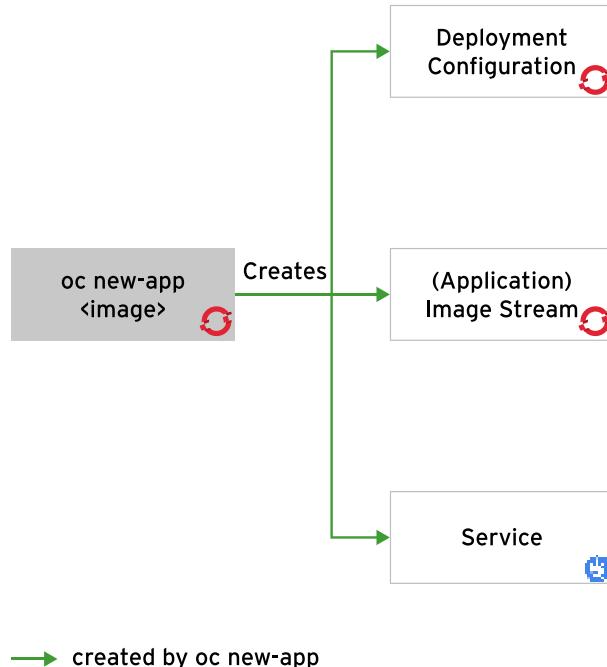


그림 6.4: 새로운 애플리케이션을 위해 생성된 리소스

다음 명령은 비공개 Docker 이미지 레지스트리에서 이미지를 기반으로 애플리케이션을 만듭니다.

```
oc new-app --docker-image=myregistry.com/mycompany/myapp --name=myapp --as-deployment-config
```

다음 명령은 Git 리포지토리에 저장된 소스 코드를 기반으로 애플리케이션을 만듭니다.

```
oc new-app https://github.com/openshift/ruby-hello-world --name=ruby-hello --as-deployment-config
```

S2I(Source-to-Image) 프로세스, 관련 개념 및 다음 섹션에서 OpenShift에 대한 애플리케이션을 빌드하기 위해 `oc new-app`을 사용하는 고급 방법에 대해 자세히 알아봅니다.

명령줄에서 OpenShift 리소스 관리

아래 설명된 대로 OpenShift 리소스를 관리하는데 사용하는 몇 가지 필수적인 명령이 있습니다.

oc get 명령은 클러스터의 리소스에 대한 정보를 검색하는데 사용합니다. 일반적으로 이 명령은 가장 중요한 리소스 특성만 출력하고 자세한 정보는 생략합니다.

oc get RESOURCE_TYPE 명령은 지정된 유형의 모든 리소스에 대한 요약을 표시합니다. 다음은 `oc get pods` 명령의 출력 예를 보여줍니다.

NAME	READY	STATUS	RESTARTS	AGE
nginx-1-5r583	1/1	Running	0	1h
myapp-1-l44m7	1/1	Running	0	1h

oc get all 실행의 샘플입니다.

`oc get all` 명령은 가장 중요한 클러스터 구성 요소에 대한 요약을 검색하는데 사용합니다. 이 명령은 현재 프로젝트에 대한 주요 리소스 유형을 반복한 다음 요약 정보를 출력합니다.

NAME	DOCKER REPO			TAGS	UPDATED
is/nginx	172.30.1.1:5000/basic-kubernetes/nginx			latest	About an hour ago
<hr/>					
NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY	
dc/nginx	1	1	1	config,image(nginx:latest)	
<hr/>					
NAME	DESIRED	CURRENT	READY	AGE	
rc/nginx-1	1	1	1	1h	
<hr/>					
NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
svc/nginx	172.30.72.75	<none>	80/TCP, 443/TCP	1h	
<hr/>					
NAME	READY	STATUS	RESTARTS	AGE	
po/nginx-1-ypp8t	1/1	Running	0	1h	

RESOURCE_TYPE RESOURCE_NAME을 설명하는 oc

`oc get`으로 제공하는 요약이 불충분한 경우 `oc describe` 명령을 사용하여 추가 정보를 검색합니다. `oc get` 명령과 달리 다양한 모든 리소스를 유형별로 반복할 수 없습니다. 대부분의 주요 리소스에 대해 설명할 수 있지만, 이 기능은 일부 리소스에서는 사용할 수 없습니다. 다음은 포드 리소스를 설명하는 예제 출력입니다.

Name:	mysql-openshift-1-glqrp
Namespace:	mysql-openshift
Priority:	0
PriorityClassName:	none
Node:	cluster-worker-1/172.25.250.52
Start Time:	Fri, 15 Feb 2019 02:14:34 +0000
Labels:	app=mysql-openshift deployment=mysql-openshift-1 deploymentconfig=mysql-openshift
Annotations:	openshift.io/deployment-config.latest-version: 1 openshift.io/deployment-config.name: mysql-openshift openshift.io/deployment.name: mysql-openshift-1 openshift.io/generated-by: OpenShiftNewApp openshift.io/scc: restricted
Status:	Running
IP:	10.129.0.85

oc get RESOURCE_TYPE RESOURCE_NAME -o yaml

이 명령은 리소스 정의를 내보내는 데 사용할 수 있습니다. 일반적인 사용 사례에는 백업 작성 또는 정의 수정을 지원하는 것이 포함됩니다. `-o yaml` 옵션은 YAML 형식으로 오브젝트 표현을 출력하지만 `-o json` 옵션을 제공하여 JSON 형식으로 변경할 수 있습니다.

oc create

이 명령은 리소스 정의에서 리소스를 생성합니다. 일반적으로 이 명령은 정의를 편집하는 `oc get RESOURCE_TYPE RESOURCE_NAME -o yaml` 명령과 쌍으로 결합됩니다.

oc edit

이 명령을 사용하면 리소스 정의의 리소스를 편집할 수 있습니다. 기본적으로 이 명령은 리소스 정의를 편집하기 위해 `vi` 편집기를 엽니다.

oc delete RESOURCE_TYPE name

`oc delete` 명령은 OpenShift 클러스터에서 리소스를 제거합니다. 포드 등의 관리된 리소스를 삭제하면 해당 리소스의 새로운 인스턴스가 자동으로 생성되므로 여기에서는 OpenShift 아키텍처에 대한 기본 지식이 있어야 합니다. 프로젝트가 삭제되면 포함된 모든 리소스와 애플리케이션이 삭제됩니다.

oc exec CONTAINER_ID 옵션 명령

`oc exec` 명령은 컨테이너 내에서 명령을 실행합니다. 이 명령을 사용하여 스크립트의 일부로 대화형 및 비대화형 배치 명령을 실행할 수 있습니다.

리소스 라벨 지정

동일한 프로젝트에서 다양한 리소스를 사용하여 작업하는 경우에는 대부분 애플리케이션, 환경 또는 기타 몇 가지 기준에 따라 해당 리소스를 그룹화하는 것이 유용합니다. 이러한 그룹을 설정하려면 프로젝트의 리소스에 레이블을 정의해야 합니다. 레이블은 리소스의 `metadata` 섹션 부분이며, 다음 예와 같이 키/값 쌍으로 정의됩니다.

```
apiVersion: v1
kind: Service
metadata:
...contents omitted...
labels:
  app: nexus
  template: nexus-persistent-template
  name: nexus
...contents omitted...
```

대부분의 `oc` 하위 명령에서는 `-l` 옵션을 지원하여 레이블 사양의 리소스를 처리합니다. `oc get` 명령의 경우 `-l` 옵션은 레이블이 일치하는 오브젝트만 검색하는 선택기 역할을 합니다.

```
$ oc get svc,dc -l app=nexus
NAME          TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/nexus ClusterIP  172.30.29.218  <none>        8081/TCP    4h

NAME                           REVISION      DESIRED      CURRENT      ...
deploymentconfig.apps.openshift.io/nexus  1           1           1           ...
```

**참고**

모든 레이블이 리소스에 표시될 수 있지만 **app**과 **template** 키는 레이블에 공통적입니다. 일반적으로 **app** 키는 이 리소스와 관련된 애플리케이션을 나타냅니다. **template** 키는 템플릿 이름과 동일한 템플릿으로 생성한 모든 리소스에 레이블을 지정합니다.

```
apiVersion: template.openshift.io/v1
kind: Template
labels:
  app: nexus
  template: nexus-persistent-template
metadata:
...contents omitted...
labels:
  maintainer: redhat
  name: nexus-persistent
...contents omitted...
objects:
- apiVersion: v1
  kind: Service
  metadata:
    name: nexus
  labels:
    version: 1
...contents omitted...
```

위 예에서는 **maintainer: redhat**이라는 단일 레이블이 있는 템플릿 리소스를 정의합니다. 이 템플릿에서는 세 가지 레이블 **app: nexus**, **template: nexus-persistent-template**, **version: 1**이 있는 서비스 리소스가 생성됩니다.

**참조**

포드 및 서비스에 관한 자세한 내용은 OpenShift Container Platform 설명서

아키텍처

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/architecture/index
의 포드 및 서비스 섹션에서 확인할 수 있습니다.

이미지 생성에 관한 자세한 내용은 OpenShift Container Platform 설명서에서 확인할 수 있습니다.

이미지 생성

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/images/index

레이블 및 레이블 선택기에 대한 자세한 내용은 Kubernetes 설명서의 Kubernetes 오브젝트 사용 섹션에서 확인할 수 있습니다.

레이블 및 선택기

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

▶ 연습 가이드

OpenShift에 데이터베이스 서버 배포

이 연습에서는 `oc new-app` 명령을 사용하여 OpenShift에서 MySQL 데이터베이스 포드를 생성하고 배포합니다.

결과

OpenShift에서 MySQL 데이터베이스 포드를 생성하고 배포할 수 있습니다.

시작하기 전에

`workstation`에서 다음 명령을 실행하여 환경을 설정합니다.

```
[student@workstation ~]$ lab openshift-resources start
```

▶ 1. 랩 환경을 준비합니다.

- 1.1. 강의실 환경 구성을 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. OpenShift 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. 이 연습에서 생성하는 리소스에 대한 RHOCP 개발자 사용자 이름을 포함하는 새 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-mysql-openshift
Now using project ...output omitted...
```

▶ 2. `oc new-app` 명령을 사용하여 `rhscl/mysql-57-rhel7` 컨테이너 이미지에서 새 애플리케이션을 생성합니다.

이 이미지를 사용하려면 `-e` 옵션을 사용하여 `MYSQL_USER`, `MYSQL_PASSWORD`, `MYSQL_DATABASE` 및 `MYSQL_ROOT_PASSWORD` 환경 변수를 설정해야 합니다.

OpenShift가 인터넷에서 이미지 가져오기를 시도하지 않도록 `--docker-image` 옵션을 `oc new-app` 명령과 함께 사용하여 강의실 프라이빗 레지스트리 URI를 지정합니다.

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> --docker-image=registry.access.redhat.com/rhscl/mysql-57-rhel7:latest \
> --name=mysql-openshift \
```

```
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e MYSQL_DATABASE=testdb \
> -e MYSQL_ROOT_PASSWORD=r00tpa55
--> Found Docker image b48e700 (5 weeks old) from registry.access.redhat.com for
"registry.access.redhat.com/rhscl/mysql-57-rhel7:latest"
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "mysql-openshift" created
deploymentconfig.apps.openshift.io "mysql-openshift" created
service "mysql-openshift" created
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/mysql-openshift'
Run 'oc status' to view your app.
```

- ▶ 3. MySQL 포드가 성공적으로 생성되었는지 확인하고 포드 및 서비스에 관한 세부 정보를 봅니다.

- 3.1. `oc status` 명령을 실행하여 새 애플리케이션의 상태를 보고 MySQL 이미지의 배포가 성공적인지 확인합니다.

```
[student@workstation ~]$ oc status
In project ${RHT_OCP4_DEV_USER}-mysql-openshift on server ...

svc/mysql-openshift - 172.30.114.39:3306
dc/mysql-openshift deploys istag/mysql-openshift:latest
  deployment #1 running for 11 seconds - 0/1 pods
...output omitted...
```

- 3.2. 이 프로젝트에 포드를 나열하여 MySQL 포드가 준비되고 실행되는지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
mysql-openshift-1-deploy  0/1     Completed   0          111s
mysql-openshift-1-xg665   1/1     Running    0          109s
```



참고

실행 중인 포드의 이름을 확인합니다. 나중에 MySQL 데이터베이스 서버에 로그인할 수 있으려면 이 정보가 필요합니다.

- 3.3. `oc describe` 명령을 사용하여 포드에 관한 자세한 정보를 봅니다.

```
[student@workstation ~]$ oc describe pod mysql-openshift-1-xg665
Name:           mysql-openshift-1-xg665
Namespace:      ${RHT_OCP4_DEV_USER}-mysql-openshift
Priority:       0
Node:           master01/192.168.50.10
Start Time:     Fri, 13 Nov 2020 08:50:03 -0500
Labels:         deployment=mysql-openshift-1
                  deploymentconfig=mysql-openshift
...output omitted...
```

```
Status:      Running
IP:          10.10.0.34
...output omitted...
```

- 3.4. 이 프로젝트에 서비스를 나열하고 MySQL 포드에 액세스하는 서비스가 생성되었는지 확인합니다.

```
[student@workstation ~]$ oc get svc
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP     PORT(S)      AGE
mysql-openshift ClusterIP  172.30.121.55   <none>        3306/TCP    10m
```

- 3.5. `oc describe` 명령을 사용하여 `mysql-openshift` 서비스의 세부 정보를 검색하고 서비스 유형이 기본적으로 `ClusterIP`인지 확인합니다.

```
[student@workstation ~]$ oc describe service mysql-openshift
Name:           mysql-openshift
Namespace:      ${RHT_OCP4_DEV_USER}-mysql-openshift
Labels:         app=mysql-openshift
                app.kubernetes.io/component=mysql-openshift
                app.kubernetes.io/instance=mysql-openshift
Annotations:   openshift.io/generated-by: OpenShiftNewApp
Selector:       deploymentconfig=mysql-openshift
Type:          ClusterIP
IP:            172.30.121.55
Port:          3306-tcp  3306/TCP
TargetPort:    3306/TCP
Endpoints:    10.10.0.34:3306
Session Affinity: None
Events:        <none>
```

- 3.6. 이 애플리케이션의 배포 구성(dc)에 관한 세부 정보를 봅니다.

```
[student@workstation ~]$ oc describe dc mysql-openshift
Name:           mysql-openshift
Namespace:      ${RHT_OCP4_DEV_USER}-mysql-openshift
Created:        15 minutes ago
Labels:         app=mysql-openshift
                app.kubernetes.io/component=mysql-openshift
                app.kubernetes.io/instance=mysql-openshift
...output omitted...
Deployment #1 (latest):
  Name:        mysql-openshift-1
  Created:    15 minutes ago
  Status:      Complete
  Replicas:    1 current / 1 desired
  Selector:   deployment=mysql-openshift-1,deploymentconfig=mysql-openshift
  Labels:      app.kubernetes.io/component=mysql-openshift,app.kubernetes.io/
               instance=mysql-openshift,app=mysql-openshift,openshift.io/deployment-
               config.name=mysql-openshift
  Pods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...output omitted...
```

- 3.7. 기본 이름 및 정규화된 도메인 이름(FQDN)으로 경로를 생성하는 서비스를 노출합니다.

```
[student@workstation ~]$ oc expose service mysql-openshift
route.route.openshift.io/mysql-openshift exposed
[student@workstation ~]$ oc get routes
NAME            HOST/PORT          ...   PORT
mysql-openshift mysql-openshift-$...-mysql...   ...   3306-tcp
```

- ▶ 4. MySQL 데이터베이스 서버에 연결하고 데이터베이스가 성공적으로 생성되었는지 확인합니다.

- 4.1. **workstation** 시스템에서 **workstation**과 포트 3306을 사용하여 OpenShift에서 실행되는 데이터베이스 포드 간의 포트 전달을 구성합니다. 명령을 실행하면 터미널이 멈춥니다.

```
[student@workstation ~]$ oc port-forward mysql-openshift-1-xg665 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

- 4.2. **workstation** 시스템에서 다른 터미널을 열고 MySQL 클라이언트를 사용하여 MySQL 서버에 연결합니다.

```
[student@workstation ~]$ mysql -uuser1 -pmypa55 --protocol tcp -h localhost
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

- 4.3. **testdb** 데이터베이스 생성을 확인합니다.

```
MySQL [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| testdb         |
+-----+
2 rows in set (0.00 sec)
```

- 4.4. MySQL 프롬프트를 종료합니다.

```
MySQL [(none)]> exit
Bye
```

터미널을 닫고 이전 항목으로 돌아갑니다. **Ctrl+C**를 눌러 포트 전달 프로세스를 완료합니다.

```
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
Handling connection for 3306
^C
```

- ▶ 5. 프로젝트 내의 모든 리소스를 제거하려면 프로젝트를 삭제합니다.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-mysql-openshift
```

완료

workstation에서 `lab openshift-resources finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab openshift-resources finish
```

이로써 연습이 완료됩니다.

경로 만들기

목표

이 섹션을 마치면 OpenShift 경로를 사용하여 서비스를 노출할 수 있습니다.

경로에 대한 작업

서비스를 통해 OpenShift 인스턴스 내부에서 포드 간 네트워크 액세스가 가능하고, 경로를 통해서는 OpenShift 인스턴스 외부의 사용자와 애플리케이션에서 포드에 대한 네트워크 액세스가 가능합니다.

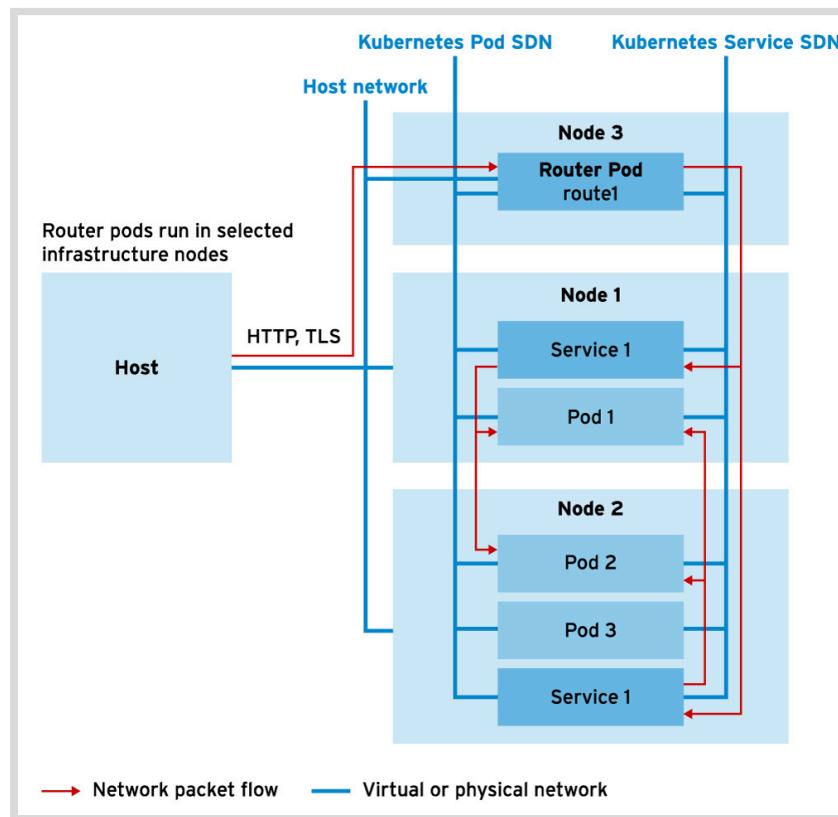


그림 6.5: OpenShift 경로 및 Kubernetes 서비스

경로는 공용 방향 IP 주소와 DNS 호스트 이름을 내부 방향 서비스 IP에 연결합니다. 서비스 리소스를 사용하여 서비스에서 노출하는 포트인 엔드 포인트를 찾습니다.

OpenShift 경로는 클러스터 수준의 라우터 서비스로 구현되며, 이 서비스는 OpenShift 클러스터에서 컨테이너화된 애플리케이션으로 실행됩니다. OpenShift는 다른 OpenShift 애플리케이션과 마찬가지로 라우터 포드를 확장하고 복제합니다.



참고

실제로 성능을 향상시키고 대기 시간을 줄이기 위해 OpenShift 라우터에서는 내부 포드 소프트웨어 정의 네트워크(SDN)를 사용하여 포드로 직접 연결합니다.

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

라우터 서비스는 HAProxy를 기본 구현으로 사용합니다.

OpenShift 관리자가 고려해야 할 중요한 점은 경로에 구성된 공용 DNS 호스트 이름이 라우터를 실행 중인 노드의 공용 방향 IP 주소를 가리켜야 한다는 것입니다. 일반 애플리케이션 포드와 달리 라우터 포드는 내부 포드 SDN이 아니라 노드의 공용 IP 주소에 바인드됩니다.

다음 예에서는 JSON 구문을 사용하여 정의된 최단 경로를 보여줍니다.

```
{
    "apiVersion": "v1",
    "kind": "Route",
    "metadata": {
        "name": "quoteapp"
    },
    "spec": {
        "host": "quoteapp.apps.example.com",
        "to": {
            "kind": "Service",
            "name": "quoteapp"
        }
    }
}
```

apiVersion, **kind** 및 **metadata** 속성은 표준 Kubernetes 리소스 정의 규칙을 따릅니다. **kind**의 **Route** 값은 이 속성이 경로 리소스임을 나타내며 **metadata.name** 속성은 이 특정 경로에 **quoteapp** 식별자를 제공합니다.

포드 및 서비스에서 주요 부분은 다음 속성을 포함하는 오브젝트인 **spec** 속성입니다.

- **host**는 경로와 연결된 FQDN을 포함하는 문자열입니다. DNS는 이 FQDN을 OpenShift 라우터의 IP 주소로 변환해야 합니다. DNS 구성 설정에 대한 세부 사항은 이 교육 과정의 범위를 벗어납니다.
- **to**는 이 경로가 가리키는 리소스를 나타내는 오브젝트입니다. 이 경우 경로는 **name**이 **quoteapp**으로 설정된 OpenShift 서비스를 가리킵니다.



참고

서로 다른 리소스 유형의 이름은 충돌하지 않습니다. **quoteapp**이라는 서비스를 나타내는 **quoteapp**이라는 경로가 있는 것은 전혀 문제가 되지 않습니다.



중요

선택기를 사용하여 특정 레이블을 포함하는 포드 리소스에 링크하는 서비스와 달리, 경로는 서비스 리소스 이름에 직접 링크합니다.

경로 만들기

oc create 명령을 사용하여 다른 OpenShift 리소스와 마찬가지로 경로 리소스를 만듭니다. 경로를 정의하는 JSON 또는 YAML 리소스 정의 파일을 **oc create** 명령에 제공해야 합니다.

컨테이너 이미지, Dockerfile 또는 애플리케이션 소스 코드에서 포드를 빌드할 때 **oc new-app** 명령은 경로 리소스를 생성하지 않습니다. 결국, **oc new-app**에서는 포드가 OpenShift 인스턴스 외부에서 액세스 가능해야 하는지를 알지 못합니다.

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

경로를 생성하는 또 다른 방법은 **oc expose service** 명령을 사용하여 서비스 리소스 이름을 입력으로 전달하는 것입니다. **--name** 옵션은 경로 리소스의 이름을 제어하는데 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ oc expose service quotedb --name quote
```

기본적으로 **oc expose**에 의해 생성된 경로는 양식의 DNS 이름을 생성합니다.

route-name-project-name.default-domain

여기서 각 항목은 다음을 나타냅니다.

- **route-name**은 경로에 할당된 이름입니다. 명시적 이름이 설정되지 않은 경우 OpenShift는 경로에 원래 리소스와 동일한 이름을 할당합니다(예: 서비스 이름).
- **project-name**은 리소스를 포함하는 프로젝트의 이름입니다.
- **default-domain**은 OpenShift 마스터에 구성되며, OpenShift를 설치하기 위한 사전 요구 사항으로 나열된 와일드카드 DNS 도메인에 해당합니다.

예를 들어, 와일드카드 도메인이 **cloudapps.example.com**인 OpenShift 인스턴스에서 **test**라는 프로젝트에 **quote**라는 경로를 생성하면 FQDN이 **quote-test.cloudapps.example.com**이 됩니다.



참고

와일드카드 도메인을 호스팅하는 DNS 서버는 경로 호스트 이름을 인지하지 못합니다. 이 서버는 이름을 구성된 IP 주소로 변환하기만 합니다. OpenShift 라우터만 경로 호스트 이름을 알고 있으며, 각각을 HTTP 가상 호스트로 처리합니다. OpenShift 라우터는 해당하는 경로가 없는 올바르지 않은 와일드카드 도메인 호스트 이름을 차단하고, HTTP 404 오류를 반환합니다.

기본 라우팅 서비스 활용

기본 라우팅 서비스는 **HAProxy** 포드로 구현됩니다. 라우터 포드, 컨테이너 및 해당 구성은 OpenShift 클러스터의 기타 모든 리소스와 마찬가지로 검사할 수 있습니다.

```
$ oc get pod --all-namespaces -l app=router
NAMESPACE          NAME                READY   STATUS    RESTARTS   AGE
openshift-ingress router-default-746b5cfb65-f6sdm 1/1     Running   1          4d
```

기본적으로 라우터는 **openshift-ingress** 프로젝트에 배포됩니다. **oc describe pod** 명령을 사용하여 라우팅 구성 세부 정보를 가져옵니다.

```
$ oc describe pod router-default-746b5cfb65-f6sdm
Name:           router-default-746b5cfb65-f6sdm
Namespace:      openshift-ingress
...output omitted...
Containers:
  router:
  ...output omitted...
Environment:
  STATS_PORT:          1936
  ROUTER_SERVICE_NAMESPACE:  openshift-ingress
  DEFAULT_CERTIFICATE_DIR: /etc/pki/tls/private
```

```
ROUTER_SERVICE_NAME:      default
ROUTER_CANONICAL_HOSTNAME: apps.cluster.lab.example.com
...output omitted...
```

모든 기본 경로에 사용되는 하위 도메인 또는 기본 도메인에는 **ROUTER_CANONICAL_HOSTNAME** 항목의 값이 사용됩니다.



참조

OpenShift의 경로 아키텍처에 대한 자세한 내용은 아키텍처 및 개발자 가이드 섹션에서 확인할 수 있습니다.

OpenShift Container Platform 설명서

https://access.redhat.com/documentation/en-us/openshift_container_platform/

▶ 연습 가이드

서비스를 경로로 노출

이 연습에서는 OpenShift 클러스터에서 애플리케이션을 생성, 빌드 및 배포하고 서비스를 경로로 노출합니다.

결과

배포된 OpenShift 애플리케이션에 대해 서비스를 경로로 노출할 수 있습니다.

시작하기 전에

workstation에서 **student** 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab openshift-routes start
```

▶ 1. 랩 환경을 준비합니다.

1. 강의실 환경 구성을 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

2. OpenShift 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

3. 이 연습에서 생성하는 리소스에 대한 RHOCP 개발자 사용자 이름을 포함하는 새 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-route
```

▶ 2. [http://github.com/\\${RHT_OCP4_GITHUB_USER}/D0180-apps/](http://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps/)에 있는 Git 리포지토리의 **php-helloworld** 디렉터리에서 S2I(Source-to-Image)를 사용하여 새 PHP 애플리케이션을 생성합니다.

1. **oc new-app** 명령을 사용하여 PHP 애플리케이션을 생성합니다.



중요

다음 예는 역슬래시(\)를 사용하여 두 번째 행이 첫 번째 행의 연속임을 나타냅니다. 역슬래시를 무시하고 싶다면 전체 명령을 한 행에 입력할 수 있습니다.

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> php:7.3~https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps \
> --context-dir php-helloworld --name php-helloworld
--> Found image fbe3911 (13 days old) in image stream "openshift/php" under tag
"7.3" for "php:7.3"
...output omitted...
--> Creating resources ...
...output omitted...
--> Success
Build scheduled, use 'oc logs -f bc/php-helloworld' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/php-helloworld'
Run 'oc status' to view your app.
```

2.2. **oc get pods -w** 명령으로 진행 상황을 모니터링하여 애플리케이션이 빌드와 배포를 완료 할 때까지 기다립니다.

```
[student@workstation ~]$ oc get pods -w
NAME           READY   STATUS    RESTARTS   AGE
php-helloworld-1-build  0/1     Init:0/2   0          2s
php-helloworld-1-build  0/1     Init:0/2   0          4s
php-helloworld-1-build  0/1     Init:1/2   0          5s
php-helloworld-1-build  0/1     PodInitializing   0          6s
php-helloworld-1-build  1/1     Running   0          7s
php-helloworld-1-deploy 0/1     Pending   0          0s
php-helloworld-1-deploy 0/1     Pending   0          0s
php-helloworld-1-deploy 0/1     ContainerCreating 0          0s
php-helloworld-1-build  0/1     Completed   0          5m8s
php-helloworld-1-cnphm  0/1     Pending   0          0s
php-helloworld-1-cnphm  0/1     Pending   0          1s
php-helloworld-1-deploy 1/1     Running   0          4s
php-helloworld-1-cnphm  0/1     ContainerCreating 0          1s
php-helloworld-1-cnphm  1/1     Running   0          62s
php-helloworld-1-deploy 0/1     Completed   0          65s
php-helloworld-1-deploy 0/1     Terminating 0          66s
php-helloworld-1-deploy 0/1     Terminating 0          66s
^C
```

정확한 출력은 이름, 상태, 타이밍 및 순서가 다를 수 있습니다. 애플리케이션이 성공적으로 배포되었음을 나타내는 **deploy** 접미사를 사용하여 **Completed** 컨테이너를 찾습니다. 임의의 접미사(예의 **cnphm**)가 붙은 **Running** 상태의 컨테이너는 해당 애플리케이션이 포함되어 있으며 애플리케이션이 실행 중임을 보여줍니다.

또는 각각 **oc logs -f bc/php-helloworld** 및 **oc logs -f dc/php-helloworld** 명령을 사용하여 빌드 및 배포 로그를 모니터링합니다. 필요하면 **Ctrl+C**를 눌러 명령을 종료합니다.

```
[student@workstation ~]$ oc logs -f bc/php-helloworld
Cloning "https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps" ...
Commit: f7cd8963ef353d9173c3a21dcccf402f3616840b (Initial commit, including all
apps previously in course)
```

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

```

...output omitted...
STEP 7: USER 1001
STEP 8: RUN /usr/libexec/s2i/assemble
---> Installing application source...
...output omitted...
Push successful
[student@workstation ~]$ oc logs -f dc/php-helloworld
-> Cgroups memory limit is set, using HTTPD_MAX_REQUEST_WORKERS=136
=> sourcing 20-copy-config.sh ...
...output omitted...
[core:notice] [pid 1] AH00094: Command line: 'httpd -D FOREGROUND'
^C

```

정확한 출력은 다를 수 있습니다.

2.3. `oc describe` 명령을 사용하여 이 애플리케이션에 대한 서비스를 검토합니다.

```

[student@workstation ~]$ oc describe svc/php-helloworld
Name:           php-helloworld
Namespace:      ${RHT_OCP4_DEV_USER}-route
Labels:          app=php-helloworld
                 app.kubernetes.io/component=php-helloworld
                 app.kubernetes.io/instance=php-helloworld
Annotations:    openshift.io/generated-by: OpenShiftNewApp
Selector:        deploymentconfig=php-helloworld
Type:           ClusterIP
IP:             172.30.228.124
Port:           8080-tcp  8080/TCP
TargetPort:     8080/TCP
Endpoints:      10.10.0.35:8080
Port:           8443-tcp  8443/TCP
TargetPort:     8443/TCP
Endpoints:      10.10.0.35:8443
Session Affinity: None
Events:         <none>

```

명령의 출력에 표시된 IP 주소는 다를 수 있습니다.

- ▶ 3. 경로를 생성하는 서비스를 노출합니다. 기본 이름과 FQDN(정규화된 도메인 이름)을 경로로 사용합니다.

```

[student@workstation ~]$ oc expose svc/php-helloworld
route.route.openshift.io/php-helloworld exposed
[student@workstation ~]$ oc describe route
Name:           php-helloworld
Namespace:      ${RHT_OCP4_DEV_USER}-route
Created:        5 seconds ago
Labels:          app=php-helloworld
                 app.kubernetes.io/component=php-helloworld
                 app.kubernetes.io/instance=php-helloworld
Annotations:    openshift.io/host.generated=true
Requested Host: php-helloworld-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDC...
                 exposed on router default (host ${RHT_OCP4_WILDCARD_DOMAIN}) 4 seconds ago
Path:           <none>
TLS Termination: <none>

```

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

```
Insecure Policy: <none>
Endpoint Port: 8080-tcp

Service: php-helloworld
Weight: 100 (100%)
Endpoints: 10.10.0.35:8443, 10.10.0.35:8080
```

- ▶ 4. 클러스터 외부의 호스트에서 서비스에 액세스하여 서비스 및 경로가 작동하는지 확인합니다.

```
[student@workstation ~]$ curl \
> php-helloworld-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World! php version is 7.3.11
```

**참고**

PHP 애플리케이션의 출력은 Git 리포지토리의 실제 코드에 따라 다릅니다. 이전 섹션에서 코드를 업데이트한 경우에는 다를 수 있습니다.

기본적으로 FQDN은 애플리케이션 이름 및 프로젝트 이름으로 구성됩니다. FQDN의 나머지인 하위 도메인은 OpenShift가 설치될 때 정의됩니다.

- ▶ 5. 이 경로를 xyz라는 경로로 바꿉니다.

- 5.1. 현재 경로를 삭제합니다.

```
[student@workstation ~]$ oc delete route/php-helloworld
route.route.openshift.io "php-helloworld" deleted
```

**참고**

경로 삭제는 선택 사항입니다. 이름이 다른 경우 동일한 서비스에 대해 여러 개의 경로가 있을 수 있습니다.

- 5.2. 이름이 \${RHT_OCP4_DEV_USER}-xyz인 서비스의 경로를 만듭니다.

```
[student@workstation ~]$ oc expose svc/php-helloworld \
> --name=${RHT_OCP4_DEV_USER}-xyz
route.route.openshift.io/${RHT_OCP4_DEV_USER}-xyz exposed
[student@workstation ~]$ oc describe route
Name: ${RHT_OCP4_DEV_USER}-xyz
Namespace: ${RHT_OCP4_DEV_USER}-route
Created: 5 seconds ago
Labels: app=php-helloworld
        app.kubernetes.io/component=php-helloworld
        app.kubernetes.io/instance=php-helloworld
Annotations: openshift.io/host.generated=true
Requested Host: ${RHT_OCP4_DEV_USER}-xyz-${RHT_OCP4_DEV_USER}-route.${RHT_...
              exposed on router default (host ${RHT_OCP4_WILDCARD_DOMAIN}) 4 seconds ago
Path: <none>
TLS Termination: <none>
Insecure Policy: <none>
```

```
Endpoint Port: 8080-tcp
Service: php-helloworld
Weight: 100 (100%)
Endpoints: 10.10.0.35:8443, 10.10.0.35:8080
```

새 경로 이름을 기반으로 생성된 새 FQDN을 확인합니다. 사용자 이름은 경로 이름 및 프로젝트 이름 모두에 포함되므로 경로 FQDN에 두 번 표시됩니다.

- 5.3. 포트 80에서 FQDN을 사용하여 HTTP 요청을 만듭니다.

```
[student@workstation ~]$ curl \
> ${RHT_OCP4_DEV_USER}-xyz-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World! php version is 7.3.11
```

완료

workstation에서 `lab openshift-routes finish` 스크립트를 실행하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab openshift-routes finish
```

이로써 안내에 따른 연습이 완료됩니다.

S2I(Source-to-Image)로 애플리케이션 생성

목표

이 섹션을 마친 수강생은 OpenShift Container Platform의 S2I(Source-to-Image) 기능을 사용하여 애플리케이션을 배포할 수 있습니다.

S2I(Source-to-Image) 프로세스

S2I(Source-to-Image)는 애플리케이션 소스 코드로 컨테이너 이미지를 쉽게 빌드할 수 있는 툴입니다. 이 툴은 Git 리포지토리에서 애플리케이션의 소스 코드를 가져와 원하는 언어 및 프레임워크를 기반으로 기본 컨테이너에 소스 코드를 삽입하고 조합된 애플리케이션을 실행하는 새 컨테이너 이미지를 생성합니다.

그림 6.6은 인수가 애플리케이션 소스 코드 리포지토리인 경우 `oc new-app` 명령으로 생성된 리소스를 보여줍니다. S2I에서도 배포 구성 및 모든 종속 리소스가 생성됩니다.

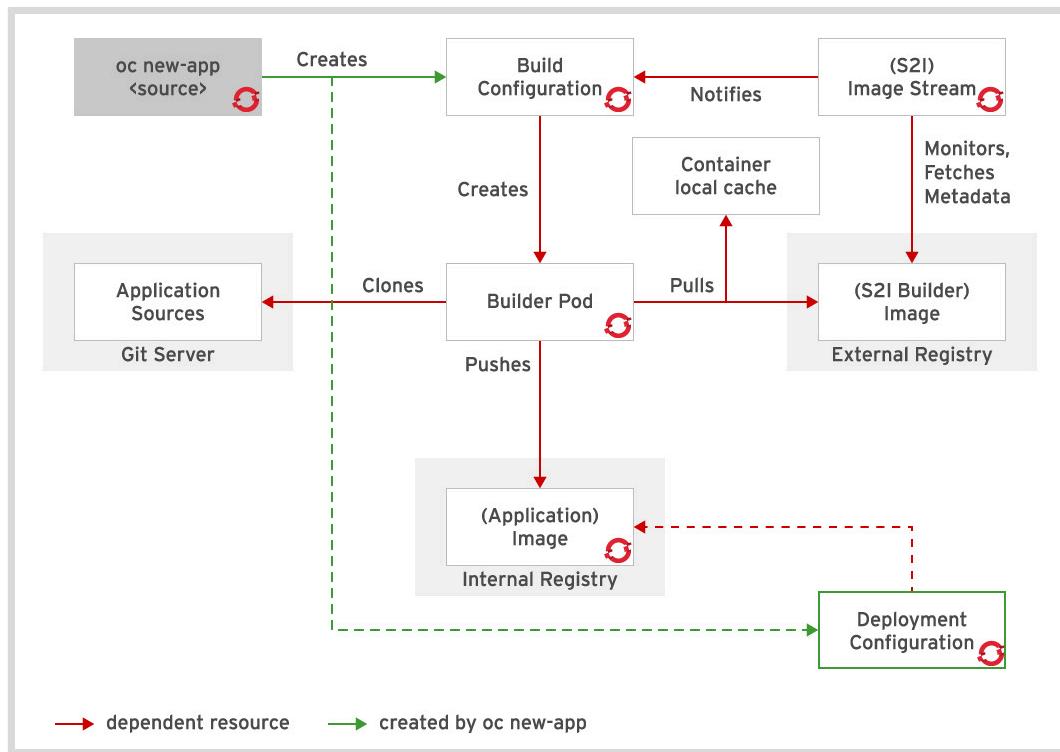


그림 6.6: 배포 구성 및 종속 리소스

S2I는 OpenShift Container Platform에 애플리케이션을 빌드하는 데 사용하는 주요 전략입니다. 소스 빌드를 사용하는 주된 이유는 다음과 같습니다.

- **사용자 효율성:** 개발자가 Dockerfile 및 yum install 등의 운영 체제 명령을 이해할 필요가 없습니다. 표준 프로그래밍 언어 툴을 사용하여 작업합니다.
- **패칭:** 보안 문제로 인해 기본 이미지에 패치가 필요한 경우 S2I를 사용하면 모든 애플리케이션을 지속적으로 다시 빌드할 수 있습니다. 예를 들어 PHP 기본 이미지에 보안 문제가 있는 경우 보안 패치로 이 이미지를 업데이트하면 이 이미지를 기본으로 사용하는 모든 애플리케이션을 업데이트합니다.

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

- 속도: S2I를 사용하면 각 단계에서 새 계층을 생성하지 않고 조합 프로세스를 통해 다수의 복합 작업을 수행할 수 있으므로 더욱 빨리 빌드할 수 있습니다.
- 에코시스템: S2I는 기본 이미지 및 스크립트를 여러 유형의 애플리케이션에 맞게 수정 가능하며 재사용할 수 있는 이미지의 공유 에코시스템을 권장합니다.

이미지 스트림 설명

OpenShift는 새로운 버전의 사용자 애플리케이션을 신속하게 포드에 배포합니다. 새 애플리케이션을 생성 하려면 애플리케이션 소스 코드 외에도 기본 이미지(S2I 빌더 이미지)가 필요합니다. 이러한 두 구성 요소 중 하나가 업데이트되면 OpenShift에서 새 컨테이너 이미지가 생성됩니다. 이전 컨테이너 이미지를 사용하여 생성된 포드는 새 이미지를 사용하여 생성된 포드로 대체됩니다.

애플리케이션 코드를 변경하면 컨테이너 이미지를 업데이트해야 하는 것은 확실하지만, 빌더 이미지를 변경하는 경우 배포한 포드도 업데이트해야 하는지는 명확하지 않을 수 있습니다.

이미지 스트림 리소스는 이러한 컨테이너 이미지의 별칭인 이미지 스트림 태그와 관련된 특정 컨테이너 이미지의 이름을 지정하는 구성입니다. OpenShift는 이미지 스트림에 대해 애플리케이션을 빌드합니다. OpenShift 설치 프로그램은 설치 동안 기본적으로 몇 개의 이미지 스트림을 채웁니다. 사용 가능한 이미지 스트림을 결정하려면 다음과 같이 **oc get** 명령을 사용합니다.

```
$ oc get is -n openshift
NAME          IMAGE REPOSITORY           TAGS
cli           ...svc:5000/openshift/cli   latest
dotnet        ...svc:5000/openshift/dotnet  2.0,2.1,latest
dotnet-runtime ...svc:5000/openshift/dotnet-runtime  2.0,2.1,latest
httpd         ...svc:5000/openshift/httpd    2.4,latest
jenkins       ...svc:5000/openshift/jenkins  1,2
mariadb       ...svc:5000/openshift/mariadb  10.1,10.2,latest
mongodb       ...svc:5000/openshift/mongodb  2.4,2.6,3.2,3.4,3.6,latest
mysql         ...svc:5000/openshift/mysql    5.5,5.6,5.7,latest
nginx         ...svc:5000/openshift/nginx    1.10,1.12,1.8,latest
nodejs        ...svc:5000/openshift/nodejs   0.10,10,11,4,6,8,latest
perl          ...svc:5000/openshift/perl     5.16,5.20,5.24,5.26,latest
php           ...svc:5000/openshift/php      5.5,5.6,7.0,7.1,latest
postgresql    ...svc:5000/openshift/postgresql  10,9.2,9.4,9.5,9.6,latest
python        ...svc:5000/openshift/python   2.7,3.3,3.4,3.5,3.6,latest
redis         ...svc:5000/openshift/redis    3.2,latest
ruby          ...svc:5000/openshift/ruby    2.0,2.2,2.3,2.4,2.5,latest
wildfly       ...svc:5000/openshift/wildfly  10.0,10.1,11.0,12.0,...
```



참고

OpenShift 인스턴스에는 로컬 추가 및 OpenShift 포인트 릴리스에 따라 이미지 스트림이 많거나 적을 수 있습니다.

OpenShift에는 이미지 스트림이 변경되는 시기를 탐지하여 해당 변경에 따라 조치를 취합니다.

nodejs-010-rhel7 이미지에서 보안 문제가 발생하면 이미지 리포지토리에서 업데이트할 수 있고, OpenShift가 애플리케이션 코드의 새 빌드를 자동으로 트리거할 수 있습니다.

조직이 Red Hat에서 지원되는 여러 기본 S2I 이미지를 선택할 가능성이 크지만, 조직 고유의 기반 이미지도 생성할 수 있습니다.

S2I 및 CLI를 사용하여 애플리케이션 빌드

S2I로 애플리케이션 빌드는 OpenShift CLI를 사용하여 수행할 수 있습니다.

애플리케이션은 CLI의 `oc new-app` 명령과 함께 S2I 프로세스를 사용하여 생성할 수 있습니다.

```
$ oc new-app ❶--as-deployment-config ❷php~http://my.git.server.com/my-app❸
  --name=myapp❹
```

- ❶ 배포 대신 DeploymentConfig 생성
- ❷ 프로세스에 사용된 이미지 스트림은 털드(~)의 왼쪽에 나타납니다.
- ❸ 털드 다음에 있는 URL은 소스 코드의 Git 리포지토리 위치를 나타냅니다.
- ❹ 애플리케이션 이름을 설정합니다.



참고

물결표를 사용하는 대신 `-i` 옵션을 사용하여 이미지 스트림을 설정할 수 있습니다.

```
$ oc new-app --as-deployment-config -i php http://services.lab.example.com/app
  --name=myapp
```

`oc new-app` 명령을 사용하면 로컬 또는 원격 Git 리포지토리의 소스 코드를 사용하여 애플리케이션을 생성할 수 있습니다. 소스 리포지토리만 지정된 경우 `oc new-app`에서 애플리케이션 빌드에 사용해야 하는 올바른 이미지 스트림을 식별하려고 시도합니다. 애플리케이션 코드 외에도 S2I는 Dockerfile을 식별하고 처리하여 새 이미지를 생성할 수도 있습니다.

다음 예에서는 현재 디렉터리에 Git 리포지토리를 사용하여 애플리케이션을 생성합니다.

```
$ oc new-app --as-deployment-config .
```



중요

로컬 Git 리포지토리를 사용하는 경우 리포지토리에는 OpenShift 인스턴스에서 액세스할 수 있는 URL을 가리키는 원격 원본이 있어야 합니다.

원격 Git 리포지토리와 컨텍스트 하위 디렉터리를 사용하여 애플리케이션을 생성할 수도 있습니다.

```
$ oc new-app --as-deployment-config \
  https://github.com/openshift/sti-ruby.git \
  --context-dir=2.0/test/puma-test-app
```

마지막으로, 특정 분기 참조가 있는 원격 Git 리포지토리를 사용하여 애플리케이션을 생성할 수 있습니다.

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

명령에 이미지 스트림이 지정되지 않은 경우, `new-app`에서는 리포지토리의 루트에 있는 특정 파일을 기반으로 사용할 언어 빌더를 판별합니다.

언어	파일
Ruby	Rakefile Gemfile config.ru
Java EE	pom.xml
Node.js	app.json package.json
PHP	index.php composer.json
Python	requirements.txt config.py
Perl	index.pl cpanfile

언어를 탐지하고 나면 **new-app** 명령에서 탐지된 언어에 대해 지원하는 이미지 스트림 태그나 탐지된 언어의 이름과 일치하는 이미지 스트림을 검색합니다.

-o json 매개 변수 및 출력 리디렉션을 사용하여 JSON 리소스 정의 파일을 만듭니다.

```
$ oc -o json new-app --as-deployment-config \
> php~http://services.lab.example.com/app \
> --name=myapp > s2i.json
```

이 JSON 정의 파일은 리소스 목록을 생성합니다. 첫 번째 리소스는 이미지 스트림입니다.

```
...output omitted...
{
    "kind": "ImageStream", ❶
    "apiVersion": "image.openshift.io/v1",
    "metadata": {
        "name": "myapp", ❷
        "creationTimestamp": null
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "lookupPolicy": {
            "local": false
        }
    },
    "status": {
        "dockerImageRepository": ""
    }
},
...output omitted...
```

❶ 이미지 스트림의 리소스 유형을 정의합니다.

❷ 이미지 스트림의 이름을 myapp으로 지정합니다.

빌드 구성(bc)은 소스 코드를 실행 가능한 이미지로 전환하기 위해 실행되는 입력 매개 변수 및 트리거 정의를 담당합니다. **BuildConfig**(BC)는 두 번째 리소스이며 다음 예는 OpenShift에서 실행 가능한 이미지를 생성하는 데 사용하는 매개 변수의 개요를 제공합니다.

```
...output omitted...
{
  "kind": "BuildConfig", ❶
  "apiVersion": "build.openshift.io/v1",
  "metadata": {
    "name": "myapp", ❷
    "creationTimestamp": null,
    "labels": {
      "app": "myapp"
    },
    "annotations": {
      "openshift.io/generated-by": "OpenShiftNewApp"
    }
  },
  "spec": {
    "triggers": [
      {
        "type": "GitHub",
        "github": {
          "secret": "S5_4BZpPabM6KrIuPBvI"
        }
      },
      {
        "type": "Generic",
        "generic": {
          "secret": "3q8K8JNDoRzhjoz1KgMz"
        }
      },
      {
        "type": "ConfigChange"
      },
      {
        "type": "ImageChange",
        "imageChange": {}
      }
    ],
    "source": {
      "type": "Git",
      "git": {
        "uri": "http://services.lab.example.com/app" ❸
      }
    },
    "strategy": {
      "type": "Source", ❹
      "sourceStrategy": {
        "from": {
          "kind": "ImageStreamTag",
          "namespace": "openshift",
          "name": "php:7.3" ❺
        }
      }
    }
  }
}
```

```

        }
    },
    "output": {
        "to": {
            "kind": "ImageStreamTag",
            "name": "myapp:latest" ❻
        }
    },
    "resources": {},
    "postCommit": {},
    "nodeSelector": null
},
"status": {
    "lastVersion": 0
}
},
...output omitted...

```

- ❶ **BuildConfig**의 리소스 유형을 정의합니다.
- ❷ **BuildConfig**의 이름을 **myapp**으로 지정합니다.
- ❸ 소스 코드 Git 리포지토리의 주소를 정의합니다.
- ❹ S2I를 사용하도록 전략을 정의합니다.
- ❺ 빌더 이미지를 php:7.3 이미지 스트림으로 정의합니다.
- ❻ 출력 이미지 스트림의 이름을 **myapp:latest**로 지정합니다.

세 번째 리소스는 배포 프로세스를 OpenShift에 맞도록 사용자 지정하는 일을 담당하는 배포 구성입니다. 새 컨테이너 인스턴스를 생성하는데 필요하며 Kubernetes에서 복제 컨트롤러로 변환되는 매개 변수 및 트리거를 포함할 수 있습니다. 다음은 **DeploymentConfig** 오브젝트에서 제공하는 일부 기능입니다.

- 기존 배포에서 새 배포로 전환하기 위한 사용자 지정 전략
- 이전 배포로 룰백합니다.
- 수동 복제 확장

```

...output omitted...
{
    "kind": "DeploymentConfig", ❶
    "apiVersion": "apps.openshift.io/v1",
    "metadata": {
        "name": "myapp", ❷
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "strategy": {
            "resources": {}
        },
        "triggers": [
            {

```

```

        "type": "ConfigChange" ③
    },
    {
        "type": "ImageChange", ④
        "imageChangeParams": {
            "automatic": true,
            "containerNames": [
                "myapp"
            ],
            "from": {
                "kind": "ImageStreamTag",
                "name": "myapp:latest"
            }
        }
    ],
    "replicas": 1,
    "test": false,
    "selector": {
        "app": "myapp",
        "deploymentconfig": "myapp"
    },
    "template": {
        "metadata": {
            "creationTimestamp": null,
            "labels": {
                "app": "myapp",
                "deploymentconfig": "myapp"
            },
            "annotations": {
                "openshift.io/generated-by": "OpenShiftNewApp"
            }
        },
        "spec": {
            "containers": [
                {
                    "name": "myapp",
                    "image": "myapp:latest", ⑤
                    "ports": [ ⑥
                        {
                            "containerPort": 8080,
                            "protocol": "TCP"
                        },
                        {
                            "containerPort": 8443,
                            "protocol": "TCP"
                        }
                    ],
                    "resources": {}
                }
            ]
        }
    },
    "status": {

```

```

        "latestVersion": 0,
        "observedGeneration": 0,
        "replicas": 0,
        "updatedReplicas": 0,
        "availableReplicas": 0,
        "unavailableReplicas": 0
    }
},
...output omitted...

```

- ❶ DeploymentConfig의 리소스 유형을 정의합니다.
- ❷ DeploymentConfig의 이름을 myapp으로 지정합니다.
- ❸ 구성 변경을 트리거하면 새로운 배포가 생성되며 언제든 복제 컨트롤러 템플릿이 변경됩니다.
- ❹ 리포지토리에서 새 버전의 myapp:latest 이미지를 사용할 수 있을 때마다 이미지 변경을 트리거하면 새 배포가 생성됩니다.
- ❺ 배포할 컨테이너 이미지를 myapp:latest로 정의합니다.
- ❻ 컨테이너 포트를 지정합니다.

마지막 항목은 서비스이며, 이미 이전 장에서 내용을 다루었습니다.

```

...output omitted...
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp",
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "ports": [
            {
                "name": "8080-tcp",
                "protocol": "TCP",
                "port": 8080,
                "targetPort": 8080
            },
            {
                "name": "8443-tcp",
                "protocol": "TCP",
                "port": 8443,
                "targetPort": 8443
            }
        ],
        "selector": {
            "app": "myapp",
            "deploymentconfig": "myapp"
        }
    },
}

```

```

    "status": {
      "loadBalancer": {}
    }
}

```

**참고**

기본적으로 `oc new-app` 명령은 경로를 생성하지 않습니다. 경로는 애플리케이션 생성 후 생성할 수 있습니다. 그러나 템플릿을 사용하므로 웹 콘솔 사용 시 경로가 자동으로 생성됩니다.

새 애플리케이션을 생성한 후 빌드 프로세스가 시작됩니다. `oc get builds` 명령을 사용하여 애플리케이션 빌드 목록을 확인합니다.

```
$ oc get builds
NAME          TYPE   FROM        STATUS     STARTED           DURATION
php-helloworld-1  Source  Git@9e17db8  Running  13 seconds ago
```

OpenShift를 사용하여 빌드 로그를 볼 수 있습니다. 다음 명령은 빌드 로그의 마지막 몇 줄을 보여줍니다.

```
$ oc logs build/myapp-1
```

**중요**

빌드가 아직 `Running`이 아니거나 OpenShift에서 `s2i-build` 포드를 아직 배포하지 않은 경우 위의 명령에서 오류가 발생합니다. 잠시 기다렸다가 다시 시도합니다.

`oc start-build build_config_name` 명령으로 새 빌드를 트리거합니다.

```
$ oc get buildconfig
NAME          TYPE   FROM        LATEST
myapp         Source  Git         1
```

```
$ oc start-build myapp
build "myapp-2" started
```

빌드 및 배포 구성 간의 관계

`BuildConfig` 포드는 OpenShift에서 이미지를 생성하고 내부 Docker 레지스트리로 내보내는 일을 담당합니다. 모든 소스 코드 또는 콘텐츠 업데이트에는 일반적으로 새 빌드가 이미지 업데이트를 보장해야 합니다.

`DeploymentConfig` 포드는 포드를 OpenShift에 배포하는 일을 담당합니다. `DeploymentConfig` 포드를 실행하면 내부 컨테이너 레지스트리에 배포된 이미지를 사용하여 포드가 생성됩니다. 기존에 실행 중인 포드는 `DeploymentConfig` 리소스 설정 방법에 따라 삭제할 수 있습니다.

`BuildConfig` 및 `DeploymentConfig` 리소스는 직접 상호 작용하지 않습니다. `BuildConfig` 리소스는 컨테이너 이미지를 생성하거나 업데이트합니다. `DeploymentConfig`는 이 새 이미지 또는 업데이트된 이미지 이벤트에 반응하며 컨테이너 이미지에서 포드를 생성합니다.



참조

S2I(Source-to-Image) 빌드

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/builds/build-strategies#build-strategy-s2i_build-strategies

S2I GitHub 리포지토리

<https://github.com/openshift/source-to-image>

▶ 연습 가이드

S2I(Source-to-Image)로 컨테이너화된 애플리케이션 생성

이 연습에서는 소스 코드에서 애플리케이션을 빌드하고, OpenShift 클러스터에 애플리케이션을 배포합니다.

결과

다음을 수행할 수 있습니다.

- OpenShift 명령줄 인터페이스를 사용하여 소스 코드에서 애플리케이션을 빌드합니다.
- OpenShift 명령줄 인터페이스를 사용하여 애플리케이션의 성공적인 배포를 확인합니다.

시작하기 전에

다음 명령을 실행하여 관련 랩 파일을 다운로드하고 환경을 구성합니다.

```
[student@workstation ~]$ lab openshift-s2i start
```

- ▶ 1. 샘플 애플리케이션의 PHP 소스 코드를 검사하고 이 연습에서 사용할 **s2i**라는 새 브랜치를 만들며 내보냅니다.

- 1.1. **D0180-apps** Git 리포지토리의 로컬 복제본을 입력하고 교육 과정 리포지토리의 **master** 분기를 체크아웃하여 알려진 양호한 상태에서 이 연습을 시작합니다.

```
[student@workstation openshift-s2i]$ cd ~/D0180-apps
[student@workstation D0180-apps]$ git checkout master
...output omitted...
```

- 1.2. 이 연습 중 변경한 내용을 저장하려면 새 분기를 생성합니다.

```
[student@workstation D0180-apps]$ git checkout -b s2i
Switched to a new branch 's2i'
[student@workstation D0180-apps]$ git push -u origin s2i
...output omitted...
 * [new branch]      s2i -> s2i
Branch s2i set up to track remote branch s2i from origin.
```

- 1.3. **php-helloworld** 폴더 내부에서 애플리케이션의 PHP 소스 코드를 검토합니다.

/home/student/D0180-apps/php-helloworld 폴더에 있는 **index.php** 파일을 엽니다.

```
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
?>
```

이 애플리케이션은 실행되는 PHP 버전을 반환하는 간단한 응답을 구현합니다.

▶ 2. 랙 환경을 준비합니다.

- 2.1. 강의실 환경 구성은 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation D0180-apps]$ source /usr/local/etc/ocp4.config
```

- 2.2. OpenShift 클러스터에 로그인합니다.

```
[student@workstation D0180-apps]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.3. 이 연습에서 생성하는 리소스에 대한 RHOCP 개발자 사용자 이름을 포함하는 새 프로젝트를 생성합니다.

```
[student@workstation D0180-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-s2i
```

▶ 3. D0180-apps Git 리포지토리의 분기에서 이전 단계에서 만든 **s2i** 분기를 사용하는 **php-helloworld** 디렉터리의 S2I(Source-to-Image)를 활용하여 새 PHP 애플리케이션을 만듭니다.

- 3.1. **oc new-app** 명령을 사용하여 PHP 애플리케이션을 생성합니다.



중요

다음 예제에서는 숫자 기호(#)를 사용하여 Git 리포지토리에서 특정 브랜치를 선택합니다(이 경우 이전 단계에서 생성된 **s2i** 브랜치).

```
[student@workstation D0180-apps]$ oc new-app --as-deployment-config php:7.3 \
> --name=php-helloworld \
> https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps#s2i \
> --context-dir php-helloworld
```

- 3.2. 빌드가 완료되고 애플리케이션이 배포될 때까지 기다립니다. **oc get pods** 명령으로 빌드 프로세스가 시작되는지 확인합니다.

```
[student@workstation openshift-s2i]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
php-helloworld-1-build 1/1     Running   0          5s
```

- 3.3. 이 빌드에 대한 로그를 검사합니다. 이 빌드에 빌드 포드 이름, **php-helloworld-1-build**를 사용합니다.

```
[student@workstation D0180-apps]$ oc logs --all-containers \
> -f php-helloworld-1-build
Cloning "https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps" ...
```

```

Commit: f7cd8963ef353d9173c3a21dcccf402f3616840b (Initial commit, including all
apps previously in course)

...output omitted...

Writing manifest to image destination
Storing signatures
Generating dockerfile with builder image image-registry.openshift-image-...
php@sha256:f3c9...7546
STEP 1: FROM image-registry.openshift-image-registry.svc:5000/...

...output omitted...

Pushing image ...openshift-image-registry.svc:5000/s2i/php-helloworld:latest...
Getting image source signatures
...output omitted...
STEP 8: RUN /usr/libexec/s2i/assemble
--> Installing application source...
...output omitted...
Copying config sha256:6ce5730f48d9c746e7cbd7ea7b8ed0f15b83932444d1d2bd7711d7...
21.45 KiB / 21.45 KiB 0s
Writing manifest to image destination
Storing signatures
Successfully pushed .../php-helloworld:latest@sha256:63e757a4c0edaeda497dab7...
Push successful

```

Git 리포지토리의 복제본을 빌드의 첫 번째 단계로 참조하십시오. 다음으로 S2I(Source-to-Image) 프로세스가 **s2i/php-helloworld:latest**라는 새 컨테이너를 빌드합니다. 빌드 프로세스의 마지막 단계는 이 컨테이너를 OpenShift 프라이빗 레지스트리로 내보내는 것입니다.

3.4. 이 애플리케이션의 DeploymentConfig를 검토합니다.

```

[student@workstation D0180-apps]$ oc describe dc/php-helloworld
Name:          php-helloworld
Namespace:     ${RHT_OCP4_DEV_USER}-s2i
Created:       About a minute ago
Labels:        app=php-helloworld
               app.kubernetes.io/component=php-helloworld
               app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/generated-by=OpenShiftNewApp
Latest Version: 1
Selector:      deploymentconfig=php-helloworld
Replicas:      1
Triggers:      Config, Image(phi-helloworld@latest, auto=true)
Strategy:      Rolling
Template:
Pod Template:
  Labels:      deploymentconfig=php-helloworld
  Annotations: openshift.io/generated-by: OpenShiftNewApp
  Containers:
    php-helloworld:
      Image:      image-registry.openshift-image-
                   registry.svc:5000/${RHT_OCP4_DEV_USER}-s2i/php-
                   helloworld@sha256:ae3ec890625f153c0163812501f2522fddc96431036374aa2472ddd52bc7ccb4

```

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

```

Ports:          8080/TCP, 8443/TCP
Host Ports:    0/TCP, 0/TCP
Environment:   <none>
Mounts:        <none>
Volumes:       <none>

Deployment #1 (latest):
Name:          php-helloworld-1
Created:       about a minute ago
Status:        Complete
Replicas:      1 current / 1 desired
Selector:      deployment=php-helloworld-1,deploymentconfig=php-helloworld
Labels:        app.kubernetes.io/component=php-helloworld,app.kubernetes.io/
instance=php-helloworld,app=php-helloworld,openshift.io/deployment-
config.name=php-helloworld
Pods Status:   1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...output omitted...

```

3.5. 애플리케이션을 테스트할 경로를 추가합니다.

```
[student@workstation D0180-apps]$ oc expose service php-helloworld \
> --name ${RHT_OCP4_DEV_USER}-helloworld
route.route.openshift.io/${RHT_OCP4_DEV_USER}-helloworld exposed
```

3.6. 새 경로와 연결된 URL을 찾습니다.

```
[student@workstation D0180-apps]$ oc get route -o jsonpath='{..spec.host}{"\n"}' \
${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.
${RHT_OCP4_WILDCARD_DOMAIN}
```

3.7. 이전 단계에서 획득한 URL에 HTTP GET 요청을 보내 애플리케이션을 테스트합니다.

```
[student@workstation D0180-apps]$ curl -s \
> ${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.\
> ${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World! php version is 7.3.11
```

- ▶ 4. Git 리포지토리에서 애플리케이션을 변경하고 새 S2I(Source-to-Image) 빌드를 시작하도록 적절한 명령을 실행하여 애플리케이션 빌드 시작을 살펴봅니다.

4.1. 소스 코드 디렉터리를 입력합니다.

```
[student@workstation D0180-apps]$ cd ~/D0180-apps/php-helloworld
```

4.2. 다음과 같이 `index.php` 파일을 편집합니다.

```
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
print "A change is a coming!\n";
?>
```

파일을 저장합니다.

- 4.3. 변경 사항을 커밋하고 코드를 다시 원격 Git 리포지토리에 내보냅니다.

```
[student@workstation php-helloworld]$ git add .
[student@workstation php-helloworld]$ git commit -m 'Changed index page contents.'
[s2i b1324aa] changed index page contents
 1 file changed, 1 insertion(+)
[student@workstation php-helloworld]$ git push origin s2i
...output omitted...
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 417 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps
  f7cd896..b1324aa s2i -> s2i
```

- 4.4. 새 S2I(Source-to-Image) 빌드 프로세스를 시작하고 빌드 및 배포가 완료될 때까지 기다립니다.

```
[student@workstation php-helloworld]$ oc start-build php-helloworld
build.build.openshift.io/php-helloworld-2 started
[student@workstation php-helloworld]$ oc logs php-helloworld-2-build -f
...output omitted...

Successfully pushed .../php-helloworld:latest@sha256:74e757a4c0edaeda497dab7...
Push successful
```



참고

빌드를 시작하면 로그를 사용할 수 있을 때까지 몇 초가 걸립니다. 이전 명령이 실패하면 기다렸다가 다시 시도합니다.

- 4.5. 두 번째 빌드가 완료된 후 `oc get pods` 명령을 사용하여 새 버전의 애플리케이션이 실행 중인지 확인합니다.

NAME	READY	STATUS	RESTARTS	AGE
php-helloworld-1-build	0/1	Completed	0	11m
php-helloworld-1-deploy	0/1	Completed	0	10m
php-helloworld-2-build	0/1	Completed	0	45s
php-helloworld-2-deploy	0/1	Completed	0	16s
php-helloworld-2-wq9wz	1/1	Running	0	13s

- 4.6. 애플리케이션에서 새 콘텐츠를 제공하는지 테스트합니다.

```
[student@workstation php-helloworld]$ curl -s \
> ${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.\ \
> ${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World! php version is 7.3.11
A change is a coming!
```

완료

workstation에서 **lab openshift-s2i finish** 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation php-helloworld]$ lab openshift-s2i finish
```

이로써 안내에 따른 연습이 완료됩니다.

▶ 랩

OpenShift에 컨테이너화된 애플리케이션 배포

수행 체크리스트

이 랩에서는 OpenShift S2I(Source-to-Image) 기능을 사용하여 애플리케이션을 생성합니다.

결과

OpenShift 애플리케이션을 생성하고 웹 브라우저를 통해 액세스할 수 있습니다.

시작하기 전에

workstation에서 **student** 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab openshift-review start
```

1. 랩 환경을 준비합니다.

- 1.1. 강의실 환경 구성을 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. OpenShift 클러스터에 로그인합니다.
- 1.3. 이 연습에서 생성하는 리소스에 대해 "\${RHT_OCP4_DEV_USER}-ocp"라는 새 프로젝트를 생성합니다.
2. **php:7.3** 이미지 스트림 태그를 사용하여 PHP에 작성된 온도 변환기 애플리케이션을 생성합니다.
소스 코드는 **temps** 디렉터리의 Git 리포지토리(<https://github.com/RedHatTraining/DO180-apps/>)에 있습니다. OpenShift 명령줄 인터페이스 또는 웹 콘솔을 사용하여 애플리케이션을 생성할 수 있습니다.
웹 브라우저에서 애플리케이션을 액세스할 수 있도록 애플리케이션 서비스를 노출합니다.
3. 웹 브라우저([http://temps-\\${RHT_OCP4_DEV_USER}-ocp.\\${RHT_OCP4_WILDCARD_DOMAIN}](http://temps-${RHT_OCP4_DEV_USER}-ocp.${RHT_OCP4_WILDCARD_DOMAIN}))에서 애플리케이션에 액세스할 수 있는지 확인하십시오.

평가

workstation에서 **lab openshift-review grade** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab openshift-review grade
```

완료

workstation에서 **lab openshift-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab openshift-review finish
```

이로써 랩이 완료됩니다.

▶ 솔루션

OpenShift에 컨테이너화된 애플리케이션 배포

수행 체크리스트

이 랩에서는 OpenShift S2I(Source-to-Image) 기능을 사용하여 애플리케이션을 생성합니다.

결과

OpenShift 애플리케이션을 생성하고 웹 브라우저를 통해 액세스할 수 있습니다.

시작하기 전에

workstation에서 student 사용자로 터미널을 열고 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab openshift-review start
```

1. 랩 환경을 준비합니다.

- 1.1. 강의실 환경 구성을 로드합니다.

다음 명령을 실행하여 첫 번째 안내에 따른 연습에서 만든 환경 변수를 로드합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. OpenShift 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. 이 연습에서 생성하는 리소스에 대해 "\${RHT_OCP4_DEV_USER}-ocp"라는 새 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-ocp
```

2. php:7.3 이미지 스트림 태그를 사용하여 PHP에 작성된 온도 변환기 애플리케이션을 생성합니다. 소스 코드는 temps 디렉터리의 Git 리포지토리(<https://github.com/RedHatTraining/D0180-apps/>)에 있습니다. OpenShift 명령줄 인터페이스 또는 웹 콘솔을 사용하여 애플리케이션을 생성할 수 있습니다.

웹 브라우저에서 애플리케이션을 액세스할 수 있도록 애플리케이션 서비스를 노출합니다.

- 2.1. 명령줄 인터페이스를 사용하는 경우 다음 명령을 실행합니다.

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> php:7.3~https://github.com/RedHatTraining/D0180-apps \
> --context-dir temps --name temps
```

6장 | OpenShift에 컨테이너화된 애플리케이션 배포

```
--> Found image fbe3911 (2 weeks old) in image stream "openshift/php" under tag
"7.3" for "php:7.3"

Apache 2.4 with PHP 7.3
-----
PHP 7.3 available as container is a base platform ...output omitted...

...output omitted...

--> Creating resources ...
imagestream.image.openshift.io "temps" created
buildconfig.build.openshift.io "temps" created
deploymentconfig.apps.openshift.io "temps" created
service "temps" created
--> Success
Build scheduled, use 'oc logs -f bc/temps' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
  'oc expose svc/temps'
Run 'oc status' to view your app.
```

2.2. 빌드 진행 상태를 모니터링합니다.

```
[student@workstation ~]$ oc logs -f bc/temps
Cloning "https://github.com/RedHatTraining/D0180-apps" ...
Commit: f7cd8963ef353d9173c3a21dccc402f3616840b (Initial commit, including all
apps previously in course)
...output omitted...
Successfully pushed image-registry.openshift-
image-registry.svc:5000/${RHT_OCP4_DEV_USER}-temps/
temps@sha256:59f713adfacdbc2a3ca81c4ef4af46517dfffa3f0029372f86fbcaf571416a74
Push successful
```

2.3. 애플리케이션이 배포되었는지 확인합니다.

```
[student@workstation ~]$ oc get pods -w
NAME        READY   STATUS    RESTARTS   AGE
temps-1-build  0/1     Completed   0          91s
temps-1-deploy  0/1     Completed   0          60s
temps-1-p4zjc   1/1     Running    0          58s
```

Ctrl+C를 눌러 oc get pods -w 명령을 종료합니다.

2.4. temps 서비스를 노출하여 애플리케이션의 외부 경로를 생성하십시오.

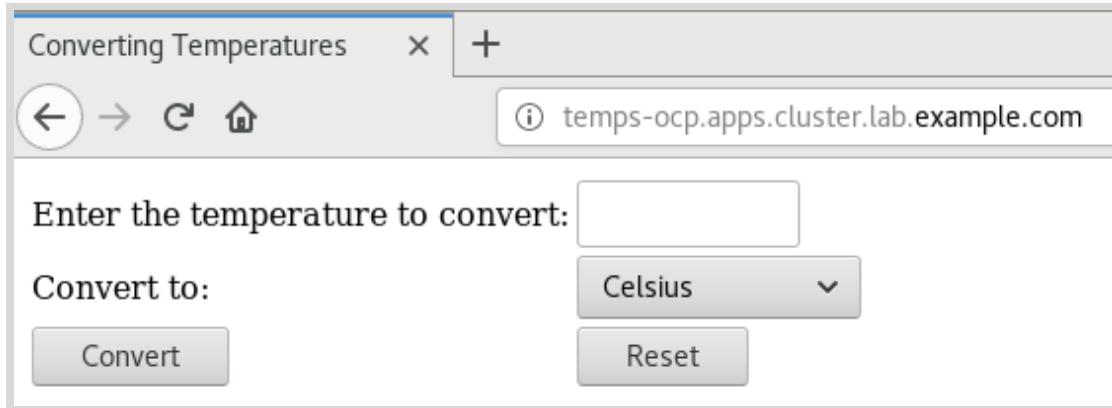
```
[student@workstation ~]$ oc expose svc/temps
route.route.openshift.io/temps exposed
```

3. 웹 브라우저([http://temps-\\${RHT_OCP4_DEV_USER}-ocp.\\${RHT_OCP4_WILDCARD_DOMAIN}](http://temps-${RHT_OCP4_DEV_USER}-ocp.${RHT_OCP4_WILDCARD_DOMAIN}))에서 애플리케이션에 액세스할 수 있는지 확인하십시오.

3.1. 경로에 대한 URL을 확인합니다.

```
[student@workstation ~]$ oc get route/temps
NAME      HOST/PORT
temps    temps-${RHT_OCP4_DEV_USER}-ocp.${RHT_OCP4_WILDCARD_DOMAIN}  ...
```

- 3.2. 웹 브라우저를 열고 이전 단계에 표시된 URL로 이동하여 온도 변환기 애플리케이션의 작동 여부를 확인합니다.



평가

workstation에서 `lab openshift-review grade` 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab openshift-review grade
```

완료

workstation에서 `lab openshift-review finish` 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab openshift-review finish
```

이로써 랩이 완료됩니다.

요약

이 장에서 학습한 내용:

- OpenShift 명령줄 클라이언트 **oc**를 사용하여 다음을 수행합니다.
 - 프로젝트를 만들고, 변경하고, 삭제합니다.
 - 프로젝트 내 애플리케이션 리소스를 생성합니다.
 - 프로젝트 내의 리소스를 삭제하고, 검사하고, 편집하여 내보냅니다.
 - 애플리케이션 포드, 배포 및 빌드 작업에서 로그를 확인합니다.
- **oc new-app** 명령은 이미지 레지스트리에서 호스팅되는 기존 컨테이너 이미지, Dockerfile 및 S2I(Source-to-Image) 프로세스를 사용하는 소스 코드를 이용하는 등 여러 다양한 방법으로 애플리케이션 포드를 생성할 수 있습니다.
- S2I(Source-to-Image)는 애플리케이션 소스 코드로 컨테이너 이미지를 쉽게 빌드할 수 있는 툴입니다. 이 툴은 Git 리포지토리에서 소스 코드를 검색하고, 원하는 언어 또는 기술을 기반으로 선택한 컨테이너 이미지에 소스 코드를 삽입하고, 조합된 애플리케이션을 실행하는 새 컨테이너 이미지를 생성합니다.
- **경로**는 공용 방향 IP 주소와 DNS 호스트 이름을 내부 방향 서비스 IP에 연결합니다. 서비스를 통해 OpenShift 인스턴스 내부에서 포드 간 네트워크 액세스가 가능한 반면, 경로를 통해서는 OpenShift 인스턴스 외부의 사용자와 애플리케이션에서 포드에 대한 네트워크 액세스가 가능합니다.

멀티컨테이너 애플리케이션 배포

목적

멀티 컨테이너 이미지를 사용하여 컨테이너화된 애플리케이션을 배포합니다.

목표

- 템플릿을 사용하여 OpenShift에 멀티컨테이너 애플리케이션을 배포합니다.

섹션

- OpenShift에 멀티컨테이너 애플리케이션 배포(안내에 따른 연습)

랩

소프트웨어 애플리케이션 컨테이너화 및 배포

OpenShift에 멀티컨테이너 애플리케이션 배포

목표

이 섹션을 마치면 템플릿을 사용하여 OpenShift에 멀티컨테이너 애플리케이션을 배포할 수 있습니다.

템플릿의 **Skeleton** 검사

OpenShift Container Platform에서 애플리케이션을 배포하려면 대부분 프로젝트 내에 여러 개의 관련 리소스를 만들어야 합니다. 예를 들어, 웹 애플리케이션에는 OpenShift 프로젝트에서 실행할 **BuildConfig**, **DeploymentConfig**, **Service**, **Route** 리소스가 있어야 합니다. 일반적으로 이러한 리소스의 속성에는 리소스의 이름 속성과 같이 동일한 값이 있습니다.

OpenShift 템플릿을 사용하면 애플리케이션에 필요한 리소스를 간단하게 생성할 수 있습니다. 템플릿은 함께 생성할 관련 리소스 집합과 애플리케이션 매개 변수 집합을 정의합니다. 템플릿 리소스의 속성은 일반적으로 리소스의 이름 속성과 같이 템플릿 매개 변수의 관점에서 정의됩니다.

예를 들어, 애플리케이션은 프런트엔드 웹 애플리케이션과 데이터베이스 서버로 구성될 수 있습니다. 각각 서비스 리소스 및 배포 구성 리소스로 구성됩니다. 해당 리소스는 프런트엔드에 대한 일련의 자격 증명(매개 변수)을 공유하여 백엔드를 인증합니다. 템플릿의 리소스 목록을 결합 애플리케이션으로 인스턴스화하기 위해 매개 변수를 지정하거나 자동으로 생성될 수 있게 허용하여(예: 고유 데이터베이스 암호) 템플릿을 처리할 수 있습니다.

OpenShift 설치 프로그램은 **openshift** 네임스페이스에서 기본적으로 여러 개의 템플릿을 생성합니다. **oc get templates** 명령을 **-n openshift** 옵션과 함께 실행하여 사전 설치된 템플릿을 나열합니다.

```
[student@workstation ~]$ oc get templates -n openshift
NAME                      DESCRIPTION
cakephp-mysql-example     An example CakePHP application ...
cakephp-mysql-persistent   An example CakePHP application ...
dancer-mysql-example      An example Dancer application with a MySQL ...
dancer-mysql-persistent   An example Dancer application with a MySQL ...
django-psql-example       An example Django application with a PostgreSQL ...
...output omitted...
rails-pgsql-persistent    An example Rails application with a PostgreSQL ...
rails-postgresql-example An example Rails application with a PostgreSQL ...
redis-ephemeral           Redis in-memory data structure store, ...
redis-persistent           Redis in-memory data structure store, ...
```

다음은 YAML 템플릿 정의를 보여줍니다.

```
[student@workstation ~]$ oc get template mysql-persistent -n openshift -o yaml
apiVersion: template.openshift.io/v1
kind: Template
labels: ...value omitted...
message: ...message omitted ...
metadata:
  annotations:
    description: ...description omitted...
```

```

iconClass: icon-mysql-database
openshift.io/display-name: MySQL
openshift.io/documentation-url: ...value omitted...
openshift.io/long-description: ...value omitted...
openshift.io/provider-display-name: Red Hat, Inc.
openshift.io/support-url: https://access.redhat.com
tags: database,mysql ①
labels: ...value omitted...
name: mysql-persistent ②
objects: ③
- apiVersion: v1
  kind: Secret
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME} ④
  stringData: ...stringData omitted...
- apiVersion: v1
  kind: Service
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME}
  spec: ...spec omitted...
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: ${DATABASE_SERVICE_NAME}
  spec: ...spec omitted...
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME}
  spec: ...spec omitted...
parameters: ⑤
- ...MEMORY_LIMIT parameter omitted...
- ...NAMESPACE parameter omitted...
- description: The name of the OpenShift Service exposed for the database.
  displayName: Database Service Name
  name: DATABASE_SERVICE_NAME ⑥
  required: true
  value: mysql
- ...MYSQL_USER parameter omitted...
- description: Password for the MySQL connection user.
  displayName: MySQL Connection Password
  from: '[a-zA-Z0-9]{16}' ⑦
  generate: expression
  name: MYSQL_PASSWORD
  required: true
- ...MYSQL_ROOT_PASSWORD parameter omitted...
- ...MYSQL_DATABASE parameter omitted...
- ...VOLUME_CAPACITY parameter omitted...
- ...MYSQL_VERSION parameter omitted...

```

- ① 이 템플릿과 연결할 임의 태그 목록을 정의합니다. 이 템플릿을 찾으려면 UI에 이러한 태그 중 하나를 입력합니다.

7장 | 멀티컨테이너 애플리케이션 배포

- ❷ 템플릿 이름을 정의합니다.
- ❸ **objects** 섹션에서는 이 템플릿에 대한 OpenShift 리소스 목록을 정의합니다. 이 템플릿에서는 네 가지 리소스, 즉 **Secret**, **Service**, **PersistentVolumeClaim**, **DeploymentConfig**가 생성됩니다.
- ❹ 네 가지 리소스 오브젝트 모두 이름이 **DATABASE_SERVICE_NAME** 매개 변수의 값으로 설정됩니다.
- ❺ **parameters** 섹션에는 9개의 매개 변수 목록이 있습니다. 템플릿 리소스는 **DATABASE_SERVICE_NAME** 매개 변수를 사용하여 시연한 대로 대부분 이러한 매개 변수의 값을 사용하여 속성을 정의합니다.
- ❻ 이 템플릿으로 애플리케이션을 만들 때 **MYSQL_PASSWORD** 매개 변수에 값을 지정하지 않으면 OpenShift에서 이 정규 표현식과 일치하는 암호를 생성합니다.

다른 개발자가 템플릿에서 애플리케이션을 빌드할 수 있도록 OpenShift 클러스터에 새 템플릿을 게시할 수 있습니다.

배포를 위해 OpenShift **DeploymentConfig**, **Service** 및 **Route** 오브젝트가 필요한 작업 목록 애플리케이션 **todo**가 있다고 가정합니다. 이러한 OpenShift 리소스에 대한 속성을 정의하는 YAML 템플릿 정의 파일을 필요한 모든 매개 변수 정의와 함께 생성합니다. 템플릿이 **todo-template.yaml** 파일에 정의되어 있다고 가정하고 **oc create** 명령을 사용하여 애플리케이션 템플릿을 게시합니다.

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml
template.template.openshift.io/todonodejs-persistent created
```

다음 예제 표시된 대로 **-n** 옵션을 사용하여 다른 프로젝트를 지정하지 않는 한 기본적으로 템플릿은 현재 프로젝트 아래에 생성됩니다.

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml \
> -n openshift
```



중요

openshift 네임스페이스(OpenShift 프로젝트)에서 생성된 템플릿은 **Catalog(카탈로그)** → **Developer Catalog(개발자 카탈로그)** 메뉴 항목에서 액세스할 수 있는 대화 상자 아래의 웹 콘솔에서 사용할 수 있습니다. 또한 현재 프로젝트에서 생성된 모든 템플릿은 해당 프로젝트에서 액세스할 수 있습니다.

매개 변수

템플릿은 값이 할당된 매개 변수 집합을 정의합니다. 해당 템플릿에 정의된 OpenShift 리소스는 이름이 지정된 매개 변수를 참조하여 구성 값을 가져올 수 있습니다. 템플릿의 매개 변수에 기본값이 있을 수 있지만 해당 매개 변수는 옵션입니다. 템플릿을 처리할 때 기본값을 교체할 수 있습니다.

각 매개 변수 값은 매개 변수 구성을 따라 **oc process** 명령을 사용하여 명시적으로 설정하거나 OpenShift를 통해 생성할 수 있습니다.

템플릿에서 사용 가능한 매개 변수를 나열하는 방법은 두 가지가 있습니다. 첫 번째는 **oc describe** 명령을 사용하는 방법입니다.

```
$ oc describe template mysql-persistent -n openshift
Name:      mysql-persistent
Namespace:  openshift
Created:   12 days ago
Labels:    samplesoperator.config.openshift.io/managed=true
```

```
Description: MySQL database service, with ...description omitted...
Annotations: iconClass=icon-mysql-database
            openshift.io/display-name=MySQL
            ...output omitted...
            tags=database,mysql

Parameters:
  Name: MEMORY_LIMIT
  Display Name: Memory Limit
  Description: Maximum amount of memory the container can use.
  Required: true
  Value: 512Mi

  Name: NAMESPACE
  Display Name: Namespace
  Description: The OpenShift Namespace where the ImageStream resides.
  Required: false
  Value: openshift

  ...output omitted...

  Name: MYSQL_VERSION
  Display Name: Version of MySQL Image
  Description: Version of MySQL image to be used (5.7, or latest).
  Required: true
  Value: 5.7

Object Labels: template=mysql-persistent-template

Message: ...output omitted... in your project: ${DATABASE_SERVICE_NAME}.

  Username: ${MYSQL_USER}
  Password: ${MYSQL_PASSWORD}
  Database Name: ${MYSQL_DATABASE}
  Connection URL: mysql://${DATABASE_SERVICE_NAME}:3306/

For more information about using this template, ...output omitted...

Objects:
  Secret      ${DATABASE_SERVICE_NAME}
  Service     ${DATABASE_SERVICE_NAME}
  PersistentVolumeClaim ${DATABASE_SERVICE_NAME}
  DeploymentConfig ${DATABASE_SERVICE_NAME}
```

두 번째 방법은 `oc process`를 `--parameters` 옵션과 함께 사용하는 것입니다.

```
$ oc process --parameters mysql-persistent -n openshift
NAME          DESCRIPTION      GENERATOR      VALUE
MEMORY_LIMIT  Maximum a...    expression     512Mi
NAMESPACE     The OpenS...    expression     openshift
DATABASE_SERVICE_NAME The name ...
MYSQL_USER    Username ...
MYSQL_PASSWORD Password ... expression [a-zA-Z0-9]{3}
                  expression [a-zA-Z0-9]{16}
```

MYSQL_ROOT_PASSWORD	Password ...	expression	[a-zA-Z0-9]{16}
MYSQL_DATABASE	Name of t...		sampledb
VOLUME_CAPACITY	Volume sp...		1Gi
MYSQL_VERSION	Version o...		5.7

CLI를 사용하여 템플릿 처리

템플릿을 처리할 때 리소스 목록을 만들어 새 애플리케이션을 생성합니다. 템플릿을 처리하려면 **oc process** 명령을 사용합니다.

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

이전 명령은 JSON 또는 YAML 형식으로 된 템플릿 파일을 처리하고 리소스 목록을 표준 출력으로 반환합니다. 출력 리소스 목록의 형식은 JSON입니다. 리소스 목록을 YAML 형식으로 출력하려면 **-o yaml**과 함께 **oc process** 명령을 사용합니다.

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

또 다른 옵션은 현재 프로젝트나 **openshift** 프로젝트에서 템플릿을 처리하는 것입니다.

```
$ oc process <uploaded-template-name>
```



참고

oc process 명령은 리소스 목록을 표준 출력에 반환합니다. 이 출력은 파일로 리디렉션될 수 있습니다.

```
$ oc process -o yaml -f filename > myapp.yaml
```

템플릿은 종종 템플릿 매개 변수를 기반으로 하는 구성 가능 속성을 사용하여 리소스를 생성합니다. 매개 변수를 재정의하려면 **-p** 옵션 다음에 **<name>=<value>** 쌍 목록을 사용합니다.

```
$ oc process -o yaml -f mysql.yaml \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi > mysqlProcessed.yaml
```

애플리케이션을 생성하려면 생성된 YAML 리소스 정의 파일을 사용합니다.

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

또는 UNIX 파이프를 사용하여 리소스 정의 파일을 저장하지 않고 애플리케이션을 생성하고 템플릿을 처리할 수 있습니다.

```
$ oc process -f mysql.yaml -p MYSQL_USER=dev \
> -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

7장 | 멀티컨테이너 애플리케이션 배포

`openshift` 프로젝트에서 템플릿을 사용하여 프로젝트에 애플리케이션을 만들려면 먼저 해당 템플릿을 내보내야 합니다.

```
$ oc get template mysql-persistent -o yaml \
> -n openshift > mysql-persistent-template.yaml
```

그런 다음 템플릿 매개 변수에 적절한 값을 확인하고 템플릿을 처리합니다.

```
$ oc process -f mysql-persistent-template.yaml \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

두 개의 슬래시(`>/>`)를 사용하여 네임스페이스를 템플릿 이름의 일부로 제공할 수도 있습니다.

```
$ oc process openshift//mysql-persistent \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

또는 `oc new-app` 명령을 통해 템플릿 이름을 `--template` 옵션 인수로 전달하여 애플리케이션을 생성 할 수 있습니다.

```
$ oc new-app --template=mysql-persistent \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi \
> --as-deployment-config
```

OpenShift 애플리케이션용 영구저장장치 구성

OpenShift 컨테이너 플랫폼에서는 영구저장장치를 풀링된 클러스터 수준의 리소스로 관리합니다. OpenShift 관리자는 클러스터에 스토리지 리소스를 추가하기 위해 스토리지 리소스의 필수 메타데이터를 정의하는 **PersistentVolume** 오브젝트를 만듭니다. 메타데이터는 클러스터가 스토리지에 액세스하는 방법 및 용량 또는 처리량과 같은 기타 스토리지 속성을 설명합니다.

클러스터에 있는 **PersistentVolume** 오브젝트를 나열하려면 `oc get pv` 명령을 사용합니다.

```
[admin@host ~]$ oc get pv
NAME      CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS     CLAIM      ...
pv0001    1Mi        RWO          Retain        Available   ...
pv0002    10Mi       RWX          Recycle       Available   ...
...output omitted...
```

지정된 **PersistentVolume**에 대한 YAML 정의를 보려면 `oc get` 명령과 함께 `-o yaml` 옵션을 사용합니다.

```
[admin@host ~]$ oc get pv pv0001 -o yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  creationTimestamp: ...value omitted...
  finalizers:
  - kubernetes.io/pv-protection
```

```

labels:
  type: local
  name: pv0001
resourceVersion: ...value omitted...
selfLink: /api/v1/persistentvolumes/pv0001
uid: ...value omitted...
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Mi
  hostPath:
    path: /data/pv0001
    type: ""
  persistentVolumeReclaimPolicy: Retain
status:
  phase: Available

```

클러스터에 **PersistentVolume** 오브젝트를 추가하려면 **oc create** 명령을 사용합니다.

```
[admin@host ~]$ oc create -f pv1001.yaml
```



참고

위 **pv1001.yaml** 파일에는 **oc get pv pv-name -o yaml** 명령의 출력과 구조가 비슷한 영구적인 볼륨 정의가 있어야 합니다.

영구적인 볼륨 요청

애플리케이션에 스토리지가 필요한 경우 **PersistentVolumeClaim**(PVC) 오브젝트를 생성하여 클러스터 풀에서 전용 스토리지 리소스를 요청합니다. **pvc.yaml**이라는 파일에 있는 다음 내용은 PVC에 대한 예제 정의입니다.

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

```

PVC는 용량 또는 처리량과 같은 애플리케이션의 스토리지 요구 사항을 정의합니다. PVC를 만들려면 **oc create** 명령을 사용합니다.

```
[admin@host ~]$ oc create -f pvc.yaml
```

PVC를 만들고 나면 OpenShift에서 PVC의 요구 사항을 충족하는 사용 가능한 **PersistentVolume** 리소스를 찾습니다. OpenShift가 일치하는 항목을 찾으면 PersistentVolume 오브젝트를

PersistentVolumeClaim 오브젝트에 바인딩합니다. 프로젝트의 PVC를 나열하려면 `oc get pvc` 명령을 사용합니다.

```
[admin@host ~]$ oc get pvc
NAME      STATUS    VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
myapp    Bound     pv0001   1Gi       RWO          default        6s
```

출력에는 PVC의 속성과 함께 영구적인 볼륨이 PVC에 바인딩되어 있는지 표시됩니다(예: 용량).

애플리케이션 포드에서 영구적인 볼륨을 사용하려면 **PersistentVolumeClaim** 오브젝트를 참조하는 컨테이너에 대한 볼륨 마운트를 정의하십시오. 아래의 애플리케이션 포드 정의는 **PersistentVolumeClaim** 오브젝트를 참조하여 애플리케이션에 대한 볼륨 마운트를 정의합니다.

```
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "myapp"
  labels:
    name: "myapp"
spec:
  containers:
    - name: "myapp"
      image: openshift/myapp
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/var/www/html"
          name: "pvol" ❶
  volumes:
    - name: "pvol" ❷
      persistentVolumeClaim:
        claimName: "myapp" ❸
```

- ❶ 이 섹션에서는 컨테이너 파일 시스템의 `/var/www/html`에 `pvol` 볼륨을 마운트함을 선언합니다.
- ❷ 이 섹션에서는 `pvol` 볼륨을 정의합니다.
- ❸ `pvol` 볼륨은 `myapp` PVC를 참조합니다. OpenShift가 사용 가능한 영구적인 볼륨을 `myapp` PVC에 연결하면 `pvol` 볼륨에서 연결된 이 볼륨을 참조합니다.

템플릿을 사용하여 영구저장장치 구성

일반적으로 템플릿은 영구저장장치가 필요한 애플리케이션을 간단히 생성하는데 사용됩니다. 이러한 템플릿에는 대부분 접미사 `-persistent`가 있습니다.

```
[student@workstation ~]$ oc get templates -n openshift | grep persistent
cakephp-mysql-persistent  An example CakePHP application with a MySQL data...
dancer-mysql-persistent   An example Dancer application with a MySQL datab...
django-psql-persistent    An example Django application with a PostgreSQL ...
dotnet-psql-persistent    An example .NET Core application with a PostgreS...
jenkins-persistent         Jenkins service, with persistent storage....
mariadb-persistent        MariaDB database service, with persistent storag...
mongodb-persistent        MongoDB database service, with persistent storag...
mysql-persistent           MySQL database service, with persistent storage....
```

nodejs-mongo-persistent	An example Node.js application with a MongoDB da...
postgresql-persistent	PostgreSQL database service, with persistent sto...
rails-pgsql-persistent	An example Rails application with a PostgreSQL d...
redis-persistent	Redis in-memory data structure store, with persi...

다음 예제 템플릿에서는 **PersistentVolumeClaim** 오브젝트와 함께 **DeploymentConfig** 오브젝트를 정의합니다.

```

apiVersion: template.openshift.io/v1
kind: Template
labels:
  template: myapp-persistent-template
metadata:
  name: myapp-persistent
  namespace: openshift
objects:
- apiVersion: v1
  kind: PersistentVolumeClaim ①
  metadata:
    name: ${APP_NAME}
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: ${VOLUME_CAPACITY}
- apiVersion: v1
  kind: DeploymentConfig ②
  metadata:
    name: ${APP_NAME}
  spec:
    replicas: 1
    selector:
      name: ${APP_NAME}
    strategy:
      type: Recreate
    template:
      metadata:
        labels:
          name: ${APP_NAME}
      spec:
        containers:
          - image: 'openshift/myapp'
            name: myapp
            volumeMounts:
              - mountPath: /var/lib/myapp/data
                name: ${APP_NAME}-data ③
        volumes:
          - name: ${APP_NAME}-data ④
            persistentVolumeClaim:
              claimName: ${APP_NAME}
parameters:
- description: The name for the myapp application.
  displayName: Application Name

```

```

name: APP_NAME ⑤
required: true
value: myapp
- description: Volume space available for data, e.g. 512Mi, 2Gi.
  displayName: Volume Capacity
  name: VOLUME_CAPACITY ⑥
  required: true
  value: 1Gi

```

- ① ② 템플릿에서 **PersistentVolumeClaim** 및 **DeploymentConfig** 오브젝트를 정의합니다. 두 오브젝트 모두 **APP_NAME** 매개 변수의 값과 일치하는 이름이 있습니다. 영구적인 볼륨 클레임은 **VOLUME_CAPACITY** 매개 변수 값에 해당하는 용량을 정의합니다.
- ③ ④ **DeploymentConfig** 오브젝트는 템플릿으로 생성한 **PersistentVolumeClaim**을 참조하는 볼륨 마운트를 정의합니다.
- ⑤ ⑥ 템플릿에서 두 매개 변수, **APP_NAME** 및 **VOLUME_CAPACITY**를 정의합니다. 템플릿은 이러한 매개 변수를 사용하여 **PersistentVolumeClaim** 및 **DeploymentConfig** 오브젝트에 대한 속성값을 지정합니다.

이 템플릿을 사용하면 **APP_NAME** 및 **VOLUME_CAPACITY** 매개 변수를 지정하여 영구저장장치가 있는 **myapp** 애플리케이션을 배포해야 합니다.

```

[student@workstation ~]$ oc create myapp-template.yaml
template.template.openshift.io/myapp-persistent created
[student@workstation ~]$ oc process myapp-persistent \
> -p APP_NAME=myapp-dev -p VOLUME_CAPACITY=1Gi \
> | oc create -f -
deploymentconfig/myapp created
persistentvolumeclaim/myapp created

```



참조

템플릿에 대한 개발자 정보는 OpenShift Container Platform 설명서의 Using Templates(템플릿 사용) 섹션에서 확인할 수 있습니다.

개발자 가이드

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/images/index#using-templates

▶ 연습 가이드

템플릿으로 애플리케이션 생성

이 연습에서는 애플리케이션에서 실행해야 하는 리소스를 정의하기 위해 템플릿을 사용하여 OpenShift Container Platform에 To Do List 애플리케이션을 배포합니다.

결과

제공된 JSON 템플릿을 사용하여 OpenShift Container Platform에 애플리케이션을 빌드하고 배포할 수 있습니다.

시작하기 전에

workstation에는 To Do List 애플리케이션 소스 코드 및 랩 파일이 있어야 합니다. 랩 파일을 다운로드하고 OpenShift 클러스터의 상태를 확인하려면 새 터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation ~]$ lab multicontainer-openshift start
```

- ▶ 1. **images/mysql** 하위 디렉터리에 **Dockerfile**을 사용하여 데이터베이스 컨테이너를 빌드합니다. 컨테이너 이미지를 **quay.io**에 **do180-mysql-57-rhel7** 태그와 함께 게시합니다.

- 1.1. MySQL 데이터베이스 이미지를 빌드합니다.

```
[student@workstation ~]$ cd ~/D0180/labs/multicontainer-openshift/images/mysql
[student@workstation mysql]$ sudo podman build -t do180-mysql-57-rhel7 .
STEP 1: FROM rhscl/mysql-57-rhel7
Getting image source signatures
Copying blob sha256:e373541...output omitted...
  69.66 MB / 69.66 MB [=====] 6s
Copying blob sha256:c5d2e94...output omitted...
  1.20 KB / 1.20 KB [=====] 0s
Copying blob sha256:b3949ae...output omitted...
  62.03 MB / 62.03 MB [=====] 5s
Writing manifest to image destination
Storing signatures
STEP 2: ADD root /
ERROR[0001] HOSTNAME is not supported for OCI image format ...output omitted...
--> b628...a079
STEP 3: COMMIT do180-mysql-57-rhel7
```



참고

ERROR 오류 메시지는 Podman 초기 버전의 알려진 문제입니다. 이 메시지는 무시해도 됩니다.

- 1.2. OpenShift에서 이미지를 사용하려면 이미지에 태그를 지정하고 이미지를 **quay.io**에 내보냅니다. 이를 수행하려면 터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation mysql]$ source /usr/local/etc/ocp4.config
[student@workstation mysql]$ sudo podman login quay.io -u ${RHT_OCP4_QUAY_USER}
Password: your_quay_password
Login Succeeded!
[student@workstation mysql]$ sudo podman tag \
> do180-mysql-57-rhel7 quay.io/${RHT_OCP4_QUAY_USER}/do180-mysql-57-rhel7
[student@workstation mysql]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-mysql-57-rhel7
Getting image source signatures
Copying blob ssha256:04dbea...b21619
205.17 MB / 205.17 MB [=====] 1m19s
...output omitted...
Writing manifest to image destination
Storing signatures
```

**참고**

위 출력에서 config sha256: 값은 사용자의 출력과 다를 수 있습니다.

**경고**

OpenShift에서 이미지를 가져올 수 있도록 리포지토리가 **quay.io**에서 공용인지 확인합니다. 부록 C의 **Repositories Visibility**(리포지토리 가시성) 섹션을 참조하여 리포지토리 가시성을 변경하는 방법에 대한 세부 정보를 읽어 보십시오.

- ▶ 2. 연습 하위 디렉터리 **images/nodejs**에 있는 Node.js Dockerfile을 사용하여 To Do List 애플리케이션의 기본 이미지를 빌드합니다. 이미지에 **do180-nodejs** 태그를 지정합니다. 이 이미지를 레지스트리에 게시하지 마십시오.

```
[student@workstation mysql]$ cd ~/D0180/labs/multicontainer-openshift
[student@workstation multicontainer-openshift]$ cd images/nodejs
[student@workstation nodejs]$ sudo podman build --layers=false -t do180-nodejs .
STEP 1: ubi7/ubi:7.7
Getting image source signatures
Copying blob sha256:5d92fc...1ce84e
...output omitted...
Storing signatures
STEP 2: MAINTAINER username <username@example.com>
...output omitted...
STEP 8: CMD ["echo", "You must create your own container from this one."]
...output omitted...
STEP 9: COMMIT ...output omitted...localhost/do180-nodejs:latest
...output omitted...
```

- ▶ 3. **deploy/nodejs** 하위 디렉터리에서 **build.sh** 스크립트를 사용하여 To Do List 애플리케이션을 빌드합니다. 이미지 태그 **do180-todonodejs**를 사용하여 **quay.io**에 애플리케이션 이미지를 게시합니다.

- 3.1. **~/D0180/labs/multicontainer-openshift/deploy/nodejs** 디렉터리로 이동하고 **build.sh** 명령을 실행하여 하위 이미지를 빌드합니다.

```
[student@workstation nodejs]$ cd ~/DO180/labs/multicontainer-openshift
[student@workstation multicontainer-openshift]$ cd deploy/nodejs
[student@workstation nodejs]$ ./build.sh
Preparing build folder
STEP 1: FROM do180-nodejs
STEP 2: ARG NEXUS_BASE_URL
STEP 3: MAINTAINER username <username@example.com>
...output omitted...
STEP 7: CMD ["scl","enable","rh-nodejs8","./run.sh"]
STEP 8: COMMIT containers-storage:...output omitted...
```

3.2. 이미지를 quay.io에 내보냅니다.

이미지를 OpenShift에서 템플릿에 사용할 수 있도록 하려면 태그를 지정하고 프라이빗 레지스터리에 내보냅니다. 이를 수행하려면 터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation nodejs]$ sudo podman tag do180/todonodejs \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-todonodejs
[student@workstation nodejs]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-todonodejs
Getting image source signatures
Copying blob sha256:24a5c62...output omitted...
...output omitted...
Copying blob sha256:5f70bf1...output omitted...
1024 B / 1024 B [=====] 0s
Copying config sha256:c43306d...output omitted...
7.26 KB / 7.26 KB [=====] 0s
Writing manifest to image destination
Copying config sha256:c43306d...output omitted...
0 B / 7.26 KB [-----] 0s
Writing manifest to image destination
Storing signatures
```



경고

OpenShift에서 이미지를 가져올 수 있도록 리포지토리가 **quay.io**에서 공용인지 확인합니다. 부록 C의 **Repositories Visibility**(리포지토리 가시성) 섹션을 참조하여 리포지토리 가시성을 변경하는 방법에 대한 세부 정보를 읽어 보십시오.

▶ 4. 제공된 JSON 템플릿에서 To Do List 애플리케이션을 만듭니다.

4.1. OpenShift Container Platform에 로그인합니다.

```
[student@workstation nodejs]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.

...output omitted...

Using project "default".
```

`oc login` 명령에서 안전하지 않은 연결 사용에 관한 메시지를 표시하는 경우 **y**(yes)라고 대답합니다.

- 4.2. OpenShift에서 이 실습에 사용할 새 프로젝트 template을 만듭니다. 다음 명령을 실행하여 **template** 프로젝트를 생성합니다.

```
[student@workstation nodejs]$ oc new-project ${RHT_OCP4_DEV_USER}-template  
Now using project ...output omitted...
```

- 4.3. 템플릿을 검토합니다.
선호하는 편집기를 사용하여 `/home/student/D0180/labs/multicontainer-openshift/todo-template.json`에 있는 템플릿을 열고 검토합니다. 템플릿에 정의된 다음 리소스를 확인하고 해당 구성을 검토합니다.
 - **todoapi** 포드 정의는 Node.js 애플리케이션을 정의합니다.
 - **mysql** 포드 정의는 MySQL 데이터베이스를 정의합니다.
 - **todoapi** 서비스는 Node.js 애플리케이션 포드에 연결을 제공합니다.
 - **mysql** 서비스는 MySQL 데이터베이스 포드에 연결을 제공합니다.
 - **dbinit** 영구 볼륨 클레임 정의는 MySQL `/var/lib/mysql/init` 볼륨을 정의합니다.
 - **dbclaim** 영구 볼륨 클레임 정의는 MySQL `/var/lib/mysql/data` 볼륨을 정의합니다.
- 4.4. 템플릿을 처리하고 애플리케이션 리소스를 생성합니다.
`oc process` 명령을 사용하여 템플릿 파일을 처리합니다. 이 템플릿에는 Quay.io 네임스페이스가 있어야 **RHT_OCP4_QUAY_USER** 매개 변수로 컨테이너 이미지를 검색할 수 있습니다.
`pipe` 명령을 사용하여 `oc create` 명령에 결과를 전송합니다.
터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation nodejs]$ cd /home/student/D0180/labs/multicontainer-openshift  
[student@workstation multicontainer-openshift]$ oc process \  
> -f todo-template.json -p RHT_OCP4_QUAY_USER=${RHT_OCP4_QUAY_USER} \  
> | oc create -f -  
pod/mysql created  
pod/todoapi created  
service/todoapi created  
service/mysql created  
persistentvolumeclaim/dbinit created  
persistentvolumeclaim/dbclaim created
```

- 4.5. 배포를 검토합니다.
포드 상태를 계속 모니터링하기 위해 `oc get pods` 명령에 `-w` 옵션을 사용하여 배포 상태를 검토합니다. 컨테이너가 둘 다 실행될 때까지 기다립니다. 두 포드가 모두 시작될 때까지 다소 시간이 걸릴 수 있습니다.

```
[student@workstation multicontainer-openshift]$ oc get pods -w
NAME      READY   STATUS            RESTARTS   AGE
mysql     0/1    ContainerCreating   0          27s
todoapi   1/1    Running           0          27s
mysql     1/1    Running           0          27s
```

Ctrl+C를 눌러 명령을 종료합니다.

▶ 5. 서비스를 노출합니다.

To Do List 애플리케이션에 OpenShift 라우터를 통해 액세스하고 공개 FQDN으로 사용할 수 있도록 하용하려면 **oc expose** 명령을 사용하여 **todoapi** 서비스를 노출합니다.

터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation multicontainer-openshift]$ oc expose service todoapi
route.route.openshift.io/todoapi exposed
```

▶ 6. 애플리케이션을 테스트합니다.

6.1. **oc status** 명령을 실행하여 애플리케이션의 FQDN을 찾고, 앱의 FQDN을 확인합니다.

터미널 창에서 다음 명령을 실행합니다.

```
[student@workstation multicontainer-openshift]$ oc status | grep -o "http:.com"
http://todoapi-$(RHT_OCP4_QUAY_USER)-template.$(RHT_OCP4_WILDCARD_DOMAIN}
```

6.2. **curl**을 사용하여 To Do List 애플리케이션의 REST API를 테스트합니다.

```
[student@workstation multicontainer-openshift]$ curl -w "\n" \
> $(oc status | grep -o "http:.com")/todo/api/items/1
{"id":1,"description":"Pick up newspaper","done":false}
```

curl 명령과 함께 **-w "\n"** 옵션을 사용하면 동일한 행에서 출력과 함께 병합되는 것이 아니라 다음 행에 쉘 프롬프트가 나타납니다.

6.3. 워크스테이션에서 Firefox를 열고 브라우저에서 [http://todoapi-\\$\(RHT_OCP4_QUAY_USER\)-template.\\$\(RHT_OCP4_WILDCARD_DOMAIN\)/todo/](http://todoapi-$(RHT_OCP4_QUAY_USER)-template.$(RHT_OCP4_WILDCARD_DOMAIN)/todo/)를 지정하면 To Do List 애플리케이션이 표시됩니다.



참고

위에서 언급한 URL에 후행 슬래시가 필요합니다. URL에 후행 슬래시를 포함하지 않으면 애플리케이션에 문제가 발생할 수 있습니다.

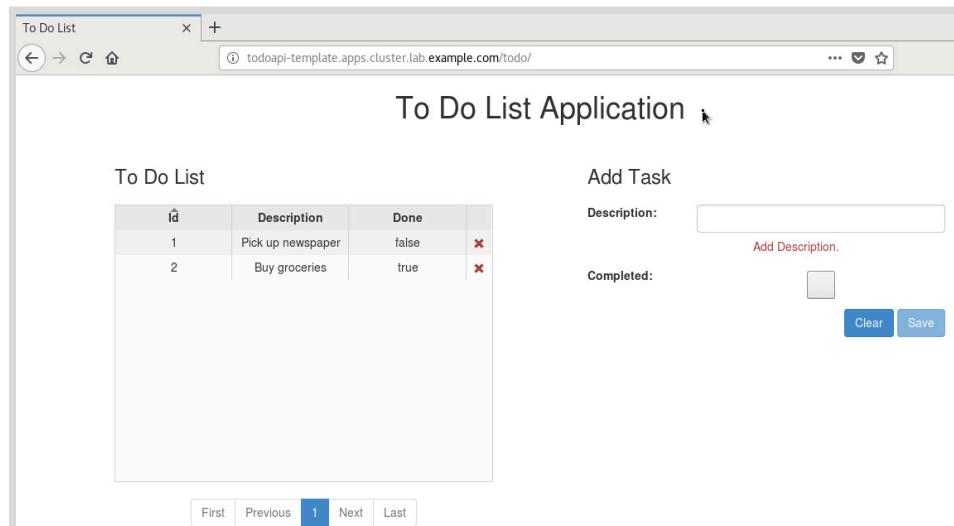


그림 7.1: To Do List 애플리케이션

완료

workstation에서 `lab multicontainer-openshift finish` 스크립트를 실행하여 랩을 완료합니다.

```
[student@workstation ~]$ lab multicontainer-openshift finish
```

이로써 안내에 따른 연습이 완료됩니다.

▶ 랩

소프트웨어 애플리케이션 컨테이너화 및 배포

이 검토에서는 Nexus Server를 컨테이너화하고 Podman 사용을 빌드 및 테스트하고, OpenShift 클러스터에 배포합니다.

결과

다음을 수행할 수 있습니다.

- Nexus 서버를 성공적으로 컨테이너화하는 Dockerfile을 작성합니다.
- Nexus 서버 컨테이너 이미지를 빌드하고 Podman을 사용하여 배포합니다.
- Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포합니다.

시작하기 전에

이 종합 검토에 대해 설정 스크립트를 실행합니다.

```
[student@workstation ~]$ lab comprehensive-review start
```

랩 파일은 `/home/student/D0180/labs/comprehensive-review` 디렉터리에 있습니다.
솔루션 파일은 `/home/student/D0180/solutions/comprehensive-review` 디렉터리에 있습니다.

지침

다음 단계를 따라 컨테이너화된 Nexus 서버를 로컬 및 OpenShift 모두에서 생성하고 테스트하십시오.

단계

1. Nexus 서버의 인스턴스를 시작하는 컨테이너 이미지를 생성합니다.
 - `/home/student/D0180/labs/comprehensive-review/image` 디렉터리에는 컨테이너 이미지를 빌드하는 데 필요한 파일이 들어있습니다. `get-nexus-bundle.sh` 스크립트를 실행하여 Nexus 서버 파일을 검색합니다.
 - Nexus 서버를 컨테이너화하는 Dockerfile을 작성합니다. Docker 파일은 `/home/student/D0180/labs/comprehensive-review/image` 디렉터리에 있습니다. 또한 Dockerfile은 다음과 같아야 합니다.
 - 기본 이미지 `ubi7/ubi:7.7`을 사용하고 임의 관리자를 설정합니다.
 - 환경 변수 `NEXUS_VERSION`을 `2.14.3-02`로 설정하고 `NEXUS_HOME`을 `/opt/nexus`로 설정합니다.
 - `java-1.8.0-openjdk-devel` 패키지 설치
 RPM 리포지토리는 제공된 `training.repo` 파일에 구성됩니다. 이 파일은 `/etc/yum.repos.d` 디렉터리의 컨테이너에 추가해야 합니다.
 - 명령을 실행하여 `nexus` 사용자 및 그룹을 만듭니다. 둘 다 `1001`이라는 UID 및 GID가 있습니다.

- \${NEXUS_HOME}/ 디렉터리에 **nexus-2.14.3-02-bundle.tar.gz** 파일의 압축을 풉니다. **nexus-start.sh**를 동일한 디렉터리에 추가합니다.

```
ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} ${NEXUS_HOME}/nexus2
명령을 실행하여 컨테이너에 심볼릭 링크를 만듭니다. Nexus 홈 디렉터리의 소유권을
nexus:nexus로 재귀적으로 변경하는 명령을 실행합니다.
```

- 컨테이너가 **nexus** 사용자로 실행되도록 설정하고 작업 디렉터리를 **/opt/nexus**로 설정합니다.
- **/opt/nexus/sonatype-work** 컨테이너 디렉터리의 볼륨 마운트 지점을 정의합니다. Nexus 서버는 이 디렉터리에 데이터를 저장합니다.
- 기본 컨테이너 명령을 **nexus-start.sh**로 설정합니다.

Nexus 계정을 생성하고 Java를 설치하는 데 필요한 명령을 제공하는 **/home/student/D0180/labs/comprehensive-review/images** 디렉터리에는 두 개의 *.snippet 파일이 있습니다. 해당 파일을 사용하면 Dockerfile을 작성하는데 도움이 됩니다.

- 이름이 **nexus**인 컨테이너 이미지를 빌드합니다.

2. 볼륨 마운트가 포함된 Podman을 사용하여 컨테이너 이미지를 빌드 및 테스트합니다.

- **/home/student/D0180/labs/comprehensive-review/deploy/local/run-persistent.sh** 스크립트를 사용하여 볼륨 마운트가 포함된 새 컨테이너를 시작합니다.
- 컨테이너 로그를 검토하여 서버가 시작되어 실행 중인지 확인합니다.
- URL을 사용하여 컨테이너 서비스에 대한 액세스를 테스트합니다. **http://<container IP address>:8081/nexus**
- 테스트 컨테이너를 제거합니다.

3. Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포합니다. 다음을 수행해야 합니다.

- Nexus 서버 컨테이너 이미지에 **quay.io/\${RHT_OCP4_QUAY_USER}/nexus:latest** 태그를 지정하고 프라이빗 레지스트리에 내보냅니다.
- 이름이 **\${RHT_OCP4_DEV_USER}-review**인 OpenShift 프로젝트를 만듭니다.
- **deploy/openshift/resources/nexus-template.json** 템플릿을 처리하고 Kubernetes 리소스를 생성합니다.
- Nexus 서비스에 대한 경로를 만듭니다. **workstation**에서 **http://nexus- \${RHT_OCP4_DEV_USER}-review.\${RHT_OCP4_WILDCARD_DOMAIN}/nexus/**에 액세스할 수 있는지 확인합니다.

평가

Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포한 후 랩 평가 스크립트를 실행하여 작업 내용을 확인합니다.

```
[student@workstation ~]$ lab comprehensive-review grade
```

완료

workstation에서 **lab comprehensive-review finish** 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab comprehensive-review finish
```

이로써 랩이 완료됩니다.

▶ 솔루션

소프트웨어 애플리케이션 컨테이너화 및 배포

이 검토에서는 Nexus Server를 컨테이너화하고 Podman 사용을 빌드 및 테스트하고, OpenShift 클러스터에 배포합니다.

결과

다음을 수행할 수 있습니다.

- Nexus 서버를 성공적으로 컨테이너화하는 Dockerfile을 작성합니다.
- Nexus 서버 컨테이너 이미지를 빌드하고 Podman을 사용하여 배포합니다.
- Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포합니다.

시작하기 전에

이 종합 검토에 대해 설정 스크립트를 실행합니다.

```
[student@workstation ~]$ lab comprehensive-review start
```

랩 파일은 `/home/student/D0180/labs/comprehensive-review` 디렉터리에 있습니다.
솔루션 파일은 `/home/student/D0180/solutions/comprehensive-review` 디렉터리에 있습니다.

지침

다음 단계를 따라 컨테이너화된 Nexus 서버를 로컬 및 OpenShift 모두에서 생성하고 테스트하십시오.

단계

1. Nexus 서버의 인스턴스를 시작하는 컨테이너 이미지를 생성합니다.

- `/home/student/D0180/labs/comprehensive-review/image` 디렉터리에는 컨테이너 이미지를 빌드하는 데 필요한 파일이 들어있습니다. `get-nexus-bundle.sh` 스크립트를 실행하여 Nexus 서버 파일을 검색합니다.
- Nexus 서버를 컨테이너화하는 Dockerfile을 작성합니다. Docker 파일은 `/home/student/D0180/labs/comprehensive-review/image` 디렉터리에 있습니다. 또한 Dockerfile은 다음과 같아야 합니다.
 - 기본 이미지 `ubi7/ubi:7.7`을 사용하고 임의 관리자를 설정합니다.
 - 환경 변수 `NEXUS_VERSION`을 `2.14.3-02`로 설정하고 `NEXUS_HOME`을 `/opt/nexus`로 설정합니다.
 - `java-1.8.0-openjdk-devel` 패키지 설치
 RPM 리포지토리는 제공된 `training.repo` 파일에 구성됩니다. 이 파일은 `/etc/yum.repos.d` 디렉터리의 컨테이너에 추가해야 합니다.
- 명령을 실행하여 `nexus` 사용자 및 그룹을 만듭니다. 둘 다 `1001`이라는 UID 및 GID가 있습니다.

7장 | 멀티컨테이너 애플리케이션 배포

- \${NEXUS_HOME}/ 디렉터리에 **nexus-2.14.3-02-bundle.tar.gz** 파일의 압축을 풉니다. **nexus-start.sh**를 동일한 디렉터리에 추가합니다.
- ```
ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} ${NEXUS_HOME}/nexus2
```
- 명령을 실행하여 컨테이너에 심볼릭 링크를 만듭니다. Nexus 홈 디렉터리의 소유권을 **nexus:nexus**로 재귀적으로 변경하는 명령을 실행합니다.
- 컨테이너가 **nexus** 사용자로 실행되도록 설정하고 작업 디렉터리를 **/opt/nexus**로 설정합니다.
  - **/opt/nexus/sonatype-work** 컨테이너 디렉터리의 볼륨 마운트 지점을 정의합니다. Nexus 서버는 이 디렉터리에 데이터를 저장합니다.
  - 기본 컨테이너 명령을 **nexus-start.sh**로 설정합니다.

Nexus 계정을 생성하고 Java를 설치하는 데 필요한 명령을 제공하는 **/home/student/D0180/labs/comprehensive-review/images** 디렉터리에는 두 개의 \*.snippet 파일이 있습니다. 해당 파일을 사용하면 Dockerfile을 작성하는데 도움이 됩니다.

- 이름이 **nexus**인 컨테이너 이미지를 빌드합니다.

- 1.1. **get-nexus-bundle.sh** 스크립트를 실행하여 Nexus 서버 파일을 검색합니다.

```
[student@workstation ~]$ cd /home/student/D0180/labs/comprehensive-review/image
[student@workstation image]$./get-nexus-bundle.sh
#####
Nexus bundle download successful
```

- 1.2. Nexus 서버를 컨테이너화하는 Dockerfile을 작성합니다. **/home/student/D0180/labs/comprehensive-review/images** 디렉터리로 이동하여 Dockerfile을 만듭니다.

- 1.2.1. 사용할 기본 이미지를 지정합니다.

```
FROM ubi7/ubi:7.7
```

- 1.2.2. 관리자 임의 이름 및 이메일을 입력합니다.

```
FROM ubi7/ubi:7.7
```

```
MAINTAINER username <username@example.com>
```

- 1.2.3. **NEXUS\_VERSION**의 빌드 인수 및 **NEXUS\_HOME**의 환경 변수를 설정합니다.

```
FROM ubi7/ubi:7.7
```

```
MAINTAINER username <username@example.com>
```

```
ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus
```

- 1.2.4. **/etc/yum.repos.d** 디렉터리에 **training.repo** 리포지토리를 추가합니다. **yum** 명령을 사용하여 Java 패키지를 설치합니다.

```
...
ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus

RUN yum install -y --setopt=tsflags=nodocs java-1.8.0-openjdk-devel && \
 yum clean all -y
```

1.2.5. 서버 홈 디렉터리 및 서비스 계정과 그룹을 생성합니다. 서비스 계정이 소유하는 홈 디렉터리를 만듭니다.

```
...
RUN groupadd -r nexus -f -g 1001 && \
 useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} -s /sbin/nologin \
 -c "Nexus User" nexus && \
 chown -R nexus:nexus ${NEXUS_HOME} && \
 chmod -R 755 ${NEXUS_HOME}
```

1.2.6. 컨테이너를 **nexus** 사용자로 실행하도록 합니다.

```
USER nexus
```

1.2.7. **NEXUS\_HOME**에 Nexus 서버 소프트웨어를 설치하고 시작 스크립트를 추가합니다. **ADD** 지시어는 Nexus 파일을 추출합니다.

Nexus 서버 디렉터리를 가리키는 **nexus2** 심볼 링크를 생성합니다.  **\${NEXUS\_HOME}** 디렉터리의 소유권을 **nexus:nexus**로 재귀적으로 변경합니다.

```
...
ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}
ADD nexus-start.sh ${NEXUS_HOME}/

RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
 ${NEXUS_HOME}/nexus2
```

1.2.8. **/opt/nexus**를 현재 작업 디렉터리로 만듭니다.

```
WORKDIR ${NEXUS_HOME}
```

1.2.9. Nexus 서버 영구 데이터를 저장하도록 볼륨 마운트 지점을 정의합니다.

```
VOLUME ["/opt/nexus/sonatype-work"]
```

1.2.10 **CMD** 명령어를 **["sh", "nexus-start.sh"]**로 설정하십시오. 작성된 Dockerfile 파일은 다음과 같이 표시됩니다.

```

FROM ubi7/ubi:7.7

MAINTAINER username <username@example.com>

ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus

RUN yum install -y --setopt=tsflags=nodocs java-1.8.0-openjdk-devel && \
 yum clean all -y

RUN groupadd -r nexus -f -g 1001 && \
 useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} -s /sbin/nologin \
 -c "Nexus User" nexus && \
 chown -R nexus:nexus ${NEXUS_HOME} && \
 chmod -R 755 ${NEXUS_HOME}

USER nexus

ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}
ADD nexus-start.sh ${NEXUS_HOME}/

RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
 ${NEXUS_HOME}/nexus2

WORKDIR ${NEXUS_HOME}

VOLUME ["/opt/nexus/sonatype-work"]

CMD ["sh", "nexus-start.sh"]

```

1.3. 이름이 **nexus**인 컨테이너 이미지를 빌드합니다.

```

[student@workstation image]$ sudo podman build --layers=false -t nexus .
STEP 1: FROM ubi7/ubi:7.7
Getting image source signatures
...output omitted...
STEP 14: COMMIT ...output omitted...localhost/nexus:latest
...output omitted...

```

2. 볼륨 마운트가 포함된 Podman을 사용하여 컨테이너 이미지를 빌드 및 테스트합니다.

- `/home/student/D0180/labs/comprehensive-review/deploy/local/run-persistent.sh` 스크립트를 사용하여 볼륨 마운트가 포함된 새 컨테이너를 시작합니다.
- 컨테이너 로그를 검토하여 서버가 시작되어 실행 중인지 확인합니다.
- URL을 사용하여 컨테이너 서비스에 대한 액세스를 테스트합니다. `http://<container IP address>:8081/nexus`
- 테스트 컨테이너를 제거합니다.

2.1. `run-persistent.sh` 스크립트를 실행합니다. `podman ps` 명령 출력에 표시된 대로 컨테이너 이름을 교체합니다.

```
[student@workstation images]$ cd /home/student/D0180/labs/comprehensive-review
[student@workstation comprehensive-review]$ cd deploy/local
[student@workstation local]$./run-persistent.sh
80970007036bbb313d8eeb7621fada0ed3f0b4115529dc50da4dccef0da34533
```

2.2. 컨테이너 로그를 검토하여 서버가 시작되어 실행 중인지 확인합니다.

```
[student@workstation local]$ sudo podman ps \
> --format="{{.ID}} {{.Names}} {{.Image}}"
81f480f21d47 inspiring_poincare localhost/nexus:latest
[student@workstation local]$ sudo podman logs inspiring_poincare
...output omitted...
... INFO [jetty-main-1] ...jetty.JettyServer - Running
... INFO [main] ...jetty.JettyServer - Started
```

2.3. 실행 중인 컨테이너를 검사하여 IP 주소를 확인합니다. 이 IP 주소를 **curl** 명령에 제공하여 컨테이너를 테스트합니다.

```
[student@workstation local]$ sudo podman inspect \
> -f '{{.NetworkSettings.IPAddress}}' inspiring_poincare
10.88.0.12
[student@workstation local]$ curl -v 10.88.0.12:8081/nexus/ 2>&1 \
> | grep -E 'HTTP|<title>'
> GET /nexus/ HTTP/1.1
< HTTP/1.1 200 OK
<title>Nexus Repository Manager</title>
```

2.4. 테스트 컨테이너를 제거합니다.

```
[student@workstation local]$ sudo podman kill inspiring_poincare
81f480f21d475af683b4b003ca6e002d37e6aaa581393d3f2f95a1a7b7eb768b
```

**3.** Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포합니다. 다음을 수행해야 합니다.

- Nexus 서버 컨테이너 이미지에 **quay.io/\${RHT\_OCP4\_QUAY\_USER}/nexus:latest** 태그를 지정하고 프라이빗 레지스트리에 내보냅니다.
- 이름이  **\${RHT\_OCP4\_DEV\_USER} - review**인 OpenShift 프로젝트를 만듭니다.
- deploy/openshift/resources/nexus-template.json** 템플릿을 처리하고 Kubernetes 리소스를 생성합니다.
- Nexus 서비스에 대한 경로를 만듭니다.  **workstation**에서 **http://nexus- \${RHT\_OCP4\_DEV\_USER}-review.\${RHT\_OCP4\_WILDCARD\_DOMAIN}/nexus/**에 액세스할 수 있는지 확인합니다.

3.1. Quay.io 계정에 로그인합니다.

```
[student@workstation local]$ sudo podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password: your_quay_password
Login Succeeded!
```

3.2. Nexus 서버 컨테이너 이미지를 quay.io 레지스트리에 게시합니다.

```
[student@workstation local]$ sudo podman push localhost/nexus:latest \
> quay.io/${RHT_OCP4_QUAY_USER}/nexus:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

3.3. 이미지를 quay.io에 내보내 만든 리포지토리는 기본적으로 비공개입니다. 부록 C의 **Repositories Visibility**(리포지토리 가시성) 섹션을 참조하여 리포지토리 가시성을 변경하는 방법에 대한 세부 정보를 읽어 보십시오.

3.4. OpenShift 프로젝트를 생성합니다.

```
[student@workstation local]$ cd ~/D0180/labs/comprehensive-review/deploy/openshift
[student@workstation openshift]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation openshift]$ oc new-project ${RHT_OCP4_DEV_USER}-review
Now using project ...output omitted...
```

3.5. 템플릿을 진행하고 Kubernetes 리소스를 생성합니다.

```
[student@workstation openshift]$ oc process -f resources/nexus-template.json \
> -p RHT_OCP4_QUAY_USER=${RHT_OCP4_QUAY_USER} \
> | oc create -f -
service/nexus created
persistentvolumeclaim/nexus created
deploymentconfig.apps.openshift.io/nexus created
[student@workstation openshift]$ oc get pods
NAME READY STATUS RESTARTS AGE
nexus-1-wk8rv 1/1 Running 1 1m
nexus-1-deploy 0/1 Completed 0 2m
```

3.6. 경로를 생성하여 서비스를 노출합니다.

```
[student@workstation openshift]$ oc expose svc/nexus
route.route.openshift.io/nexus exposed.
[student@workstation openshift]$ oc get route -o yaml
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
 kind: Route
...output omitted...
spec:
 host: nexus-${RHT_OCP4_DEV_USER}-review.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
```

- 3.7. 브라우저를 사용하여 `http://nexus-$\{RHT_OCP4_DEV_USER\}-review.`  
`$\{RHT_OCP4_WILDCARD_DOMAIN\}/nexus/`에서 Nexus 서버 웹 애플리케이션에 연결합니다.

## 평가

Nexus 서버 컨테이너 이미지를 OpenShift 클러스터에 배포한 후 랩 평가 스크립트를 실행하여 작업 내용을 확인합니다.

```
[student@workstation ~]$ lab comprehensive-review grade
```

## 완료

workstation에서 `lab comprehensive-review finish` 명령을 실행하여 이 랩을 완료합니다.

```
[student@workstation ~]$ lab comprehensive-review finish
```

이로써 랩이 완료됩니다.

## 요약

---

이 장에서 학습한 내용:

- Kubernetes와 OpenShift는 포드의 모든 컨테이너 사이에 소프트웨어 정의 네트워크를 생성합니다.
- 동일한 프로젝트 내에서 Kubernetes는 각 서비스의 변수 집합을 모든 포드에 삽입합니다.
- Kubernetes 템플릿은 다양한 리소스로 구성되는 애플리케이션 생성을 자동화합니다. 템플릿 매개 변수를 사용하면 여러 리소스를 만들 때 동일한 값을 사용할 수 있습니다.

## 8장

# Red Hat OpenShift Container Platform 설명

### 목적

OpenShift Container Platform의 아키텍처를 설명합니다.

### 목표

- 제품의 일반 사용법과 해당 기능을 설명합니다.
- Red Hat OpenShift Container Platform의 아키텍처를 설명합니다.
- 클러스터 운영자의 정의와 작동 방식을 설명하고 주요 클러스터 운영자의 이름을 밝힙니다.

### 섹션

- OpenShift Container Platform 기능 설명 및 퀴즈
- OpenStack 아키텍처 설명 및 퀴즈
- 클러스터 운영자 설명 및 퀴즈

# OpenShift Container Platform 기능 설명

## 목표

이 섹션을 마친 수강생은 제품의 일반적인 사용과 해당 기능에 대해 설명할 수 있습니다.

## OpenShift Container Platform 소개

컨테이너 오케스트레이션은 디지털 혁신 이니셔티브의 근본적인 원동력입니다. 하지만 컨테이너화된 서비스로 전환하는 모놀리식 애플리케이션으로서, 기존 인프라를 사용하여 이러한 애플리케이션을 관리하는 것은 지루한 작업이 될 수 있습니다. RHOC(Red Hat OpenShift Container Platform)를 사용하면 개발자와 IT 조직에서 애플리케이션 라이프사이클을 더 효율적으로 관리할 수 있습니다.

RHOC는 Kubernetes 오픈소스 프로젝트를 기반으로 하며, 강력하고 유연하며 확장 가능한 컨테이너 플랫폼을 고객 데이터 센터에 제공하는 기능을 통해 플랫폼을 확장하므로, 개발자가 고가용성 환경에서 워크로드를 실행할 수 있습니다.

OpenShift Container Platform과 같은 컨테이너 오케스트레이터에서는 컨테이너화된 여러 애플리케이션을 실행하는 서버 클러스터를 관리합니다. Red Hat OpenShift 제품군에는 다양한 환경에서 비즈니스 애플리케이션을 제공하는 기능을 향상시키는 일련의 솔루션이 포함되어 있습니다.

### Red Hat OpenShift Container Platform

베어 메탈 서버를 포함하여 공용 또는 사설 데이터 센터에서 컨테이너 기반 애플리케이션을 구축, 배포 및 관리할 수 있는 엔터프라이즈급 Kubernetes 환경을 제공합니다. RHOC는 여러 클라우드 및 가상화 프로바이더와 호환되므로, 애플리케이션 개발자와 관리자가 이러한 프로바이더 간 차이의 영향을 받지 않습니다. 최신 릴리스로 업데이트할 시기와 활성화할 추가 구성요소를 결정합니다.

### Red Hat OpenShift Dedicated

AWS(Amazon Web Services), GCP(Google Cloud Platform), Microsoft Azure 또는 IBM Cloud와 같은 퍼블릭 클라우드에서 관리형 OpenShift 환경을 제공합니다. 이 제품에서는 RHOC에서 제공하는 모든 기능을 제공하지만 Red Hat에서 클러스터를 관리합니다. OpenShift의 최신 릴리스로 업데이트하거나 애드온 서비스를 설치하는 경우 일부 의사결정을 계속 제어할 수 있습니다.

### Red Hat OpenShift Online

클라우드 환경에서 애플리케이션 개발, 빌드, 배포 및 호스팅 솔루션을 제공하는 호스팅된 공용 컨테이너 오케스트레이션 플랫폼을 제공합니다. 이 솔루션은 여러 고객 간에 공유되며, Red Hat에서는 업데이트 적용 또는 새로운 기능 통합을 포함하여 클러스터 라이프 사이클을 관리합니다.

### Red Hat OpenShift Kubernetes 엔진

OpenShift Container Platform에 있는 하위 기능 집합(Red Hat Enterprise Linux CoreOS 경량 트랜잭션 운영 체제, CRI-O 엔진, Kubernetes 컨테이너 오케스트레이션 플랫폼, 코어 클러스터 서비스(웹 콘솔, 공중파 업데이트, 내부 레지스트리, 운영자 라이프사이클 관리자 등))을 제공합니다.

### Red Hat Code Ready Containers

로컬 개발 및 시험을 위해 랩톱에서 실행할 수 있는 OpenShift의 최소 설치를 제공합니다.

또한 일부 클라우드 프로바이더에서는 RHOC를 기반으로 하며, 플랫폼의 다른 서비스와 긴밀하게 통합하는 기능을 추가하고, Red Hat과의 파트너 관계를 통해 프로바이더에서 지원하는 서비스를 제공합니다. 한 가지 예로 Microsoft Azure Red Hat OpenShift가 있습니다.

다음 그림은 OpenShift의 서비스와 기능을 설명합니다.

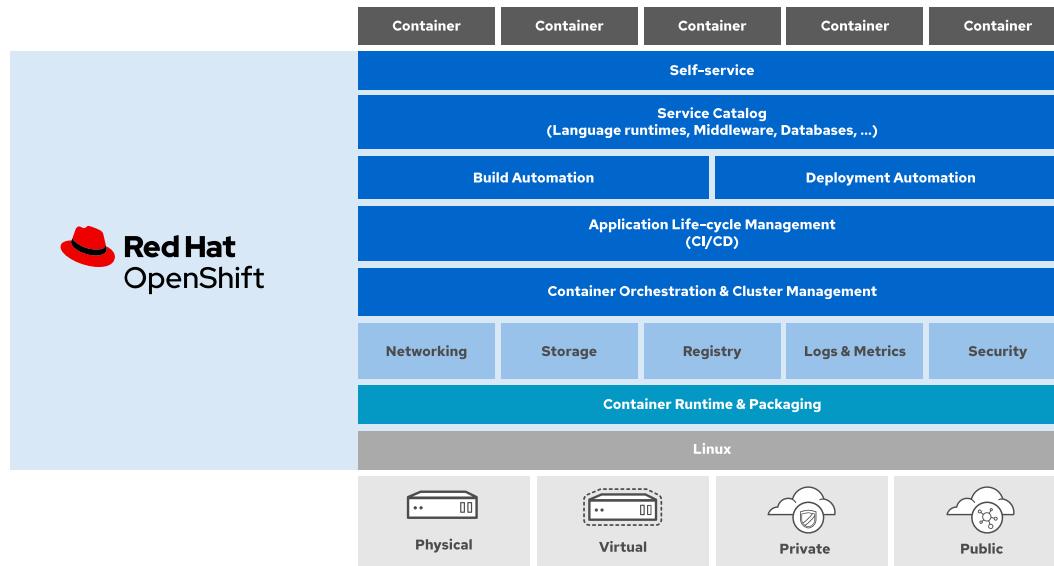


그림 8.1: OpenShift 서비스 및 기능

Red Hat OpenShift 제품군은 다음과 같은 많은 구성 요소를 통합합니다.

- Red Hat Enterprise Linux CoreOS 컨테이너에 최적화된 변경 불가능한 운영 체제
- CRI-O는 OCI(Open Container Initiative) 컨테이너 런타임 엔진으로, 크기가 매우 작아 공격 가능한 범위가 작습니다.
- Kubernetes(오픈소스 컨테이너 오피스트레이션 플랫폼)
- 셸프 서비스 웹 콘솔
- 내부 컨테이너 이미지 레지스트리 및 모니터링 프레임워크와 같은 사전 설치된 여러 애플리케이션 서비스
- 여러 프로그래밍 언어 런타임, 데이터베이스, 기타 소프트웨어 패키지의 공인 컨테이너 이미지

## OpenShift 기능 소개

OpenShift에서는 애플리케이션을 자동화, 확장 및 유지 관리하는 많은 기능을 제공합니다. 이러한 모든 기능은 Kubernetes에서 활성화하며, 대부분에는 BYO(Build-Your-Own) Kubernetes 설정에서 추가하고 구성해야 하는 추가 구성 요소가 필요합니다.

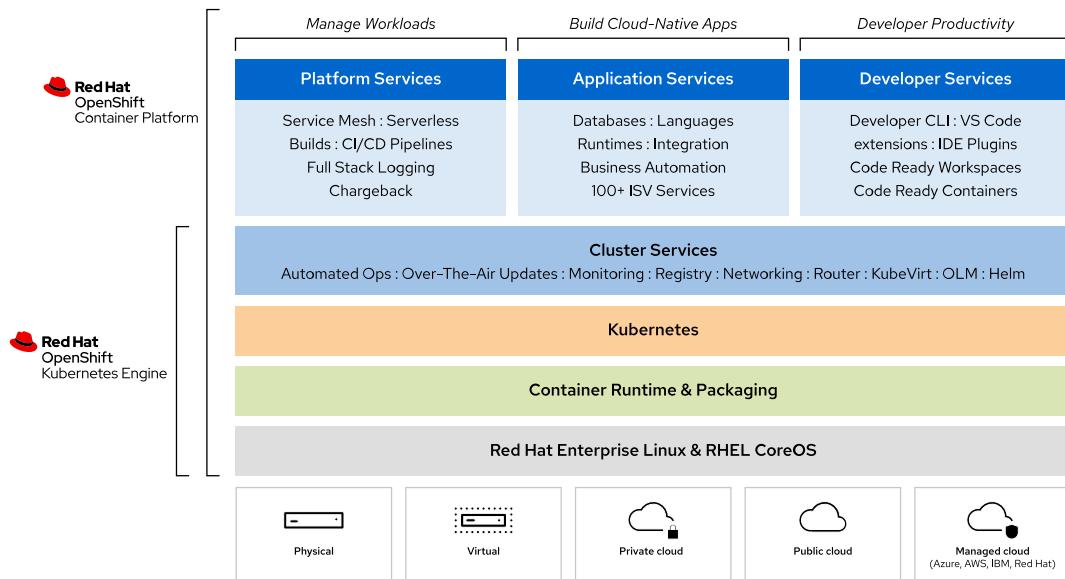


그림 8.2: OpenShift Container Platform과 OpenShift Kubernetes 엔진의 기능 차이

## 고가용성

Kubernetes는 내부 구성 요소 및 사용자 애플리케이션 모두의 고가용성을 염두에 두고 설계되었습니다. 고도로 사용 가능한 etcd 클러스터에서 OpenShift 클러스터 및 해당 애플리케이션의 상태를 저장합니다.

etcd에 저장된 배포 구성 등의 리소스에서는 애플리케이션을 항상 실행하고 결함이 있는 컨테이너를 종료하기 위해 컨테이너를 자동으로 다시 시작하는 기능을 제공합니다. 이 기능은 애플리케이션뿐만 아니라 웹 콘솔과 내부 이미지 레지스트리와 같이 클러스터를 구성하는 컨테이너화된 서비스에도 적용됩니다.

## 경량 운영 체제

RHOCP는 민첩성, 이식성 및 보안에 중점을 둔 Red Hat의 경량 운영 체제인 Red Hat Enterprise Linux CoreOS에서 실행됩니다.

RHEL CoreOS(Red Hat Enterprise Linux CoreOS)는 컨테이너화된 애플리케이션을 실행하도록 최적화된 변경 불가능 운영 체제입니다. 전체 운영 체제는 패키지별이 아니라 단일 이미지로 업데이트되며, 사용자 애플리케이션과 시스템 구성 요소(예: 네트워크 서비스)가 모두 컨테이너로 실행됩니다.

RHOCP에서는 RHEL CoreOS 및 해당 구성의 업데이트를 제어하므로, OpenShift 클러스터 관리에는 클러스터 노드에서 운영 체제를 관리하고, 시스템 관리자가 이러한 작업을 수행할 필요를 없애며, 사용자 오류의 위험을 줄이는 작업이 포함됩니다.

## 부하 분산

클러스터에서는 세 가지 유형의 로드 밸런서를 제공합니다. 즉, OpenShift API에 대한 액세스를 관리하는 외부 로드 밸런서, 외부에서 애플리케이션에 액세스하는 데 사용하는 HAProxy 로드 밸런서, 내부에서 애플리케이션과 서비스에 액세스하는 데 Netfilter 규칙을 사용하는 내부 로드 밸런서입니다.

경로 리소스에서는 HAProxy를 사용하여 클러스터에 대한 외부 액세스를 관리합니다. 서비스 리소스에서는 Netfilter 규칙을 사용하여 클러스터 내부에서 트래픽을 관리합니다. 외부 부하 분산 장치에서 사용하는 기술은 클러스터를 실행하는 클라우드 프로바이더에 따라 다릅니다.

## 확장 자동화

OpenShift 클러스터는 새 컨테이너를 자동으로 시작하고 부하가 감소하는 경우 컨테이너를 종료하여 실시간으로 증가된 애플리케이션 트래픽에 맞게 조정할 수 있습니다. 이 기능을 사용하면 동시 연결 또는 활동 수에 관계없이 애플리케이션 액세스 시간이 최적으로 유지됩니다.

또한 OpenShift 클러스터에서는 많은 애플리케이션에서 집계된 부하에 따라 클러스터에서 더 많은 작업자 노드를 추가하거나 제거하여 퍼블릭 및 프라이빗 클라우드의 응답성을 유지하고 계속 비용을 줄입니다.

## 로깅 및 모니터링

RHOCP에는 클러스터에 관한 수백 개의 지표를 수집하는 Prometheus를 기반으로 하는 고급 모니터링 솔루션이 함께 제공됩니다. 이 솔루션은 클러스터 활동 및 상태에 관한 세부 정보를 얻을 수 있는 경고 시스템과 상호 작용합니다.

RHOCP에는 Elasticsearch를 기반으로 하는 고급 집계 로깅 솔루션이 함께 제공되므로, 클러스터 노드 및 컨테이너의 장기 로그를 유지할 수 있습니다.

## 서비스 검색

RHOCP는 클러스터에서 내부 DNS 서비스를 실행하고, 해당 내부 DNS를 사용하여 이름을 확인하도록 모든 컨테이너를 구성합니다. 즉, 애플리케이션에서 외부 서비스 카탈로그의 오버헤드 없이 친숙한 이름을 사용하여 다른 애플리케이션과 서비스를 찾을 수 있습니다.

## 스토리지

Kubernetes에서는 스토리지 백엔드와 스토리지 소비 사이에 추상화 계층을 추가합니다. 따라서 애플리케이션에서는 스토리지 백엔드에 독립적인 통합 스토리지 정의를 사용하여 장기, 단기, 블록 및 파일 기반 스토리지를 사용할 수 있습니다. 이 방법으로 애플리케이션은 특정 클라우드 프로바이더 스토리지 API에 종속되지 않을 수 있습니다.

RHOCP에는 널리 사용되는 클라우드 프로바이더 및 가상화 플랫폼에서 스토리지를 자동으로 프로비저닝하는데 사용할 수 있는 많은 스토리지 프로바이저를 포함하고 있으므로 클러스터 관리자가 독점 스토리지 어레이의 세부 사항을 관리할 필요가 없습니다.

## 애플리케이션 관리

RHOCP를 사용하면 개발자가 애플리케이션 개발 및 배포를 자동화할 수 있습니다. OpenShift S2I(Source-to-Image) 기능을 사용하여 소스 코드를 기반으로 컨테이너를 자동으로 빌드하고 OpenShift에서 실행합니다. 내부 레지스트리에서는 다시 사용할 수 있는 애플리케이션 컨테이너 이미지를 저장합니다. 그러면 애플리케이션을 게시하는데 소요되는 시간이 줄어듭니다.

웹 콘솔에서 액세스할 수 있는 개발자 카탈로그에서 애플리케이션 템플릿을 게시하고 액세스할 수 있습니다. 여기에서는 Python, Ruby, Java 및 Node.js와 같은 여러 런타임 언어와 데이터베이스 및 메시징 서버도 지원합니다. 애플리케이션 배포, 업데이트 및 모니터링을 위한 운영 인텔리전스를 포함하는 사전 패키징된 애플리케이션 및 서비스인 새 운영자를 설치하여 카탈로그를 확장할 수 있습니다.

## 클러스터 확장성

RHOCP에서는 확장 API 및 사용자 지정 리소스 정의와 같은 Kubernetes의 표준 확장 메커니즘을 사용하여 업스트림 Kubernetes에서는 달리 제공하지 않는 기능을 추가합니다. OpenShift에서는 이러한 확장 기능을 운영자로 패키징하여 설치, 업데이트 및 관리를 용이하게 합니다.

OpenShift에는 운영자로 패키징된 애플리케이션 및 인프라 구성 요소의 검색, 설치 및 업데이트를 용이하게 하는 OLM(Operator Lifecycle Manager)도 포함되어 있습니다.

Red Hat에서는 AWS, Google Cloud 및 Microsoft와 협업하여 <https://operatorhub.io>에서 액세스 가능한 OperatorHub를 출시했습니다. 이 플랫폼은 OpenShift 및 OLM을 포함하는 기타 Kubernetes 배포판과 호환되는 운영자를 위한 공용 리포지토리 및 마켓플레이스입니다.

Red Hat Marketplace는 OpenShift 클러스터에 배포할 수 있도록 Kubernetes 운영자로 패키징된 인증 소프트웨어에 액세스할 수 있는 플랫폼입니다. 이러한 인증 소프트웨어에는 자동 배포 및 통합된 경험을 위한 원활한 업그레이드가 포함됩니다.



### 참조

자세한 정보는 Red Hat OpenShift Container Platform 4.5 제품 문서 [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/)에 있습니다.

### Red Hat OpenShift Kubernetes 엔진

<https://www.openshift.com/products/kubernetes-engine>

## ▶ 퀴즈

# OpenShift Container Platform 기능 설명

다음 질문에 대한 올바른 답을 선택하십시오.

### ▶ 1. 다음 중 컨테이너 오케스트레이션 플랫폼을 가장 잘 설명하는 정의는 무엇입니까?

- a. 애플리케이션의 운영 지식을 넓히고 애플리케이션을 패키징하고 배포하는 방법을 제공합니다.
- b. 컨테이너화된 애플리케이션을 실행하는 서버 클러스터를 관리하는 데 사용할 수 있습니다. 셀프 서비스, 고가용성, 모니터링 및 자동화 등의 기능을 추가합니다.
- c. AWS, GCP 및 Microsoft Azure를 비롯한 다양한 클라우드 프로바이더에서 IaaS(Infrastructure-as-a-Service) 클러스터를 프로비저닝하는 데 사용할 수 있습니다.
- d. 개발자가 애플리케이션을 작성하고 패키징하여 운영자로서 운영자 카탈로그에 게시하는 데 사용할 수 있습니다.

### ▶ 2. 다음 중 애플리케이션에 고가용성을 제공하는 세 가지 주요 기능은 무엇입니까? (세 개를 선택하십시오.)

- a. OpenShift etcd 클러스터는 모든 노드에 대해 클러스터 상태를 사용 가능으로 유지합니다.
- b. OpenShift HAProxy 부하 분산 장치를 사용하면 외부에서 애플리케이션에 액세스할 수 있습니다.
- c. OpenShift에서는 클러스터 내부에서 부하 분산을 위해 애플리케이션에 액세스하는 서비스를 제공합니다.
- d. OpenShift 배포 구성에 따라 노드 손실 등이 발생하는 경우 애플리케이션 컨테이너를 다시 시작합니다.

### ▶ 3. OpenShift에 대한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. 개발자가 소스 코드 리포지토리에서 직접 클라우드 애플리케이션을 생성하고 시작할 수 있습니다.
- b. OpenShift에서는 기타 Kubernetes 배포판에서 사용할 수 없는 기능을 추가하기 위해 Kubernetes 를 패치합니다.
- c. OpenShift Dedicated에서는 Red Hat에서 관리하고 유지 관리하는 독점적 운영자 세트에 대한 액세스를 제공합니다. 따라서 사용자 환경에서 운영자의 보안을 유지하며 안전하게 실행할 수 있습니다.
- d. OpenShift 클러스터 관리자는 운영자 카탈로그에서 새 운영자를 검색하고 설치할 수 있습니다.

### ▶ 4. 다음 중 OpenShift 구성 요소에서 트래픽의 부하를 분산하는 데 사용하는 두 가지 서비스는 무엇입니까? (두 개를 선택하십시오.)

- a. 외부 부하 분산 장치를 통해 액세스할 수 있는 OpenShift API
- b. 부하 분산에 Netfilter를 사용하는 서비스
- c. 부하 분산에 HAProxy를 사용하는 서비스
- d. 부하 분산에 Netfilter를 사용하는 경로
- e. 부하 분산에 HAProxy를 사용하는 경로

▶ 5. OpenShift 고가용성 및 확장에 관한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. OpenShift에서는 기본적으로 고가용성을 제공하지 않습니다. 타사 고가용성 제품을 사용해야 합니다.
- b. OpenShift에서는 Prometheus의 지표를 사용하여 애플리케이션 포드를 동적으로 확장합니다.
- c. 고가용성 및 확장은 REST API를 노출하는 애플리케이션으로 한정됩니다.
- d. OpenShift에서는 요구에 따라 애플리케이션을 확장 및 축소할 수 있습니다.

## ▶ 솔루션

# OpenShift Container Platform 기능 설명

다음 질문에 대한 올바른 답을 선택하십시오.

### ▶ 1. 다음 중 컨테이너 오케스트레이션 플랫폼을 가장 잘 설명하는 정의는 무엇입니까?

- a. 애플리케이션의 운영 지식을 넓히고 애플리케이션을 패키징하고 배포하는 방법을 제공합니다.
- b. 컨테이너화된 애플리케이션을 실행하는 서버 클러스터를 관리하는 데 사용할 수 있습니다. 셀프 서비스, 고가용성, 모니터링 및 자동화 등의 기능을 추가합니다.
- c. AWS, GCP 및 Microsoft Azure를 비롯한 다양한 클라우드 프로바이더에서 IaaS(Infrastructure-as-a-Service) 클러스터를 프로비저닝하는 데 사용할 수 있습니다.
- d. 개발자가 애플리케이션을 작성하고 패키징하여 운영자로서 운영자 카탈로그에 게시하는 데 사용할 수 있습니다.

### ▶ 2. 다음 중 애플리케이션에 고가용성을 제공하는 세 가지 주요 기능은 무엇입니까? (세 개를 선택하십시오.)

- a. OpenShift etcd 클러스터는 모든 노드에 대해 클러스터 상태를 사용 가능으로 유지합니다.
- b. OpenShift HAProxy 부하 분산 장치를 사용하면 외부에서 애플리케이션에 액세스할 수 있습니다.
- c. OpenShift에서는 클러스터 내부에서 부하 분산을 위해 애플리케이션에 액세스하는 서비스를 제공합니다.
- d. OpenShift 배포 구성에 따라 노드 손실 등이 발생하는 경우 애플리케이션 컨테이너를 다시 시작합니다.

### ▶ 3. OpenShift에 대한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. 개발자가 소스 코드 리포지토리에서 직접 클라우드 애플리케이션을 생성하고 시작할 수 있습니다.
- b. OpenShift에서는 기타 Kubernetes 배포판에서 사용할 수 없는 기능을 추가하기 위해 Kubernetes 를 패치합니다.
- c. OpenShift Dedicated에서는 Red Hat에서 관리하고 유지 관리하는 독점적 운영자 세트에 대한 액세스를 제공합니다. 따라서 사용자 환경에서 운영자의 보안을 유지하며 안전하게 실행할 수 있습니다.
- d. OpenShift 클러스터 관리자는 운영자 카탈로그에서 새 운영자를 검색하고 설치할 수 있습니다.

### ▶ 4. 다음 중 OpenShift 구성 요소에서 트래픽의 부하를 분산하는 데 사용하는 두 가지 서비스는 무엇입니까? (두 개를 선택하십시오.)

- a. 외부 부하 분산 장치를 통해 액세스할 수 있는 OpenShift API
- b. 부하 분산에 Netfilter를 사용하는 서비스
- c. 부하 분산에 HAProxy를 사용하는 서비스
- d. 부하 분산에 Netfilter를 사용하는 경로
- e. 부하 분산에 HAProxy를 사용하는 경로

▶ 5. OpenShift 고가용성 및 확장에 관한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. OpenShift에서는 기본적으로 고가용성을 제공하지 않습니다. 타사 고가용성 제품을 사용해야 합니다.
- b. OpenShift에서는 Prometheus의 지표를 사용하여 애플리케이션 포드를 동적으로 확장합니다.
- c. 고가용성 및 확장은 REST API를 노출하는 애플리케이션으로 한정됩니다.
- d. OpenShift에서는 요구에 따라 애플리케이션을 확장 및 축소할 수 있습니다.

# OpenShift 아키텍처 설명

## 목표

이 섹션을 마치면 Red Hat OpenShift Container Platform의 아키텍처를 설명할 수 있습니다.

## Kubernetes의 선언적 아키텍처 소개

OpenShift의 아키텍처는 Kubernetes의 선언적 특성을 기반으로 합니다. 대부분의 시스템 관리자는 지정된 서버에서 컨테이너를 시작하고 중지하는 등, 시스템 상태를 간접적으로 변경하는 작업을 수행하는 명령형 아키텍처에 익숙합니다. 선언형 아키텍처에서는 사용자가 시스템의 상태를 변경하고 시스템에서 새 상태에 맞게 자체를 업데이트합니다. 예를 들어 Kubernetes를 사용하면 특정 컨테이너를 특정 조건에서 실행하도록 지정하는 포드 리소스를 정의합니다. 그러면 Kubernetes에서 이 특정 조건에 따라 해당 컨테이너를 실행할 수 있는 서버(노드)를 찾습니다.

선언형 아키텍처를 사용하면 명령형 아키텍처보다 관리하기 쉬운 자체 최적화 및 자동 복구 시스템을 사용할 수 있습니다.

Kubernetes는 배포된 애플리케이션 세트를 포함하여 클러스터의 상태를 etcd 데이터베이스에 저장된 리소스 세트로 정의합니다. Kubernetes에서는 이러한 리소스를 모니터링하고 클러스터의 현재 상태와 비교하는 컨트롤러도 실행합니다. 이러한 컨트롤러에서는 새 포드 리소스에서 새 컨테이너를 시작하는데 충분한 CPU 용량이 있는 노드를 찾는 등, 리소스 상태를 사용하여 클러스터의 상태를 조정하는 데 필요한 모든 작업을 수행합니다.

Kubernetes에서는 이러한 리소스를 관리하는 REST API를 제공합니다. 명령줄 인터페이스 또는 웹 콘솔을 사용하여 OpenShift 사용자가 수행하는 모든 작업은 이 REST API를 호출하여 수행합니다.

## OpenShift Control Plane 소개

Kubernetes 클러스터는 kubelet 시스템 서비스와 컨테이너 엔진을 실행하는 노드 세트로 구성됩니다. OpenShift에서는 CRI-O 컨테이너 엔진을 독점적으로 실행합니다.

일부 노드는 REST API, etcd 데이터베이스 및 플랫폼 컨트롤러를 실행하는 컨트롤플레이인 노드입니다. OpenShift는 최종 사용자 애플리케이션 포드를 실행하도록 예약할 수 없고 컨트롤플레이인 서비스만 실행하도록 컨트롤플레이인 노드를 구성합니다. OpenShift는 최종 사용자 애플리케이션 포드를 작업자 노드에서 실행하도록 예약합니다.

다음 그래프에서는 OpenShift 컨트롤플레이인 노드를 개략적으로 설명하고, 일반 노드와 컨트롤플레이인 노드에서 시스템 서비스 또는 컨테이너로 실행되는 기본 프로세스를 설명합니다.

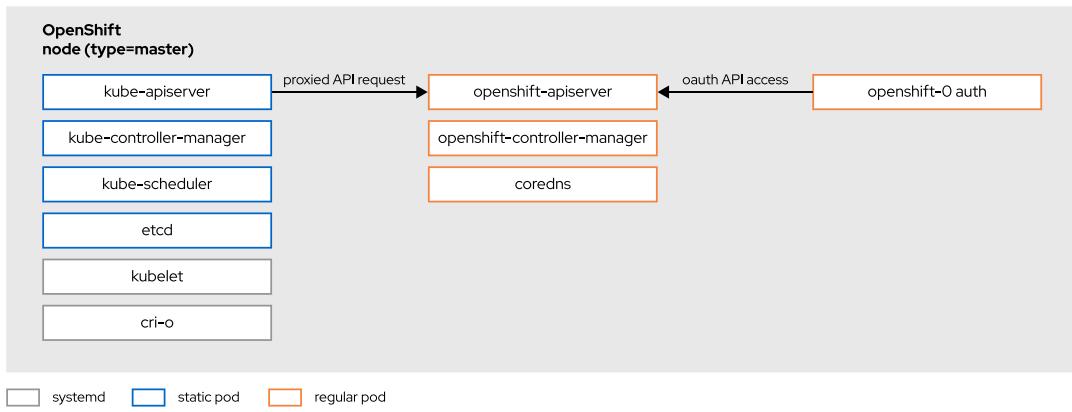


그림 8.3: OpenShift 컨트롤 플레인 노드의 아키텍처

노드 설정에 따라 **kubelet** 에이전트는 서로 다른 정적 포드 세트를 시작합니다. 정적 포드는 API 서버에 연결하지 않아도 시작할 수 있습니다. **Kubelet** 에이전트에서 포드의 라이프사이클을 관리합니다. 정적 포드에서는 컨트롤 플레인 서비스(예: 스케줄러) 또는 노드 서비스(예: 소프트웨어 정의 네트워킹(SDN))를 제공할 수 있습니다. OpenShift에서는 정적 포드를 일반 포드와 같이 모니터링하도록 정적 포드의 포드 리소스를 생성하는 운영자를 제공합니다.

## OpenShift 확장 설명

Kubernetes의 많은 기능은 인그레스(ingress) 컨트롤러, 스토리지 플러그인, 네트워크 플러그인 및 인증 플러그인과 같은 외부 구성 요소에 따라 달라집니다. Linux 배포판과 마찬가지로, 다양한 구성 요소를 선택하여 여러 가지 방법으로 Kubernetes 배포를 구축할 수 있습니다.

Kubernetes의 많은 기능은 액세스 제어 및 네트워크 격리와 같은 확장 API에 따라서도 달라집니다.

OpenShift는 이미 통합되고 구성되었으며 운영자가 관리하는 많은 구성 요소를 제공하는 Kubernetes 배포판입니다. OpenShift에서는 컨테이너 이미지 레지스트리와 웹 콘솔과 같이 운영자가 관리하는 사전 설치된 애플리케이션도 제공합니다.

OpenShift에서는 일련의 확장 API와 사용자 지정 리소스도 Kubernetes에 추가합니다. 예를 들어, Source-to-Image 프로세스의 구성을 빌드하고 클러스터에 대한 외부 액세스를 관리하도록 리소스의 경로를 지정합니다.

Red Hat에서는 모든 확장 기능을 오픈소스 프로젝트로 개발하고, 대규모 Kubernetes 커뮤니티와 함께 협력하여 Kubernetes의 공식 구성 요소를 만들 뿐만 아니라, 관리 용이성과 사용자 지정을 더 간편하게 수행할 수 있도록 Kubernetes 플랫폼을 개선합니다.

OpenShift 3에서 이러한 확장 기능은 업스트림 Kubernetes의 패치(또는 포크)인 경우가 있습니다. OpenShift 4 및 운영자에서 이러한 확장 기능은 Kubernetes 배포에 추가할 수 있는 표준 Kubernetes 확장 기능입니다.

## OpenShift Default Storage Class 소개

클라우드 네이티브의 상태 비저장 애플리케이션에 중점을 둔 많은 컨테이너 플랫폼과 달리, OpenShift에서는 표준 Twelve-Factor App 방법론을 따르지 않는 상태 저장 애플리케이션도 지원합니다.

OpenShift에서는 포괄적인 스토리지 및 지원 운영자 세트를 제공하여 상태 저장 애플리케이션을 지원합니다. OpenShift에서는 기본 클라우드 또는 가상화 플랫폼을 사용하여 동적으로 프로비저닝된 스토리지를 제공하는 통합 스토리지 플러그인 및 스토리지 클래스가 함께 제공됩니다.

예를 들어, AWS(Amazon Web Services)에 OpenShift를 설치하면 요구 시 스토리지 볼륨을 자동으로 프로비저닝하기 위해 OpenShift 클러스터에서 AWS EBS 서비스를 사용하는 기본 스토리지 클래스가 사전 구성됩니다. 사용자가 데이터베이스와 같은 영구 스토리지가 필요한 애플리케이션을 배포할 수 있으며, OpenShift에서는 애플리케이션 데이터를 호스팅하는 EBS 볼륨을 자동으로 생성합니다.

OpenShift 클러스터 관리자는 나중에 다른 EBS 서비스 계층을 사용하는 추가 스토리지 클래스를 정의할 수 있습니다. 예를 들어, 높은 초당 입출력 작업(IOPS) 속도를 유지하는 고성능 스토리지에 사용하는 스토리지 클래스 하나와 성능이 저조한 낮은 비용의 스토리지에 사용하는 스토리지 클래스가 하나 있을 수 있습니다. 그러면 클러스터 관리자가 특정 애플리케이션에서만 고성능 스토리지 클래스를 사용하도록 허용하고, 데이터 아카이브 애플리케이션에서는 낮은 성능 스토리지 클래스를 사용하도록 구성할 수 있습니다.



### 참조

자세한 정보는 Red Hat OpenShift Container Platform 4.5 제품 문서  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/)  
에 있습니다.

## ▶ 퀴즈

# OpenShift 아키텍처 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. OpenShift가 기반으로 하는 컨테이너 오케스트레이션 기술은 다음 중 무엇입니까?

- a. Docker Swarm
- b. Rancher
- c. Kubernetes
- d. Mesosphere Marathon
- e. CoreOS Fleet

▶ 2. OpenShift Container Platform에 관한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. OpenShift에서는 REST API에 대한 호출을 인증하는 OAuth 서버를 제공합니다.
- b. OpenShift에는 CRI-O 컨테이너 엔진이 필요합니다.
- c. Kubernetes는 선언형 아키텍처를 따르지만, OpenShift에서는 더 전통적인 명령형 아키텍처를 따릅니다.
- d. OpenShift 확장 API는 시스템 서비스로 실행됩니다.

▶ 3. 다음 중 Kubernetes API 구성 요소를 실행하는 서버는 무엇입니까?

- a. 작업자 노드
- b. 노드
- c. 컨트롤 플레인 노드

▶ 4. 다음 중 OpenShift에서 업스트림 Kubernetes에 추가하는 구성 요소는 무엇입니까?

- a. etcd 데이터베이스
- b. 컨테이너 엔진
- c. 레지스트리 서버
- d. 스케줄러
- e. Kubelet

▶ 5. 다음 중 OpenShift를 통한 스토리지 지원에 대해 올바른 설명은 무엇입니까?

- a. 사용자는 etcd 데이터베이스에만 영구 데이터를 저장할 수 있습니다.
- b. 사용자는 Twelve-Factor App 방법론을 준수하는 OpenShift 클라우드 네이티브 애플리케이션에만 배포할 수 있습니다.
- c. 관리자가 클라우드 프로바이더에 적절한 스토리지 플러그인을 구성해야 합니다.
- d. 관리자가 영구 볼륨을 정의해야 사용자가 영구 스토리지가 필요한 애플리케이션을 배포할 수 있습니다.
- e. 사용자는 기본 스토리지 클래스를 사용하여 영구 스토리지가 필요한 애플리케이션을 배포할 수 있습니다.

## ▶ 솔루션

# OpenShift 아키텍처 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. OpenShift가 기반으로 하는 컨테이너 오페스트레이션 기술은 다음 중 무엇입니까?

- a. Docker Swarm
- b. Rancher
- c. Kubernetes
- d. Mesosphere Marathon
- e. CoreOS Fleet

▶ 2. OpenShift Container Platform에 관한 다음 설명 중 올바른 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. OpenShift에서는 REST API에 대한 호출을 인증하는 OAuth 서버를 제공합니다.
- b. OpenShift에는 CRI-O 컨테이너 엔진이 필요합니다.
- c. Kubernetes는 선언형 아키텍처를 따르지만, OpenShift에서는 더 전통적인 명령형 아키텍처를 따릅니다.
- d. OpenShift 확장 API는 시스템 서비스로 실행됩니다.

▶ 3. 다음 중 Kubernetes API 구성 요소를 실행하는 서버는 무엇입니까?

- a. 작업자 노드
- b. 노드
- c. 컨트롤 플레인 노드

▶ 4. 다음 중 OpenShift에서 업스트림 Kubernetes에 추가하는 구성 요소는 무엇입니까?

- a. etcd 데이터베이스
- b. 컨테이너 엔진
- c. 레지스트리 서버
- d. 스케줄러
- e. Kubelet

▶ 5. 다음 중 OpenShift를 통한 스토리지 지원에 대해 올바른 설명은 무엇입니까?

- a. 사용자는 etcd 데이터베이스에만 영구 데이터를 저장할 수 있습니다.
- b. 사용자는 Twelve-Factor App 방법론을 준수하는 OpenShift 클라우드 네이티브 애플리케이션에만 배포할 수 있습니다.
- c. 관리자가 클라우드 프로바이더에 적절한 스토리지 플러그인을 구성해야 합니다.
- d. 관리자가 영구 볼륨을 정의해야 사용자가 영구 스토리지가 필요한 애플리케이션을 배포할 수 있습니다.
- e. 사용자는 기본 스토리지 클래스를 사용하여 영구 스토리지가 필요한 애플리케이션을 배포할 수 있습니다.

# 클러스터 운영자 설명

---

## 목표

이 섹션을 완료하고 나면 클러스터 운영자의 정의, 작동 방식 및 주요 클러스터 운영자의 이름을 설명할 수 있습니다.

## Kubernetes 운영자 소개

Kubernetes 운영자는 Kubernetes API를 호출하여 Kubernetes 리소스를 관리하는 애플리케이션입니다. Kubernetes 애플리케이션의 경우 운영자의 컨테이너 이미지를 참조하는 서비스 및 배포와 같은 Kubernetes 리소스를 정의하여 운영자를 배포합니다. 공통 애플리케이션과 달리 운영자에는 Kubernetes 리소스에 대한 직접 액세스 권한이 필요하므로 일반적으로 사용자 지정 보안 설정이 필요합니다.

일반적으로 운영자는 설정과 구성을 저장하는 사용자 지정 리소스(CR)를 정의합니다. OpenShift 관리자가 사용자 지정 리소스를 편집하여 운영자를 관리합니다. 사용자 지정 리소스의 구문은 사용자 지정 리소스 정의(CRD)로 정의합니다.

대부분의 운영자가 다른 애플리케이션을 관리합니다. 예를 들어, 데이터베이스 서버를 관리하는 운영자가 있습니다. 이 경우, 운영자가 사용자 지정 리소스의 정보를 사용하여 다른 애플리케이션을 설명하는 리소스를 생성합니다.

일반적으로 운영자는 사람인 관리자(또는 사람인 운영자)가 애플리케이션을 배포, 업데이트 및 관리하기 위해 수행할 작업을 자동화하기 위해 사용합니다.

## Operator Framework 소개

선호하는 프로그래밍 언어를 사용하여 운영자를 개발할 수 있습니다. 엄밀히 따지자면, 운영자를 개발하는 데 특수 용도의 SDK가 필요하지 않습니다. REST API를 호출하고 Kubernetes API에 대한 액세스 자격 증명을 포함하는 시크릿을 사용하는 기능만 있으면 됩니다.

Operator Framework는 운영자를 빌드, 테스트 및 패키징하는 데 사용하는 오픈소스 툴킷입니다. Operator Framework에서는 다음 구성 요소를 제공하므로 하위 수준 Kubernetes API를 직접 코딩하는 것보다 쉽게 이 작업을 수행할 수 있습니다.

### Operator SDK(Operator Software Development Kit)

운영자 애플리케이션에서 공통 패턴을 구현하는 Golang 라이브러리 및 소스 코드 예제를 제공합니다. Ansible을 사용하여 운영자를 개발할 수 있는 컨테이너 이미지 및 플레이북 예제도 제공합니다.

### OLM(운영자 라이프사이클 관리자)

운영자 카탈로그를 통해 배포된 운영자의 배포, 리소스 사용률, 업데이트 및 삭제를 관리하는 애플리케이션을 제공합니다. OLM 자체는 OpenShift에 사전 설치되어 제공되는 운영자입니다.

Operator Framework에서는 운영자와 CRD를 구현하는 권장 방법 세트와 운영자 카탈로그를 사용하여 운영자를 배포할 수 있는 운영자 매니페스트를 컨테이너 패키지로 패키징하는 표준 방법도 정의합니다. 가장 일반적인 운영자 카탈로그 양식은 이미지 레지스트리 서버입니다.

Operator Framework 표준을 따르는 운영자 컨테이너 이미지에는 운영자 애플리케이션을 배포하는데 필요한 모든 리소스 정의가 포함되어 있습니다. 이 방법을 통해 OLM에서 자동으로 운영자를 설치할 수 있습니다. Operator Framework 표준에 따라 운영자를 빌드하고 패키징하지 않으면 OLM에서 운영자를 설치하거나 관리할 수 없습니다.

## OperatorHub 소개

OperatorHub에서는 Operator Framework 표준을 따르는 운영자를 검색하고 게시하는 웹 인터페이스를 제공합니다. 오픈소스 운영자와 상용 운영자 모두 운영자 허브에 게시할 수 있습니다. 운영자 컨테이너 이미지는 다른 이미지 레지스트리(예: [quay.io](#))에서 호스팅할 수 있습니다.

## Red Hat Marketplace 소개

Red Hat Marketplace는 OpenShift 클러스터 또는 Kubernetes 클러스터에 배포할 수 있는 선별된 엔터프라이즈급 운영자 집합에 액세스할 수 있는 플랫폼입니다. Red Hat Marketplace에 제공되는 운영자는 소프트웨어가 모범 사례를 준수하며 컨테이너 취약점이 검사되었음을 확인하기 위한 인증 프로세스를 거친 것입니다.

Red Hat Marketplace Operator를 사용하면 OpenShift 클러스터와 Red Hat Marketplace를 원활하게 통합할 수 있습니다. 이러한 통합을 이용하면 업데이트를 관리하고 청구 및 보고 작업을 통합하여 인증된 운영자를 간편하게 배포할 수 있습니다. 벤더는 무료 평가판, 다양한 버전, 대규모 고객을 위한 할인 등 운영자를 위해 다양한 가격 옵션을 제공합니다.

## OpenShift 클러스터 운영자 소개

클러스터 운영자는 일반 운영자입니다. 단, 이 운영자는 OLM에서 관리하지 않습니다. 이 운영자는 첫 번째 수준 운영자라고도 하는 OpenShift 클러스터 버전 운영자에서 관리합니다. 모든 클러스터 운영자는 두 번째 수준 운영자라고도 합니다.

OpenShift 클러스터 운영자는 다음과 같은 OpenShift 확장 API 및 인프라 서비스를 제공합니다.

- 컨트롤 플레인 및 확장 API에 대한 액세스를 인증하는 OAuth 서버입니다.
- 클러스터 내에서 서비스 검색을 관리하는 코어 DNS 서버입니다.
- 클러스터의 그래픽 관리를 가능하게 하는 웹 콘솔입니다.
- S2I 또는 다른 메커니즘을 사용하여 개발자가 클러스터 내부에서 컨테이너 이미지를 호스팅할 수 있는 내부 이미지 레지스트리입니다.
- 클러스터 상태에 관한 지표 및 경고를 생성하는 모니터링 스택입니다.

일부 클러스터 운영자는 노드나 컨트롤 플레인 설정을 관리합니다. 예를 들어, 업스트림 Kubernetes에서는 노드 구성 파일을 편집하여 스토리지 및 네트워크 플러그인을 추가합니다. 이러한 플러그인에는 추가 구성 파일이 필요할 수 있습니다. OpenShift에서는 모든 노드에서 구성 파일을 관리하고 해당 파일의 변경사항에 영향을 받는 노드 서비스를 다시 로드하는 운영자를 지원합니다.



### 중요

OpenShift 4에서는 더 이상 SSH 세션을 사용하여 노드 구성과 시스템 서비스를 관리하지 않습니다. 따라서 노드를 사용자 지정하지 않으며, 클러스터에서 안전하게 노드를 추가하거나 제거할 수 있습니다. 사용자 지정 리소스를 편집하여 간접적으로 모든 관리 작업을 수행한 다음, 각 운영자가 변경사항을 적용할 때까지 기다립니다.

## OpenShift 클러스터 운영자 살펴보기

일반적으로 운영자와 관리되는 애플리케이션에서는 동일한 프로젝트를 공유합니다. 클러스터 운영자의 경우 `openshift-*` 프로젝트에 있습니다.

모든 클러스터 운영자가 **ClusterOperator** 유형의 사용자 지정 리소스를 정의합니다. 클러스터 운영자는 API 서버, 웹 콘솔 또는 네트워크 스택을 비롯하여 클러스터 자체를 관리합니다. 각 클러스터 운영자가 구성

요소를 추가로 제어하기 위해 사용자 지정 리소스 세트를 정의합니다. **ClusterOperator** API 리소스에서 업데이트 상태 또는 구성 요소의 버전과 같은 정보를 표시합니다.

운영자의 기능은 이름에서 명확히 알 수 있습니다. 예를 들어 콘솔 클러스터 운영자는 웹 콘솔을 제공하고, 인그레스 클러스터 운영자를 사용해서는 인그레스와 라우팅이 가능합니다. 다음은 몇 가지 클러스터 운영자를 나열합니다.

- `network`
- `ingress`
- `storage`
- `authentication`
- `console`
- `monitoring`
- `image-registry`
- `cluster-autoscaler`
- `openshift-apiserver`
- `dns`
- `openshift-controller-manager`
- `cloud-credential`



### 참조

자세한 정보는 Red Hat OpenShift Container Platform 4.5 제품 문서  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/)  
에 있습니다.

### Operator Framework 소개

<https://blog.openshift.com/introducing-the-operator-framework/>

### Red Hat Marketplace 시작하기

<https://marketplace.redhat.com/en-us/documentation/getting-started>

## ▶ 퀴즈

### 클러스터 운영자 설명

아래 용어와 일치하는 테이블의 올바른 정의를 찾습니다.

OLM(운영자 라이프사이클 관리자)

Operator

OperatorHub

Red Hat Marketplace

사용자 지정 리소스 정의

운영자 SDK

운영자 이미지

운영자 카탈로그

운영자 용어	이름
운영자를 빌드, 테스트 및 패키징하는 데 사용하는 오픈소스 툴킷입니다.	
운영자를 검색하고 설치하는 데 사용하는 리포지토리입니다.	
사용자 지정 리소스의 구문을 정의하는 Kubernetes API의 확장입니다.	
OLM 인스턴스에서 사용하기 위해 게시할 수 있는 Operator Framework에서 정의한 아티팩트입니다.	
Kubernetes 리소스를 관리하는 애플리케이션입니다.	
Kubernetes 운영자를 관리하는 애플리케이션입니다.	
OLM과 호환되는 운영자를 게시할 수 있는 공용 웹 서비스입니다.	
OpenShift 클러스터에 배포할 수 있도록 Kubernetes 운영자로 패키징된 인증 소프트웨어에 액세스할 수 있는 플랫폼입니다.	

## ▶ 솔루션

### 클러스터 운영자 설명

아래 용어와 일치하는 테이블의 올바른 정의를 찾습니다.

운영자 용어	이름
운영자를 빌드, 테스트 및 패키징하는 데 사용하는 오픈소스 툴킷입니다.	운영자 SDK
운영자를 검색하고 설치하는 데 사용하는 리포지토리입니다.	운영자 카탈로그
사용자 지정 리소스의 구문을 정의하는 Kubernetes API의 확장입니다.	사용자 지정 리소스 정의
OLM 인스턴스에서 사용하기 위해 게시할 수 있는 Operator Framework에서 정의한 아티팩트입니다.	운영자 이미지
Kubernetes 리소스를 관리하는 애플리케이션입니다.	Operator
Kubernetes 운영자를 관리하는 애플리케이션입니다.	OLM(운영자 라이프사이클 관리자)
OLM과 호환되는 운영자를 게시할 수 있는 공용 웹 서비스입니다.	OperatorHub
OpenShift 클러스터에 배포할 수 있도록 Kubernetes 운영자로 패키징된 인증 소프트웨어에 액세스할 수 있는 플랫폼입니다.	Red Hat Marketplace

## 요약

---

이 장에서 학습한 내용:

- Red Hat OpenShift Container Platform은 Red Hat Enterprise Linux CoreOS, CRI-O 컨테이너 및 Kubernetes를 기반으로 합니다.
- RHOC 4에서는 Kubernetes 외에도 내부 컨테이너 이미지 레지스트리, 스토리지, 네트워킹 프로바이더 및 중앙 집중식 로깅과 모니터링 등의 다양한 서비스를 제공합니다.
- 운영자는 Kubernetes 리소스를 관리하는 애플리케이션을 패키징하고 OLM(운영자 라이프사이클 관리자)은 운영자의 설치 및 관리를 처리합니다.
- OperatorHub.io는 운영자를 검색하는 데 사용하는 온라인 카탈로그입니다.



## 9장

# 클러스터 확인

### 목적

OpenShift 설치 방법을 설명하고 새로 설치된 클러스터의 상태를 확인합니다.

### 목표

- OpenShift 설치 프로세스, 풀스택 자동화, 기존 인프라 설치 방법을 설명합니다.
- 일반적인 문제 해결을 지원하는 명령을 실행합니다.
- 영구저장장치의 구성 요소 및 리소스를 확인하고, 영구 볼륨 클레임을 사용하는 애플리케이션을 배포합니다.

### 섹션

- 설치 방법 설명 및 퀴즈
- OpenShift 클러스터 및 애플리케이션의 문제 해결(안내에 따른 연습)
- OpenShift 동적 스토리지 소개(안내에 따른 연습)

# 설치 방법 설명

---

## 목표

이 섹션을 마치면 OpenShift 설치 프로세스, 풀스택 자동화, 기존 인프라 설치 방법을 설명할 수 있습니다.

## OpenShift 설치 방법 소개

Red Hat OpenShift Container Platform에서는 두 가지 기본 설치 방법을 제공합니다.

### 풀스택 자동화

OpenShift 설치 프로그램에서는 이 방법을 사용하여 클라우드 또는 가상화 프로바이더로부터 모든 계산, 스토리지 및 네트워크 리소스를 프로비저닝합니다. 설치 프로그램에 클라우드 프로바이더의 자격 증명 및 초기 클러스터의 크기와 같은 최소한의 데이터를 제공하면, 설치 프로그램에서 완전한 기능을 갖춘 OpenShift 클러스터를 배포합니다.

### 기존 인프라

이 방법에서는 사용자가 계산, 스토리지 및 네트워크 리소스 세트를 구성하고 OpenShift 설치 프로그램에서 이 리소스를 사용하여 초기 클러스터를 구성합니다. 이 방법을 사용하면 풀스택 자동화 방법에서 지원하지 않는 베어 메탈 서버 및 클라우드 또는 가상화 프로바이더를 사용하여 OpenShift 클러스터를 설정할 수 있습니다.

기존 인프라를 사용하는 경우 부트스트랩 노드를 포함하여 모든 클러스터 인프라 및 리소스를 제공해야 합니다. 설치 프로그램을 실행하여 필수 구성 파일을 생성한 다음, 설치 프로그램을 다시 실행하여 인프라에 OpenShift 클러스터를 배포해야 합니다.

Red Hat OpenShift Container Platform 4.5가 릴리스된 시점에, 풀스택 자동화 방법을 사용할 수 있는 클라우드 프로바이더로는 표준 Intel 아키텍처(x86)를 사용하는 AWS(Amazon Web Services), GCP(Google Cloud Platform), Microsoft Azure, Red Hat OpenStack Platform 등이 있습니다. 풀스택 자동화가 가능한 가상화 프로바이더 및 아키텍처에는 VMware, Red Hat Virtualization, IBM Power, IBM System Z 등이 있습니다.

4.x 스트림의 모든 부 릴리스에서는 더 많은 기능을 추가하고, 사전에 생성된 클라우드 리소스를 재사용하는 등 사용자 지정 기능을 추가로 지원합니다.

## OpenShift 설치 방법 비교

OpenShift의 특정 기능을 사용하려면 풀스택 자동화 방법(예: 클러스터 자동 확장)을 사용해야 합니다. 그러나 향후 릴리스에서는 이러한 요구 사항이 완화될 예정입니다.

풀스택 자동화 방법을 사용하면 새 클러스터의 모든 노드에서 RHEL CoreOS(Red Hat Enterprise Linux CoreOS)가 실행됩니다. 기존 인프라 방법에서는 RHEL(Red Hat Enterprise Linux)를 사용하여 작업자 노드를 설정할 수 있지만, 컨트롤 플레인(마스터 노드)에는 여전히 RHEL CoreOS가 필요합니다.

## 배포 프로세스 설명

설치 프로그램에서 생성하는 자산을 사용하여 Red Hat Enterprise Linux CoreOS를 실행하는 부트스트랩 시스템 생성부터 시작하여 여러 단계로 설치가 진행됩니다.

클러스터의 부트스트랩 프로세스는 다음과 같습니다.

## 9장 | 클러스터 확인

1. 부트스트랩 시스템이 부팅되고 컨트롤 플레인 시스템을 부팅하는 데 필요한 원격 리소스 호스팅을 시작 합니다.
2. 컨트롤 플레인 시스템이 부트스트랩 시스템에서 원격 리소스를 가져오고 부팅을 완료합니다.
3. 컨트롤 플레인 시스템이 Etcd 클러스터를 형성합니다.
4. 부트스트랩 시스템에서 새로 생성된 Etcd 클러스터를 사용하여 임시 Kubernetes 컨트롤 플레인을 시작 합니다.
5. 임시 컨트롤 플레인에서 컨트롤 플레인 시스템에 컨트롤 플레인을 예약합니다.
6. 임시 컨트롤 플레인의 종료되고 컨트롤 플레인의 됩니다.
7. 부트스트랩 노드에서 OpenShift 고유 구성 요소를 컨트롤 플레인에 삽입합니다.
8. 마지막으로 설치 프로그램에서 부트스트랩 시스템을 종료합니다.

이 부트스트랩 프로세스의 결과로 API 서버, 컨트롤러(예: SDN) 및 Etcd 클러스터를 포함하는 OpenShift 컨트롤 플레인이 모두 실행됩니다. 그런 다음 클러스터가 지원되는 플랫폼에서 계산 시스템을 자동으로 생성하는 기능을 포함하여 클러스터 버전 운영자를 통해 일상적인 작업에 필요한 나머지 구성 요소를 다운로드하고 구성합니다.

## OpenShift 설치 사용자 지정

OpenShift 설치 프로그램을 사용하면 프로비저닝하는 초기 클러스터를 아주 조금만 사용자 지정할 수 있습니다. 다음과 같은 대부분의 사용자 지정은 설치 후에 수행됩니다.

- 동적 스토리지 프로비저닝을 위한 사용자 지정 스토리지 클래스 정의
- 클러스터 운영자의 사용자 지정 리소스 변경
- 새 운영자를 클러스터에 추가합니다.
- 새 시스템 세트를 정의합니다.



### 참조

다양한 설치 방법에 관한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/installing/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/installing/index)

에 있는 Red Hat OpenShift Container Platform 4.5 설치 설명서를 참조하십시오.

설치 프로그램 프로비저닝 인프라에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 OpenShift 4.x 설치 - 빠른 개요(IPI 설치) 동영상을 참조하십시오.

<https://www.youtube.com/watch?v=uBsilb4cual>

사용자 프로비저닝 인프라에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 VMware vSphere를 사용한 OpenShift 4 사용자 프로비저닝 인프라 동영상을 참조하십시오.

<https://www.youtube.com/watch?v=TsAJEEDv-gg>

## ▶ 퀴즈

### 설치 방법 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 OpenShift 설치 프로그램을 사용하여 컨트롤 플레인 노드와 작업자 노드를 구성해야 하는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 2. 다음 중 Red Hat Enterprise Linux를 사용하여 노드를 설정할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 3. 다음 중 일부 OpenShift 기능을 사용하여 지원되지 않는 가상화 프로바이더를 사용할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 4. 다음 중 최소한의 작업만으로 지원되는 몇 가지 클라우드 프로바이더를 사용할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 5. 다음 중 OpenShift 설치 프로그램에 입력을 제공하여 클러스터 설정을 광범위하게 사용자 지정할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

## ▶ 솔루션

### 설치 방법 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 OpenShift 설치 프로그램을 사용하여 컨트롤 플레인 노드와 작업자 노드를 구성해야 하는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 2. 다음 중 Red Hat Enterprise Linux를 사용하여 노드를 설정할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 3. 다음 중 일부 OpenShift 기능을 사용하여 지원되지 않는 가상화 프로바이더를 사용할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 4. 다음 중 최소한의 작업만으로 지원되는 몇 가지 클라우드 프로바이더를 사용할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

▶ 5. 다음 중 OpenShift 설치 프로그램에 입력을 제공하여 클러스터 설정을 광범위하게 사용자 지정할 수 있는 설치 방법은 무엇입니까?

- a. 풀스택 자동화
- b. 기존 인프라
- c. 풀스택 자동화 및 기존 인프라 둘 다
- d. 풀스택 자동화 및 기존 인프라 둘 다 해당 없음

# OpenShift 클러스터 및 애플리케이션의 문제 해결

---

## 목표

이 섹션을 마치면 다음 작업을 할 수 있습니다.

- 일반적인 문제 해결을 지원하는 명령을 실행합니다.
- OpenShift 클러스터 노드가 정상인지 확인합니다.
- 배포의 OpenShift 및 Kubernetes 스타일과 관련된 일반적인 문제를 해결합니다.

## OpenShift 클러스터의 일반적인 문제 해결

OpenShift 클러스터의 문제 해결은 대부분 Red Hat OpenShift 4의 구성 요소는 운영자이고 운영자는 Kubernetes 애플리케이션이므로 애플리케이션 배포 문제를 해결하는 것과 매우 유사합니다. 각 운영자에 대해 해당 운영자가 있는 프로젝트, 운영자 애플리케이션을 관리하는 배포 및 포드를 식별할 수 있습니다. 해당 운영자가 변경해야 하는 구성 설정이 있는 경우 사용자 지정 리소스(CR)를 식별하거나 이러한 설정을 저장하는 구성 맵 또는 시크릿 리소스를 식별할 수 있습니다.

대부분의 OpenShift 운영자는 데몬 집합 및 배포와 같은 표준 Kubernetes 워크로드 API 리소스에서도 배포되는 애플리케이션을 관리합니다. 운영자의 역할은 일반적으로 이러한 리소스를 만들고 CR과 동기화되도록 유지하는 것입니다.

이 섹션은 운영자 또는 애플리케이션 배포와 직접적으로 관련되지 않은 클러스터 문제에 초점을 맞추어 시작합니다. 이 섹션의 뒷부분에서 애플리케이션 배포 문제를 해결하는 방법을 알아봅니다.

## OpenShift 노드의 상태 확인

다음 명령은 OpenShift 클러스터에서 노드의 상태에 대한 정보를 표시합니다.

### `oc get nodes`

각 노드의 상태를 사용하여 열을 표시합니다. 노드가 **Ready(준비)** 상태가 아닌 경우에는 OpenShift 제어판과 통신할 수 없으며 클러스터에서 효과적으로 작동하지 않습니다.

### `oc adm top nodes`

각 노드의 현재 CPU 및 메모리 사용량을 표시합니다. 이것은 OpenShift 스케줄러가 노드의 사용 가능한 용량 및 사용된 용량으로 간주하는 리소스 요청이 아닌 실제 사용 번호입니다.

### `oc describe node my-node-name`

스케줄러 관점에서 사용 가능하고 사용한 리소스와 기타 정보를 표시합니다. 출력에 "용량", "할당 가능" 및 "할당된 리소스"라는 제목을 찾습니다. 제목 "조건"은 노드가 메모리 압력, 디스크 압력 또는 다른 조건을 충족하여 노드가 새 컨테이너를 시작하지 못하게 하는지를 나타냅니다.

## 클러스터 버전 리소스 검토

OpenShift 설치 프로그램에서는 `kubeconfig` 및 `kubeadm-password` 파일을 포함하는 `auth` 디렉터리를 생성합니다. `oc login` 명령을 실행하여 `kubeadm` 사용자로 클러스터에 연결합니다. `kubeadm` 사용자의 암호는 `kubeadm-password` 파일에 있습니다.

```
[user@dem ~]$ oc login -u kubeadmin -p MMTUc-TnXjo-NFyh3-aeWmC
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

**ClusterVersion**은 업데이트 채널, 클러스터 운영자 상태, 클러스터 버전(예: 4.5.4) 등 대략적인 클러스터 정보가 포함된 사용자 지정 리소스입니다. 이 리소스를 사용하여 실행할 클러스터의 버전을 선언합니다. 클러스터의 새 버전을 정의하면 **cluster-version** 운영자가 클러스터를 해당 버전으로 업그레이드하도록 지시합니다.

클러스터 버전을 검색하여 원하는 버전을 실행 중인지 확인하고, 클러스터에서 올바른 서브스크립션 채널을 사용하는지 확인할 수 있습니다.

- **oc get clusterversion**을 실행하여 클러스터 버전을 검색합니다. 출력에서 부 릴리스를 포함하는 버전, 지정된 버전의 클러스터 가동 시간 및 클러스터의 전체 상태를 나열합니다.

```
[user@demo ~]$ oc get clusterversion
NAME VERSION AVAILABLE PROGRESSING SINCE STATUS
version 4.5.4 True False 4d23h Cluster version is 4.5.4
```

- **oc describe clusterversion**을 실행하여 클러스터 상태에 관한 자세한 정보를 얻으십시오.

```
[user@demo ~]$ oc describe clusterversion
Name: version
Namespace:
Labels: <none>
Annotations: <none>
API Version: config.openshift.io/v1
Kind: ClusterVersion
...output omitted...
Spec:
 Channel: stable-4.5 ①
 Cluster ID: f33267f8-260b-40c1-9cf3-ecc406ce035e ②
 Upstream: https://api.openshift.com/api/upgrades_info/v1/graph ③
Status:
 Available Updates: ④
 Force: false
 Image: quay.io/openshift-release-dev/ocp-release@sha256:...
 Version: 4.5.5
 Conditions:
 Last Transition Time: 2020-08-05T18:35:08Z
 Message: Done applying 4.5.4 ⑤
 Status: True
 Type: Available
 ...output omitted...
 Desired:
 Force: false
 Image: quay.io/openshift-release-dev/ocp-release@sha256:...
 Version: 4.5.4
 ...output omitted...
 History:
 Completion Time: 2020-08-05T18:35:08Z ⑥
 Image: quay.io/openshift-release-dev/ocp-release@sha256:...
```

```

Started Time: 2020-08-05T18:22:42Z
State: Completed ⑦
Verified: false
Version: 4.5.4
Observed Generation: 2
...output omitted...

```

- ❶ 클러스터의 버전과 해당 채널을 표시합니다. 서브스크립션에 따라 채널이 다를 수 있습니다.
- ❷ 클러스터의 고유 식별자를 표시합니다. Red Hat에서는 이 식별자를 사용하여 클러스터 및 클러스터 자격을 확인합니다.
- ❸ 이 URL은 Red Hat 업데이트 서버에 해당합니다. 엔드포인트를 사용하면 클러스터에서 새 버전으로 업데이트할 때 업그레이드 경로를 확인할 수 있습니다.
- ❹ 이 항목에는 클러스터 업데이트에 사용할 수 있는 이미지가 나열됩니다.
- ❺ 이 항목에는 기록이 나열됩니다. 출력은 업데이트가 완료되었음을 나타냅니다.
- ❻ 이 항목은 클러스터에서 **버전** 항목에 표시된 버전을 배포한 시기를 나타냅니다.
- ❼ 이 항목은 버전이 성공적으로 배포되었음을 나타냅니다. 이 항목을 사용하여 클러스터 상태를 확인합니다.

## 클러스터 운영자 검토

OpenShift Container Platform 클러스터 운영자는 클러스터를 관리하는 최상위 수준 운영자입니다. API 서버, 웹 콘솔, 스토리지 또는 SDN과 같은 주요 구성 요소를 담당합니다. 해당 정보는 **ClusterOperator** 리소스를 통해 액세스할 수 있으므로, 모든 클러스터 운영자의 개요 또는 지정된 운영자의 자세한 정보에 액세스할 수 있습니다.

- `oc get clusteroperators`를 실행하여 모든 클러스터 운영자 목록을 검색합니다.

```

[user@demo ~]$ oc get clusteroperators
NAME VERSION AVAILABLE PROGRESSING DEGRADED SINCE
authentication ① 4.5.4 True False False 3h58m
cloud-credential 4.5.4 True False False 4d23h
cluster-autoscaler 4.5.4 True False False 4d23h
config-operator 4.5.4 True False False 4d23h
console 4.5.4 True False False 3h58m
csi-snapshot-controller 4.5.4 True False False 4d23h
dns 4.5.4 True False False 4d23h
etcd 4.5.4 True False False 4d23h
image-registry 4.5.4 True False False 4d23h
...output omitted...

```

- ❶ 운영자의 이름을 나타내며, 이 경우 운영자는 인증 관리를 담당합니다.
- ❷ 이 항목은 운영자가 성공적으로 배포되었으며 클러스터에서 사용할 수 있음을 나타냅니다. 클러스터 운영자는 성능이 저하된 경우에도 사용 가능한 상태를 반환할 수 있습니다. 현재 자원 상태가 일정 기간 동안 원하는 상태와 일치하지 않을 경우 운영자가 성능 저하를 보고합니다. 예를 들어 운영자에게 3개의 실행 중인 포드가 필요하지만 포드 하나가 충돌하는 경우 운영자를 사용할 수 있지만 성능이 저하된 상태입니다.
- ❸ 이 항목은 상위 수준 운영자에서 운영자를 최신 버전으로 업데이트하고 있는지 여부를 나타냅니다. **클러스터 버전** 운영자가 새 리소스를 배포하는 경우 열에서 **True**를 표시합니다.
- ❹ 항목은 운영자의 상태를 반환합니다. 이 항목은 운영자가 제대로 작동하지 못하게 만드는 오류가 발생하는 경우 **True**를 표시합니다. 운영자 서비스를 계속 사용할 수 있지만 모든 요구 사항이 충족되지 않을 수 있습니다. 이는 운영자가 실패하고 사용자 개입이 필요하다는 것을 나타낼 수 있습니다.

## OpenShift 노드의 로그 표시

OpenShift의 대부분의 인프라 구성 요소는 포드 내에 있는 컨테이너입니다. 최종 사용자 애플리케이션의 로그를 보는 것과 동일한 방법으로 로그를 볼 수 있습니다. 이러한 컨테이너 중 일부는 Kubelet에서 생성되므로 대부분의 Kubernetes 배포판에서 보이지 않지만 OpenShift 클러스터 운영자는 이러한 컨테이너에 대한 포드 리소스를 만듭니다.

Red Hat Enterprise Linux CoreOS를 기반으로 하는 OpenShift 노드에서는 상태를 검사하기 위해 노드에 직접 액세스해야 하는 매우 적은 수의 로컬 서비스를 실행합니다. Red Hat Enterprise Linux CoreOS의 대부분의 시스템 서비스는 컨테이너로 실행됩니다. 기본 예외는 CRI-O 컨테이너 엔진과 Kubelet(Systemd 단위)입니다. 이러한 로그를 보려면 다음 예와 같이 **oc adm node-logs** 명령을 사용합니다.

```
[user@demo ~]$ oc adm node-logs -u crio my-node-name
```

```
[user@demo ~]$ oc adm node-logs -u kubelet my-node-name
```

또한 노드의 저널 로그를 모두 표시할 수 있습니다.

```
[user@demo ~]$ oc adm node-logs my-node-name
```

## OpenShift 노드에서 쉘 프롬프트 열기

Red Hat Openshift Cluster Platform 3 및 기타 Kubernetes의 배포를 관리하는 관리자는 해당 노드에 자주 SSH 세션을 열어 제어판과 컨테이너 엔진의 상태를 검사하거나 구성 파일을 변경합니다. 이 작업은 계속 수행할 수 있지만 Red Hat Openshift Cluster Platform 4에서는 더 이상 권장되지 않습니다.

전체 스택 자동화 방법을 사용하여 클러스터를 설치하는 경우 클러스터 노드는 가상 사설 네트워크에 있기 때문에 인터넷에서 직접 액세스할 수 없으므로 AWS는 가상 프라이빗 클라우드(VPC)를 호출합니다. SSH 세션을 열려면 공용 IP 주소도 할당하는 동일한 VPC 클러스터의 방호 서버가 필요합니다. 방호 서버를 만들려면 클라우드 프로바이더에 따라 다르며 이 과정의 범위를 벗어났습니다.

**oc debug node** 명령은 클러스터의 모든 노드에서 쉘 프롬프트를 여는 방법을 제공합니다. 이 프롬프트는 **/host** 폴더에 노드 루트 파일 시스템을 마운트하는 특수 용도의 도구 컨테이너에서 제공되며 노드에서 파일을 검사할 수 있습니다.

**oc debug node** 세션의 노드에서 직접 로컬 명령을 실행하려면 **/host** 폴더에서 chroot 쉘을 시작해야 합니다. 그런 다음 노드의 로컬 파일 시스템, systemd 서비스의 상태를 검사하고 그렇지 않은 경우 SSH 세션을 필요로 하는 기타 작업을 수행할 수 있습니다. 다음은 **oc debug node** 세션의 예입니다.

```
[user@demo ~]$ oc debug node/my-node-name
...output omitted...
sh-4.2# chroot /host
sh-4.2# systemctl is-active kubelet
active
```

**oc debug node** 명령에서 시작한 쉘 세션은 OpenShift 제어판에 따라 작동합니다. 실행 중인 포드 내에서 쉘 프롬프트를 열 수 있는 동일한 터널링 기술을 사용합니다(이 섹션 뒷부분의 **oc rsh** 명령 참조). **oc debug node** 명령은 SSH 또는 RSH 프로토콜을 기반으로 하지 않습니다.

제어판이 작동하지 않거나 노드가 준비되지 않았거나 노드가 제어판과 통신할 수 없는 이유 때문에 **oc debug node** 명령을 사용할 수 없으며 방호 호스트를 필요로하게 됩니다.

**경고**

**oc debug node** 명령을 사용할 때 주의하십시오. 일부 작업은 Kubelet를 중지하는 등 노드를 사용할 수 없도록 렌더링할 수 있으며 **oc** 명령만 사용하여 복구할 수 없습니다.

## 컨테이너 엔진 문제 해결

**oc debug node** 세션에서 **crictl** 명령을 사용하여 노드에서 실행되는 모든 로컬 컨테이너에 대한 하위 수준 정보를 가져옵니다. CRI-O로 만든 컨테이너에 대한 표시가 없으므로 이 작업에 대해 **podman** 명령을 사용할 수 없습니다. 다음 예제에서는 노드에서 실행되는 모든 컨테이너를 나열합니다. **oc describe node** 명령은 동일한 정보를 제공하지만 컨테이너 대신 포드를 통해 구성됩니다.

```
[user@demo ~]$ oc debug node/my-node-name
...output omitted...
sh-4.2# chroot /host
sh-4.2# crictl ps
...output omitted...
```

## 애플리케이션 배포 문제 해결

애플리케이션 문제를 해결할 때 일반적으로 Kubernetes 배포와 OpenShift 배포 구성의 차이점을 무시할 수 있습니다. 일반적인 오류 시나리오와 문제 해결 방법은 본질적으로 동일합니다.

이 과정의 후반부 장에서는 예약할 수 없는 포드 등 다양한 시나리오가 설명되어 있습니다. 이 섹션에서는 일반 애플리케이션에 적용되는 일반적인 시나리오를 중점적으로 다루고 있으며 일반적으로 동일한 시나리오가 운영자에게 적용됩니다.

## 시작에 실패한 포드 문제 해결

일반적인 시나리오는 OpenShift가 포드를 만들고 해당 포드에서 **Running**(실행 중) 상태를 설정하지 않는 것입니다. 즉, OpenShift가 해당 포드 내부의 컨테이너를 시작할 수 없습니다. **oc get pod** 및 **oc status** 명령을 사용하여 문제 해결을 시작하여 포드 및 컨테이너가 실행 중인지 확인합니다. 포드가 **ErrImagePull** 또는 **ImagePullBackoff**와 같은 오류 상태에 있습니다.

이 경우 첫 번째 단계는 **oc get events** 명령을 사용하여 현재 프로젝트의 이벤트를 나열하는 것입니다. 프로젝트에 많은 포드가 포함된 경우에는 **oc describe pod** 명령을 사용하여 포드별로 필터링된 이벤트 목록을 가져올 수 있습니다. 또한 유사한 **oc describe** 명령을 실행하여 배포 및 배포 구성으로 이벤트를 필터링할 수도 있습니다.

## 실행 중인 포드와 종료된 포드 문제 해결

또 다른 일반적인 시나리오는 OpenShift가 포드를 만들고 짧은 시간 동안 문제가 발생하지 않는 것입니다. 포드는 **Running**(실행 중) 상태가 되며 이는 해당 컨테이너 중 하나 이상이 실행을 시작했음을 의미합니다. 나중에 포드 컨테이너 중 하나 내에서 실행되는 애플리케이션의 작동이 중지됩니다. 오류 메시지를 종료하거나 사용자 요청으로 반환할 수 있습니다.

적절하게 설계된 배포를 통해 애플리케이션을 관리하는 경우 결국에는 애플리케이션을 종료하고 해당 컨테이너를 중지하는 상태 프로브를 포함해야 합니다. 이런 경우 OpenShift는 컨테이너를 여러 번 다시 시작하려고 시도합니다. 상태 프로브 또는 기타 이유로 인해 애플리케이션이 종료를 계속 수행하는 경우 포드는 **CrashLoopBackOff** 상태로 유지됩니다.

## 9장 | 클러스터 확인

실행 중인 컨테이너는 짧은 시간 동안에도 로그를 생성합니다. 이러한 로그는 컨테이너가 종료될 때 삭제되지 않습니다. **oc logs** 명령은 포드 내에 있는 모든 컨테이너의 로그를 표시합니다. 포드에 단일 컨테이너가 포함된 경우 **oc logs** 명령에는 포드 이름만 필요합니다.

```
[user@demo ~]$ oc logs my-pod-name
```

포드에 여러 컨테이너가 포함된 경우에는 **oc logs** 명령에 **-c** 옵션이 필요합니다.

```
[user@demo ~]$ oc logs my-pod-name -c my-container-name
```

애플리케이션 로그를 해석하려면 특정 애플리케이션을 구체적으로 알고 있어야 합니다. 모든 것이 제대로 되면 애플리케이션에서 문제를 찾는데 도움이 될 수 있는 명확한 오류 메시지를 제공합니다.

## OpenShift 집계 로깅 소개

Red Hat OpenShift Container Platform 4는 클러스터 및 해당 컨테이너에서 로그를 집계하는 Elasticsearch, Fluentd 또는 Rsyslog 및 Kibana를 기반으로 클러스터 로깅 하위 시스템을 제공합니다.

해당 운영자를 통해 OpenShift 클러스터 로깅 하위 시스템을 배포하고 구성하는 것은 이 과정의 범위를 벗어납니다. 자세한 내용은 이 섹션 끝에 있는 참조 섹션을 참조하십시오.

## 포드 문제 해결 생성

문제가 애플리케이션 컨테이너 이미지 또는 OpenShift 리소스에서 가져온 설정에 관련되는지 확실하지 않은 경우 **oc debug** 명령은 매우 유용합니다. 이 명령은 기존 포드, 배포 구성, 배포 또는 워크로드 API의 기타 리소스를 기반으로 포드를 만들습니다.

새 포드는 컨테이너 이미지의 기본 진입점 대신 대화형 쉘을 실행합니다. 또한 상태 프로브가 비활성화된 상태로 실행됩니다. 이 방법을 사용하면 환경 변수, 다른 서비스에 대한 네트워크 액세스 및 포드 내의 권한을 쉽게 확인할 수 있습니다.

**oc debug** 명령의 명령줄 옵션을 사용하여 복제하지 않을 설정을 지정할 수 있습니다. 예를 들어 컨테이너 이미지를 변경하거나 고정 사용자 ID를 지정할 수 있습니다. 일부 설정에는 클러스터 관리자 권한이 필요할 수 있습니다.

일반적인 시나리오는 배포에서 포드를 만들고 루트 사용자로 실행하여 배포가 OpenShift의 기본 보안 정책에 따라 실행되도록 설계되지 않은 컨테이너 이미지를 참조한다는 것을 증명하는 것입니다.

```
[user@demo ~]$ oc debug deployment/my-deployment-name --as-root
```

## 실행 중인 컨테이너 변경

컨테이너 이미지는 변경할 수 없고 컨테이너는 임시 상태로 있어야 하므로 실행 중인 컨테이너를 변경하지 않는 것이 좋습니다. 그러나 이러한 변경 사항을 적용하면 애플리케이션 문제 해결에 도움이 될 수 있습니다. 실행 중인 컨테이너를 변경한 후에는 컨테이너 이미지와 해당 애플리케이션 리소스에 동일한 변경 사항을 다시 적용한 다음 이제 영구 수정 사항이 예상대로 작동하는지 확인해야 합니다.

다음 명령은 실행 중인 컨테이너를 변경하는데 도움이 됩니다. 포드에 단일 컨테이너가 포함되어 있다고 모두 가정합니다. 그렇지 않은 경우 **-c my-container-name** 옵션을 추가해야 합니다.

**oc rsh my-pod-name**

포드 내부의 쉘을 열어 대화형 및 비대화형으로 쉘 명령을 실행합니다.

```
oc cp /local/path my-pod-name:/container/path
```

로컬 파일을 포드 내부의 위치에 복사합니다. 또한 포드 내부에서 로컬 파일 시스템으로 인수를 바꾸고 파일을 복사할 수 있습니다. 또한 여러 파일을 한 번에 복사하려면 `oc rsync` 명령을 참조하십시오.

```
oc port-forward my-pod-name local-port:remote-port
```

워크스테이션의 `local-port`에서 포드의 `local-port`까지 TCP 터널을 만듭니다. `oc port-forward`가 실행을 유지하는 동안 터널은 활성화되어 있습니다. 이를 사용하면 경로를 통해 노출하지 않고 포드에 대한 네트워크 액세스를 얻을 수 있습니다. 터널은 `localhost`에서 시작하므로 다른 시스템에서 액세스할 수 없습니다.

## OpenShift CLI 명령 문제 해결

예를 들어 `oc` 명령이 실패하는 이유와 하위 수준 작업의 문제를 해결하여 원인을 확인해야 하는 이유를 알 수 없는 경우가 있습니다. 예를 들어 내부적으로 수행되는 `oc` 명령의 특정 호출을 알고 있어야 하는 경우 `k8s` 모듈을 사용하여 Ansible 플레이북과 같은 OpenShift 및 Kubernetes API 요청을 수행하는 자동화 도구로 동작을 복제할 수 있습니다.

`--loglevel` 수준 옵션은 수준 6부터 시작하며 OpenShift API 요청을 표시합니다. 수준을 높이는 경우에는(최대 10) HTTP 요청 헤더 및 응답 본문과 같이 해당 요청에 대한 자세한 정보가 추가됩니다. 또한 수준 10에는 각 요청을 복제할 수 있는 `curl` 명령이 포함되어 있습니다.

이러한 두 명령을 시도하고 모든 프로젝트에서 해당 출력을 비교할 수 있습니다.

```
[user@demo ~]$ oc get pod --loglevel 6
```

```
[user@demo ~]$ oc get pod --loglevel 10
```

경우에 따라 `oc` 명령에서 OpenShift API 요청을 인증하는 데 사용하는 인증 토큰만 필요로 합니다. 이 토큰을 사용하면 자동화 도구가 사용자로 로그인한 것처럼 OpenShift API 요청을 수행할 수 있습니다. 토큰을 가져오려면 `oc whoami` 명령의 `-t` 옵션을 사용합니다.

```
[user@demo ~]$ oc whoami -t
```



### 참조

OpenShift 이벤트에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-containers-events](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-events)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서에서 클러스터 작업 장의 OpenShift Container Platform 클러스터의 시스템 이벤트 정보 보기 섹션을 참조하십시오.

실행 중인 컨테이너에 파일을 복사하는 방법에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-containers-copying-files](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-copying-files)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서에서 컨테이너 작업 장의 OpenShift Container Platform 컨테이너의 파일 복사 섹션을 참조하십시오.

실행 중인 컨테이너에서 명령을 실행하는 방법에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-containers-remote-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-remote-commands)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서에서 컨테이너 작업 장의 OpenShift Container Platform 컨테이너의 원격 명령 실행 섹션을 참조하십시오.

실행 중인 컨테이너에 로컬 포트를 전달하는 방법에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-containers-port-forwarding](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-port-forwarding)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서에서 컨테이너 작업 장의 포트 전달을 통해 컨테이너의 애플리케이션에 액세스 섹션을 참조하십시오.

집계된 로깅에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/logging/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/logging/index)

에 있는 Red Hat OpenShift Container Platform 4.5 로깅 설명서를 참조하십시오.

### ClusterOperator 사용자 지정 리소스

<https://github.com/openshift/cluster-version-operator/blob/master/docs/dev/clusteroperator.md>

## ▶ 연습 가이드

# OpenShift 클러스터 및 애플리케이션의 문제 해결

이 연습에서는 OpenShift 제어판과 애플리케이션 배포를 통해 일반적인 문제를 해결하는 데 도움이 되는 명령을 실행합니다.

### 결과

다음을 수행할 수 있습니다.

- OpenShift 클러스터의 일반 상태를 검사합니다.
- OpenShift 작업자 노드에서 실행 중인 로컬 서비스 및 포드를 검사합니다.
- 애플리케이션 배포 문제를 진단하고 해결합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 사용하면 클러스터 API에 연결할 수 있으며 활동에서 사용할 리소스 파일이 생성됩니다. 또한 이 연습을 수행하는 동안 진단하고 수정할 애플리케이션을 사용하여 `execute-troubleshoot` 프로젝트를 만듭니다.

```
[student@workstation ~]$ lab execute-troubleshoot start
```

#### ▶ 1. OpenShift 클러스터에 로그인하고 클러스터 노드의 상태를 검사합니다.

- 1.1. `/usr/local/etc/ocp4.config`에서 액세스할 수 있는 강의실 구성 파일을 찾습니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. `kubeadmin` 사용자로 클러스터에 로그인합니다. 메시지가 표시되면 비보안 인증서를 수락합니다.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PWD} \
> https://api.ocp4.example.com:6443
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y

Login successful.
...output omitted...
```

- 1.3. 클러스터의 모든 노드가 준비되었는지 확인합니다.

```
[student@workstation ~]$ oc get nodes
NAME STATUS ROLES AGE VERSION
master01 Ready master,worker 2d v1.18.3+012b3ec
master02 Ready master,worker 2d v1.18.3+012b3ec
master03 Ready master,worker 2d v1.18.3+012b3ec
```

- 1.4. 사용 가능한 모든 CPU 및 메모리를 거의 다 사용하고 있는 노드가 있는지 확인합니다.

다음 명령을 몇 번 반복하여 노드에서 CPU 및 메모리의 실제 사용량을 확인합니다. 표시되는 숫자는 명령을 반복할 때마다 조금씩 변경되어야 합니다.

```
[student@workstation ~]$ oc adm top node
NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%
master01 499m 14% 3235Mi 21%
master02 769m 21% 4933Mi 33%
master03 1016m 29% 6087Mi 40%
```

- 1.5. `oc describe` 명령을 사용하여 문제를 나타내는 모든 조건이 false인지 확인합니다.

```
[student@workstation ~]$ oc describe node master01
...output omitted...
Conditions:
Type Status ... Message

MemoryPressure False ... kubelet has sufficient memory available
DiskPressure False ... kubelet has no disk pressure
PIDPressure False ... kubelet has sufficient PID available
Ready True ... kubelet is posting ready status
Addresses:
...output omitted...
```

- ▶ 2. 내부 레지스트리 운영자의 로그, 내부 레지스트리 서버 및 노드의 Kubelet을 검토합니다. 이러한 로그에서 아무것도 검색하지는 않지만 찾을 수는 있어야 합니다.

- 2.1. `openshift-image-registry` 프로젝트 내의 모든 포드를 나열한 다음 운영자를 실행하는 포드와 내부 레지스트리 서버를 실행하는 포드를 식별합니다.

```
[student@workstation ~]$ oc get pod -n openshift-image-registry
NAME READY STATUS ...
cluster-image-registry-operator-564bd5dd8f-s46bz 2/2 Running ...
image-registry-794dfc7978-w7w69 1/1 Running ...
...output omitted...
```

- 2.2. 운영자 포드(`cluster-image-registry-operator-xxx`)의 로그를 따릅니다. 다음 명령은 해당 포드에 두 개의 컨테이너가 있으므로 실패합니다.

```
[student@workstation ~]$ oc logs --tail 3 -n openshift-image-registry \
> cluster-image-registry-operator-564bd5dd8f-s46bz
Error from server (BadRequest): a container name must be specified for pod
cluster-image-registry-operator-564bd5dd8f-s46bz, choose one of: [cluster-image-
registry-operator cluster-image-registry-operator-watch]
```

- 2.3. 운영자 포드의 첫 번째 컨테이너 로그를 따릅니다. 출력은 다음 예와 다를 수 있습니다.

```
[student@workstation ~]$ oc logs --tail 3 -n openshift-image-registry \
> -c cluster-image-registry-operator \
> cluster-image-registry-operator-564bd5dd8f-s46bz
I0925 13:15:48.252294 13 controller.go:260] event from workqueue successfully
processed
I0925 13:15:51.261479 13 controller.go:260] event from workqueue successfully
processed
I0925 13:15:54.273422 13 controller.go:260] event from workqueue successfully
processed
```

- 2.4. 이미지 레지스트리 서버 포드의 로그를 따릅니다(이전에 `oc get pod` 명령 출력의 `image-registry-xxx`에서 이전에 실행). 출력은 다음 예와 다를 수 있습니다.

```
[student@workstation ~]$ oc logs --tail 1 -n openshift-image-registry \
> image-registry-794dfc7978-w7w69
time="2019-09-25T21:18:06.827703Z" level=info msg=response
go.version=g01.11.6 http.request.host="10.129.2.44:5000"
http.request.id=f4d83df5-8ed7-4651-81d4-4ed9f758c67d http.request.method=GET
http.request.remoteaddr="10.129.2.50:59500" http.request.uri=/extensions/v2/
metrics http.request.useragent=Prometheus/2.11.0 http.response.contenttype="text/
plain; version=0.0.4" http.response.duration=12.141585ms http.response.status=200
http.response.written=2326
```

- 2.5. 이전 단계에서 CPU 및 메모리 사용량을 검사한 동일한 노드에서 Kubelet 로그를 따릅니다. 출력은 다음 예와 다를 수 있습니다.

```
[student@workstation ~]$ oc adm node-logs --tail 1 -u kubelet master01
-- Logs begin at Wed 2020-07-29 18:24:47 UTC, end at Fri 2020-07-31 20:55:53 UTC.
--
Jul 31 16:23:27.420689 master01 systemd[1]: kubelet.service: Consumed 11min
12.108s CPU time
-- Logs begin at Wed 2020-07-29 18:24:47 UTC, end at Fri 2020-07-31 20:55:53 UTC.
--
Jul 31 20:55:53.080133 master01 hyperkube[1524]: I0731 20:55:53.080122 1524
prober.go:133] Liveness probe for "ovs-22g2c_openshift-sdn(d118e034-2de7-447f-
a79b-fed5f9863840):openvswitch" succeeded
```

- ▶ 3. 이전에 OpenShift 서비스 및 포드를 검사하는데 사용한 항목과 동일한 노드로 쉘 세션을 시작합니다. 서비스를 중지하거나 구성 파일을 편집하는 등의 노드 변경 작업을 수행하지 마십시오.

- 3.1. 노드에서 쉘 세션을 시작한 다음 `chroot` 명령을 사용하여 호스트의 로컬 파일 시스템을 입력합니다.

```
[student@workstation ~]$ oc debug node/master01
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.2# chroot /host
sh-4.2#
```

## 9장 | 클러스터 확인

- 3.2. 여전히 동일한 쉘 세션을 사용하고 Kubelet 및 CRI-O 컨테이너 엔진이 실행 중인지 확인합니다. 명령을 종료하려면 **q**를 입력합니다.

```
sh-4.2# systemctl status kubelet
● kubelet.service - MCO environment configuration
 Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
 Drop-In: /etc/systemd/system/kubelet.service.d
 └─10-mco-default-env.conf
 Active: active (running) since Fri 2020-07-31 16:26:57 UTC; 4h 32min ago
 ...output omitted...
q
```

**cri-o** 서비스에 대해 동일한 명령을 다시 실행합니다. 명령에서 종료하려면 **q**를 입력합니다.

```
sh-4.2# systemctl status cri-o
● cri-o.service - MCO environment configuration
 Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset: disabled)
 Drop-In: /etc/systemd/system/crio.service.d
 └─10-mco-default-env.conf
 Active: active (running) since Fri 2020-07-31 16:26:50 UTC; 4h 35min ago
 ...output omitted...
q
```

- 3.3. 계속 동일한 쉘 세션을 사용하여 **openvswitch** 포드가 실행 중인지 확인합니다.

```
sh-4.2# crictl ps --name openvswitch
CONTAINER ID ... STATE NAME ATTEMPT POD ID
13f0b0ed3497a ... Running openvswitch 0 4bc278dddf007
```

- 3.4. **chroot** 세션 및 쉘 세션을 노드로 종료합니다. 이는 또한 **oc debug node** 명령을 종료합니다.

```
sh-4.4# exit
exit
sh-4.2# exit
exit

Removing debug pod ...
[student@workstation ~]$
```

#### ▶ 4. **execute-troubleshoot** 프로젝트를 입력하여 오류 상태에 있는 포드를 진단합니다.

- 4.1. **execute-troubleshoot** 프로젝트를 사용합니다.

```
[student@workstation ~]$ oc project execute-troubleshoot
Now using project "execute-troubleshoot" on server
"https://api.ocp4.example.com:6443".
```

- 4.2. 프로젝트에 **ErrImagePull** 또는 **ImagePullBackOff** 상태 중에 하나의 포드가 있는지 확인합니다.

```
[student@workstation ~]$ oc get pod
NAME READY STATUS ...
pgsql-7d4cc9d6d-m5r59 0/1 ImagePullBackOff ...
```

4.3. 프로젝트에 포드를 관리하는 Kubernetes 배포가 포함되어 있는지 확인합니다.

```
[student@workstation ~]$ oc status
...output omitted...
deployment/postgresql deploys registry.access.redhat.com/rhscl/postgres-96-rhel7:1
 deployment #1 running for 8 minutes - 0/1 pods
...output omitted...
```

4.4. 현재 프로젝트의 모든 이벤트를 나열하고 포드와 관련된 오류 메시지를 찾습니다.

```
[student@workstation ~]$ oc get events
LAST SEEN TYPE REASON OBJECT MESSAGE
112s Normal Scheduled pod/postgresql-7d4cc9d6d-m5r59 Successfully
 assigned execute-troubleshoot/postgresql-7d4cc9d6d-m5r59 to ip-10-0-143-197.us-
 west-1.compute.internal
21s Normal Pulling pod/postgresql-7d4cc9d6d-m5r59 Pulling
 image "registry.access.redhat.com/rhscl/postgres-96-rhel7:1"
21s Warning Failed pod/postgresql-7d4cc9d6d-m5r59 Failed to
 pull image "registry.access.redhat.com/rhscl/postgres-96-rhel7:1": rpc error:
 code = Unknown desc = Error reading manifest 1 in registry.access.redhat.com/
 rhscl/postgres-96-rhel7: name unknown: Repo not found
21s Warning Failed pod/postgresql-7d4cc9d6d-m5r59 Error:
 ErrImagePull
8s Normal BackOff pod/postgresql-7d4cc9d6d-m5r59 Back-off
 pulling image "registry.access.redhat.com/rhscl/postgres-96-rhel7:1"
8s Warning Failed pod/postgresql-7d4cc9d6d-m5r59 Error:
 ImagePullBackOff
112s Normal SuccessfulCreate replicaset/postgresql-7d4cc9d6d-m5r59 Created pod:
 postgresql-7d4cc9d6d-m5r59
112s Normal ScalingReplicaSet deployment/postgresql
 replica set postgresql-7d4cc9d6d to 1
```

또한 이 출력은 포드 배포를 위한 이미지를 가져오는 데 문제가 있음을 나타냅니다.

4.5. Skopeo를 사용하여 이벤트에서 컨테이너 이미지에 대한 정보를 찾을 수 있습니다.

```
[student@workstation ~]$ skopeo inspect \
> docker://registry.access.redhat.com/rhscl/postgres-96-rhel7:1
FATA[0000] Error parsing image name "docker://registry.access.redhat.com/rhscl/
postgres-96-rhel7:1": Error reading manifest 1 in registry.access.redhat.com/
rhscl/postgres-96-rhel7: name unknown: Repo not found
```

4.6. 컨테이너 이미지의 철자가 잘못된 것 같습니다. **postgres-96-rhel7**을 **postgresql-96-rhel7**로 대체하는 경우 이 기능이 작동하는지 확인합니다.

```
[student@workstation ~]$ skopeo inspect \
> docker://registry.access.redhat.com/rhscl/postgresql-96-rhel7:1
{
 "Name": "registry.access.redhat.com/rhscl/postgresql-96-rhel7",
...output omitted...
```

- 4.7. 이미지 이름이 오류의 근본 원인인지 확인하려면 **psql** 배포를 편집하여 컨테이너 이미지의 이름을 수정합니다. **oc edit** 명령에서는 **vi**를 기본 편집기로 사용합니다.



### 경고

실제 시나리오에서는 PostgreSQL 데이터베이스를 배포하여 YAML을 수정하고 애플리케이션을 다시 배포하도록 요청할 수 있습니다.

```
[student@workstation ~]$ oc edit deployment/sql
...output omitted...
spec:
 containers:
 - env:
 - name: POSTGRESQL_DATABASE
 value: db
 - name: POSTGRESQL_PASSWORD
 value: pass
 - name: POSTGRESQL_USER
 value: user
 image: registry.access.redhat.com/rhscl/postgresql-96-rhel7:1
...output omitted...
```

- 4.8. 새 배포가 활성 상태인지 확인합니다.

```
[student@workstation ~]$ oc status
...output omitted...
deployment #2 running for 10 seconds - 0/1 pods
deployment #1 deployed 5 minutes ago
```

- 4.9. 현재 프로젝트의 모든 포드를 나열합니다. 몇 분 동안 이전에 실패한 포드와 새 포드를 모두 볼 수 있습니다. 새 포드가 준비되고 실행 중이 될 때까지 다음 명령을 반복하고 이전 포드를 더 이상 표시하지 않습니다.

NAME	READY	STATUS	RESTARTS	AGE
sql-544c9c666f-btlw8	1/1	Running	0	55s

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab execute-troubleshoot finish
```

이로써 안내에 따른 연습이 완료됩니다.

# OpenShift 동적 스토리지 소개

## 목표

이 섹션을 마치면 영구저장장치의 구성 요소 및 리소스를 확인하고 영구 볼륨 클레임을 사용하는 애플리케이션을 배포할 수 있습니다.

## 영구저장장치 개요

컨테이너에는 기본적으로 임시 스토리지가 있습니다. 예를 들어 컨테이너를 삭제하면 컨테이너 내의 모든 파일과 데이터도 삭제됩니다. 컨테이너에서는 영구저장장치를 유지 관리하는 두 가지 주요 방법인 볼륨과 바인드 마운트를 사용하여 파일을 보존할 수 있습니다. 볼륨은 선호되는 OpenShift 영구저장장치 관리 방법입니다. 볼륨은 관리자가 수동으로 관리하거나 스토리지 클래스를 통해 동적으로 관리합니다. 로컬 시스템에서 컨테이너를 사용하여 작업하는 개발자는 바인드 마운트를 사용하여 로컬 디렉터리를 컨테이너에 마운트할 수 있습니다.

OpenShift 클러스터 관리자는 Kubernetes 영구 볼륨 프레임워크를 사용하여 클러스터 사용자의 영구저장장치를 관리합니다. 클러스터의 스토리지는 정적 및 동적 방법으로 프로비저닝할 수 있습니다. 정적 프로비저닝을 사용하려면 클러스터 관리자가 영구 볼륨을 수동으로 생성해야 합니다. 동적 프로비저닝에서는 스토리지 클래스를 사용하여 필요에 따라 영구 볼륨을 생성합니다.

OpenShift Container Platform에서는 스토리지 클래스를 사용하여 관리자가 영구저장장치를 제공할 수 있도록 허용합니다. 스토리지 클래스는 클러스터의 스토리지 유형을 설명하고 필요에 따라 동적 스토리지를 프로비저닝하는 방법입니다.

개발자는 영구 볼륨 클레임을 사용하여 영구 볼륨을 애플리케이션에 동적으로 추가합니다. 이 경우 개발자는 세부적인 스토리지 인프라 정보를 알 필요가 없습니다. 정적 프로비저닝을 사용하는 경우 개발자는 사전에 생성된 PV를 사용하거나 클러스터 관리자에게 애플리케이션에 사용할 영구 볼륨을 수동으로 생성하도록 요청합니다.

영구 볼륨 클레임(PVC)은 특정 프로젝트에 속합니다. PVC를 생성하려면 액세스 모드, 크기 등의 옵션을 지정해야 합니다. PVC를 생성한 후에는 다른 프로젝트와 공유할 수 없습니다. 개발자는 PVC를 사용하여 영구 볼륨(PV)에 액세스합니다. 영구 볼륨은 특정 프로젝트로 국한되지 않고 전체 OpenShift 클러스터에서 액세스 할 수 있습니다. 영구 볼륨이 특정 영구 볼륨 클레임에 바인딩되면 해당 영구 볼륨을 다른 영구 볼륨 클레임에 바인딩할 수 없습니다.

## 영구 볼륨(PV) 및 영구 볼륨 클레임 라이프사이클

영구 볼륨 클레임은 영구 볼륨 리소스를 요청합니다. 여기에 사용할 PV는 다른 PVC에 바인딩되어 있지 않아야 합니다. 또한 PV에서 PVC에 지정된 액세스 모드를 제공해야 하고, PVC에서 요구되는 크기 이상이어야 합니다. PVC는 추가 기준(예: 스토리지 클래스 이름)을 지정할 수 있습니다. PVC를 통해 모든 기준에 맞는 PV를 찾을 수 없는 경우 PVC는 보류 상태로 전환되고 적절한 PV를 사용할 수 있을 때까지 기다립니다. 클러스터 관리자가 수동으로 PV를 생성하거나 스토리지 클래스를 통해 PV를 동적으로 생성할 수 있습니다. 바인딩된 영구 볼륨은 포드의 특정 마운트 지점에 볼륨으로 마운트할 수 있습니다(예: PostgreSQL 데이터베이스의 경우 /var/lib/pgsql).

## 동적 프로비저닝 스토리지 확인

사용 가능한 스토리지 클래스를 확인하려면 `oc get storageclass` 명령을 사용합니다. 출력에서 기본 스토리지 클래스를 확인할 수 있습니다. 스토리지 클래스가 존재하는 경우 영구 볼륨 클레임에 맞게 영구 볼

룸이 동적으로 생성됩니다. 스토리지 클래스를 지정하지 않는 영구 볼륨 클레임에서는 기본 스토리지 클래스를 사용합니다.

```
[user@demo ~]$ oc get storageclass
NAME PROVISIONER
nfs-storage (default) nfs-storage-provisioner ...
```



### 참고

강의실 환경에서는 외부의 오픈소스 NFS 프로비전 프로그램을 사용합니다. 프로비전 프로그램은 기존 NFS 서버에서 NFS 영구 볼륨을 동적으로 생성합니다. 이 프로비전 프로그램을 프로덕션 환경에서 사용하지 않는 것이 좋습니다.

## 동적으로 프로비저닝된 스토리지 배포

애플리케이션에 볼륨을 추가하려면 **PersistentVolumeClaim** 리소스를 생성하여 애플리케이션에 볼륨으로 추가합니다. Kubernetes 매니페스트 또는 **oc set volumes** 명령을 사용하여 영구 볼륨 클레임을 만듭니다. 새 영구 볼륨 클레임을 생성하거나 기존 영구 볼륨 클레임을 사용하는 방법 외에, **oc set volumes** 명령으로 배포 또는 배포 구성을 수정하여 영구 볼륨 클레임을 포드 내 볼륨으로 마운트할 수 있습니다.

애플리케이션에 볼륨을 추가하려면 **oc set volumes** 명령을 사용합니다.

```
[user@demo ~]$ oc set volumes deployment/example-application \
> --add --name example-storage --type pvc --claim-class nfs-storage \
> --claim-mode rwo --claim-size 15Gi --mount-path /var/lib/example-app \
> --claim-name example-storage
```

이 명령은 영구 볼륨 클레임 리소스를 생성하여 애플리케이션에 포드 내 볼륨으로 추가합니다.

다음 YAML 예에서는 영구 볼륨 클레임을 지정합니다.

**PersistentVolumeClaim** API 개체를 생성하려면 다음을 수행합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim ①
metadata:
 name: example-pv-claim ②
 labels:
 app: example-application
spec:
 accessModes:
 - ReadWriteOnce ③
 resources:
 requests:
 storage: 15Gi ④
```

- ①** 영구 볼륨 클레임임을 나타냅니다.
- ②** 배포 매니페스트의 **volumes** 섹션에 있는 **persistentVolumeClaim** 요소의 **claimName** 필드에 사용할 이름입니다.
- ③** 스토리지 클래스 프로비전 프로그램에서 이 액세스 모드를 제공해야 합니다. 영구 볼륨이 정적으로 생성된 경우 적합한 영구 볼륨은 이 액세스 모드를 제공해야 합니다.
- ④** 용량을 지정합니다.

- ④ 스토리지 클래스는 이 크기 요청과 일치하는 영구 볼륨을 생성합니다. 영구 볼륨이 정적으로 생성된 경우 적합한 영구 볼륨은 요청된 크기 이상이어야 합니다.

OpenShift는 다음 테이블에 요약된 세 가지 액세스 모드를 정의합니다.

액세스 모드	CLI 약어	설명
ReadWriteMany	RWX	Kubernetes는 볼륨을 여러 노드에서 읽기/쓰기로 마운트할 수 있습니다.
ReadOnlyMany	ROX	Kubernetes는 볼륨을 여러 노드에서 읽기 전용으로 마운트할 수 있습니다.
ReadWriteOnce	RWO	Kubernetes는 볼륨을 단일 노드에서만 읽기/쓰기로 마운트할 수 있습니다.

애플리케이션에 PVC를 추가하려면 다음을 수행합니다.

```
...output omitted...
spec:
 volumes:
 - name: example-pv-storage
 persistentVolumeClaim:
 claimName: example-pv-claim
 containers:
 - name: example-application
 image: registry.redhat.io/rhel8/example-app
 ports:
 - containerPort: 1234
 volumeMounts:
 - mountPath: "/var/lib/example-app"
 name: example-pv-storage
...output omitted...
```

## 영구 볼륨 클레임 삭제

볼륨을 삭제하려면 `oc delete` 명령을 사용하여 영구 볼륨 클레임을 삭제합니다. 스토리지 클래스는 PVC가 제거된 후 볼륨을 회수합니다.

```
[user@demo ~]$ oc delete pvc/example-pvc-storage
```



### 참조

영구저장장치에 대한 자세한 내용은 다음 주소의 Red Hat OpenShift Container Platform 4.5 스토리지 설명서에 있는 영구저장장치 이해 장을 참조하십시오.  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/storage/index#understanding-persistent-storage](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/storage/index#understanding-persistent-storage)

임시 스토리지에 대한 자세한 내용은 다음 주소의 Red Hat OpenShift Container Platform 4.5 스토리지 설명서에 있는 임시 스토리지 이해 장을 참조하십시오.  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/storage/index#understanding-ephemeral-storage](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/storage/index#understanding-ephemeral-storage)

## ▶ 연습 가이드

# OpenShift 동적 스토리지 소개

이 연습에서는 영구 볼륨 클레임을 사용하여 PostgreSQL 데이터베이스를 배포하고 동적으로 할당된 볼륨을 확인합니다.

### 결과

다음을 수행할 수 있습니다.

- OpenShift 클러스터의 기본 스토리지 설정을 확인합니다.
- 영구 볼륨 클레임 생성
- 영구 볼륨을 관리합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 사용하면 클러스터 API에 연결할 수 있으며, 이 연습에 필요한 파일을 다운로드합니다.

```
[student@workstation ~]$ lab install-storage start
```

▶ 1. OpenShift 클러스터에 로그인합니다.

- 1.1. **/usr/local/etc/ocp4.config**에서 액세스할 수 있는 강의실 구성 파일을 찾습니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. **kubeadmin** 사용자로 클러스터에 로그인합니다. 메시지가 표시되면 비보안 인증서를 수락합니다.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
> https://api.ocp4.example.com:6443
...output omitted...
```

▶ 2. **install-storage**라는 새 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project install-storage
Now using project "install-storage" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

▶ 3. 기본 스토리지 클래스를 확인합니다.

```
[student@workstation ~]$ oc get storageclass
NAME PROVISIONER RECLAIMPOLICY ...
nfs-storage (default) nfs-storage-provisioner Delete ...
```

- ▶ 4. registry.redhat.io/rhel8/postgresql-12:1-43에 있는 컨테이너 이미지를 사용하여 새 데이터베이스 배포를 생성합니다.

```
[student@workstation ~]$ oc new-app --name postgresql-persistent \
> --docker-image registry.redhat.io/rhel8/postgresql-12:1-43 \
> -e POSTGRESQL_USER=redhat \
> -e POSTGRESQL_PASSWORD=redhat123 \
> -e POSTGRESQL_DATABASE=persistentdb
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "postgresql-persistent" created
deployment.apps "postgresql-persistent" created
service "postgresql-persistent" created
--> Success
...output omitted...
```



### 참고

편의를 위해 ~/D0280/labs/install-storage/commands.txt 파일에는 복사하여 붙여 넣을 수 있는 몇 가지 명령이 포함되어 있습니다.

- ▶ 5. PostgreSQL 데이터베이스용 영구 볼륨을 추가합니다.

- 5.1. 새 영구 볼륨 클레임을 생성하여 postgresql-persistent 배포에 새 볼륨을 추가합니다.

```
[student@workstation ~]$ oc set volumes deployment/postgresql-persistent \
> --add --name postgresql-storage --type pvc --claim-class nfs-storage \
> --claim-mode rwo --claim-size 10Gi --mount-path /var/lib/pgsql \
> --claim-name postgresql-storage
deployment.apps/postgresql-persistent volume updated
```

- 5.2. 새 PVC가 생성되었는지 확인합니다.

```
[student@workstation ~]$ oc get pvc
NAME STATUS ... CAPACITY ACCESS MODES STORAGECLASS AGE
postgresql-storage Bound ... 10Gi RWO nfs-storage 25s
```

- 5.3. 새 PV가 생성되었는지 확인합니다.

```
[student@workstation ~]$ oc get pv \
> -o custom-columns=NAME:.metadata.name,CLAIM:.spec.claimRef.name
NAME CLAIM
pvc-26cc804a-4ec2-4f52-b6e5-84404b4b9def image-registry-storage
pvc-65c3cce7-45eb-482d-badf-a6469640bd75 postgresql-storage
```

- ▶ 6. ~/D0280/labs/install-storage/init\_data.sh 스크립트를 사용하여 데이터베이스를 채웁니다.

- 6.1. init\_data.sh 스크립트를 실행합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/install-storage
[student@workstation install-storage]$./init_data.sh
Populating characters table
CREATE TABLE
INSERT 0 5
```

- 6.2. ~/D0280/labs/install-storage/check\_data.sh 스크립트를 사용하여 데이터베이스가 제대로 채워졌는지 확인합니다.

```
[student@workstation install-storage]$./check_data.sh
Checking characters table
+-----+
| id | name | nationality |
+-----+
| 1 | Wolfgang Amadeus Mozart | Prince-Archbishopric of Salzburg |
| 2 | Ludwig van Beethoven | Bonn, Germany |
| 3 | Johann Sebastian Bach | Eisenach, Germany |
| 4 | José Pablo Moncayo | Guadalajara, México |
| 5 | Niccolò Paganini | Genoa, Italy |
(5 rows)
```

- ▶ 7. postgresql-persistent 배포를 제거하고 동일한 영구 볼륨을 사용하는 postgresql-deployment2라는 배포를 추가로 생성합니다. 데이터가 유지되는지 확인합니다.

- 7.1. app=postgresql-persistent 레이블이 포함된 모든 리소스를 삭제합니다.

```
[student@workstation install-storage]$ oc delete all -l app=postgresql-persistent
service "postgresql-persistent" deleted
deployment.apps "postgresql-persistent" deleted
imagestream.image.openshift.io "postgresql-persistent" deleted
```

- 7.2. postgresql-persistent 배포와 동일한 초기화 데이터를 사용하여 postgresql-persistent2 배포를 생성합니다.

```
[student@workstation install-storage]$ oc new-app --name postgresql-persistent2 \
> --docker-image registry.redhat.io/rhel8/postgresql-12:1-43 \
> -e POSTGRESQL_USER=redhat \
> -e POSTGRESQL_PASSWORD=redhat123 \
> -e POSTGRESQL_DATABASE=persistentdb
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "postgresql-persistent2" created
deployment.apps "postgresql-persistent2" created
service "postgresql-persistent2" created
--> Success
...output omitted...
```

## 9장 | 클러스터 확인

- 7.3. `~/DO280/labs/install-storage/check_data.sh` 스크립트를 사용하여 데이터베이스에 문자 테이블이 없는지 확인합니다.

```
[student@workstation install-storage]$./check_data.sh
Checking characters table
ERROR: 'characters' table does not exist
```

- 7.4. 기존의 `postgresql-persistent` 영구 볼륨 클레임을 `postgresql-persistent2` 배포에 추가합니다.

```
[student@workstation install-storage]$ oc set volumes \
> deployment/postgresql-persistent2 \
> --add --name postgresql-storage --type pvc \
> --claim-name postgresql-storage --mount-path /var/lib/pgsql
deployment.apps/postgresql-persistent2 volume updated
```

- 7.5. `~/DO280/labs/install-storage/check_data.sh` 스크립트를 사용하여 영구 볼륨이 추가되었는지 확인하고, `postgresql-persistent2` 포드에서 이전에 생성한 데이터에 액세스할 수 있는지 확인합니다.

```
[student@workstation install-storage]$./check_data.sh
Checking characters table
id | name | nationality
---+-----+-----
 1 | Wolfgang Amadeus Mozart | Prince-Archbishopric of Salzburg
 2 | Ludwig van Beethoven | Bonn, Germany
...output omitted...
```

## ▶ 8. `postgresql-persistent2` 배포 및 영구 볼륨 클레임을 제거합니다.

- 8.1. `app=postgresql-persistent2` 레이블이 포함된 모든 리소스를 삭제합니다.

```
[student@workstation install-storage]$ oc delete all -l app=postgresql-persistent2
service "postgresql-persistent2" deleted
deployment.apps "postgresql-persistent2" deleted
imagestream.image.openshift.io "postgresql-persistent2" deleted
```

- 8.2. `postgresql-storage` 영구 볼륨 클레임을 제거하여 영구 볼륨을 삭제한 다음 홈 디렉터리로 돌아갑니다.

```
[student@workstation install-storage]$ oc delete pvc/postgresql-storage
persistentvolumeclaim "postgresql-storage" deleted
```

- 8.3. `/home/student` 디렉터리로 돌아갑니다.

```
[student@workstation install-storage]$ cd
```

## 완료

`workstation` 시스템에서 `lab` 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab install-storage finish
```

이로써 안내에 따른 연습이 완료됩니다.

# 요약

이 장에서 학습한 내용:

- Red Hat OpenShift Container Platform의 기본 설치 방법은 풀스택 자동화 및 기존 인프라, 두 가지입니다.
- 향후 릴리스에서는 VMware, Red Hat Virtualization 및 IBM System Z와 같은 클라우드 및 가상화 프로바이더를 추가할 예정입니다.
- Red Hat Enterprise Linux CoreOS를 기반으로 하는 OpenShift 노드에서는 상태를 검사하기 위해 노드에 직접 액세스해야 하는 매우 적은 수의 로컬 서비스를 실행합니다. 대부분의 시스템 서비스는 컨테이너로 실행되며, 주된 예외사항은 CRI 컨테이너 엔진과 Kubelet입니다.
- `oc get node`, `oc adm top`, `oc adm node-logs` 및 `oc adm debug` 명령에서는 OpenShift 노드에 관한 문제점 해결 정보를 제공합니다.

## 인증 및 권한 부여 구성

### 목적

HTPasswd Identity 프로바이더를 사용하여 인증을 구성하고 사용자 및 그룹에 역할을 할당합니다.

### 목표

- OpenShift 인증에 사용할 HTPasswd ID 프로바이더를 구성합니다.
- 역할 기반 액세스 제어를 정의하고 사용자 및 그룹에 권한을 적용합니다.

### 섹션

- ID 프로바이더 구성(안내에 따른 연습)
- RBAC를 사용하여 권한 정의 및 적용(안내에 따른 연습)

### 랩

인증 및 권한 부여 구성

# ID 프로바이더 구성

---

## 목표

이 섹션을 완료하고 나면 OpenShift 인증에 사용할 HTPasswd Identity 프로바이더를 구성할 수 있습니다.

## OpenShift 사용자 및 그룹 설명

인증 및 권한 부여와 관련된 다양한 OpenShift 리소스가 있습니다. 다음은 주요 리소스 유형 및 해당 정의 목록입니다.

### 사용자

OpenShift Container Platform 아키텍처에서 사용자는 API 서버와 상호 작용하는 엔터티입니다. 사용자 리소스는 시스템의 행위자를 나타냅니다. 사용자에게 직접 역할을 추가하거나 사용자가 멤버로 속해 있는 그룹에 역할을 추가하여 권한을 할당합니다.

### ID

Identity 리소스에는 특정 사용자 및 Identity 프로바이더의 성공한 인증 시도 기록이 보관됩니다. 인증 소스와 관련된 모든 데이터가 Identity에 저장됩니다. 단일 사용자 리소스만 ID 리소스와 연결됩니다.

### 서비스 계정

OpenShift에서는 사용자 자격 증명을 가져올 수 없을 때 애플리케이션이 독립적으로 API와 통신할 수 있습니다. 정규 사용자 자격 증명의 무결성을 유지하기 위해 자격 증명을 공유하지 않고 서비스 계정을 대신 사용합니다. 서비스 계정을 사용하면 정규 사용자의 자격 증명을 빌리지 않아도 API 액세스를 제어 할 수 있습니다.

### 그룹

그룹은 특정 사용자 세트를 나타냅니다. 사용자는 하나 또는 여러 그룹에 할당됩니다. 그룹은 권한 정책을 구현하여 동시에 여러 사용자에게 권한을 할당할 때 활용합니다. 예를 들어, 프로젝트 내의 오브젝트에 20명의 사용자가 액세스하도록 허용하려면 각 사용자에게 개별적으로 액세스 권한을 부여하지 않고 그룹을 사용하는 것이 유리합니다. OpenShift Container Platform에서는 클러스터에서 자동으로 프로비저닝하는 시스템 그룹 또는 가상 그룹도 제공합니다.

### 역할

역할은 사용자가 하나 이상의 리소스 유형에 대해 API 작업을 수행하는데 사용할 수 있는 권한 세트입니다. 역할을 할당하여 사용자, 그룹 및 서비스 계정에 권한을 부여합니다.

사용자 및 ID 리소스는 일반적으로 사전에 생성되지 않습니다. 일반적으로 OAuth를 사용하여 대화형으로 로그인하고 나면 OpenShift에서 자동으로 생성합니다.

## API 요청 인증

인증 및 권한 부여는 사용자가 클러스터와 상호 작용하는 데 필요한 두 가지 보안 계층입니다. 사용자가 API에 요청하면 API에서 해당 사용자가 요청과 연결됩니다. 인증 계층은 사용자를 인증합니다. 인증이 완료되면 권한 계층에서 API 요청을 적용할지 또는 거부할지 결정합니다. 권한 계층에서는 역할 기반 액세스 제어(RBAC) 정책을 사용하여 사용자 권한을 결정합니다.

OpenShift API에는 요청을 인증하는 두 가지 방법이 있습니다.

- OAuth 액세스 토큰
- X.509 클라이언트 인증서

요청에 액세스 토큰 또는 인증서가 없으면 인증 계층에서 **system:anonymous** 가상 사용자 및 **system:unauthenticated** 가상 그룹을 할당합니다.

## 인증 운영자 소개

OpenShift Container Platform에서는 OAuth 서버를 실행하는 인증 운영자를 제공합니다. OAuth 서버에서는 API를 인증하려고 할 때 사용자에게 OAuth 액세스 토큰을 제공합니다. ID 프로바이더를 OAuth 서버에서 구성하고 사용할 수 있어야 합니다. OAuth 서버에서 ID 프로바이더를 사용하여 요청자의 ID를 확인합니다. 서버에서는 Identity에 맞게 사용자를 조정하고 사용자에 대한 OAuth 액세스 토큰을 만듭니다. 로그인이 성공하면 OpenShift에서 Identity 및 사용자 리소스를 자동으로 생성합니다.

## ID 프로바이더 소개

다수의 Identity 프로바이더를 사용하도록 OpenShift OAuth 서버를 구성할 수 있습니다. 다음 목록에는 더 일반적인 것이 포함되어 있습니다.

### HTPasswd

**htpasswd** 명령을 사용하여 생성한 자격 증명을 저장하는 시크릿과 비교하여 사용자 이름과 암호의 유효성을 검사합니다.

### Keystone

OpenStack Keystone v3 서버를 사용하여 공유 인증을 활성화합니다.

### LDAP

단순 바인드 인증을 통해 LDAPv3 서버와 비교하여 사용자 이름과 암호의 유효성을 검사하도록 LDAP Identity 프로바이더를 구성합니다.

### GitHub 또는 GitHub Enterprise

GitHub 또는 GitHub Enterprise OAuth 인증 서버와 비교하여 사용자 이름과 암호의 유효성을 검사하도록 GitHub Identity 프로바이더를 구성합니다.

### OpenID Connect

인증 코드 흐름을 사용하여 OpenID Connect ID 프로바이더와 통합합니다.

OAuth 사용자 지정 리소스를 원하는 ID 프로바이더로 업데이트해야 합니다. 동일한 OAuth 사용자 지정 리소스에서 동일하거나 다른 유형의 ID 프로바이더를 여러 개 정의할 수 있습니다.

## 클러스터 관리자로 인증

ID 프로바이더를 구성하고 사용자를 관리할 수 있으려면 먼저 클러스터 관리자로 OpenShift 클러스터에 액세스해야 합니다. 새로 설치된 OpenShift 클러스터에서는 클러스터 관리자 권한을 사용하여 API 요청을 인증하는 두 가지 방법을 제공합니다.

- **kubeadmin** 가상 사용자로 인증합니다. 인증에 성공하면 OAuth 액세스 토큰을 부여합니다.
- 만료되지 않는 X.509 클라이언트 인증서가 포함된 **kubeconfig** 파일을 사용합니다.

추가 사용자를 만들고 서로 다른 액세스 수준을 부여하려면 ID 프로바이더를 구성하고 사용자에게 역할을 할당해야 합니다.

## X.509 인증서를 사용하여 인증

설치하는 동안 OpenShift 설치 프로그램에서 **auth** 디렉터리에 고유한 **kubeconfig** 파일을 생성합니다. **kubeconfig** 파일에는 X.509 인증서를 비롯하여 CLI에서 클라이언트를 올바른 API 서버에 연결하는데 사용하는 특정 세부 정보 및 매개 변수가 포함되어 있습니다.

설치 로그에는 **kubeconfig** 파일의 위치가 있습니다.

```
INFO Run 'export KUBECONFIG=root/auth/kubeconfig' to manage the cluster with 'oc'.
```



### 참고

강의실 환경에서 utility 시스템은 **kubeconfig** 파일을 /home/lab/ocp4/auth/ **kubeconfig**에 저장합니다.

**kubeconfig** 파일을 사용하여 **oc** 명령을 인증하려면 파일을 워크스테이션에 복사하고 절대 또는 상대 경로를 **KUBECONFIG** 환경 변수에 설정해야 합니다. 그런 다음 OpenShift에 로그인하지 않고 클러스터 관리자 권한이 필요한 **oc**를 실행할 수 있습니다.

```
[user@demo ~]$ export KUBECONFIG=/home/user/auth/kubeconfig
[user@demo ~]$ oc get nodes
```

대체 방법으로 **oc** 명령의 **--kubeconfig** 옵션을 사용할 수 있습니다.

```
[user@demo ~]$ oc --kubeconfig /home/user/auth/kubeconfig get nodes
```

## 가상 사용자를 사용하여 인증

설치가 완료되면 OpenShift에서 **kubeadmin** 가상 사용자를 생성합니다. **kube-system** 네임스페이스의 **kubeadmin** 시크릿에는 **kubeadmin** 사용자의 해시된 암호가 포함되어 있습니다. **kubeadmin** 사용자에게는 클러스터 관리자 권한이 있습니다.

OpenShift 설치 프로그램은 클러스터에 고유한 **kubeadmin** 암호를 동적으로 생성합니다. 설치 로그에서는 클러스터에 로그인하는 데 사용하는 **kubeadmin** 자격 증명을 제공합니다. 클러스터 설치 로그에서는 콘솔 액세스에 사용하는 로그인, 암호 및 URL도 제공합니다.

```
...output omitted...
INFO The cluster is ready when 'oc login -u kubeadmin -p shdU_trbi_6ucX_edbu_aqop'
...output omitted...
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.ocp4.example.com
INFO Login to the console with user: kubeadmin, password: shdU_trbi_6ucX_edbu_aqop
```



### 참고

강의실 환경에서 utility 시스템은 **kubeadmin** 사용자의 암호를 /home/lab/ocp4/auth/ **kubeconfig** 파일에 저장합니다.

## 가상 사용자 삭제

ID 프로바이더를 정의하고 새 사용자를 만들며 해당 사용자를 **cluster-admin** 역할에 할당한 후 **kubeadmin** 사용자 자격 증명을 제거하여 클러스터 보안을 향상할 수 있습니다.

```
[user@demo ~]$ oc delete secret kubeadmin -n kube-system
```

**경고**

클러스터 관리자 권한을 사용하여 다른 사용자를 구성하기 전에 **kubeadmin** 시크릿을 삭제하는 경우, 클러스터를 관리할 수 있는 유일한 방법은 **kubeconfig** 파일을 사용하는 것입니다. 안전한 위치에 이 파일의 복사본이 없으면 클러스터에 대한 관리 액세스 권한을 복구할 수 없습니다. 유일한 대안은 클러스터를 삭제하고 다시 설치하는 것입니다.

## HTPasswd ID 프로바이더 구성

HTPasswd ID 프로바이더가 Apache HTTP 서버 프로젝트에서 **htpasswd** 명령을 사용하여 생성한 사용자 이름 및 암호를 포함하는 시크릿과 비교하여 사용자의 유효성을 검사합니다. 클러스터 관리자만 HTPasswd 시크릿 내부의 데이터를 변경할 수 있습니다. 일반 사용자는 자신의 암호를 변경할 수 없습니다.

소수의 사용자 집합으로 이루어진 개념 증명 환경에서는 HTPasswd ID 프로바이더를 사용하여 사용자를 관리하는 것으로 충분할 수 있습니다. 그러나 대부분의 프로덕션 환경에는 조직의 ID 관리 시스템과 통합되는 보다 강력한 ID 프로바이더가 필요합니다.

## OAuth 사용자 지정 리소스 구성

HTPasswd ID 프로바이더를 사용하려면 OAuth 사용자 지정 리소스를 편집하여 **.spec.identityProviders** 배열에 항목을 추가해야 합니다.

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
 name: cluster
spec:
 identityProviders:
 - name: my_htpasswd_provider ❶
 mappingMethod: claim ❷
 type: HTPasswd
 htpasswd:
 fileData:
 name: htpasswd-secret ❸
```

- ❶ 이 프로바이더 이름을 프로바이더 사용자 이름 앞에 붙여 ID 이름을 구성합니다.
- ❷ 프로바이더 ID와 사용자 오브젝트 간에 매핑이 설정되는 방법을 제어합니다.
- ❸ **htpasswd** 명령을 사용하여 생성된 데이터를 포함하는 기존 시크릿입니다.

## OAuth 사용자 지정 리소스 업데이트

OAuth 사용자 지정 리소스를 업데이트하려면 **oc get** 명령을 사용하여 기존 OAuth 클러스터 리소스를 YAML 형식의 파일로 내보냅니다.

```
[user@demo ~]$ oc get oauth cluster -o yaml > oauth.yaml
```

그런 다음 텍스트 편집기에서 결과 파일을 열고 포함된 ID 프로바이더 설정을 필요한 대로 변경합니다.

수정을 완료하고 파일을 저장한 후에는 **oc replace** 명령을 사용하여 새 사용자 지정 리소스를 적용해야 합니다.

```
[user@demo ~]$ oc replace -f oauth.yaml
```

## HTPasswd ID 프로바이더로 사용자 관리

HTPasswd ID 프로바이더를 사용하여 사용자 자격 증명을 관리하려면 임시 `htpasswd` 파일을 생성하고 파일을 변경한 다음, 이러한 변경 사항을 시크릿에 적용해야 합니다.

### HTPasswd 파일 생성

`httpd-utils` 패키지는 `htpasswd` 유ти리티를 제공합니다. `httpd-utils` 패키지를 시스템에 설치하고 사용할 수 있어야 합니다.

`htpasswd` 파일을 생성합니다.

```
[user@demo ~]$ htpasswd -c -B -b /tmp/htpasswd student redhat123
```



#### 중요

`-c` 옵션은 새 파일을 만들 때만 사용하십시오. 파일이 이미 있는 경우 `-c` 옵션을 사용하면 모든 파일 콘텐츠가 교체됩니다.

자격 증명을 추가하거나 업데이트합니다.

```
[user@demo ~]$ htpasswd -b /tmp/htpasswd student redhat1234
```

자격 증명을 삭제합니다.

```
[user@demo ~]$ htpasswd -D /tmp/htpasswd student
```

### HTPasswd 시크릿 만들기

HTPasswd 프로바이더를 사용하려면 `htpasswd` 파일 데이터가 포함된 시크릿을 만들어야 합니다. 다음 예제에서 이름이 `htpasswd-secret`인 시크릿을 사용합니다.

```
[user@demo ~]$ oc create secret generic htpasswd-secret \
> --from-file htpasswd=/tmp/htpasswd -n openshift-config
```



#### 중요

HTPasswd Identity 프로바이더가 사용하는 시크릿을 사용하려면 파일 경로를 지정하기 전에 `htpasswd=` 접두사를 추가해야 합니다.

### 시크릿 데이터 추출

사용자를 추가하거나 제거하는 경우 관리자는 로컬 `htpasswd` 파일의 유효성을 가정할 수 없습니다. `htpasswd` 파일이 있는 시스템에 관리자가 없을 수도 있습니다. 실제 시나리오에서는 관리자가 `oc extract` 명령을 사용해야 합니다.

기본적으로 `oc extract` 명령은 구성 맵 또는 시크릿 내의 각 키를 별도의 파일로 저장합니다. 또는 모든 데이터의 경로를 파일로 재지정하고 표준 출력으로 표시할 수 있습니다. `htpasswd-secret` 시크릿에서 `/tmp/` 디렉터리로 데이터를 추출하려면 다음 명령을 사용합니다. 파일이 이미 있는 경우 `--confirm` 옵션을 사용하면 파일이 교체됩니다.

```
[user@demo ~]$ oc extract secret/htpasswd-secret -n openshift-config \
> --to /tmp/ --confirm
/tmp/htpasswd
```

## HTPasswd 시크릿 업데이트

사용자를 추가, 변경 또는 삭제한 후 시크릿을 업데이트해야 합니다. `oc set data secret` 명령을 사용하여 시크릿을 업데이트합니다. 파일 이름이 `htpasswd`가 아닌 경우 시크릿에서 `htpasswd` 키를 업데이트 하려면 `htpasswd=`을 지정해야 합니다.

다음 명령은 `/tmp/htpasswd` 파일의 콘텐츠를 사용하여 `openshift-config` 네임스페이스의 `htpasswd-secret` 시크릿을 업데이트합니다.

```
[user@demo ~]$ oc set data secret/htpasswd-secret \
> --from-file htpasswd=/tmp/htpasswd -n openshift-config
```

시크릿을 업데이트한 후에는 OAuth 운영자가 `openshift-authentication` 네임스페이스에 포드를 재배포합니다. 다음을 실행하여 새 OAuth 포드의 재배포를 모니터링합니다.

```
[user@demo ~]$ watch oc get pods -n openshift-authentication
```

새 포드를 배포한 후에는 시크릿에 추가, 변경 또는 삭제를 테스트합니다.

## 사용자 및 ID 삭제

사용자를 삭제해야 하는 경우가 발생하면 ID 프로바이더에서 사용자를 삭제하는 것만으로는 충분하지 않습니다. 사용자 및 ID 리소스도 삭제해야 합니다.

`htpasswd` secret에서 암호를 제거하고 로컬 `htpasswd` 파일에서 사용자를 제거한 다음 시크릿을 업데이트해야 합니다.

`htpasswd`에서 사용자를 삭제하려면 다음 명령을 실행합니다.

```
[user@demo ~]$ htpasswd -D /tmp/htpasswd manager
```

시크릿을 업데이트하여 사용자 암호의 나머지를 모두 제거합니다.

```
[user@demo ~]$ oc set data secret/htpasswd-secret \
> --from-file htpasswd=/tmp/htpasswd -n openshift-config
```

다음 명령을 사용하여 사용자 리소스를 제거합니다.

```
[user@demo ~]$ oc delete user manager
user.user.openshift.io "manager" deleted
```

ID 리소스에는 ID 프로바이더의 이름이 포함됩니다. `manager` 사용자의 ID 리소스를 삭제하려면 리소스를 찾아 삭제합니다.

```
[user@demo ~]$ oc get identities | grep manager
my_htpasswd_provider:manager my_htpasswd_provider manager manager ...
[user@demo ~]$ oc delete identity my_htpasswd_provider:manager
identity.user.openshift.io "my_htpasswd_provider:manager" deleted
```

## 관리 권한 할당

클러스터 전체 **cluster-admin** 역할은 사용자와 그룹에 클러스터 관리 권한을 부여합니다. 이 역할을 사용하면 사용자가 클러스터의 모든 리소스에 모든 작업을 수행할 수 있습니다. 다음 예제에서는 **cluster-admin** 역할을 **student** 사용자에게 할당됩니다.

```
[user@demo ~]$ oc adm policy add-cluster-role-to-user cluster-admin student
```



### 참조

ID 프로바이더에 관한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/authentication\\_and\\_authorization/index#understanding-identity-provider](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#understanding-identity-provider)

에 있는 Red Hat OpenShift Container Platform 4.5 권한 부여 및 인증 설명서의 ID 프로바이더 구성 이해 장을 참조하십시오.

## ▶ 연습 가이드

# ID 프로바이더 구성

이 연습에서는 HTPasswd ID 프로바이더를 구성하고 클러스터 관리자를 위한 사용자를 만듭니다.

### 결과

다음을 수행할 수 있습니다.

- HTPasswd 인증을 위한 사용자 및 암호를 생성합니다.
- HTPasswd 인증에 사용할 ID 프로바이더를 구성합니다.
- 사용자에게 클러스터 관리 권한을 할당합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

```
[student@workstation ~]$ lab auth-provider start
```

이 명령을 사용하면 클러스터 API에 연결할 수 있고 **httpd-utils** 패키지가 설치되어 인증 설정이 설치 기본값으로 구성됩니다.

- ▶ 1. 두 개의 htpasswd 사용자인 **admin** 및 **developer**의 항목을 추가합니다. **admin**에게는 암호로 **redhat**, **developer**에게는 **developer**를 할당합니다.

- 1.1. **/usr/local/etc/ocp4.config**에서 액세스할 수 있는 강의실 구성 파일을 찾습니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. **~/D0280/labs/auth-provider/** 디렉터리에 **htpasswd**라는 HTPasswd 인증 파일을 만듭니다. 암호가 **redhat**인 **admin** 사용자를 추가합니다. 파일 이름은 임의로 지정되지만 이 연습에서는 **~/D0280/labs/auth-provider/htpasswd** 파일을 사용합니다.

**htpasswd** 명령을 사용하여 HTPasswd 인증 파일을 사용자 이름과 암호화된 암호로 채웁니다. **-B** 옵션은 bcrypt 암호화를 사용합니다. **htpasswd** 명령은 암호화 옵션을 지정하지 않은 경우 기본적으로 MD5 암호화를 사용합니다.

```
[student@workstation ~]$ htpasswd -c -B -b ~/D0280/labs/auth-provider/htpasswd \
> admin redhat
Adding password for user admin
```

- 1.3. 암호가 **developer**인 **developer** 사용자를 **~/D0280/labs/auth-provider/htpasswd** 파일에 추가합니다.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
> developer developer
Adding password for user developer
```

- 1.4. ~/D0280/labs/auth-provider/htpasswd 파일의 콘텐츠를 검토하고 admin 사용자와 developer 사용자용으로 하나씩 해시 암호가 있는 두 개의 항목이 포함되어 있는지 확인합니다.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-provider/htpasswd
admin:$2y$05$QPuzHd106IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZU16
developer:$apr1$0Nzmc1rh$yGtne1k.JX6L5s5wNa2ye.
```

▶ 2. OpenShift에 로그인하고 HTPasswd 사용자 파일을 포함하는 암호를 생성합니다.

- 2.1. kubeadmin 사용자로 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2.2. /home/student/D0280/labs/auth-provider/htpasswd 파일에서 시크릿을 생성합니다. HTPasswd ID 프로바이더를 사용하려면 HTPasswd 사용자 파일 /home/student/D0280/labs/auth-provider/htpasswd를 포함하는 htpasswd라는 키가 있는 시크릿을 정의해야 합니다.



### 중요

HTPasswd Identity 프로바이더가 사용하는 시크릿을 사용하려면 파일 경로를 지정하기 전에 htpasswd= 접두사를 추가해야 합니다.

```
[student@workstation ~]$ oc create secret generic localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers created
```

- 2.3. admin 사용자에게 cluster-admin 역할을 할당합니다.



### 참고

admin 사용자를 찾을 수 없음을 나타내는 출력은 무시해도 안전합니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
> cluster-admin admin
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

- ▶ 3. 사용자가 인증할 수 있도록 클러스터의 HTPasswd ID 프로바이더를 업데이트합니다. 사용자 지정 리소스 파일을 구성하고 클러스터를 업데이트합니다.

- 3.1. 기존 OAuth 리소스를 ~/D0280/labs/auth-provider 디렉터리의 `oauth.yaml`라는 파일에 내보냅니다.

```
[student@workstation ~]$ oc get oauth cluster \
> -o yaml > ~/D0280/labs/auth-provider/oauth.yaml
```



### 참고

전체 사용자 지정 리소스 파일이 포함된 `oauth.yaml` 파일은 편의를 위해 ~/D0280/solutions/auth-provider에 다운로드됩니다.

- 3.2. 선호하는 텍스트 편집기로 ~/D0280/labs/auth-provider/oauth.yaml 파일을 편집합니다. `identityProviders` 및 `fileData` 구조의 이름을 선택할 수 있습니다. 이 연습에서는 각각 `myusers` 및 `localusers` 값을 사용합니다.

전체 사용자 지정 리소스는 다음과 일치해야 합니다. `htpasswd`, `mappingMethod`, `name` 및 `type`은 동일한 들여쓰기 수준에 있습니다.

```
apiVersion: config.openshift.io/v1
kind: OAuth
...output omitted...
spec:
 identityProviders:
 - htpasswd:
 fileData:
 name: localusers
 mappingMethod: claim
 name: myusers
 type: HTPasswd
```

- 3.3. 이전 단계에서 정의한 사용자 지정 리소스를 적용합니다.

```
[student@workstation ~]$ oc replace -f ~/D0280/labs/auth-provider/oauth.yaml
oauth.config.openshift.io/cluster replaced
```



### 참고

`oc replace` 명령이 성공하면 `openshift-authentication` 네임스페이스의 포드가 다시 배포됩니다. 이전에 생성된 시크릿이 제대로 생성된 경우 HTPasswd Identity 프로바이더를 사용하여 로그인할 수 있습니다.

- ▶ 4. `admin`으로 로그인하고 `developer`로 HTPasswd 사용자 구성 확인합니다.

- 4.1. `admin` 사용자로 클러스터에 로그인하여 HTPasswd 인증이 올바르게 구성되었는지 확인합니다. 인증 운영자가 이전 단계에서 구성 변경 사항을 로드하는데 약간의 시간이 걸립니다.

**참고**

인증에 실패하면 잠시 기다린 후 다시 시도하십시오.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 4.2. `oc get nodes` 명령을 사용하여 `admin` 사용자에 `cluster-admin` 역할이 있는지 확인합니다.

```
[student@workstation ~]$ oc get nodes
NAME STATUS ROLES AGE VERSION
master01 Ready master,worker 2d2h v1.18.3+012b3ec
master02 Ready master,worker 2d2h v1.18.3+012b3ec
master03 Ready master,worker 2d2h v1.18.3+012b3ec
```

- 4.3. `developer` 사용자로 클러스터에 로그인하여 HTPasswd 인증이 올바르게 구성되었는지 확인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 4.4. `oc get nodes` 명령을 사용하여 `developer` 및 `admin` 사용자가 동일한 수준의 액세스 권한을 공유하지 않는지 확인합니다.

```
[student@workstation ~]$ oc get nodes
Error from server (Forbidden): nodes is forbidden: User "developer" cannot list resource "nodes" in API group "" at the cluster scope
```

- 4.5. `admin` 사용자로 로그인하여 현재 사용자를 나열합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc get users
NAME UID FULL NAME IDENTITIES
admin 31f6ccd2-6c58-47ee-978d-5e5e3c30d617
developer d4e77b0d-9740-4f05-9af5-ecfc08a85101
```

- 4.6. 현재 ID 목록을 표시합니다.

```
[student@workstation ~]$ oc get identity
NAME IDP NAME IDP USER NAME USER NAME USER UID
myusers:admin myusers admin admin 31f6ccd2-6c58-47...
myusers:developer myusers developer developer d4e77b0d-9740-4f...
```

- ▶ 5. admin 사용자로서 암호가 redhat인 manager라는 새 HTPasswd 사용자를 생성합니다.

- 5.1. 시크릿의 파일 데이터를 ~/D0280/labs/auth-provider/htpasswd 파일로 추출합니다.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
> --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

- 5.2. 암호가 redhat인 추가 사용자 manager에 대한 항목을 ~/D0280/labs/auth-provider/htpasswd 파일에 추가합니다.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
> manager redhat
Adding password for user manager
```

- 5.3. ~/D0280/labs/auth-provider/htpasswd 파일의 콘텐츠를 검토하고 admin, developer 및 manager 사용자용 해시 암호가 하나씩 있는 세 개의 항목이 포함되어 있는지 확인합니다.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-provider/htpasswd
admin:$2y$05$QPuzHd106IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZU16
developer:$apr1$0Nzmc1rh$yGtne1k.JX6L5s5wNa2ye.
manager:$apr1$CJ/tpa6a$sLhjPkIIAy755ZArTT5EH/
```

- 5.4. 사용자를 추가한 후에는 시크릿을 업데이트해야 합니다. `oc set data secret` 명령을 사용하여 시크릿을 업데이트합니다. 오류가 표시되면 oauth 운영자가 여전히 다시 로드될 수 있으므로, 잠시 후에 명령을 다시 실행합니다.

```
[student@workstation ~]$ oc set data secret/localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers data updated
```

- 5.5. 인증 운영자가 다시 로드할 때까지 잠시 기다린 다음 manager 사용자로 클러스터에 로그인합니다.



### 참고

인증에 실패하면 잠시 기다린 후 다시 시도하십시오.

```
[student@workstation ~]$ oc login -u manager -p redhat
Login successful.
...output omitted...
```

- ▶ 6. auth-provider라는 새 프로젝트를 생성한 다음 developer 사용자가 프로젝트에 액세스할 수 없는지 확인합니다.

- 6.1. manager 사용자로 새 auth-provider 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project auth-provider
Now using project "auth-provider" on server https://api.ocp4.example.com:6443".
...output omitted...
```

6.2. developer 사용자로 로그인한 다음 auth-provider 프로젝트를 삭제해 봅니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
[student@workstation ~]$ oc delete project auth-provider
Error from server (Forbidden): projects.project.openshift.io "auth-provider"
is forbidden: User "developer" cannot delete resource "projects"
in API group "project.openshift.io" in the namespace "auth-provider"
```

#### ▶ 7. manager 사용자의 암호를 변경합니다.

7.1. admin 사용자로 로그인하고 시크릿의 파일 데이터를 ~/D0280/labs/auth-provider/htpasswd 파일로 추출합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
> --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

7.2. 임의의 사용자 암호를 생성하고 이를 MANAGER\_PASSWD 변수에 할당합니다.

```
[student@workstation ~]$ MANAGER_PASSWD="$(openssl rand -hex 15)"
```

7.3. MANAGER\_PASSWD 변수에 저장된 암호를 사용하도록 manager 사용자를 업데이트합니다.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
> manager ${MANAGER_PASSWD}
Updating password for user manager
```

7.4. 시크릿을 업데이트합니다.

```
[student@workstation ~]$ oc set data secret/localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers data updated
```

7.5. manager 사용자로 로그인하여 업데이트된 암호를 확인합니다.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}
Login successful.
...output omitted...
```

▶ 8. manager 사용자를 제거합니다.

- 8.1. admin 사용자로 로그인하고 시크릿의 파일 데이터를 ~/D0280/labs/auth-provider/htpasswd 파일로 추출합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
> --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

- 8.2. ~/D0280/labs/auth-provider/htpasswd 파일에서 manager 사용자를 삭제합니다.

```
[student@workstation ~]$ htpasswd -D ~/D0280/labs/auth-provider/htpasswd manager
Deleting password for user manager
```

- 8.3. 시크릿을 업데이트합니다.

```
[student@workstation ~]$ oc set data secret/localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers data updated
```

- 8.4. manager 사용자의 ID 리소스를 삭제합니다.

```
[student@workstation ~]$ oc delete identity "myusers:manager"
identity.user.openshift.io "myusers:manager" deleted
```

- 8.5. manager 사용자의 사용자 리소스를 삭제합니다.

```
[student@workstation ~]$ oc delete user manager
user.user.openshift.io manager deleted
```

- 8.6. 이제 manager 사용자로 로그인하려고 하면 실패합니다.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}
Login failed (401 Unauthorized)
Verify you have provided correct credentials.
```

- 8.7. 현재 사용자를 나열하여 manager 사용자가 삭제되었는지 확인합니다.

```
[student@workstation ~]$ oc get users
NAME UID FULL NAME IDENTITIES
admin 31f6cccd2-6c58-47ee-978d-5e5e3c30d617
developer d4e77b0d-9740-4f05-9af5-ecfc08a85101
```

- 8.8. 현재 ID 목록을 표시하여 manager ID가 삭제되었는지 확인합니다.

```
[student@workstation ~]$ oc get identity
NAME IDP NAME IDP USER NAME USER NAME
myusers:admin myusers admin admin ...
myusers:developer myusers developer developer ...
```

- 8.9. 시크릿을 추출하고 **admin** 및 **developer** 사용자만 표시되는지 확인합니다. **--to -** 를 사용하면 시크릿을 파일에 저장하지 않고 STDOUT로 보냅니다.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config --to -
htpasswd
admin:$2y$05$TizWp/2ct4Edn08gmeMBI09IXujpLqkKAJ0Nldxc/V2XYYMBf6wBy
developer:$apr1$8Bc6txgb$bwHke4cGRGk9C8tQLg.hi1
```

#### ▶ 9. ID 프로바이더를 제거하고 모든 사용자를 정리합니다.

- 9.1. **kubeadmin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD}
Login successful.
...output omitted...
```

- 9.2. **auth-provider** 프로젝트를 삭제합니다.

```
[student@workstation ~]$ oc delete project auth-provider
project.project.openshift.io "auth-provider" deleted
```

- 9.3. 현재 위치에서 리소스를 편집하여 OAuth에서 ID 프로바이더를 제거합니다.

```
[student@workstation ~]$ oc edit oauth
```

**spec:** 에서 모든 행을 삭제한 다음 **{}**를 **spec:** 다음에 추가합니다. 파일에 있는 다른 정보는 모두 변경하지 않고 그대로 둡니다. **spec:** 행은 다음과 일치해야 합니다.

```
...output omitted...
spec: {}
```

변경 사항을 저장하고 **oc edit** 명령이 변경 사항을 적용했는지 확인합니다.

```
oauth.config.openshift.io/cluster edited
```

- 9.4. **openshift-config** 네임스페이스에서 **localusers** 시크릿을 삭제합니다.

```
[student@workstation ~]$ oc delete secret localusers -n openshift-config
secret "localusers" deleted
```

- 9.5. 모든 사용자 리소스를 삭제합니다.

```
[student@workstation ~]$ oc delete user --all
user.user.openshift.io "admin" deleted
user.user.openshift.io "developer" deleted
```

9.6. 모든 ID 리소스를 삭제합니다.

```
[student@workstation ~]$ oc delete identity --all
identity.user.openshift.io "myusers:admin" deleted
identity.user.openshift.io "myusers:developer" deleted
```

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab auth-provider finish
```

이로써 안내에 따른 연습이 완료됩니다.

# RBAC를 사용하여 권한 정의 및 적용

## 목표

이 섹션을 완료하면 역할 기반 액세스 제어를 정의하고 사용자에게 권한을 적용할 수 있습니다.

## 역할 기반 액세스 제어(RBAC)

역할 기반 액세스 제어(RBAC)는 컴퓨터 시스템의 리소스에 대한 액세스를 관리하는 기술입니다. Red Hat OpenShift에서 RBAC를 통해 사용자가 특정 작업을 수행하는 위치가 클러스터인지 아니면 프로젝트인지 결정합니다. 사용자의 책임 수준에 따라 사용할 수 있는 역할은 클러스터와 로컬의 두 가지 유형이 있습니다.



### 참고

권한 부여는 인증을 통한 별도의 단계입니다.

## 권한 부여 프로세스

권한 부여 프로세스는 규칙, 역할 및 바인딩을 통해 관리합니다.

RBAC 오브젝트	설명
규칙	오브젝트 또는 오브젝트 그룹에 허용된 작업입니다.
역할	규칙 세트입니다. 사용자와 그룹은 여러 역할에 연결될 수 있습니다.
바인딩	사용자 또는 그룹을 역할에 할당합니다.

## RBAC 범위

RHOCP(Red Hat OpenShift Container Platform)에서는 사용자 범위와 책임에 따라 두 그룹의 역할과 바인딩을 정의합니다. 즉, 클러스터 역할과 로컬 역할입니다.

역할 수준	설명
클러스터 역할	이 역할 수준의 사용자나 그룹이 OpenShift 클러스터를 관리할 수 있습니다.
로컬 역할	이 역할 수준의 사용자 또는 그룹은 프로젝트 수준에서만 요소를 관리할 수 있습니다.



### 참고

클러스터 역할 바인딩은 로컬 역할 바인딩보다 우선 적용됩니다.

## CLI를 사용하여 RBAC 관리

클러스터 관리자는 `oc adm policy` 명령을 사용하여 클러스터 역할 및 네임스페이스 역할을 추가 및 제거할 수 있습니다.

클러스터 역할을 사용자에 추가하려면 **add-cluster-role-to-user** 하위 명령을 사용합니다.

```
[user@demo ~]$ oc adm policy add-cluster-role-to-user cluster-role username
```

예를 들어, 일반 사용자를 클러스터 관리자로 변경하려면 다음 명령을 사용합니다.

```
[user@demo ~]$ oc adm policy add-cluster-role-to-user cluster-admin username
```

사용자에서 클러스터 역할을 제거하려면 **remove-cluster-role-from-user** 하위 명령을 사용합니다.

```
[user@demo ~]$ oc adm policy remove-cluster-role-from-user cluster-role username
```

예를 들어, 클러스터 관리자를 일반 사용자로 변경하려면 다음 명령을 사용합니다.

```
[user@demo ~]$ oc adm policy remove-cluster-role-from-user cluster-admin username
```

규칙은 작업과 리소스로 정의됩니다. 예를 들어 **create user** 규칙은 **cluster-admin** 역할의 일부입니다.

**oc adm policy who-can** 명령을 사용하여 사용자가 리소스에 대한 작업을 실행할 수 있는지 판별합니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc adm policy who-can delete user
```

## 기본 역할

OpenShift에는 로컬 또는 전체 클러스터에 할당할 수 있는 기본 클러스터 역할 집합이 포함되어 있습니다. OpenShift 리소스에 대한 세밀한 액세스 제어를 위해 이러한 역할을 수정할 수 있지만, 이 과정의 범위를 벗어나는 추가 단계가 필요합니다.

기본 역할	설명
<b>admin</b>	이 역할의 사용자는 다른 사용자에게 프로젝트에 액세스할 수 있도록 액세스 권한을 부여하는 등 모든 프로젝트 리소스를 관리할 수 있습니다.
<b>basic-user</b>	이 역할의 사용자는 프로젝트에 대한 읽기 액세스 권한이 있습니다.
<b>cluster-admin</b>	이 역할의 사용자는 클러스터 리소스에 대한 수퍼유저 액세스 권한을 갖습니다. 이러한 사용자는 클러스터에서 모든 작업을 수행할 수 있으며 모든 프로젝트를 완전히 제어할 수 있습니다.
<b>cluster-status</b>	이 역할의 사용자는 클러스터 상태 정보를 가져올 수 있습니다.
<b>edit</b>	이 역할의 사용자는 서비스 및 배포 구성과 같은 프로젝트에서 일반 애플리케이션 리소스를 만들고 변경하고 삭제할 수 있습니다. 이 사용자는 한계 범위 및 할당량과 같은 관리 리소스에 대한 작업은 수행할 수 없으며 프로젝트 액세스 권한을 관리할 수 없습니다.
<b>self-provisioner</b>	이 역할의 사용자는 새 프로젝트를 만들 수 있습니다. 이는 클러스터 역할이며 프로젝트 역할이 아닙니다.

기본 역할	설명
<b>view</b>	이 역할의 사용자는 프로젝트 리소스를 볼 수 있지만 프로젝트 리소스를 수정할 수는 없습니다.

**admin** 역할은 사용자에게 새 애플리케이션을 만드는 기능 외에도 할당량 및 한계 범위와 같은 프로젝트 리소스에 대한 액세스 권한을 부여합니다. **edit** 역할은 사용자에게 프로젝트 내부에서 개발자로 작업할 수 있는 충분한 액세스 권한을 제공하지만 프로젝트 관리자가 구성한 제한 조건에 따라 작업합니다.

프로젝트 관리자는 **oc policy** 명령을 사용하여 네임스페이스 역할을 추가 및 제거할 수 있습니다.

**add-role-to-user** 하위 명령을 사용하여 사용자에게 지정된 역할을 추가합니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc policy add-role-to-user role-name username -n project
```

예를 들어, 사용자 **dev**를 **wordpress** 프로젝트의 역할 **basic-user**에 추가하려면 다음을 수행합니다.

```
[user@demo ~]$ oc policy add-role-to-user basic-user dev -n wordpress
```

## 사용자 유형

OpenShift Container Platform과의 상호 작용은 사용자와 연결되어 있습니다. OpenShift Container Platform user 오브젝트는 **rolebindings**를 통해 **roles** 역할을 해당 사용자 또는 사용자의 그룹에 추가하여 시스템에서 권한을 부여받을 수 있는 사용자를 나타냅니다.

### 일반 사용자

이 방식으로 대부분의 대화형 OpenShift Container Platform 사용자를 나타냅니다. 일반 사용자는 **User** 오브젝트로 표현됩니다. 이 유형의 사용자는 플랫폼에 대한 액세스가 허용된 사람을 나타냅니다. 일반 사용자의 예로는 **user1** 및 **user2**가 있습니다.

### 시스템 사용자

이 사용자는 대부분 인프라가 정의될 때 자동으로 생성되며, 주로 인프라가 API와 안전하게 상호 작용하도록 하기 위한 것입니다. 시스템 사용자에는 클러스터 관리자(모든 것에 대한 액세스 권한 보유), 노드별 사용자, 사용할 라우터 및 레지스트리의 사용자 및 기타 다양한 사용자가 포함됩니다. 인증되지 않은 요청에는 기본적으로 익명 시스템 사용자를 사용합니다. 시스템 사용자의 예로는 **system:admin**, **system:openshift-registry**, **system:node:node1.example.com**이 있습니다.

### 서비스 계정

프로젝트와 관련된 특별한 시스템 사용자입니다. 일부는 프로젝트가 처음 생성될 때 자동으로 생성되며 프로젝트 관리자가 각 프로젝트 내용에 대한 액세스를 정의하기 위해 추가로 생성할 수 있습니다. 서비스 계정은 포드 또는 배포 구성에 추가 권한을 부여하는 데 사용하는 경우가 많습니다. 서비스 계정은 **ServiceAccount** 오브젝트로 표시됩니다. 서비스 계정 사용자의 예로는 **system:serviceaccount:default:deployer** 및 **system:serviceaccount:foo:builder**가 있습니다.

모든 사용자는 인증을 받아야 OpenShift Container Platform에 액세스할 수 있습니다. 인증이 없거나 잘못된 인증을 사용하는 API 요청은 익명의 시스템 사용자가 요청한 것으로 인증됩니다. 인증이 성공하면 정책에 따라 사용자가 수행할 수 있는 작업이 결정됩니다.



## 참조

Kubernetes 네임스페이스에 대한 자세한 내용은

### Kubernetes 문서

<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>  
의 내용을 참조하십시오.

RBAC에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/  
html-single/authentication\\_and\\_authorization/index#using-rbac](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#using-rbac)

에 있는 Red Hat OpenShift Container Platform 4.5 권한 부여 및 인증 설명서의 RBAC를  
사용하여 권한 정의 및 적용 장을 참조하십시오.

그룹에 관한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/  
html-single/authentication\\_and\\_authorization/index#understanding-authentication](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#understanding-authentication)  
에 있는 Red Hat OpenShift Container Platform 4.5 권한 부여 및 인증 설명서의 인증 이해  
장을 참조하십시오.

## ▶ 연습 가이드

# RBAC를 사용하여 권한 정의 및 적용

이 연습에서는 역할 기반 액세스 제어를 정의하고 사용자에게 권한을 적용합니다.

### 결과

다음을 수행할 수 있습니다.

- OpenShift 클러스터 관리자가 아닌 사용자의 프로젝트 생성 권한을 제거합니다.
- OpenShift 그룹을 생성하고 이 그룹에 멤버를 추가합니다.
- 프로젝트를 생성하고 프로젝트에 프로젝트 관리 권한을 할당합니다.
- 프로젝트 관리자로서 다른 사용자 그룹에 읽기 및 쓰기 권한을 할당합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 실행하면 클러스터 API에 연결할 수 있으며 이 연습에 사용할 몇몇 HTPasswd 사용자를 생성합니다.

```
[student@workstation ~]$ lab auth-rbac start
```

- ▶ 1. OpenShift 클러스터에 로그인하고, `self-provisioner` 클러스터 역할을 할당하는 클러스터 역할 바인딩을 확인합니다.

- 1.1. `admin` 사용자로 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. `self-provisioner` 클러스터 역할을 참조하는 클러스터 역할 바인딩을 모두 나열합니다.

```
[student@workstation ~]$ oc get clusterrolebinding -o wide \
> | grep -E 'NAME|self-provisioner'
NAME ROLE
self-provisioners ... ClusterRole/self-provisioner ...
```

- ▶ 2. `system:authenticated:oauth` 가상 그룹에서 `self-provisioner` 클러스터 역할을 삭제하여 클러스터 관리자가 아닌 모든 사용자로부터 새 프로젝트를 생성하는 권한을 제거합니다.

- 2.1. 이전 단계에서 찾은 `self-provisioners` 클러스터 역할 바인딩에서 `self-provisioner` 클러스터 역할을 `system:authenticated:oauth` 그룹에 할당하는지 확인합니다.

```
[student@workstation ~]$ oc describe clusterrolebindings self-provisioners
Name: self-provisioners
Labels: <none>
Annotations: <none>
Role:
 Kind: ClusterRole
 Name: self-provisioner
Subjects:
 Kind Name Namespace
 ---- --- -----
 Group system:authenticated:oauth
```

- 2.2. `system:authenticated:oauth` 가상 그룹에서 `self-provisioner` 클러스터 역할을 제거합니다. 그러면 `self-provisioners` 역할 바인딩이 삭제됩니다. 변경 사항이 유실 된다는 경고는 무시해도 됩니다.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
> self-provisioner system:authenticated:oauth
Warning: Your changes may get lost whenever a master is restarted,
unless you prevent reconciliation of this rolebinding using the
following command: oc annotate clusterrolebinding.rbac self-provisioner
'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
"system:authenticated:oauth"
```

- 2.3. 역할이 그룹에서 제거되었는지 확인합니다. 클러스터 역할 바인딩 `self-provisioners`는 없어야 합니다.

```
[student@workstation ~]$ oc describe clusterrolebindings self-provisioners
Error from server (NotFound): clusterrolebindings.rbac.authorization.k8s.io "self-
provisioners" not found
```

- 2.4. 다른 클러스터 역할 바인딩에서 `self-provisioner` 클러스터 역할을 참조하는지 판별합니다.

```
[student@workstation ~]$ oc get clusterrolebinding -o wide \
> | grep -E 'NAME|self-provisioner'
NAME ROLE ...

```

- 2.5. 암호 `redhat`을 사용하여 `leader` 사용자로 로그인하여 프로젝트를 만듭니다. 프로젝트 생성이 실패해야 합니다.

```
[student@workstation ~]$ oc login -u leader -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

- ▶ 3. 프로젝트를 생성하고 `leader` 사용자에 프로젝트 관리 권한을 추가합니다.

- 3.1. `admin` 사용자로 로그인하여 `auth-rbac` 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project auth-rbac
Now using project "auth-rbac" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

3.2. **auth-rbac** 프로젝트의 **leader** 사용자에게 프로젝트 관리 권한을 부여합니다.

```
[student@workstation ~]$ oc policy add-role-to-user admin leader
clusterrole.rbac.authorization.k8s.io/admin added: "leader"
```

#### ▶ 4. **dev-group** 및 **qa-group** 그룹을 생성하고 각 멤버를 추가합니다.

4.1. **dev-group**이라는 그룹을 만듭니다.

```
[student@workstation ~]$ oc adm groups new dev-group
group.user.openshift.io/dev-group created
```

4.2. **developer** 사용자를 **dev-group**에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users dev-group developer
group.user.openshift.io/dev-group added: "developer"
```

4.3. **qa-group**이라는 두 번째 그룹을 만듭니다.

```
[student@workstation ~]$ oc adm groups new qa-group
group.user.openshift.io/qa-group created
```

4.4. **qa-engineer** 사용자를 **qa-group**에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users qa-group qa-engineer
group.user.openshift.io/qa-group added: "qa-engineer"
```

4.5. 기존의 모든 OpenShift 그룹을 검토하여 올바른 멤버가 있는지 확인합니다.

```
[student@workstation ~]$ oc get groups
NAME USERS
dev-group developer
qa-group qa-engineer
```

#### ▶ 5. **leader** 사용자로서 **dev-group**의 쓰기 권한과 **qa-group**의 읽기 권한을 **auth-rbac** 프로젝트에 할당합니다.

5.1. **leader** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u leader -p redhat
Login successful.
...output omitted...
Using project "auth-rbac".
```

- 5.2. auth-rbac 프로젝트에서 dev-group에 쓰기 권한을 추가합니다.

```
[student@workstation ~]$ oc policy add-role-to-group edit dev-group
clusterrole.rbac.authorization.k8s.io/edit added: "dev-group"
```

- 5.3. auth-rbac 프로젝트에서 qa-group에 읽기 권한을 추가합니다.

```
[student@workstation ~]$ oc policy add-role-to-group view qa-group
clusterrole.rbac.authorization.k8s.io/view added: "qa-group"
```

- 5.4. auth-rbac 프로젝트의 모든 역할 바인딩을 검토하여 올바른 그룹과 사용자에게 역할을 할당하는지 확인합니다. 다음 출력에서는 OpenShift에서 서비스 계정으로 할당한 기본 역할 바인딩을 생략합니다.

```
[student@workstation ~]$ oc get rolebindings -o wide
NAME AGE ROLE USERS GROUPS ...
admin 58s ClusterRole/admin admin
admin-0 51s ClusterRole/admin leader
edit 12s ClusterRole/edit dev-group
...output omitted...
view 8s ClusterRole/view qa-group
```

## ▶ 6. developer 사용자로 Apache HTTP Server를 배포하여 해당 프로젝트에서 developer 사용자에게 쓰기 권한이 있음을 증명합니다. qa-engineer 사용자에게 쓰기 권한도 부여하여 developer 사용자에게 프로젝트 관리 권한이 없음을 증명합니다.

- 6.1. developer 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
Using project "auth-rbac".
```

- 6.2. OpenShift의 표준 이미지 스트림을 사용하여 Apache HTTP Server를 배포합니다.

```
[student@workstation ~]$ oc new-app --name httpd httpd:2.4
...output omitted...
--> Creating resources ...
 imagestreamtag.image.openshift.io "httpd:2.4" created
 deployment.apps "httpd" created
 service "httpd" created
--> Success
...output omitted...
```

- 6.3. qa-engineer 사용자에 쓰기 권한을 부여합니다. 실패하게 됩니다.

```
[student@workstation ~]$ oc policy add-role-to-user edit qa-engineer
Error from server (Forbidden): rolebindings.rbac.authorization.k8s.io is
forbidden: User "developer" cannot list resource "rolebindings" in API group
"rbac.authorization.k8s.io" in the namespace "auth-rbac"
```

▶ 7. **qa-engineer** 사용자에게 **httpd** 애플리케이션에 대한 읽기 권한만 있는지 확인합니다.

7.1. **qa-engineer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u qa-engineer -p redhat
Login successful.
...output omitted...
Using project "auth-rbac".
```

7.2. **httpd** 애플리케이션 확장을 시도합니다. 실패하게 됩니다.

```
[student@workstation ~]$ oc scale deployment httpd --replicas 3
Error from server (Forbidden): deployments.apps "httpd" is forbidden: User "qa-
engineer" cannot patch resource "deployments/scale" in API group "apps" in the
namespace "auth-rbac"
```

▶ 8. 프로젝트 생성 권한을 모든 사용자로 복원합니다.

8.1. **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

8.2. OpenShift 설치 프로그램에서 생성한 **self-provisioners** 클러스터 역할 바인딩을 다시 만들어 모든 사용자의 프로젝트 생성 권한을 복원합니다. 그룹을 찾을 수 없다는 경고는 무시해도 됩니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
> --rolebinding-name self-provisioners \
> self-provisioner system:authenticated:oauth
Warning: Group 'system:authenticated:oauth' not found
clusterrole.rbac.authorization.k8s.io/self-provisioner added:
"system:authenticated:oauth"
```

## 완료

**workstation** 시스템에서 **lab** 명령을 실행하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab auth-rbac finish
```

이것으로 섹션을 완료합니다.

## ▶ 랩

# 인증 및 권한 부여 구성

## 성능 체크리스트

이 랩에서는 HTPasswd Identity 프로바이더를 구성하고, 그룹을 생성하고, 사용자 및 그룹에 풀을 할당합니다.

## 결과

다음을 수행할 수 있습니다.

- HTPasswd 인증을 위한 사용자 및 암호를 생성합니다.
- HTPasswd 인증에 사용할 ID 프로바이더를 구성합니다.
- 사용자에게 클러스터 관리 권한을 할당합니다.
- 클러스터 수준에서 프로젝트를 생성하는 기능을 제거합니다.
- 그룹을 생성하고 그룹에 사용자를 추가합니다.
- 그룹에 권한을 부여하여 프로젝트에서 사용자 권한을 관리합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

```
[student@workstation ~]$ lab auth-review start
```

이 명령을 사용하면 클러스터 API에 연결할 수 있고 **httpd-util** 패키지가 설치되며 인증 설정이 설치 기본값으로 구성됩니다.

1. 기존의 **~/D0280/labs/auth-review/tmp\_users** HTPasswd 인증 파일을 업데이트하여 **analyst** 사용자를 제거합니다. 파일의 **tester** 및 **leader** 사용자가 암호 **L@bR3v!ew**를 사용하는지 확인합니다. 사용자 **admin**과 **developer**를 위해 파일에 두 개의 새 항목을 추가합니다. 각 새 사용자의 암호로 **L@bR3v!ew**를 사용합니다.
2. **/usr/local/etc/ocp4.config** 파일에 암호로 정의된 **RHT\_OCP4\_KUBEADM\_PASSWD** 변수를 사용하여 OpenShift 클러스터에 **kubeadmin** 사용자로 로그인합니다. **~/D0280/labs/auth-review/tmp\_users** 파일에 정의된 사용자 이름과 암호를 사용하여 HTPasswd Identity 프로바이더를 사용하도록 클러스터를 구성합니다.
3. **admin** 사용자를 클러스터 관리자로 설정합니다. **admin** 및 **developer** 둘 다로 로그인하여 HTPasswd 사용자 구성과 클러스터 권한을 확인합니다.
4. **admin** 사용자로서 프로젝트 클러스터 전체를 생성하는 기능을 제거합니다.
5. **managers**라는 그룹을 만들고 **leader** 사용자를 그룹에 추가합니다. **managers** 그룹에 프로젝트 생성 권한을 부여합니다. **leader** 사용자로 **auth-review** 프로젝트를 생성합니다.
6. **developers**라는 그룹을 생성하고 **auth-review** 프로젝트에 대한 편집 권한을 부여합니다. **developer** 사용자를 그룹에 추가합니다.

7. **qa**라는 그룹을 생성하고 **auth-review** 프로젝트에 대한 보기 권한을 부여합니다. **tester** 사용자를 그룹에 추가합니다.

## 평가

**workstation** 시스템에서 **lab** 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab auth-review grade
```

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab auth-review finish
```

이로써 안내에 따른 연습이 완료됩니다.

## ▶ 솔루션

# 인증 및 권한 부여 구성

### 성능 체크리스트

이 랩에서는 HTPasswd Identity 프로바이더를 구성하고, 그룹을 생성하고, 사용자 및 그룹에 풀을 할당합니다.

### 결과

다음을 수행할 수 있습니다.

- HTPasswd 인증을 위한 사용자 및 암호를 생성합니다.
- HTPasswd 인증에 사용할 ID 프로바이더를 구성합니다.
- 사용자에게 클러스터 관리 권한을 할당합니다.
- 클러스터 수준에서 프로젝트를 생성하는 기능을 제거합니다.
- 그룹을 생성하고 그룹에 사용자를 추가합니다.
- 그룹에 권한을 부여하여 프로젝트에서 사용자 권한을 관리합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

```
[student@workstation ~]$ lab auth-review start
```

이 명령을 사용하면 클러스터 API에 연결할 수 있고 **httpd-util** 패키지가 설치되며 인증 설정이 설치 기본값으로 구성됩니다.

1. 기존의 **~/D0280/labs/auth-review/tmp\_users** HTPasswd 인증 파일을 업데이트하여 **analyst** 사용자를 제거합니다. 파일의 **tester** 및 **leader** 사용자가 암호 **L@bR3v!ew**를 사용하는지 확인합니다. 사용자 **admin**과 **developer**를 위해 파일에 두 개의 새 항목을 추가합니다. 각 새 사용자의 암호로 **L@bR3v!ew**를 사용합니다.
  - 1.1. **~/D0280/labs/auth-review/tmp\_users** HTPasswd 인증 파일에서 **analyst** 사용자를 제거합니다.

```
[student@workstation ~]$ htpasswd -D ~/D0280/labs/auth-review/tmp_users analyst
Deleting password for user analyst
```

- 1.2. 암호 **L@bR3v!ew**를 사용하도록 **tester** 및 **leader** 사용자 항목을 업데이트합니다. 암호 **L@bR3v!ew**를 사용하여 **admin** 및 **developer** 사용자 항목을 추가합니다.

```
[student@workstation ~]$ for NAME in tester leader admin developer
> do
> htpasswd -b ~/D0280/labs/auth-review/tmp_users ${NAME} 'L@bR3v!ew'
> done
Updating password for user tester
Updating password for user leader
Adding password for user admin
Adding password for user developer
```

- 1.3. `~/D0280/labs/auth-review/tmp_users` 파일의 내용을 검토합니다. 여기에는 `analyst` 사용자에 대한 행이 포함되어 있지 않습니다. `admin` 및 `developer` 사용자의 해시된 암호가 있는 두 개의 새 항목이 포함되어 있습니다.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-review/tmp_users
tester:$apr1$0eqhKgbU$DwD0CB4IumhasaRuEr6hp0
leader:$apr1$.EB5IXlu$FDV.Av16nj10CMzgolScr
admin:$apr1$ItcCncDS$xFQCUjQGTsXAup00KQfmw0
developer:$apr1$D8F1Hren$izDhAwq5DRjUHPv0i7FHn.
```

2. `/usr/local/etc/ocp4.config` 파일에 암호로 정의된 `RHT_OCP4_KUBEADM_PASSWD` 변수를 사용하여 OpenShift 클러스터에 `kubeadmin` 사용자로 로그인합니다. `~/D0280/labs/auth-review/tmp_users` 파일에 정의된 사용자 이름과 암호를 사용하여 HTPasswd Identity 프로바이더를 사용하도록 클러스터를 구성합니다.

- 2.1. `/usr/local/etc/ocp4.config`에서 액세스할 수 있는 강의실 구성 파일을 찾습니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 2.2. `kubeadmin` 사용자로 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2.3. `~/D0280/labs/auth-review/tmp_users` 파일을 사용하여 `auth-review`라는 시크릿을 생성합니다.

```
[student@workstation ~]$ oc create secret generic auth-review \
> --from-file htpasswd=/home/student/D0280/labs/auth-review/tmp_users \
> -n openshift-config
secret/auth-review created
```

- 2.4. 기존 OAuth 리소스를 `~/D0280/labs/auth-review/oauth.yaml`로 내보냅니다.

```
[student@workstation ~]$ oc get oauth cluster \
> -o yaml > ~/D0280/labs/auth-review/oauth.yaml
```

- 2.5. `~/D0280/labs/auth-review/oauth.yaml` 파일을 편집하여 `spec: {}` 행을 다음의 굵은 글꼴 행으로 교체합니다. `htpasswd`, `mappingMethod`, `name` 및 `type`은 동일한 들여쓰기 수준에 있습니다.

```
apiVersion: config.openshift.io/v1
kind: OAuth
...output omitted...
spec:
 identityProviders:
 - htpasswd:
 fileData:
 name: auth-review
 mappingMethod: claim
 name: htpasswd
 type: HTPasswd
```



### 참고

`~/D0280/solutions/auth-review/oauth.yaml` 파일에는 편의를 위해 지정된 사용자 지정과 함께 최소 버전의 OAuth 구성이 포함됩니다.

- 2.6. 이전 단계에서 정의한 사용자 지정 리소스를 적용합니다.

```
[student@workstation ~]$ oc replace -f ~/D0280/labs/auth-review/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

- 2.7. `oauth/cluster` 리소스가 업데이트되면 `openshift-authentication` 네임스페이스에 `oauth-openshift` 포드가 다시 생성됩니다.

```
[student@workstation ~]$ watch oc get pods -n openshift-authentication
```

새 `oauth-openshift` 포드가 준비되어 실행되고 이전 포드가 종료될 때까지 기다립니다.

NAME	READY	STATUS	RESTARTS	AGE
oauth-openshift-6755d8795-h8bgv	1/1	Running	0	34s
oauth-openshift-6755d8795-rk4m6	1/1	Running	0	38s

`Ctrl+C`를 눌러 `watch` 명령을 종료합니다.

3. `admin` 사용자를 클러스터 관리자로 설정합니다. `admin` 및 `developer` 둘 다로 로그인하여 `HTPasswd` 사용자 구성과 클러스터 권한을 확인합니다.

- 3.1. `admin` 사용자에게 `cluster-admin` 역할을 할당합니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
> cluster-admin admin
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

- 3.2. `admin` 사용자로 클러스터에 로그인하여 `HTPasswd` 인증이 올바르게 구성되었는지 확인합니다.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

- 3.3. `oc get nodes` 명령을 사용하여 `admin` 사용자에 `cluster-admin` 역할이 있는지 확인합니다. 클러스터의 노드 이름은 다를 수 있습니다.

```
[student@workstation ~]$ oc get nodes
NAME STATUS ROLES AGE VERSION
master01 Ready master,worker 46d v1.18.3+012b3ec
master02 Ready master,worker 46d v1.18.3+012b3ec
master03 Ready master,worker 46d v1.18.3+012b3ec
```

- 3.4. `developer` 사용자로 클러스터에 로그인하여 HTPasswd 인증이 올바르게 구성되었는지 확인합니다.

```
[student@workstation ~]$ oc login -u developer -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

- 3.5. `oc get node` 명령을 사용하여 `developer` 사용자에게 클러스터 관리 권한이 없는지 확인합니다.

```
[student@workstation ~]$ oc get nodes
Error from server (Forbidden): nodes is forbidden: User "developer" cannot list
resource "nodes" in API group "" at the cluster scope
```

#### 4. `admin` 사용자로서 프로젝트 클러스터 전체를 생성하는 기능을 제거합니다.

- 4.1. `admin` 사용자로 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

- 4.2. `system:authenticated:oauth` 가상 그룹에서 `self-provisioner` 클러스터 역할을 제거합니다.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
> self-provisioner system:authenticated:oauth
Warning: Your changes may get lost whenever a master is restarted,
unless you prevent reconciliation of this rolebinding using the
following command: oc annotate clusterrolebinding.rbac self-provisioner
'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
"system:authenticated:oauth"
```

**참고**

변경 사항이 유실된다는 경고는 무시해도 됩니다.

5. **managers**라는 그룹을 만들고 **leader** 사용자를 그룹에 추가합니다. **managers** 그룹에 프로젝트 생성 권한을 부여합니다. **leader** 사용자로 **auth-review** 프로젝트를 생성합니다.

- 5.1. **managers**라는 그룹을 만듭니다.

```
[student@workstation ~]$ oc adm groups new managers
group.user.openshift.io/managers created
```

- 5.2. **leader** 사용자를 **managers** 그룹에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users managers leader
group.user.openshift.io/managers added: "leader"
```

- 5.3. **self-provisioner** 클러스터 역할을 **managers** 그룹에 할당합니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
> self-provisioner managers
clusterrole.rbac.authorization.k8s.io/self-provisioner added: "managers"
```

- 5.4. **leader** 사용자로 **auth-review** 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc login -u leader -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

프로젝트를 생성하는 사용자에게는 프로젝트에 대한 **admin** 역할이 자동으로 할당됩니다.

```
[student@workstation ~]$ oc new-project auth-review
Now using project "auth-review" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

6. **developers**라는 그룹을 생성하고 **auth-review** 프로젝트에 대한 편집 권한을 부여합니다. **developer** 사용자를 그룹에 추가합니다.

- 6.1. **admin** 사용자로 클러스터에 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

- 6.2. **developers**라는 그룹을 만듭니다.

```
[student@workstation ~]$ oc adm groups new developers
group.user.openshift.io/developers created
```

6.3. developer 사용자를 developers 그룹에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users developers developer
group.user.openshift.io/developers added: "developer"
```

6.4. auth-review 프로젝트에서 developers 그룹에 편집 권한을 부여합니다.

```
[student@workstation ~]$ oc policy add-role-to-group edit developers
clusterrole.rbac.authorization.k8s.io/edit added: "developers"
```

7. qa라는 그룹을 생성하고 auth-review 프로젝트에 대한 보기 권한을 부여합니다. tester 사용자를 그룹에 추가합니다.

7.1. qa라는 그룹을 생성합니다.

```
[student@workstation ~]$ oc adm groups new qa
group.user.openshift.io/qa created
```

7.2. tester 사용자를 qa 그룹에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users qa tester
group.user.openshift.io/qa added: "tester"
```

7.3. auth-review 프로젝트에서 qa 그룹에 보기 권한을 부여합니다.

```
[student@workstation ~]$ oc policy add-role-to-group view qa
clusterrole.rbac.authorization.k8s.io/view added: "qa"
```

## 평가

workstation 시스템에서 lab 명령을 실행하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 스크립트를 다시 실행합니다.

```
[student@workstation ~]$ lab auth-review grade
```

## 완료

workstation 시스템에서 lab 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab auth-review finish
```

이로써 안내에 따른 연습이 완료됩니다.

## 요약

---

이 장에서 학습한 내용:

- 방금 설치한 OpenShift 클러스터에서는 관리자 액세스 권한을 부여하는 인증 방법으로 **kubeconfig** 파일과 **kubeadm** 가상 사용자를 제공합니다.
- HTPasswd ID 프로바이더는 시크릿에 저장된 자격 증명과 비교하여 사용자를 인증합니다. 시크릿의 이름과 ID 프로바이더의 기타 설정은 OAuth 사용자 지정 리소스에 저장됩니다.
- HTPasswd ID 프로바이더를 사용하여 사용자 자격 증명을 관리하려면 시크릿에서 데이터를 추출하고 **htpasswd** 명령을 사용하여 데이터를 변경한 다음 데이터를 다시 시크릿에 적용해야 합니다.
- OpenShift 사용자를 생성하려면 ID 프로바이더가 관리하는 유효한 자격 증명과 사용자 및 ID 리소스가 필요합니다.
- OpenShift 사용자를 삭제하려면 ID 프로바이더에서 자격 증명을 삭제하고 사용자 및 ID 리소스를 삭제해야 합니다.
- OpenShift에서는 역할 기반 액세스 제어(RBAC)를 사용하여 사용자 작업을 제어합니다. 역할은 OpenShift 리소스와의 상호 작용을 제어하는 규칙의 컬렉션입니다. 클러스터 관리자, 개발자 및 감사자의 기본 역할이 있습니다.
- 사용자 상호 작용을 제어하려면 사용자를 하나 이상의 역할에 할당합니다. 역할 바인딩에는 모든 역할의 연결이 사용자 및 그룹에 포함됩니다.
- 사용자에게 클러스터 관리자 권한을 부여하려면 **cluster-admin** 역할을 해당 사용자에게 할당합니다.



## 11장

# 애플리케이션 보안 구성

### 목적

보안 컨텍스트 제약 조건을 사용하여 애플리케이션 권한을 제한하고 시크릿을 사용하여 액세스 자격 증명을 보호합니다.

### 목표

- 중요한 정보를 관리하기 위한 시크릿을 생성하고 적용합니다.
- 서비스 계정을 생성하고 권한을 적용합니다.

### 섹션

- 시크릿을 사용하여 중요 정보 관리(안내에 따른 연습)
- 보안 컨텍스트 제약 조건으로 애플리케이션 권한 제어(안내에 따른 연습)

### 랩

애플리케이션 보안 구성

# 시크릿을 사용하여 중요 정보 관리

## 목표

이 섹션을 마치면 다음 작업을 할 수 있습니다.

- 중요한 정보를 관리하기 위한 시크릿을 생성하고 적용합니다.
- 애플리케이션 간에 시크릿을 공유합니다.

## 시크릿 개요

최신 애플리케이션은 코드, 구성 및 데이터를 느슨하게 결합하도록 설계되었습니다. 구성 파일 및 데이터는 소프트웨어의 일부로 하드 코딩되지 않습니다. 대신, 소프트웨어는 외부 소스에서 구성 및 데이터를 로드합니다. 그러면 애플리케이션 소스 코드를 변경하지 않고도 다양한 환경에 애플리케이션을 배포할 수 있습니다.

애플리케이션에서 중요한 정보에 액세스해야 하는 경우가 종종 있습니다. 예를 들어 백엔드 웹 애플리케이션에서는 데이터베이스 쿼리를 수행하기 위해 데이터베이스 자격 증명에 액세스해야 합니다.

Kubernetes 및 OpenShift에서는 시크릿 리소스를 사용하여 다음과 같은 중요한 정보를 보유합니다.

- 암호
- 중요한 구성 파일
- SSH 키 또는 OAuth 토큰과 같은 외부 리소스의 자격 증명

시크릿에서는 모든 유형의 데이터를 저장할 수 있습니다. 시크릿의 데이터는 Base64로 인코딩되며 일반 텍스트로 저장되지 않습니다. 시크릿 데이터는 암호화되지 않습니다. Base64 형식의 시크릿을 디코딩하여 원본 데이터에 액세스할 수 있습니다.

시크릿에서 모든 유형의 데이터를 저장할 수 있지만, Kubernetes와 OpenShift에서는 다양한 유형의 시크릿을 지원합니다. 서비스 계정 토큰, SSH 키 및 TLS 인증서를 포함한 다양한 유형의 시크릿 리소스가 있습니다. 특정 시크릿 리소스 유형에 정보를 저장하는 경우 Kubernetes에서는 데이터가 시크릿 유형을 따르는지 확인합니다.



### 참고

기본 설정은 아니지만 Etcd 데이터베이스를 암호화할 수 있습니다. 이 기능을 사용하면 Etcd에서 시크릿, 구성 맵, 경로, OAuth 액세스 토큰, OAuth 권한 부여 토큰 등의 리소스를 암호화합니다. Etcd 암호화를 사용하는 방법은 이 수업의 범위를 벗어납니다.

## 시크릿 기능

시크릿의 주요 기능은 다음과 같습니다.

- 시크릿 데이터는 프로젝트 네임스페이스 내에서 공유할 수 있습니다.
- 시크릿 데이터는 시크릿 정의와 독립적으로 참조됩니다. 관리자는 시크릿 리소스를 생성하고 관리할 수 있으며, 다른 팀 멤버는 배포 구성에서 해당 리소스를 참조할 수 있습니다.
- OpenShift에서 포드를 생성하면 시크릿 데이터가 포드에 삽입됩니다. 시크릿을 포드에 환경 변수 또는 마운트된 파일로 노출할 수 있습니다.

- 포드 실행 중 시크릿 값이 변경되면 시크릿 데이터가 포드에서 업데이트되지 않습니다. 시크릿 값이 변경되면 새 시크릿 데이터를 삽입하기 위해 새 포드를 생성해야 합니다.
- OpenShift에서 포드에 삽입하는 모든 시크릿 데이터는 임시입니다. OpenShift에서 중요한 데이터를 환경 변수로 포드에 노출하면 포드를 삭제할 때 해당 변수가 삭제됩니다.  
시크릿 데이터 볼륨은 임시 파일 스토리지에서 지원합니다. 시크릿을 포드에 파일로 마운트하면 포드를 삭제할 때 파일도 삭제됩니다. 중지된 포드에는 시크릿 데이터가 포함되지 않습니다.

## 시크릿 사용 사례

시크릿은 자격 증명을 저장하고 서비스 간의 통신을 보호하는 두 가지 용도로 주로 사용됩니다.

### Credentials(자격 증명)

암호 및 사용자 이름과 같은 중요한 정보를 시크릿에 저장합니다.

애플리케이션에서 파일의 중요한 정보를 읽어야 하는 경우 시크릿을 데이터 볼륨으로 포드에 마운트합니다. 이 애플리케이션에서는 시크릿을 일반 파일로 읽어 중요한 정보에 액세스할 수 있습니다. 예를 들어, 일부 데이터베이스에서는 사용자를 인증하기 위해 파일에서 자격 증명을 읽습니다.

일부 애플리케이션에서는 환경 변수를 사용하여 구성 및 중요한 데이터를 읽습니다. 배포 구성에서 시크릿 변수를 포드 환경 변수에 연결할 수 있습니다.

### 전송 계층 보안(TLS) 및 키 쌍

TLS 인증서 및 키를 사용하여 포드에 대한 통신을 보호합니다. TLS 시크릿은 인증서를 `tls.crt`로 저장하고 인증서 키는 `tls.key`로 저장합니다. 개발자는 시크릿을 볼륨으로 마운트하고 애플리케이션 통과 경로를 생성할 수 있습니다.

## 시크릿 생성

포드에서 중요한 정보에 액세스해야 하면 포드를 배포하기 전에 정보의 시크릿을 생성합니다.

- 명령줄에 입력한 문자 값에서 키-값 쌍을 포함하는 일반 시크릿을 생성합니다.

```
[user@demo ~]$ oc create secret generic secret_name \
> --from-literal key1=secret1 \
> --from-literal key2=secret2
```

- 명령줄에 지정된 키 이름과 파일의 값을 사용하여 일반 시크릿을 생성합니다.

```
[user@demo ~]$ oc create secret generic ssh-keys \
> --from-file id_rsa=/path-to/id_rsa \
> --from-file id_rsa.pub=/path-to/id_rsa.pub
```

- 인증서 및 관련 키를 지정하는 TLS 시크릿을 생성합니다.

```
[user@demo ~]$ oc create secret tls secret-tls \
> --cert /path-to-certificate --key /path-to-key
```

## 포드에 시크릿 노출

포드에 시크릿을 노출하려면 먼저 시크릿을 생성합니다. 중요한 데이터의 각 부분을 키에 할당합니다. 생성하고 나면 시크릿에 키-값 쌍이 포함됩니다.

다음 명령에서는 값이 `demo-user`인 `user`와 값이 `zT1KTgk`인 `root_password` 키를 사용하여 `demo-secret`이라는 일반 시크릿을 생성합니다.

```
[user@demo ~]$ oc create secret generic demo-secret \
> --from-literal user=demo-user
> --from-literal root_password=zT1KTgk
```

## 포드 환경 변수로서의 시크릿

`MYSQL_ROOT_PASSWORD` 환경 변수에서 데이터베이스 관리자 암호를 읽는 데이터베이스 애플리케이션을 고려해 보십시오. 시크릿의 값을 사용하도록 배포 구성의 환경 변수 섹션을 수정합니다.

```
env:
- name: MYSQL_ROOT_PASSWORD ❶
 valueFrom:
 secretKeyRef: ❷
 name: demo-secret ❸
 key: root_password ❹
```

- ❶ 시크릿의 데이터를 포함하는 포드의 환경 변수 이름입니다.
- ❷ `secretKeyRef` 키에는 시크릿이 필요합니다. 구성 맵에 `configMapKeyRef` 키를 사용합니다.
- ❸ 원하는 중요한 정보를 포함하는 시크릿의 이름입니다.
- ❹ 시크릿의 중요 정보가 포함된 키의 이름입니다.

`oc set env` 명령을 사용하여 시크릿 또는 구성 맵에서 애플리케이션 환경 변수를 설정할 수도 있습니다. 경우에 따라 `--prefix` 옵션을 사용하여 환경 변수의 이름과 일치하도록 키 이름을 수정할 수 있습니다. 다음 예제에서 `user` 키는 `MYSQL_USER` 환경 변수를 설정하고, `root_password` 키는 `MYSQL_ROOT_PASSWORD` 환경 변수를 설정합니다. 키 이름이 소문자이면 해당 환경 변수는 대문자로 변환됩니다.

```
[user@demo ~]$ oc set env deployment/demo --from secret/demo-secret \
> --prefix MYSQL_
```

## 포드에 파일로 저장된 시크릿

시크릿은 포드 내에 있는 디렉터리에 마운트할 수 있습니다. 키 이름을 사용하여 시크릿의 각 키에 대한 파일이 생성됩니다. 각 파일의 내용은 디코딩된 시크릿 값입니다.

```
[user@demo ~]$ oc set volume deployment/demo \
> --add --type secret ❶ \
> --secret-name demo-secret ❷ \
> --mount-path /app-secrets ❸
```

- ❶ `demo` 배포에서 볼륨 구성을 수정합니다.
- ❷ 시크릿에서 새 볼륨을 추가합니다. 구성 맵을 볼륨으로 마운트할 수도 있습니다.
- ❸ `demo-secret` 시크릿을 사용합니다.
- ❹ 포드의 `/app-secrets` 디렉터리에서 시크릿 데이터를 사용 가능하게 만듭니다. `/app-secrets/user` 파일의 내용은 `demo-user`입니다. `/app-secrets/root_password` 파일의 내용은 `zT1KTgk`입니다.

컨테이너 이미지는 마운트 지점의 위치와 필요한 파일 이름을 지시할 수 있습니다. 예를 들어 **NGINX**를 실행하는 컨테이너 이미지는 **/etc/nginx/nginx.conf** 구성 파일에서 SSL 인증서 위치 및 SSL 인증서 키 위치를 지정할 수 있습니다. 필요한 파일이 없으면 컨테이너가 실행되지 않을 수 있습니다.



### 중요

마운트 지점이 이미 포드에 있는 경우 마운트 지점에 있는 기존 파일이 마운트된 시크릿에 의해 가려집니다. 기존 파일이 보이지 않으면 액세스할 수 없습니다.

## 구성 맵 개요

구성 맵은 시크릿과 유사하게 컨테이너 이미지에서 구성 정보를 분리합니다. 시크릿과 달리 구성 맵에 포함된 정보는 보호하지 않아도 됩니다. 구성 맵의 데이터를 사용하여 컨테이너 이미지에서 환경 변수를 설정하거나 구성 맵을 컨테이너 이미지 내에 볼륨으로 마운트할 수 있습니다.

시크릿 또는 구성 맵이 변경되면 컨테이너 이미지를 다시 빌드할 필요가 없습니다. 새 포드에서는 업데이트된 시크릿 및 구성 맵을 사용합니다. 이전 시크릿 및 구성 맵을 사용하여 포드를 삭제할 수 있습니다.

구성 맵을 생성하는 구문은 시크릿을 생성하는 구문과 거의 일치합니다. 키-값 쌍을 명령줄에 입력하거나 파일 내용을 지정된 키의 값으로 사용할 수 있습니다.

```
[user@demo ~]$ oc create configmap my-config \
> --from-literal key1=config1 --from-literal key2=config2
```

## 시크릿 및 구성 맵 업데이트

시크릿 및 구성 맵은 종종 업데이트해야 합니다. **oc extract** 명령을 사용하여 보유한 데이터가 최신 상태인지 확인합니다. **--to** 옵션을 사용하여 데이터를 특정 디렉터리에 저장합니다. 시크릿 또는 구성 맵의 각 키에서는 키와 이름이 같은 파일을 생성합니다. 각 파일의 내용은 연결된 키의 값입니다. **oc extract** 명령을 두 번 이상 실행하는 경우 **--confirm** 옵션을 사용하여 기존 파일을 덮어씁니다.

```
[user@demo ~]$ oc extract secret/htpasswd-ppk1q -n openshift-config \
> --to /tmp/ --confirm
```

로컬로 저장된 파일을 업데이트한 후 **oc set data** 명령을 사용하여 시크릿 또는 구성 맵을 업데이트합니다. 업데이트가 필요한 각 키에 대해 키 이름과 관련 값을 지정합니다. 파일에 값이 포함된 경우 **--from-file** 옵션을 사용합니다.

이전의 **oc extract** 예에서 **htpasswd-ppk1q** 시크릿에는 **htpasswd**라는 키 하나만 포함되어 있습니다. **oc set data** 명령을 사용하면 **--from-file htpasswd=/tmp/htpasswd**를 사용하여 **htpasswd** 키 이름을 명시적으로 지정할 수 있습니다. 키 이름을 지정하지 않으면 파일 이름이 키 이름으로 사용됩니다.

```
[user@demo ~]$ oc set data secret/htpasswd-ppk1q -n openshift-config \
> --from-file /tmp/htpasswd
```



### 참조

시크릿에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-pods-secrets-about\\_nodes-pods-secrets](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-pods-secrets-about_nodes-pods-secrets)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서의 포드 작업 장에서 시크릿 이해 섹션을 참조하십시오.

Etcd 암호화에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 보안 설명서의 Etcd 데이터 암호화장을 참조하십시오.

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/security/index#encrypting-etcd](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/security/index#encrypting-etcd)

## ▶ 연습 가이드

# 시크릿을 사용하여 중요 정보 관리

이 연습에서는 시크릿을 사용하여 정보를 관리합니다.

### 결과

다음을 수행할 수 있습니다.

- 시크릿을 관리하고 사용하여 애플리케이션에서 환경 변수를 초기화합니다.
- MySQL 데이터베이스 애플리케이션의 시크릿을 사용합니다.
- MySQL 데이터베이스에 연결하는 애플리케이션에 시크릿을 할당합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 사용하면 클러스터 API에 연결할 수 있으며, 이 연습에 필요한 리소스 파일을 다운로드합니다.

```
[student@workstation ~]$ lab authorization-secrets start
```

- ▶ 1. OpenShift 클러스터에 로그인하고 **authorization-secrets** 프로젝트를 만듭니다.

- 1.1. 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **authorization-secrets** 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project authorization-secrets
Now using project "authorization-secrets" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

- ▶ 2. MySQL 데이터베이스에 액세스하기 위해 자격 증명 및 연결 정보를 사용하여 시크릿을 생성합니다.

```
[student@workstation ~]$ oc create secret generic mysql \
> --from-literal user=myuser --from-literal password=redhat123 \
> --from-literal database=test_secrets --from-literal hostname=mysql
secret/mysql created
```

▶ 3. 데이터베이스를 배포하고 사용자 및 데이터베이스 구성의 시크릿을 추가합니다.

- 3.1. 임시 데이터베이스 서버를 배포해 봅니다. MySQL 이미지에 초기 구성에 대한 환경 변수가 필요하기 때문에 이 작업에 실패합니다. 이러한 변수의 값은 `oc new-app` 명령을 사용하여 시크릿에서 할당할 수 없습니다.

```
[student@workstation ~]$ oc new-app --name mysql \
> --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
--> Creating resources ...
 imagestream.image.openshift.io "mysql" created
 deployment.apps "mysql" created
 service "mysql" created
--> Success
...output omitted...
```

- 3.2. 배포 상태를 실시간으로 검색하려면 `-w` 옵션을 사용하여 `oc get pods` 명령을 실행합니다. 데이터베이스 포드가 실패 상태에 있는지 확인합니다. **Ctrl+C**를 눌러 명령을 종료합니다.

```
[student@workstation ~]$ oc get pods -w
NAME READY STATUS RESTARTS AGE
mysql-786bb947f9-qz2fm 0/1 Error 3 71s
mysql-786bb947f9-qz2fm 0/1 CrashLoopBackOff 3 75s
mysql-786bb947f9-qz2fm 0/1 Error 4 103s
mysql-786bb947f9-qz2fm 0/1 CrashLoopBackOff 4 113s
```



### 참고

포드에서 오류 상태에 도달하는 데 시간이 걸릴 수 있습니다.

- 3.3. `mysql` 시크릿을 사용하여 `mysql` 배포에서 환경 변수를 초기화합니다. 초기화에 성공하면 배포에 `MYSQL_USER`, `MYSQL_PASSWORD` 및 `MYSQL_DATABASE` 환경 변수가 필요합니다. 시크릿에는 `MYSQL_` 접두사를 추가하여 환경 변수로 배포에 할당할 수 있는 `user`, `password`, `database` 등의 키가 있습니다.

```
[student@workstation ~]$ oc set env deployment/mysql --from secret/mysql \
> --prefix MYSQL_
deployment.apps/mysql updated
```

- 3.4. 시크릿을 볼륨으로 마운트하는 방법을 시연하기 위해 포드 내의 `/run/kubernetes/mysql` 디렉터리에 `mysql` 시크릿을 마운트합니다.

```
[student@workstation ~]$ oc set volume deployment/mysql --add --type secret \
> --mount-path /run/secrets/mysql --secret-name mysql
info: Generated volume name: volume-nrh7r
deployment.apps/mysql volume updated
```

- 3.5. `oc set env` 명령 또는 `oc set volume` 명령을 사용하여 배포를 수정하면 새 애플리케이션 배포가 트리거됩니다. 수정 후 `mysql` 애플리케이션이 성공적으로 배포되는지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
mysql-7cd7499d66-gm2rh 1/1 Running 0 21s
```

**Running** 상태의 포드 이름을 기록합니다. 다음 단계에서 필요합니다.

- ▶ 4. 데이터베이스가 이제 **mysql** 시크릿에서 초기화된 환경 변수를 사용하여 인증되는지 확인합니다.

- 4.1. **Running** 상태에 있는 **mysql** 포드의 원격 쉘 세션을 엽니다.

```
[student@workstation ~]$ oc rsh mysql-7cd7499d66-gm2rh
sh-4.2$
```

- 4.2. MySQL 세션을 시작하여 **mysql** 시크릿에서 초기화한 환경 변수를 사용하여 **mysql** 애플리케이션을 구성했는지 확인합니다.

```
sh-4.2$ mysql -u myuser --password=redhat123 test_secrets -e 'show databases;'
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database |
+-----+
| information_schema |
| test_secrets |
+-----+
```

- 4.3. **mysql** 패턴을 포함하는 포드의 마운트 지점을 나열합니다. 마운트 지점은 임시 파일 시스템 (**tmpfs**)에서 지원합니다. 이는 볼륨으로 마운트된 모든 시크릿에 적용됩니다.

```
sh-4.2$ df -h | grep mysql
tmpfs 7.9G 16K 7.9G 1% /run/secrets/mysql
```

- 4.4. **/run/secrets/mysql** 마운트 지점에 마운트된 파일을 검사합니다. 각 파일은 시크릿의 키 이름과 일치하며, 각 파일의 내용은 관련 키의 값입니다.

```
sh-4.2$ for FILE in $(ls /run/secrets/mysql)
> do
> echo "${FILE}: $(cat /run/secrets/mysql/${FILE})"
> done
database: test_secrets
hostname: mysql
password: redhat123
user: mysql
```

- 4.5. 원격 쉘 세션을 닫고 **workstation** 시스템에서 계속합니다.

```
sh-4.2$ exit
exit
[student@workstation ~]$
```

- ▶ 5. MySQL 데이터베이스를 사용하는 새 애플리케이션을 만듭니다.

- 5.1. Quay.io에서 `redhattraining/famous-quotes` 이미지를 사용하여 새 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc new-app --name quotes \
> --docker-image quay.io/redhattraining/famous-quotes:2.1
--> Found container image 7ff1a7b (2 days old) from quay.io for "quay.io/
redhattraining/famous-quotes:2.1"
...output omitted...
--> Creating resources ...
 imagestream.image.openshift.io "quotes" created
 deployment.apps "quotes" created
 service "quotes" created
--> Success
...output omitted...
```

- 5.2. `quotes` 애플리케이션 포드의 상태를 확인합니다. 포드를 데이터베이스에 연결할 수 없어 오류가 표시됩니다. 오류가 출력에 표시되는 데 시간이 걸릴 수 있습니다. **Ctrl+C**를 눌러 명령을 종료합니다.

```
[student@workstation ~]$ oc get pods -l deployment=quotes -w
NAME READY STATUS RESTARTS AGE
quotes-6b658d57bc-vs28q 0/1 CrashLoopBackOff 3 86s
quotes-6b658d57bc-vs28q 0/1 Error 4 108s
quotes-6b658d57bc-vs28q 0/1 CrashLoopBackOff 4 2m1s
```

- ▶ 6. `quotes` 애플리케이션에는 `QUOTES_USER`, `QUOTES_PASSWORD`, `QUOTES_DATABASE`, and `QUOTES_HOSTNAME` 환경 변수가 필요합니다. 이 변수는 `mysql` 시크릿의 `user`, `password`, `database` 및 `hostname` 키에 해당합니다. `mysql` 시크릿은 `QUOTES_` 접두사를 추가하여 `quotes` 애플리케이션의 환경 변수를 초기화할 수 있습니다.

- 6.1. `mysql` 시크릿을 사용하여 `quotes` 애플리케이션에서 데이터베이스에 연결하는 데 필요한 환경 변수를 초기화합니다.

```
[student@workstation ~]$ oc set env deployment/quotes --from secret/mysql \
> --prefix QUOTES_
deployment.apps/quotes updated
```

- 6.2. `quotes` 애플리케이션 포드가 다시 배포될 때까지 기다립니다. 이전 포드가 자동으로 종료됩니다.

```
[student@workstation ~]$ oc get pods -l deployment=quotes
NAME READY STATUS RESTARTS AGE
quotes-77df54758b-mqdtf 1/1 Running 3 7m17s
```



### 참고

포드에서 배포를 완료하는 데 시간이 걸릴 수 있습니다.

- ▶ 7. `quotes` 포드가 데이터베이스에 연결되고 애플리케이션에 `quotes` 목록이 표시되는지 확인합니다.

## 11장 | 애플리케이션 보안 구성

- 7.1. `oc logs` 명령을 사용하여 포드 로그를 검사합니다. 로그에 데이터베이스가 연결되었음이 표시됩니다.

```
[student@workstation ~]$ oc logs quotes-77df54758b-mqdtf | head -n2
... Connecting to the database: myuser:redhat123@tcp(mysql:3306)/test_secrets
... Database connection OK
```

- 7.2. 클러스터 외부에서 액세스할 수 있도록 `quotes` 서비스를 노출합니다.

```
[student@workstation ~]$ oc expose service quotes \
> --hostname quotes.apps.ocp4.example.com
route.route.openshift.io/quotes exposed
```

- 7.3. 애플리케이션 호스트 이름을 확인합니다.

```
[student@workstation ~]$ oc get route quotes
NAME HOST/PORT PATH SERVICES PORT ...
quotes quotes.apps.ocp4.example.com quotes 8000-tcp ...
```

- 7.4. `env` REST API 진입점에 액세스하여 애플리케이션에 변수가 올바르게 설정되는지 확인합니다.

```
[student@workstation ~]$ curl -s \
> http://quotes.apps.ocp4.example.com/env | grep QUOTES_
QUOTES_USER: myuser
QUOTES_PASSWORD: redhat123
QUOTES_DATABASE: test_secrets
QUOTES_HOST: mysql
```

- 7.5. 애플리케이션 `status` REST API 진입점에 액세스하여 데이터베이스 연결을 테스트합니다.

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.example.com/status
Database connection OK
```

- 7.6. `random` REST API 진입점에 액세스하여 애플리케이션 기능을 테스트합니다.

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.example.com/random
8: Those who can imagine anything, can create the impossible.
- Alan Turing
```

## ▶ 8. authorization-secrets 프로젝트를 제거합니다.

```
[student@workstation ~]$ oc delete project authorization-secrets
project.project.openshift.io "authorization-secrets" deleted
```

## 완료

`workstation` 시스템에서 `lab` 명령을 실행하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab authorization-secrets finish
```

이것으로 섹션을 완료합니다.

# 보안 컨텍스트 제약 조건으로 애플리케이션 권한 제어

---

## 목표

이 섹션을 마치면 다음 작업을 할 수 있습니다.

- 서비스 계정을 생성하고 권한을 적용합니다.
- 보안 컨텍스트 제약 조건을 관리합니다.

## 보안 컨텍스트 제약 조건(SCC)

Red Hat OpenShift에서는 리소스에 대한 액세스는 제한하지만 OpenShift 내 작업에 대한 액세스는 제한하지 않는 보안 메커니즘이 보안 컨텍스트 제약 조건(SCC)을 제공합니다.

SCC는 OpenShift에서 실행되고 있는 포드에서 호스트 환경으로의 액세스를 제한합니다. SCC 제어:

- 권한 있는 컨테이너 실행
- 컨테이너에 추가 기능 요청
- 호스트 딜렉터리를 볼륨으로 사용
- 컨테이너의 SELinux 컨텍스트 변경
- 사용자 ID 변경

커뮤니티에서 개발한 일부 컨테이너에는 기본적으로 허용되지 않은 리소스(예: 파일 시스템, 소켓) 또는 SELinux 컨텍스트에 액세스할 수 있도록 완화된 보안 컨텍스트 제약 조건이 필요할 수 있습니다.

클러스터 관리자로 다음 명령을 실행하여 OpenShift에서 정의한 SCC를 나열할 수 있습니다.

```
[user@demo ~]$ oc get scc
```

OpenShift에서는 8개의 SCC를 제공합니다.

- `anyuid`
- `hostaccess`
- `hostmount-anyuid`
- `hostnetwork`
- `node-exporter`
- `nonroot`
- `privileged`
- `restricted`

SCC에 관한 자세한 정보를 보려면 `oc describe` 명령을 사용합니다.

```
[user@demo ~]$ oc describe scc anyuid
Name: anyuid
Priority: 10
Access:
 Users: <none>
 Groups: system:cluster-admins
Settings:
...output omitted...
```

OpenShift에서 생성한 대부분의 포드에서는 OpenShift 외부 리소스에 대해 제한된 액세스 권한을 제공하는 **restricted**라는 SCC를 사용합니다. **oc describe** 명령을 사용하여 포드에서 사용하는 보안 컨텍스트 제약 조건을 확인합니다.

```
[user@demo ~]$ oc describe pod console-5df4fcbb47-67c52 \
> -n openshift-console | grep scc
 openshift.io/scc: restricted
```

Docker Hub와 같은 공용 컨테이너 레지스트리에서 다운로드한 컨테이너 이미지는 **한정된 SCC**로 실행되지 않을 수 있습니다. 예를 들어, **한정된 SCC**는 임의의 사용자 ID를 사용하여 컨테이너를 실행하기 때문에 특정 사용자 ID로 실행해야 하는 컨테이너 이미지가 실패할 수 있습니다. 포트 80 또는 포트 443에서 수신 대기하는 컨테이너 이미지는 관련 이유로 실패할 수 있습니다. **한정된 SCC**에서 사용하는 임의의 사용자 ID로는 권한 있는 네트워크 포트(포트 번호가 1024 미만인 포트)에서 수신 대기하는 서비스를 시작할 수 없습니다. 컨테이너의 제한 사항을 극복할 수 있는 모든 보안 컨텍스트 제약 조건을 나열하려면 **scc-subject-review** 하위 명령을 사용하십시오.

```
[user@demo ~]$ oc get pod podname -o yaml | \
> oc adm policy scc-subject-review -f -
```

**anyuid** SCC는 **run as user** 전략이 **RunAsAny**로 정의됩니다. 이는 컨테이너에 있는 모든 사용자 ID로 포드를 실행할 수 있음을 나타냅니다. 이 전략을 사용하면 특정 사용자가 필요한 컨테이너에서 특정 사용자 ID를 사용하여 명령을 실행할 수 있습니다.

다른 SCC를 사용하여 실행하도록 컨테이너를 변경하려면 포드에 바인딩된 서비스 계정을 만들어야 합니다. 서비스 계정을 생성하려면 **oc create serviceaccount** 명령을 사용하고, 다른 네임스페이스에 서비스 계정을 생성해야 하는 경우에는 **-n** 옵션을 사용합니다.

```
[user@demo ~]$ oc create serviceaccount service-account-name
```

서비스 계정을 SCC와 연결하려면 **oc adm policy** 명령을 사용합니다. 서비스 계정을 확인하려면 **-z** 옵션을 사용하고, 서비스 계정이 다른 네임스페이스에 있는 경우 **-n** 옵션을 사용합니다.

```
[user@demo ~]$ oc adm policy add-scc-to-user SCC -z service-account
```



### 중요

SCC를 서비스 계정에 할당하거나 서비스 계정에서 SCC를 제거하는 작업은 클러스터 관리자가 수행해야 합니다. 포드에서 멀 한정적인 SCC를 사용하여 실행할 수 있는 경우 클러스터의 보안이 저하됩니다. 주의해서 사용하십시오.

`oc set serviceaccount` 명령을 사용하여 서비스 계정을 사용하도록 기존 배포 또는 배포 구성의 설정을 수정합니다. 명령이 성공적으로 실행되면 배포 또는 배포 구성과 연결된 포드가 다시 배포됩니다.

```
[user@demo ~]$ oc set serviceaccount deployment/deployment-name \
> service-account-name
```

## 권한 있는 컨테이너

일부 컨테이너는 호스트의 런타임 환경에 액세스해야 합니다. 예를 들어, S2I 빌더 컨테이너는 컨테이너라는 자체적인 한계를 초과하여 액세스해야 하는 권한 있는 컨테이너 클래스입니다. 이러한 컨테이너는 OpenShift 노드의 모든 리소스를 사용할 수 있기 때문에 보안 위험이 발생할 수 있습니다. SCC를 사용하여 권한 있는 액세스 권한으로 서비스 계정을 만들어 권한 있는 컨테이너에 대한 액세스를 활성화합니다.



### 참조

자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/authentication\\_and\\_authorization/index#managing-pod-security-policies](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#managing-pod-security-policies)

에 있는 Red Hat OpenShift Container Platform 4.5 권한 부여 및 인증 설명서의 보안 컨테스트 제약 조건 관리 장을 참조하십시오.

## ▶ 연습 가이드

# 보안 컨텍스트 제약 조건으로 애플리케이션 권한 제어

이 연습에서는 확장된 권한이 있는 포드가 필요한 애플리케이션을 배포합니다.

### 결과

다음을 수행할 수 있습니다.

- 서비스 계정을 생성하고 이 계정에 보안 컨텍스트 제약 조건(SCC)을 할당합니다.
- 서비스 계정을 배포 구성에 할당합니다.
- `root` 권한이 필요한 애플리케이션을 실행합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 실행하면 클러스터 API에 연결할 수 있으며 이 연습에 사용할 몇몇 HTPasswd 사용자를 생성합니다.

```
[student@workstation ~]$ lab authorization-scc start
```

#### ▶ 1. OpenShift 클러스터에 로그인하고 `authorization-scc` 프로젝트를 만듭니다.

##### 1.1. 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

##### 1.2. `authorization-scc` 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project authorization-scc
Now using project "authorization-scc" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

#### ▶ 2. `quay.io/redhattraining/gitlab-ce:8.4.3-ce.0`에 있는 컨테이너 이미지를 사용하여 `gitlab`이라는 애플리케이션을 배포합니다. 이 이미지는 `docker.io/gitlab/gitlab-ce:8.4.3-ce.0`에서 사용할 수 있는 컨테이너 이미지의 사본입니다. 컨테이너 이미지에 `root` 권한이 필요하기 때문에 포드가 실패하는지 확인합니다.

##### 2.1. `gitlab` 애플리케이션을 배포합니다.

```
[student@workstation ~]$ oc new-app --name gitlab \
> --docker-image quay.io/redhattraining/gitlab-ce:8.4.3-ce.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "gitlab" created
deployment.apps "gitlab" created
service "gitlab" created
--> Success
...output omitted...
```

- 2.2. 애플리케이션을 성공적으로 배포했는지 확인합니다. 이 이미지를 제대로 배포하려면 **root** 권한이 필요하므로 오류가 표시되어야 합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
gitlab-7d67db7875-gcsjl 0/1 Error 1 60s
```



### 참고

이미자가 **Error** 상태가 되는데 시간이 걸릴 수 있습니다. 포드의 상태를 확인하는 동안 **CrashLoopBackOff** 상태도 표시될 수 있습니다.

- 2.3. 애플리케이션 로그를 검토하여 권한이 충분하지 않기 때문에 오류가 발생했는지 확인합니다.

```
[student@workstation ~]$ oc logs pod/gitlab-7d67db7875-gcsjl
...output omitted...
=====
Recipe Compile Error in /opt/gitlab/embedded/cookbooks/cache/cookbooks/gitlab/
recipes/default.rb
=====

Chef::Exceptions::InsufficientPermissions

directory[/etc/gitlab] (gitlab::default line 26) had an error:
Chef::Exceptions::InsufficientPermissions: Cannot create directory[/etc/gitlab]
at /etc/gitlab due to insufficient permissions
...output omitted...
```

- 2.4. **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 2.5. 다른 SCC를 사용하여 권한 문제를 해결할 수 있는지 확인합니다.

```
[student@workstation ~]$ oc get pod/gitlab-7d67db7875-gcsjl -o yaml \
> | oc adm policy scc-subject-review -f -
RESOURCE ALLOWED BY
Pod/gitlab-7d67db7875-gcsjl anyuid
```

- ▶ 3. 새 서비스 계정을 생성하고 이 계정에 anyuid SCC를 할당합니다.

- 3.1. gitlab-sa라는 서비스 계정을 만듭니다.

```
[student@workstation ~]$ oc create sa gitlab-sa
serviceaccount/gitlab-sa created
```

- 3.2. anyuid SCC를 gitlab-sa 서비스 계정에 할당합니다.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z gitlab-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "gitlab-sa"
```

- ▶ 4. 새로 만든 서비스 계정을 사용하도록 gitlab 애플리케이션을 수정합니다. 새 배포가 성공하는지 확인합니다.

- 4.1. developer 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 4.2. gitlab-sa 서비스 계정을 gitlab 배포에 할당합니다.

```
[student@workstation ~]$ oc set serviceaccount deployment/gitlab gitlab-sa
deployment.apps/gitlab serviceaccount updated
```

- 4.3. gitlab 재배포에 성공했는지 확인합니다. 실행 중인 애플리케이션 포드가 표시될 때까지 oc get pods 명령을 여러 번 실행해야 할 수 있습니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
gitlab-86d6d65-zm2fd 1/1 Running 0 55s
```

- ▶ 5. gitlab 애플리케이션이 제대로 작동하는지 확인합니다.

- 5.1. gitlab 애플리케이션을 공개합니다. gitlab 서비스는 포트 22, 80 및 443에서 수신 대기하므로 --port 옵션을 사용해야 합니다.

```
[student@workstation ~]$ oc expose service/gitlab --port 80 \
> --hostname gitlab.apps.ocp4.example.com
route.route.openshift.io/gitlab exposed
```

- 5.2. 노출된 경로를 가져옵니다.

```
[student@workstation ~]$ oc get routes
NAME HOST/PORT PATH SERVICES PORT ...
gitlab gitlab.apps.ocp4.example.com gitlab 80 ...
```

- 5.3. gitlab 애플리케이션에서 HTTP 쿼리에 응답하는지 확인합니다.

```
[student@workstation ~]$ curl -s \
> http://gitlab.apps.ocp4.example.com/users/sign_in | grep '<title>'

<title>Sign in · GitLab</title>
```

- ▶ 6. authorization-scc 프로젝트를 삭제합니다.

```
[student@workstation ~]$ oc delete project authorization-scc
project.project.openshift.io "authorization-scc" deleted
```

## 완료

workstation 시스템에서 lab 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab authorization-scc finish
```

이것으로 섹션을 완료합니다.

## ▶ 랩

# 애플리케이션 보안 구성

이 랩에서는 시크릿을 생성하여 중요한 구성 정보를 공유하고 덜 제한적인 설정으로 실행하도록 배포를 수정합니다.

## 결과

다음을 수행할 수 있습니다.

- 시크릿을 사용하여 배포에 중요한 정보를 제공합니다.
- 보안 컨텍스트 제약 조건을 사용하여 덜 제한적인 환경에서 애플리케이션을 실행하도록 허용합니다.

## 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

다음 명령을 실행하면 클러스터 API에 연결하고 클러스터에 로그인할 수 있습니다.

```
[student@workstation ~]$ lab authorization-review start
```

- `developer` 사용자로 `authorization-review` 프로젝트를 생성합니다. 이 연습의 모든 추가 작업에서는 `authorization-review` 프로젝트를 사용합니다.
- MySQL 데이터베이스 및 WordPress 애플리케이션에서 사용할 `review-secret`이라는 시크릿을 생성합니다. 시크릿에는 세 개의 키-값 쌍, 즉 `user=wuser`, `password=redhat123`, `database=wordpress`가 포함되어 있어야 합니다.
- `registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47`에 있는 컨테이너 이미지를 사용하여 `mysql`이라는 MySQL 데이터베이스 애플리케이션을 배포합니다. `review-secret` 시크릿을 환경 변수로 사용하려면 `mysql` 배포를 수정합니다. 환경 변수에는 `MYSQL_` 접두사를 사용해야 합니다.
- `quay.io/redhattraining/wordpress:5.3.0`에 있는 컨테이너 이미지를 사용하여 `wordpress`라는 WordPress 애플리케이션을 배포합니다. 이 이미지는 `docker.io/library/wordpress:5.3.0`에서 사용할 수 있는 컨테이너 이미지의 사본입니다. 애플리케이션을 생성할 때 `WORDPRESS_DB_HOST=mysql` 및 `WORDPRESS_DB_NAME=wordpress` 환경 변수를 추가합니다. 배포한 후 `review-secret` 시크릿을 추가 환경 변수로 사용하도록 `wordpress` 배포를 수정합니다. 추가 환경 변수에는 `WORDPRESS_DB_` 접두사를 사용해야 합니다.



### 참고

`wordpress` 포드는 덜 한정적인 보안 컨텍스트 제약 조건을 사용하도록 배포를 수정해야 성공적으로 실행됩니다.

- `admin` 사용자가 `wordpress` 배포를 실행할 수 있는 덜 한정적인 SCC를 확인합니다. `wordpress-sa`라는 서비스 계정을 만든 다음 해당 계정에 `anyuid` SCC를 부여합니다. `wordpress-sa` 서비스 계정을 사용하도록 `wordpress` 배포를 수정합니다.

6. developer 사용자가 **wordpress** 서비스에서 **wordpress-review.apps.ocp4.example.com** 호스트 이름을 사용하여 외부 요청에 액세스할 수 있도록 설정합니다. 웹 브라우저를 사용하여 경로에 액세스하고 WordPress 애플리케이션에 설정 마법사가 표시되는지 확인합니다.

## 평가

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab authorization-review grade
```

## 완료

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab authorization-review finish
```

이것으로 섹션을 완료합니다.

## ▶ 솔루션

# 애플리케이션 보안 구성

이 랩에서는 시크릿을 생성하여 중요한 구성 정보를 공유하고 덜 제한적인 설정으로 실행하도록 배포를 수정합니다.

### 결과

다음을 수행할 수 있습니다.

- 시크릿을 사용하여 배포에 중요한 정보를 제공합니다.
- 보안 컨텍스트 제약 조건을 사용하여 덜 제한적인 환경에서 애플리케이션을 실행하도록 허용합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

다음 명령을 실행하면 클러스터 API에 연결하고 클러스터에 로그인할 수 있습니다.

```
[student@workstation ~]$ lab authorization-review start
```

- `developer` 사용자로 `authorization-review` 프로젝트를 생성합니다. 이 연습의 모든 추가 작업에서는 `authorization-review` 프로젝트를 사용합니다.

- 1.1. 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. `authorization-review` 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project authorization-review
Now using project "authorization-review" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

2. MySQL 데이터베이스 및 WordPress 애플리케이션에서 사용할 `review-secret`이라는 시크릿을 생성합니다. 시크릿에는 세 개의 키-값 쌍, 즉 `user=wpuser`, `password=redhat123`, `database=wordpress`가 포함되어 있어야 합니다.

- 2.1. `review-secret`이라는 시크릿을 만듭니다.

```
[student@workstation ~]$ oc create secret generic review-secret \
> --from-literal user=wpuser --from-literal password=redhat123 \
> --from-literal database=wordpress
secret/review-secret created
```

3. `registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47`에 있는 컨테이너 이미지를 사용하여 `mysql`이라는 MySQL 데이터베이스 애플리케이션을 배포합니다. `review-secret` 시크릿을 환경 변수로 사용하려면 `mysql` 배포를 수정합니다. 환경 변수에는 `MYSQL_` 접두사를 사용해야 합니다.

- 3.1. 새 애플리케이션을 만들어 `mysql` 데이터베이스 서버를 배포합니다.

```
[student@workstation ~]$ oc new-app --name mysql \
> --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
...output omitted...
--> Creating resources ...
 imagestream.image.openshift.io "mysql" created
 deployment.apps "mysql" created
 service "mysql" created
--> Success
...output omitted...
```

- 3.2. `review-secret` 시크릿을 사용하여 `mysql` 배포에서 환경 변수를 초기화합니다. `--prefix` 옵션을 사용하면 시크릿에서 포드로 삽입되는 모든 변수가 `MYSQL_`로 시작합니다.

```
[student@workstation ~]$ oc set env deployment/mysql --prefix MYSQL_ \
> --from secret/review-secret
deployment.apps/mysql updated
```

- 3.3. `mysql` 포드가 성공적으로 재배포되는지 확인합니다.

```
[student@workstation ~]$ watch oc get pods
```

`mysql` 포드에 `1/1` 및 `Running`이 둘 다 표시되면 `Ctrl+C`를 눌러 `watch` 명령을 종료합니다.

```
Every 2.0s: oc get pods ...
```

NAME	READY	STATUS	RESTARTS	AGE
mysql-f675b96f8-vspb9	1/1	Running	0	20s



### 참고

시크릿을 설정한 후 배포가 끝나는 데 다소 시간이 걸릴 수 있습니다.

4. `quay.io/redhattraining/wordpress:5.3.0`에 있는 컨테이너 이미지를 사용하여 `wordpress`라는 WordPress 애플리케이션을 배포합니다. 이 이미지는 `docker.io/library/wordpress:5.3.0`에서 사용할 수 있는 컨테이너 이미지의 사본입니다. 애플리케이션을 생성할 때 `WORDPRESS_DB_HOST=mysql` 및 `WORDPRESS_DB_NAME=wordpress` 환경 변수를 추가합니다. 배포한 후 `review-secret` 시크릿을 추가 환경 변수로 사용하도록 `wordpress` 배포를 수정합니다. 추가 환경 변수에는 `WORDPRESS_DB_` 접두사를 사용해야 합니다.

**참고**

**wordpress** 포드는 덜 한정적인 보안 컨텍스트 제약 조건을 사용하도록 배포를 수정해야 성공적으로 실행됩니다.

- 4.1. **wordpress** 애플리케이션을 배포합니다.

```
[student@workstation ~]$ oc new-app --name wordpress \
> --docker-image quay.io/redhattraining/wordpress:5.3.0 \
> -e WORDPRESS_DB_HOST=mysql \
> -e WORDPRESS_DB_NAME=wordpress
...output omitted...
-> Creating resources ...
 imagestream.image.openshift.io "wordpress" created
 deployment.apps "wordpress" created
 service "wordpress" created
--> Success
...output omitted...
```

- 4.2. **review-secret** 시크릿을 사용하여 **wordpress** 배포에서 환경 변수를 초기화합니다. **--prefix** 옵션을 사용하면 시크릿에서 포드로 삽입되는 변수가 **WORDPRESS\_DB\_**로 시작합니다.

```
[student@workstation ~]$ oc set env deployment/wordpress \
> --prefix WORDPRESS_DB_ --from secret/review-secret
deployment.apps/wordpress updated
```

- 4.3. **review-secret** 시크릿에서 변수를 삽입한 후에도 **wordpress** 포드가 성공적으로 재배포되지 않았는지 확인합니다.

```
[student@workstation ~]$ watch oc get pods -l deployment=wordpress
```

1분 정도 기다린 다음 **Ctrl+C**를 눌러 **watch** 명령을 종료합니다. **wordpress** 포드가 **Error** 상태로 전환된 다음 **CrashLoopBackOff**로 전환될 때마다 계속 다시 시작됩니다.

```
Every 2.0s: oc get pods -l deployment=wordpress
```

```
...
```

NAME	READY	STATUS	RESTARTS	AGE
wordpress-68c49c9d4-wq46g	0/1	CrashLoopBackoff	5	4m30s

- 4.4. 포드 로그에 오류 메시지가 있는지 확인합니다.

```
[student@workstation ~]$ oc logs wordpress-68c49c9d4-wq46g
...output omitted...
(13)Permission denied: AH000072: make_sock: could not bind to address [::]:80
(13)Permission denied: AH000072: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
AH000015: Unable to open logs
```

기본적으로 OpenShift에서는 포드 번호가 1024 미만인 포트에서 수신 대기하는 서비스가 시작되지 않도록 합니다.

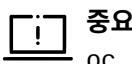
5. **admin** 사용자가 **wordpress** 배포를 실행할 수 있는 멀 한정적인 SCC를 확인합니다. **wordpress-sa**라는 서비스 계정을 만든 다음 해당 계정에 **anyuid** SCC를 부여합니다. **wordpress-sa** 서비스 계정을 사용하도록 **wordpress** 배포를 수정합니다.

- 5.1. **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 5.2. 다른 SCC를 사용하여 권한 문제를 해결할 수 있는지 확인합니다.

```
[student@workstation ~]$ oc get pod/wordpress-68c49c9d4-wq46g -o yaml \
> | oc adm policy scc-subject-review -f -
RESOURCE ALLOWED BY
Pod/wordpress-68c49c9d4-wq46g anyuid
```



### 중요

**oc adm policy** 명령은 **admin** 사용자로 실행해야 합니다.

- 5.3. **wordpress-sa**라는 서비스 계정을 만듭니다.

```
[student@workstation ~]$ oc create serviceaccount wordpress-sa
serviceaccount/wordpress-sa created
```

- 5.4. **anyuid** SCC를 **wordpress-sa** 서비스 계정에 부여합니다. **wordpress** 포드를 **root** 사용자로 실행하는 경우, 포드가 포트 80에서 서비스를 시작하도록 OpenShift에서 허용할 수 있습니다.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added:
"wordpress-sa"
```

- 5.5. **wordpress-sa** 서비스 계정을 사용하도록 **wordpress** 배포를 구성합니다.

```
[student@workstation ~]$ oc set serviceaccount deployment/wordpress \
> wordpress-sa
deployment.apps/wordpress serviceaccount updated
```

- 5.6. 서비스 계정을 구성한 후 **wordpress** 포드에서 재배포하는지 확인합니다.

```
[student@workstation ~]$ watch oc get pods -l deployment=wordpress
```

**wordpress** 포드에 **1/1** 및 **Running**이 둘 다 표시되면 **Ctrl+C**를 눌러 **watch** 명령을 종료합니다.

```
Every 2.0s: oc get pods -l deployment=wordpress
...

```

NAME	READY	STATUS	RESTARTS	AGE
wordpress-bcb5d97f6-mwljs	1/1	Running	0	21s

6. developer 사용자가 **wordpress** 서비스에서 **wordpress-review.apps.ocp4.example.com** 호스트 이름을 사용하여 외부 요청에 액세스할 수 있도록 설정합니다. 웹 브라우저를 사용하여 경로에 액세스하고 WordPress 애플리케이션에 설정 마법사가 표시되는지 확인합니다.

- 6.1. **oc expose** 명령을 사용하여 **wordpress** 애플리케이션 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service/wordpress \
> --hostname wordpress-review.apps.ocp4.example.com
route.route.openshift.io/wordpress exposed
```

- 6.2. 웹 브라우저를 사용하여 URL **http://wordpress-review.apps.ocp4.example.com**에 대한 액세스 권한을 확인합니다. 애플리케이션을 올바르게 배포하면 브라우저에 설치 마법사가 표시됩니다.

또는 **curl** 명령을 사용하여 설치 URL에 직접 액세스합니다.

```
[student@workstation ~]$ curl -s \
> http://wordpress-review.apps.ocp4.example.com/wp-admin/install.php \
> | grep Installation
<title>WordPress › Installation</title>
```

## 평가

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab authorization-review grade
```

## 완료

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab authorization-review finish
```

이것으로 섹션을 완료합니다.

## 요약

---

이 장에서 학습한 내용:

- 시크릿 리소스를 사용하면 애플리케이션 포드에서 중요한 정보를 분리할 수 있습니다. 환경 변수 또는 일반 파일로 애플리케이션 포드에 시크릿을 공개합니다.
- OpenShift에서는 보안 컨텍스트 제약 조건(SCC)을 사용하여 시스템 리소스와 허용된 포드의 상호 작용을 정의합니다. 기본적으로 포드는 노드 리소스에 대한 액세스를 제한하는 **restricted** 컨텍스트에서 작동합니다.



## 12장

# OpenShift 네트워킹 구성 요소 구성

### 목적

OpenShift SDN(소프트웨어 정의 네트워킹) 문제를 문제를 해결하고 네트워크 정책을 구성합니다.

### 목표

- 명령줄 인터페이스를 사용하여 OpenShift 소프트웨어 정의 네트워킹의 문제를 해결합니다.
- OpenShift 클러스터 내부의 애플리케이션에 대한 네트워크 연결을 허용하고 보호합니다.
- 프로젝트와 포드 간의 네트워크 트래픽을 제한합니다.

### 섹션

- OpenShift 소프트웨어 정의 네트워킹 문제 해결(안내에 따른 연습)
- 외부 액세스를 위한 애플리케이션 노출(안내에 따른 연습)
- 네트워크 정책 구성(안내에 따른 연습)

### 랩

애플리케이션을 위한 OpenShift 네트워킹 구성

# OpenShift 소프트웨어 정의 네트워킹 문제 해결

---

## 목표

이 섹션을 완료하고 나면 명령줄 인터페이스를 사용하여 OpenShift 소프트웨어 정의 네트워킹의 문제를 해결할 수 있습니다.

## OpenShift 소프트웨어 정의 네트워킹 소개

OpenShift에서는 소프트웨어 정의 네트워크(SDN)를 구현하여 클러스터와 사용자 애플리케이션의 네트워크 인프라를 관리합니다. 소프트웨어 정의 네트워킹은 여러 네트워크 계층의 요약을 통해 네트워크 서비스를 관리하도록 허용하는 네트워킹 모델입니다. 이 모델에서는 트래픽을 처리하는 소프트웨어(컨트롤 플레인)와 트래픽을 라우팅하는 기본 메커니즘(데이터 플레인)을 분리합니다. SDN의 다양한 기능 중에서 오픈 표준을 사용하면 벤더가 솔루션, 중앙 집중식 관리, 동적 라우팅 및 테넌트 격리를 제안할 수 있습니다.

OpenShift Container Platform에서 SDN은 다음 5가지 요구 사항을 충족합니다.

- 조직 팀에서 애플리케이션을 제공하는 방법을 결정할 수 있도록 네트워크 트래픽 및 네트워크 리소스를 프로그래밍 방식으로 관리합니다.
- 동일한 프로젝트에서 실행되는 컨테이너 간 통신을 관리합니다.
- 동일한 프로젝트에 속해 있든, 개별 프로젝트에서 실행되든 포드 간 통신을 관리합니다.
- 포드에서 서비스로의 네트워크 통신을 관리합니다.
- 외부 네트워크에서 서비스로 또는 컨테이너에서 외부 네트워크로의 네트워크 통신을 관리합니다.

SDN 구현에서는 포드가 포트 할당, IP 주소 임대 및 예약 면에서 가상 시스템과 유사하며 이전 버전과 호환되는 모델을 생성합니다.

## OpenShift 네트워킹 모델 설명

CNI(컨테이너 네트워크 인터페이스)는 네트워크 프로바이더와 컨테이너 실행 사이의 공통 인터페이스이며, 네트워크 플러그인으로 구현됩니다. CNI는 컨테이너 내부에 네트워크 인터페이스를 구성하기 위한 플러그인 사양을 제공합니다. 사양에 기록된 플러그인을 사용하면 다양한 네트워크 프로바이더가 OpenShift 클러스터 네트워크를 제어할 수 있게 됩니다.

SDN에서는 CNI 플러그인을 사용하여 물리적 및 가상 호스트에서 리소스와 프로세스의 사용을 분할하는 Linux 네임스페이스를 사용합니다. 이 구현을 통해 포드 내의 컨테이너가 장치, IP 스택, 방화벽 규칙 및 라우팅 테이블과 같은 네트워크 리소스를 공유할 수 있습니다. SDN에서는 동일한 네트워크의 다른 서비스에서 포드에 액세스할 수 있도록 각 포드에 고유한 라우팅 가능한 IP를 할당합니다.

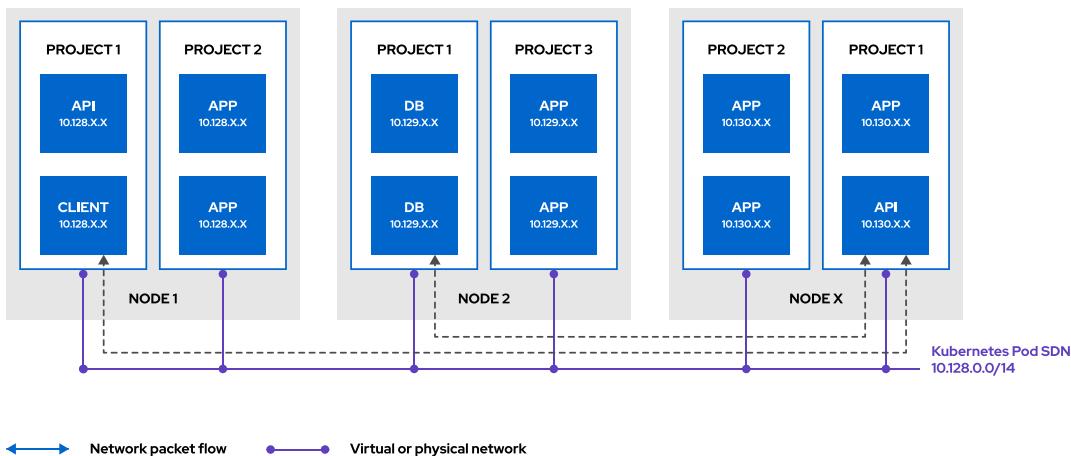
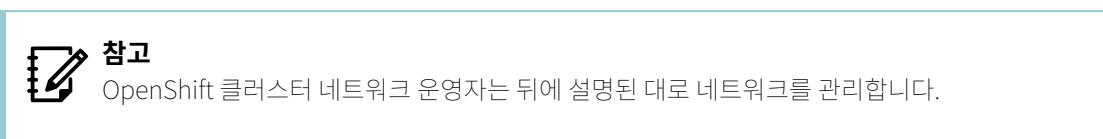
OpenShift에서 사용하는 대표적인 CNI 플러그인에는 OpenShift SDN, OVN-Kubernetes, Kuryr가 있습니다. OpenShift SDN 네트워크 프로바이더는 현재 OpenShift 4.5의 기본 네트워크 프로바이더지만, 향후 OpenShift 4 릴리스에서는 OVN-Kubernetes 네트워크 프로바이더가 기본으로 설정됩니다.

OpenShift SDN 네트워크 프로바이더는 동일한 노드의 포드를 연결하는 데에는 OVS(Open vSwitch)를 사용하고, 노드를 연결하는 데에는 VXLAN(Virtual Extensible LAN) 터널링을 사용합니다. OVN-Kubernetes는 OVN(Open Virtual Network)을 사용하여 클러스터 네트워크를 관리합니다. OVN는 가상 네트워크 추상화를 통해 OVS를 확장합니다. Kuryr는 Neutron 및 Octavia Red Hat OpenStack Platform 서비스를 통해 네트워킹을 제공합니다.

## 레거시 애플리케이션 마이그레이션

SDN 디자인을 사용하면 애플리케이션 구성 요소가 서로 통신하는 방법을 변경할 필요가 없기 때문에 레거시 애플리케이션을 쉽게 컨테이너화할 수 있습니다. 애플리케이션을 TCP/UDP 스택을 통해 통신하는 여러 서비스로 구성한 경우 포드의 컨테이너에서 동일한 네트워크 스택을 공유하므로 이 접근 방식은 여전히 작동합니다.

다음 다이어그램에서는 모든 포드가 공유 네트워크에 연결되는 방법을 보여줍니다.



**그림 12.1: Kubernetes 기본 네트워킹**

## 포드 액세스에 대한 서비스 사용

Kubernetes는 서비스라는 개념을 제공하는데, 이는 OpenShift 애플리케이션의 필수 리소스에 해당합니다. 서비스를 사용하면 공통 액세스 경로에서 포드의 논리적 그룹을 사용할 수 있습니다. 서비스는 하나 이상의 포드 앞에서 부하 분산 장치 역할을 하므로 애플리케이션에 대한 액세스로부터 애플리케이션 사양(예: 복제본을 실행하는 수)을 분리합니다. 여러 멤버 포드에 클라이언트 요청 부하를 분산하고, 개별 포드 IP 주소를 추적하지 않고 포드와 통신할 수 있는 안정적인 인터페이스를 제공합니다.

대부분의 실제 애플리케이션은 단일 포드로 실행되지 않습니다. 늘어나는 사용자 수요를 충족하기 위해 애플리케이션을 여러 포드에서 실행할 수 있도록 수평적으로 확장해야 합니다. OpenShift 클러스터에서 포드는 새 애플리케이션 버전을 배포하는 동안 또는 유지 관리를 위해 노드를 제외하는 경우 등 클러스터의 노드 전반에서 지속적으로 생성되고 제거됩니다. 포드는 만들어질 때마다 서로 다른 IP 주소가 할당됩니다. 따라서 포드는 주소 지정이 쉽지 않습니다. 포드에서 다른 포드의 IP 주소를 검색하는 대신, 포드가 실행 중인 위치와 관계없이 다른 포드에서 사용할 수 있는 하나의 고유 IP 주소를 제공하는 서비스가 있습니다.

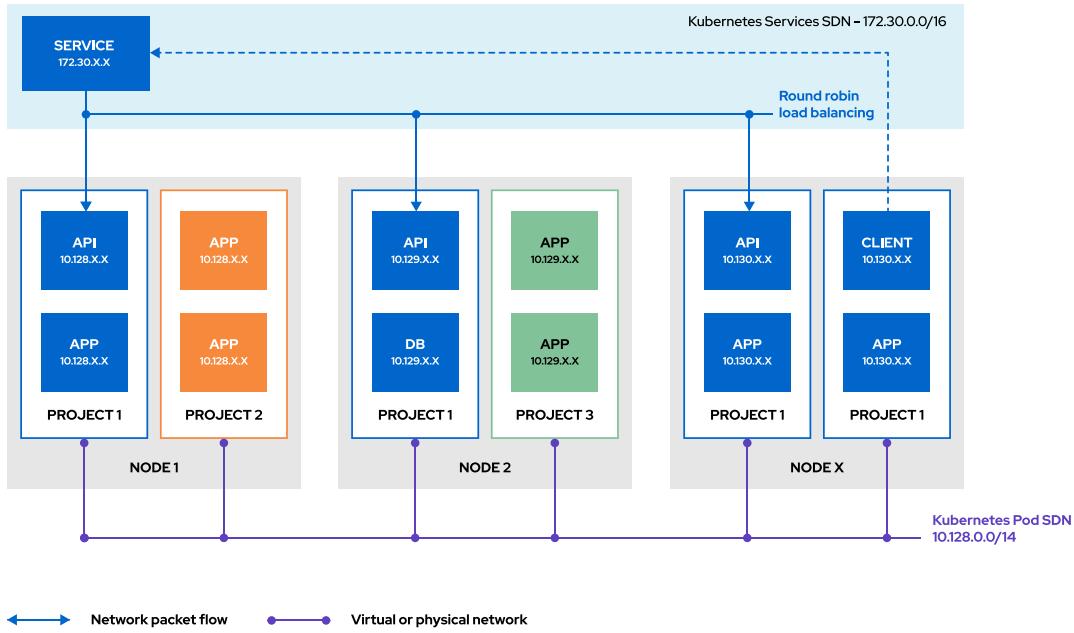
이러한 서비스에서는 해당 서비스를 통해 트래픽을 수신하는 포드를 선택기(레이블)로 표시합니다. 이러한 선택기와 일치하는 각 포드는 엔드포인트로 서비스 리소스에 추가됩니다. 포드가 생성되고 종료될 때 서비스가 엔드포인트를 자동으로 업데이트합니다.

선택기를 사용하면 아키텍처와 애플리케이션의 라우팅을 설계하는 방식이 유연해집니다. 예를 들어 애플리케이션을 계층으로 나누고 각 계층에 대한 서비스를 만들도록 결정할 수 있습니다. 선택기를 사용하면 유연하고 복원성이 뛰어난 설계가 가능합니다.

## 12장 | OpenShift 네트워킹 구성 요소 구성

OpenShift에서는 두 개의 서브넷(포드용 서브넷 하나, 서비스용 서브넷 하나)을 사용합니다. 트래픽은 포드에 투명한 방식으로 전달됩니다. 에이전트(사용하는 네트워크 모드에 따라 다름)는 선택기와 일치하는 포드에 대한 트래픽을 라우팅하는 라우팅 규칙을 관리합니다.

다음 다이어그램에서는 세 개의 API 포드가 개별 노드에서 실행되는 방법을 보여줍니다. **service1** 서비스는 이러한 세 포드에 부하를 분산합니다.



**그림 12.2: 애플리케이션 액세스에 대한 서비스 사용**

다음 YAML 정의는 서비스를 생성하는 방법을 보여줍니다. 이는 TCP 포트 443을 노출하는 가상 IP를 만드는 애플리케이션-프론트엔드 서비스를 정의합니다. 프론트엔드 애플리케이션은 권한이 없는 포트 8843에서 수신 대기합니다.

```

kind: Service
apiVersion: v1
metadata:
 name: application-frontend ①
 labels:
 app: frontend-svc ②
spec:
 ports: ③
 - name: HTTP
 protocol: TCP
 port: 443 ④
 targetPort: 8443 ⑤
 selector: ⑥
 app: shopping-cart
 name: frontend
 type: ClusterIP ⑦

```

- ①** 서비스의 이름. 이 식별자를 사용하면 서비스를 생성한 후 이를 관리할 수 있습니다.
- ②** 선택기로 사용할 수 있는 레이블. 이를 통해 서비스를 논리적으로 그룹화할 수 있습니다.
- ③** 노출할 네트워크 포트를 설명하는 오브젝트의 배열.

각 항목은 포트 매핑의 이름을 정의합니다. 이 값은 일반이며 식별 목적으로만 사용됩니다.

- ❸ 이 포트는 서비스에서 노출하는 포트입니다. 이 포트를 사용하여 서비스에서 노출하는 애플리케이션에 연결합니다.
- ❹ 이 포트는 애플리케이션에서 수신 대기하는 포트입니다. 서비스는 서비스 포트에서 서비스 대상 포트로의 전달 규칙을 만듭니다.
- ❺ 선택기는 서비스 풀에 있는 포드를 정의합니다. 서비스는 이 선택기를 사용하여 트래픽을 라우팅할 위치를 결정합니다.

이 예제에서 서비스는 해당 레이블이 **app: shopping-cart** 및 **name: frontend**와 일치하는 모든 포드를 대상으로 합니다.

- ❻ 이는 서비스가 노출되는 방식입니다. **ClusterIP**는 클러스터 내부의 IP 주소를 사용하여 서비스를 노출하며, 이 값이 기본값입니다. 기타 서비스 유형은 이 교육 과정의 다른 곳에서 설명합니다.

## DNS 운영자 설명

DNS 운영자는 GoLang으로 작성된 경량 DNS 서버인 CoreDNS에서 관리하는 DNS 서버를 배포하고 실행합니다. DNS 운영자는 포드 간에 DNS 이름 확인을 제공하여 서비스에서 해당 엔드포인트를 검색할 수 있도록 합니다.

새 애플리케이션을 생성할 때마다 OpenShift는 DNS 확인을 위해 CoreDNS 서비스 IP에 연결하도록 포드를 구성합니다.

다음 명령을 실행하여 DNS 운영자의 구성을 검토합니다.

```
[user@demo ~]$ oc describe dns.operator/default
Name: default
...output omitted...
API Version: operator.openshift.io/v1
Kind: DNS
...output omitted...
Spec:
Status:
 Cluster Domain: cluster.local
 Cluster IP: 172.30.0.10
...output omitted...
```

DNS 운영자는 다음을 담당합니다.

- 기본 클러스터 DNS 이름(**cluster.local**)을 만듭니다.
- 정의하는 서비스에 DNS 이름을 할당합니다(예: **db.backend.svc.cluster.local**).

예를 들어 **example-6c4984d949-7m26r**이라는 포드에서 서비스의 FQDN을 통해 **test** 프로젝트의 **hello** 서비스에서 **hello** 포드에 연결할 수 있습니다.

```
[user@demo ~]$ oc rsh example-6c4984d949-7m26r curl \
> hello.test.svc.cluster.local:8080
```

## 서비스에 대한 DNS 레코드 관리

이 DNS 구현을 통해 포드는 프로젝트나 클러스터의 리소스에 대한 DNS 이름을 원활하게 확인할 수 있습니다. 포드는 서비스에 액세스할 수 있는 예측 가능한 명명 스키마를 사용할 수 있습니다. 예를 들어 컨테이너에서 **db.backend.svc.cluster.local** 호스트 이름을 쿼리하면 서비스의 IP 주소가 반환됩니다. 이 예

제에서 **db**는 서비스 이름이고 **backend**는 프로젝트 이름이며 **cluster.local**은 클러스터 DNS 이름입니다.

CoreDNS는 서비스에 대한 두 가지 종류의 레코드, 즉 서비스를 확인하는 **A** 레코드 및 다음 형식과 일치하는 **SRV** 레코드를 생성합니다.

```
_port-name._port-protocol.svc-name.namespace.svc.cluster-domain.cluster-domain
```

예를 들어 **HTTPS** 서비스를 통해 TCP 포트 443을 노출하는 서비스를 사용하는 경우 **SRV** 레코드가 다음과 같이 생성됩니다.

```
_443-tcp._tcp.https.frontend.svc.cluster.local
```



### 참고

서비스에 클러스터 IP가 없으면 DNS 운영자는 서비스 뒤에 포드의 IP 집합을 확인하는 DNS **A** 레코드를 할당합니다.

마찬가지로, 새로 생성된 **SRV** 레코드는 서비스 뒤에 있는 모든 포드로 확인됩니다.

## 클러스터 네트워크 운영자 소개

OpenShift Container Platform은 클러스터 네트워크 운영자를 사용하여 SDN을 관리합니다. 여기에는 사용할 네트워크 CIDR, 네트워크 프로바이더, IP 주소 풀이 포함됩니다. 설치 전에 Cluster Network Operator 구성이 완료되지만, OpenShift SDN 기본 CNI 네트워크 프로바이더에서 OVN-Kubernetes 네트워크 프로바이더 마이그레이션할 수 있습니다.

다음 **oc get** 명령을 관리 사용자로 실행하여 **Network.config.openshift.io** 사용자 지정 리소스 정의에 의해 관리되는 SDN 구성을 참조합니다.

```
[user@demo ~]$ oc get network/cluster -o yaml
apiVersion: config.openshift.io/v1
kind: Network
...output omitted...
spec:
 clusterNetwork:
 - cidr: 10.128.0.0/14 ①
 hostPrefix: 23 ②
 externalIP:
 policy: {}
 networkType: OpenshiftSDN ③
 serviceNetwork:
 - 172.30.0.0/16
 ...output omitted...
```

- ① 클러스터의 모든 포드에 대한 CIDR를 정의합니다. 이 예에서 SDN에는 **255.252.0.0**의 네트마스크가 있으며 262144 IP 주소를 할당할 수 있습니다.
- ② 호스트 접두사를 정의합니다. **23**의 값은 512 할당 가능 IP로 변환되는 **255.255.254.0**의 네트마스크를 나타냅니다.
- ③ 현재 SDN 프로바이더를 표시합니다. **OpenShiftSDN**, **OVNKubernetes** 및 **Kuryr** 중에서 선택할 수 있습니다.

**참고**

추가 네트워크 구성은 본 과정에서 다루지 않습니다. Kubernetes 네트워크 사용자 지정 리소스 정의에 대한 자세한 내용은 참조 섹션에 나열된 Kubernetes 네트워크 사용자 지정 리소스 정의 실질적인 표준 버전 1 문서를 참조하십시오.

## Multus CNI 소개

Multus는 OpenShift에서 여러 네트워크 카드를 지원하는 오픈소스 프로젝트입니다. Multus에서 해결하는 문제 중 하나는 네트워크 기능 가상화를 컨테이너로 마이그레이션하는 것입니다. Multus는 컨테이너에 있는 보조 네트워크 장치의 구현 및 라이프사이클을 관리하는 다른 CNI 플러그인의 브로커 및 중재자 역할을 합니다. Multus는 SR-IOV, vHost CNI, Flannel 및 Calico와 같은 플러그인을 지원합니다. 통신 서비스, 에지 컴퓨팅, 가상화에서 흔히 볼 수 있는 특수한 에지 사례는 Multus에서 처리하므로 포드에 여러 네트워크 인터페이스를 사용할 수 있습니다.

모든 Kubernetes 및 OpenShift 네트워킹 기능(예: 서비스, 인그레스, 경로)은 Multus에서 제공하는 추가 네트워크 장치를 무시합니다.

**참조**

자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 네트워킹 설명서의 OpenShift Container Platform의 Cluster Network Operator 장을 참조하십시오.  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/networking/index#cluster-network-operator](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#cluster-network-operator)

**클러스터 네트워킹**

<https://kubernetes.io/docs/concepts/cluster-administration/networking/>

**Kubernetes 네트워크 사용자 지정 리소스 정의 실질적인 표준 버전 1**

<https://github.com/k8snetworkplumbingwg/multi-net-spec/blob/master/v1.0/%5Bv1%5D%20Kubernetes%20Network%20Custom%20Resource%20Definition%20De-facto%20Standard.md>

**CoreDNS: DNS 및 서비스 검색**

<https://coredns.io/>

**Multus-CNI**

<https://github.com/intel/multus-cni>

## ▶ 연습 가이드

# OpenShift 소프트웨어 정의 네트워킹 문제 해결

이 연습에서는 Kubernetes 스타일의 애플리케이션 배포를 통해 연결 문제를 진단하고 해결합니다.

## 결과

다음을 수행할 수 있습니다.

- **To Do** Node.js 애플리케이션을 배포합니다.
- 애플리케이션 서비스를 공개할 경로를 생성합니다.
- `oc debug`를 사용하여 애플리케이션의 포드 간 통신 문제를 문제 해결합니다.
- OpenShift 서비스를 업데이트합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있는지 확인합니다.

```
[student@workstation ~]$ lab network-sdn start
```

OpenShift 개발자인 여러분은 **To Do** Node.js 애플리케이션을 OpenShift로 마이그레이션하는 작업을 완료했습니다. 애플리케이션은 두 개의 배포(데이터베이스용 배포 및 프런트엔드용 배포)로 구성됩니다. 또한 포드 간 통신을 위한 두 가지 서비스가 포함되어 있습니다.

애플리케이션이 초기화되는 것처럼 보이지만 웹 브라우저를 통해 액세스할 수 없습니다. 이 활동에서는 애플리케이션 문제를 해결하고 해당 문제를 수정합니다.

### ▶ 1. OpenShift 클러스터에 로그인하고 **network-sdn** 프로젝트를 만듭니다.

#### 1.1. 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

#### 1.2. **network-sdn** 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project network-sdn
Now using project "network-sdn" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- ▶ 2. 데이터베이스를 배포하고 데이터를 복원합니다.

/home/student/D0280/labs/network-sdn/todo-db.yaml 파일은 다음 리소스를 정의합니다.

- MySQL 이미지를 기반으로 컨테이너를 만드는 배포.
- `mysql` 애플리케이션을 가리키는 서비스.

- 2.1. `network-sdn` 디렉터리로 이동하고 파일을 나열합니다. 이후 단계에서는 `db-data.sql`을 사용하여 애플리케이션에 대한 데이터베이스를 초기화합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/network-sdn
[student@workstation network-sdn]$ ls
db-data.sql todo-db.yaml todo-frontend.yaml
```

- 2.2. `todo-db.yaml`에 대해 `oc create`를 `-f` 옵션과 함께 사용하여 데이터베이스 서버 포드를 배포합니다.

```
[student@workstation network-sdn]$ oc create -f todo-db.yaml
deployment.apps/mysql created
service/mysql created
```

- 2.3. `oc status` 명령을 실행하여 프로젝트에 있는 리소스를 검토합니다. `mysql` 서비스는 데이터베이스 포드를 가리킵니다.

```
[student@workstation network-sdn]$ oc status
In project network-sdn on server https://api.ocp4.example.com:6443

svc/mysql - 172.30.223.41:3306
 deployment/mysql deploys registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
 deployment #1 running for 4 seconds - 0/1 pods
...output omitted...
```

- 2.4. 잠시 기다린 후 데이터베이스 포드가 실행되고 있는지 확인합니다. 데이터베이스 포드 이름을 검색하여 `items` 데이터베이스의 테이블을 복원합니다.

```
[student@workstation network-sdn]$ oc get pods
NAME READY STATUS RESTARTS AGE
mysql-94dc6645b-hjjqb 1/1 Running 0 33m
```

- 2.5. `oc cp` 명령을 사용하여 데이터베이스 덤프를 포드로 전송합니다. 포드 이름을 이전 단계에서 가져온 항목으로 바꿔야 합니다.

```
[student@workstation network-sdn]$ oc cp db-data.sql mysql-94dc6645b-hjjqb:/tmp/
```



### 참고

명령에서 출력을 생성하지 않습니다.

- 2.6. `oc rsh` 명령을 사용하여 포드에 연결하고 데이터베이스를 복원합니다.

```
[student@workstation network-sdn]$ oc rsh mysql-94dc6645b-hjjqb bash
bash-4.2$ mysql -u root items < /tmp/db-data.sql
```

2.7. **Item** 테이블이 데이터베이스에 있는지 확인합니다.

```
bash-4.2$ mysql -u root items -e "show tables;"
```

Tables_in_items	
+-----+	
Item	
+-----+	

2.8. 컨테이너를 종료합니다.

```
bash-4.2$ exit
exit
```

- ▶ 3. 프론트엔드 애플리케이션을 배포합니다. `/home/student/D0280/labs/network-sdn/todo-frontend.yaml` 파일은 다음 리소스를 정의합니다.

- **Todo** Node.js 애플리케이션을 만드는 배포.
- **frontend** 애플리케이션을 가리키는 서비스.

3.1. `oc create` 명령을 사용하여 프론트엔드 애플리케이션을 만듭니다.

```
[student@workstation network-sdn]$ oc create -f todo-frontend.yaml
deployment.apps/frontend created
service/frontend created
```

3.2. 프론트엔드 컨테이너가 시작될 때까지 잠시 기다린 다음 `oc get pods` 명령을 실행합니다.

```
[student@workstation network-sdn]$ oc get pods
NAME READY STATUS RESTARTS AGE
frontend-57b8b445df-f56qh 1/1 Running 0 34s
...output omitted...
```

- ▶ 4. 프론트엔드 서비스에 액세스하고 애플리케이션에 액세스할 경로를 만듭니다.

- 4.1. 외부 네트워크에서 애플리케이션에 액세스할 경로를 만들어야 합니다. 이 경로를 생성하려면 **frontend** 서비스에 대해 `oc expose` 명령을 사용합니다. `--hostname` 옵션을 사용하여 OpenShift가 만드는 기본 FQDN을 재정의합니다.

```
[student@workstation network-sdn]$ oc expose service frontend \
> --hostname todo.apps.ocp4.example.com
route.route.openshift.io/frontend exposed
```

4.2. 프로젝트의 경로를 나열합니다.

```
[student@workstation network-sdn]$ oc get routes
NAME HOST/PORT PATH SERVICES PORT ...
frontend todo.apps.ocp4.example.com frontend 8080 ...
```

예제에 표시된 것처럼 OpenShift는 애플리케이션이 수신 대기하는 포트를 감지하고 포트 80에서 포트 8080(대상 포트)으로의 전달 규칙을 생성합니다.

- 4.3. **workstation**에서 Firefox를 열고 <http://todo.apps.ocp4.example.com/todo>에 액세스합니다. URL의 끝에 후행 슬래시를 추가해야 합니다.  
다음 화면 캡처에 표시된 대로 애플리케이션에 연결할 수 없습니다.

## Application is not available

The application is currently not serving requests at this endpoint. It may not have been started or is still starting.

**i** Possible reasons you are seeing this page:

- **The host doesn't exist.** Make sure the hostname was typed correctly and that a route matching this hostname exists.
- **The host exists, but doesn't have a matching path.** Check if the URL path was typed correctly and that the route was created using the desired path.
- **Route and path matches, but all pods are down.** Make sure that the resources exposed by this route (pods, services, deployment configs, etc) have at least one pod running.

- 4.4. 포드 로그에서 오류를 검사합니다. 출력에 오류가 표시되지 않습니다.

```
[student@workstation network-sdn]$ oc logs frontend-57b8b445df-f56qh
App is ready at : 8080
```

- ▶ 5. **oc debug**를 실행하여 **frontend** 배포에 있는 기존 포드의 복사본을 만듭니다. 이 포드를 사용하여 데이터베이스에 대한 연결을 확인합니다.

- 5.1. 디버그 포드를 만들기 전에 데이터베이스 서비스 IP를 검색합니다. 이후 단계에서는 **curl** 명령을 사용하여 데이터베이스 엔드포인트에 액세스합니다.  
JSONPath 표현식을 사용하여 서비스 IP를 검색할 수 있습니다.

```
[student@workstation network-sdn]$ oc get service/mysql \
> -o jsonpath=".spec.clusterIP}{'\n'}"
172.30.103.29
```

- 5.2. 웹 애플리케이션 포드를 실행하는 프런트엔드 배포에 대해 **oc debug** 명령을 실행합니다.

```
[student@workstation network-sdn]$ oc debug -t deployment/frontend
Starting pod/frontend-debug ...
Pod IP: 10.131.0.144
If you don't see a command prompt, try pressing enter.
sh-4.2$
```

- 5.3. **frontend** 배포와 데이터베이스 간 연결을 테스트하는 한 가지 방법은 다양한 프로토콜을 지원하는 **curl**을 사용하는 것입니다.

**curl** 명령을 사용하여 포트 3306(MySQL 기본 포트)을 통해 데이터베이스에 연결합니다. 해당 IP 주소를 이전에 **mysql** 서비스를 위해 가져온 IP 주소로 교체합니다. 완료되면 **Ctrl+C**를 입력하여 세션을 종료하고 **exit**를 입력하여 디버그 포드를 종료합니다.

```
sh-4.2$ curl -v telnet://172.30.103.29:3306
* About to connect() to 172.30.103.29 port 3306 (#0)
* Trying 172.30.103.29...
* Connected to 172.30.103.29 (172.30.103.29) port 3306 (#0)
J
8.0.21
* RCVD IAC 2
* RCVD IAC 199
Ctrl+C
sh-4.2$ exit
exit

Removing debug pod ...
```

출력은 데이터베이스가 실행 중이며 **프런트엔드** 배포에서 액세스할 수 있음을 나타냅니다.

- ▶ 6. 다음 단계에서는 데이터베이스 컨테이너에서 프런트엔드 컨테이너에 연결하여 클러스터 내부의 네트워크 연결이 작동하는지 확인합니다.

**frontend** 포드에 대한 일부 정보를 가져오고 **oc debug** 명령을 사용하여 **mysql** 배포의 문제를 진단합니다.

- 6.1. 디버그 포드를 만들기 전에 **프런트엔드** 서비스의 IP 주소를 검색합니다.

```
[student@workstation network-sdn]$ oc get service/frontend \
> -o jsonpath=".spec.clusterIP\{\n'\}"
172.30.23.147
```

- 6.2. **oc debug** 명령을 실행하여 **mysql** 배포에 따른 문제 해결을 위한 컨테이너를 만듭니다. MySQL 서버 이미지가 **curl** 명령을 제공하지 않으므로 컨테이너 이미지를 재정의해야 합니다.

```
[student@workstation network-sdn]$ oc debug -t deployment/mysql \
> --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/mysql-debug ...
Pod IP: 10.131.0.146
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

- 6.3. **curl**을 사용하여 포트 8080을 통해 **frontend** 애플리케이션에 연결합니다. 해당 IP 주소를 이전에 **frontend** 서비스를 위해 가져온 IP 주소로 교체합니다.

다음 출력은 **curl** 명령이 시간 초과되었음을 나타냅니다. 이는 애플리케이션이 실행 중이 아니거나 서비스에서 애플리케이션에 액세스할 수 없음을 나타낼 수 있습니다.

```
sh-4.4$ curl -m 10 -v http://172.30.23.147:8080
* Rebuilt URL to: http://172.30.23.147:8080/
* Trying 172.30.23.147...
* TCP_NODELAY set
* Connection timed out after 10000 milliseconds
* Closing connection 0
curl: (28) Connection timed out after 10000 milliseconds
```

6.4. 디버그 포드를 종료합니다.

```
sh-4.4$ exit
exit

Removing debug pod ...
```

- ▶ 7. 다음 단계에서는 개인 IP를 통해 프런트엔드 포드에 연결합니다. 이를 통해 문제가 서비스와 관련이 있는지 테스트할 수 있습니다.

7.1. 프런트엔드 포드의 IP를 검색합니다.

```
[student@workstation network-sdn]$ oc get pods -o wide -l name=frontend
NAME READY STATUS RESTARTS AGE IP ...
frontend-57b8b445df-f56qh 1/1 Running 0 39m 10.128.2.61 ...
```

7.2. mysql 배포에서 디버그 포드를 만듭니다.

```
[student@workstation network-sdn]$ oc debug -t deployment/mysql \
> --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/mysql-debug ...
Pod IP: 10.131.1.27
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

- 7.3. 포트 8080의 frontend 포드에 대해 자세한 정보 표시 모드에서 curl 명령을 실행합니다. 해당 IP 주소를 이전에 frontend 포드를 위해 가져온 IP 주소로 교체합니다.

```
sh-4.2$ curl -v http://10.128.2.61:8080/todo/
* Trying 10.128.2.61...
* TCP_NODELAY set
* Connected to 10.128.2.61 (10.128.2.61) port 8080 (#0)
> GET /todo/ HTTP/1.1
> Host: 10.128.2.61:8080
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
...output omitted...
```

curl 명령은 포드의 개인 IP를 통해 애플리케이션에 액세스할 수 있습니다.

7.4. 디버그 포드를 종료합니다.

```
sh-4.2$ exit
exit

Removing debug pod ...
```

▶ 8. 프론트엔드 서비스의 구성을 검토합니다.

- 8.1. 프로젝트에서 서비스를 나열하고 **프론트엔드** 서비스가 있는지 확인합니다.

```
[student@workstation network-sdn]$ oc get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
frontend ClusterIP 172.30.23.147 <none> 8080/TCP 93m
mysql ClusterIP 172.30.103.29 <none> 3306/TCP 93m
```

- 8.2. **프론트엔드** 서비스의 구성과 상태를 검토합니다. 서비스가 패킷을 전달해야 하는 포드를 나타내는 서비스 선택기의 값을 확인합니다.

```
[student@workstation network-sdn]$ oc describe svc/frontend
Name: frontend
Namespace: network-sdn
Labels: app=todonodejs
 name=frontend
Annotations: <none>
Selector: name=api
Type: ClusterIP
IP: 172.30.23.147
Port: <unset> 8080/TCP
TargetPort: 8080/TCP
Endpoints: <none>
Session Affinity: None
Events: <none>
```

또한 이 출력은 서비스에 엔드포인트가 없음을 나타내므로 수신되는 트래픽을 애플리케이션으로 전달할 수 없습니다.

- 8.3. **프론트엔드** 배포의 레이블을 검색합니다. 출력에서 확인할 수 있듯이 포드는 값이 **frontend**인 **name** 레이블을 사용하여 생성되는 반면, 이전 단계의 서비스에서는 **api**를 값으로 사용합니다.

```
[student@workstation network-sdn]$ oc describe deployment/frontend | \
> grep Labels -A1
Labels: app=todonodejs
 name=frontend
--
Labels: app=todonodejs
 name=frontend
```

▶ 9. **프론트엔드** 서비스를 업데이트하고 애플리케이션에 액세스합니다.

- 9.1. **oc edit** 명령을 실행하여 **frontend** 서비스를 편집합니다. 올바른 레이블과 일치하도록 선택기를 업데이트합니다.

```
[student@workstation network-sdn]$ oc edit svc/frontend
```

선택기를 정의하는 섹션을 찾은 다음 선택기 내부의 **name: frontend** 레이블을 업데이트 합니다. 변경 후 편집기를 종료합니다.

```
...output omitted...
selector:
 name: frontend
...output omitted...
```

변경 사항을 저장하고 **oc edit** 명령이 이를 적용했는지 확인합니다.

```
service/frontend edited
```

9.2. 서비스 구성을 검토하여 서비스에 앤드포인트가 있는지 확인합니다.

```
[student@workstation network-sdn]$ oc describe svc/frontend
Name: frontend
Namespace: network-sdn
Labels: app=todonodejs
 name=frontend
Annotations: <none>
Selector: name=frontend
Type: ClusterIP
IP: 172.30.169.113
Port: <unset> 8080/TCP
TargetPort: 8080/TCP
Endpoints: 10.128.2.61:8080
Session Affinity: None
Events: <none>
```

9.3. **workstation** 시스템에서 Firefox를 열고 **http://todo.apps.ocp4.example.com/todo/**에서 To Do 애플리케이션에 액세스합니다.  
To Do 애플리케이션이 표시됩니다.

To Do List

Id	Description	Done	
1	Pick up newsp...	false	
2	Buy groceries	true	

First Previous **1** Next Last

Add Task

Description:  Add Description.

Completed:

**Clear** **Save**

- ▶ 10. 사용자 홈 디렉터리로 이동하여 **network-sdn** 프로젝트를 삭제합니다.

```
[student@workstation network-sdn]$ cd
[student@workstation ~]$ oc delete project network-sdn
project.project.openshift.io "network-sdn" deleted
```

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab network-sdn finish
```

이로써 안내에 따른 연습이 완료됩니다.

# 외부 액세스를 위해 애플리케이션 노출

## 목표

이 섹션을 마치면 OpenShift 클러스터 내부의 애플리케이션에 대한 네트워크 연결을 허용하고 보호할 수 있습니다.

## 외부 네트워크에서 애플리케이션에 액세스

OpenShift Container Platform은 애플리케이션을 외부 네트워크에 노출할 수 있는 여러 가지 방법을 제공합니다. HTTP 및 HTTPS 트래픽, TCP 애플리케이션 및 비 TCP 트래픽을 노출할 수 있습니다. 이러한 메서드 중 일부는 **NodePort** 또는 부하 분산 장치와 같은 서비스 유형에 해당하지만, 다른 메서드에서는 **인그레스** 및 **경로**와 같은 자체 API 리소스를 사용합니다.

OpenShift 경로를 사용하면 애플리케이션을 외부 네트워크에 노출할 수 있습니다. 경로를 사용하면 공개적으로 액세스할 수 있는 고유한 호스트 이름을 사용하여 애플리케이션에 액세스할 수 있습니다. 경로는 라우터 플러그인을 사용하여 공용 IP에서 포드로 트래픽을 리디렉션합니다.

다음 다이어그램은 경로에서 클러스터의 포드로 실행되는 애플리케이션을 노출하는 방법을 보여줍니다.

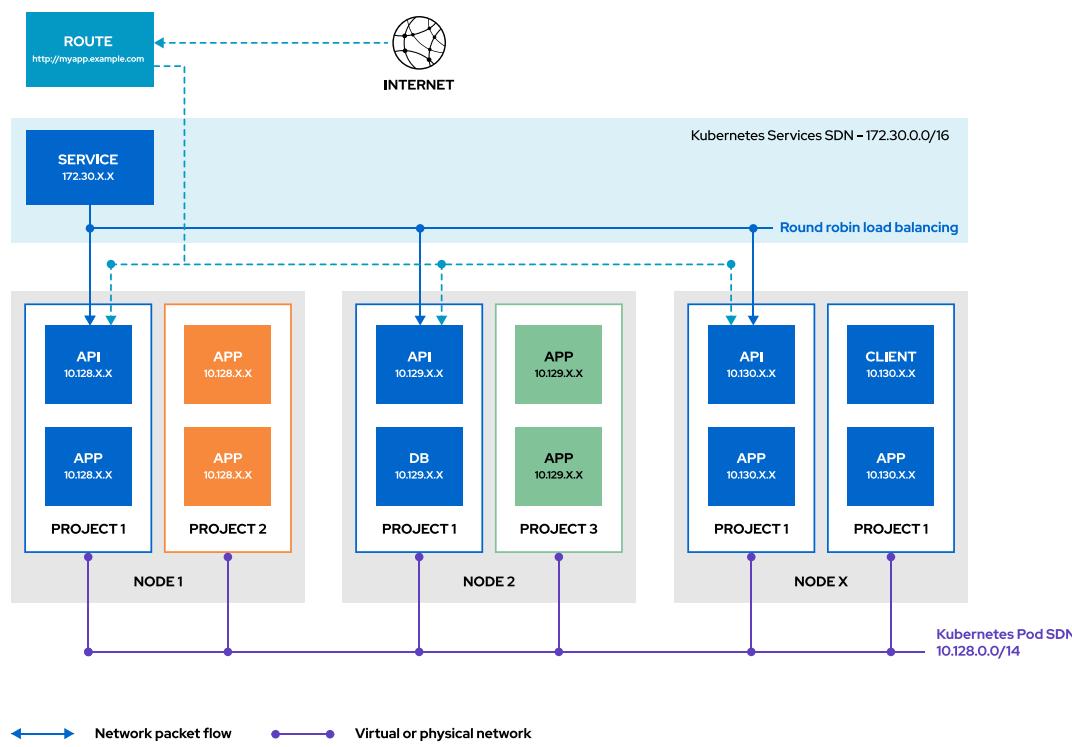


그림 12.5: 경로를 사용하여 애플리케이션 노출



### 참고

성능상의 이유로 라우터는 서비스 구성에 따라 포드에 직접 요청을 보냅니다.

점선은 이러한 구현을 나타냅니다. 즉 라우터는 서비스 네트워크를 통해 포드에 액세스합니다.

## 인그레스 트래픽을 관리하는 메서드 설명

인그레스 트래픽을 관리하는 가장 일반적인 방법은 수신 컨트롤러를 사용하는 것입니다. OpenShift는 클러스터 내에서 포드로 실행되는 공유 라우터 서비스를 통해 인그레스 컨트롤러를 구현합니다. 다른 일반 포드처럼 이 포드를 확장하고 복제할 수 있습니다. 이 라우터 서비스는 오픈소스 소프트웨어 HAProxy를 기반으로 합니다.

경로 및 인그레스는 인그레스 트래픽을 처리하는 데 필요한 주요 리소스입니다.

### 경로

경로는 인그레스 트래픽을 클러스터의 서비스에 제공합니다. 경로는 Kubernetes 인그레스 오브젝트보다 먼저 생성되었으며 더 많은 기능을 제공합니다. 경로는 TLS 재암호화, TLS 패스스루, 파랑-녹색 배포를 위한 트래픽 분할 등 표준 인터페이스를 통해 Kubernetes 인그레스 컨트롤러에서는 지원되지 않는 고급 기능을 제공합니다.

### 수신

수신은 OpenShift 리소스인 경로와 동일한 기능 중 일부를 제공하는 Kubernetes 리소스입니다. 인그레스는 외부 요청을 수락하고 경로를 기반으로 프록시 처리합니다. HTTP, HTTPS, SNI(서버 이름 확인) 및 TLS(SNI 사용) 트래픽 유형만 허용할 수 있습니다. OpenShift에서는 인그레스 오브젝트에서 지정하는 조건을 충족하기 위해 경로를 생성합니다.

인그레스 및 경로의 대안은 있지만 이러한 대안은 특별한 활용 사례를 위한 것입니다. 다음 서비스 유형에서는 외부에서 서비스에 액세스할 수 있습니다.

### 외부 부하 분산 장치

이 리소스는 OpenShift가 클라우드 환경에서 부하 분산 장치를 회전하도록 지시합니다. 부하 분산 장치는 OpenShift가 부하 분산 장치를 프로비저닝하기 위해 클러스터가 실행 중인 클라우드 프로바이더와 상호 작용하도록 지시합니다.

### 서비스 외부 IP

이 메서드는 OpenShift가 클러스터 IP 중 하나의 트래픽을 컨테이너에 리디렉션하도록 NAT 규칙을 설정하도록 지시합니다.

### NodePort

이 메서드를 사용하면 OpenShift가 노드 IP 주소의 정적 포트에 서비스를 노출합니다. 외부 IP 주소가 노드로 적절하게 라우팅되고 있는지 확인해야 합니다.

## 경로 만들기

경로(보안 또는 비보안)를 만드는 가장 쉬운 선호 방법은 `service`가 서비스에 해당하는 경우 `oc expose service service 명령`을 사용하는 것입니다. `--hostname` 옵션을 사용하여 경로에 대한 사용자 지정 호스트 이름을 제공합니다.

```
[user@demo ~]$ oc expose service api-frontend \
> --hostname api.apps.acme.com
```

호스트 이름을 생략하면 OpenShift는 <route-name>-<project-name>.default-domain이라는 구조를 사용하여 호스트 이름을 생성합니다.

예를 들어 **apps.example.com**을 와일드카드 도메인으로 사용하는 클러스터에서 **api** 프로젝트에 **프론트엔드** 경로를 만들면 경로 호스트 이름은 다음과 같이 됩니다.

frontend.api.apps.example.com.



### 중요

와일드카드 도메인을 호스트하는 DNS 서버는 경로 호스트 이름을 인식하지 못합니다. 구성된 IP에 대한 이름만 확인합니다. OpenShift 라우터만 경로 호스트 이름을 알고 있으며, 각각을 HTTP 가상 호스트로 처리합니다.

올바르지 않은 와일드카드 도메인 호스트 이름(즉, 해당하는 경로가 없는 호스트 이름)은 OpenShift 라우터에서 차단하며 HTTP 404 오류가 발생합니다.

경로를 생성할 때는 다음 설정을 사용하는 것이 좋습니다.

- 서비스의 이름. 경로는 서비스를 사용하여 트래픽을 라우팅할 포드를 결정합니다.
- 경로에 대한 호스트 이름. 경로는 항상 클러스터 와일드카드 도메인의 하위 도메인입니다. 예를 들어 **apps.dev-cluster.acme.com**의 와일드카드 도메인을 사용 중이며 경로를 통해 프론트엔드 서비스를 노출해야 하는 경우 다음과 같은 이름이 지정됩니다.

**frontend.apps.dev-cluster.acme.com**



### 참고

또한 OpenShift를 사용하여 경로에 대한 호스트 이름을 자동으로 생성할 수 있습니다.

- 경로 기반 경로에 대한 선택적 경로.
- 애플리케이션에서 수신 대기하는 대상 포트입니다. 대상 포트는 일반적으로 서비스의 **targetPort** 키에 정의된 포트에 해당합니다.
- 보안 또는 비보안 경로가 필요한지 여부에 따른 암호화 전략.

다음 목록은 경로에 대한 최소 정의를 보여줍니다.

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
 name: a-simple-route ❶
 labels: ❷
 app: API
 name: api-frontend
spec:
 host: api.apps.acme.com ❸
 to:
 kind: Service
 name: api-frontend ❹
 port: ❺
 targetPort: 8443
```

**❶** 경로의 이름. 이 이름은 고유해야 합니다.

**❷** 선택기로 사용할 수 있는 레이블 집합.

- ❸ 경로의 호스트 이름. OpenShift는 와일드카드 도메인을 라우터로 라우팅하므로 이 호스트 이름은 와일드카드 도메인의 하위 도메인이어야 합니다.
- ❹ 트래픽을 리디렉션할 서비스. 서비스 이름을 사용하더라도 경로는 이 정보만 사용하여 트래픽을 수신하는 포드 목록을 결정합니다.
- ❺ 애플리케이션 포트. 경로는 서비스를 우회하므로 이는 서비스 포트가 아니라 애플리케이션 포트와 일치해야 합니다.

## 경로 보안 설정

경로에는 보안 또는 비보안 경로가 있습니다. 보안 경로는 여러 유형의 TLS 종료를 사용하여 클라이언트에 인증서를 제공할 수 있습니다. 비보안 경로는 키 또는 인증서가 필요 없기 때문에 구성하기 가장 간단한 반면, 보안 경로는 포드의 트래픽을 암호화합니다.

보안 경로는 해당 경로의 TLS 종료를 지정합니다. 사용 가능한 종료 유형은 다음 목록에 나와 있습니다.

### OpenShift 보안 경로

#### 에지

에지 종료를 사용하면 트래픽을 포드로 라우팅하기 전에 라우터에서 TLS가 종료됩니다. 라우터는 TLS 인증서를 제공하므로 경로에 구성해야 합니다. 그렇지 않은 경우 OpenShift는 TLS 종료를 위해 라우터에 자체 인증서를 할당합니다. TLS가 라우터에서 종료되므로 내부 네트워크를 통한 라우터와 엔드포인트 간 연결이 암호화되지 않습니다.

#### 통과

패스스루 종료를 사용하면 라우터에서 TLS 종료를 제공하지 않고 암호화된 트래픽이 대상 포드로 바로 전송됩니다. 이 모드에서 애플리케이션은 트래픽에 대한 인증서 제공을 담당합니다. 현재 애플리케이션과 애플리케이션에 액세스하는 클라이언트 간의 상호 인증을 지원하는 방법은 패스스루뿐입니다.

#### 재암호화

재암호화는 에지 종료의 변형으로, 라우터에서 인증서가 있는 TLS를 종료한 다음 다른 인증서가 있을 수 있는 엔드포인트에 대한 연결을 재암호화합니다. 따라서 내부 네트워크를 통해서도 전체 연결 경로가 암호화됩니다. 라우터는 상태 점검을 사용하여 호스트 신뢰성을 확인합니다.

## 에지 경로를 사용하여 애플리케이션 보안 설정

보안 경로를 만들기 전에 TLS 인증서를 생성해야 합니다. 다음 명령은 TLS 인증서를 사용하여 보안 에지 경로를 만드는 방법을 보여줍니다.

--key 옵션에는 인증서 개인 키가 필요하며 --cert 옵션에는 해당 키로 서명된 인증서가 필요합니다.

```
[user@demo ~]$ oc create route edge \
> --service api-frontend --hostname api.apps.acme.com \
> --key api.key --cert api.crt
```

에지 모드에서 경로를 사용하는 경우 클라이언트와 라우터 간의 트래픽은 암호화되지만 라우터와 애플리케이션 간의 트래픽은 암호화되지 않습니다.

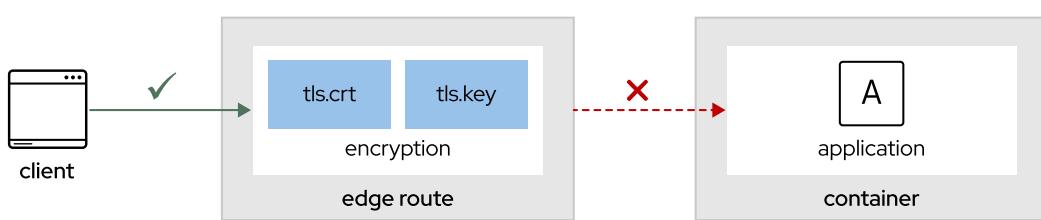


그림 12.6: 에지 경로를 사용하여 애플리케이션 보안 설정

**참고**

네트워크 정책은 애플리케이션 간에 또는 프로젝트 간에 내부 트래픽을 보호하는 데 도움이 될 수 있습니다. 이 작업을 수행하는 방법에 대한 자세한 내용은 참조 섹션의 활동 중인 네트워크 정책 오브젝트를 참조하십시오.

## 패스스루 경로를 사용하여 애플리케이션 보안 설정

이전 예에서는 에지에서 인증서를 제공하는 OpenShift 경로인 에지 경로를 만드는 방법을 보여 줍니다. 패스스루 경로는 애플리케이션에서 TLS 인증서를 노출하므로 보안 설정의 대안이 됩니다. 따라서 클라이언트와 애플리케이션 간에 트래픽이 암호화됩니다.

패스스루 경로를 만들려면 인증서 및 애플리케이션에서 그 인증서에 액세스할 방법이 필요합니다. 이 작업을 수행하는 가장 좋은 방법은 OpenShift TLS 시크릿을 사용하는 것입니다. 마운트 지점을 통해 컨테이너에 시크릿이 노출됩니다.

다음 다이어그램에서는 컨테이너에 시크릿 리소스를 마운트하는 방법을 보여줍니다. 그러면 애플리케이션이 인증서에 액세스할 수 있습니다. 이 모드에서는 클라이언트와 라우터 간의 암호화가 없습니다.

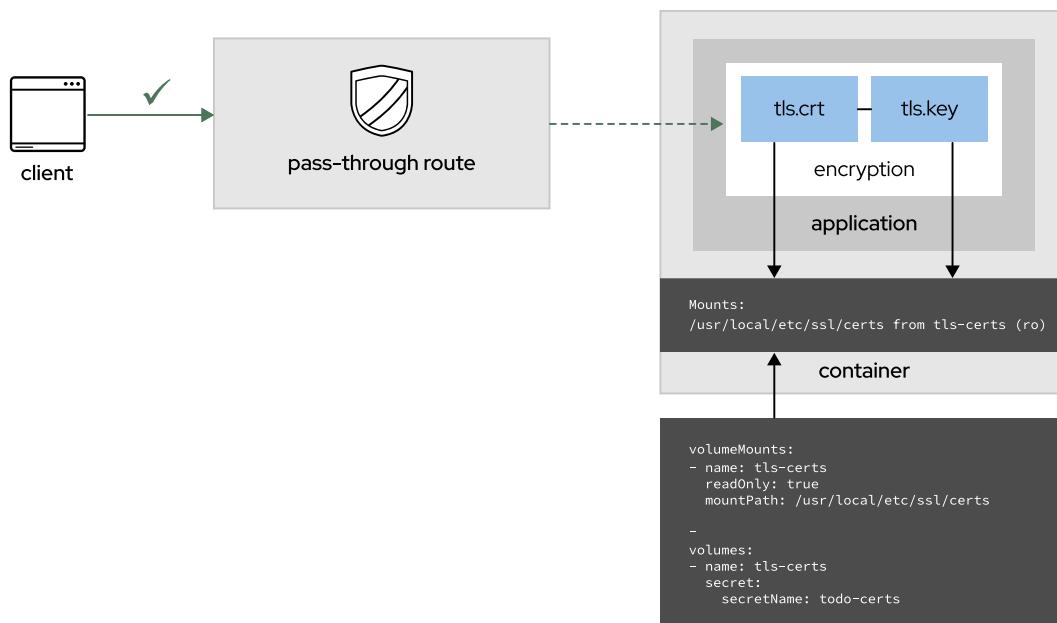


그림 12.7: 패스스루 경로를 사용하여 애플리케이션 보안 설정



### 참조

경로를 관리하는 방법에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/networking/index#configuring-routes](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#configuring-routes)

에 있는 Red Hat OpenShift Container Platform 4.5 네트워킹 설명서의 경로 구성 장을 참조하십시오.

수신 클러스터 트래픽을 구성하는 방법에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/networking/index#configuring-ingress-cluster-traffic](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#configuring-ingress-cluster-traffic)

에 있는 Red Hat OpenShift Container Platform 4.5 네트워킹 설명서의 수신 클러스터 트래픽 구성 장을 참조하십시오.

### 경로

[https://docs.openshift.com/online/pro/dev\\_guide/routes.html](https://docs.openshift.com/online/pro/dev_guide/routes.html)

### Kubernetes 인그레스 vs OpenShift 경로 – Openshift 블로그

<https://blog.openshift.com/kubernetes-ingress-vs-openshift-route/>

### 활동 중인 네트워크 정책 오브젝트 – OpenShift 블로그

<https://blog.openshift.com/network-policy-objects-action/>

## ▶ 연습 가이드

# 외부 액세스를 위해 애플리케이션 노출

이 연습에서는 TLS 인증서로 보호되는 애플리케이션을 노출합니다.

### 결과

다음을 수행할 수 있습니다.

- 애플리케이션을 배포하고 암호화되지 않은 경로를 만듭니다.
- 암호화를 사용하여 OpenShift 에지 경로를 만듭니다.
- 새 버전의 애플리케이션을 지원하도록 OpenShift 배포를 업데이트합니다.
- OpenShift TLS 암호를 만들고 이를 애플리케이션에 마운트합니다.
- 애플리케이션 통신이 암호화되어 있는지 확인합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

명령은 클러스터 API에 연결할 수 있는지 확인하고 **network-ingress** OpenShift 프로젝트를 만듭니다. 또한 **developer** 사용자에게 프로젝트에 대한 편집 액세스 권한을 제공합니다.

```
[student@workstation ~]$ lab network-ingress start
```

애플리케이션 개발자는 OpenShift에 애플리케이션을 배포할 준비가 되었습니다. 이 활동에서는 암호화되지 않은 트래픽(HTTP)을 통해 노출되는 두 가지 버전의 애플리케이션을 배포하고 보안 트래픽을 통해 노출되는 애플리케이션을 배포합니다.

<https://quay.io/redhattraining/todo-angular>에서 액세스할 수 있는 컨테이너 이미지에는 두 개의 태그인 **v1.1**(애플리케이션의 비보안 버전)과 **v1.2**(보안 버전)가 있습니다. 해당 조직은 **\*.apps.ocp4.example.com** 및 **\*.ocp4.example.com** 도메인에 대한 인증서에 서명할 수 있는 자체 CA(인증 기관)를 사용합니다.

CA 인증서는 **~/D0280/labs/network-ingress/certs/training-CA.pem**에서 액세스할 수 있습니다. **passphrase.txt**에는 CA 키를 보호하는 고유한 암호가 포함되어 있습니다. 인증서 폴더에는 CA 키도 포함되어 있습니다.

#### ▶ 1. OpenShift 클러스터에 로그인하고 **network-ingress** 프로젝트를 만듭니다.

##### 1.1. 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

1.2. `network-ingress` 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project network-ingress
Now using project "network-ingress" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- ▶ 2. 애플리케이션용 OpenShift 배포 파일은 `~/DO280/labs/network-ingress/todo-app-v1.yaml`에서 액세스할 수 있습니다. 배포는 애플리케이션의 초기 및 암호화되지 않은 버전인 `quay.io/redhattraining/todo-angular:v1.1`을 가리킵니다. 파일은 애플리케이션 포드를 가리키는 `todo-http` 서비스를 정의합니다.

애플리케이션을 만들고 서비스를 노출합니다.

- 2.1. `oc create` 명령을 사용하여 `network-ingress` OpenShift 프로젝트에 애플리케이션을 배포합니다.

```
[student@workstation ~]$ oc create -f \
> ~/DO280/labs/network-ingress/todo-app-v1.yaml
deployment.apps/todo-http created
service/todo-http created
```

- 2.2. 몇 분 정도 기다리면 애플리케이션이 시작될 수 있으며 프로젝트의 리소스를 검토합니다.

```
[student@workstation ~]$ oc status
In project network-ingress on server https://api.ocp4.example.com:6443

svc/todo-http - 172.30.247.75:80 -> 8080
 deployment/todo-http deploys quay.io/redhattraining/todo-angular:v1.1
 deployment #1 running for 16 seconds - 1 pod
...output omitted...
```

- 2.3. `oc expose` 명령을 실행하여 애플리케이션에 액세스할 수 있는 경로를 만듭니다. 경로에 `todo-http.apps.ocp4.example.com`의 호스트 이름을 지정합니다.

```
[student@workstation ~]$ oc expose svc todo-http \
> --hostname todo-http.apps.ocp4.example.com
route.route.openshift.io/todo-http exposed
```

- 2.4. 경로 이름을 검색하여 클립보드에 복사합니다.

```
[student@workstation ~]$ oc get routes
NAME HOST/PORT PATH SERVICES PORT ...
todo-http todo-http.apps.ocp4.example.com todo-http 8080 ...
```

- 2.5. `workstation` 시스템에서 Firefox를 열고 `http://todo-http.apps.ocp4.example.com`에 액세스합니다.

애플리케이션이 표시되는지 확인합니다.

- 2.6. 새 터미널 탭을 열고 다음 옵션과 함께 `tcpdump` 명령을 실행하여 포트 80의 트래픽을 가로챕니다.

- `-i eth0`는 기본 인터페이스에서 트래픽을 가로챕니다.

- **-A**는 헤더를 제거하고 ASCII 형식으로 패킷을 인쇄합니다.
- **-n**은 DNS 확인을 비활성화합니다.
- **포트 80**은 애플리케이션 포트입니다.

선택적으로 **grep** 명령을 사용하면 JavaScript 리소스에 대한 필터링을 수행할 수 있습니다. IP가 **172.25.250.9**인 기본 인터페이스의 이름을 검색하여 시작합니다.

```
[student@workstation ~]$ ip a | grep 172.25.250.9
inet 172.25.250.9/24 brd 172.25.250.255 scope global noprefixroute eth0
[student@workstation ~]$ sudo tcpdump -i eth0 -A \
> -n port 80 | grep js
```



### 참고

전체 명령은 `~/D0280/labs/network-ingress/tcpdump-command.txt`에서 사용할 수 있습니다.

- 2.7. Firefox에서 페이지를 새로 고치고 터미널에서 활동을 확인합니다. **Ctrl+C**를 눌러 캡처를 중지합니다.

```
...output omitted...
 toBe('Pretty text with some links: http://angularjs.org/',
us@somewhere.org, ' +
 toBe('Pretty text with some links: http://angularjs.org/',
mailto:us@somewhere.org, ' +
 toBe('http://angularjs.org/');
...output omitted...
/*jshint validthis: true */
/*jshint validthis: true */
...output omitted...
```

3. 보안 에지 경로를 만듭니다. 에지 인증서는 클라이언트와 라우터 간의 트래픽을 암호화하지만 라우터와 서비스 간의 트래픽을 암호화되지 않은 상태로 둡니다. OpenShift는 CA에 서명하는 자체 인증서를 생성합니다.

이후 단계에서는 CA를 추출하여 경로 인증서에 서명되었는지 확인합니다.

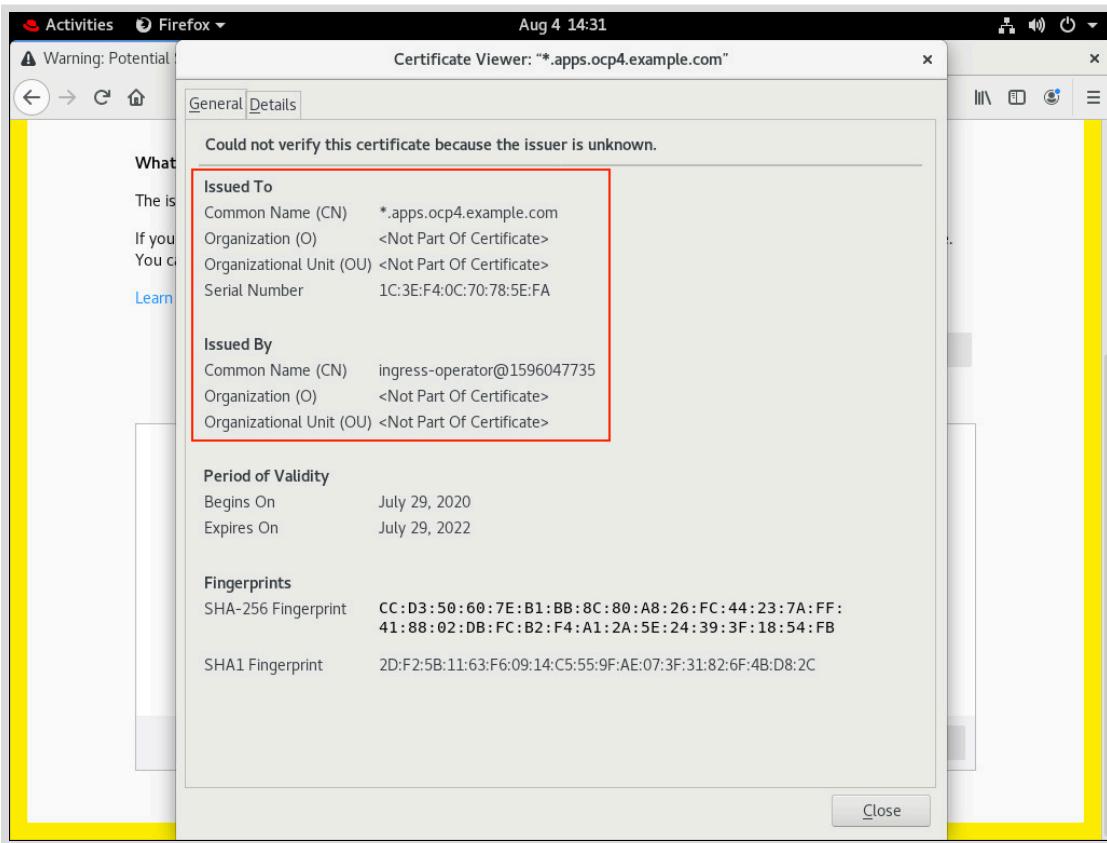
- 3.1. `~/D0280/labs/network-ingress`로 이동하고 **oc create route** 명령을 실행하여 새 경로를 정의합니다.

경로에 `todo-https.apps.ocp4.example.com`의 호스트 이름을 지정합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/network-ingress
[student@workstation network-ingress]$ oc create route edge todo-https \
> --service todo-http \
> --hostname todo-https.apps.ocp4.example.com
route.route.openshift.io/todo-https created
```

- 3.2. 경로를 테스트하고 인증서를 읽으려면 Firefox를 열고 `https://todo-https.apps.ocp4.example.com`에 액세스합니다. 첫 번째 액세스 시에는 Firefox에서 인증서에 대한 경고를 표시합니다. **Advanced(고급)**를 클릭하고 아래로 스크롤한 다음 **View Certificate(인증서 확인)**를 클릭하여 인증서를 확인합니다.

OpenShift 인그레스 운영자가 자체 CA를 사용하여 인증서를 생성했음을 나타내는 CN 항목을 찾습니다.



### 3.3. curl 명령을 사용하여 인증서 거부를 다시 확인합니다.

```
[student@workstation network-ingress]$ curl \
> https://todo-https.apps.ocp4.example.com
curl: (60) SSL certificate problem: self signed certificate in certificate chain
...output omitted...
```

- 3.4. 인증서가 OpenShift로 서명되는 방식을 확인하는 한 가지 방법은 인그레스 운영자가 사용하는 CA를 검색하는 것입니다. 이를 통해 CA에 대한 예지 인증서의 유효성을 검사할 수 있습니다.

**workstation** 시스템에서 클러스터에 **admin** 사용자로 로그인합니다. **developer** 사용자는 CA 검색을 허용하지 않습니다.



#### 중요

실제 시나리오에서 네임스페이스에 대한 액세스 권한이 있는 관리자는 해당 CA를 검색하고 개발자에게 제공합니다.

```
[student@workstation network-ingress]$ oc login -u admin -p redhat
```

- 3.5. **oc extract** 명령을 실행하여 **openshift-ingress-operator** 네임스페이스에 있는 CA를 검색합니다.

```
[student@workstation network-ingress]$ oc extract secrets/router-ca \
> --keys tls.crt -n openshift-ingress-operator
tls.crt
```

- 3.6. 터미널에서 **curl** 명령을 사용하여 연결 헤더를 검색합니다. **--cacert** 옵션을 사용하여 CA 를 **curl** 명령에 전달합니다.

서버 인증서 섹션에는 인증서에 대한 일부 정보가 표시되며 대체 이름은 경로 이름과 일치합니다.

```
[student@workstation network-ingress]$ curl -I -v \
> --cacert tls.crt https://todo-https.apps.ocp4.example.com
...output omitted...
* Server certificate:
* subject: CN=*.apps.ocp4.example.com
* start date: Jul 29 18:35:37 2020 GMT
* expire date: Jul 29 18:35:38 2022 GMT
* subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's
"*.apps.ocp4.example.com"
* issuer: CN=ingress-operator@1596047735
* SSL certificate verify ok.
...output omitted...
```

출력은 이 CA와 일치하므로 원격 인증서를 신뢰함을 나타냅니다.

- 3.7. **developer** 사용자로 다시 로그인합니다.

```
[student@workstation network-ingress]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 3.8. 트래픽이 인증서를 사용하여 에지에서 암호화되지만 서비스 수준에서 비보안 트래픽에 계속 액세스할 수 있습니다. 서비스에 사용되는 포드에서 암호화된 경로를 제공하지 않기 때문입니다.

**todo-http** 서비스의 IP 주소를 검색합니다.

```
[student@workstation network-ingress]$ oc get svc todo-http \
> -o jsonpath=".spec.clusterIP{`\n'}"
172.30.102.29
```

- 3.9. **todo-http** 배포에서 디버그 포드를 만듭니다. 컨테이너와 상호 작용하는 몇 가지 기본 도구를 포함하는 Red Hat UBI(Universal Base Image)를 사용합니다.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-http \
> --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/todo-http-debug ...
Pod IP: 10.131.0.255
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

- 3.10. 디버그 포드에서 **curl** 명령을 사용하여 HTTP를 통해 서비스에 액세스합니다. IP 주소를 이전 단계에서 가져온 항목으로 바꿉니다.

출력은 HTTP를 통해 애플리케이션을 사용할 수 있음을 나타냅니다.

```
sh-4.4$ curl -v 172.30.102.29
* Rebuilt URL to: 172.30.102.29/
* Trying 172.30.102.29...
* TCP_NODELAY set
* Connected to 172.30.102.29 (172.30.102.29) port 80 (#0)
> GET / HTTP/1.1
> Host: 172.30.102.29
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
...output omitted...
```

3.11. 디버그 포드를 종료합니다.

```
sh-4.4$ exit
Removing debug pod ...
```

3.12. 에지 경로를 삭제합니다. 다음 단계에서는 패스스루 경로를 정의합니다.

```
[student@workstation network-ingress]$ oc delete route todo-https
route.route.openshift.io "todo-https" deleted
```

#### ▶ 4. 애플리케이션용 TLS 인증서를 생성합니다.

다음 단계에서는 포드에 시크릿으로 연결하는 CA 서명 인증서를 생성합니다. 그런 다음 패스스루 모드에서 보안 경로를 구성하고 애플리케이션에서 해당 인증서를 노출하도록 설정합니다.

4.1. `~/DO280/labs/network-ingress/certs` 디렉터리로 이동하고 파일을 나열합니다.

```
[student@workstation network-ingress]$ cd certs
[student@workstation certs]$ ls -l
total 16
-rw-rw-r--. 1 student student 604 Nov 29 17:35 openssl-commands.txt
-rw-r--r--. 1 student student 33 Nov 29 17:35 passphrase.txt
-rw-r--r--. 1 student student 1743 Nov 29 17:35 training-CA.key
-rw-r--r--. 1 student student 1363 Nov 29 17:35 training-CA.pem
-rw-r--r--. 1 student student 406 Nov 29 17:35 training.ext
```

4.2. CA 서명 인증서에 대한 개인 키를 생성합니다.



#### 참고

서명된 인증서를 생성하는 다음 명령을 디렉터리에서 사용할 수 있는 `openssl-commands.txt` 파일에서 모두 사용할 수 있습니다.

```
[student@workstation certs]$ openssl genrsa -out training.key 2048
Generating RSA private key, 2048 bit long modulus
.....+...
.....+++
.....+...
e is 65537 (0x10001)
```

- 4.3. `todo-https.apps.ocp4.example.com`에 대한 CSR(인증서 서명 요청)을 생성합니다. 요청의 제목을 한 줄에 입력해야 합니다. 또는 `-subj` 옵션과 해당 콘텐츠를 제거합니다. `-subj` 옵션을 사용하지 않으면 `openssl` 명령에서 값을 입력하라는 메시지가 표시됩니다. `todo-https.apps.ocp4.example.com`의 CN(일반 이름)을 표시해야 합니다.

```
[student@workstation certs]$ openssl req -new \
> -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/" \
> CN=todo-https.apps.ocp4.example.com" \
> -key training.key -out training.csr
```

- 4.4. 마지막으로 서명된 인증서를 생성합니다. CA에 대한 인증서에 서명하기 위해 `-CA` 및 `-CAkey` 옵션을 사용함을 확인합니다. `-passin` 옵션을 사용하면 CA의 암호를 재사용할 수 있습니다. `extfile` 옵션을 사용하면 SAN(주체 대체 이름)을 정의할 수 있습니다.

```
[student@workstation certs]$ openssl x509 -req -in training.csr \
> -passin file:passphrase.txt \
> -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
> -out training.crt -days 1825 -sha256 -extfile training.ext
Signature ok
subject=C = US, ST = North Carolina, L = Raleigh, O = Red Hat, CN = todo-
https.apps.ocp4.example.com
Getting CA Private Key
```

- 4.5. 새로 생성된 인증서와 키가 현재 디렉터리에 있는지 확인합니다.

```
[student@workstation certs]$ ls -l
total 36
-rw-r--r-- 1 student student 599 Jul 31 09:35 openssl-commands.txt
-rw-r--r-- 1 student student 33 Aug 3 12:38 passphrase.txt
-rw----- 1 student student 1743 Aug 3 12:38 training-CA.key
-rw-r--r-- 1 student student 1334 Aug 3 12:38 training-CA.pem
-rw-rw-r-- 1 student student 41 Aug 3 13:40 training-CA.srl
-rw-rw-r-- 1 student student 1391 Aug 3 13:40 training.crt
-rw-rw-r-- 1 student student 1017 Aug 3 13:39 training.csr
-rw-r--r-- 1 student student 352 Aug 3 12:38 training.ext
-rw----- 1 student student 1675 Aug 3 13:38 training.key
```

- 4.6. `network-ingress` 디렉터리로 돌아갑니다. 다음 단계에서는 자체 서명 인증서를 사용하여 경로를 생성하는 작업이 포함되어 있어 중요합니다.

```
[student@workstation certs]$ cd ~/D0280/labs/network-ingress
```

- ▶ 5. 새 버전의 애플리케이션을 배포합니다. 새 버전의 애플리케이션에서는 `/usr/local/etc/ssl/certs`의 컨테이너 내에 인증서와 키가 필요합니다. 해당 버전의 웹 서버는 SSL 지원으로 구성됩니다. 시크릿을 만들어 `workstation` 시스템에서 인증서를 가져옵니다. 이후 단계에서 애플리케이션 배포는 해당 시크릿을 요청하고 `/usr/local/etc/ssl/certs`의 컨테이너에 해당 콘텐츠를 노출합니다.

- 5.1. `todo-certs`라는 tls OpenShift 시크릿을 만듭니다. `--cert` 및 `key` 옵션을 사용하여 TLS 인증서를 포함합니다. `training.csr`을 인증서로 사용하고 `training.key`를 키로 전달합니다.

```
[student@workstation network-ingress]$ oc create secret tls todo-certs \
> --cert certs/training.crt \
> --key certs/training.key
secret/todo-certs created
```

- 5.2. `~/DO280/labs/network-ingress/todo-app-v2.yaml`에서 액세스할 수 있는 배포 파일은 컨테이너 이미지의 버전 2를 가리킵니다. 새 버전의 애플리케이션은 SSL 인증서를 지원하도록 구성됩니다. 해당 이미지를 사용하여 새 배포를 만들도록 `oc create`를 실행합니다.

```
[student@workstation network-ingress]$ oc create -f todo-app-v2.yaml
deployment.apps/todo-https created
service/todo-https created
```

- 5.3. 몇 분 정도 기다린 후 애플리케이션 포드가 실행되고 있는지 확인합니다. 포드 이름을 클립보드에 복사합니다.

```
[student@workstation network-ingress]$ oc get pods
NAME READY STATUS RESTARTS AGE
...output omitted...
todo-https-59d8fc9d47-265ds 1/1 Running 0 62s
```

- 5.4. 포드 내부에 마운트된 볼륨을 검토합니다. 출력은 인증서가 `/usr/local/etc/ssl/certs`에 마운트된다는 것을 나타냅니다.

```
[student@workstation network-ingress]$ oc describe pod \
> todo-https-59d8fc9d47-265ds | grep Mounts -A2
Mounts:
 /usr/local/etc/ssl/certs from tls-certs (ro)
 /var/run/secrets/kubernetes.io/serviceaccount from default-token-prz4k (ro)
```

- ▶ 6. 보안 경로를 만듭니다.

- 6.1. `oc create route` 명령을 실행하여 새 경로를 정의합니다.

경로에 `todo-https.apps.ocp4.example.com`의 호스트 이름을 지정합니다.

```
[student@workstation network-ingress]$ oc create route passthrough todo-https \
> --service todo-https --port 8443 \
> --hostname todo-https.apps.ocp4.example.com
route.route.openshift.io/todo-https created
```

- 6.2. 자세한 정보 표시 모드에서 **curl** 명령을 사용하여 경로를 테스트하고 인증서를 읽습니다. --cacert 옵션을 사용하여 CA 인증서를 **curl** 명령에 전달합니다.

출력은 인증서 체인과 애플리케이션 인증서 간에 일치함을 나타냅니다. 이러한 일치는 OpenShift 라우터가 애플리케이션 웹 서버 인증서로 암호화된 패킷만 전달함을 나타냅니다.

```
[student@workstation network-ingress]$ curl -vVI \
> --cacert certs/training-CA.pem \
> https://todo-https.apps.ocp4.example.com
...output omitted...
* Server certificate:
* subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=todo-
https.apps.ocp4.example.com
* start date: Aug 3 17:40:30 2020 GMT
* expire date: Aug 2 17:40:30 2025 GMT
* subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's
"*.apps.ocp4.example.com"
* issuer: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=ocp4.example.com
* SSL certificate verify ok.
...output omitted...
```

- ▶ 7. 새 디버그 포드를 만들어 서비스 수준에서 적절한 암호화를 추가로 확인합니다.

- 7.1. **todo-https** 서비스의 IP 주소를 검색합니다.

```
[student@workstation network-ingress]$ oc get svc todo-https \
> -o jsonpath='{.spec.clusterIP}{'\n'}'
172.30.121.154
```

- 7.2. **todo-https** 배포에서 Red Hat UBI를 사용하여 디버그 포드를 만듭니다.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-https \
> --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/todo-https-debug ...
Pod IP: 10.128.2.129
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

- 7.3. 디버그 포드에서 **curl** 명령을 사용하여 HTTP를 통해 서비스에 액세스합니다. IP 주소를 이전 단계에서 가져온 항목으로 바꿉니다.

출력은 HTTP를 통해 애플리케이션을 사용할 수 없음을 나타내며 웹 서버는 사용자를 보안 버전으로 리디렉션합니다.

```
sh-4.4$ curl -I http://172.30.121.154
HTTP/1.1 301 Moved Permanently
Server: nginx/1.14.1
Date: Sat, Thu, 20 Aug 2020 00:01:35 GMT
Content-Type: text/html
Connection: keep-alive
Location: https://172.30.121.154:8443/
```

- 7.4. 마지막으로 HTTPS를 통해 애플리케이션에 액세스합니다. 컨테이너에 CA 인증서에 대한 액세스 권한이 없으므로 **-k** 옵션을 사용합니다.

```
sh-4.4$ curl -s -k https://172.30.121.154:8443 | head -n5
<!DOCTYPE html>
<html lang="en" ng-app="todoItemsApp" ng-controller="appCtl">
<head>
 <meta charset="utf-8">
 <title>ToDo app</title>
```

- 7.5. 디버그 포드를 종료합니다.

```
sh-4.4$ exit
Removing debug pod ...
```

▶ 8. 홈 디렉터리로 이동하여 **network-ingress** 프로젝트를 삭제합니다.

```
[student@workstation network-ingress]$ cd
[student@workstation ~]$ oc delete project network-ingress
project.project.openshift.io "network-ingress" deleted
```

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab network-ingress finish
```

이로써 안내에 따른 연습이 완료됩니다.

# 네트워크 정책 구성

## 목표

이 섹션을 마치면 프로젝트와 포드 간 네트워크 트래픽을 제한할 수 있습니다.

## OpenShift에서 네트워크 정책 관리

네트워크 정책을 사용하면 개별 포드에 대한 격리 정책을 구성할 수 있습니다. 네트워크 정책에는 관리자 권한이 필요하지 않으므로 개발자에게 프로젝트의 애플리케이션을 제어할 수 있는 권한이 더 많이 제공됩니다.

네트워크 정책을 사용하여 조직 네트워크 영역에 매핑되는 SDN에 논리 영역을 만들 수 있습니다. 이 접근 방식의 이점은 발생한 위치와 관계없이 네트워크 정책을 통해 트래픽을 분리할 수 있기 때문에 실행 중인 포드의 위치는 관련이 없다는 점입니다.

두 네임스페이스 간 네트워크 통신을 관리하려면 다른 네임스페이스에 액세스해야 하는 네임스페이스에 레이블을 할당합니다. 다음 명령에서는 **name=network-1** 레이블을 **network-1** 네임스페이스에 할당합니다.

```
[user@demo ~]$ oc label namespace network-1 name=network-1
```

다음 예제에서는 **network-1** 및 **network-2** 네임스페이스 간 통신을 허용하는 네트워크 정책을 설명합니다.

- 다음 네트워크 정책은 **network-1** 네임스페이스에서 **deployment="product-catalog"** 레이블이 있는 모든 포드에 적용됩니다. 이 정책은 **network-2** 네임스페이스에서 레이블이 **role="qa"**인 포드에서 포트 8080을 통한 TCP 트래픽을 허용합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: network-1-policy
spec:
 podSelector: ①
 matchLabels:
 deployment: product-catalog

 ingress: ②
 - from: ③
 - namespaceSelector:
 matchLabels:
 name: network-2
 podSelector:
 matchLabels:
 role: qa
 ports: ④
 - port: 8080
 protocol: TCP
```

- ❶ 최상위의 **podSelector** 필드는 필수이며 네트워크 정책을 사용하는 포드를 정의합니다.  
**podSelector**가 비어 있는 경우 네임스페이스의 모든 포드가 일치합니다.
  - ❷ **ingress** 필드에서는 최상위 **podSelector**에서 일치하는 포드에 적용할 인그레스 트래픽 규칙 목록을 정의합니다.
  - ❸ **from** 필드는 모든 소스에서 발생하는 트래픽과 일치하는 규칙 목록을 정의합니다. 선택기는 네트워크 정책이 정의된 프로젝트에 국한되지 않습니다.
  - ❹ **ports** 필드는 선택한 포드에 트래픽이 도달하도록 허용하는 대상 포트 목록입니다.
- 다음 네트워크 정책에서는 **network-1** 네임스페이스의 모든 포드 및 포트에서 **network-2** 네임스페이스의 모든 포드 및 포트로의 트래픽을 허용합니다. 이 정책은 **network-1** 네임스페이스의 모든 포드에서 발생하는 트래픽을 제한하지 않으므로 **network-1** 정책보다 덜 제한적입니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: network-2-policy
spec:
 podSelector: {}

 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 name: network-1
```



### 참고

네트워크 정책은 Kubernetes 리소스입니다. 따라서 **oc** 명령을 사용하여 관리할 수 있습니다.

네트워크 정책 사용의 한 가지 이점은 프로젝트(테넌트) 간 보안 관리이며 VLAN과 같은 계층 2 기술로는 수행 할 수 없습니다. 이러한 접근 방식을 통해 프로젝트 간 맞춤형 정책을 생성하여 사용자가 필요한 항목에만 액세스하도록 할 수 있습니다(최소 권한 접근법 준수).

오브젝트 목록을 가져오는 네트워크 정책의 필드는 동일한 오브젝트로 결합되거나 여러 오브젝트로 표시될 수 있습니다. 결합되는 경우 조건은 논리적 AND로 결합됩니다. 목록에 분리되어 있는 경우 조건은 논리적 OR로 결합됩니다. 논리적 옵션을 사용하면 매우 구체적인 정책 규칙을 생성할 수 있습니다. 다음 예제에는 구문의 다른 부분이 강조 표시되어 있습니다.

- 이 예제에서는 선택기를 하나의 규칙에 결합하므로 **app=mobile** 레이블이 있는 **dev** 네임스페이스의 포드에서만 액세스할 수 있습니다. 이는 논리적 AND의 예입니다.

```
...output omitted...
ingress:
- from:
 - namespaceSelector:
 matchLabels:
 name: dev
 podSelector:
 matchLabels:
 app: mobile
```

- 위 예제의 **podSelector** 필드를 **from** 목록의 항목으로 변경하면 **dev** 네임스페이스의 모든 포드 및 **app=mobile** 레이블이 지정된 네임스페이스의 모든 포드가 최상위 **podSelector** 필드와 일치하는 포드에 접근할 수 있습니다. 이는 논리적 OR의 예입니다.

```
...output omitted...
ingress:
- from:
 - namespaceSelector:
 matchLabels:
 name: dev
 - podSelector:
 matchLabels:
 app: mobile
```

포드가 하나 이상의 네트워크 정책에 있는 선택기와 일치하는 경우 포드는 해당 네트워크 정책 중 하나 이상에서 허용하는 연결만 수락합니다. 엄격한 예로는 프로젝트 내부의 다른 포드를 포함하여 프로젝트의 포드로 향하는 모든 인그레스 트래픽을 거부하는 정책이 있습니다. 빈 포드 선택기는 이 정책이 이 프로젝트의 모든 포드에 적용됨을 의미합니다. 다음 정책은 인그레스 규칙이 정의되지 않았기 때문에 모든 트래픽을 차단합니다. 이 기본 동작을 무시하는 명시적 정책을 정의하지 않는 한 트래픽이 차단됩니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: default-deny
spec:
 podSelector: {}
```

클러스터 모니터링 또는 노출된 경로가 있는 경우 해당 경로에서 오는 인그레스도 허용해야 합니다. 다음 정책을 사용하면 OpenShift 모니터링 및 인그레스 컨트롤러에서 수신할 수 있습니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-from-openshift-ingress
spec:
 podSelector: {}
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 network.openshift.io/policy-group: ingress

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-from-openshift-monitoring
spec:
 podSelector: {}
 ingress:
 - from:
```

```
- namespaceSelector:
 matchLabels:
 network.openshift.io/policy-group: monitoring
```



### 중요

`default` 인그레스 컨트롤러에서 `HostNetwork` 엔드포인트 게시 전략을 사용하는 경우,  
`default` 네임스페이스에 `network.openshift.io/policy-group=ingress` 레이블이 있어야 합니다.

`openshift-ingress-controller` 네임스페이스의 `ingresscontroller/default` 리소스를 설명하는 `oc describe` 명령을 사용하여 엔드포인트 게시 전략을 확인하십시오.

자세한 내용은 아래의 참조 자료에 링크된 문서를 참조하십시오.



### 참조

자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 네트워킹 설명서의 네트워크 정책 장을 참조하십시오.

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/networking/index#network-policy](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#network-policy)

## ▶ 연습 가이드

# 네트워크 정책 구성

이 연습에서는 네트워크 정책을 생성하고 이러한 네트워크 정책에 의해 생성된 포드 격리를 검토합니다.

## 결과

다음을 수행할 수 있습니다.

- 포드 간 통신을 제어하는 네트워크 정책을 구성합니다.
- 인그레스 트래픽이 포드로 제한되는지 확인합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령을 사용하면 환경을 준비하고 연습에 필요한 리소스 파일을 다운로드할 수 있습니다.

```
[student@workstation ~]$ lab network-policy start
```

### ▶ 1. OpenShift 클러스터에 로그인하고 **network-policy** 프로젝트를 생성합니다.

- 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- network-policy** 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project network-policy
Now using project "network-policy" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

### ▶ 2. 두 개의 배포를 생성하고 그 중 하나에 대한 경로를 만듭니다.

- oc new-app** 명령과 **quay.io/redhattraining/hello-world-nginx:v1.0** 이미지를 사용하여 두 개의 배포를 만듭니다. 첫 번째 배포는 **hello**로, 두 번째 배포는 **test**로 이름을 지정합니다.

```
[student@workstation ~]$ oc new-app --name hello --docker-image \
> quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
```

```

imagestream.image.openshift.io "hello" created
deployment.apps "hello" created
service "hello" created
--> Success
...output omitted...
[student@workstation ~]$ oc new-app --name test --docker-image \
> quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "test" created
deployment.apps "test" created
service "test" created
--> Success
...output omitted...

```

2.2. `oc expose` 명령을 사용하여 `hello` 서비스에 대한 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service hello
route.route.openshift.io/hello exposed
```

### ▶ 3. `oc rsh` 및 `curl` 명령을 사용하여 `hello` 포드에 대한 액세스를 확인합니다.

3.1. 두 번째 터미널을 열고 `~/DO280/labs/network-policy/display-project-info.sh`에 있는 스크립트를 실행합니다. 이 스크립트는 이 연습의 나머지 부분에서 사용하는 포드, 서비스 및 경로에 대한 정보를 제공합니다.

```
[student@workstation ~]$ ~/DO280/labs/network-policy/display-project-info.sh
=====
PROJECT: network-policy

POD NAME IP ADDRESS
hello-6c4984d949-g28c4 10.8.0.13
test-c4d74c9d5-5pq9s 10.8.0.14

SERVICE NAME CLUSTER-IP
hello 172.30.137.226
test 172.30.159.119

ROUTE NAME HOSTNAME PORT
hello hello-network-policy.apps.ocp4.example.com 8080-tcp
=====
```

3.2. `oc rsh` 및 `curl` 명령을 사용하여 `test` 포드에서 `hello` 포드의 IP 주소에 액세스할 수 있는지 확인합니다.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s curl 10.8.0.13:8080 | \
> grep Hello
<h1>Hello, world from nginx!</h1>
```

3.3. `oc rsh` 및 `curl` 명령을 사용하여 `test` 포드에서 `hello` 서비스의 IP 주소에 액세스할 수 있는지 확인합니다.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s curl 172.30.137.226:8080 | \
> grep Hello
<h1>Hello, world from nginx!</h1>
```

- 3.4. `hello` 경로의 URL에 대해 `curl` 명령을 사용하여 `hello` 포드에 대한 액세스를 확인합니다.

```
[student@workstation ~]$ curl -s hello-network-policy.apps.ocp4.example.com | \
> grep Hello
<h1>Hello, world from nginx!</h1>
```

▶ 4. `network-test` 프로젝트와 `sample-app`이라는 배포를 만듭니다.

- 4.1. `network-test` 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project network-test
Now using project "network-test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 4.2. `quay.io/redhattraining/hello-world-nginx:v1.0` 이미지를 사용하여 `sample-app` 배포를 만듭니다. 웹 앱은 포트 8080에서 수신 대기합니다.

```
[student@workstation ~]$ oc new-app --name sample-app --docker-image \
> quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "sample-app" created
deployment.apps "sample-app" created
service "sample-app" created
--> Success
...output omitted...
```

▶ 5. 다른 네임스페이스의 포드에서 `network-policy` 네임스페이스의 `hello` 및 `test` 포드에 액세스할 수 있는지 확인합니다.

- 5.1. 두 번째 터미널에서 `display-project-info.sh` 스크립트를 다시 실행하여 `sample-app` 포드의 전체 이름을 확인합니다.

```
[student@workstation ~]$ ~/D0280/labs/network-policy/display-project-info.sh
...output omitted...
PROJECT: network-test

POD NAME
sample-app-d5f945-spx9q
=====
```

- 5.2. 첫 번째 터미널로 돌아와 `oc rsh` 및 `curl` 명령을 사용하여 `ample-app` 포드에서 `hello` 포드의 IP 주소에 액세스할 수 있는지 확인합니다.

```
[student@workstation ~]$ oc rsh sample-app-d5f945-spx9q curl 10.8.0.13:8080 | \
> grep Hello
<h1>Hello, world from nginx!</h1>
```

- 5.3. `oc rsh` 및 `curl` 명령을 사용하여 `sample-app` 포드에서 `test` 포드에 액세스할 수 있는지 확인합니다. 이전에 `test` 포드로 검색된 IP 주소를 대상으로 합니다.

```
[student@workstation ~]$ oc rsh sample-app-d5f945-spx9q curl 10.8.0.14:8080 | \
> grep Hello
<h1>Hello, world from nginx!</h1>
```

- ▶ 6. `network-policy` 프로젝트에서 `~/D0280/labs/network-policy/deny-all.yaml`에 있는 리소스 파일을 사용하여 `deny-all` 네트워크 정책을 만듭니다.

- 6.1. `network-policy` 프로젝트로 전환합니다.

```
[student@workstation ~]$ oc project network-policy
Now using project "network-policy" on server "https://api.ocp4.example.com:6443".
```

- 6.2. `~/D0280/labs/network-policy/` 디렉터리로 이동합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/network-policy/
```

- 6.3. 네임스페이스의 모든 포드를 대상으로 하도록 텍스트 편집기에서 빈 `podSelector`를 사용하여 `deny-all.yaml` 파일을 업데이트합니다. 솔루션은 `~/D0280/solutions/network-policy/deny-all.yaml`에 있습니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: deny-all
spec:
 podSelector: {}
```

- 6.4. `oc create` 명령을 사용하여 네트워크 정책을 만듭니다.

```
[student@workstation network-policy]$ oc create -f deny-all.yaml
networkpolicy.networking.k8s.io/deny-all created
```

- ▶ 7. `network-policy` 네임스페이스의 포드에 더 이상 액세스할 수 없는지 확인합니다.

- 7.1. 노출된 경로를 통해 `hello` 포드에 더 이상 액세스할 수 없는지 확인합니다. 몇 초 동안 기다린 후 `Ctrl+C`를 눌러 응답하지 않는 `curl` 명령을 종료합니다.

```
[student@workstation network-policy]$ curl -s \
> hello-network-policy.apps.ocp4.example.com | grep Hello
^C
```

**중요**

`hello` 포드가 `router-default` 포드와 동일한 노드에 있는 경우, 트래픽이 해당 라우터 포드를 통과할 때 `curl` 명령이 작동합니다. 이는 3노드 클러스터의 경우에만 해당합니다. 컨트롤 플레인이나 인프라 노드가 작업자 노드와 분리된 기존의 OpenShift 클러스터에서는 네트워크 정책이 클러스터의 모든 라우터 포드에 적용됩니다.

`curl` 명령이 성공하면 명령을 다시 실행하여 네트워크 정책이 예상대로 작동하는지 확인합니다. 이 두 번째 시도는 클러스터의 다른 라우터 포드를 거쳐야 합니다.

- 7.2. `test` 포드에서 `hello` 포드의 IP 주소에 더 이상 액세스할 수 없는지 확인합니다. 몇 초 동안 기다린 후 `Ctrl+C`를 눌러 응답하지 않는 `curl` 명령을 종료합니다.

```
[student@workstation network-policy]$ oc rsh test-c4d74c9d5-5pq9s curl \
> 10.8.0.13:8080 | grep Hello
^Ccommand terminated with exit code 130
```

- 7.3. `network-test` 프로젝트에서 `sample-app` 포드가 `test` 포드의 IP 주소에 더 이상 액세스할 수 없는지 확인합니다. 몇 초 동안 기다린 후 `Ctrl+C`를 눌러 응답하지 않는 `curl` 명령을 종료합니다.

```
[student@workstation network-policy]$ oc project network-test
Now using project "network-test" on server "https://api.ocp4.example.com:6443".
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
> 10.8.0.14:8080 | grep Hello
^Ccommand terminated with exit code 130
```

- ▶ 8. 포트 8080의 TCP를 통해 `network-test` 네임스페이스의 `sample-app` 포드에서 `network-policy` 네임스페이스의 `hello` 포드로 향하는 트래픽을 허용하는 네트워크 정책을 생성합니다. `~/D0280/labs/network-policy/allow-specific.yaml`에 제공된 리소스 파일을 사용합니다.

- 8.1. 텍스트 편집기를 사용하여 다음과 같이 `allow-specific.yaml` 파일의 `CHANGE_ME` 섹션을 교체합니다. 솔루션은 `~/D0280/solutions/network-policy/allow-specific.yaml`에 있습니다.

```
...output omitted...
spec:
podSelector:
 matchLabels:
 deployment: hello
ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 name: network-test
 podSelector:
 matchLabels:
 deployment: sample-app
```

```
ports:
- port: 8080
 protocol: TCP
```

- 8.2. `oc create` 명령을 사용하여 네트워크 정책을 만듭니다.

```
[student@workstation network-policy]$ oc create -n network-policy -f \
> allow-specific.yaml
networkpolicy.networking.k8s.io/allow-specific created
```

- 8.3. `network-policy` 네임스페이스에서 네트워크 정책을 확인합니다.

```
[student@workstation network-policy]$ oc get networkpolicies -n network-policy
NAME POD-SELECTOR AGE
allow-specific deployment=hello 11s
deny-all <none> 5m6s
```

- ▶ 9. `admin` 사용자로 `network-test` 네임스페이스의 레이블을 `name=network-test`로 지정합니다.

- 9.1. `admin` 사용자로 로그인합니다.

```
[student@workstation network-policy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 9.2. `oc label` 명령을 사용하여 `name=network-test` 레이블을 적용합니다.

```
[student@workstation network-policy]$ oc label namespace network-test \
> name=network-test
namespace/network-test labeled
```



### 중요

`allow-specific` 네트워크 정책에서는 레이블을 사용하여 네임스페이스의 이름을 대조합니다. 기본적으로 네임스페이스와 프로젝트에는 레이블이 자동으로 지정되지 않습니다.

- 9.3. 레이블이 적용되었는지 확인하고 `developer` 사용자로 로그인합니다.

```
[student@workstation network-policy]$ oc describe namespace network-test
Name: network-test
Labels: name=network-test
...output omitted...
[student@workstation network-policy]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- ▶ 10. `sample-app` 포드에서 `hello` 포드의 IP 주소에 액세스할 수 있지만 `test` 포드의 IP 주소에는 액세스할 수 없음을 확인합니다.

- 10.1. `network-policy` 네임스페이스의 `hello` 포드에 액세스할 수 있는지 확인합니다.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
> 10.8.0.13:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

10.2. 다른 포트에서 **hello** 포드의 응답이 없는지 확인합니다. 네트워크 정책에서 **hello** 포드의 포트 8080에 대한 액세스만 허용하므로 다른 포트에 대한 요청이 무시되고 결국 시간이 초과됩니다. 몇 초 동안 기다린 후 **Ctrl+C**를 눌러 응답하지 않는 **curl** 명령을 종료합니다.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
> 10.8.0.13:8181 | grep Hello
^Ccommand terminated with exit code 130
```

10.3. **test** 포드에 대한 액세스 권한이 없는지 확인합니다. 몇 초 동안 기다린 후 **Ctrl+C**를 눌러 응답하지 않는 **curl** 명령을 종료합니다.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
> 10.8.0.14:8080 | grep Hello
^Ccommand terminated with exit code 130
```

▶ 11. 노출된 경로에서 **hello** 포드로 향하는 트래픽을 허용하는 네트워크 정책을 만듭니다. **~/D0280/labs/network-policy/allow-from-openshift-ingress.yaml**에 제공된 리소스 파일을 사용합니다.

11.1. 텍스트 편집기를 사용하여 다음과 같이 **allow-from-openshift-ingress.yaml** 파일의 **CHANGE\_ME** 값을 교체합니다. 솔루션은 **~/D0280/solutions/network-policy/allow-from-openshift-ingress.yaml**에 있습니다.

```
...output omitted...
spec:
 podSelector: {}
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 network.openshift.io/policy-group: ingress
```

11.2. **oc create** 명령을 사용하여 네트워크 정책을 만듭니다.

```
[student@workstation network-policy]$ oc create -n network-policy -f \
> allow-from-openshift-ingress.yaml
networkpolicy.networking.k8s.io/allow-from-openshift-ingress created
```

11.3. **network-policy** 네임스페이스에서 네트워크 정책을 확인합니다.

```
[student@workstation network-policy]$ oc get networkpolicies -n network-policy
NAME POD-SELECTOR AGE
allow-from-openshift-ingress <none> 10s
allow-specific deployment=hello 8m16s
deny-all <none> 13m
```

- 11.4. `admin` 사용자로 `default` 네임스페이스 레이블을 `network.openshift.io/policy-group=ingress`로 지정합니다.

```
[student@workstation network-policy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation network-policy]$ oc label namespace default \
> network.openshift.io/policy-group=ingress
namespace/default labeled
```



### 참고

강의실의 `default` 인그레스 컨트롤러에서는 `HostNetwork` 엔드포인트 게시 전략을 사용하므로 이 레이블은 `default` 네임스페이스에만 적용하면 됩니다.

- 11.5. 노출된 경로를 통해 `hello` 포드에 액세스할 수 있는지 확인합니다.

```
[student@workstation network-policy]$ curl -s \
> hello-network-policy.apps.ocp4.example.com | grep Hello
<h1>Hello, world from nginx!</h1>
```

- ▶ 12. `display-project-info.sh` 스크립트의 출력이 있는 터미널 창을 닫습니다. 홈 디렉터리로 이동합니다.

```
[student@workstation network-policy]$ cd
```

## 완료

`workstation` 시스템에서 `lab` 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab network-policy finish
```

이로써 안내에 따른 연습이 완료됩니다.

## ▶ 랩

# 애플리케이션을 위한 OpenShift 네트워킹 구성

이 랩에서는 애플리케이션을 위한 TLS 패스스루 경로를 구성합니다.

## 결과

다음을 수행할 수 있습니다.

- 애플리케이션을 배포하고 비보안 경로를 구성합니다.
- 애플리케이션으로 향하는 트래픽을 제한합니다.
- 애플리케이션용 TLS 인증서를 생성합니다.
- TLS 인증서를 사용하여 애플리케이션의 패스스루 경로를 구성합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있는지 확인하고 이 랩에서 사용하는 자체 서명 CA(인증 기관)를 만듭니다.

```
[student@workstation ~]$ lab network-review start
```

이 검토에서는 시스템에 대한 일부 정보를 출력하는 PHP 애플리케이션을 배포합니다. 이 애플리케이션은 두 가지 구성으로 사용할 수 있습니다. 하나는 포트 8080에서 수신 대기하는 암호화되지 않은 네트워크로 실행되고, 다른 하나는 TLS 인증서를 사용하여 포트 8443에서 수신 대기하는 네트워크 트래픽을 암호화합니다.

이 검토의 컨테이너 이미지는 [quay.io/redhattraining/php-ssl](https://quay.io/redhattraining/php-ssl)에서 액세스할 수 있습니다. 여기에는 두 개의 태그(애플리케이션의 비보안 버전에 대한 **v1.0** 및 보안 버전의 **v1.1**)가 있습니다.

1. OpenShift **developer** 사용자로, **network-review** 프로젝트를 만듭니다.
2. **developer** 사용자로 **network-review** 프로젝트에 비보안 버전의 PHP 애플리케이션을 배포합니다. [~/D0280/labs/network-review/php-http.yaml](#)에 제공된 리소스 파일을 사용합니다.

애플리케이션을 배포하기 전에 파일에 대한 필수 변경 사항, 구체적으로는 컨테이너 이미지의 위치 및 수신 대기하는 포트를 변경합니다.

애플리케이션을 만든 후 잠시 기다렸다가 하나의 포드가 실행되고 있는지 확인합니다.

3. 호스트 이름이 **php-http.apps.ocp4.example.com**인 **php-http**라는 경로를 만들어 애플리케이션에 액세스합니다.

**workstation** 시스템에서 Firefox를 사용하여 <http://php-http.apps.ocp4.example.com>에 액세스합니다. 다음 단계를 진행하기 전에 애플리케이션 가용성을 확인합니다.

4. 기본적으로 모든 인그레스 트래픽을 거부하도록 **network-review** 네임스페이스에 네트워크 정책을 생성합니다. 네트워크 정책을 올바르게 구성하면 **network-review** 네임스페이스 내의 포드 간 통신도 금지할 수 있습니다.

~/D0280/labs/network-review/deny-all.yaml에 제공된 리소스 파일을 사용합니다. 네임스페이스의 모든 포드를 대상으로 하는 데 필요한 변경 작업을 수행합니다.

5. **network-review** 네임스페이스의 경로로 향하는 인그레스 트래픽을 허용하는 네트워크 정책을 생성합니다.

~/D0280/labs/network-review/allow-from-openshift-ingress.yaml에 제공된 리소스 파일을 사용합니다. 네임스페이스의 모든 포드를 대상으로 하고 기본 수신 컨트롤러의 트래픽을 허용하는 데 필요한 변경 작업을 수행합니다.

강의실 환경에서는 **HostNetwork** 엔드포인트 전략을 사용하므로 **default** 네임스페이스에 **network.openshift.io/policy-group=ingress** 레이블을 지정합니다. 이 작업은 **admin** 사용자로 수행해야 합니다.

6. **developer** 사용자로 암호화된 버전의 애플리케이션에 대한 TLS 인증서를 생성하고 서명합니다.

**php-https.apps.ocp4.example.com** 호스트 이름에 대한 CSR(인증서 서명 요청)을 생성합니다. CSR을 ~/D0280/labs/network-review/certs/training.csr에 저장합니다.

CSR을 사용하여 인증서를 생성하고 ~/D0280/labs/network-review/certs/training.crt에 저장합니다. 인증서를 생성하려면 ~/D0280/labs/network-review/certs/training-CA.pem 및 CSR에서 액세스할 수 있는 CA 인증서를 인수로 전달합니다.

~/D0280/labs/network-review/certs/openssl-commands.txt 텍스트 파일을 사용하여 도움을 받을 수 있습니다. 이 파일에는 인증서 서명 요청 및 인증서를 생성하는 명령이 포함되어 있습니다. OpenSSL 명령을 복사하고 실행하기 전에 파일의 값을 바꾸어야 합니다.

7. **network-review** 프로젝트에서 **php-certs**라는 OpenShift TLS 시크릿을 만듭니다. 인증서에는 ~/D0280/labs/network-review/certs/training.crt 파일을, 키에는 ~/D0280/labs/network-review/certs/training.key 파일을 사용합니다.

8. ~/D0280/labs/network-review/php-https.yaml에 있는 리소스 파일을 사용하여 PHP 애플리케이션의 보안 버전을 배포합니다. **network-review** 프로젝트에 애플리케이션을 배포합니다.

애플리케이션을 배포하기 전에 리소스 파일에 필요한 변경 작업을 수행합니다. 구체적으로는 다음과 같습니다.

- 컨테이너의 위치.
- 애플리케이션에서 수신 대기하는 포트.
- 볼륨으로 마운트할 시크릿의 이름.

9. 호스트 이름이 **php-https.apps.ocp4.example.com**인 **php-https**라는 보안 패스스루 경로를 만들어 애플리케이션의 보안 버전에 액세스합니다.

**workstation** 시스템에서 Firefox를 사용하여 **https://php-https.apps.ocp4.example.com**에 액세스합니다. 서명된 인증서를 수락하고 애플리케이션 가용성을 확인합니다.

10. **선택적 단계:** **workstation** 시스템에서 **curl** 명령을 사용하여 애플리케이션의 HTTPS 버전에 액세스합니다.

CA 인증서를 **curl** 명령에 전달하여 보안 연결을 검증합니다.

11. **lab network-review finish** 명령을 사용하면 **network-review** 디렉터리가 삭제되므로 홈 디렉터리로 돌아갑니다.

```
[student@workstation network-review]$ cd
```

## 평가

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab network-review grade
```

## 완료

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab network-review finish
```

이로써 랩이 완료됩니다.

## ▶ 솔루션

# 애플리케이션을 위한 OpenShift 네트워킹 구성

이 랩에서는 애플리케이션을 위한 TLS 패스스루 경로를 구성합니다.

### 결과

다음을 수행할 수 있습니다.

- 애플리케이션을 배포하고 비보안 경로를 구성합니다.
- 애플리케이션으로 향하는 트래픽을 제한합니다.
- 애플리케이션용 TLS 인증서를 생성합니다.
- TLS 인증서를 사용하여 애플리케이션의 패스스루 경로를 구성합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있는지 확인하고 이 랩에서 사용하는 자체 서명 CA(인증 기관)를 만듭니다.

```
[student@workstation ~]$ lab network-review start
```

이 검토에서는 시스템에 대한 일부 정보를 출력하는 PHP 애플리케이션을 배포합니다. 이 애플리케이션은 두 가지 구성으로 사용할 수 있습니다. 하나는 포트 8080에서 수신 대기하는 암호화되지 않은 네트워크로 실행되고, 다른 하나는 TLS 인증서를 사용하여 포트 8443에서 수신 대기하는 네트워크 트래픽을 암호화합니다.

이 검토의 컨테이너 이미지는 `quay.io/redhattraining/php-ssl`에서 액세스할 수 있습니다. 여기에는 두 개의 태그(애플리케이션의 비보안 버전에 대한 `v1.0` 및 보안 버전의 `v1.1`)가 있습니다.

#### 1. OpenShift developer 사용자로, `network-review` 프로젝트를 만듭니다.

##### 1.1. 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

##### 1.2. `network-review` 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project network-review
Now using project "network-review" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

2. developer 사용자로 `network-review` 프로젝트에 비보안 버전의 PHP 애플리케이션을 배포합니다. `~/D0280/labs/network-review/php-http.yaml`에 제공된 리소스 파일을 사용합니다.

애플리케이션을 배포하기 전에 파일에 대한 필수 변경 사항, 구체적으로는 컨테이너 이미지의 위치 및 수신 대기하는 포트를 변경합니다.

애플리케이션을 만든 후 잠시 기다렸다가 하나의 포드가 실행되고 있는지 확인합니다.

- 2.1. `~/D0280/labs/network-review/` 디렉터리로 이동합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/network-review/
```

- 2.2. 다음과 같이 텍스트 편집기를 사용하여 `php-http.yaml` 파일을 업데이트합니다.

- 이미지 항목을 찾습니다. `quay.io/redhattraining/php-ssl:v1.0`에서 액세스할 수 있는 컨테이너 이미지를 사용하도록 설정합니다.

```
...output omitted...
 cpu: '0.5'
image: 'quay.io/redhattraining/php-ssl:v1.0'
name: php-http
...output omitted...
```

- `containerPort` 항목을 찾습니다. 비보안 엔드포인트에 해당하는 **8080**으로 설정합니다.

```
...output omitted...
ports:
- containerPort: 8080
 name: php-http
...output omitted...
```

변경 사항을 만든 후 파일을 저장하고 종료합니다.

- 2.3. `oc create` 명령을 사용하여 애플리케이션을 배포합니다. 그러면 배포 및 서비스가 생성됩니다.

```
[student@workstation network-review]$ oc create -f php-http.yaml
deployment.apps/php-http created
service/php-http created
```

- 2.4. 잠시 기다린 다음 `oc get pods` 명령을 실행하여 실행되고 있는 포드가 있는지 확인합니다.

```
[student@workstation network-review]$ oc get pods
NAME READY STATUS RESTARTS AGE
php-http-6cb58c847b-7qsbd 1/1 Running 0 8m11s
```

3. 호스트 이름이 `php-http.apps.ocp4.example.com`인 `php-http`라는 경로를 만들어 애플리케이션에 액세스합니다.

`workstation` 시스템에서 Firefox를 사용하여 `http://php-http.apps.ocp4.example.com`에 액세스합니다. 다음 단계를 진행하기 전에 애플리케이션 가용성을 확인합니다.

- 3.1. `oc expose` 명령을 실행하여 애플리케이션에 액세스할 수 있는 경로를 만듭니다. 경로에 호스트 이름 `php-http.apps.ocp4.example.com`을 지정합니다.

```
[student@workstation network-review]$ oc expose svc php-http \
> --hostname php-http.apps.ocp4.example.com
route.route.openshift.io/php-http exposed
```

- 3.2. 경로 이름을 검색하여 클립보드에 복사합니다.

```
[student@workstation network-review]$ oc get routes
NAME HOST/PORT PATH SERVICES PORT ...
php-http php-http.apps.ocp4.example.com php-http 8080 ...
```

- 3.3. `workstation` 시스템에서 Firefox를 열고 `http://php-http.apps.ocp4.example.com`에 액세스합니다.

애플리케이션이 표시되는지 확인합니다.

## About this application

### **A The application is currently served over HTTP**

- **Current system load:** 2.5
- **Number of connections:** 1
- **Memory usage:** 8 Mb

4. 기본적으로 모든 인그레스 트래픽을 거부하도록 `network-review` 네임스페이스에 네트워크 정책을 생성합니다. 네트워크 정책을 올바르게 구성하면 `network-review` 네임스페이스 내의 포드 간 통신도 금지할 수 있습니다.

`~/DO280/labs/network-review/deny-all.yaml`에 제공된 리소스 파일을 사용합니다. 네임스페이스의 모든 포드를 대상으로 하는 데 필요한 변경 작업을 수행합니다.

- 4.1. 네임스페이스의 모든 포드를 대상으로 하도록 텍스트 편집기에서 빈 포드를 사용하여 `deny-all.yaml` 파일을 업데이트합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: deny-all
spec:
 podSelector: {}
```

- 4.2. `oc create` 명령을 사용하여 네트워크 정책을 만듭니다.

```
[student@workstation network-review]$ oc create -f deny-all.yaml
networkpolicy.networking.k8s.io/deny-all created
```

- 4.3. `curl` 명령을 사용하여 경로에서 `php-http` 포드에 액세스할 수 없는지 확인합니다. 몇 초 동안 기다린 후 `Ctrl+C`를 눌러 `curl` 명령을 종료합니다.

```
[student@workstation network-review]$ curl http://php-http.apps.ocp4.example.com
^C
```



### 중요

`php-http` 포드가 `router-default` 포드와 동일한 노드에 있는 경우, 트래픽이 해당 라우터 포드를 통과할 때 `curl` 명령이 작동합니다. 이는 3노드 클러스터의 경우에만 해당합니다. 컨트롤 플레인이나 인프라 노드가 작업자 노드와 분리된 기존의 OpenShift 클러스터에서는 네트워크 정책이 클러스터의 모든 라우터 포드에 적용됩니다.

`curl` 명령이 성공하면 명령을 다시 실행하여 네트워크 정책이 예상대로 작동하는지 확인합니다. 이 두 번째 시도는 클러스터의 다른 라우터 포드를 거쳐야 합니다.

5. `network-review` 네임스페이스의 경로로 향하는 인그레스 트래픽을 허용하는 네트워크 정책을 생성합니다.

`~/D0280/labs/network-review/allow-from-openshift-ingress.yaml`에 제공된 리소스 파일을 사용합니다. 네임스페이스의 모든 포드를 대상으로 하고 기본 수신 컨트롤러의 트래픽을 허용하는 데 필요한 변경 작업을 수행합니다.

강의실 환경에서는 `HostNetwork` 엔드포인트 전략을 사용하므로 `default` 네임스페이스에 `network.openshift.io/policy-group=ingress` 레이블을 지정합니다. 이 작업은 `admin` 사용자로 수행해야 합니다.

- 5.1. 네임스페이스의 모든 포드를 대상으로 하도록 텍스트 편집기에서 빈 포드를 사용하여 `allow-from-openshift-ingress.yaml` 파일을 업데이트합니다. `network.openshift.io/policy-group=ingress` 레이블과 일치하는 네임스페이스 선택기를 포함합니다.

```
...output omitted...
spec:
 podSelector: {}
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 network.openshift.io/policy-group: ingress
```

- 5.2. `oc create` 명령을 사용하여 네트워크 정책을 만듭니다.

```
[student@workstation network-review]$ oc create -f \
> allow-from-openshift-ingress.yaml
networkpolicy.networking.k8s.io/allow-from-openshift-ingress created
```

- 5.3. `admin` 사용자로 `default` 네임스페이스 레이블을 `network.openshift.io/policy-group=ingress`로 지정합니다.

```
[student@workstation network-review]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation network-policy]$ oc label namespace default \
> network.openshift.io/policy-group=ingress
namespace/default labeled
```

- 5.4. `curl` 명령을 사용하여 경로에서 `php-http` 포드에 액세스할 수 있는지 확인합니다. 강의실에서는 3노드 클러스터를 실행하므로 `curl` 명령을 여러 번 실행하여 모든 라우터 포드의 액세스를 검증합니다.

```
[student@workstation network-review]$ for X in {1..4}
> do
> curl -s http://php-http.apps.ocp4.example.com | grep "PHP"
> done
<title>PHP Application</title>
<title>PHP Application</title>
<title>PHP Application</title>
<title>PHP Application</title>
```

6. `developer` 사용자로 암호화된 버전의 애플리케이션에 대한 TLS 인증서를 생성하고 서명합니다.

`php-https.apps.ocp4.example.com` 호스트 이름에 대한 CSR(인증서 서명 요청)을 생성합니다. CSR을 `~/D0280/labs/network-review/certs/training.csr`에 저장합니다.

CSR을 사용하여 인증서를 생성하고 `~/D0280/labs/network-review/certs/training.crt`에 저장합니다. 인증서를 생성하려면 `~/D0280/labs/network-review/certs/training-CA.pem` 및 CSR에서 액세스할 수 있는 CA 인증서를 인수로 전달합니다.

`~/D0280/labs/network-review/certs/openssl-commands.txt` 텍스트 파일을 사용하여 도움을 받을 수 있습니다. 이 파일에는 인증서 서명 요청 및 인증서를 생성하는 명령이 포함되어 있습니다. OpenSSL 명령을 복사하고 실행하기 전에 파일의 값을 바꾸어야 합니다.

- 6.1. 이 랩의 나머지 부분을 완료하려면 `developer` 사용자로 로그인합니다.

```
[student@workstation network-review]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 6.2. `~/D0280/labs/network-review/certs` 디렉터리로 이동합니다.

```
[student@workstation network-review]$ cd certs
```

- 6.3. `php-https.apps.ocp4.example.com`에 대한 CSR(인증서 서명 요청)을 생성합니다. 요청의 제목을 한 줄에 입력해야 합니다. 또는 `-subj` 옵션과 해당 컨텐츠를 제거합니다. 이 명령을 실행하면 값을 입력하라는 메시지가 표시됩니다. `php-https.apps.ocp4.example.com`의 `CN`(일반 이름)을 표시해야 합니다.



### 참고

조직(Red Hat)과 CN(일반 이름)의 후행 슬래시 뒤에 공백이 없는지 확인합니다.

```
[student@workstation certs]$ openssl req -new -key training.key \
> -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
> CN=php-https.apps.ocp4.example.com" \
> -out training.csr
```

또는 `openssl-commands` 텍스트 파일을 엽니다. 첫 번째 `openssl` 명령을 복사하여 터미널에 붙여 넣습니다. 와일드카드 도메인을 `apps.ocp4.example.com`으로 교체하고 출력 파일은 `training.csr`로 교체합니다.



### 참고

명령에서 출력을 생성하지 않습니다.

- 서명된 인증서를 생성합니다. CA의 인증서에 서명하기 위한 `-CA` 및 `-CAkey` 옵션을 사용하는 것을 확인합니다.

```
[student@workstation certs]$ openssl x509 -req -in training.csr \
> -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
> -passin file:passphrase.txt \
> -out training.crt -days 3650 -sha256 -extfile training.ext
Signature ok
subject=C = US, ST = North Carolina, L = Raleigh, O = Red Hat, CN = php-
https.apps.ocp4.example.com
Getting CA Private Key
```

또는 `openssl-commands.txt` 파일의 두 번째 `openssl` 명령을 복사하여 터미널에 붙여 넣습니다. CSR 파일을 `training.csr`로 바꾸고 CA를 `training-CA.pem`로 바꾸고 출력 인증서를 `training.crt`로 바꿉니다.

- 새로 생성된 인증서와 키가 현재 디렉터리에 있는지 확인합니다.

```
[student@workstation certs]$ ls -l
total 36
-rw-r--r-- 1 student student 564 Jul 31 09:35 openssl-commands.txt
-rw-r--r-- 1 student student 33 Aug 3 13:59 passphrase.txt
-rw----- 1 student student 1751 Aug 3 13:59 training-CA.key
-rw-r--r-- 1 student student 1334 Aug 3 13:59 training-CA.pem
-rw-rw-r-- 1 student student 41 Aug 3 14:20 training-CA.srl
-rw-rw-r-- 1 student student 1395 Aug 3 14:20 training.crt
-rw-rw-r-- 1 student student 1021 Aug 3 14:16 training.csr
-rw-r--r-- 1 student student 352 Aug 3 13:59 training.ext
-rw----- 1 student student 1675 Aug 3 13:59 training.key
```

- `network-review` 디렉터리로 돌아갑니다. 다음 단계에서는 서명된 인증서를 사용하여 경로를 생성하는 작업이 포함되어 있어 중요합니다.

```
[student@workstation certs]$ cd ~/D0280/labs/network-review
```

- `network-review` 프로젝트에서 `php-certs`라는 OpenShift TLS 시크릿을 만듭니다. 인증서에는 `~/D0280/labs/network-review/certs/training.crt` 파일을, 키에는 `~/D0280/labs/network-review/certs/training.key` 파일을 사용합니다.

## 12장 | OpenShift 네트워킹 구성 요소 구성

- 7.1. `oc create secret` 명령을 사용하여 `php-certs` TLS 시크릿을 만듭니다.  
`training.csr` 파일을 인증서로 전달하고 `training.key`를 키로 전달합니다.

```
[student@workstation network-review]$ oc create secret tls php-certs \
> --cert certs/training.crt \
> --key certs/training.key
secret/php-certs created
```

- 7.2. 시크릿 목록이 있는지 확인하기 위해 검색합니다.

```
[student@workstation network-review]$ oc get secrets
NAME TYPE DATA AGE
...output omitted...
php-certs kubernetes.io/tls 2 93s
```

8. ~/D0280/labs/network-review/php-https.yaml에 있는 리소스 파일을 사용하여 PHP 애플리케이션의 보안 버전을 배포합니다. `network-review` 프로젝트에 애플리케이션을 배포합니다.

애플리케이션을 배포하기 전에 리소스 파일에 필요한 변경 작업을 수행합니다. 구체적으로는 다음과 같습니다.

- 컨테이너의 위치.
- 애플리케이션에서 수신 대기하는 포트.
- 볼륨으로 마운트할 시크릿의 이름.

- 8.1. 다음과 같이 텍스트 편집기를 사용하여 `php-https.yaml` 파일을 업데이트합니다.

- 이미지 항목을 찾습니다. `quay.io/redhattraining/php-ssl:v1.1`에서 액세스할 수 있는 컨테이너 이미지를 사용하도록 설정합니다.

```
...output omitted...
cpu: '0.5'
image: 'quay.io/redhattraining/php-ssl:v1.1'
name: php-https
...output omitted...
```

- `containerPort` 항목을 찾습니다. 보안 엔드포인트에 해당하는 **8443**으로 설정합니다.

```
...output omitted...
name: php-https
ports:
- containerPort: 8443
 name: php-https
...output omitted...
```

- `secretName` 항목을 찾습니다. 이전 단계에서 만든 시크릿 이름에 해당하는 `php-certs`로 설정합니다.

```
...output omitted...
volumes:
- name: tls-certs
 secret:
 secretName: php-certs
...output omitted...
```

변경 사항을 만든 후 파일을 저장하고 종료합니다.

- 8.2. **oc create** 명령을 사용하여 보안 애플리케이션을 배포합니다. 그러면 배포 및 서비스가 생성됩니다.

```
[student@workstation network-review]$ oc create -f php-https.yaml
deployment.apps/php-https created
service/php-https created
```

- 8.3. 잠시 기다린 다음 **oc get pods** 명령을 실행하여 **php-https** 포드가 실행되고 있는지 확인합니다.

```
[student@workstation network-review]$ oc get pods
NAME READY STATUS RESTARTS AGE
php-http-6cb58c847b-7qsb7d 1/1 Running 0 8m11s
php-https-84498cd794-hvf7w 1/1 Running 0 26s
```

9. 호스트 이름이 **php-https.apps.ocp4.example.com**인 **php-https**라는 보안 패스스루 경로를 만들어 애플리케이션의 보안 버전에 액세스합니다.

**workstation** 시스템에서 Firefox를 사용하여 **https://php-https.apps.ocp4.example.com**에 액세스합니다. 서명된 인증서를 수락하고 애플리케이션 가용성을 확인합니다.

- 9.1. **oc create route** 명령을 실행하여 애플리케이션 액세스를 위한 패스스루 경로를 만듭니다. 경로에 **php-https.apps.ocp4.example.com**이라는 호스트 이름을 지정합니다. **port** 옵션을 사용하여 보안 포트 8443을 나타냅니다.

```
[student@workstation network-review]$ oc create route passthrough php-https \
> --service php-https --port 8443 --hostname php-https.apps.ocp4.example.com
route.route.openshift.io/php-https created
```

- 9.2. 경로 이름을 검색하여 클립보드에 복사합니다.

```
[student@workstation network-review]$ oc get routes
NAME HOST/PORT ... SERVICES PORT TERMINATION
php-http php-http.apps.ocp4.example.com ... php-http 8080
php-https php-https.apps.ocp4.example.com ... php-https 8443 passthrough
```

- 9.3. **workstation** 시스템에서 Firefox를 열고 **https://php-https.apps.ocp4.example.com**에 액세스합니다.

서명된 인증서를 수락하고 애플리케이션의 보안 버전이 표시되는지 확인합니다.

## About this application

### The application is currently served over TLS

- Current system load: 1
- Number of connections: 0
- Memory usage: 8 Mb

10. 선택적 단계: **workstation** 시스템에서 **curl** 명령을 사용하여 애플리케이션의 HTTPS 버전에 액세스합니다.

CA 인증서를 **curl** 명령에 전달하여 보안 연결을 검증합니다.

**--cacert** 옵션을 사용하여 CA 인증서를 **curl** 명령에 전달합니다.

```
[student@workstation network-review]$ curl -v --cacert certs/training-CA.pem \
> https://php-https.apps.ocp4.example.com
...output omitted...
* TLSv1.3 (OUT), TLS handshake, Finished (20):
...output omitted...
* Server certificate:
* subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; \
CN=php-https.apps.ocp4.example.com
...output omitted...
 The application is currently served over TLS
...output omitted...
```

11. **lab network-review finish** 명령을 사용하면 **network-review** 디렉터리가 삭제되므로 홈 디렉터리로 돌아갑니다.

```
[student@workstation network-review]$ cd
```

## 평가

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab network-review grade
```

## 완료

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab network-review finish
```

이로써 랩이 완료됩니다.

## 요약

이 장에서 학습한 내용:

- OpenShift는 SDN(소프트웨어 정의 네트워킹)을 구현하여 클러스터의 네트워크 인프라를 관리합니다. SDN은 트래픽을 라우팅하는 기본 메커니즘에서 트래픽을 처리하는 소프트웨어를 분리합니다.
- Kubernetes는 공통 액세스 경로에서 포드의 논리적 그룹을 사용할 수 있는 서비스를 제공합니다. 서비스는 하나 이상의 포드 앞에서 부하 분산 장치 기능을 합니다.
- 서비스에서는 선택기(레이블)를 사용하여 서비스에 사용 가능한 포드를 나타냅니다.
- 두 가지 종류의 경로(보안 및 비보안)가 있습니다. 보안 경로는 TLS 인증서를 사용하여 트래픽을 암호화하며 비보안 경로는 암호화되지 않은 연결을 통해 트래픽을 전달합니다.  
보안 경로에서는 에지, 패스스루, 재암호화의 세 가지 모드를 지원합니다.
- 네트워크 정책은 포드로 향하는 네트워크 트래픽을 제어합니다. SDN에서 논리 영역을 생성하면 모든 네임 스페이스의 포드 간 트래픽을 분리할 수 있습니다.



## 13장

# 포드 스케줄링 제어

### 목적

포드에서 실행하는 노드를 제어합니다.

### 목표

- 포드 스케줄링 알고리즘, 스케줄링에 영향을 주는 메서드를 설명하고 이러한 메서드를 적용하는 방법에 대해 설명합니다.
- 컨테이너, 포드, 프로젝트에서 사용하는 리소스를 제한합니다.
- 포드의 복제 수를 제어합니다.

### 섹션

- 포드 스케줄링 동작 제어(안내에 따른 연습)
- 애플리케이션의 리소스 사용 제한(안내에 따른 연습)
- 애플리케이션 확장(안내에 따른 연습)

### 랩

포드 스케줄링 제어

# 포드 스케줄링 동작 제어

## 목표

이 섹션을 마치면 포드 스케줄링 알고리즘과 스케줄링에 영향을 주는 메서드를 설명하고 이러한 메서드를 적용할 수 있습니다.

## OpenShift 스케줄러 알고리즘 소개

포드 스케줄러는 OpenShift 클러스터의 노드에서 새 포드의 배치를 결정합니다. 다른 클러스터에 구성 가능하고 적용 가능하도록 고안되었습니다. Red Hat OpenShift Container Platform에 제공되는 기본 구성은 노드 레이블, 선호도 규칙 및 선호도 방지 규칙을 사용하여 영역 및 지역의 일반 데이터 센터 개념을 지원합니다.

OpenShift 포드 스케줄러 알고리즘은 3단계 프로세스를 따릅니다.

1. 노드 필터링.

스케줄러는 호스트 포트의 가용성과 같은 서술자 집합 또는 디스크 또는 메모리 압력을 발생하는 노드에 대해 포드를 예약할 수 있는지 여부에 대해 각 노드를 평가하여 실행 중인 노드의 목록을 필터링합니다.

또는 포드는 클러스터 노드에서 레이블과 일치하는 노드 선택기를 정의할 수 있습니다. 레이블이 일치하지 않는 노드는 자격이 되지 않습니다.

또한 포드는 CPU, 메모리 및 스토리지 등의 컴퓨팅 리소스에 대한 리소스 요청을 정의할 수 있습니다. 여유 컴퓨터 리소스가 충분하지 않은 노드는 자격이 되지 않습니다.

또 다른 필터링 검사는 노드 목록에 taints가 있는지 평가하고 해당 포드에 taint를 수락할 수 있는 관련 toleration이 있는지 여부를 평가합니다. 포드에서 노드의 taint를 수락할 수 없는 경우 해당 노드는 적합하지 않습니다. 기본적으로 컨트롤 플레인 노드에는 taint **node-role.kubernetes.io/master:NoSchedule**이 포함되어 있습니다. 이 taint에 맞는 toleration이 없는 포드는 컨트롤 플레인 노드에 예약되지 않습니다.



### 참고

강의실 환경에서는 추가 계산 노드가 포함되지 않은 3노드 클러스터를 사용합니다. 3노드 클러스터는 OpenShift Container Platform 4.5의 베어 메탈 설치에 사용할 수 있습니다. 이 유형의 클러스터는 원거리 에지 배포와 같은 리소스 제한 환경에 적용할 수 있습니다.

3노드 클러스터의 컨트롤 플레인 노드에는 **node-role.kubernetes.io/master:NoSchedule** taint가 없습니다. 일반 애플리케이션 포드를 컨트롤 플레인 노드에 예약할 수 있습니다.

필터링 단계의 최종 결과는 일반적으로 포드를 실행하는 데 적합한 노드 후보의 짧은 목록입니다. 어떤 경우에는 필터링되는 노드가 아무 것도 없으므로 모든 노드에서 포드를 실행할 수 있습니다. 그 밖의 경우 모든 노드가 필터링되며 이는 원하는 필수 구성 요소를 가진 노드를 사용할 수 있게 될 때까지 포드를 예약할 수 없음을 의미합니다.

전체 서술자 목록과 해당 설명은 참조 섹션에서 찾을 수 있습니다.

2. 필터링된 노드 목록의 우선 순위 지정.

후보 노드 목록은 가중치 점수에 추가하는 여러 우선 순위 기준을 사용하여 평가됩니다. 높은 값이 있는 노드는 포드를 실행하기에 더 좋은 후보입니다.

기준에는 **선호도** 및 **선호 방지** 규칙이 있습니다. 포드에 더 높은 선호도가 있는 노드에는 더 높은 점수가 있으며 더 높은 선호 방지가 있는 노드에는 더 낮은 점수가 있습니다.

**선호도** 규칙의 일반적인 사용은 성능의 이유로 관련된 포드를 서로 가깝게 스케줄링하는 것입니다. 예시는 서로 동기화하는 포드의 동일한 네트워크 백본을 사용하는 것입니다.

**선호 방지** 규칙의 일반적인 사용은 고가용성을 위해 관련된 포드를 서로 너무 가깝지 않게 스케줄링하는 것입니다. 한 가지 예는 동일한 애플리케이션에서 동일한 노드로 모든 포드 스케줄링을 피하는 것입니다.

### 3. 최적의 노드 선택.

이러한 점수를 기반으로 후보 목록을 정렬하며 포드를 호스트하도록 점수가 가장 높은 노드를 선택합니다. 여러 노드에 똑같이 높은 점수가 있으면 하나의 노드가 라운드 로빈 형식으로 선택됩니다.

스케줄러는 유연성이 있으며 고급 스케줄링 환경에 맞게 사용자 지정할 수 있습니다. 또한 이 과정은 노드 레이블 및 노드 선택기를 사용하여 포드 배치에 집중하지만 포드는 포드 선호도 및 선호도 방지 규칙과 노드 선호도 및 선호도 방지 규칙을 사용하여 배치할 수도 있습니다. 스케줄러를 사용자 지정하고 이러한 대체 포드 배치 시나리오를 처리하는 것은 이 과정의 범위를 벗어납니다.

## 스케줄링 및 토플로지

클라우드 프로바이더 등 대형 데이터 센터의 일반 토플로지는 **지역** 및 **영역**으로 호스트를 구성하는 것입니다.

- **지역**은 지리적으로 가까운 영역에 있는 호스트 집합으로, 이러한 영역 간 고속 연결을 보장합니다.
- **영역(가용성 영역이라고도 함)**은 네트워크 스위치, 스토리지 어레이 또는 인터럽트 없는 전력 등의 일반적인 중요한 인프라 구성 요소를 공유하므로 함께 오류가 발생할 수 있는 호스트 집합입니다.

지역 및 영역을 예로 들어 AWS(Amazon Web Services)는 북쪽 버지니아(**us-east-1**)에 6개의 가용성 영역을 포함한 지역이 있으며 오하이오(**us-east-2**)에 3개의 가용성 영역이 포함된 또 다른 지역이 있습니다. 각 AWS 가용성 영역에는 수십만 개의 서버로 구성된 여러 데이터 센터가 포함될 수 있습니다.

OpenShift 포드 스케줄러의 표준 구성은 **지역** 및 **영역** 레이블을 기반으로 서술자를 정의하여 이 종류의 클러스터 토플로지를 지원합니다. 서술자는 이러한 방법으로 정의됩니다.

- 동일한 배포(또는 배포 구성)에서 생성된 복제 포드는 **지역** 레이블에 대해 동일한 값을 보유하는 노드에서 실행하도록 예약됩니다.
- 복제 포드는 **영역** 레이블에 대해 다른 값을 보유하는 노드에서 실행되도록 예약됩니다.

아래의 그림은 각각 여러 영역으로 된 여러 지역과 여러 노드로 된 각 영역을 구성하는 샘플 토플로지를 보여줍니다.

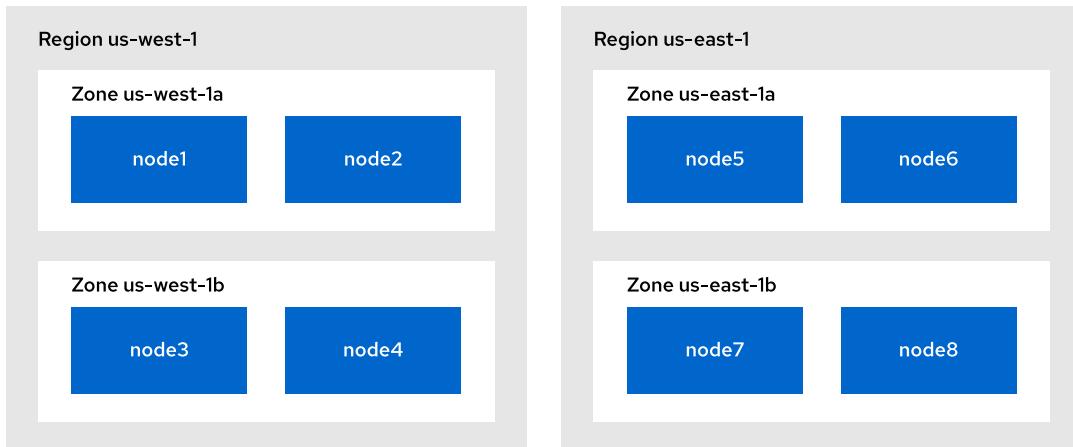


그림 13.1: 지역 및 영역을 사용하는 샘플 클러스터 토플로지

## 노드 레이블 지정

OpenShift 클러스터 관리자는 노드에 다른 레이블을 추가할 수 있습니다. 예를 들어 **dev**, **qa** 또는 **prod**의 값을 사용하여 개발, 품질 보장 및 생산 워크로드를 노드의 특정 하위 집합에 배포할 수 있도록 **env** 레이블에 노드 레이블을 지정할 수 있습니다. 선택하는 레이블은 임의이지만 적절하게 애플리케이션을 구성할 수 있도록 레이블 및 관련 값을 개발자에게 게시해야 합니다.

**oc label** 명령을 클러스터 관리자로 사용하여 노드 레이블을 즉시 추가, 업데이트 또는 제거합니다. 예를 들어 다음 명령을 사용하여 **env=dev**를 사용하여 노드에 레이블을 지정합니다.

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env=dev
```

기존 레이블을 변경하려면 **--overwrite** 옵션을 사용합니다.

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env=prod --overwrite
```

**env-** 등 레이블 이름 뒤에 하이픈을 지정하여 레이블을 제거합니다.

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env-
```



### 중요

레이블과 해당 값은 대소문자를 구분합니다. 애플리케이션 노드 선택기는 실제 레이블의 대소문자와 일치해야 하며 노드에 적용된 값과 일치해야 합니다.

**oc get nodes** 명령과 함께 **--show-labels** 옵션을 사용하여 노드에 할당된 대소문자를 구분하는 레이블을 확인합니다.

```
[user@demo ~]$ oc get node node2.us-west-1.compute.internal --show-labels
NAME ... ROLES ... LABELS
node2.us-west-1.compute.internal ... worker ... beta.kubernetes.io/
arch=amd64,beta.kubernetes.io/instance-type=m4.xlarge,beta.kubernetes.io/
os=linux,tier=gold,failure-domain.beta.kubernetes.io/region=us-
west-1,failure-domain.beta.kubernetes.io/zone=us-west-1c,kubernetes.io/
arch=amd64,kubernetes.io/hostname=node2,kubernetes.io/os=linux,node-
role.kubernetes.io/worker=,node.openshift.io/os_id=rhcos
```

노드에는 OpenShift에서 할당한 몇 가지 기본 레이블이 있을 수 있습니다. 키에 `kubernetes.io` 접미사가 포함된 레이블은 스케줄러에서 내부적으로 사용하므로 클러스터 관리자가 변경할 수 없습니다. 다음의 샘플 명령에 표시된 노드에서는 AWS 풀스택 자동화 설정을 사용합니다.

또한 클러스터 관리자는 `-L` 옵션을 사용하여 단일 레이블의 값을 결정할 수 있습니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc get node -L failure-domain.beta.kubernetes.io/region
NAME ... ROLES ... REGION
ip-10-0-131-214.us-west-1.compute.internal ... master ... us-west-1
ip-10-0-139-250.us-west-1.compute.internal ... worker ... us-west-1
ip-10-0-141-144.us-west-1.compute.internal ... master ... us-west-1
ip-10-0-152-57.us-west-1.compute.internal ... master ... us-west-1
ip-10-0-154-226.us-west-1.compute.internal ... worker ... us-west-1
```

동일한 `oc get` 명령의 여러 `-L` 옵션이 지원됩니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc get node -L failure-domain.beta.kubernetes.io/region \
> -L failure-domain.beta.kubernetes.io/zone -L env
NAME ... REGION ZONE ENV
ip-10-0-131-214.us-west-1.compute.internal ... us-west-1 us-west-1b
ip-10-0-139-250.us-west-1.compute.internal ... us-west-1 us-west-1b dev
ip-10-0-141-144.us-west-1.compute.internal ... us-west-1 us-west-1b
ip-10-0-152-57.us-west-1.compute.internal ... us-west-1 us-west-1c
ip-10-0-154-226.us-west-1.compute.internal ... us-west-1 us-west-1c
```

## 시스템 세트에 레이블 지정

노드 레이블은 영구적이지만 OpenShift 클러스터에 시스템 집합이 포함된 경우 시스템 집합 구성에도 레이블을 추가해야 합니다. 이렇게 하면 새 시스템(및 해당 시스템에서 생성된 노드)에 원하는 레이블도 포함됩니다. 시스템 집합은 풀스택 자동화로 설치된 클러스터 및 기존 인프라 방법으로 설치되어 클라우드 프로바이더 통합을 지원하는 일부 클러스터에서 발견됩니다. 베어 메탈 클러스터에서는 시스템 집합을 사용하지 않습니다.

`openshift-machine-api` 네임스페이스에 시스템을 나열하고 `-o wide` 옵션을 포함하여 시스템과 노드 간의 관계를 식별할 수 있습니다.

```
[user@demo ~]$ oc get machines -n openshift-machine-api -o wide
NAME ... NODE
...output omitted...
ocp-qz7hf-worker-us-west-1b-rvx6w ... ip-10-0-139-250.us-west-1.compute.internal
ocp-qz7hf-worker-us-west-1c-v4n4n ... ip-10-0-154-226.us-west-1.compute.internal
```

**작업자** 노드에 사용되는 시스템은 시스템 세트에서 가져와야 합니다. 시스템의 이름에는 생성된 시스템 집합의 이름이 포함되어 있습니다.

```
[user@demo ~]$ oc get machineset -n openshift-machine-api
NAME DESIRED CURRENT READY AVAILABLE ...
ocp-qz7hf-worker-us-west-1b 1 1 1 1 ...
ocp-qz7hf-worker-us-west-1c 1 1 1 1 ...
```

시스템 집합에서 생성된 새 시스템에 원하는 레이블이 있는 시스템 집합을 편집합니다. 시스템 집합을 수정해도 기존 시스템 또는 노드에 변경 사항이 적용되지 않습니다.

```
[user@demo ~]$ oc edit machineset ocp-qz7hf-worker-us-west-1b \
> -n openshift-machine-api
```

아래에 강조 표시된 행은 시스템 집합 내에서 레이블을 추가할 위치를 표시합니다.

```
...output omitted...
spec:
 metadata:
 creationTimestamp: null
 labels:
 env: dev
 providerSpec:
...output omitted...
```

## 포드 배치 제어

OpenShift 클러스터에 있는 대부분의 인프라 관련 포드는 컨트롤 플레인 노드에서 실행되도록 구성되어 있습니다. 예시에는 DNS 운영자, OAuth 운영자, OpenShift API 서버에 대한 포드가 포함되어 있습니다. 일부 경우에는 데몬 집합 또는 복제본 집합의 구성에서 노드 선택기 `node-role.kubernetes.io/master: ''`을 사용하여 이 작업을 수행합니다.

이와 유사하게 일부 사용자 애플리케이션에는 특정 노드 집합에서의 실행이 필요할 수 있습니다. 예를 들어 특정 노드가 워크로드의 특정 유형에 대해 하드웨어 가속화를 제공하거나 클러스터 관리자가 개발 애플리케이션과 프로덕션 애플리케이션 혼합을 원하지 않습니다. 노드 레이블 및 노드 선택기를 사용하여 이러한 종류의 시나리오를 구현합니다.

노드 선택기는 개별 포드 정의의 일부입니다. 배포 또는 배포 구성 리소스에 노드 선택기를 정의하여 해당 리소스에서 생성된 모든 새 포드에 원하는 노드 선택기를 지정합니다. 배포 또는 배포 구성 리소스가 버전 제어에 있는 경우 리소스 파일을 수정하고 `oc apply -f` 명령을 사용하여 변경 사항을 적용합니다.

또는 `oc edit` 명령 또는 `oc patch` 명령을 사용하여 노드 선택기를 추가하거나 수정할 수 있습니다. 예를 들어 포드가 `env=qa` 레이블이 있는 노드에서만 실행되도록 `myapp` 배포를 구성하려면 `oc edit` 명령을 사용합니다.

```
[user@demo ~]$ oc edit deployment/myapp
```

```
...output omitted...
spec:
 ...output omitted...
 template:
 metadata:
```

```

annotations:
 openshift.io/generated-by: OpenShiftNewApp
creationTimestamp: null
labels:
 deployment: myapp
spec:
 nodeSelector:
 env: dev
 containers:
 - image: quay.io/redhattraining/scaling:v1.0
...output omitted...

```

다음 `oc patch` 명령은 동일한 작업을 수행합니다.

```
[user@demo ~]$ oc patch deployment/myapp --patch \
> '{"spec":{"template":{"spec":{"nodeSelector":{"env":"dev"}}}}}'
```

`oc edit` 명령을 사용하는 `oc patch` 명령을 사용하는 이 변경은 새 배포를 트리거하며 새 포드는 새 노드 선택기에 따라 예약됩니다.

## 프로젝트에 대한 노드 선택기 구성

클러스터 관리자가 포드에 대해 개발자가 노드 선택기를 제어할 수 없도록 하려는 경우 기본 노드 선택기는 프로젝트 리소스에서 구성되어야 합니다. 클러스터 관리자는 프로젝트가 생성될 때 노드 선택기를 정의하거나 프로젝트가 생성된 후 노드 선택기를 추가하거나 업데이트할 수 있습니다. `oc adm new-project` 명령을 사용하여 프로젝트 생성 시 노드 선택기를 추가합니다. 예를 들어 다음 명령은 `demo`라는 새 프로젝트를 생성하며 모든 포드는 `tier=1`의 레이블이 있는 노드에 배포됩니다.

```
[user@demo ~]$ oc adm new-project demo --node-selector "tier=1"
```

기존 프로젝트의 기본 노드 선택기를 구성하려면 주석을 `openshift.io/node-selector` 키가 있는 네임스페이스 리소스에 추가합니다. `oc annotate` 명령은 노드 선택기 주석을 추가, 수정 또는 제거할 수 있습니다.

```
[user@demo ~]$ oc annotate namespace demo \
> openshift.io/node-selector="tier=2" --overwrite
```

## 포드 복제본 수 확장

대부분의 배포 및 배포 구성 리소스는 단일 포드를 생성하는 것으로 시작하지만 포드의 복제본(또는 복사본) 수는 빈번하게 증가합니다. 배포 또는 배포 구성을 확장하여 이 작업을 수행할 수 있습니다. 확장에 대한 여러 가지 메서드는 나중에 다를 것이지만 한 가지 메서드는 `oc scale` 명령을 사용합니다. 예를 들어, 다음 명령을 사용하여 `myapp` 배포의 포드 수를 3개로 확장할 수 있습니다.

```
[user@demo ~]$ oc scale --replicas 3 deployment/myapp
```



### 참조

자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서의 노드에서 포드 배치 제어(스케줄링) 장을 참조하십시오.

### Amazon Web Services 지역 및 가용성 영역

[https://aws.amazon.com/about-aws/global-infrastructure/regions\\_az/](https://aws.amazon.com/about-aws/global-infrastructure/regions_az/)

## ▶ 연습 가이드

# 포드 스케줄링 동작 제어

이 연습에서는 클러스터 작업자 노드의 하위 집합에서 실행하도록 애플리케이션을 구성합니다.

### 결과

OpenShift 명령줄 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- 노드에 새 레이블을 추가합니다.
- 지정된 레이블과 일치하는 노드에 포드를 배포합니다.
- 노드에서 레이블을 제거합니다.
- 포드가 노드에 배포하지 못한 경우 문제를 해결합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

명령을 사용하면 클러스터 API에 연결할 수 있으며 활동에서 사용할 프로젝트가 생성됩니다.

```
[student@workstation ~]$ lab schedule-pods start
```

#### ▶ 1. developer 사용자로 schedule-pods라는 새 프로젝트를 만듭니다.

- 1.1. OpenShift 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **schedule-pods**라는 새 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project schedule-pods
Now using project "schedule-pods" on server "https://api.ocp4.example.com".
...output omitted...
```

#### ▶ 2. 테스트 애플리케이션을 배포하고 확장합니다.

- 2.1. [quay.io/redhattraining/hello-world-nginx:v1.0](https://quay.io/redhattraining/hello-world-nginx:v1.0)에 있는 컨테이너를 사용하여 **hello**라는 새 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc new-app --name hello \
> --docker-image quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
```

```
--> Creating resources ...
imagestream.image.openshift.io "hello" created
deployment.apps "hello" created
service "hello" created
--> Success
...output omitted...
```

2.2. 해당 애플리케이션의 경로를 만듭니다.

```
[student@workstation ~]$ oc expose svc/hello
route.route.openshift.io/hello exposed
```

2.3. 4개의 포드가 실행되고 있는 경우 수동으로 애플리케이션 크기를 조정합니다.

```
[student@workstation ~]$ oc scale --replicas 4 deployment/hello
deployment.apps/hello scaled
```

2.4. 실행 중인 4개의 포드가 해당 노드 간에 배포되는지 확인합니다.

```
[student@workstation ~]$ oc get pods -o wide
NAME READY STATUS ... IP NODE ...
hello-6c4984d949-78qsp 1/1 Running ... 10.9.0.30 master02 ...
hello-6c4984d949-cf6tb 1/1 Running ... 10.10.0.20 master01 ...
hello-6c4984d949-kwgbg 1/1 Running ... 10.8.0.38 master03 ...
hello-6c4984d949-mb8z7 1/1 Running ... 10.10.0.19 master01 ...
```



### 참고

각 노드의 기존 부하에 따라 출력이 다를 수 있습니다. 스케줄러가 포드 배포를 시도하지만 이 경우에도 배포가 가능하지 않을 수 있습니다.

- ▶ 3. env 레이블을 할당하여 애플리케이션 워크로드를 개발 또는 프로덕션 노드에 배포할 수 있도록 노드를 준비합니다. env=dev 레이블을 master01 노드에 할당하고, env=prod 레이블을 master02 노드에 할당합니다.

3.1. OpenShift 클러스터에 admin 사용자로 로그인합니다. 일반 사용자는 노드에 대한 정보를 볼 수 없으며 노드 레이블을 지정할 수 없습니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

3.2. env 레이블을 사용하는 노드가 없는지 확인합니다.

```
[student@workstation ~]$ oc get nodes -L env
NAME STATUS ROLES AGE VERSION ENV
master01 Ready master,worker 5d18h v1.18.3+012b3ec
master02 Ready master,worker 5d18h v1.18.3+012b3ec
master03 Ready master,worker 5d18h v1.18.3+012b3ec
```

3.3. env=dev 레이블을 master01 노드에 추가하여 해당 노드가 개발 노드임을 나타냅니다.

```
[student@workstation ~]$ oc label node master01 env=dev
node/master01 labeled
```

- 3.4. `env=prod` 레이블을 `master02` 노드에 추가하여 해당 노드가 프로덕션 노드임을 나타냅니다.

```
[student@workstation ~]$ oc label node master02 env=prod
node/master02 labeled
```

- 3.5. 노드에 올바른 `env` 레이블이 설정되어 있는지 확인합니다. 나중에 애플리케이션 포드가 해당 노드에 배포되었는지 확인하는 것과 같이 `env=dev` 레이블이 있는 노드를 기록해 둡니다.

```
[student@workstation ~]$ oc get nodes -L env
NAME STATUS ROLES AGE VERSION ENV
master01 Ready master,worker 5d18h v1.18.3+012b3ec dev
master02 Ready master,worker 5d18h v1.18.3+012b3ec prod
master03 Ready master,worker 5d18h v1.18.3+012b3ec
```

- ▶ 4. `developer` 사용자로 다시 전환하여 개발 노드에 배포하도록 `hello` 애플리케이션을 수정합니다. 이 변경 사항을 확인한 후 `schedule-pods` 프로젝트를 삭제합니다.

- 4.1. OpenShift 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
Using project "schedule-pods".
```

- 4.2. `hello` 애플리케이션의 배포 리소스를 수정하여 개발 노드를 선택합니다. `template(템플릿)` 섹션에서 노드 선택기에 `spec` 그룹을 추가해야 합니다.

```
[student@workstation ~]$ oc edit deployment/hello
```

아래에 강조 표시된 행을 배포 리소스에 추가하고 표시된 대로 들여쓰기합니다.

```
...output omitted...
 terminationMessagePath: /dev/termination-log
 terminationMessagePolicy: File
 dnsPolicy: ClusterFirst
 nodeSelector:
 env: dev
 restartPolicy: Always
...output omitted...
```

`oc edit`의 다음 출력은 변경 사항을 저장한 후 표시됩니다.

```
deployment.apps/hello edited
```

- 4.3. 애플리케이션 포드가 `env=dev` 레이블을 사용하여 노드에 배포되었는지 확인합니다. 다시 배포하는 데는 많은 시간이 소요되지만 `master01` 노드에 애플리케이션 포드를 배포해야 합니다.

```
[student@workstation ~]$ oc get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE
hello-b556ccf8b-8scxd 1/1 Running 0 80s 10.10.0.14 master01
hello-b556ccf8b-hb24w 1/1 Running 0 77s 10.10.0.16 master01
hello-b556ccf8b-qxlj8 1/1 Running 0 80s 10.10.0.15 master01
hello-b556ccf8b-sdwpj 1/1 Running 0 76s 10.10.0.17 master01
```

- 4.4. `schedule-pods` 프로젝트를 제거합니다.

```
[student@workstation ~]$ oc delete project schedule-pods
project.project.openshift.io "schedule-pods" deleted
```

#### ▶ 5. 모든 노드에서 `env` 레이블을 제거하여 이 연습 부분의 정리를 완료합니다.

- 5.1. OpenShift 클러스터에 `admin` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 5.2. `env` 레이블이 있는 모든 노드에서 이 레이블을 제거합니다.

```
[student@workstation ~]$ oc label node -l env env-
node/master01 labeled
node/master02 labeled
```

#### ▶ 6. `schedule-pods-ts` 프로젝트에는 `client=ACME`로 표시된 노드에서만 실행되는 애플리케이션을 포함합니다. 다음 예제에서는 애플리케이션 포드가 보류 중이며 다음 단계를 사용하여 문제를 진단해야 합니다.

- 6.1. OpenShift 클러스터에 `developer` 사용자로 로그인하고 `schedule-pods-ts` 프로젝트를 사용하고 있는지 확인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
Using project "schedule-pods-ts".
```

위의 출력이 `schedule-pods-ts` 프로젝트를 사용하고 있는 것으로 표시되지 않는 경우 전환합니다.

```
[student@workstation ~]$ oc project schedule-pods-ts
Now using project "schedule-pods-ts" on server
"https://api.ocp4.example.com:6443".
```

- 6.2. 애플리케이션 포드에 `Pending(보류 중)` 상태가 있는지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-ts-5dbff9f44-w6csj 0/1 Pending 0 6m19s
```

- 6.3. 상태가 보류 중인 포드에는 로그가 없으므로 포드 설명이 유용한 정보를 제공하는지 확인하도록 `oc describe pod` 명령을 사용하여 포드의 세부 정보를 확인합니다.

```
[student@workstation ~]$ oc describe pod hello-ts-5dbff9f44-8h7c7
...output omitted...
QoS Class: BestEffort
Node-Selectors: client=acme
Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s
 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
 Type Reason ... Message
 ---- ----- ... -----
 Warning FailedScheduling ... 0/3 nodes are available: 3 node(s) didn't match
node selector.
```

이 정보를 기반으로 포드는 레이블 `client=acme`를 사용하여 노드에 예약해야 하지만 세 노드 중에는 이 레이블이 있는 노드가 없습니다. 제공된 정보는 하나 이상의 작업자 노드가 레이블 `client=ACME`가 있음을 나타냅니다. 문제를 발견했습니다. 올바른 노드 선택기를 사용하도록 애플리케이션을 수정해야 합니다.

- 6.4. 올바른 노드 선택기를 사용하도록 애플리케이션의 배포 리소스를 편집합니다.

```
[student@workstation ~]$ oc edit deployment/hello-ts
```

아래에 표시된 대로 `acme`를 `ACME`로 변경합니다.

```
...output omitted...
 dnsPolicy: ClusterFirst
 nodeSelector:
 client: ACME
 restartPolicy: Always
...output omitted...
```

`oc edit`의 다음 출력은 변경 사항을 저장한 후 표시됩니다.

```
deployment.apps/hello-ts edited
```

- 6.5. 새 애플리케이션 포드가 배포되고 `Running`(실행 중) 상태가 있는지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-ts-69769f64b4-wwhpc 1/1 Running 0 11s
```

## 완료

`workstation` 시스템에서 `lab` 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab schedule-pods finish
```

이로써 랩이 완료됩니다.

# 애플리케이션의 리소스 사용 제한

## 목표

이 섹션을 마치면 컨테이너, 포드, 프로젝트에서 사용하는 리소스를 제한할 수 있습니다.

## 포드의 리소스 요청 및 제한 정의

포드 정의에는 리소스 요청 및 리소스 제한이 모두 포함될 수 있습니다.

### 리소스 요청

스케줄링에 사용되며 포드가 컴퓨팅 리소스의 지정된 양보다 적은 양으로 실행할 수 없음을 나타냅니다.

스케줄러는 포드 요청을 충족하기 위해 충분한 컴퓨팅 리소스로 노드 찾기를 시도합니다.

### 리소스 제한

포드가 노드에서 모든 컴퓨팅 리소스를 사용하지 않도록 방지하는 데 사용됩니다. 포드를 실행하는 노드는 포드의 리소스 제한을 강제 적용하도록 Linux 커널 **cgroups** 기능을 구성합니다.

리소스 요청 및 리소스 제한은 배포 또는 배포 구성 리소스의 각 컨테이너에 대해 정의되어야 합니다. 요청 및 제한이 정의되지 않은 경우 각 컨테이너에 대한 **resources: {}** 행을 찾을 수 있습니다.

**resources: {}** 행을 수정하여 원하는 요청 및/또는 제한을 지정합니다. 예를 들면 다음과 같습니다.

```
...output omitted...
spec:
 containers:
 - image: quay.io/redhattraining/hello-world-nginx:v1.0
 name: hello-world-nginx
 resources:
 requests:
 cpu: "10m"
 memory: 20Mi
 limits:
 cpu: "80m"
 memory: 100Mi
 status: {}
```

**oc edit** 명령을 사용하여 배포 또는 배포 구성을 수정하는 경우 들여쓰기를 올바르게 사용해야 합니다. 들여쓰기 오류로 인해 편집기에서 변경 내용을 저장하는 것을 거부할 수 있습니다. 들여쓰기 문제를 피하려면 **oc set resources** 명령을 사용하여 리소스 요청 및 제한을 지정할 수 있습니다. 다음 명령은 앞의 예제와 동일한 요청 및 제한 사항을 설정합니다.

```
[user@demo ~]$ oc set resources deployment hello-world-nginx \
> --requests cpu=10m, memory=20Mi --limits cpu=80m, memory=100Mi
```

리소스 할당량이 리소스 요청에 적용되는 경우 포드는 리소스 요청을 정의해야 합니다. 리소스 할당량이 리소스 제한에 적용되는 경우 포드는 리소스 제한도 정의해야 합니다. 할당량을 사용하지 않는 경우에도 리소스 요청 및 제한을 정의하는 것이 좋습니다.

## 요청, 제한 및 실제 사용량 보기

클러스터 관리자는 OpenShift 명령줄 인터페이스를 사용하여 개별 노드에서 컴퓨팅 사용 정보를 볼 수 있습니다. **oc describe node** 명령은 노드에서 실행되는 포드에 대한 정보를 포함하여 노드에 대한 자세한 정보를 표시합니다. 각 포드에는 CPU 요청 및 제한뿐만 아니라 메모리 요청 및 제한도 표시됩니다. 요청 또는 제한이 지정되지 않은 경우 포드는 해당 열에 대해 0을 표시합니다. 모든 리소스 요청 및 제한에 대한 요약도 표시됩니다.

```
[user@demo ~]$ oc describe node node1.us-west-1.compute.internal
Name: node1.us-west-1.compute.internal
Roles: worker
Labels: beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/instance-type=m4.xlarge
 beta.kubernetes.io/os=linux
...output omitted...
Non-terminated Pods: (20 in total)
... Name CPU Requests ... Memory Requests Memory Limits AGE
... --- ----- ... ----- ----- ---
... tuned-vdwt4 10m (0%) ... 50Mi (0%) 0 (0%) 8d
... dns-default-2rpwf 110m (3%) ... 70Mi (0%) 512Mi (3%) 8d
... node-ca-6xwmn 10m (0%) ... 10Mi (0%) 0 (0%) 8d
...output omitted...
Resource Requests Limits
----- -----
cpu 600m (17%) 0 (0%)
memory 1506Mi (9%) 512Mi (3%)
...output omitted...
```



### 참고

요청 및 제한에 대한 요약 열에는 정의된 요청 및 제한의 총 합계가 표시됩니다. 위 출력의 경우 노드에서 실행되는 포드 20개 중 1개에만 메모리 제한이 정의되었으며 해당 제한은 512Mi입니다.

**oc describe node** 명령은 요청과 제한을 표시하고 **oc adm top** 명령은 실제 사용량을 표시합니다.

예를 들어 포드에서 CPU의 10m를 요청하는 경우 스케줄러는 사용 가능한 용량이 있는 노드에 포드를 배치하도록 보장합니다. 포드는 CPU의 10m를 요청했지만 CPU 제한으로 제한되지 않는 한 이 값보다 많거나 적을 수 있습니다. 마찬가지로 리소스 요청을 지정하지 않는 포드는 여전히 일부 리소스를 사용합니다.

**oc adm nodes** 명령은 클러스터에 있는 하나 이상의 노드에 대한 실제 사용량을 표시하고 **oc adm top pods** 명령은 프로젝트의 각 포드에 대한 실제 사용량을 표시합니다.

```
[user@demo ~]$ oc adm top nodes -l node-role.kubernetes.io/worker
NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%
node1.us-west-1.compute.internal 519m 14% 3126Mi 20%
node2.us-west-1.compute.internal 167m 4% 1178Mi 7%
```

## 할당량 적용

OpenShift Container Platform은 두 가지 종류의 리소스 사용을 추적하고 제한하는 할당량을 시행할 수 있습니다.

### 오브젝트 개수

포드, 서비스, 경로 등의 Kubernetes 리소스.

## 컴퓨팅 리소스

CPU, 메모리 및 스토리지 용량 등 실제 또는 가상 하드웨어 리소스의 수.

Kubernetes 리소스 수에 할당량을 설정하면 **Etcd** 데이터베이스의 바인딩되지 않은 증가가 방지되어 OpenShift 컨트롤 플레인의 안정성이 향상됩니다. 또한 Kubernetes 리소스에 할당량을 적용하면 서비스의 IP 주소 등 다른 제한된 소프트웨어 리소스의 고갈을 방지할 수 있습니다.

유사한 방식으로 컴퓨팅 리소스의 양에 할당량을 부과하면 OpenShift 클러스터에서 단일 노드의 컴퓨팅 용량 사용을 방지할 수 있습니다. 또한 모든 클러스터 용량을 사용하여 하나의 애플리케이션이 공유된 클러스터에서 다른 애플리케이션을 사용하지 않도록 방지할 수 있습니다.

OpenShift는 **ResourceQuota** 리소스 또는 **할당량**을 사용하여 클러스터에서 리소스 수 및 컴퓨팅 리소스 사용에 대한 할당량을 관리합니다. 할당량은 프로젝트에 대한 하드 리소스 사용 제한을 지정합니다. 할당량의 모든 특성은 선택적으로, 할당량으로 제한되지 않은 모든 리소스를 바운드 없이 사용할 수 있음을 의미합니다.



### 참고

단일 할당량 리소스로 프로젝트의 할당량을 모두 정의할 수 있지만, 하나의 프로젝트에 할당량이 여러 개 포함될 수도 있습니다. 예를 들어, 하나의 할당량 리소스는 허용된 총 CPU 또는 허용된 총 메모리와 같이 계산 리소스를 제한할 수 있습니다. 다른 할당량 리소스는 허용되는 포드 수 또는 허용되는 서비스 수와 같은 오브젝트 개수를 제한할 수 있습니다. 여러 할당량의 효과는 누적되지만 동일한 프로젝트의 두 가지 다른 **ResourceQuota** 리소스가 동일한 유형의 Kubernetes 또는 컴퓨팅 리소스 사용을 제한하지 않는다고 예상됩니다. 예를 들어 프로젝트에 있는 두 개의 다른 할당량은 허용되는 최대 포드 수를 제한하지 않아야 합니다.

다음 테이블에서는 할당량이 개수 또는 번호로 제한될 수 있는 일부 리소스에 대해 설명합니다.

리소스 이름	할당량 설명
포드	총 포드 수
replicationcontrollers	총 복제 컨트롤러 수
서비스	총 서비스 수
secrets	총 시크릿 수
persistentvolumeclaims	총 영구 볼륨 클레임 수

다음 테이블에서는 할당량으로 제한할 수 있는 일부 컴퓨팅 리소스에 대해 설명합니다.

컴퓨팅 리소스 이름	할당량 설명
cpu(requests.cpu)	모든 컨테이너의 총 CPU 사용량
메모리(requests.memory)	모든 컨테이너의 총 메모리 사용량
스토리지(requests.storage)	모든 영구 볼륨 클레임의 컨테이너별 총 스토리지 요청

할당량 속성은 프로젝트에 있는 모든 포드의 리소스 요청 또는 리소스 제한을 추적할 수 있습니다. 기본적으로 할당량 특성은 리소스 요청을 추적합니다. 대신 리소스 제한을 추적하려면 계산 리소스 이름 앞에 **limits**를 붙입니다(예: **limits.cpu**).

다음 목록에는 YAML 구문을 사용하여 정의된 **ResourceQuota** 리소스가 표시됩니다. 이 예제에서는 리소스 수와 계산 리소스 사용에 대한 할당량을 지정합니다.

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: dev-quota
spec:
 hard:
 services: "10"
 cpu: "1300m"
 memory: "1.5Gi"
```

리소스 단위는 포드 리소스 요청 및 리소스 제한에서 동일합니다. 예를 들어, **Gi**는 GiB를, **m**은 millicores를 나타냅니다. 1밀리코어는 단일 CPU 코어의 1/1000에 해당합니다.

리소스 할당량은 다른 OpenShift Container Platform 리소스와 동일한 방식, 즉 YAML 또는 JSON 리소스 정의 파일을 **oc create** 명령에 전달하는 방식으로 생성할 수 있습니다.

```
[user@demo ~]$ oc create --save-config -f dev-quota.yml
```

리소스 할당량을 생성하는 다른 방법은 **oc create quota** 명령을 사용하는 것입니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc create quota dev-quota --hard services=10,cpu=1300,memory=1.5Gi
```

**oc get resourcequota** 명령을 사용하여 사용 가능한 할당량을 나열하고 **oc describe resourcequota** 명령을 사용하여 할당량에 정의된 하드 제한과 관련된 사용량 통계를 확인합니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc get resourcequota
NAME AGE REQUEST
compute-quota 51s cpu: 500m/10, memory: 300Mi/1Gi ...
count-quota 28s pods: 1/3, replicationcontrollers: 1/5, services: 1/2 ...
```

인수 없이 **oc describe quota** 명령은 프로젝트의 모든 **ResourceQuota** 리소스에 대해 설정된 누적 제한을 표시합니다.

```
[user@demo ~]$ oc describe quota
Name: compute-quota
Namespace: schedule-demo
Resource Used Hard

cpu 500m 10
memory 300Mi 1Gi

Name: count-quota
Namespace: schedule-demo
Resource Used Hard

```

Pods	1	3
replicationcontrollers	1	5
services	1	2

활성 할당량은 `oc delete` 명령을 사용하여 이름별로 삭제할 수 있습니다.

```
[user@demo ~]$ oc delete resourcequota QUOTA
```

프로젝트에 할당량이 처음 생성되면 업데이트된 사용량 통계를 계산할 때까지 프로젝트가 할당량 제한조건을 위반할 수 있는 새 리소스를 생성하는 기능을 제한합니다. 할당량이 생성되고 사용량 통계가 최신 상태이면 프로젝트에서 콘텐츠 생성을 허용합니다. 새 리소스를 생성하면 할당량 사용이 즉시 증가합니다. 리소스를 삭제하면 다음에 프로젝트의 할당량 통계를 모두 다시 계산하는 동안 할당량 사용이 감소합니다.

할당량은 새 리소스에 적용되지만 기존 리소스에는 영향을 주지 않습니다. 예를 들어 프로젝트를 15개의 포드로 제한하는 할당량을 생성하지만 이미 20개의 포드가 실행 중인 경우 할당량을 초과하는 5개의 추가 포드는 제거되지 않습니다.



### 중요

**ResourceQuota** 제한조건이 프로젝트 전체에 적용되지만, 빌드 및 배포와 같은 여러 OpenShift 프로세스에서 프로젝트에 포드를 생성합니다. 따라서 포드를 시작하면 프로젝트 할당량이 초과되므로 해당 프로세스가 실패할 수 있습니다.

프로젝트를 수정한 결과 리소스 개수 할당량을 초과하는 경우에는 OpenShift에서 해당 작업을 거부하고 사용자에게 적절한 오류 메시지를 반환합니다. 하지만 수정한 결과 컴퓨팅 리소스의 할당량을 초과하는 경우 작업은 즉시 실패하지 않습니다. OpenShift는 작업을 여러 번 시도하여 관리자에게 할당량을 증가하거나 새 노드를 온라인으로 가져오는 등의 다른 관련 작업을 수행할 수 있는 기회를 제공합니다.



### 중요

프로젝트에 대한 컴퓨팅 리소스 사양을 제한하는 할당량이 설정되면 OpenShift가 해당 컴퓨팅 리소스에 대한 리소스 요청 또는 리소스 제한을 지정하지 않는 포드 생성을 거절합니다. 할당량으로 제한된 프로젝트와 함께 대부분의 템플릿 및 빌더를 사용하려면 프로젝트는 컨테이너 리소스 요청에 대한 기본값을 지정하는 한계 범위 리소스도 포함해야 합니다.

## 한계 범위 적용

**LimitRange** 리소스(**limit**이라고도 함)는 계산 리소스 요청에 대해 기본, 최소 및 최대 값과 프로젝트 내부에 정의된 단일 포드 또는 컨테이너에 대한 제한을 정의합니다. 포드의 리소스 요청 또는 한계는 컨테이너의 합계입니다.

제한 범위 및 리소스 할당량 간의 차이를 이해하려면 제한 범위가 단일 포드의 유효한 범위 및 기본값을 정의함을 고려하십시오. 리소스 할당량은 프로젝트에서 모든 포드의 합계에 대한 상위 값만 정의합니다. OpenShift 클러스터의 리소스 사용과 관련하여 클러스터 관리자는 일반적으로 프로젝트에 대한 한계 및 할당량을 모두 정의합니다.

제한 범위 리소스는 또한 이미지, 이미지 스트림 또는 영구 볼륨 클레임에서 요청하는 스토리지 용량에 대한 기본, 최소 및 최대값을 정의할 수 있습니다. 프로젝트에 추가된 리소스가 컴퓨팅 리소스 요청을 제공하지 않는 경우 프로젝트의 제한 범위에서 제공하는 기본값을 가져옵니다. 새 리소스가 프로젝트 제한 범위에서 지정한 최소값보다 작은 컴퓨팅 리소스 요청 또는 제한을 제공하는 경우 리소스가 생성되지 않습니다. 마찬가지로 새 리소스가 프로젝트 제한 범위에서 지정한 최대값보다 큰 계산 리소스 요청 또는 제한을 제공하는 경우 리소스가 생성되지 않습니다.

다음 목록에는 YAML 구문을 사용하여 정의된 한계 범위가 표시됩니다.

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
 name: "dev-limits"
spec:
 limits:
 - type: "Pod"
 max: ❶
 cpu: "500m"
 memory: "750Mi"
 min: ❷
 cpu: "10m"
 memory: "5Mi"
 - type: "Container"
 max: ❸
 cpu: "500m"
 memory: "750Mi"
 min: ❹
 cpu: "10m"
 memory: "5Mi"
 default: ❺
 cpu: "100m"
 memory: "100Mi"
 defaultRequest: ❻
 cpu: "20m"
 memory: "20Mi"
 - type: openshift.io/Image ❼
 max:
 storage: 1Gi
 - type: openshift.io/ImageStream ❽
 max:
 openshift.io/image-tags: 10
 openshift.io/images: 20
 - type: "PersistentVolumeClaim" ❾
 min:
 storage: "1Gi"
 max:
 storage: "50Gi"

```

- ❶ 포드 내의 모든 컨테이너에서 사용할 수 있는 최대 CPU 및 메모리 양입니다. 최대 제한을 초과하는 새 포드는 생성되지 않습니다. 최대 제한을 초과하는 기존 포드는 재시작됩니다.
- ❷ 포드 내의 모든 컨테이너에서 사용할 수 있는 최소 CPU 및 메모리 양입니다. 최소 요구 사항을 충족하지 않는 포드는 생성되지 않습니다. 대부분의 포드에는 컨테이너가 하나만 있기 때문에 최소 포드 값을 최소 컨테이너 값과 동일한 값으로 설정할 수 있습니다.
- ❸ 포드 내의 개별 컨테이너에서 사용할 수 있는 최대 CPU 및 메모리 양입니다. 최대 제한을 초과하는 새 컨테이너에서는 연결된 포드가 생성되지 않습니다. 최대 제한을 초과하는 기존 컨테이너는 전체 포드를 재시작합니다.
- ❹ 포드 내의 개별 컨테이너에서 사용할 수 있는 최소 CPU 및 메모리 양입니다. 최소 요구 사항을 충족하지 않는 컨테이너에서는 연결된 포드가 생성되지 않습니다.
- ❺ 개별 컨테이너에서 사용할 수 있는 기본 최대 CPU 및 메모리 양입니다. 이 값은 컨테이너에 CPU 리소스 제한 또는 메모리 제한이 정의되지 않은 경우 사용됩니다.

- ⑥ 개별 컨테이너 요청에 대한 기본 CPU 및 메모리입니다. 이 기본값은 컨테이너에 CPU 리소스 요청 또는 메모리 요청이 정의되지 않은 경우 사용됩니다. 네임스페이스에 CPU 및 메모리 할당량을 사용할 수 있는 경우에는 컨테이너에서 리소스 요청을 지정하지 않아도 **defaultRequest** 섹션을 구성하여 포드를 시작할 수 있습니다.
- ⑦ 내부 레지스트리로 내보낼 수 있는 최대 이미지 크기입니다.
- ⑧ 이미지 스트림 리소스에서 참조할 수 있는 최대 이미지 태그 및 버전 수입니다.
- ⑨ 영구 볼륨 클레임에 사용할 수 있는 최소 및 최대 크기입니다.

사용자는 다른 OpenShift 리소스와 동일한 방식, 즉 YAML 또는 JSON 리소스 정의 파일을 **oc create** 명령에 전달하는 방식으로 제한 범위 리소스를 생성할 수 있습니다.

```
[user@demo ~]$ oc create --save-config -f dev-limits.yml
```

Red Hat OpenShift Container Platform은 리소스 할당량에 대해서는 **oc create** 명령을 제공하지만, 제한 범위에 대해서는 특별히 이 명령을 제공하지 않습니다. 유일한 대안은 YAML 또는 JSON 파일을 사용하는 것입니다.

**oc describe limitrange** 명령을 사용하여 프로젝트에 적용된 한계 제한사항을 확인합니다.

```
[user@demo ~]$ oc describe limitrange dev-limits
Name: dev-limits
Namespace: schedule-demo
Type Resource Min Max Default Request ...
----- ----- --- --- ----- ...
Pod cpu 10m 500m - ...
Pod memory 5Mi 750Mi - ...
Container memory 5Mi 750Mi 20Mi ...
Container cpu 10m 500m 20m ...
openshift.io/Image storage - 1Gi - ...
openshift.io/ImageStream openshift.io/image-tags - 10 - ...
openshift.io/ImageStream openshift.io/images - 20 - ...
PersistentVolumeClaim storage 1Gi 50Gi - ...
```

활성 제한 범위는 **oc delete** 명령을 사용하여 이름별로 삭제할 수 있습니다.

```
[user@demo ~]$ oc delete limitrange dev-limits
```

제한 범위가 프로젝트에 생성되고 나면 프로젝트의 각 한계 범위 리소스에 대해 모든 새로운 리소스 생성 요청을 평가합니다. 새 리소스가 제한 범위를 통해 열거한 최소 또는 최대 제한사항을 위반하는 경우 리소스가 거부됩니다. 새 리소스에서 명시적으로 값을 설정하지 않고 제한조건에서 기본값을 지원하면 기본값이 사용 값으로 새 리소스에 적용됩니다.

프로젝트의 각 제한 범위 리소스에 대해 모든 리소스 업데이트 요청을 평가합니다. 업데이트된 리소스가 제한 조건을 위반하면 업데이트가 거절됩니다.



### 중요

너무 높은 **LimitRange** 제한조건 또는 너무 낮은 **ResourceQuota** 제한조건 설정은 피하십시오. **LimitRange** 제한조건을 위반하면 포드가 생성되지 않고 오류 메시지가 표시됩니다. **ResourceQuota** 제한조건을 위반하면 포드가 어떤 노드에도 예약되지 않습니다. 포드가 생성될 수 있지만 보류 상태로 유지됩니다.

## 여러 프로젝트에 할당량 적용

**ClusterResourceQuota** 리소스는 영구 볼륨과 유사한 방식으로 클러스터 수준에서 생성되며 여러 프로젝트에 적용되는 리소스 제한조건을 지정합니다.

클러스터 관리자는 다음의 두 가지 방식으로 클러스터 리소스 할당량 대상인 프로젝트를 지정할 수 있습니다.

- **openshift.io/requester** 주석을 사용하여 프로젝트 소유자를 지정합니다. 지정된 소유자가 있는 모든 프로젝트는 할당량이 적용됩니다.
- 선택기를 사용합니다. 레이블에 선택기와 일치하는 모든 프로젝트는 할당량이 적용됩니다.

다음은 **qa** 사용자가 소유한 모든 프로젝트에 대한 클러스터 리소스 할당량 생성의 예입니다.

```
[user@demo ~]$ oc create clusterquota user-qa \
> --project-annotation-selector openshift.io/requester=qa \
> --hard pods=12, secrets=20
```

다음은 **environment=qa** 레이블이 할당된 모든 프로젝트에 대한 클러스터 리소스 할당량 생성의 예입니다.

```
[user@demo ~]$ oc create clusterquota env-qa \
> --project-label-selector environment=qa \
> --hard pods=10, services=5
```

프로젝트 사용자는 **oc describe QUOTA** 명령을 사용하여 현재 프로젝트에 적용되는 클러스터 리소스 할당량을 볼 수 있습니다.

**oc delete** 명령을 사용하여 클러스터 리소스 할당량을 삭제합니다.

```
[user@demo ~]$ oc delete clusterquota QUOTA
```



### 참고

대규모 잠금 오버헤드를 방지하려면 단일 클러스터 리소스 할당량이 100개 이하의 프로젝트와 일치하도록 설정하는 것이 좋습니다. 프로젝트 리소스를 업데이트하거나 생성할 때 해당하는 모든 리소스 할당량을 검색하는 동안 해당 프로젝트가 잠깁니다.

## 기본 프로젝트 템플릿 사용자 지정

클러스터 관리자는 기본 프로젝트 템플릿을 사용자 지정할 수 있습니다. 사용자가 프로젝트를 새로 생성하면 할당량, 제한 범위, 네트워크 정책 등의 추가 리소스가 생성됩니다.

클러스터 관리자로 **oc adm create-bootstrap-project-template** 명령을 사용하여 새 프로젝트 템플릿을 만들고 해당 출력을 파일로 리디렉션합니다.

```
[user@demo ~]$ oc adm create-bootstrap-project-template \
> -o yaml > /tmp/project-template.yaml
```

템플릿 리소스 파일을 사용자 지정하여 할당량, 제한 범위, 네트워크 정책 등의 추가 리소스를 추가합니다. 할당량은 클러스터 관리자가 구성하며 프로젝트 관리자는 추가, 수정 또는 삭제할 수 없습니다. 해당 리소스가 프로젝트 템플릿으로 생성되는 경우에도 프로젝트 관리자는 제한 범위 및 네트워크 정책을 수정하고 삭제할 수 있습니다.

새 프로젝트에서는 **objects** 섹션에 지정된 리소스를 생성합니다. 추가 오브젝트는 **Project** 및 **RoleBinding** 리소스와 동일하게 들여쓰기해야 합니다.

다음 예제에서는 프로젝트 이름을 사용하여 할당량을 만듭니다. 할당량은 해당 프로젝트에 CPU 3개, 메모리 10GB, 포드 10개를 부과합니다.

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
 creationTimestamp: null
 name: project-request
objects:
- apiVersion: project.openshift.io/v1
 kind: Project
 ...output omitted...
- apiVersion: rbac.authorization.k8s.io/v1
 kind: RoleBinding
 ...output omitted...
- apiVersion: v1
 kind: ResourceQuota
 metadata:
 name: ${PROJECT_NAME}-quota
 spec:
 hard:
 cpu: "3"
 memory: 10Gi
 pods: "10"
parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER
```

`oc create` 명령을 사용하여 **openshift-config** 네임스페이스에 새 템플릿 리소스를 생성합니다.

```
[user@demo ~]$ oc create -f /tmp/project-template.yaml -n openshift-config
template.template.openshift.io/project-request created
```

새 프로젝트 템플릿을 사용하도록 **projects.config.openshift.io/cluster** 리소스를 업데이트 합니다. **spec** 섹션을 수정합니다. 기본적으로 프로젝트 템플릿 이름은 **project-request**입니다.

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

`projects.config.openshift.io/cluster` 리소스를 성공적으로 업데이트하면 `openshift-apiserver` 네임스페이스에 새 `apiserver` 포드가 생성됩니다. 새 `apiserver` 포드가 실행되면 새 프로젝트에서 사용자 지정 프로젝트 템플릿에 지정된 리소스를 생성합니다.

원래 프로젝트 템플릿으로 되돌리려면 `projects.config.openshift.io/cluster` 리소스를 수정하여 `spec: {}`과 일치하도록 `spec` 리소스를 정리합니다.



### 참조

자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/applications/index#quotas](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/applications/index#quotas)

에 있는 Red Hat OpenShift Container Platform 4.5 애플리케이션 설명서의 할당량 장을 참조하십시오.

제한 범위에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서의 클러스터 작업 장에서 제한 범위 설정 섹션을 참조하십시오.  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-cluster-limit-ranges](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-cluster-limit-ranges)

### OpenShift 프로젝트 생성 사용자 지정

<https://developers.redhat.com/blog/2020/02/05/customizing-openshift-project-creation/>

## ▶ 연습 가이드

# 애플리케이션의 리소스 사용 제한

이 연습에서는 클러스터 및 해당 작업자 노드에서 모든 컴퓨팅 리소스를 사용하지 않도록 애플리케이션을 구성합니다.

### 결과

OpenShift 명령줄 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- CPU 및 메모리 사용량에 대한 리소스 요청을 지정하도록 애플리케이션을 구성합니다.
- 기존 클러스터 제한 내에서 작동하도록 애플리케이션을 수정합니다.
- 프로젝트에 사용할 수 있는 CPU, 메모리 및 구성 맵의 총 양을 제한하도록 할당량을 만듭니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

명령을 사용하면 클러스터 API에 연결할 수 있으며 활동에서 사용할 리소스 파일이 생성됩니다.

```
[student@workstation ~]$ lab schedule-limit start
```

#### ▶ 1. developer 사용자로 **schedule-limit**라는 새 프로젝트를 만듭니다.

- 1.1. OpenShift 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **schedule-limit**라는 이 단계별 연습에 대한 새 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project schedule-limit
Now using project "schedule-limit" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

#### ▶ 2. CPU 및 메모리에 대한 컨테이너 리소스를 명시적으로 요청하는 이 연습에 대한 테스트 애플리케이션을 배포합니다.

- 2.1. 배포 리소스 파일을 만들고 **~/D0280/labs/schedule-limit/hello-limit.yaml**에 저장합니다. 애플리케이션에 **hello-limit**라는 이름을 지정하고 **quay.io/redhattraining/hello-world-nginx:v1.0**에 있는 컨테이너 이미지를 사용합니다.

```
[student@workstation ~]$ oc create deployment hello-limit \
> --image quay.io/redhattraining/hello-world-nginx:v1.0 \
> --dry-run=client -o yaml > ~/D0280/labs/schedule-limit/hello-limit.yaml
```

- 2.2. ~/D0280/labs/schedule-limit/hello-limit.yaml을 편집하여 resources: {} 행을 아래에 강조 표시된 행으로 교체합니다. 파일을 저장하기 전에 적절한 들여쓰기가 있는지 확인합니다.

```
[student@workstation ~]$ vim ~/D0280/labs/schedule-limit/hello-limit.yaml

...output omitted...

spec:
 containers:
 - image: quay.io/redhattraining/hello-world-nginx:v1.0
 name: hello-world-nginx
 resources:
 requests:
 cpu: "3"
 memory: 20Mi
 status: {}
```

- 2.3. 리소스 파일을 사용하여 새 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc create --save-config \
> -f ~/D0280/labs/schedule-limit/hello-limit.yaml
deployment.apps/hello-limit created
```

- 2.4. 애플리케이션에 대한 새 배포가 생성되었지만 애플리케이션 포드는 Pending(보류 중) 상태여야 합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-limit-d86874d86b-fpmrt 0/1 Pending 0 10s
```

- 2.5. 충분한 CPU 리소스가 있는 작업자 노드가 없으므로 포드를 예약할 수 없습니다. 경고 이벤트를 보고 확인할 수 있습니다.

```
[student@workstation ~]$ oc get events --field-selector type=Warning
LAST SEEN TYPE REASON OBJECT MESSAGE
88s Warning FailedScheduling pod/hello-limit-d86874d86b-fpmrt 0/3
nodes are available: 3 Insufficient cpu.
```

### ▶ 3. CPU 리소스를 더 적게 요청하도록 애플리케이션을 다시 배포합니다.

- 3.1. ~/D0280/labs/schedule-limit/hello-limit.yaml을 편집하여 컨테이너에 1.2 개의 CPU를 요청합니다. 아래의 강조 표시된 행과 일치하도록 cpu: "3" 행을 변경합니다.

```
[student@workstation ~]$ vim ~/D0280/labs/schedule-limit/hello-limit.yaml

...output omitted...

resources:
 requests:
 cpu: "1200m"
 memory: 20Mi
```

- 3.2. 변경 사항을 애플리케이션에 적용합니다.

```
[student@workstation ~]$ oc apply -f ~/D0280/labs/schedule-limit/hello-limit.yaml
deployment.apps/hello-limit configured
```

- 3.3. 애플리케이션이 성공적으로 배포하는지 확인합니다. 실행 중인 포드가 표시될 때까지 **oc get pods**를 여러 번 실행해야 할 수 있습니다. 보류 중인 상태가 있는 이전 포드는 종료되며 결국 사라집니다.

NAME	READY	STATUS	RESTARTS	AGE
hello-limit-d86874d86b-fpmrt	0/1	Terminating	0	2m19s
hello-limit-7c7998ff6b-ctsjp	1/1	Running	0	6s



### 참고

애플리케이션 포드가 여전히 예약되지 않는 경우 **~/D0280/labs/schedule-limit/hello-limit.yaml** 파일을 수정하여 CPU 요청을 **1100m**으로 줄입니다. 변경 사항을 다시 적용하고 포드 상태가 **Running**인지 확인합니다.

- ▶ 4. 애플리케이션 크기를 포드 1개에서 4개로 확장해 보십시오. 이 변경 사항이 클러스터 용량을 초과하는지 확인한 후 **hello-limit** 애플리케이션과 연결된 리소스를 삭제합니다.

- 4.1. **hello-limit** 애플리케이션을 최대 4개의 포드까지 수동으로 확장합니다.

```
[student@workstation ~]$ oc scale --replicas 4 deployment/hello-limit
deployment.apps/hello-limit scaled
```

- 4.2. 포드 4개가 모두 실행 중인지 확인합니다. 하나 이상의 포드가 보류 중임이 표시될 때까지 **oc get pods**를 여러 번 실행해야 할 수 있습니다. 현재 클러스터 부하에 따라 여러 개의 포드가 보류 중 상태일 수 있습니다.



### 참고

애플리케이션 포드가 여전히 배포되지 않는 경우 애플리케이션 포드 수를 0으로 조정한 다음 **~/D0280/labs/schedule-limit/hello-limit.yaml** 파일을 수정하여 CPU 요청을 **1000m**으로 줄입니다. **oc apply -f ~/D0280/labs/schedule-limit/hello-limit.yaml**을 실행하여 변경 사항을 적용한 다음 **oc scale** 명령을 다시 실행하여 4개의 포드를 확장합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-limit-d55cd65c-765s9 1/1 Running 0 12s
hello-limit-d55cd65c-hmblw 0/1 Pending 0 12s
hello-limit-d55cd65c-r81vw 1/1 Running 0 12s
hello-limit-d55cd65c-vkrhk 0/1 Pending 0 12s
```

- 4.3. 경고 이벤트는 충분한 CPU 리소스가 있는 노드가 없으므로 하나 이상의 포드를 예약할 수 없음을 나타냅니다. 경고 메시지는 약간 다를 수 있습니다.

```
[student@workstation ~]$ oc get events --field-selector type=Warning
LAST SEEN TYPE REASON OBJECT
MESSAGE
...output omitted...
76s Warning FailedScheduling pod/hello-limit-d55cd65c-vkrhk 0/3
nodes are available: 3 Insufficient cpu.
```

- 4.4. `hello-limit` 애플리케이션 관련 리소스를 모두 삭제합니다.

```
[student@workstation ~]$ oc delete all -l app=hello-limit
pod "hello-limit-d55cd65c-765s9" deleted
pod "hello-limit-d55cd65c-hmblw" deleted
pod "hello-limit-d55cd65c-r81vw" deleted
pod "hello-limit-d55cd65c-vkrhk" deleted
deployment.apps "hello-limit" deleted
replicaset.apps "hello-limit-5cc86ff6b8" deleted
replicaset.apps "hello-limit-7d6bdcc99b" deleted
```

- ▶ 5. 두 번째 애플리케이션을 배포하여 메모리 사용을 테스트합니다. 이 두 번째 애플리케이션에서는 컨테이너별로 메모리 제한을 200MB로 설정합니다.

- 5.1. `/home/student/D0280/labs/schedule-limit/loadtest.yaml`에 있는 리소스 파일을 사용하여 `loadtest` 애플리케이션을 만듭니다. 배포를 만드는 것 외에도 이 리소스 파일은 서비스와 경로를 만듭니다.

```
[student@workstation ~]$ oc create --save-config \
> -f ~/D0280/labs/schedule-limit/loadtest.yaml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

- 5.2. `loadtest` 컨테이너 이미지는 애플리케이션 API에 대한 요청을 만들어 컨테이너의 CPU 또는 메모리 로드를 증가하도록 설계되었습니다. 경로에 사용되는 정규화된 도메인 이름을 식별합니다.

```
[student@workstation ~]$ oc get routes
NAME HOST/PORT ...
loadtest loadtest.apps.ocp4.example.com ...
```

- ▶ 6. 컨테이너에서 처리할 수 있는 추가 메모리 로드를 생성합니다.

- 6.1. 두 개의 추가 터미널 창을 열어 애플리케이션 부하를 지속적으로 모니터링합니다. 첫 번째 터미널에서 애플리케이션 API에 액세스하여 컨테이너에서 추가 메모리 압력을 시뮬레이션합니다. 두 번째 터미널 창에서 `watch oc get pods`를 실행하여 각 포드의 상태를 지속적으로 모니터링합니다.
- 세 번째 터미널 창에서 `watch oc adm top pod`를 실행하여 각 포드의 CPU 및 메모리 사용량을 지속적으로 모니터링합니다.
- 6.2. 첫 번째 터미널 창에서 애플리케이션 API를 사용하여 60초 동안 150MB에서 메모리 로드를 늘립니다. 경로에서 이전에 식별된 정규화된 도메인 이름을 사용합니다. `curl` 명령이 완료될 때 까지 기다리는 동안 다른 두 개의 터미널 창에서 출력을 관찰합니다.

```
[student@workstation ~]$ curl -X GET \
> http://loadtest.apps.ocp4.example.com/api/loadtest/v1/mem/150/60
curl: (52) Empty reply from server
```

- 6.3. 두 번째 터미널 창에서 `watch oc get pods`의 출력을 관찰합니다. 컨테이너는 추가 로드를 처리할 수 있으므로 단일 애플리케이션 포드에 전체 `curl` 요청에 대한 **Running(실행 중)** 상태가 있음을 확인해야 합니다.

```
Every 2.0s: oc get pods
...
NAME READY STATUS RESTARTS AGE
loadtest-f7495948-tlxgm 1/1 Running 0 7m34s
```

- 6.4. 세 번째 터미널 창에서 `watch oc adm top pod`의 출력을 관찰합니다. 포드의 시작 메모리 사용량은 약 20~30Mi입니다.

```
Every 2.0s: oc adm top pod
...
NAME CPU(cores) MEMORY(bytes)
loadtest-f7495948-tlxgm 0m 20Mi
```

API 요청이 생성되면 포드 메모리 사용량이 약 170~180Mi로 증가하는 것을 확인할 수 있습니다.

```
Every 2.0s: oc adm top pod
...
NAME CPU(cores) MEMORY(bytes)
loadtest-f7495948-tlxgm 0m 172Mi
```

`curl` 요청이 완료되면 잠시 후에 메모리 사용량이 약 20~30Mi로 다시 감소되는 것을 확인할 수 있습니다.

```
Every 2.0s: oc adm top pod
...
NAME CPU(cores) MEMORY(bytes)
loadtest-f7495948-tlxgm 0m 20Mi
```

## ▶ 7. 컨테이너에서 처리할 수 없는 추가 메모리 로드를 생성합니다.

- 7.1. 애플리케이션 API를 사용하여 60초 동안 200MB에서 메모리 로드를 늘립니다. 다른 두 개의 터미널 창에서 출력을 관찰합니다.

```
[student@workstation ~]$ curl -X GET \
> http://loadtest.apps.ocp4.example.com/api/loadtest/v1/mem/200/60
<html><body><h1>502 Bad Gateway</h1>
The server returned an invalid or incomplete response.
</body></html>
```

- 7.2. 두 번째 터미널 창에서 `watch oc get pods`의 출력을 관찰합니다. `curl` 명령을 실행한 직후에는 포드의 상태가 `OOMKilled`로 전환됩니다. `Error(오류)` 상태도 표시될 수 있습니다. 포드에 메모리가 부족하여 종료하고 다시 시작해야 합니다. 상태는 `Running(실행 중)` 상태로 돌아가기 전에 `CrashLoopBackOff`로 변경될 수 있습니다. 또한 재시작 횟수가 증가합니다.

```
Every 2.0s: oc get pods
...
NAME READY STATUS RESTARTS AGE
loadtest-f7495948-tlxgm 0/1 OOMKilled 0 9m13s
```

일부 경우에는 두 번째 터미널 창으로 전환할 시간이 되기 전에 포드를 다시 시작하고 `Running(실행 중)` 상태로 변경했을 수 있습니다. 재시작 횟수가 0에서 1씩 증가합니다.

```
Every 2.0s: oc get pods
...
NAME READY STATUS RESTARTS AGE
loadtest-f7495948-tlxgm 1/1 Running 1 9m33s
```

- 7.3. 세 번째 터미널 창에서 `watch oc adm top pod`의 출력을 관찰합니다. 포드를 종료한 후에는 잠시 동안 지표에 포드에서 리소스를 전혀 또는 거의 사용하지 않는 것으로 표시됩니다.

```
Every 2.0s: oc adm top pod
...
NAME CPU(cores) MEMORY(bytes)
loadtest-f7495948-tlxgm 8m 0Mi
```

- 7.4. 첫 번째 터미널 창에서 두 번째 애플리케이션과 연결된 모든 리소스를 삭제합니다.

```
[student@workstation ~]$ oc delete all -l app=loadtest
pod "loadtest-f7495948-tlxgm" deleted
service "loadtest" deleted
deployment.apps "loadtest" deleted
route.route.openshift.io "loadtest" deleted
```

두 번째 및 세 번째 터미널에서 `Ctrl+C`를 눌러 `watch` 명령을 종료합니다. 선택적으로 두 번째 및 세 번째 터미널 창을 닫습니다.

## ▶ 8. 클러스터 관리자로 `schedule-limit` 프로젝트에 대한 할당량을 만듭니다.

- 8.1. OpenShift 클러스터에 `admin` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 8.2. `schedule-limit` 프로젝트를 3개의 CPU, 메모리 1GB 및 3개의 구성 맵으로 제한하는 `project-quota`라는 할당량을 만듭니다.

```
[student@workstation ~]$ oc create quota project-quota \
> --hard cpu="3",memory="1G",configmaps="3" \
> -n schedule-limit
resourcequota/project-quota created
```



### 참고

이 연습에서는 사용자가 할당량을 초과하려고 할 때 수행되는 작업을 보여 주는 할당량을 구성 맵에 배치합니다.

- ▶ 9. 개발자로서 프로젝트에 대한 구성 맵 할당량을 초과하려고 시도합니다.

- 9.1. OpenShift 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 9.2. 반복문을 사용하여 네 개의 구성 맵을 만듭니다. 처음 세 개는 성공하고 네 번째는 할당량을 초과하므로 실패합니다.

```
[student@workstation ~]$ for X in {1..4}
> do
> oc create configmap my-config${X} --from-literal key${X}=value${X}
> done
configmap/my-config1 created
configmap/my-config2 created
configmap/my-config3 created
Error from server (Forbidden): configmaps "my-config4" is forbidden: exceeded
quota: project-quota, requested: configmaps=1, used: configmaps=3, limited:
configmaps=3
```

- ▶ 10. 클러스터 관리자로 모든 새 프로젝트를 기본 할당량 및 제한 범위 리소스로 구성합니다.

- 10.1. OpenShift 클러스터에 `admin` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 10.2. 프로젝트 생성을 사용자 지정할 수 있도록 `oc adm create-bootstrap-project-template` 명령 출력을 리디렉션합니다.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
> -o yaml > /tmp/project-template.yaml
```

- 10.3. `/tmp/project-template.yaml` 파일을 편집하여 `/home/student/D0280/labs/schedule-limit/quota-limits.yaml` 파일에 정의된 할당량 및 제한 범위 리소스를 추가합니다. `parameters` 섹션 앞에 행을 추가합니다.

```
...output omitted...
- apiVersion: v1
 kind: ResourceQuota
 metadata:
 name: ${PROJECT_NAME}-quota
 spec:
 hard:
 cpu: 3
 memory: 10G
- apiVersion: v1
 kind: LimitRange
 metadata:
 name: ${PROJECT_NAME}-limits
 spec:
 limits:
 - type: Container
 defaultRequest:
 cpu: 30m
 memory: 30M
 parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER
```



### 참고

`/home/student/D0280/solutions/schedule-limit/project-template.yaml` 파일에는 올바른 구성이 포함되어 있으며 비교에 사용할 수 있습니다.

- 10.4. `/tmp/project-template.yaml` 파일을 사용하여 `openshift-config` 네임스페이스에 새 템플릿 리소스를 생성합니다.

```
[student@workstation ~]$ oc create -f /tmp/project-template.yaml \
> -n openshift-config
template.template.openshift.io/project-request created
```

- 10.5. 사용자 지정 프로젝트 템플릿을 사용하도록 클러스터를 업데이트합니다.

```
[student@workstation ~]$ oc edit projects.config.openshift.io/cluster
```

spec 섹션을 수정하여 다음의 굵은 글꼴 행을 사용합니다.

```
...output omitted...
spec:
 projectRequestTemplate:
 name: project-request
```

## 13장 | 포드 스케줄링 제어

들여쓰기가 올바른지 확인한 다음 변경 사항을 저장합니다.

```
project.config.openshift.io/cluster edited
```

- 10.6. 변경이 완료되면 **openshift-apiserver** 네임스페이스에 **apiserver** 포드가 다시 생성됩니다.

```
[student@workstation ~]$ watch oc get pods -n openshift-apiserver
```

새 **apiserver** 포드 3개가 모두 준비되어 실행될 때까지 기다립니다.

Every 2.0s: oc get pods -n openshift-apiserver				
NAME	READY	STATUS	RESTARTS	AGE
apiserver-868dccf5fb-885dz	1/1	Running	0	63s
apiserver-868dccf5fb-8j4vh	1/1	Running	0	39s
apiserver-868dccf5fb-r4j9b	1/1	Running	0	24s

**Ctrl+C**를 눌러 **watch** 명령을 종료합니다.

- 10.7. 테스트 프로젝트를 만들어 사용자 지정 프로젝트 템플릿이 예상대로 작동하는지 확인합니다.

```
[student@workstation ~]$ oc new-project template-test
Now using project "template-test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 10.8. 테스트 프로젝트의 리소스 할당량 및 제한 범위 리소스를 나열합니다.

[student@workstation ~]\$ oc get resourcequotas,limitranges			
NAME	AGE	REQUEST	LIMIT
resourcequota/template-test-quota	87s	cpu: 0/3, memory: 0/10G	

NAME	CREATED AT
limitrange/template-test-limits	2020-09-16T21:13:25Z

- ▶ 11. 이 연습에서 생성한 프로젝트를 삭제하여 랩 환경을 정리합니다.

- 11.1. **schedule-limit** 프로젝트를 삭제합니다.

```
[student@workstation ~]$ oc delete project schedule-limit
project.project.openshift.io "schedule-limit" deleted
```

- 11.2. **template-test** 프로젝트를 삭제합니다.

```
[student@workstation ~]$ oc delete project template-test
project.project.openshift.io "template-test" deleted
```

## 완료

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab schedule-limit finish
```

이로써 랩이 완료됩니다.

# 애플리케이션 크기 조정

## 목표

이 섹션을 마친 수강생은 다음 작업을 할 수 있습니다.

- 포드의 복제 수를 제어합니다.
- 배포 또는 배포 구성 리소스의 복제본 수를 지정합니다.
- 복제본 수를 수동으로 확장합니다.
- HPA(수평선 포드 자동 확장기) 리소스를 만듭니다.

## 구성 워크로드에서 포드 복제본 지정

특정 배포 또는 배포 구성의 포드 복제본 수는 요구 사항에 맞게 늘리거나 줄일 수 있습니다. **ReplicaSet** 및 **ReplicationController** 리소스에도 불구하고 애플리케이션에 필요한 복제본 수는 일반적으로 배포 또는 배포 구성 리소스에 정의되어 있습니다. 복제본 집합 또는 복제본 컨트롤러(배포 또는 배포 구성으로 관리)를 사용하면 항상 지정된 수의 포드 복제본이 실행됩니다. 복제 집합 또는 복제 컨트롤러는 필요에 따라 포드를 추가하거나 제거하여 원하는 복제본 수를 따르도록 할 수 있습니다.

배포 구성과 배포 구성에는 다음이 포함됩니다.

- 원하는 복제본 수
- 관리되는 포드를 식별하기 위한 선택기
- 복제 포드를 생성하는 포드 정의 또는 템플릿(포드에 적용할 레이블 포함)

다음 배포 리소스(`oc create deployment` 명령을 사용하여 만들어짐)에는 다음 항목이 표시됩니다.

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
 replicas: 1 ①
 selector:
 matchLabels:
 deployment: scale ②
 strategy: {}
 template: ③
 metadata:
 labels:
 deployment: scale ④
 spec:
 containers:
 ...output omitted...
```

① 실행할 포드의 원하는 사본(또는 복제본) 수를 지정합니다.

② 복제본 집합(배포용) 또는 복제 컨트롤러(배포 구성용)는 선택기를 사용하여 부하 분산시킬 포드를 찾는 것과 동일한 방식으로 복제 컨트롤러에서 선택기를 사용하여 실행 중인 포드의 수를 셉니다.

- ③ 복제본 집합 또는 복제 컨트롤러가 만드는 포드의 템플릿.
- ④ 템플릿에 의해 생성된 포드의 레이블은 선택기에 사용된 레이블과 일치해야 합니다.

일반적으로 웹 콘솔 또는 `oc new-app` 명령과 `--as-deployment-config` 옵션을 사용하여 생성하는 배포 구성에서는 동일한 정보를 정의하지만 방식이 약간 다릅니다. 배포 구성에서는 포드 메타데이터에서 일치하는 `deploymentconfig` 레이블이 있는 `deploymentconfig` 선택기를 사용합니다.

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
...output omitted...
spec:
 replicas: 1
 selector:
 deploymentconfig: hello
 strategy:
 resources: {}
 template:
 metadata:
 annotations:
 openshift.io/generated-by: OpenShiftNewApp
 labels:
 deploymentconfig: hello
 ...output omitted...
```

배포 또는 배포 구성 리소스가 버전 제어에 있는 경우 리소스 파일에서 `replicas` 행을 수정하고 `oc apply` 명령을 사용하여 변경 사항을 적용합니다.

배포 또는 배포 구성 리소스에 관계없이 선택기는 복제 집합 또는 복제 컨트롤러에서 관리하는 모든 포드가 일치해야 하는 레이블 집합입니다. 배포 또는 배포 구성이 인스턴트화하는 포드 정의에 동일한 레이블 집합이 포함되어어야 합니다. 이 선택기는 필요한 대로 조정하기 위해 이미 실행 중인 포드의 인스턴스 수를 판별하는데 사용합니다.



### 참고

복제 컨트롤러는 부하 또는 트래픽을 추적하지 않으므로 자동 크기 조정을 수행하지 않습니다. 이 섹션 후반에 제시되는 수평 포드 자동 크기 조정 프로그램 리소스는 자동 크기 조정을 관리합니다.

## 수동으로 포드 복제본 수 확장

개발자와 관리자는 프로젝트의 포드 복제본 수를 수동으로 확장하도록 선택할 수 있습니다. 트래픽에서 예상되는 서지에 더 많은 포드가 필요할 수 있으며 클러스터의 다른 곳에서 사용할 수 있는 리소스를 회수하기 위해 포드 개수를 감소시킬 수 있습니다. 포드 복제본 개수를 늘리거나 줄이는 경우 첫 번째 단계는 `oc get` 명령을 사용하여 확장할 적절한 배포 또는 배포 구성(dc)을 식별하는 것입니다.

```
[user@demo ~]$ oc get deployment
NAME READY UP-TO-DATE AVAILABLE AGE
scale 1/1 1 1 8h
```

배포 또는 배포 구성 리소스에 있는 복제본의 수는 `oc scale` 명령을 사용하여 수동으로 변경할 수 있습니다.

```
[user@demo ~]$ oc scale --replicas 5 deployment/scale
```

배포 리소스는 변경 사항을 복제 집합으로 확장합니다. 원하는 새 복제 개수가 기존 개수보다 작거나 큰지 여부에 따라 새 포드(복제본)를 만들거나 기존 항목을 삭제하여 변경 사항에 반응합니다.

직접 복제본 집합 또는 복제 컨트롤러 리소스를 조작할 수 있더라도 권장되는 관례는 배포 또는 배포 구성 리소스를 대신 조작하는 것입니다. 새 배포는 새 복제본 집합 또는 새 복제 컨트롤러를 생성하고 이전 복제본 집합 또는 복제 컨트롤러에 직접 변경한 내용은 무시됩니다.

## 포드 자동 확장

OpenShift는 **HorizontalPodAutoscaler** 리소스 유형으로 애플리케이션 포드에 있는 현재 부하를 기반으로 배포 또는 배포 구성의 자동 확장할 수 있습니다.

수평 포드 자동 확장기 리소스는 OpenShift Metrics 하위 시스템에서 수집한 성능 지표를 사용합니다. Metrics 하위 시스템은 OpenShift 3에서와 같이 별도의 설치가 필요하지 않고 OpenShift 4에 사전 설치됩니다. 배포 또는 배포 구성의 자동으로 조정하려면 수평 포드 자동 확장기에서 사용 비율을 계산할 수 있도록 포드에 대한 리소스 요청을 지정해야 합니다.

수평 포드 자동 확장기 리소스를 생성하는 권장 방식은 **oc autoscale** 명령을 사용하는 것입니다. 예를 들면

```
[user@demo ~]$ oc autoscale dc/hello --min 1 --max 10 --cpu-percent 80
```

이전 명령은 전체 요청된 CPU 사용률의 80% 미만으로 포드를 유지하도록 **hello** 배포에서 복제본의 수를 변경하는 수평 포드 자동 확장기 리소스를 생성합니다.

**oc autoscale** 명령은 배포 또는 배포 구성 이름을 인수(이전 예제에서 **hello**)로 사용하여 수평 포드 자동 확장기 리소스를 생성합니다.



### 참고

메모리 사용률에 대한 자동 확장은 Red Hat OpenShift Container Platform 4.5의 기술 프리뷰 기능으로 이어집니다.

수평 포드 자동 확장기 리소스의 최대값 및 최소값은 대량의 부하를 수용하고 OpenShift 클러스터의 과부하를 방지합니다. 애플리케이션의 부하가 너무 빨리 변경되는 경우 대량의 갑작스러운 사용자 요청에 적응하도록 다수의 여유 포드를 유지하는 것이 권장될 수 있습니다. 반대로 너무 많은 수의 포드는 모든 클러스터 용량을 다 사용하고 동일한 OpenShift 클러스터를 공유하는 다른 애플리케이션에 영향을 줄 수 있습니다.

현재 프로젝트에서 수평 포드 자동 확장기 리소스에 관한 정보를 얻으려면 **oc get** 명령을 사용합니다. 예를 들면 다음과 같습니다.

```
[user@demo ~]$ oc get hpa
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS ...
hello DeploymentConfig/hello <unknown>/80% 1 10 1 ...
scale Deployment/scale 60%/80% 2 10 2 ...
```



### 중요

수평 포드 자동 확장기의 초기 TARGETS 열 값은 <unknown>입니다. <unknown>이 변경되어 현재 사용량에 대한 백분율이 표시될 때까지 최대 5분이 걸릴 수 있습니다.

TARGETS 열에 <unknown> 값이 일관되게 표시되면 배포 또는 배포 구성에서 리소스 요청을 지표로 정의하지 않음을 나타낼 수 있습니다. 수평 포드 자동 확장기에서는 이러한 포드를 확장하지 않습니다.

`oc new-app` 명령을 사용하여 생성한 대부분의 포드는 리소스 요청을 정의하지 않습니다. 따라서 OpenShift 자동 확장기를 사용하면 배포 또는 배포 구성 리소스를 편집하거나 애플리케이션의 사용자 지정 YAML 또는 JSON 리소스 파일을 생성하거나 기본 리소스 요청을 정의하는 프로젝트에 제한 범위 리소스를 추가해야 할 수 있습니다.



### 참조

자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/nodes/index#nodes-pods-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-pods-autoscaling)

에 있는 Red Hat OpenShift Container Platform 4.5 노드 설명서의 포드 작업 장에서 수평 포드 자동 확장기를 사용하여 포드 자동 확장 섹션을 참조하십시오.

## ▶ 연습 가이드

# 애플리케이션 크기 조정

이 연습에서는 애플리케이션 크기를 수동 및 자동으로 조정합니다.

### 결과

OpenShift 명령줄 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- 애플리케이션 크기를 수동으로 조정합니다.
- 용도에 따라 자동으로 확장되도록 애플리케이션을 구성합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

명령을 사용하면 클러스터 API에 연결할 수 있으며 활동에서 사용할 리소스 파일이 생성됩니다.

```
[student@workstation ~]$ lab schedule-scale start
```

- ▶ 1. **developer** 사용자로 **schedule-scale**이라는 새 프로젝트를 만듭니다.

- 1.1. OpenShift 클러스터에 **developer** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **schedule-scale**이라는 이 단계별 연습에 대한 새 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project schedule-scale
Now using project "schedule-scale" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- ▶ 2. CPU 및 메모리에 대한 컨테이너 리소스를 명시적으로 요청하는 이 연습에 대한 테스트 애플리케이션을 배포합니다.

- 2.1. **~/D0280/labs/schedule-scale/loadtest.yaml**에 있는 리소스 파일을 수정하여 CPU 및 메모리 사용량에 대한 요청과 제한을 모두 설정합니다.

```
[student@workstation ~]$ vim ~/D0280/labs/schedule-scale/loadtest.yaml
```

- 2.2. **resources: {}** 행을 아래에 나열된 강조 표시된 행으로 바꿉니다. 파일을 저장하기 전에 적절한 들여쓰기가 있는지 확인합니다.

```
...output omitted...
spec:
 containers:
 - image: quay.io/redhattraining/loadtest:v1.0
 name: loadtest
 resources:
 requests:
 cpu: "25m"
 memory: 25Mi
 limits:
 cpu: "100m"
 memory: 100Mi
 status: []
```

2.3. 리소스 파일을 사용하여 새 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc create --save-config \
> -f ~/DO280/labs/schedule-scale/loadtest.yaml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

2.4. 애플리케이션 포드에 **Running**(실행 중) 상태가 있는지 확인합니다. **oc get pds** 명령을 여러 번 실행해야 할 수도 있습니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
loadtest-5f9565dbfb-jm9md 1/1 Running 0 23s
```

2.5. 애플리케이션 포드에서 리소스 제한 및 요청을 지정하는지 확인합니다.

```
[student@workstation ~]$ oc describe pod/loadtest-5f9565dbfb-jm9md \
> | grep -A2 -E "Limits|Requests"
Limits:
 cpu: 100m
 memory: 100Mi
Requests:
 cpu: 25m
 memory: 25Mi
```

### ▶ 3. 먼저 실행 중인 포드 수를 늘리고 **loadtest** 배포를 수동으로 확장합니다.

3.1. **loadtest** 배포를 최대 5개의 포드까지 확장합니다.

```
[student@workstation ~]$ oc scale --replicas 5 deployment/loadtest
deployment.apps/loadtest scaled
```

3.2. 5개의 애플리케이션 포드 모두 실행 중인지 확인합니다. **oc get pds** 명령을 여러 번 실행해야 할 수도 있습니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
loadtest-5f9565dbfb-22f9s 1/1 Running 0 54s
loadtest-5f9565dbfb-8l2rn 1/1 Running 0 54s
loadtest-5f9565dbfb-jm9md 1/1 Running 0 3m17s
loadtest-5f9565dbfb-lfhns 1/1 Running 0 54s
loadtest-5f9565dbfb-prjkl 1/1 Running 0 54s
```

3.3. **loadtest** 배포를 1개의 포드로 다시 줄입니다.

```
[student@workstation ~]$ oc scale --replicas 1 deployment/loadtest
deployment.apps/loadtest scaled
```

3.4. 하나의 애플리케이션 포드만 실행되고 있는지 확인합니다. 다른 포드가 종료될 때까지 대기하면서 **oc get pods** 명령을 여러 번 실행해야 할 수 있습니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
loadtest-5f9565dbfb-prjkl 1/1 Running 0 72s
```

▶ 4. 부하에 따라 자동으로 크기를 조정하도록 **loadtest** 애플리케이션을 구성한 다음 부하를 적용하여 애플리케이션을 테스트합니다.

4.1. **loadtest** 애플리케이션에 항상 2개의 애플리케이션 포드가 실행되고 있는지 확인하는 수평 포드 자동 확장기를 만듭니다. 이 숫자는 CPU 로드가 50%를 초과하면 최대 10개의 포드까지 증가할 수 있습니다.

```
[student@workstation ~]$ oc autoscale deployment/loadtest \
> --min 2 --max 10 --cpu-percent 50
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

4.2. **loadtest** 수평 포드 자동 확장기에서 **TARGETS** 열에 사용량을 보고할 때까지 기다립니다.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

<unknown>이 백분율로 변경되면 Ctrl+C를 눌러 **watch** 명령을 종료합니다.

```
Every 2.0s: oc get hpa/loadtest
...
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS ...
loadtest Deployment/loadtest 0%/50% 2 10 2 ...
```



### 참고

**TARGETS** 열의 <unknown> 항목이 0%로 변경될 때까지 최대 5분이 걸릴 수 있습니다.  
<unknown> 항목이 변경되지 않으면 **oc describe** 명령을 사용하여 **loadtest** 애플리케이션 컨테이너에서 CPU 리소스를 요청하는지 확인합니다.

- 4.3. **loadtest** 컨테이너 이미지는 애플리케이션 API에 대한 요청을 만들어 컨테이너의 CPU 또는 메모리 로드를 증가하도록 설계되었습니다. 경로에 사용되는 정규화된 도메인 이름을 식별합니다.

```
[student@workstation ~]$ oc get route/loadtest
NAME HOST/PORT ...
loadtest loadtest-schedule-scale.apps.ocp4.example.com ...
```

- 4.4. 애플리케이션 API로 액세스하여 컨테이너에서 추가 CPU 압력을 시뮬레이션합니다. 다음 단계로 이동하고 **curl** 명령이 완료될 때까지 기다립니다.

```
[student@workstation ~]$ curl -X GET \
> http://loadtest-schedule-scale.apps.ocp4.example.com/api/loadtest/v1/cpu/1
curl: (52) Empty reply from server
```

- 4.5. 두 번째 터미널 창을 열고 수평 포드 자동 확장기의 상태를 지속적으로 모니터링합니다.



### 참고

애플리케이션의 증가된 활동은 자동 확장기를 즉시 트리거하지 않습니다. 복제본 수에 대한 변경 사항이 표시되지 않는 경우 잠시 기다립니다.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

부하가 증가하면(대상 열에 표시됨) **복제본**의 개수가 증가하는 것을 확인할 수 있습니다. 다음 단계로 이동할 때까지 1~2분 동안 출력을 관찰합니다. 출력은 아래 표시된 내용과 다를 수 있습니다.

```
Every 2.0s: oc get hpa/loadtest ...
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS ...
loadtest Deployment/loadtest 172%/50% 2 10 9 ...
```



### 참고

수평 포드 자동 확장기 리소스를 신속하게 확장할 수 있지만 확장하는 것이 더 느립니다. **loadtest** 애플리케이션의 크기가 두 개의 포드로 확장될 때까지 기다리지 않고 나머지 연습을 계속 진행합니다.

- ▶ 5. 첫 번째 터미널 창에서 배포 구성 리소스를 사용하는 **scaling**이라는 두 번째 애플리케이션을 만듭니다. 애플리케이션의 크기를 조정한 다음 애플리케이션 포드에서 수신되는 응답을 확인합니다.

- 5.1. **quay.io/redhattraining/scaling:v1.0**에 있는 컨테이너 이미지를 사용하여 **oc new-app** 명령을 사용하여 새 애플리케이션을 만듭니다. **--as-deployment-config** 옵션을 사용하여 기본 **deployment** 리소스 대신 **deploymentconfig(dc)** 리소스를 생성합니다.

```
[student@workstation ~]$ oc new-app --name scaling --as-deployment-config \
> --docker-image quay.io/redhattraining/scaling:v1.0
...output omitted...
```

```
--> Creating resources ...
imagestream.image.openshift.io "scaling" created
deploymentconfig.apps.openshift.io "scaling" created
service "scaling" created
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/scaling'
Run 'oc status' to view your app.
```

- 5.2. 애플리케이션 서비스를 노출하여 애플리케이션을 위한 경로를 만듭니다.

```
[student@workstation ~]$ oc expose svc/scaling
route.route.openshift.io/scaling exposed
```

- 5.3. 애플리케이션에 대한 배포 구성을 사용하여 애플리케이션을 최대 3개의 포드로 확장합니다.

```
[student@workstation ~]$ oc scale --replicas 3 dc/scaling
deploymentconfig.apps.openshift.io/scaling scaled
```

- 5.4. 확장 애플리케이션에 대한 3개의 포드가 모두 실행 중인지 확인하고 연결된 IP 주소를 식별합니다.

```
[student@workstation ~]$ oc get pods -o wide -l deploymentconfig=scaling
NAME READY STATUS RESTARTS AGE IP NODE ...
scaling-1-bm4m2 1/1 Running 0 45s 10.10.0.29 master01 ...
scaling-1-w7whl 1/1 Running 0 45s 10.8.0.45 master03 ...
scaling-1-xqvs2 1/1 Running 0 6m1s 10.9.0.58 master02 ...
```

- 5.5. 확장 애플리케이션으로 요청을 라우팅하는데 사용되는 호스트 이름을 표시합니다.

```
[student@workstation ~]$ oc get route/scaling
NAME HOST/PORT ...
scaling scaling-schedule-scale.apps.ocp4.example.com ...
```

- 5.6. 애플리케이션의 호스트 이름에 액세스하면 PHP 페이지에서는 요청에 응답한 포드의 IP 주소를 출력합니다. 여러 요청을 애플리케이션에 전송하고 응답을 정렬하여 각 포드에 전송되는 요청의 수를 셉니다. ~/D0280/labs/schedule-scale/curl-route.sh에 있는 스크립트를 실행합니다.

```
[student@workstation ~]$ ~/D0280/labs/schedule-scale/curl-route.sh
34 Server IP: 10.10.0.29
34 Server IP: 10.8.0.45
32 Server IP: 10.9.0.58
```

- ▶ 6. 선택 사항: **loadtest** 애플리케이션을 실행하는 수평 포드 자동 확장기에서는 상태를 확인합니다. **watch oc hpa/loadtest** 명령이 두 번째 터미널 창에서 여전히 실행되고 있는 경우로 전환하여 출력을 관찰합니다. 충분한 시간이 경과하면 복제 횟수가 2개로 줄어듭니다. 완료되면 **Ctrl+C**를 눌러 **watch** 명령을 종료한 다음 두 번째 터미널 창을 닫습니다.

```
Every 2.0s: oc get hpa/loadtest ...
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS ...
loadtest Deployment/loadtest 0%/50% 2 10 2 ...
```

- ▶ 7. `schedule-scale` 프로젝트를 삭제하여 랩 환경을 정리합니다.

```
[student@workstation ~]$ oc delete project schedule-scale
project.project.openshift.io "schedule-scale" deleted
```

## 완료

`workstation` 시스템에서 `lab` 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab schedule-scale finish
```

이로써 랩이 완료됩니다.

## ▶ 랩

# 포드 스케줄링 제어

## 성능 체크리스트

이 랩에서는 클러스터 노드의 하위 집합에서 실행하고 부하를 사용하여 확장하도록 애플리케이션을 구성합니다.

## 결과

OpenShift 명령줄 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- 노드에 새 레이블을 추가합니다.
- 지정된 레이블과 일치하는 노드에 포드를 배포합니다.
- 포드에 대한 CPU 및 메모리 리소스를 요청합니다.
- 자동으로 확장되도록 애플리케이션을 구성합니다.
- 할당량을 만들어 프로젝트가 사용할 수 있는 리소스의 양을 제한합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 연습 파일용 디렉터리를 만듭니다.

```
[student@workstation ~]$ lab schedule-review start
```

1. **admin** 사용자로 **tier** 레이블을 사용하여 두 개의 노드에 레이블을 지정합니다. **master01** 노드에 **tier=gold** 레이블을 지정하고, **master02** 노드에 **tier=silver** 레이블을 지정합니다.
2. **developer** 사용자로 전환하고 **schedule-review**라는 새 프로젝트를 만듭니다.
3. **quay.io/redhattraining/loadtest:v1.0**에 있는 컨테이너 이미지를 사용하여 **loadtest**라는 새 애플리케이션을 만듭니다. **loadtest** 애플리케이션은 **tier=silver**로 레이블이 지정된 노드에 배포해야 합니다. 각 컨테이너가 CPU의 **100m** 및 메모리의 **20Mi**를 요청하는지 확인합니다.
4. 자동으로 생성된 기본 호스트 이름을 사용하여 **loadtest**라는 애플리케이션의 경로를 만듭니다. 애플리케이션을 만든 방식에 따라 경로를 만들기 전에 서비스를 만들어야 할 수도 있습니다. 실행 중인 **curl http://loadtest-schedule-review.apps.ocp4.example.com/api/loadtest/v1/healthz**에서 **{"health": "ok"}**를 반환하는 경우 애플리케이션이 예상대로 작동하는 것입니다.
5. CPU 부하가 **70%**를 초과하는 경우 **2**개의 포드에서 최대 **40**개의 포드로 확장하는 **loadtest** 애플리케이션에 대해 **loadtest**라는 수평 포드 자동 확장기를 만듭니다. 다음 명령을 사용하여 수평 포드 자동 확장기를 테스트할 수 있습니다. **curl -X GET http://loadtest-schedule-review.apps.ocp4.example.com/api/loadtest/v1/cpu/3**

**참고**

수평 포드 자동 확장기에서는 이 **loadtest** 애플리케이션을 확장하지만, 최대 40개의 포드에 도달하기 전에 OpenShift 클러스터는 리소스를 모두 실행합니다.

6. admin 사용자로 **schedule-review** 프로젝트에서 **review-quota**라는 할당량을 구현합니다. **schedule-review** 프로젝트를 최대 1개의 전체 CPU, 2G의 메모리 및 20개의 포드로 제한합니다.

**평가**

다음 **lab** 명령을 실행하여 작업을 확인합니다. **lab** 명령이 오류를 보고하면 변경 사항을 검토하고 수정 사항을 만들고 성공할 때까지 **lab** 명령을 실행합니다.

```
[student@workstation ~]$ lab schedule-review grade
```

**완료**

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab schedule-review finish
```

이로써 랩이 완료됩니다.

## ▶ 솔루션

# 포드 스케줄링 제어

### 성능 체크리스트

이 랙에서는 클러스터 노드의 하위 집합에서 실행하고 부하를 사용하여 확장하도록 애플리케이션을 구성합니다.

### 결과

OpenShift 명령줄 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- 노드에 새 레이블을 추가합니다.
- 지정된 레이블과 일치하는 노드에 포드를 배포합니다.
- 포드에 대한 CPU 및 메모리 리소스를 요청합니다.
- 자동으로 확장되도록 애플리케이션을 구성합니다.
- 할당량을 만들어 프로젝트가 사용할 수 있는 리소스의 양을 제한합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 연습 파일용 디렉터리를 만듭니다.

```
[student@workstation ~]$ lab schedule-review start
```

1. **admin** 사용자로 **tier** 레이블을 사용하여 두 개의 노드에 레이블을 지정합니다. **master01** 노드에 **tier=gold** 레이블을 지정하고, **master02** 노드에 **tier=silver** 레이블을 지정합니다.

- 1.1. OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **tier** 레이블을 이미 사용하고 있는 노드가 있는지 확인합니다.

```
[student@workstation ~]$ oc get nodes -L tier
NAME STATUS ROLES AGE VERSION TIER
master01 Ready master,worker 5d20h v1.18.3+012b3ec
master02 Ready master,worker 5d20h v1.18.3+012b3ec
master03 Ready master,worker 5d20h v1.18.3+012b3ec
```

- 1.3. 레이블 **tier=gold**를 사용하여 **master01** 노드에 레이블을 지정합니다.

```
[student@workstation ~]$ oc label node master01 tier=gold
node/master01 labeled
```

1.4. 레이블 `tier=silver`를 사용하여 `master02` 노드에 레이블을 지정합니다.

```
[student@workstation ~]$ oc label node master02 tier=silver
node/master02 labeled
```

1.5. 레이블이 올바르게 추가되었는지 확인합니다.

```
[student@workstation ~]$ oc get nodes -L tier
NAME STATUS ROLES AGE VERSION TIER
master01 Ready master,worker 5d20h v1.18.3+012b3ec gold
master02 Ready master,worker 5d20h v1.18.3+012b3ec silver
master03 Ready master,worker 5d20h v1.18.3+012b3ec
```

2. `developer` 사용자로 전환하고 `schedule-review`라는 새 프로젝트를 만듭니다.

2.1. OpenShift 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

2.2. `schedule-review` 프로젝트를 만듭니다.

```
[student@workstation ~]$ oc new-project schedule-review
Now using project "schedule-review" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

3. `quay.io/redhattraining/loadtest:v1.0`에 있는 컨테이너 이미지를 사용하여 `loadtest`라는 새 애플리케이션을 만듭니다. `loadtest` 애플리케이션은 `tier=silver`로 레이블이 지정된 노드에 배포해야 합니다. 각 컨테이너가 CPU의 `100m` 및 메모리의 `20Mi`를 요청하는지 확인합니다.

3.1. 앞으로 더 쉽게 조정할 수 있도록 하려면 실제로 애플리케이션을 만들지 않고 배포 리소스 파일을 만듭니다.

```
[student@workstation ~]$ oc create deployment loadtest --dry-run=client \
> --image quay.io/redhattraining/loadtest:v1.0 \
> -o yaml > ~/D0280/labs/schedule-review/loadtest.yaml
```

3.2. `~/D0280/labs/schedule-review/loadtest.yaml`을 편집하여 노드 선택기를 지정합니다. 아래에 강조 표시된 행을 추가하고 적절한 들여쓰기가 있는지 확인합니다.

```
[student@workstation ~]$ vim ~/D0280/labs/schedule-review/loadtest.yaml
...output omitted...
spec:
 nodeSelector:
```

```

tier: silver
containers:
- image: quay.io/redhattraining/loadtest:v1.0
 name: loadtest
 resources: {}
status: {}

```

- 3.3. `~/D0280/labs/schedule-review/loadtest.yaml`을 계속 편집합니다.  
`resources: {}` 행을 아래에 나열된 강조 표시된 행으로 바꿉니다. 파일을 저장하기 전에 적절한 들여쓰기가 있는지 확인합니다.

```

...output omitted...
spec:
 nodeSelector:
 tier: silver
 containers:
 - image: quay.io/redhattraining/loadtest:v1.0
 name: loadtest
 resources:
 requests:
 cpu: "100m"
 memory: 20Mi
 status: {}

```

- 3.4. `loadtest` 애플리케이션을 만듭니다.

```
[student@workstation ~]$ oc create --save-config \
> -f ~/D0280/labs/schedule-review/loadtest.yaml
deployment.apps/loadtest created
```

- 3.5. 애플리케이션 포드가 실행되고 있는지 확인합니다. `oc get pods` 명령을 여러 번 실행해야 할 수도 있습니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
loadtest-85f7669897-z4mq7 1/1 Running 0 53s
```

- 3.6. 애플리케이션 포드에서 리소스 요청을 지정하는지 확인합니다.

```
[student@workstation ~]$ oc describe pod/loadtest-85f7669897-z4mq7 \
> | grep -A2 Requests
Requests:
 cpu: 100m
 memory: 20Mi
```

4. 자동으로 생성된 기본 호스트 이름을 사용하여 `loadtest`라는 애플리케이션의 경로를 만듭니다. 애플리케이션을 만든 방식에 따라 경로를 만들기 전에 서비스를 만들어야 할 수도 있습니다. 실행 중인 `curl http://loadtest-schedule-review.apps.ocp4.example.com/api/loadtest/v1/healthz`에서 `{"health": "ok"}`를 반환하는 경우 애플리케이션이 예상대로 작동하는 것입니다.

- 4.1. `loadtest` 애플리케이션에 대한 배포를 노출하여 서비스를 만듭니다.

```
[student@workstation ~]$ oc expose deployment/loadtest \
> --port 80 --target-port 8080
service/loadtest exposed
```

4.2. **loadtest** 서비스를 노출하여 **loadtest**라는 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service/loadtest --name loadtest
route.route.openshift.io/loadtest exposed
```

4.3. **loadtest** 경로에서 생성한 호스트 이름을 식별합니다.

```
[student@workstation ~]$ oc get route/loadtest
NAME HOST/PORT
loadtest loadtest-schedule-review.apps.ocp4.example.com ...
```

4.4. 이전 단계에서 식별한 호스트 이름을 사용하여 **loadtest** 애플리케이션 액세스를 확인합니다.

```
[student@workstation ~]$ curl http://loadtest-schedule-review.\n> apps.ocp4.example.com/api/loadtest/v1/healthz
{"health": "ok"}
```

5. CPU 부하가 **70%**를 초과하는 경우 **2**개의 포드에서 최대 **40**개의 포드로 확장하는 **loadtest** 애플리케이션에 대해 **loadtest**라는 수평 포드 자동 확장기를 만듭니다. 다음 명령을 사용하여 수평 포드 자동 확장기를 테스트할 수 있습니다. `curl -X GET http://loadtest-schedule-review.apps.ocp4.example.com/api/loadtest/v1/cpu/3`



### 참고

수평 포드 자동 확장기에서는 이 **loadtest** 애플리케이션을 확장하지만, 최대 40개의 포드에 도달하기 전에 OpenShift 클러스터는 리소스를 모두 실행합니다.

5.1. **loadtest** 애플리케이션에 대한 수평 포드 자동 확장기를 만듭니다.

```
[student@workstation ~]$ oc autoscale deployment/loadtest --name loadtest \
> --min 2 --max 40 --cpu-percent 70
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

5.2. **loadtest** 수평 포드 자동 확장기에서 **TARGETS** 열에 기본 사용량을 보고할 때까지 기다립니다.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

<unknown>이 0%로 변경되면 Ctrl+C를 눌러 **watch** 명령을 종료합니다.

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	...
loadtest	Deployment/loadtest	0%/70%	2	40	2	...

**참고**

TARGETS 열의 <unknown> 항목이 0%로 변경될 때까지 최대 5분이 걸릴 수 있습니다. <unknown> 항목이 변경되지 않으면 **oc describe** 명령을 사용하여 **loadtest** 애플리케이션 컨테이너에서 CPU 리소스를 요청하는지 확인합니다.

- 5.3. CPU 부하를 적용하여 수평 포드 자동 확장기를 테스트합니다. 이전에 식별된 호스트 이름을 **loadtest** 경로에 사용합니다. **curl** 명령이 완료될 때까지 기다립니다.

```
[student@workstation ~]$ curl -X GET http://loadtest-schedule-review.\> apps.ocp4.example.com/api/loadtest/v1/cpu/3
```

- 5.4. 추가 포드가 추가되었는지 확인합니다. **oc get hpa/loadtest** 명령을 여러 번 실행해야 할 수도 있습니다. 출력은 다를 수 있지만 복제본 수가 2보다 큰지 확인합니다.

[student@workstation ~]\$ oc get hpa/loadtest						
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	...
loadtest	Deployment/loadtest	1043%/70%	2	40	21	...

6. **admin** 사용자로 **schedule-review** 프로젝트에서 **review-quota**라는 할당량을 구현합니다. **schedule-review** 프로젝트를 최대 1개의 전체 CPU, 2G의 메모리 및 20개의 포드로 제한합니다.

- 6.1. OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 6.2. 리소스 할당량을 만듭니다.

```
[student@workstation ~]$ oc create quota review-quota \
> --hard cpu="1",memory="2G",pods="20"
resourcequota/review-quota created
```

**참고**

할당량은 기존 포드에 영향을 미치지 않지만 포드와 같은 새 리소스가 요청되면 스케줄러는 할당량을 평가합니다.

**평가**

다음 **lab** 명령을 실행하여 작업을 확인합니다. **lab** 명령이 오류를 보고하면 변경 사항을 검토하고 수정 사항을 만들고 성공할 때까지 **lab** 명령을 실행합니다.

```
[student@workstation ~]$ lab schedule-review grade
```

**완료**

**workstation** 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab schedule-review finish
```

이로써 랩이 완료됩니다.

## 요약

---

이 장에서 학습한 내용:

- 기본 포드 스케줄러는 지역 및 영역을 사용하여 성능과 중복성을 모두 달성합니다.
- 노드 레이블을 지정하고 노드 선택기를 사용하여 포드 배치에 영향을 줍니다.
- 리소스 요청은 예약하는 데 포드에 필요한 최소한의 리소스 양을 정의합니다.
- 할당량은 프로젝트에 사용할 수 있는 리소스의 양을 제한합니다.
- 사용자 지정 프로젝트 템플릿을 통해 새 프로젝트에 대한 할당량 및 제한 범위를 자동으로 만들 수 있습니다.
- `oc scale` 명령은 포드 복제본 수를 수동으로 확장합니다.
- 수평 포드 자동 확장기는 로드를 기반으로 포드 복제본을 동적으로 확장합니다.



## 클러스터 업데이트 설명

### 목적

클러스터 업데이트를 수행하는 방법에 대해 설명합니다.

### 목표

클러스터 업데이트 프로세스에 대해 설명합니다.

### 섹션

- 클러스터 업데이트 프로세스 설명 및 퀴즈

# 클러스터 업데이트 프로세스 설명

## 목표

이 섹션을 마치면 클러스터 업데이트 프로세스를 설명할 수 있습니다.

## 클러스터 업데이트 소개

Red Hat Openshift Container Platform 4는 Red Hat Enterprise Linux CoreOS를 사용하여 많은 새 기능을 추가합니다. Red Hat은 클러스터 및 기본 운영 체제를 업데이트할 수 있는 최적의 업그레이드 경로를 제공하는 새 소프트웨어 배포 시스템을 출시했습니다. 이 새로운 배포 시스템은 OpenShift 4 아키텍처 변경의 중요한 이점 중 하나이며 클러스터를 통해 OTA(Over-the-Air)를 업그레이드할 수 있도록 합니다.

이 OTA용 소프트웨어 배포 시스템은 컨트롤러 매니페스트, 클러스터 역할 및 클러스터를 특정 버전으로 업데이트하는 데 필요한 기타 리소스를 관리합니다.

이 기능을 사용하면 클러스터에서 사용 가능한 최신 버전을 원활하게 실행할 수 있습니다. 또한 OTA를 사용하면 최신 버그 픽스 및 보안 패치를 포함하여 클러스터에서 새 기능을 사용할 수 있게 됩니다. OTA는 업그레이드로 인한 가동 중지 시간을 크게 줄입니다.

Red Hat은 <https://cloud.redhat.com>에서 이 서비스를 호스팅 및 관리하며 <https://quay.io>에서 클러스터 이미지를 호스팅합니다.



### 중요

OpenShift 4.5의 경우 OTA 시스템을 사용하려면 인터넷에 지속적으로 연결되어 있어야 합니다. 이 기능은 온 프레미스로 배포할 수 없습니다.

연결이 끊긴 클러스터를 업데이트하는 방법에 대한 자세한 내용은 참조 섹션에 나열된 업데이트 가이드 및 설치 구성 장을 참조하십시오.

OTA를 사용하면 중간 버전을 건너뛸 수 있기 때문에 더 빠른 업데이트를 수행할 수 있습니다. 예를 들어 4.5.1에서 4.5.3으로 업데이트할 수 있으므로 4.5.2를 무시할 수 있습니다.

단일 인터페이스(<https://cloud.redhat.com>)를 사용하여 모든 OpenShift 클러스터의 라이프사이클을 관리합니다.

The screenshot shows the Red Hat OpenShift Cluster Management interface. On the left, there's a sidebar with navigation links: Clusters, Subscriptions, Overview, Documentation, Support Cases, Cluster Manager Feedback, and Red Hat Marketplace. The main area is titled 'Clusters' and contains a table with three rows of cluster information. The columns are: Name, Status, Type, Created, Version, and Provider (Lo...). The first cluster is named 'rhos-ocp4-60-day-trial', status 'Ready', type 'OCP', created '60-day trial', version '4.2.2', and provider 'AWS (US East, Ohio)'. The second cluster is named 'rhos-ocp4-60-day-trial-2', status 'Ready', type 'OCP', created '60-day trial', version '4.5.3', and provider 'OPENSTACK (N/A)'. The third cluster is named 'rhos-ocp4-60-day-trial-3', status 'Ready', type 'OCP', created '60-day trial', version '4.5.3', and provider 'OPENSTACK (N/A)'. There are also 'Update' and '⋮' buttons for each row.

그림 14.1: cloud.redhat.com에서 클러스터 관리

서비스는 특정 업데이트의 클러스터 자격에 해당하는 업그레이드 경로를 정의합니다.

업데이트 경로는 채널 업그레이드에 속합니다. 채널은 업그레이드 경로의 표현으로 시각화할 수 있습니다. 채널은 업데이트의 주기와 안정성을 제어합니다. OTA 정책 엔진은 채널을 업그레이드 경로 내 특정 버전에 대한 일련의 포인터로 나타냅니다.

채널 이름은 계층(release candidate, fast 및 stable), 주 버전(4) 및 부 버전(. 2)의 세 부분으로 구성됩니다. 채널 이름의 예로는 **stable-4.5**, **fast-4.5**, **candidate-4.5**가 있습니다. 각 채널에서는 지정된 클러스터 버전에 대한 패치를 제공합니다.

## 후보 채널 설명

candidate 채널에서는 다음 버전 OpenShift Container Platform의 기능 수용을 테스트하는 데 필요한 업데이트를 제공합니다. 릴리스 후보 버전은 추가 검사 대상이며 품질 표준을 만족하는 경우 빠른 또는 안정적인 채널로 승격됩니다.



### 참고

candidate 채널에 나열되는 업데이트는 Red Hat에서 지원하지 않습니다.

## fast 채널 설명

fast 채널은 업데이트가 준비되는 즉시 업데이트를 제공합니다. 이 채널은 사전 프로덕션 및 QA 환경에 가장 적합합니다. **fast-4.5** 채널을 사용하여 이전 부 버전의 OpenShift Container Platform을 업그레이드할 수 있습니다.



### 참고

고객은 Red Hat에 연결된 고객 프로그램을 통해 OpenShift를 개선하는 데 도움을 줄 수 있습니다. 이 프로그램에 참여하면 클러스터가 fast 채널에 등록됩니다.

## stable 채널 설명

안정적인 채널에는 지연된 업데이트가 포함되는 데 이는 지정된 클러스터 버전에 대한 마이너 업데이트만 제공함을 의미하며, 이러한 채널은 프로덕션 환경에 더 적합합니다.

## 14장 | 클러스터 업데이트 설명

Red Hat 지원 및 SRE(사이트 안정성 엔지니어링) 팀은 새로운 빠른 업데이트로 운영 클러스터를 모니터링합니다. 운영 클러스터가 추가 테스트 및 검증을 통과하는 경우 빠른 채널의 업데이트가 stable 채널에서 활성화됩니다.

Red Hat이 fast 채널 업데이트의 운영 문제를 관찰하는 경우 stable 채널에서 해당 업데이트를 건너뜁니다. stable 채널 지연에서는 테스트에 노출되지 않았던 실제 OpenShift 클러스터에서 예기치 않은 문제를 관찰할 수 있는 시간을 제공합니다.

## 업그레이드 경로 설명

다음은 이러한 업그레이드 경로가 Red Hat OpenShift Container Platform 버전 4.5에 적용되는 방법을 설명합니다.

- **stable-4.5** 채널을 사용하는 경우 클러스터를 4.5.0에서 4.5.1 또는 4.5.2로 업그레이드할 수 있습니다. 4.5.3 릴리스에서 문제가 발견되면 해당 버전으로 업그레이드할 수 없습니다. 4.5.4 릴리스에 대한 패치가 제공되면 클러스터를 해당 버전으로 업데이트할 수 있습니다.

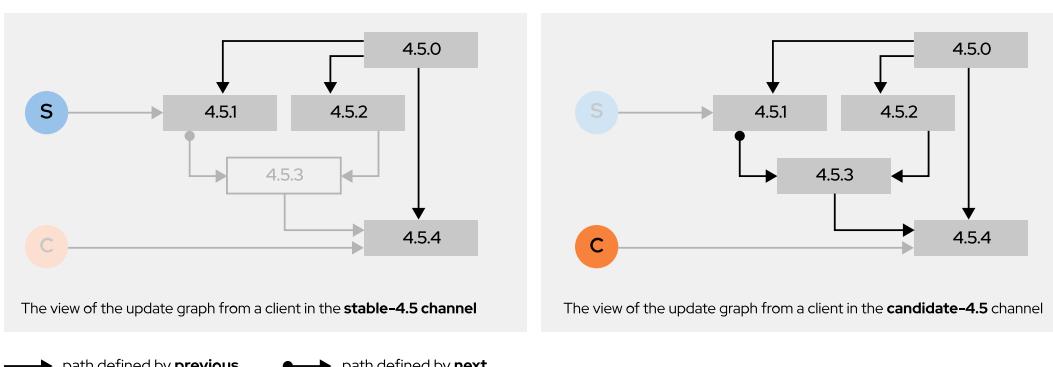
이 채널은 해당 채널의 릴리스를 Red Hat SRE 및 지원 서비스를 통해 테스트하므로 프로덕션 환경에 적합합니다.

- **fast-4.5** 채널은 4.5.1 및 4.5.2 업데이트를 제공할 수 있지만 4.6.1에서는 제공하지 않습니다. 이 채널은 Red Hat에서도 지원하며 프로덕션 환경에 적용할 수 있습니다.

새 부 버전의 새 릴리스로 업그레이드하려면 관리자가 다른 부 버전 채널(예: **fast-4.6**)을 구체적으로 선택해야 합니다.

- **candidate-4.5** 채널을 사용하면 OpenShift의 최신 기능을 설치할 수 있습니다. 이 채널을 사용하면 4.5.1, 4.5.2, 4.5.3 등과 같은 모든 z-stream 릴리스로 업그레이드할 수 있습니다.

제품이 릴리스될 때 이 채널을 사용하여 제품의 최신 기능에 액세스할 수 있습니다. 이 채널은 개발 및 사전 프로덕션 환경에 적합합니다.



**그림 14.2: 안정적인 후보 채널을 위한 그래프 업데이트**

stable 및 fast 채널은 GA(General Availability)로 분류되는 반면 candidate 채널(릴리스 후보 채널)은 Red Hat에서 지원하지 않습니다.

클러스터의 안정성과 적절한 지원 수준을 보장하기 위해 stable 채널은 fast 채널로만, fast 채널은 stable 채널로만 전환해야 합니다. stable 채널이나 fast 채널을 candidate 채널로 전환할 수 있지만 바람직한 방법은 아닙니다. candidate 채널은 다음 버전의 OpenShift Container Platform을 검증할 때 기능 수용을 테스트하고 지원하는 데 가장 적합합니다.



## 참고

패치 및 CVE 팩스에 대한 업데이트 릴리스는 몇 시간에서 하루까지 다양합니다. 이 지연은 OpenShift 클러스터에 대한 모든 작업의 영향을 평가하는 데 필요한 시간을 제공합니다.

## 업데이트 채널 변경

웹 콘솔 또는 OpenShift CLI 클라이언트를 사용하여 업데이트 채널을 **stable-4.5**, **fast-4.5** 또는 **candidate-4.5**로 변경할 수 있습니다.

- 웹 콘솔에서 Details(세부 정보) 탭의 Administration(관리) → Cluster Settings(클러스터 설정) 페이지로 이동한 다음 연필 아이콘을 클릭합니다.

그림 14.3: 웹 콘솔의 현재 업데이트 채널

창에 업데이트 채널을 선택할 수 있는 옵션이 표시됩니다.

그림 14.4: 웹 콘솔의 업데이트 채널 변경

- `oc` 클라이언트를 사용하는 다른 업데이트 채널로 전환하려면 다음 명령을 실행합니다. 다른 업데이트 채널(예: **stable-4.6**)로 전환하여 OpenShift Container Platform의 다음 부 버전으로 업데이트할 수도 있습니다.

```
[user@demo ~]$ oc patch clusterversion version --type="merge" --patch \
> '{"spec":{"channel":"fast-4.5"}}'
clusterversion.config.openshift.io/version patched
```

## OTA 설명

OTA는 클라이언트-서버 접근 방식을 따릅니다. Red Hat은 클러스터 이미지와 업데이트 인프라를 호스팅합니다. OTA의 한 가지 기능은 클러스터에 사용할 수 있는 모든 업데이트 경로를 생성하는 것입니다. OTA는 클러스터에 대한 정보와 사용 가능한 업그레이드 경로를 결정하는 자격 부여를 수집합니다. 새 업데이트를 사용할 수 있는 경우 웹 콘솔에서 알림을 전송합니다.

다음 다이어그램은 업데이트 아키텍처를 설명합니다. Red Hat은 클러스터 이미지와 "감시자"를 모두 호스트하여 자동으로 새 이미지를 감지합니다. CVO(클러스터 버전 운영자)는 해당 감시자로부터 업데이트 상태를 받습니다. CVO는 해당 운영자를 통해 클러스터 구성 요소를 업데이트한 다음 OLM(운영자 라이프사이클 관리자)에서 관리하는 추가 구성 요소를 업데이트합니다.

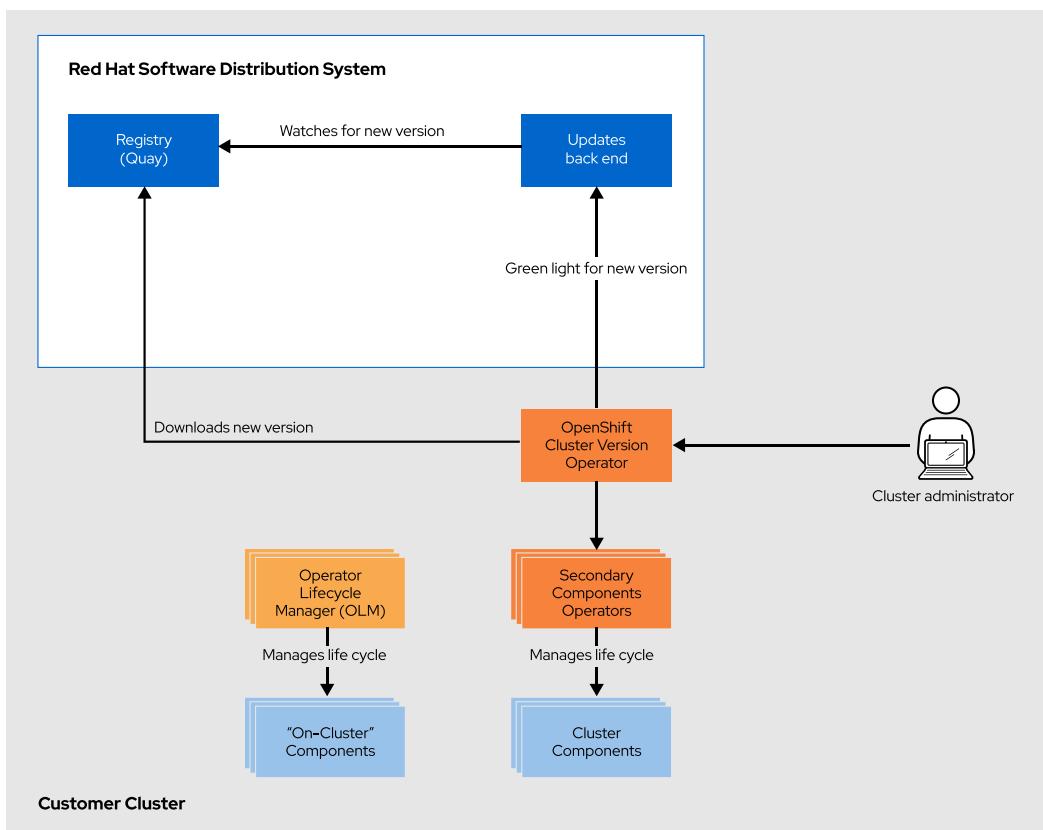


그림 14.5: OpenShift Container Platform 업데이트 아키텍처

원격 분석을 통해 Red Hat이 업데이트 경로를 확인할 수 있습니다. 클러스터는 프로메테우스 기반 원격 분석을 사용하여 각 클러스터 운영자의 상태를 보고합니다. 데이터가 익명으로 처리되고 잠재적인 새 릴리스에 대해 클러스터 관리자에게 알리는 Red Hat 서버로 다시 전송됩니다.

**참고**

Red Hat은 고객의 프라이버시를 소중하게 생각합니다. 원격 분석에서 수집하는 전체 데이터 목록을 보려면 참조 섹션에 나열된 샘플 지표 문서를 참조하십시오.

향후 Red Hat은 ISV(독립 소프트웨어 벤더) 운영자를 포함하도록 업그레이드 경로에 포함된 업데이트된 운영자 목록을 확장하려고 합니다.

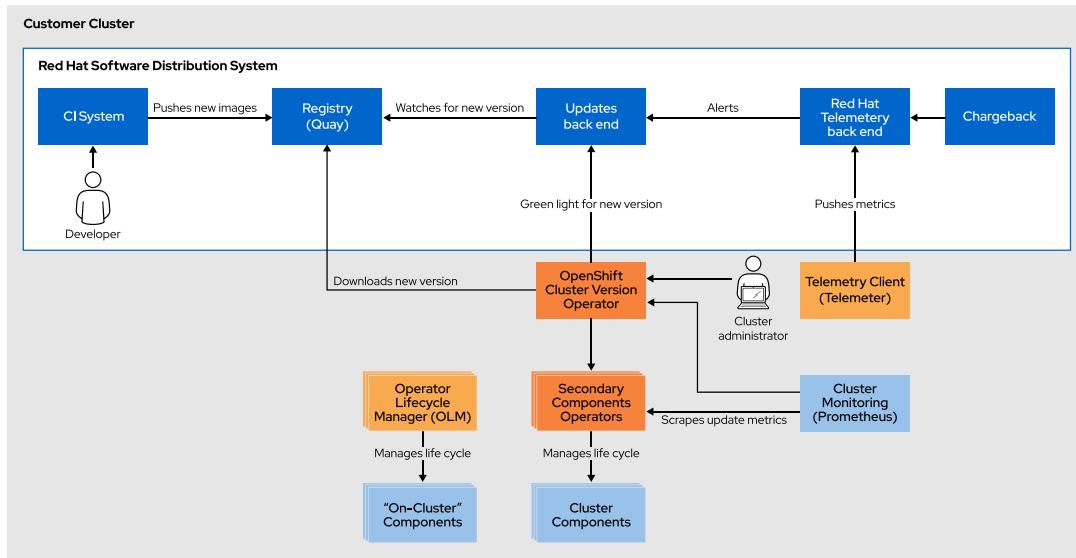


그림 14.6: 원격 분석을 사용하여 클러스터 업데이트 관리

## 업데이트 프로세스 설명

### 시스템 구성 운영자

시스템 구성 운영자는 원하는 시스템 상태를 각 노드에 적용합니다. 이 구성 요소는 또한 클러스터의 노드 롤링 업그레이드를 처리하고 구성 형식으로 CoreOS Ignition을 사용합니다.

### OLM(운영자 라이프사이클 관리자)

OLM(운영자 라이프사이클 관리자)은 클러스터에서 실행 중인 모든 운영자의 업데이트를 조정합니다.

## 클러스터 업데이트

웹 콘솔 또는 명령줄을 통해 클러스터를 업데이트할 수 있습니다. 명령줄을 사용하는 것보다 웹 콘솔을 통해 업데이트하는 것이 더 간편합니다. 새 업데이트를 사용할 수 있는 경우 Administration(관리) → Cluster Settings(클러스터 설정) 페이지에서 Update Status가 Update available로 표시됩니다. 이 페이지에서 Update now(지금 업데이트)를 클릭하여 프로세스를 시작합니다.

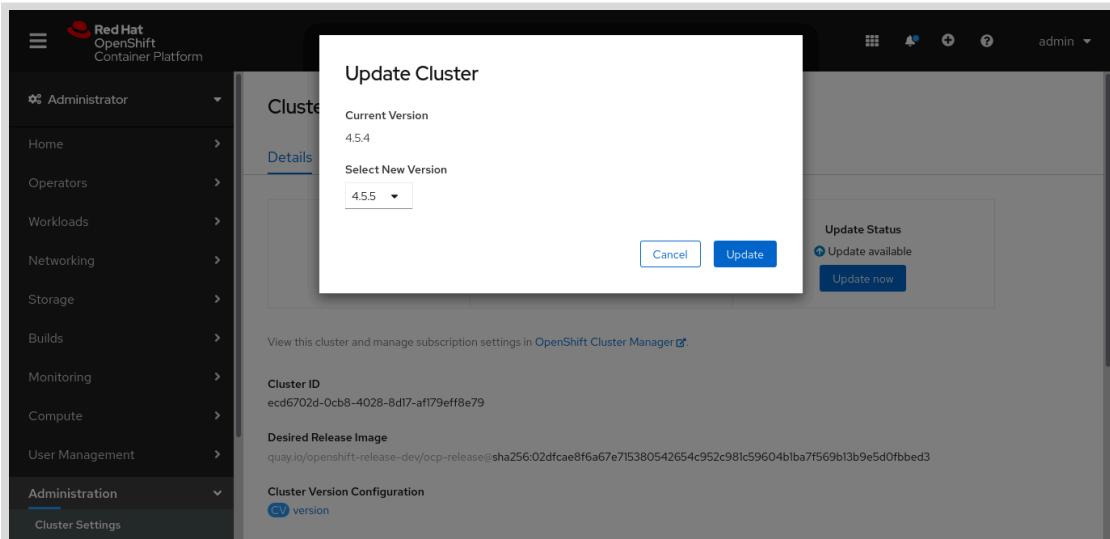


그림 14.7: 웹 콘솔을 사용하여 클러스터 업데이트

**중요**

Red Hat은 클러스터를 이전 버전 또는 룰백으로 되돌리는 작업을 지원하지 않습니다.  
Red Hat은 최신 버전으로의 업그레이드만 지원합니다.

업데이트 프로세스는 또한 사용 가능한 업데이트가 있는 경우 기본 운영 체제를 업데이트합니다. 이는 **rpm-ostree** 기술을 사용하여 트랜잭션 업그레이드를 관리합니다. 업데이트는 컨테이너 이미지를 통해 제공되며 OpenShift 업데이트 프로세스의 일부입니다. 업데이트가 배포되면 노드는 새 이미지를 가져와서 압축을 풀고 디스크에 패키지를 쓴 다음 부트로더를 수정하여 새 버전으로 부팅합니다. 시스템이 재부팅되고 롤링 업데이트를 구현하여 클러스터 용량이 최소한으로 영향을 받는 것을 확인합니다.

다음 단계에서는 명령줄 인터페이스를 사용하여 클러스터 관리자로 클러스터를 업데이트하는 절차를 설명합니다.

**중요**

OpenShift 클러스터를 업데이트하기 전에 OLM(운영자 라이프사이클 관리자)을 통해 설치된 모든 운영자를 최신 버전으로 업데이트해야 합니다.

1. 클러스터 버전을 검색하여 현재 업데이트 채널 정보를 검토하고 채널을 확인합니다. 프로덕션 환경에서 클러스터를 실행하는 경우 채널이 **stable**인지 확인합니다.

```
[user@demo ~]$ oc get clusterversion
NAME VERSION AVAILABLE PROGRESSING SINCE STATUS
version 4.5.4 True False 5d Cluster version is 4.5.4

[user@demo ~]$ oc get clusterversion -o json | jq ".items[0].spec.channel"
"stable-4.5"
```

2. 사용 가능한 업데이트를 보고 적용하려는 업데이트의 버전 번호를 확인합니다.

```
[user@demo ~]$ oc adm upgrade
Cluster version is 4.5.4

Updates:

VERSION IMAGE
4.5.5 quay.io/openshift-release-dev/ocp-release@sha256:...
...output omitted...
```

3. 최신 업데이트를 클러스터에 적용하거나 특정 버전으로 업데이트합니다.
    - 다음 명령을 실행하여 클러스터에 사용할 수 있는 최신 업데이트를 설치합니다.
- ```
[user@demo ~]$ oc adm upgrade --to-latest=true
```
- 다음 명령을 실행하여 특정 버전을 설치합니다. 버전은 `oc adm upgrade` 명령이 반환하는 사용 가능한 버전 중 하나에 해당합니다.
4. 이전 명령은 업데이트 프로세스를 초기화합니다. 다음 명령을 실행하여 CVO(클러스터 버전 운영자) 및 설치된 클러스터 운영자의 상태를 검토합니다.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION      AVAILABLE      PROGRESSING      SINCE      STATUS
version   4.5.4        True          True            32m        Working towards 4.5.5 ...

[user@demo ~]$ oc get clusteroperators
NAME                  VERSION      AVAILABLE      PROGRESSING      DEGRADED
authentication       4.5.4        True          False           False
cloud-credential     4.5.5        False         True            False
openshift-apiserver 4.5.5        True          False           True
...output omitted...
```

5. 다음 명령을 사용하여 업데이트 상태를 모니터링하도록 클러스터 버전 상태 기록을 검토할 수 있습니다. 모든 오브젝트가 업데이트를 완료하는 데 시간이 걸릴 수 있습니다.
- 기록에는 클러스터에 적용된 최신 버전 목록이 포함되어 있습니다. 이 값은 CVO에서 업데이트를 적용할 때 업데이트됩니다. 목록은 날짜별로 정렬되며 최신 업데이트가 목록의 첫 번째에 표시됩니다.
- 룰아웃이 완료되면 기록의 업데이트 상태가 **Completed**가 됩니다. 또는 업데이트가 실패했거나 완료되지 않은 경우 업데이트에 **Partial**(부분) 상태가 포함됩니다.

```
[user@demo ~]$ oc describe clusterversion
...output omitted...

History:
Completion Time: 2020-09-28T16:02:18Z
Image: quay.io/openshift-release-dev/ocp-release@sha256:...
Started Time: 2020-09-28T15:31:13Z
State: Completed
Verified: true
Version: 4.5.5
```

| | |
|----------------------|--|
| Completion Time: | 2020-08-05T18:35:08Z |
| Image: | quay.io/openshift-release-dev/ocp-release@sha256:... |
| Started Time: | 2020-08-05T18:22:42Z |
| State: | Completed |
| Verified: | true |
| Version: | 4.5.4 |
| Observed Generation: | 5 |
| Version Hash: | AF5-oeav9wI= |
| Events: | none |



중요

업그레이드에 실패하면 운영자가 이를 중지하고 실패한 구성 요소의 상태를 보고합니다. 클러스터를 이전 버전으로 다시 롤링하는 것은 지원되지 않습니다. 업그레이드에 실패하면 Red Hat 지원 팀에 문의하십시오.

6. 업데이트가 완료되면 클러스터 버전이 새 버전으로 업데이트되었는지 확인할 수 있습니다.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION      AVAILABLE      PROGRESSING      SINCE      STATUS
version   4.5.5        True          False           11m       Cluster version is 4.5.5
```



참조

연결되지 않은 환경에 Red Hat OpenShift Container Platform을 설치하는 데 대한 자세한 내용은

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/installing/index#installation-configuration

에 있는 Red Hat OpenShift Container Platform 4.5 설치 설명서의 설치 구성 장을 참조하십시오.

업데이트 채널, 업데이트 사전 요구 사항, 연결이 끊긴 환경에서 클러스터를 업데이트하는 방법에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 클러스터 업데이트 설명서의 제한된 네트워크 클러스터 업데이트 및 부 버전 간 클러스터 업데이트 장을 참조하십시오.

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/updating_clusters/index#updating-restricted-network-cluster

OLM(운영자 라이프사이클 관리자)을 통해 설치된 운영자 업데이트에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 운영자 사용 설명서의 관리자 태스크 장에서 설치된 운영자 업그레이드 섹션을 참조하십시오.

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/operators/index#olm-upgrading-operators

OpenShift Container Platform 업그레이드 경로에 대한 자세한 내용은 고객 포털의 다음 페이지를 참조하십시오.

<https://access.redhat.com/solutions/4583231/>

샘플 지표

<https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/sample-metrics.md>

Cincinnati

<https://github.com/openshift/cincinnati/blob/master/docs/design/cincinnati.md#cincinnati>

▶ 퀴즈

클러스터 업데이트 프로세스 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 fast-4.5 업데이트 채널에서 사용할 수 있는 두 가지 업데이트는 무엇입니까? (두 개를 선택하십시오.)

- a. 4.4.2
- b. 4.5.1
- c. 4.6.1
- d. 4.5.5

▶ 2. 다음 중 Quay.io에서 업데이트된 클러스터 이미지를 검색하는 구성 요소는 무엇입니까?

- a. 클러스터 모니터링(프로메테우스)
- b. OLM(운영자 라이프사이클 관리자)
- c. CVO(클러스터 버전 운영자)
- d. 원격 분석 클라이언트(Telemeter)

▶ 3. 다음 구성 요소 중 클러스터 운영자가 아닌 운영자의 업데이트를 관리하는 구성 요소는 무엇입니까?

- a. OLM(운영자 라이프사이클 관리자)
- b. 원격 분석 클라이언트(Telemeter)
- c. CVO(클러스터 버전 운영자)

▶ 4. 다음 중 현재 실행 중인 클러스터의 버전을 검색하는 데 사용할 수 있는 두 가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- a. oc adm upgrade
- b. oc get clusterchannel
- c. oc get clusterversion

▶ 5. 다음 중 OTA 기능과 관련하여 올바른 설명은 무엇입니까?

- a. stable 채널은 GA(General Availability)로 분류되는 반면 fast 채널은 RC(Release Candidate)로 분류됩니다.
- b. stable 채널을 사용하는 경우에는 중간 버전을 건너뛸 수 없습니다. 예를 들어 4.3.27에서 4.3.29로 업데이트하는 경우 OpenShift에서 4.3.28 버전을 먼저 설치해야 합니다.
- c. stable 채널이나 fast 채널에서 candidate 채널로 전환하는 것은 바람직하지 않습니다. 그러나 fast 채널에서 stable 채널로 전환하거나 그 반대로 전환할 수 있습니다.
- d. 실패한 업데이트를 롤백하는 것은 z-stream에서 다른 버전으로의 업데이트를 시도하는 경우(예: 4.5.2에서 4.5.3으로의 업데이트(4.5.3에서 4.6으로의 업데이트는 해당하지 않음))에만 Red Hat에서 지원합니다.

▶ 6. 다음 중 일반 가용성으로 분류되는 채널은 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. candidate-4.5.stable
- b. stable-4.5
- c. candidate-stable-4.5
- d. fast-4.5
- e. fast-4.5.1

▶ 솔루션

클러스터 업데이트 프로세스 설명

다음 질문에 대한 올바른 답을 선택하십시오.

▶ 1. 다음 중 fast-4.5 업데이트 채널에서 사용할 수 있는 두 가지 업데이트는 무엇입니까? (두 개를 선택하십시오.)

- a. 4.4.2
- b. 4.5.1
- c. 4.6.1
- d. 4.5.5

▶ 2. 다음 중 Quay.io에서 업데이트된 클러스터 이미지를 검색하는 구성 요소는 무엇입니까?

- a. 클러스터 모니터링(프로메테우스)
- b. OLM(운영자 라이프사이클 관리자)
- c. CVO(클러스터 버전 운영자)
- d. 원격 분석 클라이언트(Telemeter)

▶ 3. 다음 구성 요소 중 클러스터 운영자가 아닌 운영자의 업데이트를 관리하는 구성 요소는 무엇입니까?

- a. OLM(운영자 라이프사이클 관리자)
- b. 원격 분석 클라이언트(Telemeter)
- c. CVO(클러스터 버전 운영자)

▶ 4. 다음 중 현재 실행 중인 클러스터의 버전을 검색하는 데 사용할 수 있는 두 가지 명령은 무엇입니까? (두 개를 선택하십시오.)

- a. oc adm upgrade
- b. oc get clusterchannel
- c. oc get clusterversion

▶ 5. 다음 중 OTA 기능과 관련하여 올바른 설명은 무엇입니까?

- a. stable 채널은 GA(General Availability)로 분류되는 반면 fast 채널은 RC(Release Candidate)로 분류됩니다.
- b. stable 채널을 사용하는 경우에는 중간 버전을 건너뛸 수 없습니다. 예를 들어 4.3.27에서 4.3.29로 업데이트하는 경우 OpenShift에서 4.3.28 버전을 먼저 설치해야 합니다.
- c. stable 채널이나 fast 채널에서 candidate 채널로 전환하는 것은 바람직하지 않습니다. 그러나 fast 채널에서 stable 채널로 전환하거나 그 반대로 전환할 수 있습니다.
- d. 실패한 업데이트를 롤백하는 것은 z-stream에서 다른 버전으로의 업데이트를 시도하는 경우(예: 4.5.2에서 4.5.3으로의 업데이트(4.5.3에서 4.6으로의 업데이트는 해당하지 않음))에만 Red Hat에서 지원합니다.

▶ 6. 다음 중 일반 가용성으로 분류되는 채널은 두 가지는 무엇입니까? (두 개를 선택하십시오.)

- a. candidate-4.5.stable
- b. stable-4.5
- c. candidate-stable-4.5
- d. fast-4.5
- e. fast-4.5.1

요약

이 장에서 학습한 내용:

- OpenShift 4 아키텍처 변경의 주요 이점 중 하나는 OTA(Over-the-Air)를 통해 클러스터를 업데이트할 수 있다는 것입니다.
- Red Hat은 클러스터 및 기본 운영 체제를 업데이트할 수 있는 최적의 경로를 보장하는 새 소프트웨어 배포 시스템을 제공합니다.
- 배포 채널에는 다음 세 가지가 있습니다.
 - stable 채널은 지연된 업데이트를 제공합니다.
 - fast 채널은 사용할 수 있는 즉시 업데이트를 제공합니다.
 - candidate 채널에서는 다음 버전 OpenShift Container Platform의 기능 수용을 테스트하는데 필요한 업데이트를 제공합니다.
- Red Hat은 클러스터를 이전 버전으로 되돌리는 작업을 지원하지 않습니다. Red Hat은 최신 버전으로의 업그레이드만 지원합니다.

15장

웹 콘솔로 클러스터 관리

목적

웹 콘솔을 사용하여 Red Hat OpenShift 클러스터를 관리합니다.

목표

- 웹 콘솔을 사용하여 클러스터 관리를 수행합니다.
- 웹 콘솔을 사용하여 애플리케이션 및 Kubernetes 운영자를 관리합니다.
- 클러스터 노드와 애플리케이션의 성능 및 상태 지표를 검사합니다.

섹션

- 클러스터 관리 수행(안내에 따른 연습)
- 워크로드 및 운영자 관리(안내에 따른 연습)
- 클러스터 지표 검사(안내에 따른 연습)

랩

웹 콘솔로 클러스터 관리

클러스터 관리 수행

목표

이 섹션을 완료하면 웹 콘솔을 사용하여 클러스터 관리를 수행할 수 있습니다.

웹 콘솔 설명

Red Hat OpenShift 웹 콘솔은 그래픽 사용자 인터페이스를 제공하여 관리 및 문제 해결 작업을 수행할 수 있습니다. 이는 **관리자** 및 **개발자** 관점을 모두 지원합니다. 이 과정에서는 **관리자** 관점을 살펴봅니다.

다음 목록에서는 주요 탐색 메뉴 항목으로 그룹화된 웹 콘솔의 가장 중요한 부분을 간략하게 설명합니다. 개별 항목을 볼 수 있는 기능은 사용자와 연결된 역할 및 역할 바인딩에 따라 다릅니다. **cluster-admin** 클러스터 역할의 사용자는 모든 항목을 보고 수정할 수 있습니다. **view** 클러스터 역할의 사용자는 대부분의 항목을 볼 수 있지만 변경할 수는 없습니다. 추가 역할로 특정 항목에 대한 액세스 권한을 제공할 수 있습니다.

홈

Home(홈) → Overview(개요) 페이지에서는 상태 지표, 리소스 개수 및 시스템 업데이트 또는 포드 오류 등의 이벤트 스트리밍 목록을 포함하여 클러스터에 대한 간략한 개요를 제공합니다.

Home(홈) → Search(검색) 페이지로 이동하여 모든 유형의 리소스를 찾거나 만들 수 있습니다. 이는 메뉴에 전용 탐색 기능이 없는 리소스로 이동할 수 있는 유용한 시작점이기도 합니다.

Home(홈) → Events(이벤트) 페이지에는 클러스터에서 발생하는 필터링 가능한 이벤트 스트림이 표시되며 문제 해결을 위한 출발점으로도 유용합니다.

운영자

OperatorHub를 사용하여 Red Hat에서 선별한 운영자를 탐색하고 설치한 다음 **Installed Operators**(설치된 운영자) 페이지로 이동하여 운영자를 관리합니다.

워크로드, 네트워킹 및 스토리지

배포, 서비스 및 영구 볼륨과 같은 일반적인 리소스를 관리합니다. 문제 해결에 대한 구체적인 관심은 포드 로그를 보고 터미널에 연결하는 기능입니다.

빌드

빌드 구성, 빌드 및 이미지 스트림을 관리합니다.

모니터링

경고를 보고 ad hoc 프로메테우스 쿼리를 수행합니다.

컴퓨팅

노드, 시스템 및 시스템 자동 확장기 등의 컴퓨팅 리소스를 보고 관리합니다.

사용자 관리

사용자, 그룹, 서비스 계정, 역할 및 역할 바인딩을 보고 관리합니다.

관리

클러스터 관리자가 특히 관심을 갖는 다양한 설정을 보고 관리합니다(예: 클러스터 업데이트, 클러스터 운영자, CRD, 리소스 할당량).

OpenShift 웹 콘솔 액세스

OpenShift 웹 콘솔은 `openshift-console` 프로젝트에서 포드로 실행되며 `openshift-console-operator` 프로젝트에서 실행되는 운영자에 의해 관리됩니다. 경로를 나열하여 URL을 검색할 수 있습니다.

```
[student@workstation ~]$ oc get routes -n openshift-console
NAME      HOST/PORT          ... PORT ...
console   console-openshift-console.apps.cluster.example.com ... https ...
downloads downloads-openshift-console.apps.cluster.example.com ... http ...
```

프로덕션이 아닌 시스템에서 자체 서명된 인증서는 일반적으로 HTTPS 엔드포인트에 사용됩니다. 웹 브라우저는 인증서에 대해 경고하며 처음으로 웹 콘솔로 이동할 때 보안 예외를 추가해야 합니다.

리소스 찾기

웹 UI는 리소스를 찾을 수 있는 여러 가지 방법을 제공합니다. 배포 및 서비스와 같은 많은 일반 리소스는 왼쪽에 있는 주 메뉴에서 사용할 수 있습니다. Home(홈) → Search(검색) 페이지를 사용하여 다른 리소스 유형을 찾을 수 있습니다. 이 페이지에서는 리소스 유형 및 레이블 선택기 필드에 대한 전체 메뉴를 제공합니다.

이름 필터를 사용하여 Projects(프로젝트) 페이지와 같은 긴 목록으로 페이지에서 리소스를 빠르게 찾습니다.

| Name | Display Name | Status | Requester | Memory | CPU |
|---------------------------------|-----------------|--------|--------------|-----------|-------------|
| PR openshift-apiserver | No display name | Active | No requester | 361.0 MiB | 0.033 cores |
| PR openshift-apiserver-operator | No display name | Active | No requester | 68.5 MiB | 0.015 cores |

포드를 상태별로 필터링하여 잠재적인 문제나 문제가 있는 배포를 식별하는 것이 유용할 수 있습니다.

| Status | Namespace | Status | Ready | Owner |
|---------|------------------------|---------|-------|------------------------|
| Running | NS openshift-apiserver | Running | 1/1 | RS apiserver-84db66db8 |
| Running | NS openshift-apiserver | Running | 1/1 | RS apiserver-84db66db8 |
| Running | NS openshift-apiserver | Running | 1/1 | RS apiserver-84db66db8 |

리소스의 세부 정보 페이지에는 일반적인 유용한 정보가 표시됩니다. 이 페이지의 내용은 유형에 따라 다릅니다. 예를 들어 **Pod Details(포드 세부 정보)** 페이지는 지표 및 상태 정보를 표시하고 **Secret Details(시크릿 세부 정보)** 페이지에서는 시크릿에 저장된 데이터를 노출하거나 복사할 수 있습니다.

세부 정보 페이지는 웹 콘솔에서 리소스 사양을 보고 수정할 수 있는 YAML 편집기를 제공합니다. 시크릿 및 역할 바인딩과 같은 일부 리소스 유형은 리소스 유형에 맞게 구성된 고급 UI를 제공합니다.

사용자 및 그룹 만들기

User Management(사용자 관리) → Users(사용자)에서 액세스할 수 있는 Users(사용자) 페이지에는 OpenShift에 이전에 로그인한 사용자가 표시됩니다. 10장. 인증 및 권한 부여 구성에서 설명한 대로 OpenShift는 HTPasswd, LDAP, OpenID Connect를 포함하여 다양한 IdP(ID 프로바이더)를 지원합니다.

HTPasswd ID 프로바이더를 사용하는 경우 **secrets** 편집기를 사용하면 HTPasswd 시크릿의 항목을 간단히 추가, 업데이트 또는 제거할 수 있습니다. 터미널을 사용하여 신규 또는 업데이트된 HTPasswd 항목을 생성한 후 웹 콘솔로 전환하여 시크릿을 수정합니다.

웹 UI에서 **openshift-config** 프로젝트의 시크릿을 찾은 다음 Actions(작업) → Edit Secret(시크릿 편집)을 클릭합니다. **Edit Key/Value Secret(키/값 시크릿 편집)** 툴은 base64 암호화를 처리합니다. 새 사용자가 OpenShift에 로그인할 수 있도록 행을 추가합니다. 사용자 암호를 변경하려면 행을 업데이트합니다. 사용자가 OpenShift에 로그인할 수 없도록 행을 삭제합니다.

User Management(사용자 관리) → Groups(그룹)에서 액세스할 수 있는 Groups(그룹) 페이지에는 기존 그룹이 표시되고 새 그룹을 만들 수 있는 기능이 있습니다.

프로젝트 만들기

웹 UI는 프로젝트를 구성하는 데 필요한 다양한 페이지와 양식을 갖추고 있습니다.

1. Home(홈) → Projects(프로젝트) 페이지로 이동하여 전체 프로젝트 목록을 표시합니다. Create Project(프로젝트 만들기)를 클릭하고 양식을 완료하여 새 프로젝트를 만듭니다.
2. 새 프로젝트를 만든 후에는 프로젝트 세부 정보 페이지에서 Role Bindings(역할 바인딩) 탭으로 이동할 수 있습니다.
3. Red Hat은 멀티테넌트 클러스터를 담당하는 관리자가 Resource Quotas(리소스 할당량) 및 Limit Ranges(제한 범위)를 구성하여 전체 프로젝트 제한과 컨테이너 제한을 각각 적용하는 것을 권장합니다. Administration(관리) → Resource Quotas(리소스 할당량) 또는 Administration(관리) → Limit Ranges(제한 범위)로 이동하여 이러한 제한을 구성할 수 있는 적절한 YAML 편집기에 액세스합니다.

제한 사항 설명

OpenShift 웹 콘솔은 그래픽으로 OpenShift 클러스터를 관리하는 강력한 도구이지만 일부 관리 작업은 현재 웹 콘솔에서 사용할 수 없습니다. 예를 들어 노드 로그를 보고 노드 디버그 세션을 실행하려면 **oc** 명령줄 도구가 필요합니다.



참조

자세한 내용은

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/web_console/index

에 있는 Red Hat OpenShift Container Platform 4.5 웹 콘솔 설명서를 참조하십시오.

▶ 연습 가이드

클러스터 관리 수행

이 연습에서는 웹 콘솔을 사용하여 클러스터 관리를 수행합니다.

결과

OpenShift 웹 콘솔을 사용하여 다음을 수행할 수 있어야 합니다.

- 운영자와 연결된 리소스를 찾습니다.
- 포드의 상태, YAML 정의 및 로그를 검토합니다.
- 클러스터 구성 리소스를 보고 편집합니다.
- 새 프로젝트를 만들고 리소스 할당량, 제한 범위 및 역할 기반 액세스 제어(RBAC)를 구성합니다.

시작하기 전에

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 이 연습에 필요한 리소스를 만듭니다.

```
[student@workstation ~]$ lab console-admin start
```

▶ 1. admin 사용자로 이동하여 OpenShift 웹 콘솔을 찾아 이동합니다.

- OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 웹 콘솔의 URL을 확인합니다.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 웹 브라우저를 열고 <https://console-openshift-console.apps.ocp4.example.com>으로 이동합니다.
- Advanced(고급)를 클릭하여 신뢰할 수 없는 인증서 메시지를 표시한 다음 Add Exception(예외 추가)을 클릭합니다. Add Security Exception(보안 예외 추가) 대화 상자에서 Confirm Security Exception(보안 예외 확인)을 클릭합니다.
이 작업을 두 번 수행하여 **console-openshift-console** 및 **oauth-openshift** 하위 도메인의 자체 서명 SSL 인증서에 대한 경고를 건너뛰어야 합니다.

- 1.5. **localusers(로컬 사용자)**를 클릭하고 **redhat**을 암호로 사용하여 **admin** 사용자로 로그인합니다.

▶ 2. **openshift-console-operator** 및 **openshift-console** 포드 로그를 검토합니다.

- 2.1. Red Hat Openshift Container Platform 웹 UI에서 **Home(홈) → Projects(프로젝트)**를 클릭하여 **Projects(프로젝트)** 페이지를 표시합니다.
- 2.2. **Search by name(이름으로 검색)** 필드에 **console**을 입력한 다음 **openshift-console-operator** 링크를 클릭합니다.

The screenshot shows the 'Projects' page with a search bar at the top containing 'console'. Below the search bar, there are filter options: 'Name' (selected), 'console' (search term), and 'Clear all filters'. A table lists two projects:

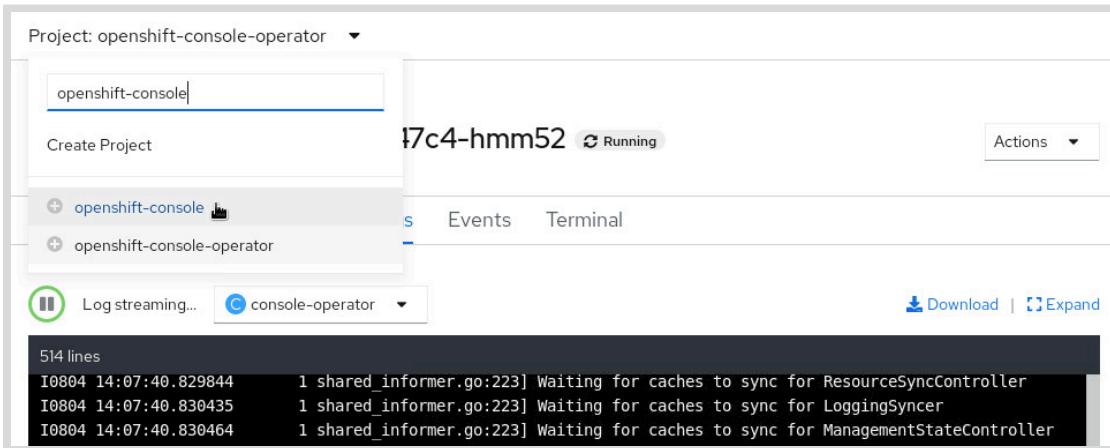
| Name | Display Name | Status | Requester |
|--------------------------------------|-----------------|--------|--------------|
| PR openshift-console | No display name | Active | No requester |
| PR openshift-console-operator | No display name | Active | No requester |

- 2.3. **Workloads(워크로드)**를 클릭한 다음 **포드 1개 중 1개**를 클릭하여 **console-operator** 복제본 집합으로 이동합니다. P 아이콘으로 표시된 포드 이름을 클릭하여 **console-operator** 포드로 이동합니다.

The screenshot shows the 'Replica Sets' details for the 'console-operator' project. It displays a table of pods:

| Name | Namespace | Status | Ready | Node | Memory | CPU |
|--|--------------------------------------|----------------|------------|-------------------|-----------------|--------------------|
| P console-operator-5877c747c4-hmm52 | NS openshift-console-operator | Running | 1/1 | N master02 | 43.4 MiB | 0.004 cores |

- 2.4. **Pod Details(포드 세부 정보)** 페이지를 검토하고 포드 지표, 실행 상태, 볼륨을 확인합니다.
- 2.5. **YAML**을 클릭하여 포드 리소스 편집기로 이동합니다.
- 2.6. **Logs(로그)**를 클릭하여 콘솔 운영자 로그를 확인합니다.
- 2.7. **Project(프로젝트)** 목록을 열고 **openshift-console**을 입력하여 **openshift-console** 프로젝트로 전환합니다.



2.8. 테이블에서 첫 번째 포드를 클릭한 다음 **Logs(로그)**를 클릭하여 콘솔 포드 로그를 확인합니다.

▶ 3. 콘솔, 이미지 및 OAuth 클러스터 설정을 검토합니다.

- 3.1. **Administration(관리) → Cluster Settings(클러스터 설정)**를 클릭하여 **Cluster Settings(클러스터 설정)** 페이지를 확인합니다. 클러스터의 업데이트 채널과 현재 버전에 대한 정보는 상단에 나열되어 있으며 클러스터의 업데이트 기록에 대한 섹션은 아래에 자세히 나와 있습니다.
- 3.2. **Global Configuration(글로벌 구성)**을 클릭하여 클러스터 구성 리소스 목록으로 이동합니다.
- 3.3. **Console(콘솔)**을 클릭한 다음 **YAML**을 클릭하여 **Console(콘솔)** 리소스를 검토합니다.
- 3.4. **Cluster Settings(클러스터 설정)** 글로벌 구성 페이지로 돌아갑니다. **Image(이미지)**를 클릭한 다음 **YAML**을 클릭합니다. **internalRegistryHostname**이 내부 이미지 레지스트리를 사용하도록 구성되어 있는지 확인합니다.
- 3.5. **Cluster Settings(클러스터 설정)** 글로벌 구성 페이지로 돌아가 **OAuth**를 클릭합니다. **OAuth Details(OAuth 세부 정보)** 페이지에는 ID 프로바이더를 나열하고 추가하는 특수 섹션이 있습니다. **YAML** 페이지로 이동하여 추가 구성 세부 정보를 확인합니다.

▶ 4. admin을 검토하고 클러스터 역할을 편집하고 확인합니다.

- 4.1. **User Management(사용자 관리) → Roles(역할)**를 클릭하여 **Roles(역할)** 페이지를 확인합니다.
- 4.2. CR 아이콘 옆에 있는 **admin**을 클릭합니다. 다양한 리소스에 대해 허용된 작업을 설명하는 **Rules(규칙)** 테이블을 검토합니다.

| Roles | | Create Role |
|--|----------------|-----------------------------|
| Name | Namespace | |
| CR admin | All Namespaces | ... |
| CR aggregate-olm-edit | All Namespaces | ... |
| CR aggregate-olm-view | All Namespaces | ... |

- 4.3. Cluster Roles(클러스터 역할) 페이지로 돌아가서 edit라는 클러스터 역할을 클릭하여 edit 클러스터 역할 세부 정보를 확인합니다.
- 4.4. Cluster Roles(클러스터 역할) 페이지로 돌아가 Search by name(이름으로 검색) 필드에 view를 입력합니다. view라는 클러스터 역할을 클릭하여 view 클러스터 역할 세부 정보로 이동합니다. 이 역할은 나열된 리소스에 대한 get, list 및 watch 작업만 허용합니다.

▶ 5. tester 사용자 항목을 localusers 시크릿에 추가합니다.

- 5.1. OpenShift Container Platform 웹 UI에서 Workloads(워크로드) → Secrets(시크릿)를 클릭한 다음 Project(프로젝트) 필터 목록에서 openshift-config를 선택하여 openshift-config 프로젝트에 대한 시크릿을 표시합니다.
- 5.2. 필터를 사용하거나 페이지 아래쪽으로 스크롤하여 localusers 링크를 클릭합니다.
- 5.3. 페이지 오른쪽 상단에 있는 Actions(작업) → Edit Secret(시크릿 편집)을 클릭하여 Edit Key/Value Secret(키/값 시크릿 편집) 도구로 이동합니다.
- 5.4. workstation 터미널에서 redhat을 암호로 사용하여 htpasswd 항목을 생성합니다.

```
[student@workstation ~]$ htpasswd -n -b tester redhat
tester:$apr1$oQ3BtWOp.HtW97.$wVbJBofBNsNd4sd
```

- 5.5. htpasswd 명령의 터미널 출력을 OpenShift 웹 콘솔의 시크릿 편집기에 있는 htpasswd 값에 추가하고 Save(저장)를 클릭합니다.

```
admin:$apr1$Au9.fFr$0k5wvUBd3eeBt0baa77.dae
leader:$apr1$/abo4Hybn7a.tG5ZoOBn.QwefXckiy1
developer:$apr1$RjqTY4cv$xql3.BQfg42moSxwnTNkh.
tester:$apr1$oQ3BtWOp.HtW97.$wVbJBofBNsNd4sd
```

▶ 6. console-apps라는 새 프로젝트를 만들고 구성합니다.

- 6.1. Home(홈) → Projects(프로젝트)를 클릭하여 Projects(프로젝트) 페이지를 표시한 다음 Create Project(프로젝트 만들기)를 클릭합니다.
- 6.2. 새 프로젝트에 다음 정보를 사용한 다음 Create(생성)를 클릭합니다.

프로젝트 양식 생성

| 필드 | 값 |
|-------|------------------------------|
| 이름 | console-apps |
| 표시 이름 | Console chapter applications |
| 설명 | Example project |

- 6.3. Administration(관리) → Resource Quotas(리소스 할당량)를 클릭한 다음 Create Resource Quota(리소스 할당량 만들기)를 클릭합니다. 다음과 같이 YAML 문서를 수정합니다.

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: console-apps
spec:
  hard:
    pods: '10'
    requests.cpu: '2'
    requests.memory: 8Gi
    limits.cpu: '4'
    limits.memory: 12Gi

```

Create(생성)를 클릭합니다.

- 6.4. Administration(관리) → Limit Ranges(제한 범위)를 클릭한 다음 Create Limit Range(제한 범위 만들기)를 클릭합니다. YAML 문서를 수정하여 제한 범위의 이름을 지정합니다. 기본 메모리 및 CPU 컨테이너 제한과 요청을 지정합니다.

```

apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
  namespace: console-apps
spec:
  limits:
    - default:
        cpu: 500m
        memory: 5Gi
    defaultRequest:
        cpu: 10m
        memory: 100Mi
  type: Container

```

Create(생성)를 클릭합니다.

- 6.5. User Management(사용자 관리) → Groups(그룹)를 클릭한 다음 Create Group(그룹 만들기)를 클릭하고 편집기를 사용하여 다음과 같이 그룹 리소스를 정의합니다.

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  name: project-team
users:
  - developer
  - tester

```

Create(만들기)를 클릭하여 새 project-team 그룹을 만듭니다.

- 6.6. User Management(사용자 관리) → Role Bindings(역할 바인딩)를 클릭한 다음 Create Binding(바인딩 만들기)를 클릭합니다. 다음과 같이 양식을 작성하여 project-team 그룹에 대한 역할 바인딩을 만듭니다.

팀 역할 바인딩 양식

| 필드 | 값 |
|--------|--------------------------------------|
| 바인딩 유형 | Namespace Role Binding (RoleBinding) |
| 이름 | team |
| 네임스페이스 | console-apps |
| 역할 이름 | edit |
| 제목 | 그룹 |
| 제목 이름 | project-team |

Create(만들기)를 클릭하여 네임스페이스 지정된 RoleBinding을 만듭니다.

- Role Bindings(역할 바인딩) 페이지로 돌아가서 Create Binding(바인딩 만들기)을 클릭하여 **leader** 사용자에 대한 역할 바인딩을 만듭니다. 아래와 같이 양식을 작성합니다.

리더 역할 바인딩 양식

| 필드 | 값 |
|--------|--------------------------------------|
| 바인딩 유형 | Namespace Role Binding (RoleBinding) |
| 이름 | leader |
| 네임스페이스 | console-apps |
| 역할 이름 | admin |
| 제목 | User |
| 제목 이름 | leader |

Create(만들기)를 클릭하여 네임스페이스 지정된 RoleBinding을 만듭니다.

- admin → Log out(로그아웃)을 클릭한 다음 developer를 암호로 사용하여 developer 사용자로 다시 로그인합니다.
developer 계정이 console-apps 프로젝트만 볼 수 있는지 확인합니다.



참고

완료 시 삭제되지 않은 단계별 연습의 이전 프로젝트는 목록에도 표시될 수 있습니다.

- 다음 섹션에서 새 console-apps 프로젝트를 계속 사용할 수 있으므로 삭제하지 않아도 됩니다.

완료

workstation 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab console-admin finish
```



중요

console-apps 프로젝트를 삭제하지 마십시오. 다음 섹션에서 사용될 예정입니다.

이것으로 섹션을 완료합니다.

워크로드 및 운영자 관리

목표

이 섹션을 완료하면 웹 콘솔을 사용하여 애플리케이션과 Kubernetes 운영자를 관리할 수 있습니다.

워크로드 리소스 살펴보기

포드, 배포, 상태 저장 집합 및 구성 맵과 같은 워크로드 리소스는 **Workloads(워크로드)** 메뉴에 나열됩니다. 리소스 유형을 클릭하여 리소스 목록을 표시한 다음 리소스의 이름을 클릭하여 해당 리소스에 대한 세부 정보 페이지로 이동합니다.

예를 들어 OpenShift DNS 운영자 포드로 이동하려면 **Workloads(워크로드) → Pods(포드)**를 클릭하고 페이지 상단의 프로젝트 목록에서 **openshift-dns-operator**를 선택한 다음 테이블에 나열된 포드의 이름을 클릭합니다.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a dark theme with a navigation menu. The 'Workloads' section is expanded, and 'Pods' is selected. The main content area is titled 'Pod Details' for a pod named 'dns-operator-746bbc44b5-tjf2n'. It shows several monitoring charts: Memory Usage (40 MiB), CPU Usage (0.8m), Filesystem (40 KiB), Network In (600 Bps), and Network Out (400 Bps). Below the charts, the pod's name is listed as 'dns-operator-746bbc44b5-tjf2n' and its status as 'Running'.

주로 여러 가지 방법으로 일반적인 리소스로 이동할 수 있습니다. 웹 UI 전반에 걸쳐 연결된 리소스가 서로 연결되는 경우가 많습니다. 배포 세부 정보 페이지에 포드 목록이 표시됩니다. 해당 포드에 대한 포드 세부 정보 페이지를 표시하려면 해당 목록에서 포드 이름을 클릭합니다.

워크로드 관리

웹 콘솔은 많은 워크로드 리소스에 대한 특수 편집기 페이지를 제공합니다. 리소스의 세부 정보 페이지에서 **Actions(작업)** 메뉴를 사용하여 특수 편집기 페이지로 이동합니다.

The screenshot shows the 'Deployment Details' page for the 'openshift-apiserver-operator' deployment. The 'Actions' dropdown menu is highlighted with a red border. The menu items include:

- Edit Pod Count
- Pause Rollouts
- Add Health Checks
- Add Storage
- Edit Update Strategy
- Edit Labels
- Edit Annotations
- Edit Deployment
- Delete Deployment

그림 15.8: 작업 메뉴를 사용하여 배포를 수정합니다.

다음은 몇 가지 유용한 작업 페이지에 대한 설명입니다.

- 모든 리소스에는 **Edit Labels**(레이블 편집) 및 **Edit Annotations**(주석 편집) 편집기 기능이 있습니다.
- Actions(작업)** → **Add Storage**(스토리지 추가)를 클릭하여 배포에 영구 볼륨 클레임(PVC)을 추가합니다.
- 복제본 개수를 편집하려면 **Deployment Details**(배포 세부 정보) 페이지로 이동하고 **Actions(작업)** → **Edit Pod Count**(포드 개수 편집)를 클릭합니다.
- 롤링 업데이트 매개변수 변경 등 배포에 대한 업데이트 전략을 수정하려면 **Deployment Details** (배포 세부 정보) 페이지로 이동하고 **Actions(작업)** → **Edit Update Strategy**(업데이트 전략 편집)을 클릭합니다. 배포 구성에 대한 업데이트 전략을 수정하려면 **Deployment Config Details** (배포 구성 세부 정보) 페이지로 이동하고 **Actions(작업)** → **Edit Deployment Config**(배포 구성 편집)를 클릭합니다.
- Secret Details**(시크릿 세부 정보) 페이지로 이동하고 **Actions(작업)** → **Edit Secrets**(시크릿 편집)를 클릭하여 Base64를 자동으로 사용하여 값을 인코딩 및 디코딩하는 **Edit Key/Value Secret**(키/값 시크릿 편집) 도구를 표시합니다.

또한 포함된 YAML 편집기를 사용하여 워크로드 리소스를 만들거나 수정할 수 있습니다. **oc** 명령을 사용하지 않고 JSON 또는 YAML 파일을 브라우저 기반 편집기로 끌어서 놓아 파일에서 리소스를 업데이트합니다.

```

1 kind: Deployment
2 apiVersion: apps/v1
3 metadata:
4   annotations:
5     deployment.kubernetes.io/revision: '1'
6     exclude.release.openshift.io/internal-openshift-hosted: 'true'
7   selfLink: >-
8     /apis/apps/v1/namespaces/openshift-apiserver-operator/deployments/openshift-apiserver-operator
9   resourceVersion: '35597'
10  name: openshift-apiserver-operator
11  uid: 93b3f536-65c3-4b47-baa1-67511a26aad4
12  creationTimestamp: '2020-07-29T18:30:38Z'
13  generation: 1
14  managedFields:
15    - manager: cluster-version-operator
16      operation: Update
17      apiVersion: apps/v1
18      time: '2020-07-29T18:30:38Z'
19      fieldsType: FieldsV1
20      fieldsV1:
21        'f:metadata':
22          'f:annotations':
23            .: {}
24            'f:exclude.release.openshift.io/internal-openshift-hosted': {}
25        'f:labels':
26          .: {}
27        'f:app': {}

```

Save Reload Cancel Download

그림 15.9: 포함된 YAML 편집기를 사용하여 리소스 편집

전용 페이지 또는 포함된 YAML 편집기에서 리소스를 편집하는 기능과 함께 OpenShift 웹 콘솔에서 직접 많은 다른 일반적인 작업을 수행할 수 있습니다.

예를 들어 리소스를 삭제하려면 리소스의 세부 정보 페이지로 이동하고 **Actions(작업) → Delete Resource Type(리소스 유형 삭제)**를 클릭합니다.

일반적으로 특정 작업을 수행하는 방법에는 여러 가지가 있습니다. 예를 들어 배포를 수동으로 확장하려면 **Deployment Details(배포 세부 정보)** 페이지로 이동한 다음 **Actions(작업) → Edit Pod Count(포드 개수 편집)**를 클릭하거나 페이지를 벗어나지 않고 포드 개수 옆에 있는 화살표를 클릭할 수 있습니다.

애플리케이션 배포

Workloads(워크로드) → Deployments(배포) 및 **Workloads(워크로드) → Deployment Configs(배포 구성)** 페이지에서 배포 및 배포 구성을 각각 만들 수 있습니다. 각 섹션은 YAML 리소스를 정의하는 미리 채워져 있는 사양으로 YAML 편집기를 제공합니다.

Builds(빌드) 섹션에는 다음에 대한 도구가 포함되어 있습니다.

- S2I(Source-to-Image), Dockerfile 또는 사용자 지정 빌드에 대한 빌드 구성 만들기
- 빌드를 나열하고 검사합니다.
- 이미지 스트림을 관리합니다.

배포 또는 빌드를 시작한 후에는 리소스의 세부 정보 및 이벤트 페이지를 사용하여 성공 여부를 확인하거나 배포 실패의 원인을 조사합니다.

운영자 설치 및 사용

OpenShift 웹 콘솔의 **Operators(운영자)** → **OperatorHub** 페이지에서 커뮤니티 및 파트너 운영자를 탐색합니다. 360개 이상의 운영자는 웹 UI에서 설치하는 데 사용할 수 있습니다. 여기에는 Red Hat이 지원하지 않는 커뮤니티 운영자가 포함됩니다.

운영자는 배포 조정 또는 자동 백업 등의 사람 운영자가 수행한 자동화 기능과 함께 클러스터에 기능 및 서비스를 추가합니다. 운영자는 다음을 포함한 광범위한 범주를 다룹니다.

- PostgreSQL 및 MySQL과 같은 전통적인 데이터베이스.
- Apache Spark와 같은 널리 사용되는 빅 데이터 프레임워크.
- Red Hat AMQ 스트림과 같은 Kafka 기반 스트리밍 플랫폼.
- Knative 서비스 프레임워크 OpenShift 서비스 운영자.

운영자 목록을 클릭하여 운영자에 대한 세부 정보(예: 버전 및 설명서를 찾을 수 있는 위치)를 확인합니다.

운영자를 설치할 준비가 되면 **Install(설치)**를 클릭하여 설치를 시작합니다. **Operator Installation(운영자 설치)** 양식을 완료하여 타겟 네임스페이스 및 운영자 승인 전략을 선택합니다. 전체 네임스페이스 또는 특정 네임스페이스만을 대상으로 하는 운영자를 설치할 수 있습니다. 그러나 모든 운영자가 모든 설치 대상 옵션을 지원하는 것은 아니라는 점에 유념하십시오.

운영자를 설치하고 나면 **Operators(운영자)** → **Installed Operators(설치된 운영자)** 페이지에 표시됩니다. 특정 네임스페이스에 대해 설치된 경우 페이지의 맨 위에 있는 프로젝트 필터를 사용하여 올바른 프로젝트를 선택해야 합니다.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar is collapsed, and the top navigation bar shows the Red Hat logo and 'OpenShift Container Platform'. The user is logged in as 'admin'. The main content area is titled 'example-project'. Under the 'Operators' menu, 'Installed Operators' is selected. The 'etcd' operator is highlighted. The 'Details' tab is active, showing the operator's name ('etcd'), version ('0.9.4 provided by CNCF'), and an 'Actions' dropdown. Below the tabs are buttons for 'YAML', 'Subscription', 'Events', 'All Instances', 'etcd Cluster', 'etcd Backup', and 'etcd Restore'. The 'Provided APIs' section contains two items: 'etcd Cluster' (described as 'Represents a cluster of etcd nodes.' with a 'Create Instance' link) and 'etcd Backup' (described as 'Represents the intent to backup an etcd cluster.' with a 'Create Instance' link). To the right of these items are columns for 'Provider' (CNCF), 'Created At' (a minute ago), 'Links' (Blog, Documentation), and 'Documentation' (link to https://coreos.com/etcd).

운영자 세부 정보 페이지에는 운영자에서 제공한 API가 나열되며 해당 리소스의 인스턴스를 만들 수 있습니다. 예를 들어 etcd 운영자 페이지에서 etcd 클러스터, 백업 요청 또는 복원 요청의 인스턴스를 만들 수 있습니다.



참조

자세한 내용은

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/web_console/index

에 있는 Red Hat OpenShift Container Platform 4.5 웹 콘솔 설명서를 참조하십시오.

자세한 내용은

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/operators/index

에 있는 Red Hat OpenShift Container Platform 4.5 운영자 설명서를 참조하십시오.

▶ 연습 가이드

워크로드 및 운영자 관리

이 연습에서는 웹 콘솔을 사용하여 클러스터 워크로드를 관리합니다.

결과

OpenShift 웹 콘솔을 사용하여 다음을 수행할 수 있어야 합니다.

- OperatorHub에서 운영자를 설치합니다.
- 사용자 지정 리소스를 사용하여 데이터베이스를 만듭니다.
- 운영자가 관리하는 리소스를 사용하는 애플리케이션을 배포하고 문제를 해결합니다.

시작하기 전에

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 이 활동에 필요한 리소스를 만듭니다.

```
[student@workstation ~]$ lab console-workloads start
```

▶ 1. **admin** 사용자로 이동하여 OpenShift 웹 콘솔을 찾아 이동합니다.

- 1.1. OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. 웹 콘솔의 URL을 확인합니다.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 1.3. 웹 브라우저를 열고 **https://console-openshift-console.apps.ocp4.example.com**으로 이동합니다.
- 1.4. Advanced(고급)를 클릭하여 신뢰할 수 없는 인증서 메시지를 표시한 다음 Add Exception(예외 추가)을 클릭합니다. Add Security Exception(보안 예외 추가) 대화 상자에서 Confirm Security Exception(보안 예외 확인)을 클릭합니다.
이 작업을 두 번 수행하여 **console-openshift-console** 및 **oauth-openshift** 하위 도메인의 자체 서명 SSL 인증서에 대한 경고를 건너뛰어야 합니다.
- 1.5. localusers(로컬 사용자)를 클릭하고 **redhat**을 암호로 사용하여 **admin** 사용자로 로그인합니다.

- ▶ 2. `openshift-console-operator` 및 `openshift-console` 배포, 복제본 집합 및 포드를 검사합니다.

- 2.1. Workloads(워크로드) → Deployments(배포)를 클릭하고 맨 위의 프로젝트 목록에서 **all projects**를 선택합니다. Filter by name(이름으로 필터링) 필드에 `console`을 입력합니다.

Openshift에는 `openshift-console` 네임스페이스에서 `console`이라는 배포를 운영하는 `openshift-console-operator` 네임스페이스에 단일 포드가 있는 `console-operator`라는 배포가 있습니다.

| Name | Namespace | Status | Labels | Pod Selector |
|-------------------------------|--|-------------|---|--|
| <code>console</code> | <code>NS openshift-console</code> | 2 of 2 pods | <code>app=console</code>
<code>component=ui</code> | <code>app=console,</code>
<code>component=ui</code> |
| <code>console-operator</code> | <code>NS openshift-console-operator</code> | 1 of 1 pods | No labels | <code>name=console-operator</code> |

- 2.2. Workloads(워크로드) → Replica Sets(복제본 집합)를 클릭하고 Search by name(이름으로 검색) 필드에 `console`을 입력합니다.

배포에서는 지정된 수의 포드가 항상 실행되고 있는지 확인하기 위해 `ReplicaSet`를 선언합니다.

- 2.3. 상태 열에서 2개의 포드 중 2개를 클릭하여 콘솔 `ReplicaSet` 포드 목록을 표시합니다.

- ▶ 3. OperatorHub 페이지에서 Dev4Devs.com이 제공하는 커뮤니티 PostgreSQL 운영자를 설치합니다.

- 3.1. Operators(운영자) → OperatorHub를 클릭한 다음 Database(데이터베이스)를 클릭하여 OperatorHub에서 사용할 수 있는 데이터베이스 운영자 목록을 표시합니다.

- 3.2. Filter by keyword(키워드로 필터) 필드에 `postgres`를 입력한 다음 PostgreSQL Operator by Dev4Devs.com(Dev4Devs.com의 PostgreSQL 운영자)을 클릭합니다. Continue(계속)를 클릭하여 커뮤니티 운영자 페이지를 확인한 다음 Install(설치)을 클릭합니다.

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

| All Items | Database | 4 items |
|--|---------------------------------------|---------|
| AI/Machine Learning | <input type="text" value="postgres"/> | |
| Application Runtime | | |
| Big Data | | |
| Cloud Provider | | |
| Database | | |
| Developer Tools | | |
| Integration & Delivery | | |
| Logging & Tracing | | |
| Monitoring | | |
| Networking | | |
| OpenShift Optional | | |
| Security | | |
| Storage | | |
| Streaming & Messaging | | |
| Install State | | |
| <input type="checkbox"/> Installed (0) | | |
| <input type="checkbox"/> Not Installed (4) | | |
| Provider Type | | |
| <input type="checkbox"/> Red Hat (0) | | |

- 3.3. **console-apps** 네임스페이스를 선택한 다음 **Install(설치)**을 클릭하여 **console-apps** 프로젝트에서 사용할 운영자를 설치합니다. 다른 양식 필드는 변경하지 않고 그대로 둡니다.
4. **admin** 사용자로 로그아웃하고 **developer** 사용자로 로그인합니다.
 - 4.1. **admin** → **Log out(로그아웃)**을 클릭합니다.
 - 4.2. 암호 **developer**를 사용하여 사용자 **developer**로 로그인합니다.
5. 설치된 운영자와 **Database(데이터베이스)** 사용자 지정 리소스 정의(CRD)를 사용하여 PostgreSQL 데이터베이스를 프로비저닝합니다.
 - 5.1. **Projects(프로젝트)** 페이지에서 **console-apps** 링크를 클릭하여 **console-apps** 프로젝트와 연결된 리소스를 확인합니다.
 - 5.2. **Operators(운영자)** → **Installed Operators(설치된 운영자)**를 클릭한 다음 **PostgreSQL Operator by Dev4Devs.com** 링크를 클릭하여 **Operator Details(운영자 세부 정보)** 페이지를 표시합니다.



참고

Installed Operators(설치된 운영자) 목록이 로드되지 않는 경우 페이지 상단에 **console-apps** 프로젝트가 선택되어 있는지 확인합니다.

Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the [Operator Lifecycle Manager documentation](#). Or create an Operator and Cluster Service Version using the [Operator SDK](#).

| Name | Managed Namespaces | Status | Provided APIs |
|---|--------------------|-----------|--------------------------------------|
|  PostgreSQL Operator by Dev4Devs.com | NS console-apps | Succeeded | Database Backup
Database Database |
| 0.1.1 provided by Dev4Devs.com | | | |

- 5.3. Database Database(데이터베이스 데이터베이스) 탭을 클릭한 다음 Create Database(데이터베이스 만들기)를 클릭합니다.
- 5.4. Form View(양식 보기)에서 YAML View(YAML 보기)로 전환한 다음 Database YAML을 업데이트하여 Red Hat에서 제공하는 PostgreSQL 이미지를 지정합니다. 다른 기본값은 변경하지 마십시오.

```
apiVersion: postgresql.dev4devs.com/v1alpha1
kind: Database
metadata:
  name: database
  namespace: console-apps
spec:
  ...output omitted...
  databaseUserKeyEnvVar: POSTGRESQL_USER
  image: registry.redhat.io/rhsc1/postgresql-96-rhel7:1-51
  size: 1
```

- 5.5. Create(만들기)를 클릭하여 Database(데이터베이스) 리소스를 추가합니다. PostgreSQL 운영자는 사양을 읽고 새 데이터베이스에 대한 워크로드, 네트워크 및 스토리지를 자동으로 만듭니다.
- ▶ 6. 운영자가 만든 리소스를 검토합니다.
 - 6.1. Workloads(워크로드) → Deployments(배포)를 클릭하고 배포 목록을 검사합니다. 데이터베이스 배포 및 postgresql-operator 배포를 확인합니다.
 - 6.2. 데이터베이스 배포를 클릭한 다음 Pods(포드) 탭을 클릭하여 데이터베이스 배포에 의해 배포된 포드를 확인합니다. 포드 이름을 클릭하여 Pod Details(포드 세부 정보) 페이지를 표시합니다.
 - 6.3. Networking(네트워킹) → Services(서비스)를 클릭한 다음 database 서비스 이름을 클릭하여 PostgreSQL 운영자에서 만든 서비스의 세부 정보를 확인합니다.
 - 6.4. Storage(스토리지) → Persistent Volume Claims(영구 볼륨 클레임)를 클릭한 다음 database PVC를 클릭하여 PostgreSQL 운영자에서 만든 영구 볼륨 클레임의 세부 정보를 확인합니다.

- ▶ 7. 간단한 웹 애플리케이션에 대한 배포, 서비스 및 경로를 만듭니다. 애플리케이션에는 데이터베이스에 저장된 서적 목록이 표시됩니다.

- 7.1. Workloads(워크로드) → Deployments(배포)를 클릭한 다음 Create Deployment(배포 만들기)를 클릭하여 웹 콘솔 YAML 편집기를 표시합니다. 다음과 같이 YAML을 업데이트 다음 Create(만들기)를 클릭합니다.



참고

workstation의 ~/D0280/labs/console-workloads/deployment.yaml 파일에서 YAML을 복사할 수 있습니다.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: books
  namespace: console-apps
spec:
  replicas: 1
  selector:
    matchLabels:
      app: books
  template:
    metadata:
      labels:
        app: books
    spec:
      containers:
        - name: books
          image: 'quay.io/redhattraining/books:v0.9'
          ports:
            - containerPort: 8080
              protocol: TCP
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
          env:
            - name: DB_HOST
              value: database.console-apps.svc.cluster.local
            - name: DB_PORT
              value: '5432'
            - name: DB_USER
              value: postgres
            - name: DB_PASSWORD
              value: postgres
            - name: DB_NAME
              value: postgres
```

**중요**

이 단계를 완료한 후 포드가 성공적으로 실행될 것으로 예상하지 마십시오. 이 연습의 뒷부분에서 배포 문제를 해결하게 됩니다.

- 7.2. Networking(네트워킹) → Services(서비스)를 클릭한 다음 Create Service(서비스 만들기)를 클릭하여 웹 콘솔 YAML 편집기를 표시합니다. 다음과 같이 YAML을 업데이트한 다음 Create(만들기)를 클릭합니다.

**참고**

workstation의 ~/DO280/labs/console-workloads/service.yaml 파일에서 YAML을 복사할 수 있습니다.

```
kind: Service
apiVersion: v1
metadata:
  name: books
  namespace: console-apps
spec:
  selector:
    app: books
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

- 7.3. Networking(네트워킹) → Routes(경로)를 클릭한 다음 Create Route(경로 만들기)를 클릭합니다. 다른 필드는 변경하지 않고 그대로 두고 페이지를 다음과 같이 완성한 다음 Create(만들기)를 클릭합니다.

경로 양식 만들기

| 필드 | 값 |
|-------|-------------------|
| 이름 | books |
| 서비스 | books |
| 대상 포트 | 8080 → 8080 (TCP) |

- ▶ 8. 배포 문제를 해결하고 수정합니다.

- 8.1. Home(홈) → Events(이벤트)를 클릭하고 오류 이벤트를 확인합니다. "quay.io/redhattraining/books:v0.9" 이미지 가져오기 실패 및 오류: ImagePullBackOff는 이미지 이름 또는 이미지 태그와 관련된 문제를 나타냅니다.

Events

Resources All ▾ All Types ▾ Filter Events by name or message...

Resource A All X

Streaming events... Showing 39 events

| Event Type | Pod | Namespace | Timestamp | Message |
|------------|------------------------|--------------|-------------------|--|
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Generated from kubelet on master01 |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Pulling image "quay.io/redhattraining/books:v0.9" |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Failed to pull image "quay.io/redhattraining/books:v0.9": rpc error: code = Unknown desc = Error reading manifest v0.9 in quay.io/redhattraining/books: manifest unknown: manifest unknown |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Error: ErrImagePull |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Generated from kubelet on master01 |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Back-off pulling image "quay.io/redhattraining/books:v0.9" |
| P | books-5f7fbffdb7-ngbk5 | console-apps | a few seconds ago | Error: ImagePullBackOff |

- 8.2. Workloads(워크로드) → Deployments(배포)를 클릭한 다음 books 배포를 클릭합니다. 페이지 하단으로 스크롤하여 Conditions(조건) 테이블을 검사합니다. Available(사용 가능) 조건 유형에서 False 상태가 표시되는지 확인합니다.

Conditions

| Type | Status | Updated | Reason | Message |
|-------------|--------|---------------|----------------------------|--|
| Available | False | 4 minutes ago | MinimumReplicasUnreachable | Deployment does not have minimum availability. |
| Progressing | True | 4 minutes ago | ReplicaSetUpdated | ReplicaSet "books-695647ff54" is progressing. |

- 8.3. Deployment Details(배포 세부 정보) 화면 상단에 있는 Pods(포드) 탭을 클릭하고 포드 상태를 찾습니다. ImagePullBackOff가 표시됩니다.
- 8.4. Deployment Details(배포 세부 정보) 페이지의 상단에 있는 YAML 탭을 클릭하여 YAML 편집기로 이동하고 문제를 해결합니다. spec 이미지 값을 'quay.io/redhattraining/books:v1.4'로 업데이트한 다음 Save(저장)를 클릭합니다.



참고

업데이트를 시도하는 동안 OpenShift가 배포 리소스를 업데이트하는 경우 YAML 편집기에서 최신 버전을 가져오지 않고 변경 사항을 저장할 수 없습니다. 이 경우 Reload(다시 로드)를 클릭하고 편집을 다시 수행한 다음 Save(저장)를 클릭합니다.

- 8.5. **Deployment Details(배포 세부 정보)** 페이지 상단에 있는 **Details(세부 정보)** 탭을 클릭하고 포드 배포를 모니터링합니다. 하지만 포드는 여전히 시작되지 않습니다.
- 8.6. **Home(홈) → Events(이벤트)**를 클릭하고 추가 문제에 대한 증거를 검색합니다. 새 이벤트 메시지에 할당량 문제가 표시됩니다.

```
Error creating: pods "books-5c65dc95-z9bss" is forbidden: exceeded quota:
quota, requested: limits.memory=5Gi, used: limit.memory=10752Mi, limited:
limits.memory=12Gi
```

books 배포를 업데이트하여 새 복제본 집합이 생성되었지만 새 복제본 집합의 포드를 예약하면 메모리 제한에 대한 프로젝트 할당량을 초과합니다.

- 8.7. 이 문제를 해결하려면 기존 포드가 포함된 **books** 복제본 집합을 확인하고 삭제합니다. 실패한 포드가 포함된 복제본 집합을 삭제하면 할당량 사용이 줄어들고 새 복제본 집합에서 포드를 예약할 수 있습니다. **Workloads(워크로드) → Replica Sets(복제본 집합)**를 클릭합니다.
books 배포를 위한 두 개의 복제본 집합이 있어야 합니다. 상태가 **1 of 1 pods**인 **books** 복제본 집합은 잘못된 컨테이너 이미지 버전에 해당합니다. 행에 대한 세로 줄임표 메뉴를 사용하고 **Delete Replica Set(복제본 집합 삭제)**를 선택하여 복제본 집합을 삭제합니다.
Delete(삭제)를 클릭하여 삭제를 확인합니다.

| Name | Namespace | Status | Labels | Owner | Created | Actions |
|-----------------------------------|-----------------|-------------|--|-----------------------|---------------|---------|
| RS books-5c65dc95 | NS console-apps | 0 of 1 pods | app=books
pod-template...=5c65d... | D books | 2 minutes ago | ⋮ |
| RS books-5f7fbffdb7 | NS console-apps | 1 of 1 pods | app=books
pod-template...=5f7fb... | D books | 8 minutes ago | ⋮ |
| RS database-599f5f4fb8 | NS console-apps | 1 of 1 pods | cr=database
own...="postgresqloperat...
pod-templat...=599f5f... | D database | 8 minutes | ⋮ |
| RS postgresql-operator-67df97f444 | NS console-apps | 1 of 1 pods | na...="postgresql-operat...
pod-templat...=67df97... | D postgresql-operator | 9 minutes | ⋮ |

- 8.8. **Workloads(워크로드) → Deployments(배포)**를 클릭한 다음 **books** 배포에 대한 링크를 클릭합니다. 도넛에 1개의 포드가 실행되고 있는 것으로 표시될 때까지 기다립니다.
- 8.9. **Networking(네트워킹) → Routes(경로)**를 클릭한 다음 **Location(위치)** 옆에서 링크를 클릭합니다. Firefox에서는 데이터베이스에서 가져온 서적 목록을 새로 렌더링하여 새 탭을 엽니다.
- 8.10. 다음 섹션에서 새 **console-apps** 프로젝트와 서적 배포를 계속 사용하므로 삭제하지 않아도 됩니다.

완료

workstation 시스템에서 **lab** 명령을 사용하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab console-workloads finish
```



중요

`console-apps` 프로젝트 또는 이 섹션에서 수행한 작업을 삭제하지 마십시오. 다음 섹션에서 사용될 예정입니다.

이것으로 섹션을 완료합니다.

클러스터 지표 검사

목표

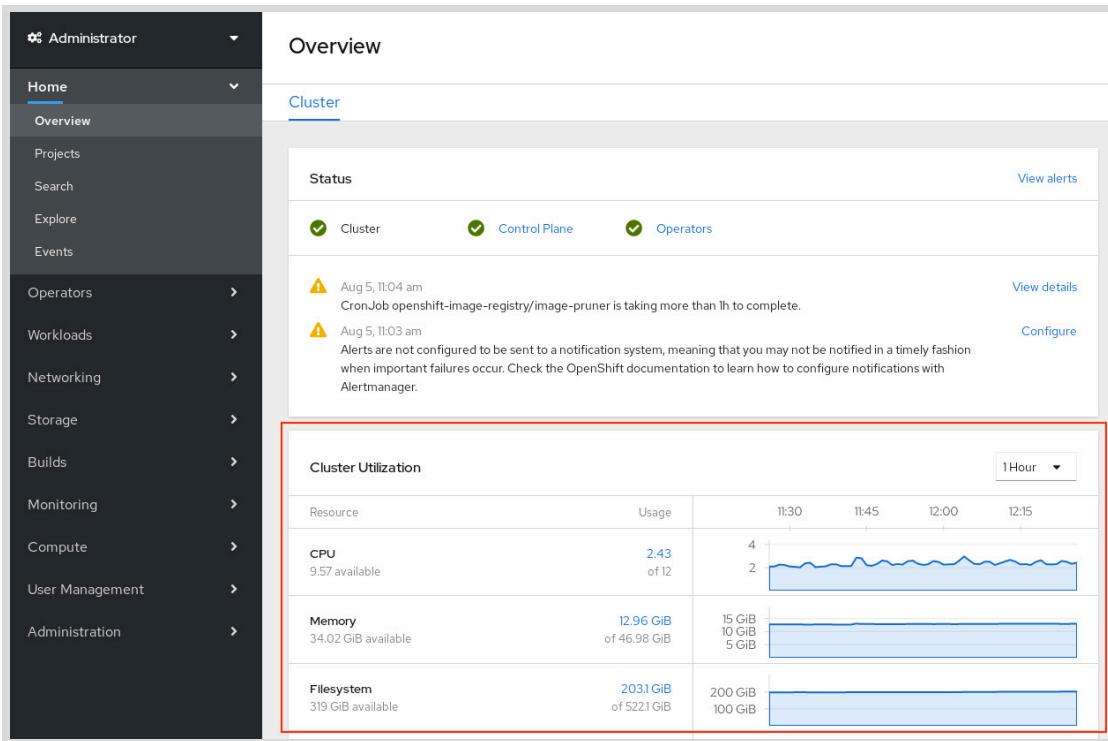
이 섹션을 마치면 클러스터 노드와 애플리케이션의 성능 및 상태 지표를 검사할 수 있습니다.

클러스터 지표 확인

OpenShift 웹 콘솔은 클러스터 및 리소스 분석을 시각화하는 데 유용한 그래프를 통합합니다. 클러스터 관리자와 **view** 클러스터 역할 또는 **cluster-monitoring-view** 클러스터 역할의 사용자는 Home(홈) → Overview(개요) 페이지에 액세스할 수 있습니다. 클러스터 전체 지표의 컬렉션을 표시하는 Overview(개요) 페이지에서는 클러스터의 전체 상태에 대한 간략한 보기를 제공합니다.

개요에는 다음이 포함됩니다.

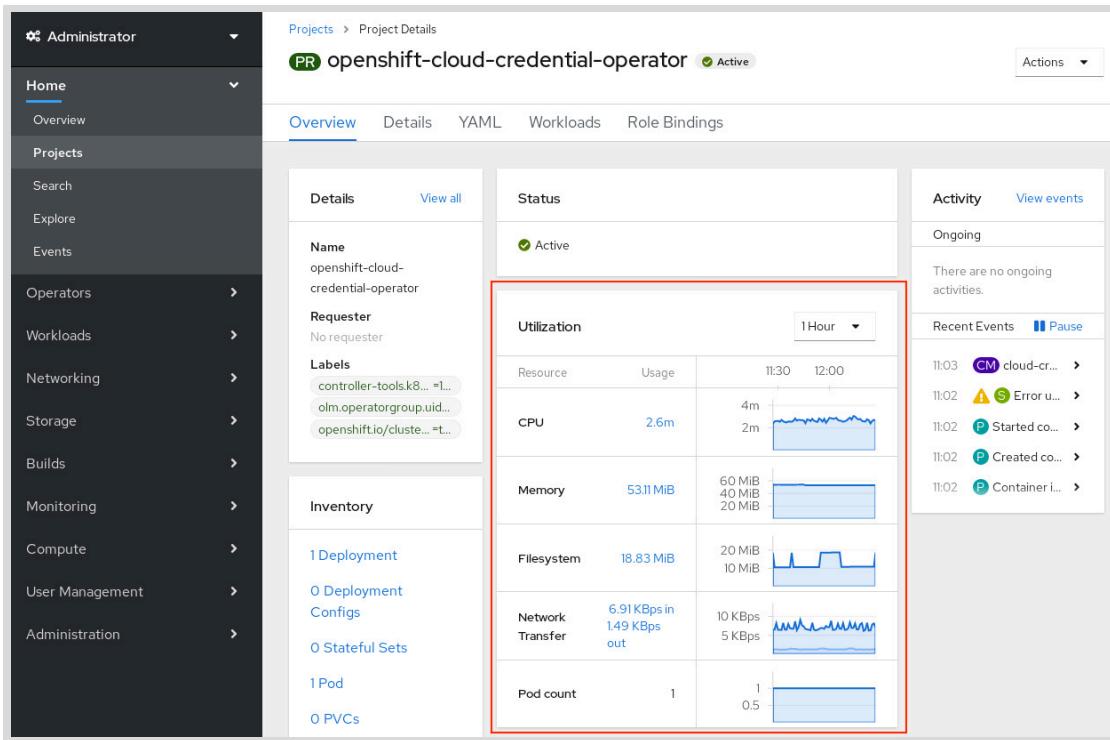
- CPU, 메모리, 스토리지 및 네트워크 사용량을 기반으로 하는 현재 클러스터 용량.
- 총 CPU, 메모리 및 디스크 사용률에 대한 시계열 그래프.
- CPU, 메모리, 스토리지의 상위 소비자를 표시할 수 있는 기능



Cluster Utilization(클러스터 사용률) 섹션에 나열된 리소스의 경우 관리자가 현재 리소스 사용량에 대한 링크를 클릭할 수 있습니다. 링크를 클릭하면 해당 리소스의 상위 소비자가 분류된 창이 표시됩니다. 상위 소비자는 프로젝트, 포드 또는 노드를 기준으로 정렬할 수 있습니다. 상위 소비자 목록은 문제가 있는 포드 또는 노드를 식별하는데 유용할 수 있습니다. 예를 들어 예기치 않은 메모리 누수가 있는 포드는 목록의 맨 위에 나타날 수 있습니다.

프로젝트 지표 보기

Project Details(프로젝트 세부 정보) 페이지에는 특정 프로젝트의 범위 내에서 사용된 리소스에 대한 개요를 제공하는 지표가 표시됩니다. **사용률** 섹션에는 CPU 및 메모리와 같은 리소스에 대한 사용량 정보와 함께 각 리소스의 상위 소비자를 표시하는 기능이 표시됩니다.



모든 지표는 프로메테우스에서 가져옵니다. 그래프를 클릭하여 **Metrics(지표)** 페이지로 이동합니다. 실행된 쿼리를 보고 데이터를 더 자세히 검사합니다.

프로젝트에 대한 리소스 할당량이 생성되면 현재 프로젝트 요청 및 제한이 **Project Details(프로젝트 세부 정보)** 페이지에 표시됩니다.

리소스 지표 보기

문제를 해결하는 경우 전체 클러스터 또는 전체 프로젝트보다 더 작은 세분성에서 지표를 확인하는 것이 유용할 수 있습니다. **Pod Details(포드 세부 정보)** 페이지에는 특정 포드에 대한 CPU, 메모리 및 파일 시스템 사용량의 시계열 그래프가 표시됩니다.

높은 부하로 인해 발생하는 CPU 스파이크와 같이 이러한 중요 지표의 갑작스러운 변경은 이 페이지에 표시됩니다.

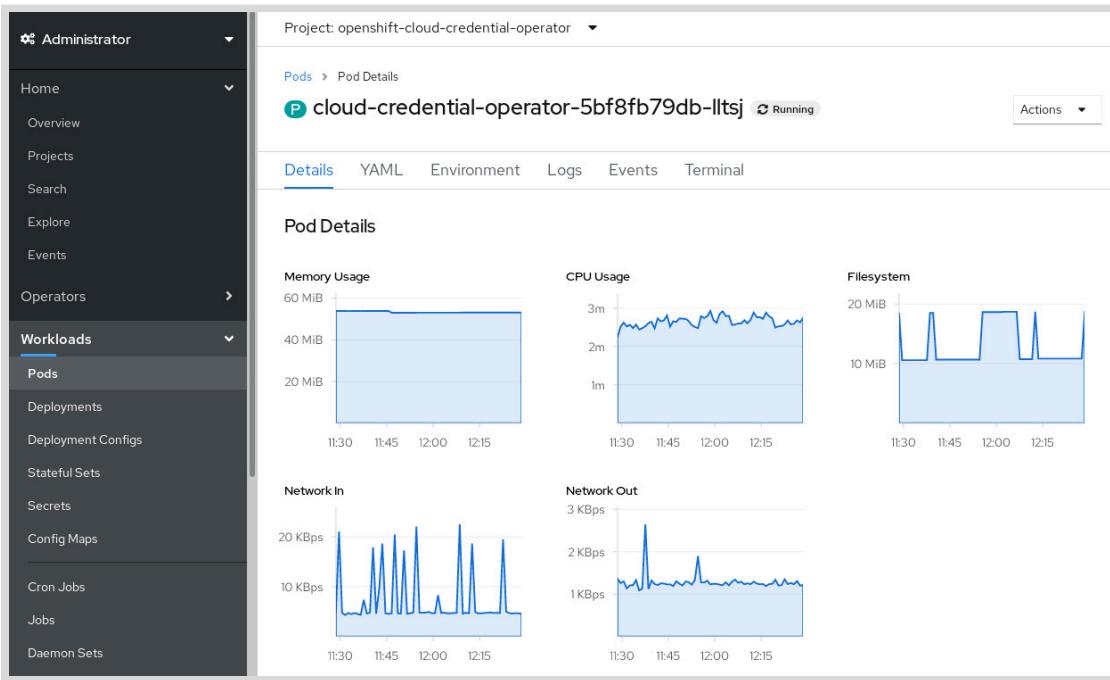


그림 15.19: 포드에 대한 다양한 지표를 보여주는 시계열 그래프.

웹 콘솔에서 프로메테우스 쿼리 수행

프로메테우스 UI는 지표를 시각화하고 경고를 구성하는데 사용할 수 있는 풍부한 기능을 갖춘 도구입니다. OpenShift 웹 콘솔은 웹 콘솔에서 직접 프로메테우스 쿼리를 실행하기 위한 인터페이스를 제공합니다.

쿼리를 수행하려면 Monitoring(모니터링) → Metrics(지표)로 이동하여 텍스트 필드에 프로메테우스 쿼리 언어식을 입력하고 Run Queries(쿼리 실행)를 클릭합니다. 쿼리 결과는 시계열 그래프로 표시됩니다.

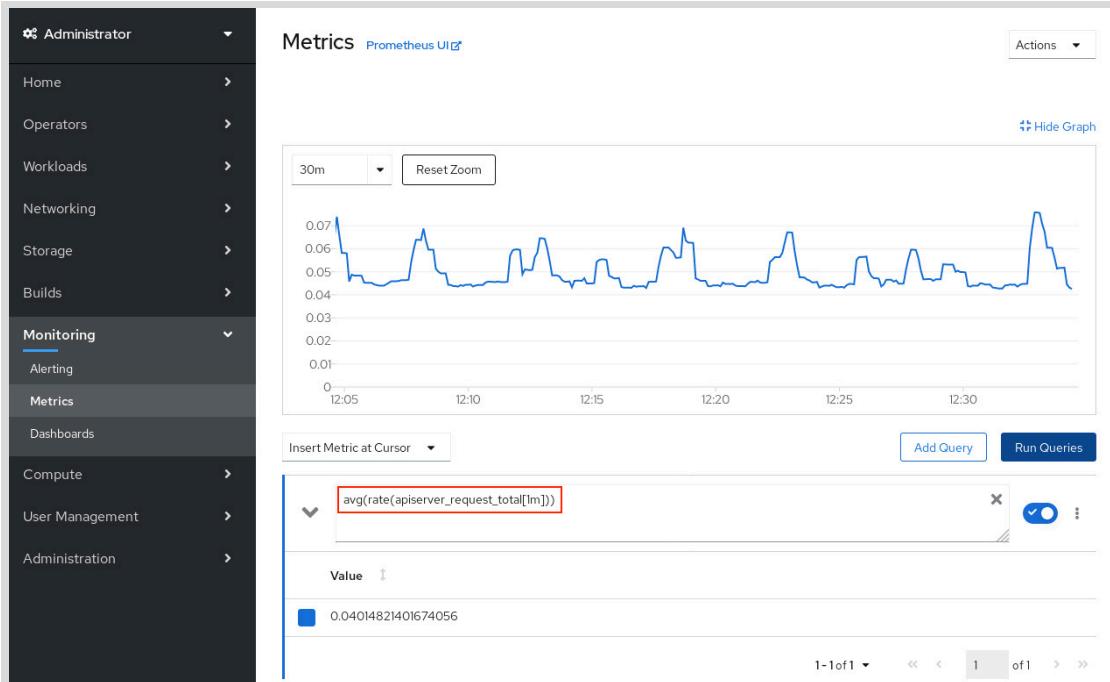


그림 15.20: 프로메테우스 쿼리를 사용하여 시계열 그래프를 표시합니다.

**참고**

이 과정에서는 프로메테우스 쿼리 언어에 대해 자세히 설명하지 않습니다. 공식 문서에 대한 링크는 아래 참조를 참조하십시오.

**참조**

자세한 내용은

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/monitoring/index

에 있는 Red Hat OpenShift Container Platform 4.5 모니터링 설명서를 참조하십시오.

프로메테우스 쿼리

<https://prometheus.io/docs/prometheus/latest/querying/basics/>

▶ 연습 가이드

클러스터 지표 검사

이 연습에서는 웹 콘솔 내에서 지표 페이지 및 대시보드를 검사합니다.

결과

Red Hat OpenShift 웹 콘솔을 사용하여 다음을 수행할 수 있어야 합니다.

- 클러스터, 프로젝트, 포드 및 노드 지표를 확인합니다.
- 많은 양의 메모리 또는 CPU를 소비하는 포드를 식별합니다.

시작하기 전에

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 이 연습에 필요한 리소스를 만듭니다.

```
[student@workstation ~]$ lab console-metrics start
```

▶ 1. **admin** 사용자로 이동하여 OpenShift 웹 콘솔을 찾아 이동합니다.

- 1.1. OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. 웹 콘솔의 URL을 확인합니다.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 1.3. 웹 브라우저를 열고 <https://console-openshift-console.apps.ocp4.example.com>으로 이동합니다.
- 1.4. Advanced(고급)를 클릭하여 신뢰할 수 없는 인증서 메시지를 표시한 다음 Add Exception(예외 추가)를 클릭합니다. Add Security Exception(보안 예외 추가) 대화 상자에서 Confirm Security Exception(보안 예외 확인)을 클릭합니다.
이 작업을 두 번 수행하여 **console-openshift-console** 및 **oauth-openshift** 하위 도메인의 자체 서명 SSL 인증서에 대한 경고를 건너뛰어야 합니다.
- 1.5. localusers(로컬 사용자)를 클릭하고 **redhat**을 암호로 사용하여 **admin** 사용자로 로그인합니다.

- ▶ 2. 이 단계별 연습에서는 부하의 변경 사항이 웹 콘솔에 표시되는 방식을 확인합니다. 먼저 개요, 포드 세부 정보 및 프로젝트 세부 정보 페이지에서 기준의 정상적인 지표를 관찰합니다.

- 2.1. Home(홈) → Overview(개요)를 클릭하여 Overview(개요) 페이지를 표시합니다. 아래로 스크롤하여 클러스터의 CPU, 메모리 및 디스크 사용량에 대한 시계열 기록 그래프가 있는 Cluster Utilization(클러스터 사용률) 섹션으로 이동합니다.
- 2.2. 테이블의 각 리소스(예: CPU, Memory(메모리) 또는 Filesystem(파일 시스템))에서 Usage(사용률) 열에 있는 링크를 클릭하여 해당 리소스의 Top Consumers(상위 소비자)를 확인합니다. 기본적으로 해당 창에는 상위 소비자가 프로젝트로 필터링되어 표시되지만, 포드 또는 노드로 필터링할 수도 있습니다.
- 2.3. Memory(메모리)의 사용량 링크를 클릭하고 상위 소비자를 포드로 필터링한 다음 메모리 리소스를 가장 많이 사용하는 포드의 이름을 클릭합니다.

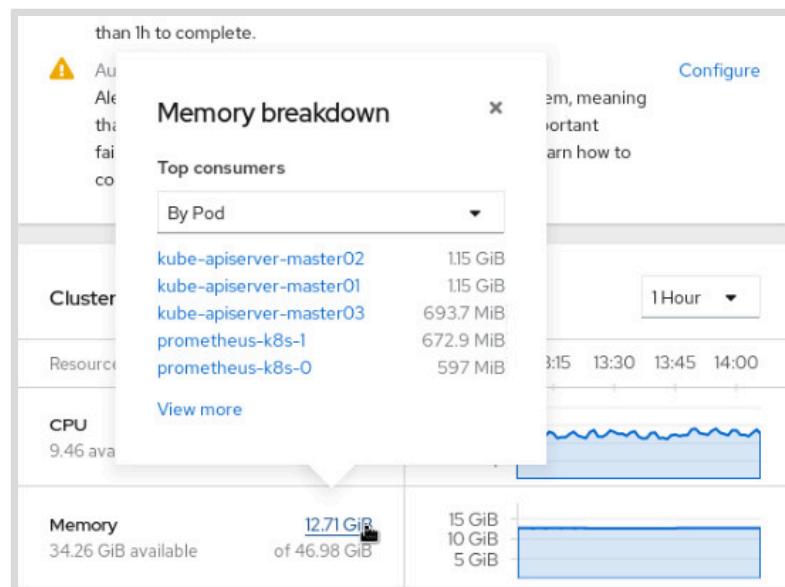


그림 15.21: 메모리 분석: 포드 상위 소비자

- 2.4. Pod Details(포드 세부 정보) 페이지의 페이지 상단에는 Memory Usage(메모리 사용량), CPU Usage(CPU 사용량) 및 Filesystem(파일 시스템) 시계열 기록 그래프가 표시됩니다.
 - 2.5. Home(홈) → Projects(프로젝트)를 클릭한 다음 console-apps를 클릭하여 console-apps Project Details(프로젝트 세부 정보) 페이지를 표시합니다. console-apps 프로젝트에서 실행되는 워크로드에 대한 지표가 표시되는 Utilization(사용률) 섹션을 확인합니다. Usage(사용량) 열의 링크를 클릭하면 창이 열리고 가장 많은 리소스를 사용하는 포드가 표시됩니다. 워크로드는 제한 내에서 안전하게 실행됩니다.
 - 2.6. Resource Quotas(리소스 할당량) 섹션까지 아래로 스크롤합니다. 현재 CPU 및 메모리 사용량이 할당된 할당량과 비교하여 표시됩니다.
- ▶ 3. 계산 노드의 기준 상태 지표를 찾아 검토합니다.

- 3.1. Compute(컴퓨팅) → Nodes(노드)를 클릭한 다음 목록에서 노드 중 하나를 클릭합니다.
- 3.2. Node Details(노드 세부 정보) 페이지에서 선택한 개별 노드에 대한 지표를 표시하는 시계열 그래프를 확인합니다.

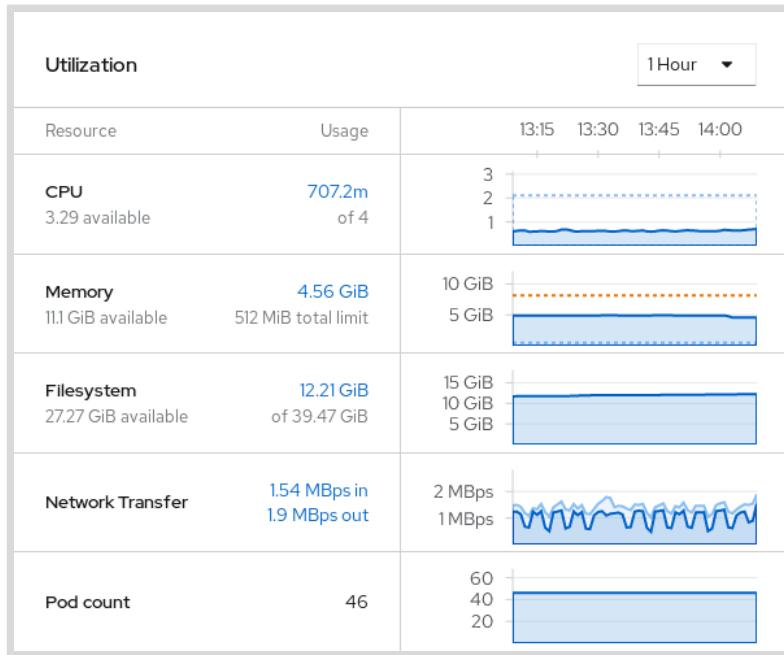


그림 15.22: 노드에 대한 다양한 지표를 보여주는 시계열 그래프.

- ▶ 4. workstation에서 `load.sh` 스크립트를 실행하여 예제 `books` 배포에서 부하를 생성합니다. 애플리케이션에는 `/leak` 경로에 대한 모든 요청에 대해 RAM(MB)을 사용하는 메모리 누수가 의도적으로 포함되어 있습니다.

- 4.1. `workstation` 시스템의 터미널에서 다음 명령을 실행합니다.

```
[student@workstation ~]$ ~/DO280/labs/console-metrics/load.sh
```

- ▶ 5. OpenShift 웹 콘솔에서 지표의 변화를 관찰하고 문제가 있는 포드를 식별합니다. 웹 콘솔에 표시된 데이터는 자동으로 새로 고쳐지므로 페이지를 다시 로드할 필요가 없습니다.

- 5.1. Home(홈) → Projects(프로젝트)를 클릭한 다음 `console-apps`를 클릭하여 `console-apps Project Details`(프로젝트 세부 정보) 페이지를 표시합니다. `Memory Usage`(메모리 사용량) 시계열 그래프를 보고 변경 사항을 모니터링합니다.

메모리 누수는 표시하기에 충분하므로 1~2분 정도 걸릴 수 있습니다. CPU 및 메모리가 모두 증가해도 총 CPU 사용량은 낮은 상태로 유지됩니다.

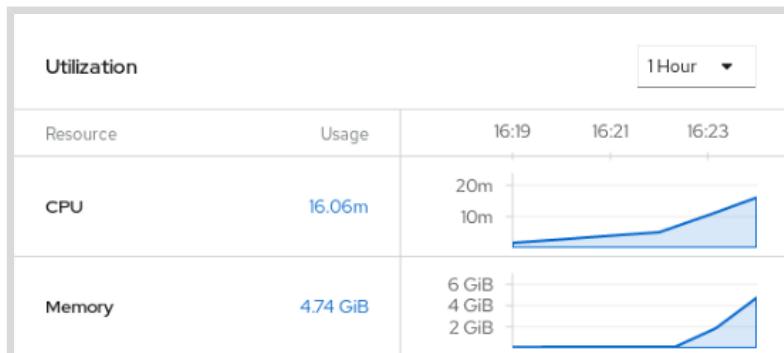


그림 15.23: 메모리 누수 가능성을 나타내는 사용률 그래프

- 5.2. Home(홈) → Overview(개요)를 클릭하여 Overview(개요) 페이지를 표시합니다. 부하 테스트에서 사용하는 메모리는 대규모 클러스터에서 확인하기에 너무 작을 수 있지만 **Memory breakdown(메모리 분류)** 창(포드를 기준으로 정렬)에서 가장 많은 메모리를 사용하는 편리한 포드 목록을 제공합니다. **Memory(메모리)**에 대한 사용량 링크를 클릭하여 **Memory breakdown(메모리 분류)** 창을 확인합니다. 포드를 기준으로 상위 소비자를 정렬합니다.

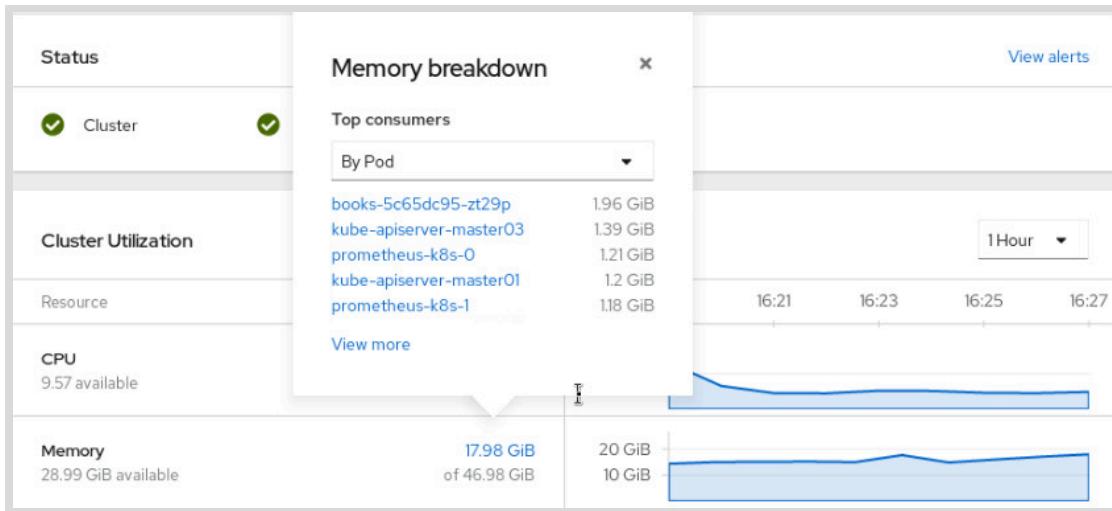
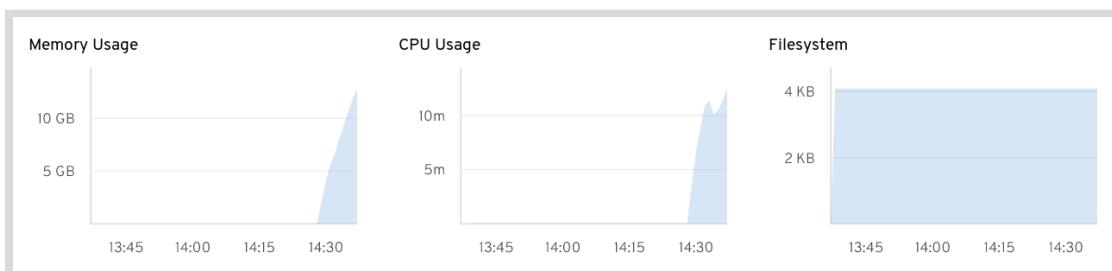


그림 15.24: books 포드는 상위 메모리 소비자입니다.

books 포드는 목록 맨 위 또는 맨 위 가까이에 표시됩니다. 목록에 없는 경우 부하 스크립트를 완료하는 데 몇 분 정도 더 기다려야 할 수 있습니다.

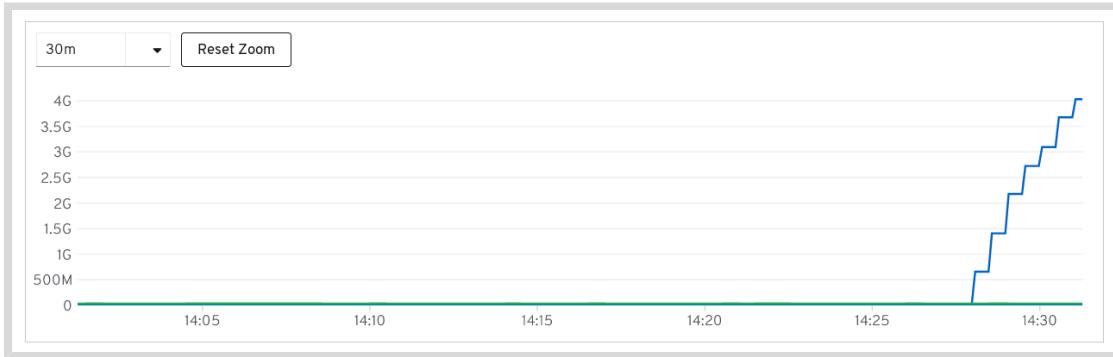
- 5.3. **Memory breakdown(메모리 분류)** 창에서 books 포드 링크를 클릭하여 **Pod Details(포드 세부 사항)** 페이지로 이동합니다. **Memory Usage(메모리 사용량)** 시계열 그래프에 증가 메모리 누수가 표시되는지 확인합니다.



- 5.4. Monitoring(모니터링) → Metrics(지표)를 클릭하여 웹 콘솔 Metrics(지표) 페이지를 표시합니다. 표현식 입력 필드에 다음 프로메테우스 쿼리를 입력합니다.

```
avg(container_memory_working_set_bytes{namespace='console-apps'}) BY (pod)
```

Run Queries(쿼리 실행)를 클릭하여 OpenShift 웹 콘솔에서 결과를 확인합니다.



▶ 6. console-apps 프로젝트를 삭제하고 부하 테스트를 중지합니다.

- 6.1. Home(홈) → Projects(프로젝트)를 클릭한 다음 console-apps 행의 끝에 있는 메뉴에서 Delete Project(프로젝트 삭제)를 클릭합니다.

| Project | Status | Owner | Labels | Action |
|-----------------|--------|--------------|-----------|---|
| console-apps | Active | admin | No labels | ⋮ |
| default | Active | No requester | No labels | Edit Project |
| kube-node-lease | Active | No requester | No labels | Delete Project |
| kube-public | Active | No requester | No labels | ⋮ |

- 6.2. Delete Project(프로젝트 삭제) 대화 상자에서 console-apps를 입력한 다음 Delete(삭제)를 클릭합니다.
- 6.3. load.sh가 여전히 workstation 터미널에서 실행 중인 경우 터미널에서 Ctrl+C를 눌러 부하 테스트를 중지합니다.

완료

workstation 시스템에서 lab 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab console-metrics finish
```

이것으로 섹션을 완료합니다.

▶ 랩

웹 콘솔로 클러스터 관리

이 랩에서는 웹 콘솔을 사용하여 OpenShift 클러스터를 관리합니다.

결과

OpenShift 웹 콘솔을 사용하여 다음을 수행할 수 있어야 합니다.

- 시크릿을 수정하여 새 사용자의 htpasswd 항목을 추가합니다.
- 역할 기반 액세스 제어 및 리소스 할당량을 사용하여 새 프로젝트를 구성합니다.
- OperatorHub 운영자를 사용하여 데이터베이스를 배포합니다.
- 웹 애플리케이션에 대한 배포, 서비스 및 경로를 만듭니다.
- 이벤트 및 로그를 사용하여 애플리케이션 문제 해결

시작하기 전에

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 이 연습 파일에 대한 디렉터리를 만듭니다.

```
[student@workstation ~]$ lab console-review start
```

- OpenShift 웹 콘솔에 **admin** 사용자로 로그인합니다.
- redhat**을 암호로 사용하여 **dba** 및 **tester**라는 사용자에 대한 **localusers** 시크릿에 **htpasswd** 항목을 추가합니다.
- developer** 및 **dba** 사용자를 포함하는 새 **app-team** 그룹을 만듭니다.
- tester** 사용자에게는 **view** 역할 바인딩을, **app-team** 그룹에는 **edit** 역할 바인딩을 사용하여 새 **console-review** 프로젝트를 만듭니다. 프로젝트를 4개의 포드로 제한하는 리소스 할당량을 설정합니다.
- console-review** 네임스페이스에서 사용할 인증된 버전의 Cockroach Operator를 설치합니다.
- dba** 사용자가 **openshift-operators** 프로젝트에서 리소스를 볼 수 있는 RoleBinding을 만듭니다.
- dba** 사용자로 OpenShift 웹 콘솔을 사용하여 CockroachDB 클러스터 데이터베이스 인스턴스를 **console-review** 프로젝트에 배포합니다. TLS를 비활성화하려면 **CrdbCluster** 리소스를 수정해야 합니다. YAML View(YAML 보기)에서 **tlsEnabled** 설정 값이 **false**인지 확인합니다.
- 다음 단계에서 문제 해결할 문제가 있는 **console-review** 프로젝트에서, **developer** 사용자로 배포, 서비스 및 경로를 만듭니다. **quay.io/redhattraining/exoplanets:v1.0** 이미지를 사용하고 새 리소스 **exoplanets**의 이름을 모두 지정합니다. 올바르게 구성된 경우 **exoplanets** 애플리케이션에서는 CockroachDB 클러스터에 연결하고 태양계의 외부에 있는 행성 목록을 표시합니다.

**참고**

workstation 시스템의 `~/DO280/labs/console-review/`에서 배포 및 서비스 YAML 리소스를 복사할 수 있습니다.

배포에서 다음 환경 변수를 지정합니다.

배포 환경 변수

| 이름 | 값 |
|---------|-----------|
| DB_HOST | localhost |
| DB_PORT | '26257' |
| DB_USER | root |
| DB_NAME | postgres |

**중요**

다음 단계에서 배포 관련 문제를 해결합니다.

9. 배포 문제를 해결하고 수정합니다.

10. 브라우저에서 exoplanets 웹 사이트로 이동하여 작동하는 애플리케이션을 관찰합니다.

평가

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab console-review grade
```

완료

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab console-review finish
```

이것으로 섹션을 완료합니다.

▶ 솔루션

웹 콘솔로 클러스터 관리

이 랩에서는 웹 콘솔을 사용하여 OpenShift 클러스터를 관리합니다.

결과

OpenShift 웹 콘솔을 사용하여 다음을 수행할 수 있어야 합니다.

- 시크릿을 수정하여 새 사용자의 htpasswd 항목을 추가합니다.
- 역할 기반 액세스 제어 및 리소스 할당량을 사용하여 새 프로젝트를 구성합니다.
- OperatorHub 운영자를 사용하여 데이터베이스를 배포합니다.
- 웹 애플리케이션에 대한 배포, 서비스 및 경로를 만듭니다.
- 이벤트 및 로그를 사용하여 애플리케이션 문제 해결

시작하기 전에

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 클러스터 API에 연결할 수 있도록 하며 이 연습 파일에 대한 디렉터리를 만듭니다.

```
[student@workstation ~]$ lab console-review start
```

- OpenShift 웹 콘솔에 **admin** 사용자로 로그인합니다.

- OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 웹 콘솔의 URL을 확인합니다.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 웹 브라우저를 열고 <https://console-openshift-console.apps.ocp4.example.com>으로 이동합니다.



참고

신뢰할 수 없는 인증서 메시지가 표시되면 **Add Exception(예외 추가)**를 클릭한 다음 **Confirm Security Exception(보안 예외 확인)**을 클릭합니다.

- 1.4. **localusers(로컬 사용자)**를 클릭하고 **redhat**을 암호로 사용하여 **admin** 사용자로 로그인합니다.
2. **redhat**을 암호로 사용하여 **dba** 및 **tester**라는 사용자에 대한 **localusers** 시크릿에 **htpasswd** 항목을 추가합니다.
 - 2.1. Red Hat OpenShift Container Platform 웹 UI에서 **Workloads(워크로드) → Secrets(시크릿)**를 클릭한 다음 **Project(프로젝트)** 검색 목록에서 **openshift-config**를 선택하여 **openshift-config** 프로젝트에 대한 시크릿을 표시합니다.
 - 2.2. 페이지 하단으로 스크롤한 다음 **localusers** 링크를 클릭하여 **localusers Secret Details(시크릿 세부 정보)**를 표시합니다.
 - 2.3. 페이지 상단에 있는 **Actions(작업) → Edit Secret(시크릿 편집)**을 클릭하여 **Edit Key/Value Secret(키/값 시크릿 편집)** 도구로 이동합니다.
 - 2.4. **workstation** 터미널을 사용하여 두 사용자 모두에 대해 암호화된 **htpasswd** 항목을 생성합니다.

```
[student@workstation ~]$ htpasswd -n -b dba redhat
dba:$apr1$YF4ack.9$qho0TH1wTC.cLByNEHDaV
[student@workstation ~]$ htpasswd -n -b tester redhat
tester:$apr1$XdTSqET7$i0hkC5bIs7PhYUm2KhiI.0
```

- 2.5. **htpasswd** 명령의 터미널 출력을 OpenShift 웹 콘솔의 시크릿 편집기에 있는 **htpasswd** 값에 추가하고 **Save(저장)**를 클릭합니다.
- ```
admin:$apr1$Au9.fFr$0k5wvUBd3eeBt0baa77.dae
leader:$apr1$/abo4Hybn7a.tG5ZoOBn.QwefXckiy1
developer:$apr1$RjqTY4cv$xql3.BQfg42moSxwnTNkh.
dba:$apr1$YF4ack.9$qho0TH1wTC.cLByNEHDaV
tester:$apr1$XdTSqET7$i0hkC5bIs7PhYUm2KhiI.0
```
3. **developer** 및 **dba** 사용자를 포함하는 새 **app-team** 그룹을 만듭니다.

- 3.1. **User Management(사용자 관리) → Groups(그룹)**를 클릭한 다음 **Create Group(그룹 만들기)**를 클릭합니다. YAML 편집기를 사용하여 다음과 같이 그룹 리소스를 정의합니다.

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
 name: app-team
users:
 - developer
 - dba
```

**Create(만들기)**를 클릭하여 새 **app-team** 그룹을 추가합니다.

4. **tester** 사용자에게는 **view** 역할 바인딩을, **app-team** 그룹에는 **edit** 역할 바인딩을 사용하여 새 **console-review** 프로젝트를 만듭니다. 프로젝트를 4개의 포드로 제한하는 리소스 할당량을 설정합니다.
  - 4.1. **Home(홈) → Projects(프로젝트)**를 클릭하여 **Projects(프로젝트)** 페이지를 표시한 다음 **Create Project(프로젝트 만들기)**를 클릭합니다. Name(이름) 필드에 **console-**

**review**를 입력한 다음 선택적 **Display Name(표시 이름)** 및 **Description(설명)**을 입력합니다. **Create(생성)**를 클릭합니다.

- 4.2. **Role Bindings(역할 바인딩)**를 클릭한 다음 **Create Binding(바인딩 만들기)**을 클릭합니다. 다음과 같이 양식을 작성하여 **app-team** 그룹에 대해 네임스페이스가 지정된 역할 바인딩을 만듭니다.

#### 애팀 역할 바인딩 양식

필드	값
이름	app-team
네임스페이스	console-review
역할 이름	edit
제목	그룹
제목 이름	app-team

**Create(만들기)**를 클릭하여 네임스페이스 지정된 RoleBinding을 만듭니다.

- 4.3. **Role Bindings(역할 바인딩)** 링크를 클릭하여 **Role Bindings(역할 바인딩)** 페이지로 돌아간 다음 **Create Binding(바인딩 만들기)**를 클릭합니다. 다음과 같이 양식을 작성하여 **tester** 사용자에 대해 네임스페이스가 지정된 역할 바인딩을 만듭니다.

#### 테스터 역할 바인딩 양식

필드	값
바인딩 유형	Namespace Role Binding (RoleBinding)
이름	tester
네임스페이스	console-review
역할 이름	view
제목	User
제목 이름	tester

**Create(만들기)**를 클릭하여 네임스페이스 지정된 RoleBinding을 만듭니다.

- 4.4. **Administration(관리) → Resource Quotas(리소스 할당량)**를 클릭한 다음 **Create Resource Quota(리소스 할당량 만들기)**를 클릭합니다. YAML 문서를 수정하여 다음과 같이 4개의 포드 제한을 지정합니다.

```

apiVersion: v1
kind: ResourceQuota
metadata:
 name: quota
 namespace: console-review
spec:
 hard:
 pods: '4'

```

CPU 및 메모리 요청 및 제한을 제거한 다음 **Create(만들기)**를 클릭합니다.

5. **console-review** 네임스페이스에서 사용할 인증된 버전의 Cockroach Operator를 설치합니다.
  - 5.1. Operators(운영자) → OperatorHub를 클릭한 다음 Database(데이터베이스)를 클릭하여 OperatorHub에서 사용할 수 있는 데이터베이스 운영자 목록을 표시합니다.
  - 5.2. Certified(인증됨) 상태의 운영자만 표시하도록 Provider Type(프로바이더 유형)을 필터링한 다음 Cockroach Operator를 클릭합니다.
  - 5.3. Cockroach Operator 페이지에서 Install(설치)를 클릭합니다. Install Operator(설치 운영자) 페이지의 Installed Namespace(설치된 네임스페이스)에서 console-review를 선택하고 Install(설치)를 클릭합니다.
6. dba 사용자가 openshift-operators 프로젝트에서 리소스를 볼 수 있는 RoleBinding을 만듭니다.
  - 6.1. User Management(사용자 관리) → Role Bindings(역할 바인딩)를 클릭한 다음 Create Binding(바인딩 만들기)을 클릭합니다. 다음과 같이 양식을 작성합니다.

#### DBA OpenShift-Operator 역할 바인딩 양식

필드	값
바인딩 유형	Namespace Role Binding (RoleBinding)
이름	dba
네임스페이스	openshift-operators
역할 이름	view
제목	User
제목 이름	dba

Create(만들기)를 클릭하여 네임스페이스가 지정된 RoleBinding을 추가합니다.

7. dba 사용자로 OpenShift 웹 콘솔을 사용하여 CockroachDB 클러스터 데이터베이스 인스턴스를 **console-review** 프로젝트에 배포합니다. TLS를 비활성화하려면 **CrdbCluster** 리소스를 수정해야 합니다. YAML View(YAML 보기)에서 **tlsEnabled** 설정 값이 false인지 확인합니다.
  - 7.1. admin → Log out(로그아웃)을 클릭한 다음 redhat을 암호로 사용하여 dba 사용자로 로그인합니다.
  - 7.2. Home(홈) → Projects(프로젝트)를 클릭하고 **console-review** 프로젝트 링크를 클릭하여 **console-review** 프로젝트로 전환합니다.

- 7.3. Operators(운영자) → Installed Operators(설치된 운영자)를 클릭한 다음 Cockroach Operator 이름을 클릭합니다.
- 7.4. Operator Details(운영자 세부 정보) 페이지에서 Create Instance(인스턴스 생성)를 클릭합니다. YAML View(YAML 보기)를 사용하여 CrdbCluster 리소스를 구성합니다. 이름 crdb-example를 사용하여 tlsEnabled 설정 값이 false인지 확인합니다. Create(생성)를 클릭하여 CrdbCluster 리소스를 생성합니다.

```
apiVersion: crdb.cockroachlabs.com/v1alpha1
kind: CrdbCluster
metadata:
 name: crdb-example
 namespace: console-review
spec:
 tlsEnabled: false
 nodes: 3
 dataStore:
 pvc:
 spec:
 accessModes:
 - ReadWriteOnce
 resources:
 requests:
 storage: 10Gi
 volumeMode: Filesystem
```

8. 다음 단계에서 문제 해결할 문제가 있는 console-review 프로젝트에서 developer 사용자로 배포, 서비스 및 경로를 만듭니다. quay.io/redhattraining/exoplanets:v1.0 이미지를 사용하고 새 리소스 exoplanets의 이름을 모두 지정합니다. 올바르게 구성된 경우 exoplanets 애플리케이션에서는 CockroachDB 클러스터에 연결하고 태양계의 외부에 있는 행성 목록을 표시합니다.



### 참고

workstation 시스템의 ~/D0280/labs/console-review/에서 배포 및 서비스 YAML 리소스를 복사할 수 있습니다.

배포에서 다음 환경 변수를 지정합니다.

### 배포 환경 변수

이름	값
DB_HOST	localhost
DB_PORT	'26257'
DB_USER	root
DB_NAME	postgres

**중요**

다음 단계에서 배포 관련 문제를 해결합니다.

- 8.1. dba → Log out(로그아웃)을 클릭한 다음 developer를 암호로 사용하여 developer 사용자로 로그인합니다.
- 8.2. Home(홈) → Projects(프로젝트)를 클릭한 다음 console-review 프로젝트를 클릭하여 console-review 프로젝트로 전환합니다.
- 8.3. Workloads(워크로드) → Deployments(배포)를 클릭한 다음 Create Deployment(배포 만들기)를 클릭하여 웹 콘솔 YAML 편집기를 표시합니다. 다음과 같이 YAML을 업데이트한 다음 Create(만들기)를 클릭합니다.

```
kind: Deployment
apiVersion: apps/v1
metadata:
 name: exoplanets
 namespace: console-review
spec:
 replicas: 1
 selector:
 matchLabels:
 app: exoplanets
 template:
 metadata:
 labels:
 app: exoplanets
 spec:
 containers:
 - name: exoplanets
 image: 'quay.io/redhattraining/exoplanets:v1.0'
 ports:
 - containerPort: 8080
 protocol: TCP
 readinessProbe:
 httpGet:
 path: /healthz
 port: 8080
 env:
 - name: DB_HOST
 value: localhost
 - name: DB_PORT
 value: '26257'
 - name: DB_USER
 value: root
 - name: DB_NAME
 value: postgres
```

- 8.4. Networking(네트워킹) → Services(서비스)를 클릭한 다음 Create Service(서비스 만들기)를 클릭하여 웹 콘솔 YAML 편집기를 표시합니다. 다음과 같이 YAML을 업데이트한 다음 Create(만들기)를 클릭합니다.

```

kind: Service
apiVersion: v1
metadata:
 name: exoplanets
 namespace: console-review
spec:
 selector:
 app: exoplanets
 ports:
 - protocol: TCP
 port: 8080
 targetPort: 8080

```

- 8.5. Networking(네트워킹) → Routes(경로)를 클릭한 다음 Create Route(경로 만들기)를 클릭합니다. 다른 필드는 변경하지 않고 그대로 두고 양식을 다음과 같이 완성한 다음 Create(만들기)를 클릭합니다.

### 경로 양식 만들기

필드	값
이름	exoplanets
서비스	exoplanets
대상 포트	8080 → 8080 (TCP)

9. 배포 문제를 해결하고 수정합니다.

- 9.1. developer → Log out(로그아웃)을 클릭한 다음 redhat을 암호로 사용하여 admin 사용자로 로그인합니다.
- 9.2. Home(홈) → Events(이벤트)를 클릭한 다음 상단의 프로젝트 목록 필터에서 console-review를 선택합니다. exoplanets 할당량 오류를 확인합니다.

```
(combined from similar events): Error creating: pods "exoplanets-5f88574546-lsnmx" is forbidden: exceeded quota: quota, requested: pods=1, used: pods=4, limited: pods=4
```

- 9.3. Administration(관리) → Resource Quotas(리소스 할당량)를 클릭한 다음 Project(프로젝트) 필터 목록에서 console-review를 선택합니다.
- 9.4. 리소스 할당량 목록에서 quota(할당량) 링크를 클릭한 다음 YAML 탭을 클릭합니다. spec(사양)을 수정하여 다음과 같이 포드 제한을 6개로 지정한 다음 Save(저장)를 클릭합니다.

```

kind: ResourceQuota
apiVersion: v1
metadata:
 name: quota
 namespace: console-review
...output omitted...

```

```
spec:
 hard:
 pods: '6'
...output omitted...
```

**참고**

프로젝트에는 변경 사항을 둘아웃하기 위해 exoplanet의 지정된 복제본 및 추가 포드에 대한 포드가 필요합니다.

- 9.5. Workloads(워크로드) → Pods(포드)를 클릭하고 포드 목록을 검토합니다. **exoplanets** 포드는 목록에 표시되는 데 1~2분이 걸릴 수 있으며 CrashLoopBackOff 상태를 표시합니다.

- 9.6. 포드 이름을 클릭한 다음 Logs(로그) 탭을 클릭합니다. 연결 오류를 확인합니다.

```
2020/09/25 13:33:25 Connecting to database with: host=localhost port=26257
user=root password= dbname=postgres sslmode=disable
2020/09/25 13:33:25 dial tcp [::1]:26257: connect: connection refused
```

- 9.7. Networking(네트워킹) → Services(서비스)를 클릭하여 **console-review** 네임스페이스에서 서비스를 확인합니다. Cockroach 데이터베이스와 관련된 두 가지 서비스가 있습니다. - **public** 접미사가 있는 서비스 이름은 IP 주소가 있는 유일한 Cockroach 서비스이며, 해당 서비스는 포트 26257을 사용합니다.

**exoplanets** 포드는 데이터베이스 서버의 포트 26257 대신 localhost의 포트 26257에 연결을 시도합니다. - **public** 접미사가 있는 서비스 이름을 클립보드에 복사합니다.

- 9.8. Workloads(워크로드) → Deployments(배포)를 클릭한 다음 **exoplanets** 배포 링크를 클릭합니다. Environment(환경) 탭을 클릭하고 **localhost**의 DB\_HOST 값을 앞에서 복사한 Cockroach 서비스 이름으로 변경합니다. Save(저장)를 클릭합니다.

- 9.9. Pods(포드) 탭을 클릭하여 새 exoplanets 포드가 **Running**(실행 중) 상태로 업데이트되는지 확인합니다.

10. 브라우저에서 exoplanets 웹 사이트로 이동하여 작동하는 애플리케이션을 관찰합니다.

- 10.1. Networking(네트워킹) → Routes(경로)를 클릭하고 **exoplanets** 경로 이름을 클릭한 다음 Location(위치) 열에서 링크를 클릭합니다. Firefox에서는 exoplanets 테이블을 렌더링하는 새 탭을 엽니다.

**평가**

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab console-review grade
```

**완료**

workstation 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다. 이전 연습 문제의 리소스가 다음 연습 문제에 영향을 미치지 않도록 하는 것이 중요합니다.

```
[student@workstation ~]$ lab console-review finish
```

이것으로 섹션을 완료합니다.

# 요약

이 장에서 학습한 내용:

- OpenShift 웹 콘솔은 OpenShift 리소스를 시각화하고 관리하는 GUI를 제공합니다.
- 일부 리소스는 Base64 인코딩 및 디코딩을 자동으로 처리하는 **Edit Key/Value Secret(키/값 시크릿 편집)** 편집기와 같이 수동으로 리소스를 만들고 편집하는 것보다 편리하게 리소스를 만들고 편집하는 데 사용할 수 있는 특수 페이지를 특징으로 합니다.
- 포함된 **OperatorHub** 페이지에서 파트너 및 커뮤니티 운영자를 설치할 수 있습니다.
- CPU, 메모리 및 스토리지 사용량과 같은 클러스터 전반의 지표는 **Dashboards(대시보드)** 페이지에 표시됩니다.
- **Project Details(프로젝트 세부 정보)** 페이지에는 프로젝트와 관련된 지표(예: 포드별 상위 10개의 메모리 소비자 및 현재 리소스 할당량 사용)가 표시됩니다.

## 종합 검토

### 목적

다음에서 작업 검토 Containers, Kubernetes, and Red Hat OpenShift Administration II

### 목표

- 다음에서 작업 검토 Containers, Kubernetes, and Red Hat OpenShift Administration II

### 섹션

- 종합 검토

### 랩

- OpenShift 클러스터 및 애플리케이션의 문제 해결
- 리소스 및 네트워크 제한 사항을 사용하여 프로젝트 템플릿 구성

# 종합 검토

## 목표

이 섹션을 마치면 Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster에서 배운 지식과 기술을 검토하고 다시 기억할 수 있습니다.

## Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster 검토

이 교육 과정을 포괄적으로 검토하기 전에 각 장에서 다룬 주제를 충분히 이해하고 있어야 합니다.

추가 학습을 위해 교과서의 앞 섹션을 참조할 수 있습니다.

## Red Hat OpenShift Container Platform 설명

OpenShift Container Platform의 아키텍처를 설명합니다.

- 제품의 일반 사용법과 해당 기능을 설명합니다.
- Red Hat OpenShift Container Platform의 아키텍처를 설명합니다.
- 클러스터 운영자의 정의와 작동 방식을 설명하고 주요 클러스터 운영자의 이름을 밝힙니다.

## 클러스터의 상태 확인

OpenShift 설치 방법을 설명하고 새로 설치된 클러스터의 상태를 확인합니다.

- OpenShift 설치 프로세스, 풀스택 자동화, 기존 인프라 설치 방법을 설명합니다.
- 일반적인 문제 해결을 지원하는 명령을 실행합니다.
- 영구저장장치의 구성 요소 및 리소스를 확인하고, 영구 볼륨 클레임을 사용하는 애플리케이션을 배포합니다.

## 인증 및 권한 부여 구성

HTPasswd Identity 프로바이더를 사용하여 인증을 구성하고 사용자 및 그룹에 역할을 할당합니다.

- OpenShift 인증에 사용할 HTPasswd ID 프로바이더를 구성합니다.
- 역할 기반 액세스 제어를 정의하고 사용자 및 그룹에 권한을 적용합니다.

## 애플리케이션 보안 구성

보안 컨텍스트 제약 조건을 사용하여 애플리케이션 권한을 제한하고 시크릿을 사용하여 액세스 자격 증명을 보호합니다.

- 중요한 정보를 관리하기 위한 시크릿을 생성하고 적용합니다.
- 서비스 계정을 생성하고 권한을 적용합니다.

## 애플리케이션을 위한 OpenShift 네트워킹 구성

OpenShift SDN(소프트웨어 정의 네트워킹) 문제를 문제 해결하고 네트워크 정책을 구성합니다.

- 명령줄 인터페이스를 사용하여 OpenShift 소프트웨어 정의 네트워킹의 문제를 해결합니다.
- OpenShift 클러스터 내부의 애플리케이션에 대한 네트워크 연결을 허용하고 보호합니다.
- 프로젝트와 포드 간의 네트워크 트래픽을 제한합니다.

## 포드 스케줄링 제어

포드에서 실행하는 노드를 제어합니다.

- 포드 스케줄링 알고리즘, 스케줄링에 영향을 주는 메서드를 설명하고 이러한 메서드를 적용하는 방법에 대해 설명합니다.
- 컨테이너, 포드, 프로젝트에서 사용하는 리소스를 제한합니다.
- 포드의 복제 수를 제어합니다.

## 클러스터 업데이트 설명

클러스터 업데이트를 수행하는 방법에 대해 설명합니다.

클러스터 업데이트 프로세스에 대해 설명합니다.

## 웹 콘솔로 클러스터 관리

웹 콘솔을 사용하여 Red Hat OpenShift 클러스터를 관리합니다.

- 웹 콘솔을 사용하여 클러스터 관리를 수행합니다.
- 웹 콘솔을 사용하여 애플리케이션 및 Kubernetes 운영자를 관리합니다.
- 클러스터 노드와 애플리케이션의 성능 및 상태 지표를 검사합니다.

## ▶ 랩

# OpenShift 클러스터 및 애플리케이션의 문제 해결

이 검토에서는 개발자가 클러스터에 액세스하여 애플리케이션 배포 문제를 문제 해결할 수 있도록 허용합니다.

## 결과

다음을 수행할 수 있습니다.

- 새 프로젝트를 만듭니다.
- Source-to-Image 프로세스를 통해 애플리케이션을 생성하여 OpenShift 클러스터의 스모크 테스트를 수행합니다.
- deployment** 또는 **deploymentconfig** 리소스를 사용하여 애플리케이션을 만듭니다.
- 사용자 관리를 위해 HTPasswd ID 프로바이더를 사용합니다.
- 그룹을 만들고 관리합니다.
- 사용자와 그룹에 대해 RBAC 및 SCC를 관리합니다.
- 데이터베이스 및 애플리케이션에 대한 시크릿을 관리합니다.
- 일반 문제를 해결합니다.

## 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 다음을 수행합니다.

- 클러스터 API에 연결할 수 있는지 확인합니다.
- 기존 사용자 및 그룹을 제거합니다.
- 기존의 ID 프로바이더를 제거합니다.
- admin** 사용자의 **cluster-admin** 클러스터 역할 바인딩을 제거합니다.
- 인증된 사용자가 새 프로젝트를 만들 수 있는지 확인합니다.

```
[student@workstation ~]$ lab review-troubleshoot start
```

## 지침

다음 작업을 완료합니다.

- kubeadmin** 사용자로 **review-troubleshoot** 프로젝트를 생성합니다. **kubeadmin** 사용자의 암호는 **RHT\_OCP4\_KUBEADM\_PASSWD** 행의 **/usr/local/etc/ocp4.config** 파일에 있습니다. **review-troubleshoot** 프로젝트의 모든 후속 작업을 수행합니다.

2. 클러스터의 스모크 테스트를 수행하여 기본 클러스터 기능을 확인합니다. **deployment configuration**을 사용하여 **hello-world-nginx**라는 애플리케이션을 만듭니다. 애플리케이션 소스 코드는 <https://github.com/RedHatTraining/D0280-apps> 리포지토리의 **hello-world-nginx** 하위 디렉터리에 있습니다.  
**apps.ocp4.example.com** 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 애플리케이션에 대한 경로를 만든 다음 애플리케이션이 외부 요청에 응답하는지 확인합니다.
3. HTPasswd ID 프로바이더를 사용하도록 클러스터를 구성합니다. ID 프로바이더의 이름은 **cluster-users**입니다. ID 프로바이더는 **compreview-users** 시크릿에서 **htpasswd** 자격 증명을 읽습니다.  
**admin, leader, developer, qa-engineer**라는 네 가지 사용자 계정이 존재하는지 확인합니다. 모든 사용자 계정은 암호로 **review**를 사용해야 합니다.  
**cluster-admin** 역할을 **admin** 사용자에게 추가합니다.
4. **admin** 사용자가 세 개의 사용자 그룹(**leaders, developers, qa**)을 만듭니다.  
**leader** 사용자를 **leaders** 그룹에 할당하고 **developer** 사용자를 **developers** 그룹에 할당하고 **qa-engineer** 사용자를 **qa** 그룹에 할당합니다.  
각 그룹에 역할을 할당합니다.
  - **self-provisioner** 역할을 **leaders** 그룹에 할당하여 멤버가 이 프로젝트를 만들 수 있도록 합니다. 이 역할을 적용하려면 인증된 사용자의 기능을 제거하여 새 프로젝트를 만들어야 합니다.
  - **review-troubleshoot** 프로젝트에 대해 **developers** 그룹에 **edit** 역할을 할당합니다. 이 역할의 멤버는 프로젝트 리소스를 만들고 삭제할 수 있습니다.
  - **review-troubleshoot** 프로젝트에 대해 **qa** 그룹에 **view** 역할을 할당합니다. 이 역할의 멤버에게는 프로젝트 리소스에 대한 읽기 액세스 권한이 제공됩니다.
5. **developer** 사용자로 **deployment**을 사용하여 **review-troubleshoot** 프로젝트에 **mysql**이라는 애플리케이션을 만듭니다. **registry.access.redhat.com/rhsc1/mysql-57-rhel7:5.7**에서 사용할 수 있는 컨테이너 이미지를 사용합니다. 이 애플리케이션에서는 다른 프로젝트 애플리케이션에 대한 공유 데이터베이스 서비스를 제공합니다.  
**password**를 키로 사용하고 **r3dh4t123**을 값으로 사용하여 **mysql**이라는 일반 시크릿을 생성합니다.  
**MYSQL\_ROOT\_PASSWORD** 환경 변수를 **mysql** 시크릿의 **password** 키 값으로 설정합니다. PVC(영구 볼륨 클레임)를 포드 내의 **/var/lib/mysql/data** 디렉터리에 마운트하도록 **mysql** 데이터베이스 애플리케이션을 구성합니다. PVC는 크기가 **2GB**여야 하며, **ReadWriteOnce** 액세스 모드만 요청해야 합니다.
6. **developer** 사용자로 **deployment**을 사용하여 **wordpress**라는 애플리케이션을 만듭니다. **review-troubleshoot** 프로젝트에 애플리케이션을 생성합니다. **quay.io/**

`redhattraining/wordpress:5.3.0`에서 사용할 수 있는 이미지를 사용합니다. 이 이미지는 `docker.io/library/wordpress:5.3.0`에서 사용할 수 있는 컨테이너 이미지의 사본입니다.

`WORDPRESS_DB_HOST` 환경 변수가 `mysql` 값을 가지도록 구성합니다. 애플리케이션에서는 이 변수를 사용하여 `mysql` 애플리케이션에서 제공하는 `mysql` 데이터베이스 서비스를 연결합니다.

`WORDPRESS_DB_NAME` 환경 변수가 `wordpress` 값을 가지도록 구성합니다. 애플리케이션에서는 이 변수를 사용하여 데이터베이스의 이름을 식별합니다. 데이터베이스가 데이터베이스 서버에 없는 경우에는 애플리케이션에서 이 이름으로 새 데이터베이스를 만들려고 시도합니다.

`WORDPRESS_DB_USER` 환경 변수를 `root` 값으로 설정합니다. `WORDPRESS_DB_PASSWORD` 환경 변수를 `mysql` 시크릿의 `password` 키 값으로 설정합니다. `WORDPRESS_DB_PASSWORD` 환경 변수의 값은 `mysql` 루트 사용자 암호와 같아야 합니다.

`wordpress` 애플리케이션에는 `anyuid` 보안 컨텍스트 제약도 필요합니다. `wordpress-sa`라는 서비스 계정을 만든 다음 `anyuid` 보안 컨텍스트 제약 조건을 할당합니다. `wordpress-sa` 서비스 계정을 사용하도록 `wordpress` 배포를 구성합니다.

`apps.ocp4.example.com` 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 애플리케이션에 대한 경로를 만듭니다. 애플리케이션을 올바르게 배포하면 브라우저에서 애플리케이션에 액세스할 때 설치 마법사가 표시됩니다.

- developer 사용자로 `~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh` 스크립트를 사용하여 `review-troubleshoot` 프로젝트에 `famous-quotes` 애플리케이션을 배포합니다. 이 스크립트는 `defaultdb` 데이터베이스와 `~/D0280/labs/review-troubleshoot/famous-quotes.yaml` 파일에 정의된 리소스를 만듭니다.

스크립트를 실행한 후에는 초기에 애플리케이션 포드가 배포되지 않습니다. `famous-quotes` 배포 구성에 노드 선택기가 지정되며 일치하는 노드 레이블이 있는 클러스터 노드가 없습니다.

배포 구성에서 노드 선택기를 제거하여 OpenShift가 사용 가능한 노드에서 애플리케이션 포드를 예약 할 수 있도록 합니다.

`apps.ocp4.example.com` 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 `famous-quotes` 애플리케이션에 대한 경로를 만든 다음 애플리케이션이 외부 요청에 응답하는지 확인합니다.

## 평가

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab review-troubleshoot grade
```

## 완료

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab review-troubleshoot finish
```

이로써 종합 검토가 완료됩니다.

## ▶ 솔루션

# OpenShift 클러스터 및 애플리케이션의 문제 해결

이 검토에서는 개발자가 클러스터에 액세스하여 애플리케이션 배포 문제를 문제 해결할 수 있도록 허용합니다.

### 결과

다음을 수행할 수 있습니다.

- 새 프로젝트를 만듭니다.
- Source-to-Image 프로세스를 통해 애플리케이션을 생성하여 OpenShift 클러스터의 스모크 테스트를 수행합니다.
- `deployment` 또는 `deploymentconfig` 리소스를 사용하여 애플리케이션을 만듭니다.
- 사용자 관리를 위해 HTPasswd ID 프로바이더를 사용합니다.
- 그룹을 만들고 관리합니다.
- 사용자와 그룹에 대해 RBAC 및 SCC를 관리합니다.
- 데이터베이스 및 애플리케이션에 대한 시크릿을 관리합니다.
- 일반 문제를 해결합니다.

### 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 다음을 수행합니다.

- 클러스터 API에 연결할 수 있는지 확인합니다.
- 기존 사용자 및 그룹을 제거합니다.
- 기존의 ID 프로바이더를 제거합니다.
- `admin` 사용자의 `cluster-admin` 클러스터 역할 바인딩을 제거합니다.
- 인증된 사용자가 새 프로젝트를 만들 수 있는지 확인합니다.

```
[student@workstation ~]$ lab review-troubleshoot start
```

### 지침

다음 작업을 완료합니다.

1. kubeadmin 사용자로 review-troubleshoot 프로젝트를 생성합니다. kubeadmin 사용자의 암호는 RHT\_OCP4\_KUBEADM\_PASSWD 행의 /usr/local/etc/ocp4.config 파일에 있습니다. review-troubleshoot 프로젝트의 모든 후속 작업을 수행합니다.

- 1.1. /usr/local/etc/ocp4.config에서 액세스할 수 있는 강의실 구성 파일을 찾고 kubeadmin 사용자로 로그인합니다.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. review-troubleshoot 프로젝트를 생성합니다.

```
[student@workstation ~]$ oc new-project review-troubleshoot
Now using project "review-troubleshoot" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

2. 클러스터의 스모크 테스트를 수행하여 기본 클러스터 기능을 확인합니다. deployment configuration을 사용하여 hello-world-nginx라는 애플리케이션을 만듭니다. 애플리케이션 소스 코드는 <https://github.com/RedHatTraining/D0280-apps> 리포지토리의 hello-world-nginx 하위 디렉터리에 있습니다.

- apps.ocp4.example.com 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 애플리케이션에 대한 경로를 만든 다음 애플리케이션이 외부 요청에 응답하는지 확인합니다.

- 2.1. oc new-app 명령을 사용하여 hello-world-nginx 배포 구성을 생성합니다. 배포 구성을 사용하여 애플리케이션을 생성하려면 --as-deployment-config 옵션을 사용해야 합니다.

```
[student@workstation ~]$ oc new-app --name hello-world-nginx \
> --as-deployment-config \
> https://github.com/RedHatTraining/D0280-apps \
> --context-dir hello-world-nginx
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "ubi8" created
imagestream.image.openshift.io "hello-world-nginx" created
buildconfig.build.openshift.io "hello-world-nginx" created
deploymentconfig.apps.openshift.io "hello-world-nginx" created
service "hello-world-nginx" created
--> Success
...output omitted...
```

- 2.2. hello-world-nginx 서비스를 노출하여 애플리케이션 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service hello-world-nginx \
> --hostname hello-world.apps.ocp4.example.com
route.route.openshift.io/hello-world-nginx exposed
```

- 2.3. 애플리케이션 포드가 실행될 때까지 기다립니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-world-nginx-1-build 0/1 Completed 0 2m59s
hello-world-nginx-1-deploy 0/1 Completed 0 108s
hello-world-nginx-1-m2lzs 1/1 Running 0 100s
```

2.4. 애플리케이션에 대한 액세스 권한을 확인합니다.

```
[student@workstation ~]$ curl -s http://hello-world.apps.ocp4.example.com \
> | grep Hello
<h1>Hello, world from nginx!</h1>
```

3. HTPasswd ID 프로바이더를 사용하도록 클러스터를 구성합니다. ID 프로바이더의 이름은 **cluster-users**입니다. ID 프로바이더는 **compreview-users** 시크릿에서 **htpasswd** 자격 증명을 읽습니다.

**admin, leader, developer, qa-engineer**라는 네 가지 사용자 계정이 존재하는지 확인합니다. 모든 사용자 계정은 암호로 **review**를 사용해야 합니다.

**cluster-admin** 역할을 **admin** 사용자에게 추가합니다.

- 3.1. **/tmp/cluster-users**에서 임시 **htpasswd** 인증 파일을 만듭니다.

```
[student@workstation ~]$ touch /tmp/cluster-users
```

- 3.2. 필요한 사용자 및 암호 값을 사용하여 **/tmp/cluster-users** 파일을 채웁니다.

```
[student@workstation ~]$ for user in admin leader developer qa-engineer
> do
> htpasswd -B -b /tmp/cluster-users ${user} review
> done
Adding password for user admin
Adding password for user leader
Adding password for user developer
Adding password for user qa-engineer
```

- 3.3. **/tmp/cluster-users** 파일에서 **compreview-users** 시크릿을 생성합니다.

```
[student@workstation ~]$ oc create secret generic compreview-users \
> --from-file htpasswd=/tmp/cluster-users -n openshift-config
secret/compreview-users created
```

- 3.4. 기존 OAuth 리소스를 YAML 파일로 내보냅니다.

```
[student@workstation ~]$ oc get oauth cluster -o yaml > /tmp/oauth.yaml
```

- 3.5. **/tmp/oauth.yaml** 파일을 편집하여 HTPasswd ID 프로바이더 정의를 **identityProviders** 목록에 추가합니다. ID 프로바이더 이름을 **cluster-users**로 설정하고 **fileData** 이름을 **compreview-users**로 설정합니다.

이렇게 수정하면 파일이 다음과 같이 표시됩니다.

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
...output omitted...
 name: cluster
...output omitted...
spec:
 identityProviders:
 - name: cluster-users
 mappingMethod: claim
 type: HTPasswd
 htpasswd:
 fileData:
 name: comprevew-users
```

**참고**

`name`, `mappingMethod`, `type` 및 `htpasswd` 키는 모두 동일한 들여쓰기를 사용합니다.

- 3.6. 기존 OAuth 리소스를 수정된 파일의 리소스 정의로 바꿉니다.

```
[student@workstation ~]$ oc replace -f /tmp/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

- 3.7. `admin` 사용자에게 `cluster-admin` 역할을 할당합니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
> cluster-admin admin
Warning: User 'admin' not found
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

**중요**

확인되지 않은 `admin` 사용자에 대한 경고는 무시해도 됩니다. `user/admin` 리소스는 `admin` 사용자가 처음 로그하기 전에는 OpenShift 클러스터에 존재하지 않습니다.

4. `admin` 사용자가 세 개의 사용자 그룹(`leaders`, `developers`, `qa`)을 만듭니다.

`leader` 사용자를 `leaders` 그룹에 할당하고 `developer` 사용자를 `developers` 그룹에 할당하고 `qa-engineer` 사용자를 `qa` 그룹에 할당합니다.

각 그룹에 역할을 할당합니다.

- `self-provisioner` 역할을 `leaders` 그룹에 할당하여 멤버가 이 프로젝트를 만들 수 있도록 합니다. 이 역할을 적용하려면 인증된 사용자의 기능을 제거하여 새 프로젝트를 만들어야 합니다.
- `review-troubleshoot` 프로젝트에 한해 `developers` 그룹에 `edit` 역할을 할당합니다. 이 역할의 멤버는 프로젝트 리소스를 만들고 삭제할 수 있습니다.
- `review-troubleshoot` 프로젝트에 한해 `qa` 그룹에 `view` 역할을 할당합니다. 이 역할의 멤버에게는 프로젝트 리소스에 대한 읽기 액세스 권한이 제공됩니다.

- 4.1. **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p review
Login successful.
...output omitted...
```

- 4.2. 세 개의 사용자 그룹을 만듭니다.

```
[student@workstation ~]$ for group in leaders developers qa
> do
> oc adm groups new ${group}
> done
group.user.openshift.io/leaders created
group.user.openshift.io/developers created
group.user.openshift.io/qa created
```

- 4.3. 각 사용자를 적절한 그룹에 추가합니다.

```
[student@workstation ~]$ oc adm groups add-users leaders leader
group.user.openshift.io/leaders added: "leader"
[student@workstation ~]$ oc adm groups add-users developers developer
group.user.openshift.io/developers added: "developer"
[student@workstation ~]$ oc adm groups add-users qa qa-engineer
group.user.openshift.io/qa added: "qa-engineer"
```

- 4.4. **leaders** 그룹의 멤버가 새 프로젝트를 만들도록 허용합니다.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
> self-provisioner leaders
clusterrole.rbac.authorization.k8s.io/self-provisioner added: "leaders"
```

- 4.5. **system:authenticated:oauth** 그룹에서 **self-provisioner** 클러스터 역할을 제거합니다.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
> self-provisioner system:authenticated:oauth
Warning: Your changes may get lost whenever a master is restarted,
unless you prevent reconciliation of this rolebinding using the
following command: oc annotate clusterrolebinding.rbac self-provisioners
'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
"system:authenticated:oauth"
```

- 4.6. **developers** 그룹의 멤버가 **review-troubleshoot** 프로젝트에서 리소스를 만들고 삭제하도록 허용합니다.

```
[student@workstation ~]$ oc policy add-role-to-group edit developers
clusterrole.rbac.authorization.k8s.io/edit added: "developers"
```

- 4.7. **qa** 그룹의 멤버가 프로젝트 리소스를 보도록 허용합니다.

```
[student@workstation ~]$ oc policy add-role-to-group view qa
clusterrole.rbac.authorization.k8s.io/view added: "qa"
```

5. developer 사용자로 deployment를 사용하여 review-troubleshoot 프로젝트에 mysql이라는 애플리케이션을 만듭니다. [registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7](https://registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7)에서 사용할 수 있는 컨테이너 이미지를 사용합니다. 이 애플리케이션에서는 다른 프로젝트 애플리케이션에 대한 공유 데이터베이스 서비스를 제공합니다.

password를 키로 사용하고 r3dh4t123을 값으로 사용하여 mysql이라는 일반 시크릿을 생성합니다.

MYSQL\_ROOT\_PASSWORD 환경 변수를 mysql 시크릿의 password 키 값으로 설정합니다.

PVC(영구 볼륨 클레임)를 포드 내의 /var/lib/mysql/data 디렉터리에 마운트하도록 mysql 데이터베이스 애플리케이션을 구성합니다. PVC는 크기가 2GB여야 하며, **ReadWriteOnce** 액세스 모드만 요청해야 합니다.

5.1. 클러스터에 developer 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p review
Login successful.
...output omitted...
```

5.2. 새 애플리케이션을 만들어 mysql 데이터베이스 서버를 배포합니다. oc new-app 명령을 사용하여 배포를 만듭니다.

```
[student@workstation ~]$ oc new-app --name mysql \
> --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
...output omitted...
--> Creating resources ...
 imagestream.image.openshift.io "mysql" created
 deployment.apps "mysql" created
 service "mysql" created
--> Success
...output omitted...
```

5.3. password를 키로 사용하고 r3dh4t123을 값으로 사용하여 MySQL 데이터베이스에 mysql이라는 일반 시크릿을 생성합니다.

```
[student@workstation ~]$ oc create secret generic mysql \
> --from-literal password=r3dh4t123
secret/mysql created
```

5.4. mysql 시크릿을 사용하여 mysql 배포의 환경 변수를 초기화합니다.

```
[student@workstation ~]$ oc set env deployment mysql \
> --prefix MYSQL_ROOT_ --from secret/mysql
deployment.apps/mysql updated
```

5.5. oc set volumes 명령을 사용하여 mysql 배포에 사용할 영구저장장치를 구성합니다. 이 명령은 지정된 크기 및 액세스 모드를 사용하여 영구 볼륨 클레임을 자동으로 생성합니다. /var/lib/mysql/data 디렉터리에 볼륨을 마운트하면 하나의 데이터베이스 포드를 삭제하고 다른 데이터베이스 포드를 생성해도 저장된 데이터에 액세스할 수 있습니다.

```
[student@workstation ~]$ oc set volumes deployment/mysql --name mysql-storage \
> --add --type pvc --claim-size 2Gi --claim-mode rwo \
> --mount-path /var/lib/mysql/data
deployment.apps/mysql volume updated
```

- 5.6. 시크릿 및 볼륨을 사용하도록 배포를 구성한 후 **mysql** 포드에서 재배포하는지 확인합니다.  
**mysql** 포드에 **1/1** 및 **Running**이 둘 다 표시될 때까지 **oc get pods** 명령을 여러 번 실행해야 할 수 있습니다.

```
[student@workstation ~]$ oc get pods -l deployment=mysql
NAME READY STATUS RESTARTS AGE
mysql-bbb6b5fbb-dmq9x 1/1 Running 0 63s
```

- 5.7. 영구 볼륨 클레임이 올바른 크기 및 액세스 모드로 존재하는지 확인합니다.

```
[student@workstation ~]$ oc get pvc
NAME STATUS ... CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-ks52v Bound ... 2G RWO nfs-storage 2m33s
```

- 5.8. 실행 중인 **mysql** 포드에서 **/var/lib/mysql/data** 디렉터리에 볼륨을 마운트하는지 확인합니다.

```
[student@workstation ~]$ oc exec mysql-bbb6b5fbb-dmq9x -- df -h \
> | grep /var/lib/mysql/data
192.168.50.254:/exports/review-troubleshoot-pvc-ks52v-pvc-0d6a63bd-286e-44ec-b29c-
ffbb34928b86 40G 859M 40G 3% /var/lib/mysql/data
```

6. **developer** 사용자로 **deployment**을 사용하여 **wordpress**라는 애플리케이션을 만듭니다. **review-troubleshoot** 프로젝트에 애플리케이션을 생성합니다. **quay.io/redhattraining/wordpress:5.3.0**에서 사용할 수 있는 이미지를 사용합니다. 이 이미지는 **docker.io/library/wordpress:5.3.0**에서 사용할 수 있는 컨테이너 이미지의 사본입니다.
- WORDPRESS\_DB\_HOST** 환경 변수가 **mysql** 값을 가지도록 구성합니다. 애플리케이션에서는 이 변수를 사용하여 **mysql** 애플리케이션에서 제공하는 **mysql** 데이터베이스 서비스를 연결합니다.
- WORDPRESS\_DB\_NAME** 환경 변수가 **wordpress** 값을 가지도록 구성합니다. 애플리케이션에서는 이 변수를 사용하여 데이터베이스의 이름을 식별합니다. 데이터베이스가 데이터베이스 서버에 없는 경우에는 애플리케이션에서 이 이름으로 새 데이터베이스를 만들려고 시도합니다.
- WORDPRESS\_DB\_USER** 환경 변수를 **root** 값으로 설정합니다. **WORDPRESS\_DB\_PASSWORD** 환경 변수를 **mysql** 시크릿의 **password** 키 값으로 설정합니다. **WORDPRESS\_DB\_PASSWORD** 환경 변수의 값은 **mysql** 루트 사용자 암호와 같아야 합니다.
- wordpress** 애플리케이션에는 **anyuid** 보안 컨텍스트 제약도 필요합니다. **wordpress-sa**라는 서비스 계정을 만든 다음 **anyuid** 보안 컨텍스트 제약 조건을 할당합니다. **wordpress-sa** 서비스 계정을 사용하도록 **wordpress** 배포를 구성합니다.
- apps.ocp4.example.com** 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 애플리케이션에 대한 경로를 만듭니다. 애플리케이션을 올바르게 배포하면 브라우저에서 애플리케이션에 액세스할 때 설치 마법사가 표시됩니다.

- 6.1. **wordpress** 애플리케이션을 **deployment**로 배포합니다.

```
[student@workstation ~]$ oc new-app --name wordpress \
> --docker-image quay.io/redhattraining/wordpress:5.3.0 \
> -e WORDPRESS_DB_HOST=mysql -e WORDPRESS_DB_USER=root \
> -e WORDPRESS_DB_NAME=wordpress
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "wordpress" created
deployment.apps "wordpress" created
service "wordpress" created
--> Success
...output omitted...
```

- 6.2. `wordpress` 배포에 `WORDPRESS_DB_PASSWORD` 환경 변수를 추가합니다. `mysql` 시크릿의 `password` 키 값을 변수의 값으로 사용합니다.

```
[student@workstation ~]$ oc set env deployment/wordpress \
> --prefix WORDPRESS_DB_ --from secret/mysql
deployment.apps/wordpress updated
```

- 6.3. `wordpress-sa` 서비스 계정을 생성합니다.

```
[student@workstation ~]$ oc create serviceaccount wordpress-sa
serviceaccount/wordpress-sa created
```

- 6.4. `admin` 사용자로 클러스터에 로그인하고 `anyuid` 권한을 `wordpress` 서비스 계정에 부여합니다.

```
[student@workstation ~]$ oc login -u admin -p review
Login successful.
...output omitted...
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added:
"wordpress-sa"
```

- 6.5. 나머지 단계를 수행하려면 `developer` 사용자로 다시 전환합니다. 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u developer -p review
Login successful.
...output omitted...
```

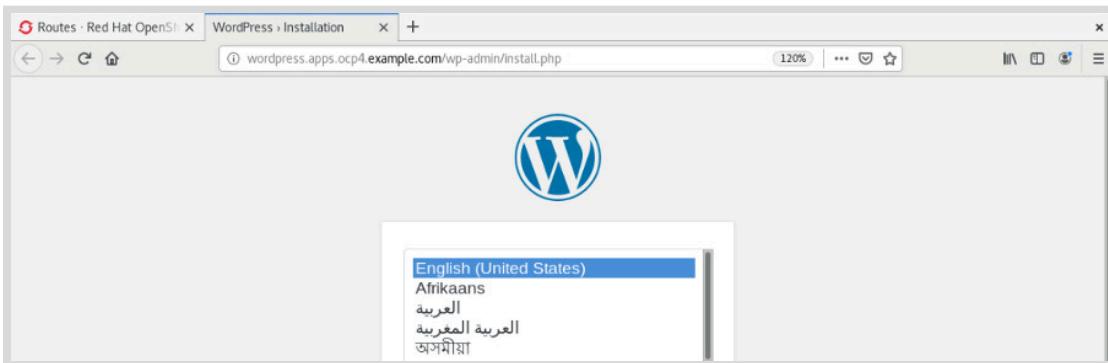
- 6.6. `wordpress-sa` 서비스 계정을 사용하도록 `wordpress` 배포를 구성합니다.

```
[student@workstation ~]$ oc set serviceaccount deployment/wordpress \
> wordpress-sa
deployment.apps/wordpress serviceaccount updated
```

- 6.7. `wordpress` 애플리케이션의 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service wordpress \
> --hostname wordpress.apps.ocp4.example.com
route.route.openshift.io/wordpress exposed
```

- 6.8. 웹 브라우저를 사용하여 URL <http://wordpress.apps.ocp4.example.com>에 대한 액세스 권한을 확인합니다. 애플리케이션을 올바르게 배포하면 브라우저에 설치 마법사가 표시됩니다.



7. developer 사용자로 `~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh` 스크립트를 사용하여 review-troubleshoot 프로젝트에 famous-quotes 애플리케이션을 배포합니다. 이 스크립트는 defaultdb 데이터베이스와 `~/D0280/labs/review-troubleshoot/famous-quotes.yaml` 파일에 정의된 리소스를 만듭니다.

스크립트를 실행한 후에는 초기에 애플리케이션 포드가 배포되지 않습니다. `famous-quotes` 배포 구성에 노드 선택기가 지정되며 일치하는 노드 레이블이 있는 클러스터 노드가 없습니다.

배포 구성에서 노드 선택기를 제거하여 OpenShift가 사용 가능한 노드에서 애플리케이션 포드를 예약 할 수 있도록 합니다.

`apps.ocp4.example.com` 하위 도메인에서 사용 가능한 호스트 이름을 사용하여 `famous-quotes` 애플리케이션에 대한 경로를 만든 다음 애플리케이션이 외부 요청에 응답하는지 확인합니다.

- 7.1. `~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh` 스크립트를 실행합니다.

```
[student@workstation ~]$ ~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh
Creating famous-quotes database
Deploying famous-quotes application
deploymentconfig.apps.openshift.io/famous-quotes created
service/famous-quotes created
```

- 7.2. `famous-quotes` 애플리케이션 포드가 배포되도록 예약되지 않았는지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
famous-quotes-1-deploy 0/1 Pending 0 41s
...output omitted...
```

- 7.3. 문제에 대한 정보를 제공하는 프로젝트 이벤트가 있는지 확인합니다.

```
[student@workstation ~]$ oc get events --sort-by='{"lastTimestamp"}'
...output omitted...
34s Warning FailedScheduling pod/famous-quotes-1-deploy 0/3 nodes are
available: 3 node(s) didn't match node selector.
...output omitted...
```

7.4. **famous-quotes** 배포 구성 리소스를 파일에 저장합니다.

```
[student@workstation ~]$ oc get dc/famous-quotes -o yaml > /tmp/famous-dc.yaml
```

7.5. 편집기를 사용하여 `/tmp/famous-dc.yaml` 파일에서 노드 선택기를 제거합니다. 파일에서 `nodeSelector` 행을 검색합니다. `/tmp/famous-dc.yaml` 파일에서 다음 두 행을 삭제합니다.

```
nodeSelector:
 env: quotes
```

7.6. **famous-quotes** 배포 구성을 수정된 파일로 교체합니다.

```
[student@workstation ~]$ oc replace -f /tmp/famous-dc.yaml
deploymentconfig.apps.openshift.io/famous-quotes replaced
```

7.7. 잠시 기다렸다가 `oc get pods` 명령을 실행하여 **famous-quotes** 애플리케이션이 현재 실행되고 있는지 확인합니다.

```
[student@workstation ~]$ oc get pods -l app=famous-quotes
NAME READY STATUS RESTARTS AGE
famous-quotes-2-gmz2j 1/1 Running 0 53s
```

7.8. **famous-quotes** 애플리케이션의 경로를 만듭니다.

```
[student@workstation ~]$ oc expose service famous-quotes \
> --hostname quotes.apps.ocp4.example.com
route.route.openshift.io/famous-quotes exposed
```

7.9. **famous-quotes** 애플리케이션이 `http://quotes.apps.ocp4.example.com` URL로 전송된 요청에 응답하는지 확인합니다.

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.example.com \
> | grep Quote
<title>Quotes</title>
<h1>Quote List</h1>
```

## 평가

**workstation** 시스템에서 **student** 사용자로 `lab` 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab review-troubleshoot grade
```

## 완료

workstation 시스템에서 student 사용자로 lab 명령을 사용하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab review-troubleshoot finish
```

이로써 종합 검토가 완료됩니다.

## ▶ 랩

# 리소스 및 네트워크 제한 사항을 사용하여 프로젝트 템플릿 구성

이 검토에서는 새 프로젝트에서 기본 할당량, 리소스 제한, 네트워크 정책을 강제 적용하도록 클러스터의 기본 프로젝트 템플릿을 구성합니다.

## 결과

다음을 수행할 수 있습니다.

- 기본 프로젝트 템플릿을 수정하여 제한 범위, 리소스 할당량, 네트워크 정책을 자동으로 생성합니다.
- 제공된 파일을 사용하여 TLS 시크릿을 생성합니다.
- 시크릿을 애플리케이션 내에 볼륨으로 마운트합니다.
- 애플리케이션에 대한 패스스루 경로를 만듭니다.
- 자동으로 확장되도록 애플리케이션을 구성합니다.

## 시작하기 전에

`workstation` 시스템에서 `student` 사용자로 `lab` 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 다음을 수행합니다.

- 클러스터 API에 연결할 수 있는지 확인합니다.
- HTPasswd Identity 프로바이더를 구성하고 `admin`, `leader` 및 `developer` 사용자에게 액세스 권한을 제공합니다.
- 샘플 YAML 파일을 `~/DO280/labs/review-template/sample-files/`에 다운로드합니다.
- `review-template-test` 프로젝트를 만들고, 네트워크 정책을 테스트하는데 사용할 수 있는 애플리케이션을 프로젝트에 배포합니다.
- `default` 네임스페이스에 `network.openshift.io/policy-group=ingress` 레이블을 추가합니다.
- 보안 애플리케이션에 필요한 인증서 파일을 생성합니다.

```
[student@workstation ~]$ lab review-template start
```



### 참고

시작하는 데 도움이 필요한 경우 Red Hat OpenShift Container Platform 설치 후 구성 가이드의 설치 후 네트워크 구성 장을 참조로 사용하십시오.

## 지침

다음 작업을 완료합니다.

- admin** 사용자로 새 프로젝트 템플릿을 사용하도록 OpenShift 클러스터를 업데이트합니다. 프로젝트 템플릿은 새 프로젝트에 대한 네트워크 정책, 제한 범위 및 할당량 리소스를 자동으로 생성해야 합니다. 새 프로젝트에는 프로젝트 이름과 일치하는 레이블이 자동으로 지정되어야 합니다. 예를 들어 **test**라는 프로젝트에는 **name=test** 레이블이 있습니다.

다음 테이블에는 필수 리소스에 대한 지침이 있습니다.

리소스	요구 사항
프로젝트	<ul style="list-style-type: none"> <li>프로젝트 이름과 레이블이 포함됩니다.</li> </ul>
NetworkPolicy	<p>정책 1:</p> <ul style="list-style-type: none"> <li>경로는 외부 트래픽에 액세스할 수 있습니다. 즉, <b>network.openshift.io/policy-group=ingress</b> 레이블이 있는 네임스페이스의 포드에서 발생하는 트래픽을 허용합니다.</li> </ul> <p>정책 2:</p> <ul style="list-style-type: none"> <li>동일한 네임스페이스의 포드는 서로 통신할 수 있습니다.</li> <li>포드는 <b>network.openshift.io/policy-group=ingress</b> 레이블이 있는 네임스페이스를 제외한 다른 네임스페이스의 포드에는 응답하지 않습니다.</li> </ul>
LimitRange	<ul style="list-style-type: none"> <li>각 컨테이너는 CPU 30밀리코어를 요청합니다.</li> <li>각 컨테이너는 메모리 30MiB를 요청합니다.</li> <li>각 컨테이너는 CPU 100밀리코어로 제한됩니다.</li> <li>각 컨테이너는 메모리 100MiB로 제한됩니다.</li> </ul>
ResourceQuota	<ul style="list-style-type: none"> <li>프로젝트는 포드 10개로 제한됩니다.</li> <li>프로젝트에서 최대 1GiB의 메모리를 요청할 수 있습니다.</li> <li>프로젝트에서 최대 두 개의 CPU를 요청할 수 있습니다.</li> <li>프로젝트에서 최대 4GiB의 메모리를 사용할 수 있습니다.</li> <li>프로젝트에서 최대 4개의 CPU를 사용할 수 있습니다.</li> </ul>

- developer** 사용자로 **review-template**이라는 프로젝트를 만듭니다. **review-template** 프로젝트가 새 프로젝트 템플릿에 지정된 설정을 상속하는지 확인합니다. **review-template** 프로젝트에서 **quay.io/redhattraining/hello-world-secure:v1.0**에 있는 컨테이너 이미지를 사용하여 **hello-secure**라는 배포를 생성합니다.



### 참고

NGINX 서버에 필요한 TLS 인증서 및 키에 대한 액세스 권한을 제공할 때까지 **hello-secure** 포드는 실행되지 않습니다.

3. developer 사용자로 ~/D0280/labs/review-template/ 디렉터리에 있는 **hello-secure-combined.pem** 인증서 및 **hello-secure-key.pem** 키를 사용하여 TLS 시크릿을 생성합니다. 실패한 **hello-secure** 포드의 로그를 사용하여 인증서의 예상 마운트 지점을 확인합니다. 확인된 디렉터리를 사용하여 TLS 시크릿을 포드에 볼륨으로 마운트합니다. **hello-secure** 포드가 재배포되는지 확인합니다.
4. **hello-secure-combined.pem** 인증서는 단일 호스트 이름에 유효합니다. **openssl x509** 명령을 **-noout** 및 **-ext 'subjectAltName'** 옵션과 함께 사용하여 **hello-secure-combined.pem** 인증서를 읽고 호스트 이름을 확인합니다. developer 사용자로서 확인된 호스트 이름을 사용하여 **hello-secure** 서비스에 대한 패스스루 경로를 만듭니다. 이 경로가 외부 요청에 응답하는지 확인합니다.



### 참고

X509(1) 도움말 페이지에 **openssl x509** 명령에 대한 정보가 나와 있습니다.

5. developer 사용자로서 **hello-secure** 배포를 자동으로 크기를 조정하도록 구성합니다. 배포에는 하나 이상의 포드가 실행되고 있어야 합니다. 평균 CPU 사용률이 80%를 초과하면 배포가 최대 5개의 포드로 확장됩니다.



### 참고

~/D0280/solutions/review-template/test-hpa.sh에 있는 스크립트를 사용하여 배포가 예상대로 확장되는지 테스트할 수 있습니다.

## 평가

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab review-template grade
```

## 완료

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab review-template finish
```

이로써 종합 검토가 완료됩니다.

## ▶ 솔루션

# 리소스 및 네트워크 제한 사항을 사용하여 프로젝트 템플릿 구성

이 검토에서는 새 프로젝트에서 기본 할당량, 리소스 제한, 네트워크 정책을 강제 적용하도록 클러스터의 기본 프로젝트 템플릿을 구성합니다.

### 결과

다음을 수행할 수 있습니다.

- 기본 프로젝트 템플릿을 수정하여 제한 범위, 리소스 할당량, 네트워크 정책을 자동으로 생성합니다.
- 제공된 파일을 사용하여 TLS 시크릿을 생성합니다.
- 시크릿을 애플리케이션 내에 볼륨으로 마운트합니다.
- 애플리케이션에 대한 패스스루 경로를 만듭니다.
- 자동으로 확장되도록 애플리케이션을 구성합니다.

### 시작하기 전에

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습에 사용할 시스템을 준비합니다.

이 명령은 다음을 수행합니다.

- 클러스터 API에 연결할 수 있는지 확인합니다.
- HTPasswd Identity 프로바이더를 구성하고 **admin**, **leader** 및 **developer** 사용자에게 액세스 권한을 제공합니다.
- 샘플 YAML 파일을 **~/D0280/labs/review-template/sample-files/**에 다운로드합니다.
- **review-template-test** 프로젝트를 만들고, 네트워크 정책을 테스트하는 데 사용할 수 있는 애플리케이션을 프로젝트에 배포합니다.
- **default** 네임스페이스에 **network.openshift.io/policy-group=ingress** 레이블을 추가합니다.
- 보안 애플리케이션에 필요한 인증서 파일을 생성합니다.

```
[student@workstation ~]$ lab review-template start
```



#### 참고

시작하는 데 도움이 필요한 경우 Red Hat OpenShift Container Platform 설치 후 구성 가이드의 설치 후 네트워크 구성 장을 참조로 사용하십시오.

## 지침

다음 작업을 완료합니다.

1. **admin** 사용자로 새 프로젝트 템플릿을 사용하도록 OpenShift 클러스터를 업데이트합니다. 프로젝트 템플릿은 새 프로젝트에 대한 네트워크 정책, 제한 범위 및 할당량 리소스를 자동으로 생성해야 합니다. 새 프로젝트에는 프로젝트 이름과 일치하는 레이블이 자동으로 지정되어야 합니다. 예를 들어 **test**라는 프로젝트에는 **name=test** 레이블이 있습니다.

다음 테이블에는 필수 리소스에 대한 지침이 있습니다.

리소스	요구 사항
프로젝트	<ul style="list-style-type: none"> <li>프로젝트 이름과 레이블이 포함됩니다.</li> </ul>
NetworkPolicy	<p>정책 1:</p> <ul style="list-style-type: none"> <li>경로는 외부 트래픽에 액세스할 수 있습니다. 즉, <b>network.openshift.io/policy-group=ingress</b> 레이블이 있는 네임스페이스의 포드에서 발생하는 트래픽을 허용합니다.</li> </ul> <p>정책 2:</p> <ul style="list-style-type: none"> <li>동일한 네임스페이스의 포드는 서로 통신할 수 있습니다.</li> <li>포드는 <b>network.openshift.io/policy-group=ingress</b> 레이블이 있는 네임스페이스를 제외한 다른 네임스페이스의 포드에는 응답하지 않습니다.</li> </ul>
LimitRange	<ul style="list-style-type: none"> <li>각 컨테이너는 CPU 30밀리코어를 요청합니다.</li> <li>각 컨테이너는 메모리 30MiB를 요청합니다.</li> <li>각 컨테이너는 CPU 100밀리코어로 제한됩니다.</li> <li>각 컨테이너는 메모리 100MiB로 제한됩니다.</li> </ul>
ResourceQuota	<ul style="list-style-type: none"> <li>프로젝트는 포드 10개로 제한됩니다.</li> <li>프로젝트에서 최대 1GiB의 메모리를 요청할 수 있습니다.</li> <li>프로젝트에서 최대 두 개의 CPU를 요청할 수 있습니다.</li> <li>프로젝트에서 최대 4GiB의 메모리를 사용할 수 있습니다.</li> <li>프로젝트에서 최대 4개의 CPU를 사용할 수 있습니다.</li> </ul>

- 1.1. OpenShift 클러스터에 **admin** 사용자로 로그인합니다.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. **oc adm create-bootstrap-project-template** 명령을 사용하여 이 연습을 위해 사용자 지정하는 새 YAML 파일을 만듭니다. 파일을 **~/DO280/labs/review-template/project-template.yaml**에 저장합니다.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
> -o yaml > ~/D0280/labs/review-template/project-template.yaml
```

1.3. ~/D0280/labs/review-template/ 디렉터리로 변경합니다.

```
[student@workstation ~]$ cd ~/D0280/labs/review-template/
```

1.4. **project-template.yaml** 파일을 편집하여 새 프로젝트의 레이블을 추가합니다. 굵은 글꼴 행을 추가하고 들여쓰기가 올바른지 확인한 다음 파일을 저장합니다. **PROJECT\_NAME** 변수는 프로젝트 이름 값 사용합니다.

```
...output omitted...
- apiVersion: project.openshift.io/v1
 kind: Project
 metadata:
 labels:
 name: ${PROJECT_NAME}
 annotations:
 ...output omitted...
```

1.5. ~/D0280/labs/review-template/sample-files/ 디렉터리의 파일을 나열 합니다. 이 디렉터리에서는 샘플 네트워크 정책 파일 2개, 샘플 제한 범위 파일, 샘플 리소스 할당량 파일을 제공합니다.

```
[student@workstation review-template]$ ls sample-files/
allow-from-openshift-ingress.yaml allow-same-namespace.yaml limitrange.yaml
resourcequota.yaml
```

1.6. **sample-files/allow-\*.yaml** 파일의 내용을 **project-template.yaml** 파일에 추가합니다. **project-template.yaml** 파일에는 리소스 목록이 필요하므로 각 리소스의 첫 번째 행을 **-**로 시작해야 하며, 나머지 내용을 적절하게 들여쓰기해야 합니다.

**project-template.yaml** 파일을 편집하여 네트워크 정책 리소스를 추가합니다. 굵은 글꼴 행을 추가하고 들여쓰기가 올바른지 확인한 다음 파일을 저장합니다.

```
...output omitted...
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: User
 name: ${PROJECT_ADMIN_USER}
- apiVersion: networking.k8s.io/v1
 kind: NetworkPolicy
 metadata:
 name: allow-from-openshift-ingress
 spec:
 podSelector: {}
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 network.openshift.io/policy-group: ingress
```

```

- apiVersion: networking.k8s.io/v1
 kind: NetworkPolicy
 metadata:
 name: allow-same-namespace
 spec:
 podSelector: {}
 ingress:
 - from:
 - podSelector: {}
 parameters:
 - name: PROJECT_NAME
...output omitted...

```

- 1.7. `sample-files/limitrange.yaml` 및 `sample-files/resourcequota.yaml` 파일의 내용을 `project-template.yaml` 파일에 추가합니다. `project-template.yaml` 파일에는 리소스 목록이 필요하므로 각 리소스의 첫 번째 행을 `-`로 시작해야 하며, 나머지 내용을 적절하게 들여쓰기해야 합니다.

`project-template.yaml` 파일을 편집하여 제한 범위 및 리소스 할당량 리소스를 추가합니다. 굵은 글꼴 행을 추가하고 들여쓰기가 올바른지 확인한 다음 파일을 저장합니다.

```

...output omitted...
 ingress:
 - from:
 - podSelector: {}
- apiVersion: v1
 kind: LimitRange
 metadata:
 name: project-limitrange
 spec:
 limits:
 - default:
 memory: 100Mi
 cpu: 100m
 defaultRequest:
 memory: 30Mi
 cpu: 30m
 type: Container
- apiVersion: v1
 kind: ResourceQuota
 metadata:
 name: project-quota
 spec:
 hard:
 pods: '10'
 requests.cpu: '2'
 requests.memory: 1Gi
 limits.cpu: '4'
 limits.memory: 4Gi
 parameters:
 - name: PROJECT_NAME
...output omitted...

```

- 1.8. `oc create` 명령을 사용하여 `openshift-config` 네임스페이스에 `project-template.yaml` 파일을 사용하는 새 템플릿 리소스를 생성합니다.



### 참고

`~/D0280/solutions/review-template/project-template.yaml` 파일에는 올바른 구성이 포함되어 있어 비교에 사용할 수 있습니다.

```
[student@workstation review-template]$ oc create -f project-template.yaml \
> -n openshift-config
template.template.openshift.io/project-request created
```

- 1.9. `openshift-config` 네임스페이스에 템플릿을 나열합니다. 다음 단계에서 해당 템플릿 이름을 사용합니다.

```
[student@workstation review-template]$ oc get templates -n openshift-config
NAME DESCRIPTION PARAMETERS OBJECTS
project-request 5 (5 blank) 6
```

- 1.10. 새 템플릿을 사용하도록 `projects.config.openshift.io/cluster` 리소스를 업데이트합니다.

```
[student@workstation review-template]$ oc edit \
> projects.config.openshift.io/cluster
```

다음의 굵은 글꼴 행과 일치하도록 `spec: {}` 행을 수정합니다. `openshift-config` 네임스페이스에 확인된 템플릿 이름을 사용합니다. 들여쓰기가 올바른지 확인한 다음 변경 사항을 저장합니다.

```
...output omitted...
spec:
 projectRequestTemplate:
 name: project-request
```

- 1.11. 변경이 완료되면 `openshift-apiserver` 네임스페이스에 `apiserver` 포드가 다시 배포됩니다. 재배포를 모니터링합니다.

```
[student@workstation review-template]$ watch oc get pods -n openshift-apiserver
```

새 포드 3개가 모두 실행되면 `Ctrl+C`를 눌러 `watch` 명령을 종료합니다.

```
Every 2.0s: oc get pods -n openshift-apiserver ...
NAME READY STATUS RESTARTS AGE
apiserver-75cfdfc877-257vs 1/1 Running 0 61s
apiserver-75cfdfc877-12xnv 1/1 Running 0 29s
apiserver-75cfdfc877-rn9fs 1/1 Running 0 47s
```

2. `developer` 사용자로 `review-template`이라는 프로젝트를 만듭니다. `review-template` 프로젝트가 새 프로젝트 템플릿에 지정된 설정을 상속하는지 확인합니다. `review-template` 프로젝

트에서 `quay.io/redhattraining/hello-world-secure:v1.0`에 있는 컨테이너 이미지를 사용하여 `hello-secure`라는 배포를 생성합니다.



### 참고

NGINX 서버에 필요한 TLS 인증서 및 키에 대한 액세스 권한을 제공할 때까지 `hello-secure` 포드는 실행되지 않습니다.

2.1. OpenShift 클러스터에 `developer` 사용자로 로그인합니다.

```
[student@workstation review-template]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

2.2. `review-template` 프로젝트를 생성합니다.

```
[student@workstation review-template]$ oc new-project review-template
Now using project "review-template" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

2.3. `review-template` 프로젝트의 네트워크 정책, 제한 범위 및 할당량 리소스를 나열합니다.

```
[student@workstation review-template]$ oc get \
> networkpolicy,limitrange,resourcequota
NAME ...
networkpolicy.networking.k8s.io/allow-from-openshift-ingress ...
networkpolicy.networking.k8s.io/allow-same-namespace ...

NAME CREATED AT
limitrange/project-limitrange 2020-10-19T16:17:19Z

NAME AGE REQUEST
resourcequota/project-quota 37s pods: 0/10, requests.cpu: 0/2,
 requests.memory: 0/1Gi

 LIMIT
 limits.cpu: 0/4, limits.memory: 0/4Gi
```

2.4. `review-template` 프로젝트에 `name=review-template` 레이블이 있는지 확인합니다.

```
[student@workstation review-template]$ oc get project/review-template \
> --show-labels
NAME DISPLAY NAME STATUS LABELS
review-template Active name=review-template
```

2.5. `oc new-app` 명령과 `quay.io/redhattraining/hello-world-secure:v1.0` 컨테이너 이미지를 사용하여 `hello-secure` 배포를 생성합니다.

```
[student@workstation review-template]$ oc new-app --name hello-secure \
> --docker-image quay.io/redhattraining/hello-world-secure:v1.0
...output omitted...
```

```
--> Creating resources ...
imagestream.image.openshift.io "hello-secure" created
deployment.apps "hello-secure" created
service "hello-secure" created
--> Success
...output omitted...
```

- 2.6. **hello-secure** 포드가 시작되지 않는지 확인합니다.

```
[student@workstation review-template]$ watch oc get pods
```

포드의 상태가 **CrashLoopBackOff** 또는 **Error**인 경우 **Ctrl+C**를 눌러 **watch** 명령을 종료합니다.

NAME	READY	STATUS	RESTARTS	AGE
hello-secure-6475f657c9-rmgsr	0/1	<b>CrashLoopBackOff</b>	1	14s

3. **developer** 사용자로 `~/D0280/labs/review-template/` 디렉터리에 있는 **hello-secure-combined.pem** 인증서 및 **hello-secure-key.pem** 키를 사용하여 TLS 시크릿을 생성합니다. 실패한 **hello-secure** 포드의 로그를 사용하여 인증서의 예상 마운트 지점을 확인합니다. 확인된 디렉터리를 사용하여 TLS 시크릿을 포드에 볼륨으로 마운트합니다. **hello-secure** 포드가 재배포되는지 확인합니다.

- 3.1. **hello-secure-combined.pem** 인증서 및 **hello-secure-key.pem** 키를 사용하여 TLS 시크릿을 생성합니다.

```
[student@workstation review-template]$ oc create secret tls hello-tls \
> --cert hello-secure-combined.pem --key hello-secure-key.pem
secret/hello-tls created
```

- 3.2. 실패한 포드의 이름을 확인합니다.

NAME	READY	STATUS	RESTARTS	AGE
hello-secure-6475f657c9-rmgsr	0/1	<b>CrashLoopBackOff</b>	4	2m45s

- 3.3. 실패한 포드의 로그를 검사합니다. 포드에서 `/run/secrets/nginx/tls.crt` 파일을 사용하려고 하지만 파일이 존재하지 않는다는 내용이 로그에 표시됩니다.

```
[student@workstation review-template]$ oc logs hello-secure-6475f657c9-rmgsr
...output omitted...
nginx: [emerg] BIO_new_file("/run/secrets/nginx/tls.crt") failed (SSL:
error:02001002:system library:fopen:No such file or directory:fopen('/run/
secrets/nginx/tls.crt','r') error:2006D080:BIO routines:BIO_new_file:no such file)
```

- 3.4. **oc set volumes** 명령을 사용하여 시크릿을 `/run/secrets/nginx` 디렉터리에 마운트합니다.

```
[student@workstation review-template]$ oc set volumes deployment/hello-secure \
> --add --type secret --secret-name hello-tls --mount-path /run/secrets/nginx
info: Generated volume name: volume-hlrf
deployment.apps/hello-secure volume updated
```

3.5. `hello-secure` 포드가 재배포되는지 확인합니다.

```
[student@workstation review-template]$ watch oc get pods
```

포드에 `1/1` 및 `Running`이 표시되면 `Ctrl+C`를 눌러 `watch` 명령을 종료합니다.

Every 2.0s: `oc get pods`

NAME	READY	STATUS	RESTARTS	AGE
hello-secure-6bd8fcccb4-nhwr2	<b>1/1</b>	<b>Running</b>	0	25s

...

4. `hello-secure-combined.pem` 인증서는 단일 호스트 이름에 유효합니다. `openssl x509` 명령을 `-noout` 및 `-ext 'subjectAltName'` 옵션과 함께 사용하여 `hello-secure-combined.pem` 인증서를 읽고 호스트 이름을 확인합니다. `developer` 사용자로서 확인된 호스트 이름을 사용하여 `hello-secure` 서비스에 대한 패스스루 경로를 만듭니다. 이 경로가 외부 요청에 응답하는지 확인합니다.



### 참고

X509(1) 도움말 페이지에 `openssl x509` 명령에 대한 정보가 나와 있습니다.

4.1. `openssl x509` 명령을 사용하여 `hello-secure-combined.pem` 인증서를 검사합니다.

```
[student@workstation review-template]$ openssl x509 \
> -in hello-secure-combined.pem -noout -ext 'subjectAltName'
X509v3 Subject Alternative Name:
DNS:hello-secure.apps.ocp4.example.com
```

4.2. `hello-secure.apps.ocp4.example.com`을 가리키는 `hello-secure` 서비스에 대한 패스스루 경로를 만듭니다.

```
[student@workstation review-template]$ oc create route passthrough \
> --service hello-secure --hostname hello-secure.apps.ocp4.example.com
route.route.openshift.io/hello-secure created
```

4.3. `/home/student` 디렉터리로 돌아갑니다.

```
[student@workstation review-template]$ cd
```

`curl` 명령을 사용하여 이 경로가 외부 요청에 응답하는지 확인합니다.

```
[student@workstation ~]$ curl -s https://hello-secure.apps.ocp4.example.com \
> | grep Hello
<h1>Hello, world from nginx!</h1>
```

5. developer 사용자로서 **hello-secure** 배포를 자동으로 크기를 조정하도록 구성합니다. 배포에는 하나 이상의 포드가 실행되고 있어야 합니다. 평균 CPU 사용률이 80%를 초과하면 배포가 최대 5개의 포드로 확장됩니다.

**참고**

`~/D0280/solutions/review-template/test-hpa.sh`에 있는 스크립트를 사용하여 배포가 예상대로 확장되는지 테스트할 수 있습니다.

- 5.1. **oc autoscale** 명령을 사용하여 가로 포드 자동 확장기 리소스를 생성합니다.

```
[student@workstation ~]$ oc autoscale deployment/hello-secure \
> --min 1 --max 5 --cpu-percent 80
horizontalpodautoscaler.autoscaling/hello-secure autoscaled
```

- 5.2. 제공된 **test-hpa.sh** 스크립트를 실행합니다. 해당 스크립트에서는 Apache 벤치마킹 툴을 사용하여 부하를 적용하도록 **ab** 명령을 사용합니다.

```
[student@workstation ~]$ ~/D0280/solutions/review-template/test-hpa.sh
...output omitted...
Benchmarking hello-secure.apps.ocp4.example.com (be patient)
Completed 10000 requests
Completed 20000 requests
...output omitted...
Finished 100000 requests
...output omitted...
```

- 5.3. 2개 이상 5개 이하의 **hello-secure** 포드가 실행 중인지 확인합니다.

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
hello-secure-6bd8fcccb4-7qjcz 1/1 Running 0 96s
hello-secure-6bd8fcccb4-d67xd 1/1 Running 0 66s
hello-secure-6bd8fcccb4-m8vxp 1/1 Running 0 96s
hello-secure-6bd8fcccb4-nhwr2 1/1 Running 0 9m36s
```

**평가**

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 작업을 평가합니다. 보고된 오류를 모두 수정하고 성공할 때까지 명령을 다시 실행합니다.

```
[student@workstation ~]$ lab review-template grade
```

**완료**

**workstation** 시스템에서 **student** 사용자로 **lab** 명령을 사용하여 이 연습을 완료합니다.

```
[student@workstation ~]$ lab review-template finish
```

이로써 종합 검토가 완료됩니다.



## A부록

# GitHub 계정 만들기

### 목적

이 교육 과정의 랩에 사용할 GitHub 계정을 만드는 방법을 설명합니다.

# GitHub 계정 만들기

## 목표

이 섹션을 완료하면 이 교육 과정의 랩에 사용할 GitHub 계정과 공용 Git 리포지토리를 만들 수 있습니다.

## GitHub 계정 만들기

이 교육 과정의 랩에 사용할 공용 Git 리포지토리를 1개 이상 생성하려면 GitHub 계정이 있어야 합니다. GitHub 계정이 이미 있는 경우 이 부록에 있는 단계를 건너뛸 수 있습니다.



### 중요

이미 GitHub 계정이 있는 경우 이 교육 과정의 랩에 사용할 공용 Git 리포지토리만 만들어야 합니다. 랩 평가 스크립트 및 지침을 수행하려면 리포지토리를 복제하는 데 인증되지 않은 액세스가 필요합니다. 리포지토리는 암호, SSH 키 또는 GPG 키를 제공하지 않고 액세스할 수 있어야 합니다.

새 GitHub 계정을 만들려면 다음 단계를 수행하십시오.

1. 웹 브라우저를 사용하여 <https://github.com>로 이동합니다.
2. 필요한 세부 정보를 입력한 다음 **Sign up for GitHub**(GitHub에 등록)를 클릭합니다.

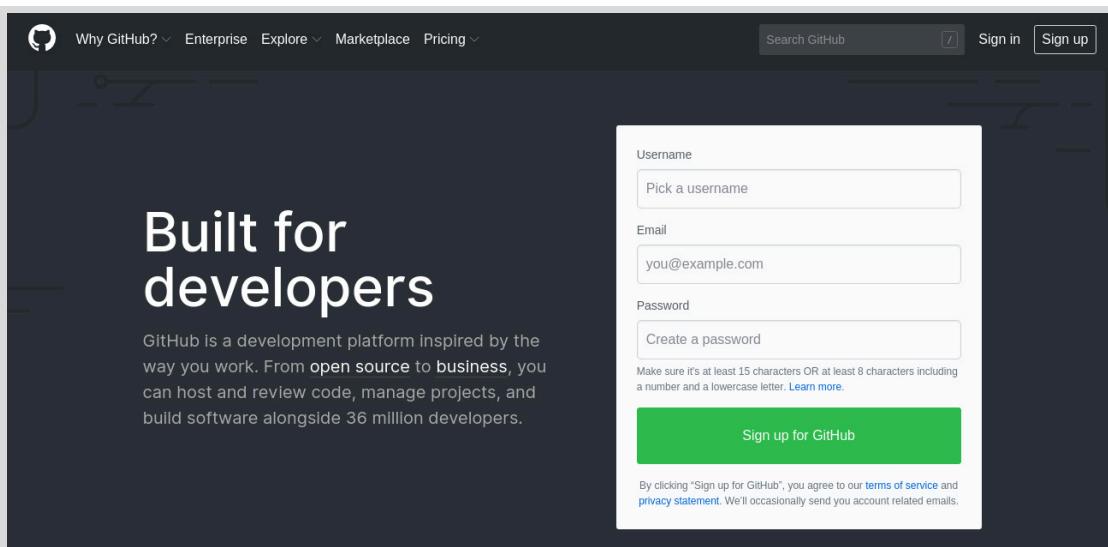


그림 A.1: GitHub 계정 만들기

3. GitHub 계정을 활성화하는 방법에 대한 지침이 포함된 이메일을 받게 됩니다. 이메일 주소를 확인한 다음 계정을 생성하는 동안 제공한 사용자 이름과 암호를 사용하여 GitHub 웹사이트에 로그인합니다.
4. GitHub에 로그인한 후 GitHub 홈 페이지 왼쪽에 있는 **리포지토리** 창에서 **New(신규)**를 클릭하여 새 Git 리포지토리를 만들 수 있습니다.

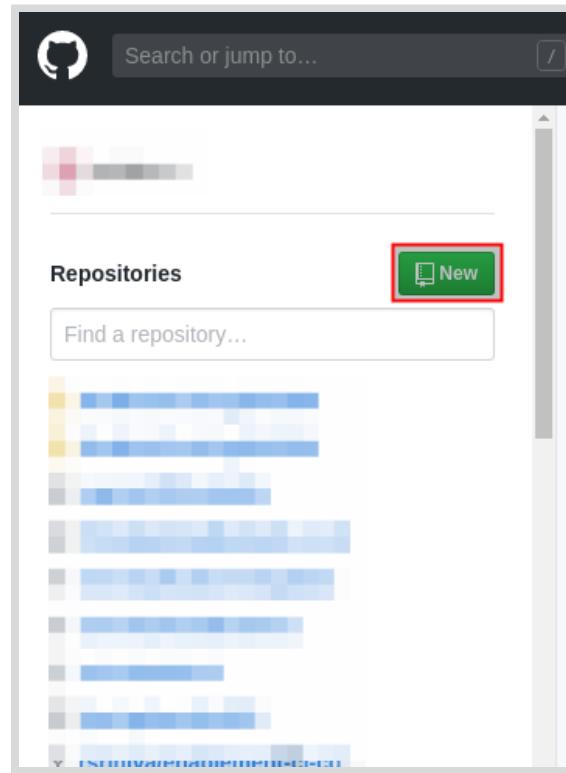


그림 A.2: 새 Git 리포지토리 만들기

또는 오른쪽 상단(벨 아이콘 오른쪽)에 있는 더하기 아이콘(+)을 클릭한 다음 New repository(새 리포지토리)를 클릭합니다.

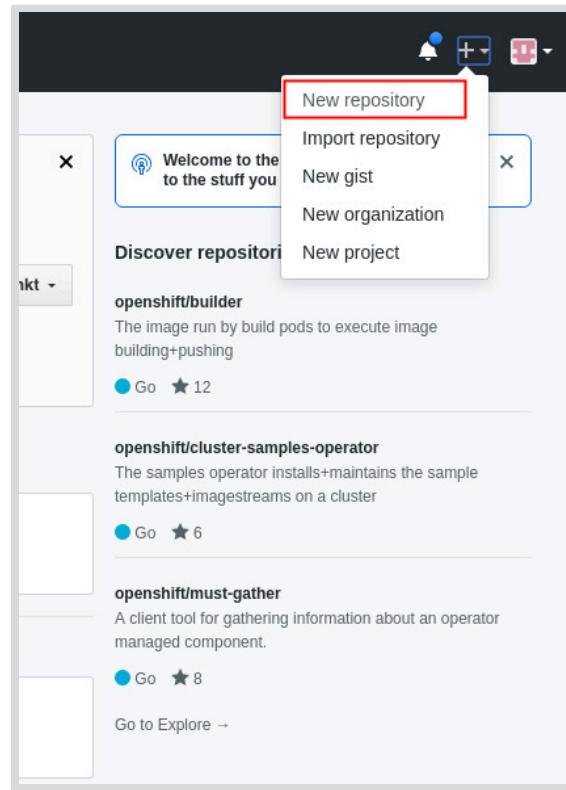


그림 A.3: 새 Git 리포지토리 만들기



### 참조

#### 새 GitHub 계정에 가입

<https://help.github.com/en/articles/signing-up-for-a-new-github-account>

## B부록

# Quay 계정 만들기

### 목적

이 교육 과정의 랩에 사용할 Quay 계정을 만드는 방법을 설명합니다.

# Quay 계정 만들기

## 목표

이 섹션을 완료하면 이 교육 과정의 랩에 사용할 Quay 계정과 공용 컨테이너 이미지 리포지토리를 만들 수 있습니다.

## Quay 계정 만들기

이 교육 과정의 랩에 사용할 공용 컨테이너 이미지 리포지토리를 1개 이상 생성하려면 Quay 계정이 있어야 합니다. Quay 계정이 이미 있는 경우 이 부록에 있는 새 계정 생성 단계를 건너뛸 수 있습니다.



### 중요

Quay 계정이 이미 있는 경우 이 교육 과정의 랩에 사용할 공용 컨테이너 이미지 리포지토리만 만들어야 합니다. 랩 평가 스크립트 및 지침을 수행하려면 리포지토리에서 컨테이너 이미지를 가져오는데 인증되지 않은 액세스가 필요합니다.

새 Quay 계정을 만들려면 다음 단계를 수행하십시오.

1. 웹 브라우저를 사용하여 <https://quay.io>로 이동합니다.
2. 오른쪽 상단(검색바 옆)에 있는 **Sign in(로그인)**을 클릭합니다.
3. **Sign in(로그인)** 페이지에서는 Google 또는 GitHub 자격 증명(부록 A에서 생성)을 사용하여 로그인할 수 있습니다.

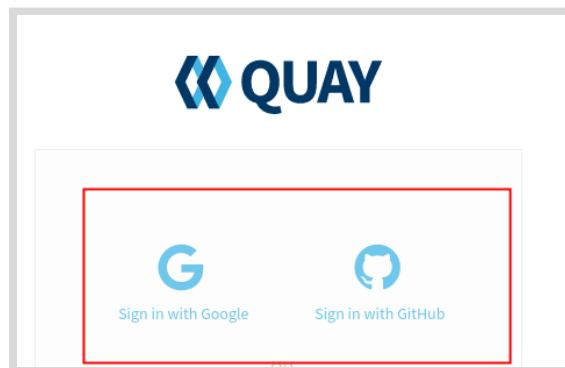


그림 B.1: Google 또는 GitHub 자격 증명을 사용하여 로그인합니다.

또는 Create account(계정 만들기)를 클릭하여 새 계정을 만듭니다.

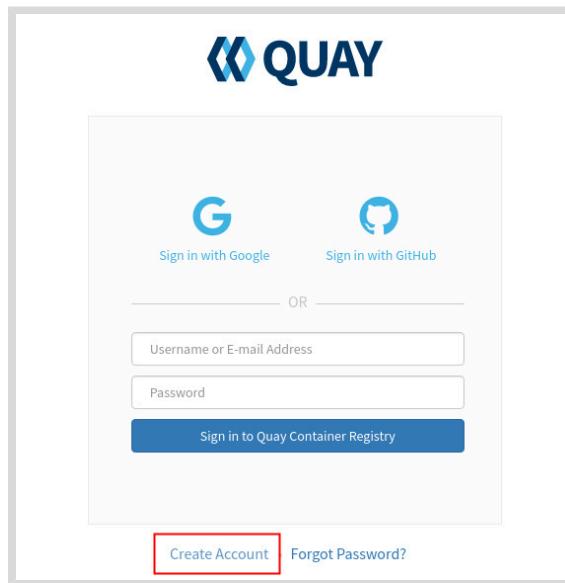


그림 B.2: 새 계정 만들기

4. Google 또는 GitHub 로그인 방법을 선택하는 대신 새 계정을 생성하도록 선택하면, Quay 계정을 활성화하는 방법에 대한 지침이 포함된 이메일을 받게 됩니다. 이메일 주소를 확인한 다음 계정을 생성하는 동안 제공한 사용자 이름과 암호를 사용하여 Quay 웹사이트에 로그인합니다.
5. Quay에 로그인한 후에는 **Repository(리포지토리)** 페이지에서 **Create New Repository(새 리포지토리 만들기)**를 클릭하여 새 이미지 리포지토리를 만들 수 있습니다.

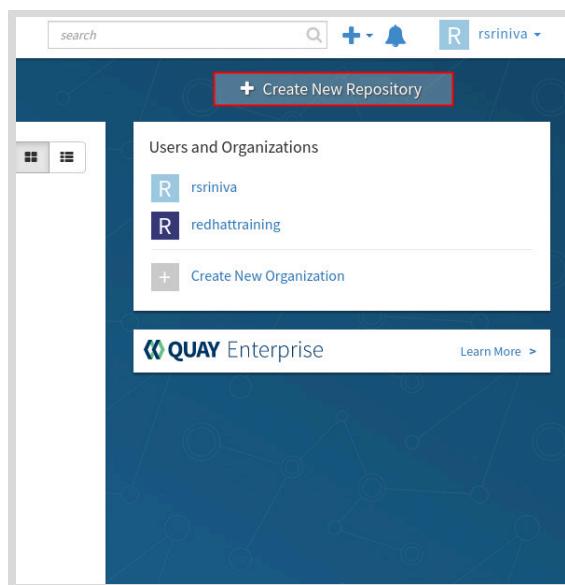


그림 B.3: 새 이미지 리포지토리 만들기

또는 오른쪽 상단(벨 아이콘 왼쪽)에 있는 더하기 아이콘(+)을 클릭한 다음 **New Repository(새 리포지토리)**를 클릭합니다.

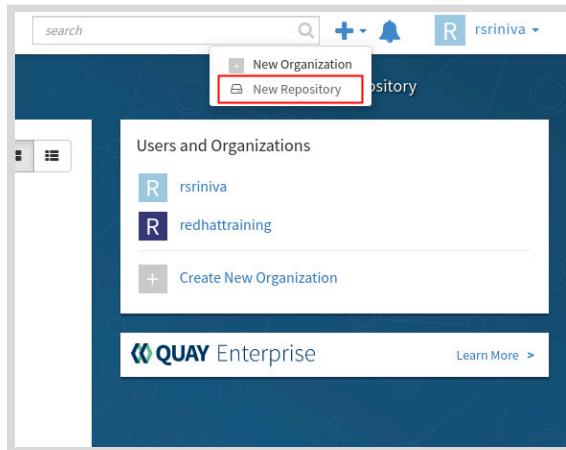


그림 B.4: 새 이미지 리포지토리 만들기



### 참조

#### Quay.io 시작하기

<https://docs.quay.io/solution/getting-started.html>

# 리포지토리 가시성

---

## 목표

이 섹션을 완료하면 Quay.io에서 리포지토리 가시성을 제어할 수 있게 됩니다.

## Quay.io 리포지토리 가시성

Quay.io는 공용 및 개인 리포지토리를 만들 수 있는 가능성을 제공합니다. 쓰기 권한은 명시적으로 부여되어야 하지만, 제한 없이 누구나 공용 리포지토리를 읽을 수 있습니다. 개인 리포지토리에 읽기 및 쓰기 권한이 모두 제한되어 있습니다. 그런데도 **quay.io**의 개인 리포지토리 수는 네임스페이스의 계획에 따라 제한됩니다.

## 기본 리포지토리 가시성

이미지를 **quay.io**에 내보내 만든 리포지토리는 기본적으로 비공개입니다. OpenShift(또는 기타 도구)에서 이러한 이미지를 가져오기 위해 OpenShift 및 Quay 모두에서 개인 키를 구성하거나 리포지토리를 공용으로 설정할 수 있으므로 인증이 필요하지 않습니다. 개인 키를 설정하는 것은 이 문서의 범위를 벗어납니다.

The screenshot shows the Quay.io interface. At the top, there is a navigation bar with icons for back, forward, and refresh, followed by a URL field containing <https://quay.io/repository/>. Below the URL is a logo for Red Hat Quay.io and links for EXPLORE, APPLICATIONS, REPOSITORIES (which is highlighted in pink), and TUTORIAL. The main content area has a dark background with a network-like graphic. A large title "Repositories" is centered at the top of the content area. Below it, a section titled "Starred" shows a message: "You haven't starred any repositories yet." with a subtitle "Stars allow you to easily access your favorite repositories." There are four repository cards displayed in a grid:

- RHT\_OCP4\_QUAY\_USER**
  - do180-mysql-57-rhel7** (with a star icon)
  - do180-todonodejs** (with a star icon)
  - do180-quote-php** (with a star icon)
  - nexus** (with a lock icon and a star icon)

## 리포지토리 가시성 업데이트

리포지토리 가시성을 공용으로 설정하려면 <https://quay.io/repository/>에서 적절한 리포지토리(필요한 경우 계정에 로그인)를 선택하고 왼쪽 하단 가장자리의 톱니바퀴를 클릭하여 **Settings**(설정) 페이지를 엽니다. **Repository Visibility**(리포지토리 가시성) 섹션으로 스크롤하고 **Make Public**(공용으로 만들기) 버튼을 클릭합니다.

The screenshot shows the Quay.io repository settings interface. At the top, there is a message "Trust Disabled" with a gear icon and a button to "Enable Trust". Below this is the "Events and Notifications" section, which displays a message stating "No notifications have been setup for this repository." and a button to "Create Notification". The "Repository Visibility" section shows that the repository is currently private, with a lock icon and a button to "Make Public". The final section is "Delete Repository", which contains a warning message "Deleting a Repository cannot be undone. Here be dragons!" and a red "Delete Repository" button.

리포지토리 목록을 다시 가져옵니다. 리포지토리 이름을 제외한 잠금 아이콘이 사라지고 리포지토리가 공용으로 표시됩니다.



## C부록

# 유용한 Git 명령

### 목적

이 교육 과정의 랩에 사용되는 유용한 Git 명령을 설명합니다.

# GIT 명령

---

## 목표

이 섹션을 완료하면 이 교육 과정의 연습을 다시 시작하고 다시 실행할 수 있습니다. 또한 완료되지 연습에서 다른 연습을 수행하도록 전환하고, 나중에 이전에 중단한 위치에서 연습을 계속할 수 있습니다.

## Git 분기 사용

이 과정에서는 GitHub에서 호스팅되는 Git 리포지토리를 사용하여 애플리케이션 과정 코드 소스 코드를 저장합니다. 과정의 시작 부분에서 GitHub에서 호스팅되는 이 리포지토리의 고유한 포크를 만듭니다.

이 과정에서 **workstation** VM에 복제하는 포크의 로컬 복사본을 사용하여 작업합니다. **origin**이라는 용어는 로컬 리포지토리가 복제되는 원격 리포지토리를 나타냅니다.

이 교육 과정의 연습을 수행하는 동안 각 연습에 별도의 Git 분기를 사용합니다. 소스 코드에 대한 모든 변경 사항은 해당 연습만을 위해 만든 새 분기에서 적용됩니다. **master** 분기를 변경하지 마십시오.

분기로 작업하고 알려진 양호한 상태로 복구하는 데 사용할 수 있는 시나리오 및 해당 Git 명령 목록이 아래에 나열되어 있습니다.

## 처음부터 다시 연습하기

연습을 완료한 후 처음부터 다시 수행하려면 다음 단계를 수행합니다.

- 로컬 분기의 모든 변경 사항을 연습 진행의 일부로 커밋하고 내보냅니다. 모든 리소스를 정리하도록 **finish** 하위 명령을 실행하여 연습을 완료했습니다.

```
[student@workstation ~]$ lab your-exercise finish
```

- D0180-apps** 리포지토리의 로컬 복제본으로 변경하고 **master** 분기로 전환합니다.

```
[student@workstation ~]$ cd ~/D0180-apps
[student@workstation D0180-apps]$ git checkout master
```

- 로컬 분기를 삭제합니다.

```
[student@workstation D0180-apps]$ git branch -d your-branch
```

- 개인 GitHub 계정에서 원격 분기를 삭제합니다.

```
[student@workstation D0180-apps]$ git push origin --delete your-branch
```

- start** 하위 명령을 사용하여 연습을 다시 시작합니다.

```
[student@workstation D0180-apps]$ cd ~
[student@workstation ~]$ lab your-exercise start
```

## 부분적으로 완료된 연습을 중단하고 처음부터 다시 시작하기

이 연습의 몇 가지 단계를 부분적으로 완료한 후 현재의 시도를 중단하고 처음부터 다시 시작하길 원하는 경우가 있을 수 있습니다. 다음 단계를 수행하십시오.

- 연습의 **finish** 하위 명령을 실행하여 모든 리소스를 정리합니다.

```
[student@workstation ~]$ lab your-exercise finish
```

- D0180-apps 리포지토리의 로컬 복제본을 입력하고 **git stash**를 사용하여 현재 분기에서 보류 중인 변경 사항을 삭제합니다.

```
[student@workstation ~]$ cd ~/D0180-apps
[student@workstation D0180-apps]$ git stash
```

- 로컬 리포지토리의 **master** 분기로 전환합니다.

```
[student@workstation D0180-apps]$ git checkout master
```

- 로컬 분기를 삭제합니다.

```
[student@workstation D0180-apps]$ git branch -d your-branch
```

- 개인 GitHub 계정에서 원격 분기를 삭제합니다.

```
[student@workstation D0180-apps]$ git push origin --delete your-branch
```

- 그러면 **start** 하위 명령을 실행하여 연습을 다시 시작할 수 있습니다.

```
[student@workstation D0180-apps]$ cd ~
[student@workstation ~]$ lab your-exercise start
```

## 완료되지 않은 연습에서 다른 연습으로 전환

연습의 몇 가지 단계를 부분적으로 완료했지만 다른 연습으로 전환한 후 나중에 현재 연습을 다시 방문하길 원하는 경우가 있을 수 있습니다.

나중에 다시 방문하려면 너무 많은 연습을 완료하지 않은 상태로 두지 마십시오. 이러한 연습은 클라우드 리소스를 연결하고, 클라우드 프로바이더 및 다른 학생들과 공유하는 OpenShift 클러스터에 할당된 할당량을 사용할 수 있습니다. 현재 연습으로 돌아갈 수 있을 때까지 시간이 걸릴 것으로 생각되는 경우 연습을 종단하고 나중에 처음부터 다시 시작하는 것이 좋습니다.

현재 연습을 일시 중지하고 다음 연습을 수행하려면 다음 단계를 수행합니다.

- 로컬 리포지토리에서 보류 중인 변경 사항을 모두 커밋하고 개인 GitHub 계정으로 내보냅니다. 연습을 중지한 단계를 기록하고자 할 수 있습니다.

```
[student@workstation ~]$ cd ~/D0180-apps
[student@workstation D0180-apps]$ git commit -a -m 'Paused at step X.Y'
[student@workstation D0180-apps]$ git push
```

2. 원래 연습의 **finish** 명령을 실행하지 마십시오. 나중에 다시 시작할 수 있도록 기존 OpenShift 프로젝트를 변경하지 않은 상태로 유지해야 합니다.
3. **start** 하위 명령을 실행하여 다음 연습을 시작합니다.

```
[student@workstation ~]$ lab your-exercise start
```

4. 다음 연습에서는 **master** 분기로 전환하고 필요에 따라 변경 사항에 사용할 새 분기를 만듭니다. 그러면 원래 분기의 원래 연습에 대한 변경 사항이 그대로 유지됩니다.
5. 나중에 다음 연습을 완료한 후 원래 연습으로 돌아가 해당 분기로 다시 전환하고자 할 수 있습니다.

```
[student@workstation ~]$ git checkout original-branch
```

이 경우 원래 연습을 중단했던 단계에서 계속할 수 있습니다.



### 참조

#### Git 분기 도움말 페이지

<https://git-scm.com/docs/git-branch>

#### Git 분기란 무엇입니까?

<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>

#### Git 툴 - Stashing

<https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>

## D부록

# OpenShift 클러스터 확장

### 목적

OpenShift 클러스터의 크기를 제어합니다.

### 목표

- OpenShift 클러스터를 확장하는 데 사용하는 리소스를 설명하고, 계산 노드를 수동으로 확장하는 방법을 확인합니다.
- 매개 변수를 정의하여 클러스터의 크기를 자동으로 제어합니다.

### 섹션

- OpenShift 클러스터 수동 확장
- OpenShift 클러스터 자동 확장

# OpenShift 클러스터 수동 확장

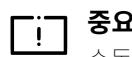
## 목표

이 섹션을 마치면 OpenShift 클러스터를 확장하는 데 사용하는 리소스를 설명하고, 계산 노드를 수동으로 확장하는 방법을 확인할 수 있습니다.

## 시스템 API 소개

OpenShift Container Platform에서는 시스템 API를 통해 계산 노드를 추가하거나 제거하여 클러스터의 크기를 조정할 수 있습니다. 이러한 확장 기능을 통해 클러스터는 애플리케이션에 충분한 컴퓨팅 성능을 제공할 수 있습니다. 확장 프로세스는 필요에 따라 수동 또는 자동으로 수행할 수 있습니다.

다음 다이어그램은 시스템 API(운영자로 실행) 및 관리하는 API 리소스를 보여줍니다. 운영자는 계산 노드 그룹을 설명하는 **Machineset** 컨트롤러와 같은 클러스터 리소스와 상호 작용하는 다양한 컨트롤러를 제공합니다.



### 중요

수동 또는 자동 확장을 수행하려면 시스템 API 운영자가 작동 중이어야 합니다. 시스템 API 운영자를 사용하려면 클라우드 프로바이더를 통합해야 하며, 풀스택 자동화 방법으로 설치된 클러스터에서 사용할 수 있습니다.

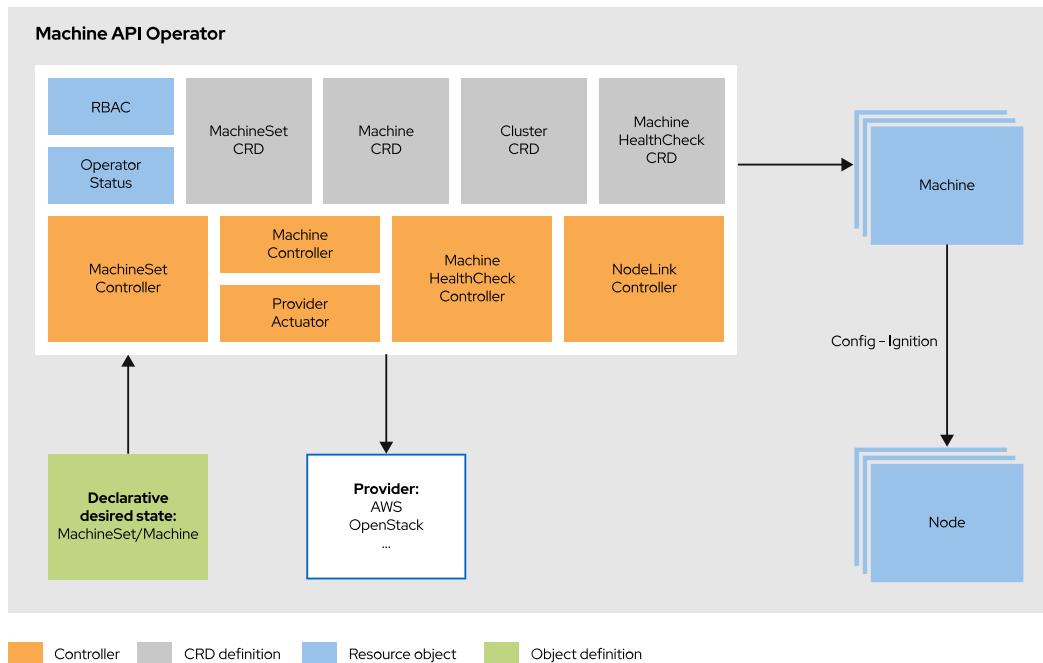


그림 D.1: 시스템 API 운영자

**참고**

자동 확장을 위한 API 리소스(**ClusterAutoscaler** 및 **MachineAutoscaler**)는 시스템 API 운영자가 관리하지 않습니다. 클러스터 자동 확장기 운영자는 이러한 두 가지 리소스를 관리합니다.

시스템 API는 다음과 같은 사용자 지정 리소스를 제공합니다.

**시스템**

시스템은 클러스터의 기본 계산 단위입니다. 각 시스템 리소스는 물리적 또는 가상 노드에 상호 연결합니다. 시스템은 시스템 집합의 인스턴스이며 시스템 집합에 정의된 정보를 상속합니다.

**MachineSets**

**MachineSet**은 시스템 그룹을 나타내지만 컨트롤 플레인 노드는 포함하지 않습니다. 시스템 집합은 복제본 집합이 포드인 시스템에 해당합니다. 시스템 집합에 따라 시스템 집합의 모든 시스템에 적용되는 설정이 정의됩니다. 이러한 설정에는 지역 및 영역 등에 대한 레이블과 AWS **m4.xlarge** 인스턴스 유형과 같은 인스턴스 유형이 포함됩니다. 시스템 집합의 시스템 수는 시스템 집합 리소스의 **replicas** 설정을 수정하거나 **oc scale** 명령을 실행하여 조정할 수 있습니다.

복제본 집합과 마찬가지로 시스템 집합은 레이블을 사용하여 멤버를 확인합니다. 시스템 API는 이 정보를 사용하여 시스템 집합의 시스템 복제본 수가 올바른지 확인합니다.

시스템 집합을 사용하여 클러스터 토플로지를 사용자 지정합니다. 예를 들어, 지역별로 고유한 시스템 집합을 정의할 수 있습니다. AWS에 배포된 OpenShift 클러스터는 가용성 영역별로 하나의 시스템 집합을 생성합니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
 name: ocp-xxxy-worker-us-west-2b
 ...output omitted...
spec:
 replicas: 1
 selector:
 matchLabels:
 machine.openshift.io/cluster-api-cluster: ocp-rnh2q
 machine.openshift.io/cluster-api-machineset: ocp-xxxy-worker-us-west-2b
 template:
 metadata:
 labels:
 machine.openshift.io/cluster-api-cluster: ocp-xxxy
 machine.openshift.io/cluster-api-machine-role: worker
 machine.openshift.io/cluster-api-machine-type: worker
 machine.openshift.io/cluster-api-machineset: ocp-xxxy-worker-us-west-2b
 spec:
 providerSpec:
 ...output omitted...
 placement:
 availabilityZone: us-west-2b
 region: us-west-2
 credentialsSecret:
 name: aws-cloud-credentials
 instanceType: m4.xlarge
 ...output omitted...
```

**참고**

확장 외에도 시스템 집합에 대한 변경 사항은 기존 시스템에 영향을 주지 않습니다. 새 시스템은 새 레이블 또는 업데이트된 레이블과 같은 시스템 집합에 대한 변경 사항을 상속합니다.

**MachineAutoscalers**

**MachineAutoscalers**는 자동으로 확장 가능한 시스템 집합을 정의합니다. 수동 확장에는 시스템 자동 확장기가 필요하지 않습니다.

**ClusterAutoscaler**

**ClusterAutoscaler** 리소스를 사용하면 클라우드 배포에서 시스템을 자동으로 확장할 수 있습니다. 자동 확장을 사용하면 계산 노드가 응답하지 않는 경우를 확인하고 가용성 향상을 위해 포드를 다른 계산 노드로 신속하게 이동할 수 있습니다. 자동 확장은 클러스터의 부하가 높은 경우 또는 컴퓨팅 요구 사항이 변경된 워크로드에 유용합니다.

클러스터 자동 확장기 리소스를 사용하면 노드, 코어 및 메모리와 같은 다양한 구성 요소에 대한 제한 사항을 지정할 수 있습니다. 수동 확장에는 클러스터 자동 확장기가 필요하지 않습니다.

**참고**

다음 섹션에는 **MachineAutoscalers** 및 **ClusterAutoscaler** 리소스가 모두 설명되어 있습니다.

**MachineHealthChecks**

**MachineHealthCheck**는 시스템(예: 계산 노드)의 상태를 확인하고 리소스가 비정상인 경우 조치를 취합니다.

**계산 노드 수동 확장**

클러스터 확장은 수동 및 자동의 두 가지 방법으로 수행할 수 있습니다. 자동 확장 가능 여부는 프로바이더에 따라 다르며 전체 스택 자동화 메서드(IPI)로 지원됩니다. 클러스터 확장은 특정 워크로드에 대한 계산 노드 집합을 전담하는 등 리소스 혼잡을 최소화하고 리소스 관리를 개선하기 위해 수행됩니다.

배포 및 배포 구성과 마찬가지로 시스템 집합 리소스에서 지정한 복제본을 조정하여 인스턴스 수를 변경할 수 있습니다. 시스템 집합 리소스의 **replicas** 행을 수정하거나 **oc scale** 명령을 사용합니다.

```
[user@demo ~]$ oc scale --replicas 2 \
> machineset MACHINE-SET -n openshift-machine-api
```

시스템 집합의 시스템 수가 복제본 수와 일치하지 않는 경우, OpenShift는 원하는 복제본 수에 도달할 때까지 시스템을 추가하거나 제거합니다.

시스템 집합을 축소하면 제거 대상 시스템에서 실행되는 포드를 이동하도록 스케줄러에 지시합니다. 시스템 복제본 수는 나머지 계산 노드의 용량이 충분하여 포드를 실행할 수 있는 경우에만 줄일 수 있습니다. 리소스가 충분하지 않으면 컨트롤러에서 시스템 복제본 수를 줄이는 것을 거부합니다.



### 참조

계산 노드의 크기를 수동으로 조정하는 방법에 대한 자세한 내용은 다음 주소에 있는 Red Hat OpenShift Container Platform 4.5 시스템 관리 설명서의 MachineSet 수동 확장장을 참조하십시오.

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/machine\\_management/index#manually-scaling-machineset](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/machine_management/index#manually-scaling-machineset)

# OpenShift 클러스터 자동 확장

## 목표

이 섹션을 마치면 매개 변수를 정의하여 클러스터의 크기를 자동으로 제어할 수 있습니다.

## 클러스터 자동 확장

시스템 API는 클러스터의 워크로드를 관리하는 데 필요한 여러 리소스를 제공합니다. 클러스터 리소스를 수동 및 자동의 두 가지 방식으로 확장할 수 있습니다.

수동 확장을 위해서는 시스템 집합의 복제본 수를 업데이트해야 합니다. 클러스터 자동 확장에는 두 가지 사용자 지정 리소스(**MachineAutoscaler** 및 **ClusterAutoscaler**)를 사용하는 작업이 포함됩니다.



### 중요

수동 또는 자동 확장을 수행하려면 시스템 API 운영자가 작동 중이어야 합니다. 시스템 API 운영자를 사용하려면 클라우드 프로바이더를 통합해야 하며, 풀스택 자동화 방법으로 설치된 클러스터에서 사용할 수 있습니다.

시스템 자동 확장기 리소스는 부하에 따라 시스템 집합의 복제본 수를 자동으로 조정합니다. 이 API 리소스는 시스템 집합과 상호 작용하고 시스템 집합에 계산 노드를 추가 또는 제거하도록 지시합니다. 시스템 자동 확장기 리소스는 하한 및 상한에 대한 정의를 지원합니다. 클러스터 자동 확장기 리소스는 전체 클러스터에 대한 제한 사항을 적용합니다(예: 총 노드 수). 예를 들어 **MaxNodesTotal**은 전체 클러스터의 최대 노드 수를 설정하고 **MaxMemoryTotal**은 전체 클러스터에서 최대 메모리를 설정합니다. 그런 다음 필요한 경우 부하가 감소하면 노드 수를 줄이도록 클러스터 자동 확장기를 구성합니다.

각 OpenShift 클러스터에는 클러스터 자동 확장기 리소스가 한 개만 있을 수 있습니다. 클러스터 자동 확장기 리소스는 더 높은 수준에서 작동하며 최대 노드 수와 코어, 메모리 및 GPU(그래픽 처리 장치) 등의 기타 리소스를 정의합니다. 이러한 제한을 통해 시스템 자동 확장기 리소스에서 제어되지 않은 방식으로 확장하는 것을 방지할 수 있습니다.



### 참고

클러스터에 대한 **ClusterAutoscaler** 리소스를 정의해야 합니다. 그렇지 않으면 자동 확장이 작동하지 않습니다.

다음 밸류는 클러스터 자동 확장기 리소스에 대해 설명합니다.

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
 name: "default"
spec:
 podPriorityThreshold: -10
 resourceLimits:
 maxNodesTotal: 6
 scaleDown:
```

```
enabled: true
delayAfterAdd: 3m
unneededTime: 3m
```

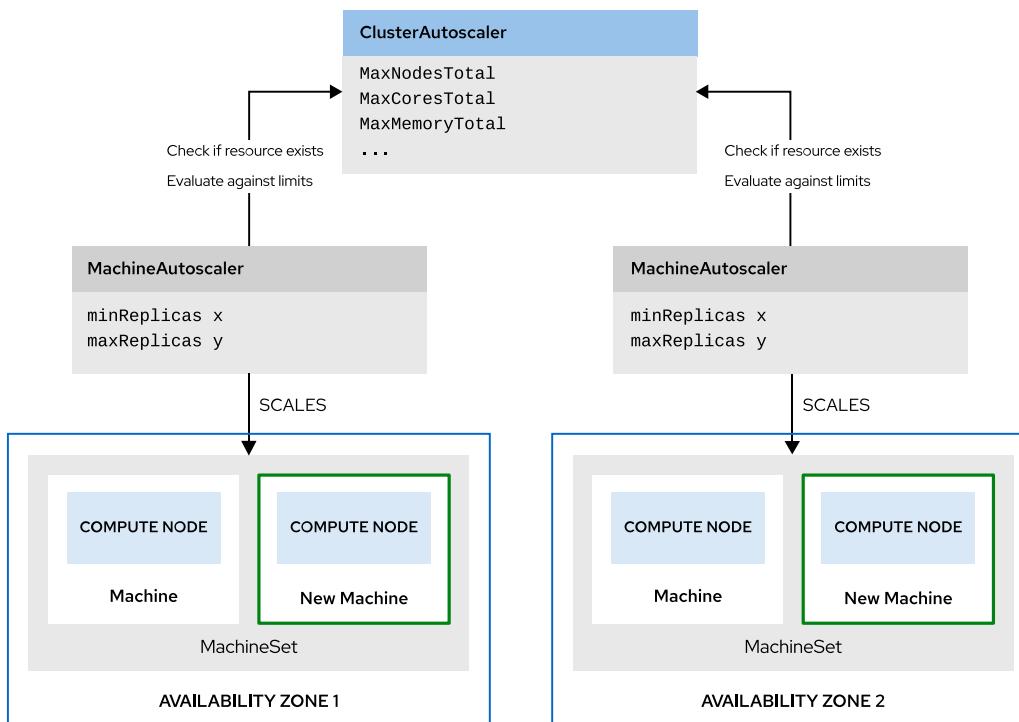
시스템 자동 확장기 리소스를 사용하여 시스템 집합 리소스에서 정의한 시스템 수를 확장합니다. 확장은 클러스터 자동 확장기 리소스에서 정의한 값을 초과하지 않고 새 시스템을 추가하는 경우에만 작동합니다. 예를 들어 AWS에서 하나의 **m4.xlarge** 시스템을 추가하면 노드 1개, CPU 코어 4개, 16GB 메모리가 추가됩니다. 시스템 자동 확장기는 클러스터 자동 확장이 허용되지 않는 한 시스템 수를 자체적으로 확장하거나 축소하지 않습니다.

다음 다이어그램에서는 시스템 자동 확장기 리소스가 시스템 집합과 상호 작용하여 계산 노드의 수를 조정하는 방법을 보여줍니다.

- 시스템 자동 확장기 리소스를 확장해야 하는 경우 클러스터 자동 확장기 리소스가 있는지 확인합니다. 해당 리소스가 없으면 확장이 수행되지 않습니다.

클러스터 자동 확장기 리소스가 있으면 시스템 자동 확장기 리소스에서 새 시스템을 추가하는 것이 클러스터 자동 확장기 리소스에 정의된 제한 사항을 위반하는지 평가합니다.

- 요청이 제한을 초과하지 않으면 새 시스템이 생성됩니다.
- 새 시스템이 준비되면 OpenShift가 여기에 포드를 새 노드로 예약합니다.



**그림 D.2: 전체 스택 자동화를 통한 동적 확장**

시스템 자동 확장기와 시스템 집합 리소스는 1:1로 매핑됩니다. 이러한 매핑을 통해 각 시스템 자동 확장기 리소스에서 자체 시스템 집합을 관리할 수 있습니다. 시스템 집합을 확장할 수 있도록 하려면 해당 기능을 위한 시스템 자동 확장기 리소스를 생성합니다. 크기를 조정하지 않아야 하는 시스템 집합의 경우 시스템 자동 확장기 리소스를 만들지 마십시오.

기본 전체 스택 자동화 배포에서는 각 시스템 집합 리소스가 가용성 영역에 매핑되지만 이 설계는 임의적입니다. 예를 들어 시스템 집합 리소스를 사용하여 디스크 속도 또는 기능 역할에 따라 계산 노드를 분리할 수 있습니다.

## 자동 확장 구현

자동 확장을 성공적으로 수행하려면 다음이 필요합니다.

- 풀스택 자동화 방법으로 배포된 클러스터. 자동 확장은 계산 노드를 추가하거나 제거할 때 클라우드 서비스와 상호 작용해야 하기 때문입니다.
- 인프라에서 지원하는 경우 클러스터 자동 확장기 리소스. 또한 클러스터 자동 확장기 리소스에서 최대 노드 수를 제한하고 코어, 메모리, GPU에 대한 최소 및 최대 값을 정의할 수도 있습니다. 클러스터 자동 확장기 리소스의 `scaleDown` 섹션에 있는 `enabled: true` 항목에서는 사용하지 않을 때 시스템 수를 자동으로 축소할 수 있도록 이 리소스에 권한을 부여합니다.
- 하나 이상의 시스템 자동 확장기 리소스. 각 시스템 자동 확장기 리소스는 특정 시스템 집합에 대한 최소 및 최대 복제본 수를 정의합니다.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
 name: "MACHINE-AUTOSCALER-NAME"
 namespace: "openshift-machine-api"
spec:
 minReplicas: 1
 maxReplicas: 4
 scaleTargetRef:
 apiVersion: machine.openshift.io/v1beta1 kind: MachineSet
 name: MACHINE-SET-NAME
```

이러한 리소스를 만든 후 클러스터에서 부하를 관리할 수 없는 경우 계산 노드 자동 추가가 트리거됩니다.

클러스터 자동 확장기 리소스는 리소스가 부족하여 현재 노드를 예약하지 못하거나 배포 요구 사항을 충족하는데 다른 노드가 필요한 경우 계산 노드 수를 늘립니다.



### 중요

OpenShift는 클러스터 리소스를 `ClusterAutoscaler` 리소스에 지정한 제한 이상으로 늘리지는 않습니다.



### 참조

클러스터 자동 확장에 대한 자세한 내용은

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.5/html-single/machine\\_management/index#applying-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/machine_management/index#applying-autoscaling)  
에 있는 Red Hat OpenShift Container Platform 4.5 시스템 관리 설명서의 OpenShift Container Platform 클러스터에 자동 확장 적용 장을 참조하십시오.

### OpenShift 4 클러스터 확장

<https://github.com/openshift/training/blob/master/docs/04-scaling-cluster.md>

## 요약

이 부록에서는 다음 내용을 배웠습니다.

- OpenShift Container Platform에서는 클러스터의 계산 노드 수를 자동으로 늘리고 줄이는 메커니즘을 제공합니다. 수동으로 또는 자동으로 클러스터의 크기를 조정할 수 있습니다.
- 수동으로 클러스터의 크기를 조정하는 경우 **시스템** 리소스를 제어하는 **MachineSets** 리소스와 상호 작용합니다. **시스템**은 노드에 매팅됩니다.
- 클러스터의 크기를 자동으로 조정하는 경우 **MachineAutoscaler** 리소스와 상호 작용하여 **MachineSet** 리소스에서 복제본 수를 업데이트하도록 지시합니다.
- **ClusterAutoscaler** 리소스는 **MachineAutoscalers**에서 만들 수 있는 최대 리소스 수를 제어합니다. 이를 통해 클러스터의 제어되지 않은 증가를 방지할 수 있습니다.

