



# OpenShift Container Platform 4.15

## 아키텍처

OpenShift Container Platform의 아키텍처 개요



# OpenShift Container Platform 4.15 아키텍처

---

OpenShift Container Platform의 아키텍처 개요

## Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

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

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

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

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

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

이 문서에서는 OpenShift Container Platform의 플랫폼 및 애플리케이션 아키텍처에 관한 개요를 제공합니다.

## Table of Contents

<b>1장. 아키텍처 개요</b>	<b>4</b>
1.1. OPENSIFT CONTAINER PLATFORM 아키텍처의 일반 용어집	4
1.2. 설치 및 업데이트 정보	8
1.3. 컨트롤 플레인 정보	8
1.4. 개발자를 위한 컨테이너화된 애플리케이션 정보	8
1.5. RHCOS(RED HAT ENTERPRISE LINUX COREOS) 및 IGNITION 정보	9
1.6. 승인 플러그인 정보	9
<b>2장. OPENSIFT CONTAINER PLATFORM 아키텍처</b>	<b>10</b>
2.1. OPENSIFT CONTAINER PLATFORM 소개	10
<b>3장. 설치 및 업데이트</b>	<b>16</b>
3.1. OPENSIFT CONTAINER PLATFORM 설치 정보	16
3.2. OPENSIFT 업데이트 서비스 정보	23
3.3. 관리되지 않는 OPERATOR에 대한 지원 정책	24
3.4. 다음 단계	25
<b>4장. RED HAT OPENSIFT CLUSTER MANAGER</b>	<b>26</b>
4.1. RED HAT OPENSIFT CLUSTER MANAGER에 액세스	26
4.2. 일반 작업	26
4.3. 클러스터 탭	26
4.4. 추가 리소스	29
<b>5장. KUBERNETES OPERATOR의 다중 클러스터 엔진 정보</b>	<b>30</b>
5.1. OPENSIFT CONTAINER PLATFORM에서 다중 클러스터 엔진으로 클러스터 관리	30
5.2. RED HAT ADVANCED CLUSTER MANAGEMENT를 통한 클러스터 관리	30
5.3. 추가 리소스	30
<b>6장. 컨트롤 플레인 아키텍처</b>	<b>31</b>
6.1. 머신 구성 풀을 사용한 노드 구성 관리	31
6.2. OPENSIFT CONTAINER PLATFORM의 머신 역할	32
6.3. OPENSIFT CONTAINER PLATFORM의 OPERATOR	35
6.4. ETCD 개요	37
6.5. 호스트된 컨트롤 플레인 소개	38
<b>7장. NVIDIA GPU 아키텍처 개요</b>	<b>42</b>
7.1. NVIDIA GPU 사전 요구 사항	42
7.2. NVIDIA GPU 사용	42
7.3. GPU 공유 방법	46
7.4. OPENSIFT CONTAINER PLATFORM 용 NVIDIA GPU 기능	48
<b>8장. OPENSIFT CONTAINER PLATFORM 개발 이해</b>	<b>50</b>
8.1. 컨테이너화된 애플리케이션 개발 정보	50
8.2. 간단한 컨테이너 빌드	50
8.3. OPENSIFT CONTAINER PLATFORM용 KUBERNETES 매니페스트 생성	53
8.4. OPERATOR용 개발	55
<b>9장. RHCOS(RED HAT ENTERPRISE LINUX COREOS)</b>	<b>57</b>
9.1. RHCOS 소개	57
9.2. IGNITION 구성 파일 보기	61
9.3. 설치 후 IGNITION 구성 변경	63
<b>10장. 승인 플러그인</b>	<b>65</b>
10.1. 승인 플러그인 정보	65

10.2. 기본 승인 플러그인	65
10.3. WEBHOOK 승인 플러그인	68
10.4. 웹 후크 승인 플러그인의 유형	69
10.5. 동적 승인 구성	72
10.6. 추가 리소스	79



# 1장. 아키텍처 개요

OpenShift Container Platform은 클라우드 기반 Kubernetes 컨테이너 플랫폼입니다. OpenShift Container Platform의 기초는 Kubernetes를 기반으로 하므로 동일한 기술을 공유합니다. OpenShift Container Platform 및 Kubernetes에 대한 자세한 내용은 [제품 아키텍처](#)를 참조하십시오.

## 1.1. OPENSIFT CONTAINER PLATFORM 아키텍처의 일반 용어집

이 용어집은 아키텍처 콘텐츠에 사용되는 일반적인 용어를 정의합니다.

### 액세스 정책

클러스터 내의 사용자, 애플리케이션 및 엔터티가 서로 상호 작용하는 방식을 지정하는 일련의 역할입니다. 액세스 정책은 클러스터 보안을 강화합니다.

### 승인 플러그인

승인 플러그인은 보안 정책, 리소스 제한 또는 구성 요구 사항을 적용합니다.

### 인증

OpenShift Container Platform 클러스터에 대한 액세스를 제어하기 위해 클러스터 관리자는 승인된 사용자만 클러스터에 액세스할 수 있도록 사용자 인증을 구성할 수 있습니다. OpenShift Container Platform 클러스터와 상호 작용하려면 OpenShift Container Platform API로 인증해야 합니다. OpenShift Container Platform API에 대한 요청에 OAuth 액세스 토큰 또는 X.509 클라이언트 인증서를 제공하여 인증할 수 있습니다.

### 부트스트랩

최소한의 Kubernetes를 실행하고 OpenShift Container Platform 컨트롤 플레인을 배포하는 임시 시스템입니다.

### CSR(인증서 서명 요청)

리소스는 표시된 서명자가 인증서에 서명하도록 요청합니다. 이 요청은 승인되거나 거부될 수 있습니다.

### Cluster Version Operator (CVO)

OpenShift Container Platform Update Service를 사용하여 현재 구성 요소 버전 및 그래프의 정보를 기반으로 유효한 업데이트 및 업데이트 경로를 확인하는 Operator입니다.

### 컴퓨팅 노드

클러스터 사용자에게 대한 워크로드를 실행하는 노드입니다. 컴퓨팅 노드는 작업자 노드라고도 합니다.

### 구성 드리프트

노드의 구성이 머신 구성에서 지정하는 것과 일치하지 않는 경우입니다.

### 컨테이너

소프트웨어와 모든 종속 항목으로 구성된 경량 및 실행 가능한 이미지입니다. 컨테이너는 운영 체제를 가상화하므로 데이터 센터, 퍼블릭 또는 프라이빗 클라우드, 로컬 호스트와 같은 컨테이너를 어디에서나 실행할 수 있습니다.

### 컨테이너 오케스트레이션 엔진

컨테이너의 배포, 관리, 확장 및 네트워킹을 자동화하는 소프트웨어입니다.

### 컨테이너 워크로드

컨테이너에 패키지 및 배포되는 애플리케이션입니다.

### 컨트롤그룹(cgroups)

프로세스의 소비를 관리하고 제한하기 위해 프로세스의 파티션 세트를 그룹으로 분할합니다.

### 컨트롤 플레인



컨테이너의 라이프사이클을 정의, 배포, 관리하기 위해 API 및 인터페이스를 노출하는 컨테이너 오케스트레이션 계층입니다. 컨트롤 플레인, 컨트롤 플레인 시스템이라고도 합니다.

## CRI-O

운영 체제와 통합되는 Kubernetes 네이티브 컨테이너 런타임 구현으로 효율적인 Kubernetes 환경을 제공합니다.

## 배포

애플리케이션의 라이프사이클을 유지 관리하는 Kubernetes 리소스 오브젝트입니다.

## Dockerfile

터미널에서 이미지를 어셈블하기 위해 수행할 사용자 명령이 포함된 텍스트 파일입니다.

## 호스트 컨트롤 플레인

데이터 플레인 및 작업자의 OpenShift Container Platform 클러스터에서 컨트롤 플레인을 호스팅할 수 있는 OpenShift Container Platform 기능입니다. 이 모델은 다음 작업을 수행합니다.

- 컨트롤 플레인에 필요한 인프라 비용을 최적화합니다.
- 클러스터 생성 시간을 개선합니다.
- Kubernetes 기본 상위 수준 프리미티브를 사용하여 컨트롤 플레인 호스팅을 활성화합니다. 예를 들어 배포 및 상태 저장 세트가 있습니다.
- 컨트롤 플레인과 워크로드 간 강력한 네트워크 분할을 허용합니다.

## 하이브리드 클라우드 배포

베어 메탈, 가상, 프라이빗 및 퍼블릭 클라우드 환경 전체에 일관된 플랫폼을 제공하는 배포. 이를 통해 속도, 민첩성 및 이식성을 제공합니다.

## Ignition

초기 구성 중에 RHCOS가 디스크를 조작하는 데 사용하는 유틸리티입니다. 디스크 파티셔닝, 파티션 포맷, 파일 작성 및 사용자 구성을 포함한 일반적인 디스크 작업을 완료합니다.

## 설치 프로그램에서 제공하는 인프라

설치 프로그램은 클러스터가 실행되는 인프라를 배포하고 구성합니다.

## kubelet

Pod에서 컨테이너가 실행 중인지 확인하기 위해 클러스터의 각 노드에서 실행되는 기본 노드 에이전트입니다.

## Kubernetes 매니페스트

JSON 또는 YAML 형식의 Kubernetes API 오브젝트의 사양입니다. 구성 파일에는 배포, 구성 맵, 시크릿, 데몬 세트가 포함될 수 있습니다.

## MCD(Machine Config Daemon)

노드에서 구성 드리프트가 있는지 정기적으로 확인하는 데몬입니다.

## Machine Config Operator (MCO)

클러스터 머신에 새 구성을 적용하는 Operator입니다.

## 머신 구성 풀(MCP)

컨트롤 플레인 구성 요소 또는 사용자 워크로드와 같은 머신 그룹을 처리하는 리소스를 기반으로 합니다.

## 메타데이터

클러스터 배포 아티팩트에 대한 추가 정보입니다.

## 마이크로 서비스

소프트웨어 작성을 위한 접근 방식. 애플리케이션은 마이크로 서비스를 사용하여 서로 독립적으로 가장 작은 구성 요소로 분리할 수 있습니다.

### **미러 레지스트리**

OpenShift Container Platform 이미지의 미러가 있는 레지스트리입니다.

### **모놀리식 애플리케이션**

자체 포함, 빌드 및 패키징된 애플리케이션입니다.

### **네임스페이스**

네임스페이스는 모든 프로세스에 표시되는 특정 시스템 리소스를 격리합니다. 네임스페이스 내에서 해당 네임스페이스의 멤버인 프로세스만 해당 리소스를 볼 수 있습니다.

### **네트워킹**

OpenShift Container Platform 클러스터의 네트워크 정보입니다.

### **노드**

OpenShift Container Platform 클러스터의 작업자 시스템입니다. 노드는 VM(가상 머신) 또는 물리적 머신입니다.

### **OpenShift CLI(oc)**

터미널에서 OpenShift Container Platform 명령을 실행하는 명령줄 툴입니다.

### **OpenShift Dedicated**

AWS(Amazon Web Services) 및 Google Cloud에서 관리형 RHEL OpenShift Container Platform. OpenShift Dedicated는 애플리케이션 빌드 및 확장에 중점을 둡니다.

### **OSUS(OpenShift Update Service)**

인터넷에 액세스할 수 있는 클러스터의 경우 RHEL(Red Hat Enterprise Linux)은 공용 API 뒤에 있는 호스팅 서비스로 OpenShift 업데이트 서비스를 사용하여 무선 업데이트를 제공합니다.

### **OpenShift 이미지 레지스트리**

이미지를 관리하기 위해 OpenShift Container Platform에서 제공하는 레지스트리입니다.

### **Operator**

OpenShift Container Platform 클러스터에서 Kubernetes 애플리케이션을 패키징, 배포 및 관리하는 기본 방법입니다. Operator는 사람의 운영 지식을 패키징하고 고객과 공유하는 소프트웨어로 인코딩합니다.

### **OperatorHub**

설치할 다양한 OpenShift Container Platform Operator가 포함된 플랫폼입니다.

### **OLM(Operator Lifecycle Manager)**

OLM은 Kubernetes 네이티브 애플리케이션의 라이프사이클을 설치, 업데이트 및 관리할 수 있도록 지원합니다. OLM은 효과적이고 자동화되고 확장 가능한 방식으로 Operator를 관리하도록 설계된 오픈 소스 툴킷입니다.

### **ostree**

전체 파일 시스템 트리 트리의 원자 업그레이드를 수행하는 Linux 기반 운영 체제의 업그레이드 시스템입니다. ostree는 주소 지정 가능 오브젝트 저장소를 사용하여 파일 시스템 트리에 대한 의미 있는 변경을 추적하고 기존 패키지 관리 시스템을 보완하도록 설계되었습니다.

### **OTA(Over-the-Air) 업데이트**

OSUS(OpenShift Container Platform Update Service)는 RHCOS(Red Hat Enterprise Linux CoreOS)를 포함하여 OpenShift Container Platform에 대한 무선 업데이트를 제공합니다.

### **Pod**

OpenShift Container Platform 클러스터에서 실행되는 볼륨 및 IP 주소와 같은 공유 리소스가 있는 하나 이상의 컨테이너입니다. Pod는 정의, 배포 및 관리되는 최소 컴퓨팅 단위입니다.

### **프라이빗 레지스트리**

OpenShift Container Platform은 컨테이너 이미지 레지스트리 API를 구현하는 모든 서버를 이미지의 소스로 사용하여 개발자가 개인 컨테이너 이미지를 푸시하고 가져올 수 있습니다.

### 퍼블릭 레지스트리

OpenShift Container Platform은 컨테이너 이미지 레지스트리 API를 구현하는 모든 서버를 이미지의 소스로 사용하여 개발자가 공용 컨테이너 이미지를 푸시하고 가져올 수 있습니다.

### RHEL OpenShift Container Platform Cluster Manager

OpenShift Container Platform 클러스터를 설치, 수정, 운영 및 업그레이드할 수 있는 관리형 서비스입니다.

### RHEL Quay 컨테이너 레지스트리

대부분의 컨테이너 이미지 및 Operator를 OpenShift Container Platform 클러스터에 제공하는 Quay.io 컨테이너 레지스트리입니다.

### 복제 컨트롤러

한 번에 실행하는 데 필요한 Pod 복제본 수를 나타내는 자산입니다.

### RBAC(역할 기반 액세스 제어)

클러스터 사용자와 워크로드가 역할을 실행하는 데 필요한 리소스에만 액세스할 수 있도록 하는 주요 보안 제어입니다.

### 라우트

경로는 OpenShift Container Platform 인스턴스 외부의 사용자 및 애플리케이션에서 Pod에 대한 네트워크 액세스를 허용하는 서비스를 노출합니다.

### 스케일링

리소스 용량이 증가하거나 감소합니다.

### 서비스

서비스는 Pod 집합에서 실행 중인 애플리케이션을 노출합니다.

### S2I(Source-to-Image) 이미지

애플리케이션을 배포하기 위해 OpenShift Container Platform의 애플리케이션 소스 코드의 프로그래밍 언어를 기반으로 생성된 이미지입니다.

### 스토리지

OpenShift Container Platform은 온프레미스 및 클라우드 공급자를 위해 다양한 유형의 스토리지를 지원합니다. OpenShift Container Platform 클러스터에서 영구 및 비영구 데이터에 대한 컨테이너 스토리지를 관리할 수 있습니다.

### telemetry

OpenShift Container Platform의 크기, 상태 및 상태와 같은 정보를 수집하는 구성 요소입니다.

### 템플릿

템플릿은 오브젝트 세트를 설명합니다. 이러한 오브젝트를 매개변수화하고 처리하여 OpenShift Container Platform에서 생성할 오브젝트 목록을 만들 수 있습니다.

### 사용자 프로비저닝 인프라

사용자가 제공하는 인프라에 OpenShift Container Platform을 설치할 수 있습니다. 설치 프로그램을 사용하여 클러스터 인프라를 프로비저닝하고 클러스터 인프라를 생성한 다음 제공한 인프라에 클러스터를 배포하는 데 필요한 자산을 생성할 수 있습니다.

### 웹 콘솔

OpenShift Container Platform을 관리할 UI(사용자 인터페이스)입니다.

### 작업자 노드

클러스터 사용자에 대한 워크로드를 실행하는 노드입니다. 작업자 노드는 컴퓨팅 노드라고도 합니다.

### 추가 리소스

- 네트워킹에 대한 자세한 내용은 [OpenShift Container Platform 네트워킹](#) 을 참조하십시오.
- 스토리지에 대한 자세한 내용은 [OpenShift Container Platform 스토리지](#) 를 참조하십시오.
- 인증에 대한 자세한 내용은 [OpenShift Container Platform 인증](#) 을 참조하십시오.
- OLM (Operator Lifecycle Manager)에 대한 자세한 내용은 OLM을 참조하십시오.
- 로깅에 대한 자세한 내용은 로깅 정보를 참조하십시오.  
[https://docs.redhat.com/en/documentation/openshift\\_container\\_platform/4.15/html-single/logging/#cluster-logging](https://docs.redhat.com/en/documentation/openshift_container_platform/4.15/html-single/logging/#cluster-logging)
- OTA(Over-the-air) 업데이트에 대한 자세한 내용은 [OpenShift 업데이트 소개](#) 를 참조하십시오.

## 1.2. 설치 및 업데이트 정보

클러스터 관리자는 OpenShift Container Platform [설치 프로그램](#) 을 사용하여 다음 방법 중 하나를 사용하여 클러스터를 설치하고 배포할 수 있습니다.

- 설치 관리자 프로비저닝 인프라
- 사용자 프로비저닝 인프라

## 1.3. 컨트롤 플레인 정보

[컨트롤 플레인](#)은 클러스터의 작업자 노드 및 Pod를 관리합니다. MCP(Machine config pool)를 사용하여 노드를 구성할 수 있습니다. MCP는 컨트롤 플레인 구성 요소 또는 사용자 워크로드와 같은 머신 그룹으로, 처리하는 리소스를 기반으로 합니다. OpenShift Container Platform은 호스트에 다양한 역할을 할당합니다. 이러한 역할은 클러스터에서 머신의 기능을 정의합니다. 클러스터에는 표준 컨트롤 플레인 및 작업자 역할 유형에 대한 정의가 포함되어 있습니다.

Operator를 사용하여 컨트롤 플레인에서 서비스를 패키지, 배포, 관리할 수 있습니다. Operator는 다음 서비스를 제공하므로 OpenShift Container Platform에서 중요한 구성 요소입니다.

- 상태 점검 수행
- 애플리케이션 조사 방법 제공
- 무선(Over-the-Air) 업데이트 관리
- 애플리케이션이 지정된 상태로 유지됨

## 1.4. 개발자를 위한 컨테이너화된 애플리케이션 정보

개발자는 다양한 툴, 방법 및 형식을 사용하여 고유한 요구 사항에 따라 [컨테이너화된 애플리케이션](#)을 개발할 수 있습니다.

- 간단한 컨테이너 애플리케이션을 빌드하려면 다양한 빌드 툴, 기본 이미지 및 레지스트리 옵션을 사용합니다.
- OperatorHub 및 템플릿과 같은 지원 구성 요소를 사용하여 애플리케이션을 개발합니다.
- 애플리케이션을 Operator로 패키징하고 배포합니다.

Kubernetes 매니페스트를 생성하여 Git 리포지토리에 저장할 수도 있습니다. Kubernetes는 pod라는 기본 단위에서 작동합니다. Pod는 클러스터에서 실행 중인 프로세스의 단일 인스턴스입니다. Pod에는 하나 이

상의 컨테이너가 포함될 수 있습니다. Pod 세트와 액세스 정책을 그룹화하여 서비스를 생성할 수 있습니다. 서비스는 다른 애플리케이션이 Pod를 생성하고 삭제할 때 사용할 영구 내부 IP 주소 및 호스트 이름을 제공합니다. Kubernetes는 애플리케이션 유형에 따라 워크로드를 정의합니다.

## 1.5. RHCOS(RED HAT ENTERPRISE LINUX COREOS) 및 IGNITION 정보

클러스터 관리자는 다음 RHCOS(Red Hat Enterprise Linux CoreOS) 작업을 수행할 수 있습니다.

- [차세대 단일 목적 컨테이너 운영 체제 기술에](#) 대해 알아보십시오.
- RHCOS(Red Hat Enterprise Linux CoreOS) 구성 방법 선택
- RHCOS(Red Hat Enterprise Linux CoreOS) 배포 방법을 선택합니다.
  - 설치 프로그램에서 제공하는 배포
  - 사용자 프로비저닝 배포

OpenShift Container Platform 설치 프로그램은 클러스터를 배포하는 데 필요한 Ignition 구성 파일을 생성합니다. RHCOS(Red Hat Enterprise Linux CoreOS)는 초기 구성 중에 Ignition을 사용하여 파티셔닝, 포맷, 파일 작성, 사용자 구성과 같은 일반적인 디스크 작업을 수행합니다. 첫 번째 부팅 중에 Ignition은 설치 미디어 또는 사용자가 지정한 위치에서 구성을 읽고 해당 구성을 머신에 적용합니다.

[Ignition 작동 방법](#), OpenShift Container Platform 클러스터에서 RHCOS (Red Hat Enterprise Linux CoreOS) 머신의 프로세스, Ignition 구성 파일을 보고 설치 후 Ignition 구성을 변경할 수 있습니다.

## 1.6. 승인 플러그인 정보

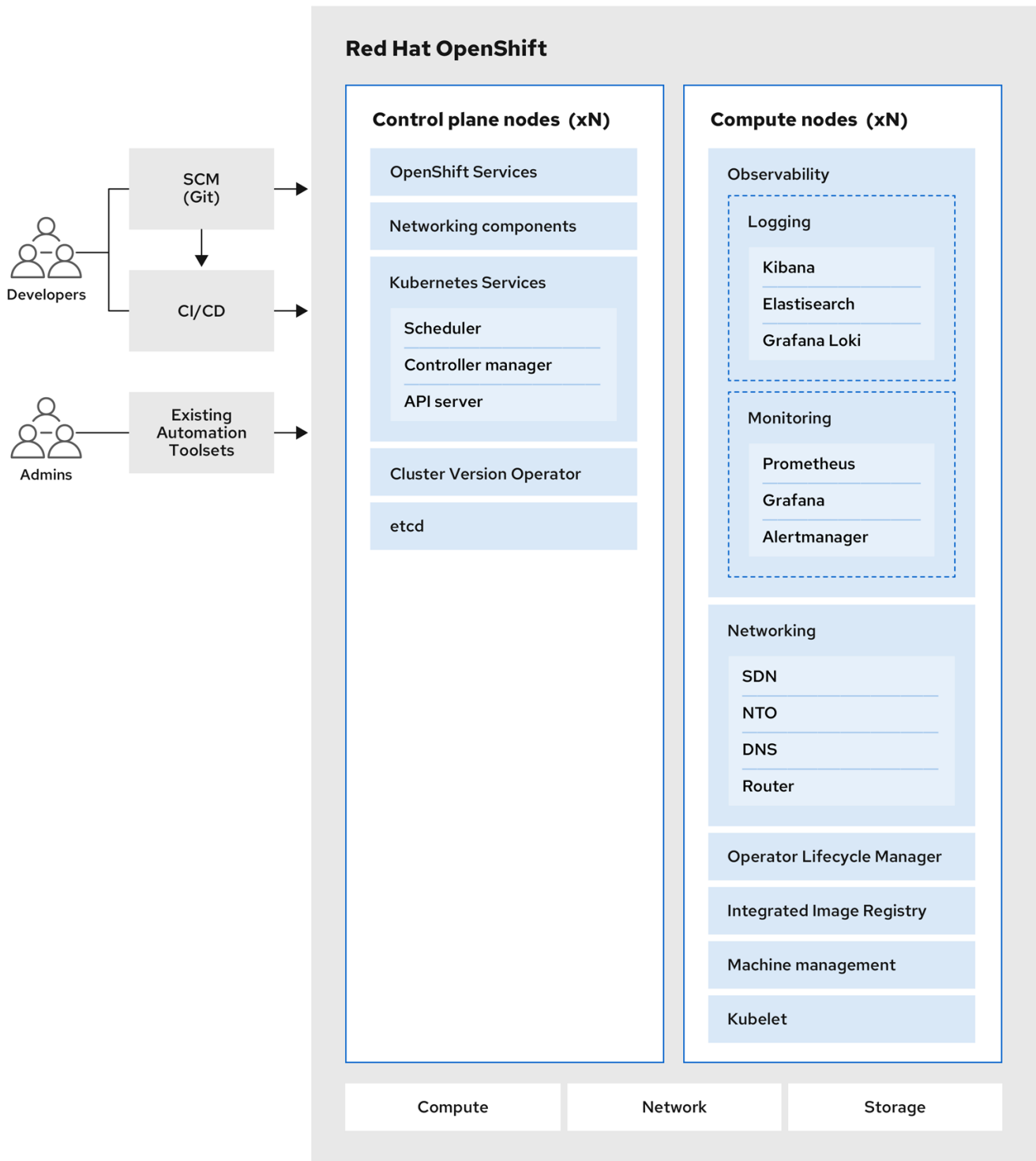
[승인 플러그인](#) 을 사용하여 OpenShift Container Platform의 기능을 규제할 수 있습니다. 리소스 요청이 인증되고 승인된 후 승인 플러그인은 마스터 API에 대한 리소스 요청을 가로채서 리소스 요청의 유효성을 검사하고 확장 정책이 준수되도록 합니다. 승인 플러그인은 보안 정책, 리소스 제한, 구성 요구 사항 및 기타 설정을 적용하는 데 사용됩니다.

## 2장. OPENSIFT CONTAINER PLATFORM 아키텍처

### 2.1. OPENSIFT CONTAINER PLATFORM 소개

OpenShift Container Platform은 컨테이너화된 애플리케이션을 개발하고 실행하기 위한 플랫폼입니다. 애플리케이션 및 애플리케이션을 지원하는 데이터센터를 몇 대의 머신 및 애플리케이션에서 수백만 클라이언트에 서비스를 제공하는 수천 대의 컴퓨터로 확장 가능하도록 설계되었습니다.

Kubernetes에 기반을 둔 OpenShift Container Platform은 대규모 통신, 스트리밍 비디오, 게임, बैं킹 및 기타 애플리케이션의 엔진 역할을 하는 동일한 기술을 통합합니다. Red Hat의 오픈 기술로 구현하면 컨테이너화된 애플리케이션을 단일 클라우드에서 온프레미스 및 다중 클라우드 환경으로 확장할 수 있습니다.



475\_OpenShift\_0824

### 2.1.1. Kubernetes 정보

컨테이너 이미지와 컨테이너 이미지에서 실행되는 컨테이너는 최신 애플리케이션 개발을 위한 기본 빌딩 블록이지만 규모에 맞게 실행하려면 안정적이고 유연한 배포 시스템이 필요합니다. Kubernetes는 컨테이너를 오케스트레이션하는 사실상의 표준입니다.

Kubernetes는 컨테이너화된 애플리케이션의 배포, 확장 및 관리를 자동화하기 위한 오픈 소스 컨테이너 오케스트레이션 엔진입니다. Kubernetes의 일반적인 개념은 다음과 같이 매우 간단합니다.

- 하나 이상의 작업자 노드에서 시작하여 컨테이너 워크로드를 실행합니다.

- 하나 이상의 컨트롤 플레인 노드에서 해당 워크로드의 배포를 관리합니다.
- 컨테이너를 포드라는 배포 단위로 래핑합니다. 포드를 사용하면 컨테이너에 추가 메타데이터가 제공되고 단일 배포 엔티티에서 여러 컨테이너를 그룹화할 수 있습니다.
- 특별한 종류의 자산을 생성합니다. 예를 들어 서비스는 포드 세트와 포드에 액세스하는 방법을 정의하는 정책으로 표시됩니다. 이 정책을 통해 컨테이너는 서비스에 대한 특정 IP 주소가 없어도 필요한 서비스에 연결할 수 있습니다. 복제 컨트롤러는 한 번에 실행하는 데 필요한 포드 복제본 수를 나타내는 또 다른 특수 자산입니다. 이 기능을 사용하여 현재 수요에 맞게 애플리케이션을 자동으로 확장할 수 있습니다.

불과 몇 년 만에 Kubernetes는 대규모 클라우드와 온프레미스에서 채택되었습니다. 오픈 소스 개발 모델을 사용하면 많은 사용자가 네트워킹, 스토리지 및 인증과 같은 구성 요소에 다른 기술을 구현하여 Kubernetes를 확장할 수 있습니다.

## 2.1.2. 컨테이너화된 애플리케이션의 이점

컨테이너화된 애플리케이션을 사용하면 기존 배포 방법을 사용하는 것보다 많은 이점이 있습니다. 애플리케이션이 모든 종속 항목을 포함하는 운영 체제에 한 번 설치될 것으로 예상되는 경우 컨테이너를 통해 애플리케이션이 해당 애플리케이션과 함께 종속 항목을 유지할 수 있습니다. 컨테이너화된 애플리케이션을 생성하면 많은 이점이 있습니다.

### 2.1.2.1. 운영 체제에서의 이점

컨테이너에서는 커널이 없는 소규모의 전용 Linux 운영 체제를 사용합니다. 파일 시스템, 네트워킹, cgroup, 프로세스 테이블 및 네임스페이스는 호스트 Linux 시스템과 별개이지만 필요한 경우 컨테이너를 호스트에 완벽하게 통합할 수 있습니다. 컨테이너는 Linux 기반이므로 빠르게 혁신되는 오픈 소스 개발 모델에 함께 제공되는 모든 이점을 사용할 수 있습니다.

각 컨테이너는 전용 운영 체제를 사용하므로 충돌하는 소프트웨어 종속 항목이 필요한 애플리케이션을 동일한 호스트에 배포할 수 있습니다. 각 컨테이너에서는 자체 종속 소프트웨어를 제공하고 네트워킹 및 파일 시스템과 같은 자체 인터페이스를 관리하므로 애플리케이션이 해당 자산과 경쟁할 필요가 없습니다.

### 2.1.2.2. 배포 및 스케일링에서의 이점

애플리케이션의 주요 릴리스 간에 롤링 업그레이드를 사용하는 경우 다운타임 없이 애플리케이션을 지속적으로 개선하고 현재 릴리스와의 호환성을 계속 유지할 수 있습니다.

기존 버전과 더불어 새 버전의 애플리케이션을 배포하고 테스트할 수도 있습니다. 컨테이너가 테스트를 통과하면 새 컨테이너를 추가로 배포하고 이전 컨테이너를 제거하면 됩니다.

애플리케이션의 모든 소프트웨어 종속 항목은 컨테이너 자체 내에서 해결되므로 데이터센터의 각 호스트에서 표준화된 운영 체제를 사용할 수 있습니다. 애플리케이션 호스트마다 특정 운영 체제를 구성할 필요가 없습니다. 데이터센터에 더 많은 용량이 필요하다면 또 다른 일반 호스트 시스템을 배포하면 됩니다.

컨테이너화된 애플리케이션도 이와 마찬가지로 간단하게 스케일링할 수 있습니다. OpenShift Container Platform에서는 컨테이너화된 서비스를 스케일링하는 간단한 표준 방법을 제공합니다. 예를 들어, 대규모 모놀리식 애플리케이션이 아닌 일련의 마이크로서비스로 애플리케이션을 빌드하는 경우 요구에 맞게 각 마이크로서비스를 개별적으로 스케일링할 수 있습니다. 이 기능을 사용하면 전체 애플리케이션이 아니라 필요한 서비스만 스케일링할 수 있으므로 최소한의 리소스만 사용하여 애플리케이션 요구사항을 충족할 수 있습니다.

## 2.1.3. OpenShift Container Platform 개요



OpenShift Container Platform은 Kubernetes에 다음과 같은 향상된 기능을 포함한 엔터프라이즈급 개선 사항을 제공합니다.

- 하이브리드 클라우드 배포. 다양한 퍼블릭 클라우드 플랫폼 또는 데이터센터에 OpenShift Container Platform 클러스터를 배포할 수 있습니다.
- 통합된 Red Hat 기술. OpenShift Container Platform의 주요 구성 요소는 RHEL(Red Hat Enterprise Linux) 및 관련 Red Hat 기술을 기반으로 합니다. OpenShift Container Platform은 Red Hat의 엔터프라이즈급 소프트웨어에 대한 강력한 테스트 및 인증 이니셔티브의 이점을 제공합니다.
- 오픈 소스 개발 모델. 개발은 공개적으로 완료되었으며 소스 코드는 공개 소프트웨어 리포지토리에서 구할 수 있습니다. 이 오픈 협업을 통해 빠른 혁신과 개발을 촉진할 수 있습니다.

Kubernetes는 애플리케이션 관리에 뛰어나지만 플랫폼 수준 요구사항 또는 배포 프로세스를 지정하거나 관리하지는 않습니다. 강력하고 유연한 플랫폼 관리 도구 및 프로세스는 OpenShift Container Platform 4.15에서 제공하는 중요한 이점입니다. 다음 섹션에서는 OpenShift Container Platform의 고유한 기능과 이점에 대해 설명합니다.

### 2.1.3.1. 사용자 정의 운영 체제

OpenShift Container Platform은 OpenShift Container Platform에서 컨테이너화된 애플리케이션을 실행하도록 특별히 설계된 컨테이너 지향 운영 체제인 RHCOS(Red Hat Enterprise Linux CoreOS)를 사용하며 새로운 툴을 사용하여 빠른 설치, Operator 기반 관리 및 간소화된 업그레이드를 제공합니다.

RHCOS는 다음을 포함합니다.

- OpenShift Container Platform이 머신을 처음 시작하고 구성하기 위한 최초 부팅 시스템 구성으로 사용하는 Ignition
- 운영 체제와 밀접하게 통합되어 효율적이고 최적화된 Kubernetes 환경을 제공하는 Kubernetes 기본 컨테이너 런타임 구현인 CRI-O. CRI-O는 컨테이너 실행, 중지 및 다시 시작 기능을 제공합니다. OpenShift Container Platform 3에서 사용하던 Docker Container Engine을 완전히 대체합니다.
- 컨테이너 시작 및 모니터링을 담당하는 Kubernetes의 기본 노드 에이전트인 Kubelet

OpenShift Container Platform 4.15에서는 모든 컨트롤 플레인 머신에 RHCOS를 사용해야 하지만 작업자 머신이라고도 하는 컴퓨팅 머신의 운영 체제로 RHEL(Red Hat Enterprise Linux)을 사용할 수 있습니다. RHEL 작업자를 사용하도록 선택한 경우 모든 클러스터 머신에 RHCOS를 사용할 때보다 시스템 유지보수를 더 많이 수행해야 합니다.

### 2.1.3.2. 단순화된 설치 및 업데이트 프로세스

OpenShift Container Platform 4.15를 사용하면 적절한 권한이 있는 계정이 있는 경우 단일 명령을 실행하고 몇 가지 값을 제공하여 지원되는 클라우드에 프로덕션 클러스터를 배포할 수 있습니다. 지원되는 플랫폼을 사용하는 경우 클라우드 설치를 사용자 정의하거나 데이터센터에 클러스터를 설치할 수도 있습니다.

모든 머신, 업데이트 또는 업그레이드에 RHCOS를 사용하는 클러스터의 경우 OpenShift Container Platform은 고도로 자동화된 간단한 프로세스입니다. OpenShift Container Platform은 운영 체제 자체를 포함하여 각 머신에서 실행되는 시스템 및 서비스를 중앙 컨트롤 플레인에서 완전히 제어하므로 업그레이드가 자동 이벤트가 되도록 설계되었습니다. 클러스터에 RHEL 작업자 머신이 포함된 경우 컨트롤 플레인 은 간소화된 업데이트 프로세스를 통해 이점을 얻을 수 있지만 RHEL 머신을 업그레이드하려면 추가 작업을 수행해야 합니다.

### 2.1.3.3. 기타 주요 기능

Operator는 OpenShift Container Platform 4.15 코드 베이스의 기본 단위이자 애플리케이션에서 사용할 애플리케이션 및 소프트웨어 구성 요소를 편리하게 배포할 수 있는 방법입니다. OpenShift Container Platform에서 Operator는 플랫폼 기반 역할을 하며 운영 체제 및 컨트롤 플레인 애플리케이션을 수동으로 업그레이드할 필요가 없습니다. Cluster Version Operator 및 Machine Config Operator와 같은 OpenShift Container Platform Operator를 사용하면 중요한 구성 요소를 클러스터 전체에서 간단하게 관리할 수 있습니다.

OLM(Operator Lifecycle Manager)과 OperatorHub에서는 애플리케이션을 개발 및 배포하는 사용자에게 Operator를 저장하고 배포하는 기능을 제공합니다.

Red Hat Quay Container Registry는 대부분의 컨테이너 이미지와 Operator를 OpenShift Container Platform 클러스터에 제공하는 Quay.io 컨테이너 레지스트리입니다. Quay.io는 수백만 개의 이미지와 태그를 저장하는 Red Hat Quay의 공개 레지스트리 버전입니다.

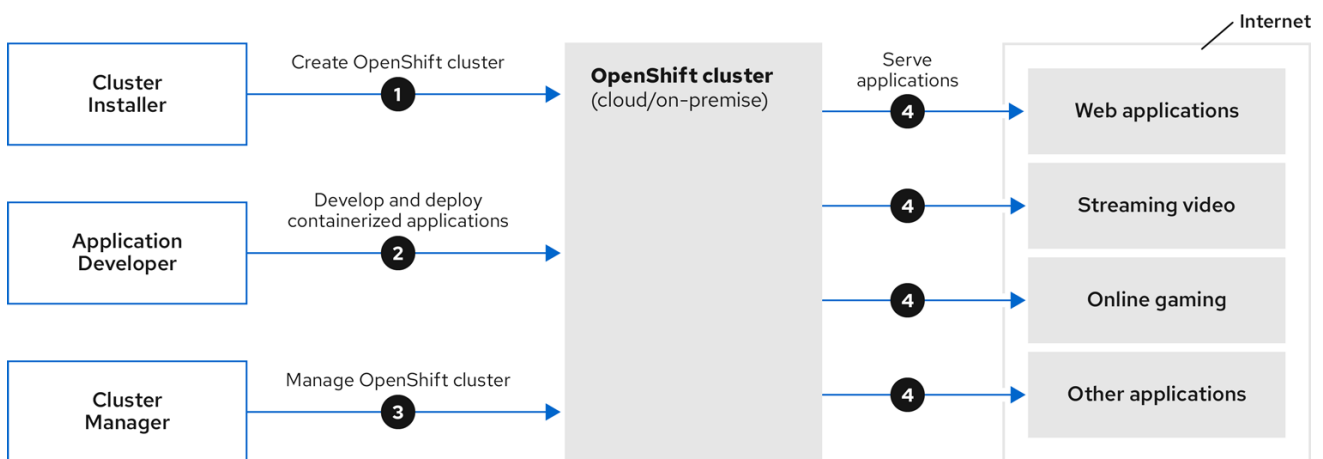
OpenShift Container Platform의 Kubernetes의 기타 향상된 기능에는 소프트웨어 정의 네트워킹(SDN), 인증, 로그 집계, 모니터링 및 라우팅 개선이 포함됩니다. OpenShift Container Platform에서는 포괄적인 웹 콘솔과 사용자 정의 OpenShift CLI(**oc**) 인터페이스도 제공합니다.

#### 2.1.3.4. OpenShift Container Platform 라이프사이클

다음 그림에서는 기본 OpenShift Container Platform 라이프사이클을 보여줍니다.

- OpenShift Container Platform 클러스터 생성
- 클러스터 관리
- 애플리케이션 개발 및 배포
- 애플리케이션 스케일링

그림 2.1. 상위 레벨 OpenShift Container Platform 개요



475\_OpenShift\_0824

#### 2.1.4. OpenShift Container Platform 용 인터넷 액세스

OpenShift Container Platform 4.15에서 클러스터를 설치하려면 인터넷 액세스가 필요합니다.

다음의 경우 인터넷 액세스가 필요합니다.

- [OpenShift Cluster Manager](#) 에 액세스하여 설치 프로그램을 다운로드하고 서브스크립션 관리를 수행합니다. 클러스터가 인터넷에 액세스할 수 있고 Telemetry 서비스를 비활성화하지 않은 경우, 클러스터에 자동으로 권한이 부여됩니다.
- [Quay.io](#)에 액세스. 클러스터를 설치하는 데 필요한 패키지를 받을 수 있습니다.
- 클러스터 업데이트를 수행하는 데 필요한 패키지를 받을 수 있습니다.



### 중요

클러스터가 직접 인터넷에 액세스할 수 없는 경우, 프로비저닝하는 일부 유형의 인프라에서 제한된 네트워크 설치를 수행할 수 있습니다. 이 프로세스 동안 필요한 콘텐츠를 다운로드하고 이를 사용하여 설치 패키지로 미러 레지스트리를 채웁니다. 설치 유형에 따라서는 클러스터를 설치하는 환경에 인터넷 액세스가 필요하지 않을 수도 있습니다. 클러스터를 업데이트하기 전에 미러 레지스트리의 내용을 업데이트합니다.

## 3장. 설치 및 업데이트

### 3.1. OPENSIFT CONTAINER PLATFORM 설치 정보

OpenShift Container Platform 설치 프로그램은 클러스터를 배포하는 네 가지 방법을 제공하며, 다음 목록에서 자세히 설명합니다:

- **대화형:** 웹 기반 [지원 설치 관리자](#)를 사용하여 클러스터를 배포할 수 있습니다. 이는 인터넷에 연결된 네트워크가 있는 클러스터에 이상적인 방법입니다. 지원 설치 프로그램은 OpenShift Container Platform을 설치하는 가장 쉬운 방법이며, 스마트 기본값을 제공하며 클러스터를 설치하기 전에 사전 진행 중 검증을 수행합니다. 자동화 및 고급 구성 시나리오를 위한 RESTful API도 제공합니다.
- **로컬 에이전트 기반:** 연결이 끊긴 환경 또는 제한된 네트워크를 위해 에이전트 기반 설치 관리자를 사용하여 로컬로 클러스터를 배포할 수 있습니다. 지원 설치 관리자의 많은 이점을 제공하지만 먼저 [에이전트 기반 설치 관리자](#)를 다운로드하고 구성해야 합니다. 구성은 명령줄 인터페이스를 사용하여 수행됩니다. 이 방법은 연결이 끊긴 환경에 이상적입니다.
- **자동화:** 설치 관리자 프로비저닝 인프라에 클러스터를 배포할 수 있습니다. 설치 프로그램은 프로비저닝에 각 클러스터 호스트의 BMC(Baseboard Management Controller)를 사용합니다. 연결되거나 연결이 끊긴 환경에서 클러스터를 배포할 수 있습니다.
- **전체 제어:** 준비 및 유지 관리하는 인프라에 클러스터를 배포하여 최대 사용자 지정 가능성을 제공할 수 있습니다. 연결되거나 연결이 끊긴 환경에서 클러스터를 배포할 수 있습니다.

각 방법은 다음과 같은 특성을 가진 클러스터를 배포합니다.

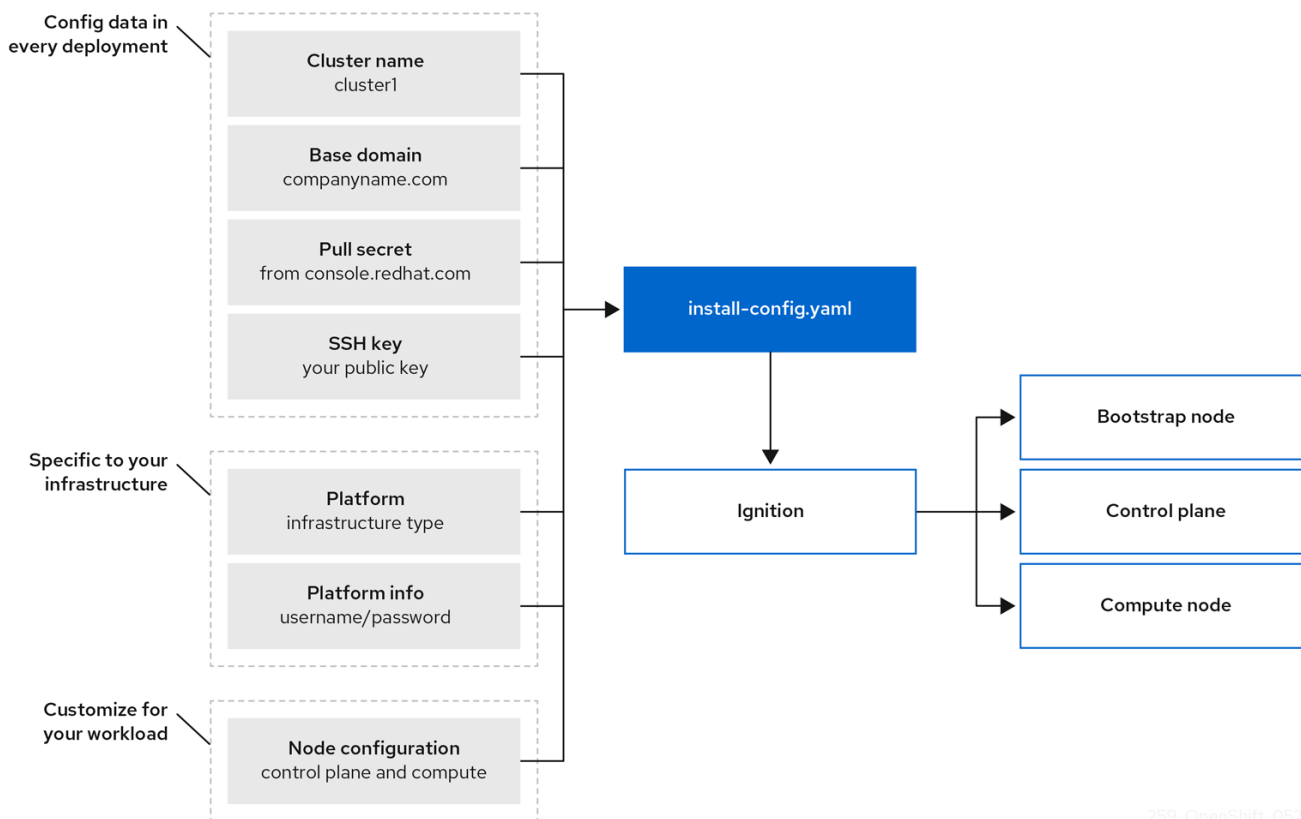
- 기본적으로 사용 가능한 단일 장애 지점이 없는고가용성 인프라입니다.
- 관리자는 적용되는 업데이트 및 시기를 제어할 수 있습니다.

#### 3.1.1. 설치 프로그램 정보

설치 프로그램을 사용하여 각 유형의 클러스터를 배포할 수 있습니다. 설치 프로그램은 부트스트랩, 컨트롤 플레인 및 컴퓨팅 머신의 Ignition 구성 파일과 같은 주요 자산을 생성합니다. 인프라를 올바르게 구성한 경우 세 가지 머신 구성으로 OpenShift Container Platform 클러스터를 시작할 수 있습니다.

OpenShift Container Platform 설치 프로그램은 일련의 대상 및 종속 항목을 사용하여 클러스터 설치를 관리합니다. 설치 프로그램에는 달성해야 할 대상 세트가 있으며 각 대상에는 종속 항목 세트가 있습니다. 각 대상은 자체 종속 항목에만 관련되므로 설치 프로그램은 최종 대상과 동시에 여러 대상을 실행 중인 클러스터가 되도록 할 수 있습니다. 설치 프로그램은 종속성을 충족하기 때문에 명령을 실행하여 다시 생성하는 대신 기존 구성 요소를 인식하고 사용합니다.

그림 3.1. OpenShift Container Platform 설치 대상 및 종속 항목



259\_OpenShift\_0522

### 3.1.2. RHCOS(Red Hat Enterprise Linux CoreOS) 정보

설치 후 각 클러스터 시스템은 운영 체제로 RHCOS (Red Hat Enterprise Linux CoreOS)를 사용합니다. RHCOS는 RHEL(Red Hat Enterprise Linux)의 변경 불가능한 컨테이너 호스트 버전이며 기본적으로 SELinux가 활성화된 RHEL 커널을 제공합니다. RHCOS에는 Kubernetes 노드 에이전트인 **kubelet** 과 Kubernetes에 최적화된 CRI-O 컨테이너 런타임이 포함됩니다.

OpenShift Container Platform 4.15 클러스터의 모든 컨트롤 플레인 시스템은 Ignition이라는 중요한 최초 부팅 프로비저닝 도구를 포함하는 RHCOS를 사용해야 합니다. 이 도구를 사용하면 클러스터가 머신을 구성할 수 있습니다. 운영 체제 업데이트는 **OSTree** 를 백엔드로 사용하여 Machine Config Operator에 의해 클러스터 전체에 배포되는 부팅 가능한 컨테이너 이미지로 제공됩니다. 실제 운영 체제 변경은 **rpm-ostree** 를 사용하여 각 머신에서 원자 작업으로 수행됩니다. 이러한 기술을 함께 사용하면 OpenShift Container Platform에서 전체 플랫폼을 최신 상태로 유지하는 인플레이스 업그레이드를 통해 클러스터의 다른 애플리케이션을 관리하는 것처럼 운영 체제를 관리할 수 있습니다. 이러한 내부 업데이트는 운영 팀의 부담을 줄일 수 있습니다.

모든 클러스터 머신의 운영 체제로 RHCOS를 사용하는 경우 클러스터가 운영 체제를 포함한 구성 요소 및 머신의 모든 측면을 관리합니다. 그러면 설치 프로그램 및 Machine Config Operator만 머신을 변경할 수 있습니다. 설치 프로그램에서는 Ignition 구성 파일을 사용하여 각 머신의 정확한 상태를 설정하고 Machine Config Operator는 설치 후 새 인증서 또는 키 적용과 같은 머신에 대한 추가 변경을 완료합니다.

### 3.1.3. OpenShift Container Platform 클러스터에서 지원되는 플랫폼

OpenShift Container Platform 4.15에서는 다음 플랫폼에서 설치 관리자 프로비저닝 인프라를 사용하는 클러스터를 설치할 수 있습니다.

- Alibaba Cloud
- AWS(Amazon Web Services)

- 베어 메탈
- GCP(Google Cloud Platform)
- IBM Cloud®
- Microsoft Azure
- Microsoft Azure Stack Hub
- Nutanix
- Red Hat OpenStack Platform (RHOSP)
  - 최신 OpenShift Container Platform 릴리스는 최신 RHOSP 긴 수명 릴리스 및 중간 릴리스를 모두 지원합니다. 완전한 RHOSP 릴리스 호환성에 대해서는 [RHOSP의 OpenShift Container Platform on RHOSP 지원 매트릭스](#)를 참조하십시오.
- VMware vSphere

이러한 클러스터의 경우, 설치 프로세스를 실행하는 컴퓨터를 포함한 모든 머신은 플랫폼 컨테이너의 이미지를 가져오고 원격 분석 데이터를 Red Hat에 제공하기 위해 인터넷에 직접 액세스할 수 있어야 합니다.



### 중요

설치 후 다음 변경 사항은 지원되지 않습니다.

- 클라우드 공급자 플랫폼 혼합.
- 클라우드 공급자 구성 요소 혼합. 예를 들어 클러스터를 설치한 플랫폼의 다른 플랫폼의 영구 스토리지 프레임워크를 사용합니다.

OpenShift Container Platform 4.15에서는 다음 플랫폼에서 사용자 프로비저닝 인프라를 사용하는 클러스터를 설치할 수 있습니다.

- AWS
- Azure
- Azure Stack Hub
- 베어 메탈
- GCP
- IBM Power®
- IBM Z® 또는 IBM® LinuxONE
- RHOSP
  - 최신 OpenShift Container Platform 릴리스는 최신 RHOSP 긴 수명 릴리스 및 중간 릴리스를 모두 지원합니다. 완전한 RHOSP 릴리스 호환성에 대해서는 [RHOSP의 OpenShift Container Platform on RHOSP 지원 매트릭스](#)를 참조하십시오.
- AWS의 VMware Cloud

- VMware vSphere

플랫폼의 지원되는 사례에 따라 사용자 프로비저닝 인프라에서 설치를 수행하여 완전한 인터넷 액세스 권한이 있는 머신을 실행하거나 클러스터를 프록시 뒤에 배치하거나, 연결이 끊긴 설치를 수행할 수 있습니다.

연결이 끊긴 설치에서는 클러스터를 설치하는 데 필요한 이미지를 다운로드하여 미러 레지스트리에 배치한 다음 해당 데이터를 사용하여 클러스터를 설치할 수 있습니다. vSphere 또는 베어 메탈 인프라에 연결이 끊긴 설치와 함께 플랫폼 컨테이너의 이미지를 가져오려면 인터넷 액세스가 필요하지만 클러스터 시스템에는 직접 인터넷 액세스가 필요하지 않습니다.

다른 플랫폼의 통합 테스트에 대한 자세한 내용은 [OpenShift Container Platform 4.x 통합 테스트](#) 페이지를 참조하십시오.

### 3.1.4. 설치 프로세스

지원 설치 관리자를 제외하고 OpenShift Container Platform 클러스터를 설치할 때 OpenShift [Cluster Manager Hybrid Cloud Console](#)의 [적절한 클러스터 유형](#) 페이지에서 설치 프로그램을 다운로드해야 합니다. 이 콘솔은 다음을 관리합니다.

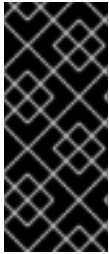
- 계정용 REST API.
- 필수 구성 요소를 가져오는 데 사용하는 풀 시크릿인 레지스트리 토큰입니다.
- 사용 지표 수집이 용이하도록 클러스터 ID를 Red Hat 계정에 연결하는 클러스터 등록.

OpenShift Container Platform 4.15에서 설치 프로그램은 자산 세트에서 일련의 파일 변환을 수행하는 Go 바이너리 파일입니다. 설치 프로그램과 상호 작용하는 방법은 설치 유형에 따라 다릅니다. 다음 설치 사용 사례를 고려하십시오.

- 지원 설치 관리자를 사용하여 클러스터를 배포하려면 [지원 설치 관리자](#)를 사용하여 클러스터 설정을 구성해야 합니다. 다운로드 및 구성할 설치 프로그램이 없습니다. 클러스터 구성 설정을 완료한 후 검색 ISO를 다운로드한 다음 해당 이미지를 사용하여 클러스터 시스템을 부팅합니다. Nutanix, vSphere 및 베어 메탈에 지원 설치 관리자를 사용하여 클러스터를 완전히 통합 및 기타 플랫폼에 설치할 수 있습니다. 베어 메탈에 설치하는 경우 네트워킹, 로드 밸런싱, 스토리지 및 개별 클러스터 머신을 포함한 모든 클러스터 인프라 및 리소스를 제공해야 합니다.
- 에이전트 기반 설치 관리자를 사용하여 클러스터를 배포하려면 먼저 [에이전트 기반 설치 관리자](#)를 다운로드할 수 있습니다. 그런 다음 클러스터를 구성하고 검색 이미지를 생성할 수 있습니다. 검색 이미지를 사용하여 클러스터 시스템을 부팅합니다. 이 에이전트는 설치 프로그램과 상호 작용하거나 프로비저너 시스템을 직접 설정하는 대신 설치 프로그램과 통신하는 에이전트를 처리합니다. 네트워킹, 로드 밸런싱, 스토리지 및 개별 클러스터 시스템을 포함하여 모든 클러스터 인프라 및 리소스를 제공해야 합니다. 이 방법은 연결이 끊긴 환경에 이상적입니다.
- 설치 관리자 프로비저닝 인프라가 있는 클러스터의 경우 인프라 부트스트랩 및 프로비저닝을 직접 수행하는 대신 설치 프로그램에 위임합니다. 설치 프로그램은 베어 메탈에 설치하는 경우를 제외하고 클러스터를 지원하는 데 필요한 모든 네트워킹, 머신 및 운영 체제를 생성합니다. 베어 메탈에 설치하는 경우 부트스트랩 머신, 네트워킹, 로드 밸런싱, 스토리지 및 개별 클러스터 머신을 포함한 모든 클러스터 인프라 및 리소스를 제공해야 합니다.
- 클러스터의 인프라를 프로비저닝하고 관리하는 경우 부트스트랩 머신, 네트워킹, 부하 분산, 스토리지 및 개별 클러스터 머신을 포함한 모든 클러스터 인프라 및 리소스를 제공해야 합니다.

설치 프로그램의 경우 프로그램은 설치 중에 `install-config.yaml`, Kubernetes 매니페스트 및 머신 유형에 대한 Ignition 구성 파일의 세 가지 파일 세트를 사용합니다.



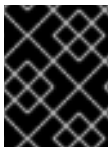


### 중요

설치 중에 기본 RHCOS 운영 체제를 제어하는 Kubernetes 및 Ignition 구성 파일을 수정할 수 있습니다. 그러나 이러한 오브젝트를 수정한 내용이 적합한지 확인할 수 있는 유효성 검사는 없습니다. 이러한 오브젝트를 수정하면 클러스터가 작동하지 않을 수 있습니다. 이 위험 때문에 문서화된 절차를 따르거나 Red Hat 지원 부서에서 지시하지 않는 한 Kubernetes 및 Ignition 구성 파일 수정은 지원되지 않습니다.

설치 구성 파일은 Kubernetes 매니페스트로 변환된 다음 매니페스트가 Ignition 구성 파일로 래핑됩니다. 설치 프로그램은 이러한 Ignition 구성 파일을 사용하여 클러스터를 생성합니다.

설치 프로그램을 실행할 때 설치 구성 파일이 모두 정리되므로 다시 사용하려는 모든 구성 파일을 백업하십시오.



### 중요

설치 중에 설정한 매개변수는 수정할 수 없지만 설치 후에는 많은 클러스터 속성을 수정할 수 있습니다.

#### 3.1.4.1. 지원 설치 관리자를 사용한 설치 프로세스

[지원 설치 관리자](#)를 사용하여 설치하려면 웹 기반 사용자 인터페이스 또는 RESTful API를 사용하여 대화식으로 클러스터 구성을 생성해야 합니다. 지원 설치 관리자 사용자 인터페이스에서 필요한 값을 입력하라는 메시지를 표시하고 사용자 인터페이스 또는 API에서 변경하지 않는 한 나머지 매개변수에 대한 적절한 기본값을 제공합니다. 지원 설치 프로그램은 다운로드하여 클러스터 시스템을 부팅하는 데 사용하는 검색 이미지를 생성합니다. 이 이미지는 RHCOS 및 에이전트를 설치하고 에이전트가 프로비저닝을 처리합니다. 지원 설치 관리자를 사용하여 OpenShift Container Platform을 설치하고 Nutanix, vSphere 및 베어 메탈에서 전체 통합을 설치할 수 있습니다. 또한 통합 없이 다른 플랫폼에 지원 설치 프로그램을 사용하여 OpenShift Container Platform을 설치할 수 있습니다.

OpenShift Container Platform은 운영 체제 자체를 포함하여 클러스터의 모든 측면을 관리합니다. 각 머신은 결합하는 클러스터에서 호스팅되는 리소스를 참조하는 구성으로 부팅됩니다. 이 구성을 사용하면 업데이트가 적용될 때 클러스터가 자체적으로 관리될 수 있습니다.

가능한 경우 지원 설치 관리자 기능을 사용하여 에이전트 기반 설치 관리자를 다운로드하고 구성할 필요가 없습니다.

#### 3.1.4.2. 에이전트 기반 인프라를 사용하는 설치 프로세스

에이전트 기반 설치에는 [에이전트 기반 설치 관리자](#)를 다운로드하여 설치해야 한다는 점을 제외하고 지원 설치 관리자를 사용하는 것과 유사합니다. 에이전트 기반 설치에는 지원 설치 프로그램의 편의를 원하지만 연결이 끊긴 환경에 클러스터를 설치해야 하는 경우 유용합니다.

가능한 경우 에이전트 기반 설치 기능을 사용하여 부트스트랩 VM으로 프로비저너 시스템을 생성한 다음 클러스터 인프라를 프로비저닝 및 유지 관리할 필요가 없습니다.

#### 3.1.4.3. 설치 관리자가 프로비저닝한 인프라를 사용하는 설치 프로세스

기본 설치 유형에서는 설치 관리자 프로비저닝 인프라를 사용합니다. 기본적으로 설치 프로그램은 설치 마법사 역할을 하여 자체적으로 결정할 수 없는 값을 입력하라는 메시지를 표시하고 나머지 매개변수에 대한 적절한 기본값을 제공합니다. 고급 인프라 시나리오를 지원하도록 설치 프로세스를 사용자 정의할 수도 있습니다. 설치 프로그램은 클러스터의 기본 인프라를 프로비저닝합니다.

표준 클러스터 또는 사용자 정의된 클러스터를 설치할 수 있습니다. 표준 클러스터에서는 클러스터를 설치하는 데 필요한 최소 세부 정보를 제공합니다. 사용자 지정 클러스터를 사용하면 컨트롤 플레인에서 사



용하는 머신 수, 클러스터가 배포하는 가상 머신 유형 또는 Kubernetes 서비스 네트워크의 CIDR 범위와 같은 플랫폼에 대한 세부 정보를 지정할 수 있습니다.

가능하면 이 기능을 사용하여 클러스터 인프라를 프로비저닝 및 유지보수하지 않아도 됩니다. 다른 모든 환경에서는 설치 프로그램을 사용하여 클러스터 인프라를 프로비저닝하는 데 필요한 자산을 생성합니다.

OpenShift Container Platform은 설치 프로그램에서 프로비저닝한 인프라 클러스터를 통해 운영 체제 자체를 포함하여 클러스터의 모든 측면을 관리합니다. 각 머신은 결합하는 클러스터에서 호스팅되는 리소스를 참조하는 구성으로 부팅됩니다. 이 구성을 사용하면 업데이트가 적용될 때 클러스터가 자체적으로 관리될 수 있습니다.

#### 3.1.4.4. 사용자 프로비저닝 인프라를 사용하는 설치 프로세스

제공하는 인프라에 OpenShift Container Platform도 설치할 수 있습니다. 설치 프로그램을 사용하여 클러스터 인프라를 프로비저닝하고 클러스터 인프라를 생성한 다음 제공한 인프라에 클러스터를 배포하는 데 필요한 자산을 생성합니다.

설치 프로그램이 프로비저닝한 인프라를 사용하지 않는 경우 클러스터 리소스를 직접 관리하고 유지보수해야 합니다. 다음 목록에서는 이러한 자체 관리 리소스 중 일부를 자세히 설명합니다.

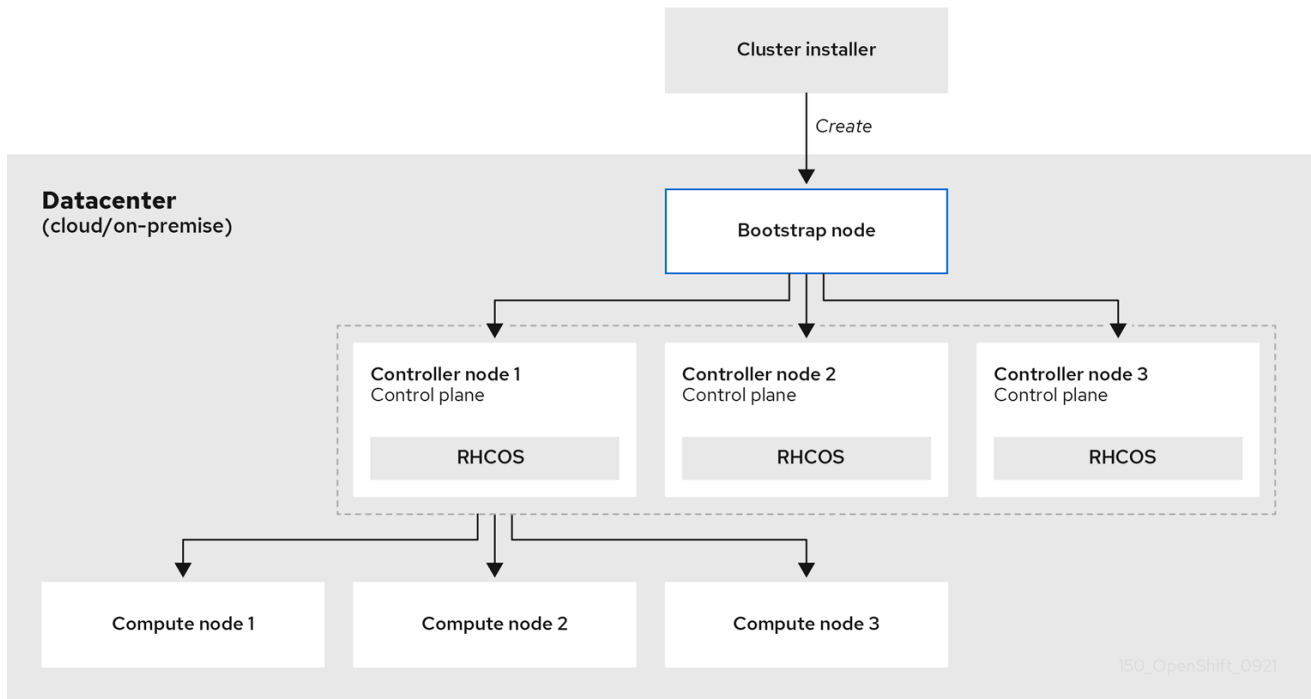
- 클러스터를 구성하는 컨트롤 플레인 및 컴퓨팅 머신의 기본 인프라
- 로드 밸런서
- DNS 레코드 및 필수 서브넷을 포함한 클러스터 네트워킹
- 클러스터 인프라 및 애플리케이션용 스토리지

클러스터가 사용자 프로비저닝 인프라를 사용하는 경우 RHEL 컴퓨팅 머신을 클러스터에 추가할 수 있습니다.

#### 3.1.4.5. 설치 프로세스 세부사항

클러스터가 프로비저닝되면 클러스터의 각 시스템에 클러스터에 대한 정보가 필요합니다. OpenShift Container Platform은 초기 구성 중에 임시 부트스트랩 머신을 사용하여 필요한 정보를 영구 컨트롤 플레인에 제공합니다. 임시 부트스트랩 머신은 클러스터 생성 방법을 설명하는 Ignition 구성 파일을 사용하여 부팅됩니다. 부트스트랩 시스템은 컨트롤 플레인을 구성하는 컨트롤 플레인 시스템을 생성합니다. 그런 다음 컨트롤 플레인 시스템에서는 작업자 머신이라고도 하는 컴퓨팅 머신을 만듭니다. 다음 그림은 이 프로세스를 보여줍니다.

그림 3.2. 부트스트랩, 컨트롤 플레인 및 컴퓨팅 시스템 생성



### 중요

클러스터를 배포하려는 동안 성능 및 확장성에 대한 권장 사례, 특히 etcd 스토리지에 대한 입력/출력 대기 시간 요구 사항 및 권장 컨트롤 플레인 노드 크기 조정에 대한 요구 사항을 숙지해야 합니다. 자세한 내용은 "Recommended etcd practices" 및 "Control Plane 노드 크기 조정"을 참조하십시오.

클러스터 머신이 초기화되면 부트스트랩 머신이 손상됩니다. 모든 클러스터에서는 부트스트랩 프로세스를 사용하여 클러스터를 초기화하지만, 클러스터의 인프라를 프로비저닝하는 경우 많은 단계를 수동으로 완료해야 합니다.

### 중요

- 설치 프로그램에서 생성하는 Ignition 구성 파일에 24시간 후에 만료되는 인증서가 포함되어 있습니다. 이 인증서는 그 후에 갱신됩니다. 인증서를 갱신하기 전에 클러스터가 종료되고 24시간이 지난 후에 클러스터가 다시 시작되면 클러스터는 만료된 인증서를 자동으로 복구합니다. 예외적으로 kubelet 인증서를 복구하려면 대기 중인 **node-bootstrapper** 인증서 서명 요청(CSR)을 수동으로 승인해야 합니다. 자세한 내용은 *만료된 컨트롤 플레인 인증서에서 복구 문서*를 참조하십시오.
- 24시간 인증서는 클러스터를 설치한 후 16시간에서 22시간 사이에 회전하므로 생성된 후 12시간 이내에 Ignition 구성 파일을 사용하는 것이 좋습니다. 12시간 이내에 Ignition 구성 파일을 사용하면 설치 중에 인증서 업데이트가 실행되는 경우 설치 실패를 방지할 수 있습니다.

클러스터 부트스트랩에는 다음 단계가 포함됩니다.

- 부트스트랩 머신이 부팅되고 컨트롤 플레인 머신을 부팅하는 데 필요한 원격 리소스 호스팅이 시작됩니다. 인프라를 프로비저닝하는 경우 이 단계에는 수동 개입이 필요합니다.
- 부트스트랩 머신은 단일 노드 etcd 클러스터와 임시 Kubernetes 컨트롤 플레인을 시작합니다.

3. 컨트롤 플레인 머신은 부트스트랩 머신에서 원격 리소스를 가져오고 부팅을 완료합니다. 인프라를 프로비저닝하는 경우 이 단계에는 수동 개입이 필요합니다.
4. 임시 컨트롤 플레인 프로덕션 컨트롤러 플레인을 프로덕션 컨트롤 플레인 머신에 예약합니다.
5. CVO(Cluster Version Operator)가 온라인 상태가 되어 etcd Operator를 설치합니다. etcd Operator는 모든 컨트롤 플레인 노드에서 etcd를 확장합니다.
6. 임시 컨트롤 플레인이 종료되고 제어를 프로덕션 컨트롤 플레인에 전달합니다.
7. 부트스트랩 머신은 OpenShift Container Platform 구성 요소를 프로덕션 컨트롤 플레인에 주입합니다.
8. 설치 프로그램이 부트 스트랩 머신을 종료합니다. 인프라를 프로비저닝하는 경우 이 단계에는 수동 개입이 필요합니다.
9. 컨트롤 플레인이 컴퓨팅 노드를 설정합니다.
10. 컨트롤 플레인은 일련의 Operator 형태로 추가 서비스를 설치합니다.

이 부트스트랩 프로세스의 결과는 실행 중인 OpenShift Container Platform 클러스터입니다. 그런 다음 클러스터는 지원되는 환경에서 컴퓨팅 머신 생성을 포함하여 일상적인 작업에 필요한 나머지 구성 요소를 다운로드하고 구성합니다.

#### 추가 리소스

- [etcd 관련 권장 사례](#)
- [컨트롤 플레인 노드 크기 조정](#)

#### 3.1.5. 설치 범위

OpenShift Container Platform 설치 프로그램의 범위는 의도적으로 한정됩니다. 단순성과 성공을 보장하도록 설계되었습니다. 설치가 완료된 후 많은 추가 구성 작업을 완료할 수 있습니다.

#### 추가 리소스

- OpenShift Container Platform 구성 [리소스에 대한 자세한 내용은 사용 가능한 클러스터 사용자 정의를](#) 참조하십시오.

### 3.2. OPENSIFT 업데이트 서비스 정보

OSUS(OpenShift Update Service)는 RHCOS(Red Hat Enterprise Linux CoreOS)를 포함하여 OpenShift Container Platform에 대한 업데이트 권장 사항을 제공합니다. 구성 요소 Operator의 [정점](#)과 이를 연결하는 [에지](#)를 포함하는 그래프 또는 다이어그램을 제공합니다. 그래프의 에지에는 안전하게 업데이트할 수 있는 버전이 표시됩니다. 정점은 관리형 클러스터 구성 요소의 상태를 지정하는 업데이트 페이로드입니다.

클러스터의 CVO (Cluster Version Operator)는 OpenShift Update Service를 확인하여 현재 구성 요소 버전 및 그래프의 정보를 기반으로 유효한 업데이트 및 업데이트 경로를 확인합니다. 업데이트를 요청하면 CVO는 해당 릴리스 이미지를 사용하여 클러스터를 업데이트합니다. 릴리스 아티팩트는 Quay에서 컨테이너 이미지로 호스팅됩니다.

OpenShift Update Service가 호환 가능한 업데이트만 제공할 수 있도록 자동화를 지원하는 버전 확인 파이프라인이 제공됩니다. 각 릴리스 아티팩트는 지원되는 클라우드 플랫폼 및 시스템 아키텍처 및 기타 구성 요소 패키지와의 호환성 여부를 확인합니다. 파이프라인에서 적용 가능한 버전이 있음을 확인한 후

OpenShift Update Service는 해당 버전 업데이트를 사용할 수 있음을 알려줍니다.

OSUS(OpenShift Update Service)는 하나의 릴리스 버전만 활성 상태이고 언제든지 지원되는 단일 스트림 릴리스 모델을 지원합니다. 새 릴리스가 배포되면 이전 릴리스를 완전히 대체합니다.

업데이트된 릴리스에서는 4.8부터 새 릴리스 버전으로의 모든 OpenShift Container Platform 버전에서 업그레이드를 지원합니다.



#### 중요

OpenShift Update Service는 현재 클러스터에 권장되는 모든 업데이트를 표시합니다. OpenShift Update Service에서 업데이트 경로를 권장하지 않는 경우 업데이트 또는 대상 릴리스와 관련된 알려진 문제로 인해 발생할 수 있습니다.

연속 업데이트 모드에서는 두 개의 컨트롤러가 실행됩니다. 하나의 컨트롤러는 페이로드 매니페스트를 지속적으로 업데이트하여 매니페스트를 클러스터에 적용한 다음 Operator의 제어된 롤아웃 상태를 출력하여 사용 가능한지, 업그레이드했는지 또는 실패했는지의 여부를 나타냅니다. 두 번째 컨트롤러는 OpenShift Update Service를 폴링하여 업데이트를 사용할 수 있는지 확인합니다.



#### 중요

최신 버전으로만 업데이트할 수 있습니다. 클러스터를 이전 버전으로 되돌리거나 롤백을 수행하는 것은 지원되지 않습니다. 업데이트에 실패하면 Red Hat 지원에 문의하십시오.

업데이트 프로세스 중에 MCO(Machine Config Operator)는 새 설정을 클러스터 머신에 적용합니다. MCO는 머신 구성 풀의 **maxUnavailable** 필드에 지정된 노드 수를 제한하고 사용할 수 없음을 표시합니다. 기본적으로 이 값은 **1**로 설정됩니다. MCO는 **topology.kubernetes.io/zone** 레이블을 기반으로 영역별로 영향을 받는 노드를 업데이트합니다. 영역에 둘 이상의 노드가 있으면 가장 오래된 노드가 먼저 업데이트됩니다. 베어 메탈 배포에서와 같이 영역을 사용하지 않는 노드의 경우 노드가 수명에 따라 업데이트되며 가장 오래된 노드가 먼저 업데이트됩니다. MCO는 머신 구성 풀의 **maxUnavailable** 필드에 지정된 노드 수를 한 번에 업데이트합니다. MCO는 새 설정을 적용하여 컴퓨터를 다시 시작합니다.



#### 주의

**maxUnavailable**의 기본 설정은 OpenShift Container Platform의 모든 머신 구성 풀에 대해 **1**입니다. 이 값을 변경하지 않고 한 번에 하나의 컨트롤 플레인 노드를 업데이트하는 것이 좋습니다. 컨트롤 플레인 풀의 경우 이 값을 **3**으로 변경하지 마십시오.

RHEL (Red Hat Enterprise Linux) 머신을 작업자로 사용하는 경우 먼저 시스템에서 OpenShift API를 업데이트해야 하기 때문에 MCO는 이 머신에서 kubelet을 업데이트하지 않습니다.

새 버전의 사양이 이전 kubelet에 적용되므로 RHEL 머신을 **Ready** 상태로 되돌릴 수 없습니다. 컴퓨터를 사용할 수 있을 때까지 업데이트를 완료할 수 없습니다. 그러나 사용 불가능한 최대 노드 수를 설정하면 사용할 수 없는 머신의 수가 이 값을 초과하지 않는 경우에도 정상적인 클러스터 작업을 계속할 수 있습니다.

OpenShift Update Service는 Operator 및 하나 이상의 애플리케이션 인스턴스로 구성됩니다.

### 3.3. 관리되지 않는 OPERATOR에 대한 지원 정책

Operator의 **관리 상태**는 Operator가 설계 의도에 따라 클러스터의 해당 구성 요소에 대한 리소스를 적극적으로 관리하고 있는지 여부를 판별합니다. *Unmanaged* 상태로 설정된 Operator는 구성 변경에 응답하지 않고 업데이트되지도 않습니다.

비프로덕션 클러스터 또는 디버깅 중에는 이 기능이 유용할 수 있지만, *Unmanaged* 상태의 Operator는 지원되지 않으며 개별 구성 요소의 구성 및 업그레이드를 클러스터 관리자가 전적으로 통제하게 됩니다.

다음과 같은 방법으로 Operator를 *Unmanaged* 상태로 설정할 수 있습니다.

- **개별 Operator 구성**

개별 Operator는 구성에 **managementState** 매개변수가 있습니다. Operator에 따라 다양한 방식으로 이 매개변수에 액세스할 수 있습니다. 예를 들어, Red HAt OpenShift Logging Operator는 관리 대상인 사용자 정의 리소스(CR)를 수정하여 이를 수행하는 반면 Cluster Samples Operator는 클러스터 전체의 구성 리소스를 사용합니다.

**managementState** 매개변수를 **Unmanaged**로 변경하면 Operator가 리소스를 적극적으로 관리하지 않으며 해당하는 구성 요소와 관련된 조치도 수행하지 않습니다. 클러스터가 손상되고 수동 복구가 필요할 가능성이 있으므로 이 관리 상태를 지원하지 않는 Operator도 있습니다.



#### 주의

개별 Operator를 **Unmanaged** 상태로 변경하면 특정 구성 요소 및 기능이 지원되지 않습니다. 지원을 계속하려면 보고된 문제를 **Managed** 상태에서 재현해야 합니다.

- **Cluster Version Operator(CVO) 재정의**

**spec.overrides** 매개변수를 CVO 구성에 추가하여 관리자가 구성 요소에 대한 CVO 동작에 대한 재정의 목록을 제공할 수 있습니다. 구성 요소에 대해 **spec.overrides[].unmanaged** 매개변수를 **true**로 설정하면 클러스터 업그레이드가 차단되고 CVO 재정의가 설정된 후 관리자에게 경고합니다.

Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.



#### 주의

CVO 재정의 설정하면 전체 클러스터가 지원되지 않는 상태가 됩니다. 지원을 계속하려면 재정의 제거 후 보고된 문제를 재현해야 합니다.

## 3.4. 다음 단계

- 클러스터 설치 방법 선택 및 사용자를 위한 준비

## 4장. RED HAT OPENSIFT CLUSTER MANAGER

Red Hat OpenShift Cluster Manager는 Red Hat OpenShift 클러스터를 설치, 수정, 운영 및 업그레이드할 수 있는 관리형 서비스입니다. 이 서비스를 사용하면 단일 대시보드에서 모든 조직의 클러스터 작업을 수행할 수 있습니다.

OpenShift Cluster Manager는 OpenShift Container Platform, Red Hat OpenShift Service on AWS(ROSA) 및 OpenShift Dedicated 클러스터를 설치하도록 안내합니다. 또한 ROSA 및 OpenShift Dedicated 클러스터뿐만 아니라 자체 설치 후 OpenShift Container Platform 클러스터를 모두 관리합니다.

OpenShift Cluster Manager를 사용하여 다음 작업을 수행할 수 있습니다.

- 새 클러스터 생성
- 클러스터 세부 정보 및 메트릭 보기
- 노드 레이블 변경, 네트워킹, 인증과 같은 작업으로 클러스터를 관리
- 액세스 제어 관리
- 클러스터 모니터링
- 업그레이드 예약

### 4.1. RED HAT OPENSIFT CLUSTER MANAGER에 액세스

구성된 OpenShift 계정을 사용하여 OpenShift Cluster Manager에 액세스할 수 있습니다.

#### 사전 요구 사항

- OpenShift 조직의 일부인 계정이 있습니다.
- 클러스터를 생성하는 경우 조직에서 할당량을 지정합니다.

#### 프로세스

- 로그인 인증 정보를 사용하여 [OpenShift Cluster Manager](#)에 로그인합니다.

### 4.2. 일반 작업

클러스터 페이지의 오른쪽 상단에는 사용자가 전체 클러스터에서 수행할 수 있는 몇 가지 작업이 있습니다.

- 클러스터 소유자가 클러스터에 명령을 실행할 수 있도록 **콘솔을 열고** 웹 콘솔을 시작합니다.
- **작업** 드롭다운 메뉴를 사용하면 클러스터 소유자가 클러스터의 표시 이름 변경, 해당하는 경우 클러스터의 로드 밸런서 및 영구 스토리지의 이름을 변경하고, 노드 수를 수동으로 설정하고, 클러스터를 삭제할 수 있습니다.
- **새로 고침** 아이콘은 클러스터를 강제 새로 고칩니다.

### 4.3. 클러스터 탭

활성으로 설치된 클러스터를 선택하면 해당 클러스터와 연결된 탭이 표시됩니다. 클러스터 설치가 완료되면 다음 탭이 표시됩니다.

- 개요
- 액세스 제어
- 애드온
- 네트워킹
- Insights 어드바이저
- 머신 풀
- 지원
- 설정

#### 4.3.1. 개요 탭

개요 탭에서는 클러스터 구성 방법에 대한 정보를 제공합니다.

- **클러스터 ID**는 생성된 클러스터의 고유한 ID입니다. 이 ID는 명령줄에서 클러스터에 명령을 실행할 때 사용할 수 있습니다.
- **유형**은 클러스터가 사용 중인 OpenShift 버전을 보여줍니다.
- **리전**은 서버 영역입니다.
- **공급자**는 클러스터가 구축된 클라우드 공급자를 보여줍니다.
- **가용성**은 클러스터가 단일 또는 다중 영역 중 하나를 사용하는 가용성 영역의 유형을 보여줍니다.
- **버전**은 클러스터에 설치된 OpenShift 버전입니다. 사용 가능한 업데이트가 있는 경우 이 필드에서 업데이트할 수 있습니다.
- 이 **생성된 날짜**는 클러스터가 생성된 날짜 및 시간을 표시합니다.
- **소유자**는 클러스터를 생성한 사용자와 소유자 권한이 있는 사람을 식별합니다.
- **서브스크립션 유형**은 생성 시 선택한 서브스크립션 모델을 표시합니다.
- **인프라 유형**은 클러스터가 사용하는 계정 유형입니다.
- **상태**에 클러스터의 현재 상태가 표시됩니다.
- **총 vCPU**는 이 클러스터에 사용 가능한 총 가상 CPU를 보여줍니다.
- **총 메모리**는 이 클러스터에 사용 가능한 총 메모리를 보여줍니다.
- **로드 밸런서**
- **영구 스토리지**는 이 클러스터에서 사용 가능한 스토리지의 양을 표시합니다.
- **노드**에는 클러스터에서 실제 및 원하는 노드가 표시됩니다. 이러한 수는 클러스터 확장으로 인해 일치하지 않을 수 있습니다.

- network 필드에는 네트워크 연결의 주소 및 접두사가 표시됩니다.
- 탭의 리소스 사용량 섹션에는 그래프와 함께 사용되는 리소스가 표시됩니다.
- 어드바이저 권장 사항 섹션에서는 보안, 성능, 가용성 및 안정성에 대한 정보를 제공합니다. 이 섹션에서는 원격 상태 기능을 사용해야 합니다. 추가 리소스 섹션에서 *Insights*를 사용하여 클러스터 문제 식별을 참조하십시오.
- 클러스터 내역 섹션에는 생성 및 새 버전이 식별되는 시기를 포함하여 클러스터와 함께 수행된 모든 작업이 표시됩니다.

#### 4.3.2. 액세스 제어 탭

액세스 제어 탭에서는 클러스터 소유자가 ID 공급자를 설정하고 승격된 권한을 부여하며 다른 사용자에게 역할을 부여할 수 있습니다.

##### 사전 요구 사항

- 클러스터 소유자여야 하거나 클러스터에 역할을 부여할 수 있는 올바른 권한이 있어야 합니다.

##### 프로세스

1. 역할 부여 버튼을 선택합니다.
2. 클러스터에 역할을 부여하려는 사용자의 Red Hat 계정 로그인을 입력합니다.
3. 대화 상자에서 역할 부여 버튼을 선택합니다.
4. 대화 상자가 닫히고 선택한 사용자에게 "클러스터 편집기" 액세스 권한이 표시됩니다.

#### 4.3.3. 애드온 탭

애드온 탭에는 클러스터에 추가할 수 있는 모든 선택적 애드온이 표시됩니다. 원하는 추가 기능을 선택한 다음 표시되는 애드온에 대한 설명 아래에 있는 설치를 선택합니다.

#### 4.3.4. Insights 어드바이저 탭

Insights 어드바이저 탭은 OpenShift Container Platform의 원격 상태 기능을 사용하여 보안, 성능, 가용성 및 안정성에 대한 위험을 식별하고 완화합니다. OpenShift Container Platform 설명서에서 [클러스터 관련 문제를 식별하는 방법은 Insights 사용을 참조하십시오](#).

#### 4.3.5. 머신 풀 탭

머신 풀 탭을 사용하면 클러스터 소유자가 사용 가능한 할당량이 충분한 경우 새 머신 풀을 생성하거나 기존 머신 풀을 편집할 수 있습니다.

기타 옵션 > 스케일링을 선택하면 "노드 수 편집" 대화 상자가 열립니다. 이 대화 상자에서 가용성 영역당 노드 수를 변경할 수 있습니다. 자동 스케일링이 활성화된 경우 자동 스케일링의 범위를 설정할 수도 있습니다.

#### 4.3.6. 지원 탭



지원 탭에서 클러스터 알림을 받아야 하는 개인에 대한 알림 연락처를 추가할 수 있습니다. 사용자가 제공하는 사용자 이름 또는 이메일 주소는 클러스터가 배포된 Red Hat 조직의 사용자 계정과 관련이 있어야 합니다.

또한 이 탭에서 클러스터에 대한 기술 지원을 요청하기 위해 지원 케이스를 열 수 있습니다.

#### 4.3.7. 설정 탭

설정 탭에는 클러스터 소유자에 대한 몇 가지 옵션이 있습니다.

- **모니터링**은 기본적으로 활성화되어 있는 사용자 정의 작업에 대해 보고할 수 있습니다. [모니터링 스택 이해](#)를 참조하십시오.
- **업데이트 전략**을 사용하면 지정된 시간에 특정 요일에 클러스터가 자동으로 업데이트되는지 또는 모든 업데이트가 수동으로 예약되는지 확인할 수 있습니다.
- **노드 드레이닝**은 업데이트 중에 보호 워크로드가 적용되는 기간을 설정합니다. 이 기간이 지나면 노드가 강제로 제거됩니다.
- **업데이트 상태**에 현재 버전이 표시되고 사용 가능한 업데이트가 있는 경우 표시됩니다.

#### 4.4. 추가 리소스

- OpenShift Cluster Manager에 대한 전체 문서는 [OpenShift Cluster Manager 설명서](#)를 참조하십시오.

## 5장. KUBERNETES OPERATOR의 다중 클러스터 엔진 정보

Kubernetes 환경 확장의 과제 중 하나는 증가하는 플릿의 라이프사이클을 관리하는 것입니다. 이러한 문제를 충족하기 위해 다중 클러스터 엔진 Operator를 사용할 수 있습니다. Operator는 관리형 OpenShift Container Platform 클러스터 및 다른 Kubernetes 배포에 대한 부분 라이프사이클 관리를 위한 전체 라이프사이클 기능을 제공합니다. 두 가지 방법으로 사용할 수 있습니다.

- OpenShift Container Platform 또는 OpenShift Kubernetes Engine 서브스크립션의 일부로 설치하는 독립 실행형 Operator
- [Red Hat Advanced Cluster Management for Kubernetes](#) 의 일부로

### 5.1. OPENSIFT CONTAINER PLATFORM에서 다중 클러스터 엔진으로 클러스터 관리

OpenShift Container Platform에서 다중 클러스터 엔진을 활성화하면 다음 기능을 얻을 수 있습니다.

- HyperShift 프로젝트를 기반으로 하는 기능인 [호스트된 컨트롤 플레인](#). 중앙 집중식 호스트 컨트롤 플레인을 사용하면 하이퍼스케일 방식으로 OpenShift Container Platform 클러스터를 작동할 수 있습니다.
- Hive: 자체 관리형 OpenShift Container Platform 클러스터를 hub에 프로비저닝하고 해당 클러스터의 초기 구성을 완료합니다.
- 관리 클러스터를 허브에 등록하는 klusterlet 에이전트입니다.
- 인프라 Operator: 베어 메탈의 단일 노드 OpenShift와 같은 OpenShift Container Platform의 온프레미스 베어 메탈 및 vSphere 설치를 오케스트레이션하기 위해 지원 서비스 배포를 관리합니다. Infrastructure Operator에는 [GitOps ZTP\(ZTP\)가 포함되어 있습니다. 이 프로비저닝](#)에는 GitOps 워크플로우를 사용하여 베어 메탈 및 vSphere 프로비저닝에서 클러스터 생성을 완전히 자동화하여 배포 및 구성 변경 사항을 관리합니다.
- Kubernetes 클러스터를 관리하는 데 필요한 리소스를 제공하는 오픈 클러스터 관리입니다.

멀티 클러스터 엔진은 OpenShift Container Platform 지원 서브스크립션에 포함되어 있으며 코어 페이로드와 별도로 제공됩니다. 다중 클러스터 엔진을 사용하려면 OpenShift Container Platform 클러스터를 배포한 다음 Operator를 설치합니다. 자세한 내용은 [다중 클러스터 엔진 Operator 설치 및 업그레이드](#)를 참조하십시오.

### 5.2. RED HAT ADVANCED CLUSTER MANAGEMENT를 통한 클러스터 관리

다중 클러스터 엔진을 사용하여 OpenShift Container Platform에서 제공할 수 있는 클러스터 관리 기능이 필요한 경우 Red Hat Advanced Cluster Management를 고려하십시오. 멀티 클러스터 엔진은 Red Hat Advanced Cluster Management의 통합된 부분이며 기본적으로 활성화되어 있습니다.

### 5.3. 추가 리소스

다중 클러스터 엔진에 대한 전체 문서는 Red Hat Advanced [Cluster Management](#)의 제품 설명서에 포함된 [다중 클러스터 엔진 설명서](#)가 포함된 클러스터 라이프사이클을 참조하십시오.

## 6장. 컨트롤 플레인 아키텍처

컨트롤 플레인 시스템으로 구성된 컨트롤 플레인 OpenShift Container Platform 클러스터를 관리합니다. 컨트롤 플레인 머신에서는 작업자 머신이라고도 하는 컴퓨팅 머신의 워크로드를 관리합니다. 클러스터 자체는 CVO(Cluster Version Operator), Machine Config Operator 및 개별 Operator 세트를 통해 머신의 모든 업그레이드를 관리합니다.

### 6.1. 머신 구성 풀을 사용한 노드 구성 관리

컨트롤 플레인 구성 요소 또는 사용자 워크로드를 실행하는 머신은 처리하는 리소스 유형에 따라 그룹으로 나뉩니다. 이러한 머신 그룹을 MCP(Machine config pool)라고 합니다. 각 MCP는 노드 세트와 해당 머신 구성을 관리합니다. 노드의 역할은 해당 노드가 속한 MCP를 결정합니다. MCP는 할당된 노드 역할 레이블을 기반으로 노드를 관리합니다. MCP의 노드에는 동일한 구성이 있습니다. 즉, 워크로드 증가 또는 감소에 따라 노드를 확장하고 제거할 수 있습니다.

기본적으로 클러스터가 설치될 때 생성되는 **master** 및 **worker**라는 두 개의 MCP가 있습니다. 각 기본 MCP에는 MCP(Machine Config Operator)에 의해 적용되는 정의된 구성이 있으며 MCP를 관리하고 MCP 업데이트를 용이하게 합니다.

작업자 노드의 경우 추가 MCP 또는 사용자 지정 풀을 생성하여 기본 노드 유형 외부에서 확장되는 사용자 정의 사용 사례로 노드를 관리할 수 있습니다. 컨트롤 플레인 노드의 사용자 정의 MCP는 지원되지 않습니다.

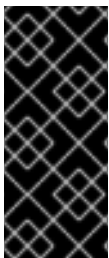
사용자 지정 풀은 작업자 풀에서 구성을 상속하는 풀입니다. 작업자 풀을 대상으로 하는 머신 구성을 사용하지만 사용자 지정 풀을 대상으로 하는 변경 사항만 배포할 수 있는 기능을 추가합니다. 사용자 지정 풀은 작업자 풀의 구성을 상속하므로 작업자 풀에 대한 모든 변경 사항이 사용자 정의 풀에도 적용됩니다. 작업자 풀에서 구성을 상속하지 않는 사용자 지정 풀은 MCO에서 지원되지 않습니다.



#### 참고

노드는 하나의 MCP에만 포함될 수 있습니다. 노드에 **worker,infra**와 같이 여러 MCP에 해당하는 여러 레이블이 있는 경우 작업자 풀이 아닌 인프라 사용자 지정 풀로 관리됩니다. 사용자 지정 풀은 노드 레이블을 기반으로 관리할 노드를 선택하는 데 우선 순위를 두고 사용자 지정 풀에 속하지 않은 노드는 작업자 풀에서 관리합니다.

클러스터에서 관리할 모든 노드 역할에 대한 사용자 지정 풀을 사용하는 것이 좋습니다. 예를 들어 인프라 워크로드를 처리하기 위해 인프라 노드를 생성하는 경우 사용자 지정 인프라 MCP를 생성하여 해당 노드를 그룹화하는 것이 좋습니다. 작업자 노드에 **infra** 역할 레이블을 적용하면 **worker,infra** 이중 레이블이 있지만 사용자 지정 인프라 MCP가 없는 경우 MCO는 이를 작업자 노드로 간주합니다. 노드에서 **worker** 레이블을 제거하고 사용자 지정 풀에서 그룹화하지 않고 **infra** 레이블을 적용하면 노드는 MCO에서 인식되지 않으며 클러스터에서 관리되지 않습니다.



#### 중요

인프라 워크로드를 실행하는 경우에만 **infra** 역할로 레이블이 지정된 노드는 총 서브스크립션 수에 포함되지 않습니다. 인프라 노드를 관리하는 MCP는 클러스터가 서브스크립션을 결정하는 방법에서 상호 배타적일 수 있습니다. 적절한 **infra** 역할로 노드를 태그하고 태그를 사용하여 사용자 워크로드가 해당 노드에서 예약되지 않도록 하는 것이 인프라 워크로드에 대한 서브스크립션 요금을 부과하지 않도록 하는 유일한 요구 사항입니다.

MCO는 풀에 독립적으로 업데이트를 적용합니다. 예를 들어 모든 풀에 영향을 미치는 업데이트가 있는 경우 각 풀의 노드는 서로 병렬로 업데이트됩니다. 사용자 지정 풀을 추가하면 해당 풀에서 노드도 마스터 및 작업자 노드와 동시에 업데이트를 시도합니다.

노드의 구성이 현재 적용된 머신 구성에서 지정하는 것과 완전히 일치하지 않는 경우가 있을 수 있습니다. 이 상태를 구성 드리프트라고 합니다. MCO(Machine Config Daemon)는 노드에서 구성 드리프트를 정기적으로 확인합니다. MCO가 구성 드리프트를 감지하면 관리자가 노드 구성을 수정할 때까지 MCO는 노드가 **degraded** 상태로 표시됩니다. 성능이 저하된 노드는 온라인 상태이고 작동하지만 업데이트할 수 없습니다.

## 추가 리소스

- [구성 드리프트 탐지 이해](#)

## 6.2. OPENSIFT CONTAINER PLATFORM의 머신 역할

OpenShift Container Platform은 호스트에 다른 역할을 할당합니다. 이 역할은 클러스터 내에서 머신의 기능을 정의합니다. 클러스터에는 표준 **master** 및 **worker** 역할 유형에 대한 정의가 포함되어 있습니다.



### 참고

클러스터에는 **bootstrap** 역할의 정의도 포함되어 있습니다. 부트스트랩 머신은 클러스터 설치 중에만 사용되므로 해당 기능은 클러스터 설치 설명서에 설명되어 있습니다.

### 6.2.1. 컨트롤 플레인 및 노드 호스트 호환성

OpenShift Container Platform 버전은 컨트롤 플레인 호스트와 노드 호스트 간에 일치해야 합니다. 예를 들어 4.15 클러스터에서 모든 컨트롤 플레인 호스트는 4.15이고 모든 노드가 4.15여야 합니다.

클러스터 업그레이드 중에 일시적인 불일치가 허용됩니다. 예를 들어 이전 OpenShift Container Platform 버전에서 4.15로 업그레이드할 때 일부 노드는 다른 노드보다 먼저 4.15로 업그레이드됩니다. 컨트롤 플레인 호스트 및 노드 호스트가 오래 걸리는 경우 이전 컴퓨팅 머신이 버그 및 누락된 기능에 노출될 수 있습니다. 사용자는 가능한 한 빨리 스쿼드 컨트롤 플레인 호스트 및 노드 호스트를 확인해야 합니다.

**kubelet** 서비스는 **kube-apiserver** 보다 최신이 아니어야 하며 OpenShift Container Platform 버전이 홀수인지 여부에 따라 최대 두 개의 마이너 버전일 수 있습니다. 아래 표에서는 적절한 버전 호환성을 보여줍니다.

OpenShift Container Platform 버전	지원되는 kubelet skew
홀수의 OpenShift Container Platform 마이너 버전 [1]	최대 하나의 이전 버전
짝수의 OpenShift Container Platform 마이너 버전 [2]	최대 두 개의 이전 버전

1. 예를 들면 OpenShift Container Platform 4.11, 4.13입니다.
2. 예를 들면 OpenShift Container Platform 4.10, 4.12입니다.

### 6.2.2. 클러스터 작업자

Kubernetes 클러스터에서 작업자 노드는 Kubernetes 사용자가 요청한 실제 워크로드를 실행하고 관리합니다. 작업자 노드는 용량을 알리고 컨트롤 플레인 서비스인 스케줄러는 Pod 및 컨테이너를 시작할 노드를 결정합니다. 다음 중요한 서비스는 각 작업자 노드에서 실행됩니다.

- CRI-O - 컨테이너 엔진입니다.

- kubelet: 컨테이너 워크로드 실행 및 중지 요청을 수락하고 이행하는 서비스입니다.
- 작업자 간 Pod의 통신을 관리하는 서비스 프록시입니다.
- 컨테이너를 생성하고 실행하는 runC 또는 crun 하위 수준 컨테이너 런타임입니다.



#### 참고

기본 runC 대신 crun을 활성화하는 방법에 대한 자세한 내용은 **ContainerRuntimeConfig** CR 생성 설명서를 참조하십시오.

OpenShift Container Platform에서 컴퓨팅 머신 세트는 **worker** 머신 역할이 할당된 컴퓨팅 머신을 제어합니다. **worker** 역할 드라이브가 있는 머신은 자동 스케일링하는 특정 머신 풀이 관리하는 워크로드를 컴퓨팅합니다. OpenShift Container Platform에는 여러 머신 유형을 지원할 수 있는 용량이 있으므로 **worker** 역할의 시스템은 *compute* 머신으로 분류됩니다. 이 릴리스에서는 기본 유형의 컴퓨팅 머신이 작업자 머신이기 때문에 *작업자 머신*과 *컴퓨팅 머신*이라는 용어는 서로 바꿔 사용할 수 있습니다. 이후 버전의 OpenShift Container Platform에서는 기본적으로 인프라 머신과 같은 다른 유형의 컴퓨팅 머신이 사용될 수 있습니다.



#### 참고

컴퓨팅 머신 세트는 **machine-api** 네임스페이스에서 컴퓨팅 머신 리소스의 그룹입니다. 컴퓨팅 머신 세트는 특정 클라우드 공급자에서 새 컴퓨팅 머신을 시작하도록 설계된 구성입니다. 반대로 MCP(Machine config pool)는 MCO(Machine Config Operator) 네임스페이스의 일부입니다. MCP는 MCO가 구성을 관리하고 업그레이드를 원활하게 수행할 수 있도록 머신을 그룹화하는데 사용됩니다.

### 6.2.3. 클러스터 컨트롤 플레인

Kubernetes 클러스터에서 *master* 노드는 Kubernetes 클러스터를 제어하는 데 필요한 서비스를 실행합니다. OpenShift Container Platform에서 컨트롤 플레인은 **master** 머신 역할이 있는 컨트롤 플레인 시스템으로 구성됩니다. 여기에는 OpenShift Container Platform 클러스터 관리를 위한 Kubernetes 서비스 외에도 많은 것이 포함되어 있습니다.

대부분의 OpenShift Container Platform 클러스터의 경우 컨트롤 플레인 시스템은 일련의 독립 실행형 머신 API 리소스로 정의됩니다. 지원되는 클라우드 공급자 및 OpenShift Container Platform 버전 조합의 경우 컨트롤 플레인 머신 세트를 사용하여 컨트롤 플레인을 관리할 수 있습니다. 모든 컨트롤 플레인 시스템을 삭제하고 클러스터를 중단하지 못하도록 컨트롤 플레인 시스템에 추가 제어가 적용됩니다.



#### 참고

모든 프로덕션 배포에 정확히 세 개의 컨트롤 플레인 노드를 사용해야 합니다.

컨트롤 플레인의 Kubernetes 카테고리에 속하는 서비스에는 Kubernetes API 서버, etcd, Kubernetes 컨트롤러 관리자 및 Kubernetes 스케줄러가 포함됩니다.

표 6.1. 컨트롤 플레인에서 실행되는 Kubernetes 서비스

구성 요소	설명
Kubernetes API 서버	Kubernetes API 서버는 포트, 서비스 및 복제 컨트롤러의 데이터를 검증하고 구성합니다. 클러스터의 공유 상태에 대한 초점도 제공합니다.

구성 요소	설명
etcd	etcd는 영구 컨트롤 플레인 상태를 저장하고 다른 구성 요소는 etcd에서 변경 사항을 감시하여 지정된 상태로 전환합니다.
Kubernetes 컨트롤러 관리자	Kubernetes 컨트롤러 관리자는 etcd에서 복제, 네임스페이스 및 서비스 계정 컨트롤러 오브젝트와 같은 오브젝트의 변경사항을 감시한 다음 API를 사용하여 지정된 상태를 적용합니다. 이러한 여러 프로세스는 한 번에 하나의 활성 리더가 있는 클러스터를 생성합니다.
Kubernetes 스케줄러	Kubernetes 스케줄러는 할당된 노드가 없는 새로 생성된 Pod를 감시하고 Pod를 호스팅할 수 있는 최상의 노드를 선택합니다.

OpenShift API 서버, OpenShift 컨트롤러 관리자 및 OpenShift OAuth API 서버 및 OpenShift OAuth 서버를 포함하는 컨트롤 플레인에서 실행되는 OpenShift 서비스도 있습니다.

## 표 6.2. 컨트롤 플레인에서 실행되는 OpenShift 서비스

구성 요소	설명
OpenShift API 서버	OpenShift API 서버는 프로젝트, 경로 및 템플릿과 같은 OpenShift 리소스의 데이터 유효성을 검사하고 구성합니다.  OpenShift API 서버는 OpenShift API Server Operator가 관리합니다.
OpenShift 컨트롤러 관리자	OpenShift 컨트롤러 관리자는 etcd에서 프로젝트, 경로 및 템플릿 컨트롤러 오브젝트와 같은 OpenShift 오브젝트의 변경사항을 감시한 다음 API를 사용하여 지정된 상태를 적용합니다.  OpenShift 컨트롤러 관리자는 OpenShift Controller Manager Operator가 관리합니다.
OpenShift OAuth API 서버	OpenShift OAuth API 서버는 사용자, 그룹 및 OAuth 토큰과 같은 OpenShift Container Platform에 인증할 데이터의 유효성을 검사하고 구성합니다.  OpenShift OAuth API 서버는 Cluster Authentication Operator가 관리합니다.
OpenShift OAuth 서버	사용자는 OpenShift OAuth 서버에서 토큰을 요청하여 API에 자신을 인증합니다.  OpenShift OAuth 서버는 Cluster Authentication Operator가 관리합니다.

컨트롤 플레인 시스템에서 이러한 서비스 중 일부는 systemd 서비스로 실행되는 반면 다른 서비스는 정적 Pod로 실행됩니다.

시스템 서비스는 특정 시스템이 시작된 직후에 항상 필요한 서비스에 적합합니다. 컨트롤 플레인 시스템의 경우 원격 로그인을 허용하는 sshd가 포함됩니다. 다음과 같은 서비스도 포함됩니다.

- 컨테이너를 실행하고 관리하는 CRI-O 컨테이너 엔진(crio). OpenShift Container Platform 4.15는 Docker Container Engine 대신 CRI-O를 사용합니다.
- kubelet(kubelet): 컨트롤 플레인 서비스에서 머신의 컨테이너 관리 요청을 수락합니다.

CRI-O 및 Kubelet은 다른 컨테이너를 실행하기 전에 실행되어야 하기 때문에 systemd 서비스로 호스트에서 직접 실행해야 합니다.

**installer-\*** 및 **revision-pruner-\*** 컨트롤 플레인 Pod는 root 사용자가 소유한 **/etc/kubernetes** 디렉토리에 쓰기 때문에 root 권한으로 실행해야 합니다. 이러한 pod에는 다음과 같은 네임스페이스에 있습니다.

- **openshift-etcd**
- **openshift-kube-apiserver**
- **openshift-kube-controller-manager**
- **openshift-kube-scheduler**

## 6.3. OPENSIFT CONTAINER PLATFORM의 OPERATOR

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

Operator는 **kubectl** 및 **oc** 명령과 같은 CLI 툴 및 Kubernetes API와 통합됩니다. 애플리케이션을 모니터링하고, 상태 점검 수행, OTA(Over-the-Air) 업데이트를 관리하며 애플리케이션이 지정된 상태로 유지되도록 합니다.

Operator에서는 더 세분화된 구성 환경도 제공합니다. 글로벌 구성 파일을 수정하는 대신 Operator가 표시하는 API를 수정하여 각 구성 요소를 구성합니다.

CRI-O 및 Kubelet은 모든 노드에서 실행되므로 Operator를 사용하여 거의 모든 다른 클러스터 기능을 컨트롤 플레인에서 관리할 수 있습니다. Operator를 사용하여 컨트롤 플레인에 추가되는 구성 요소에는 중요한 네트워킹 및 인증 정보 서비스가 포함됩니다.

두 가지 모두 유사한 Operator 개념과 목표를 따르지만 OpenShift Container Platform의 Operator는 목적에 따라 두 가지 다른 시스템에서 관리합니다.

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

### 6.3.1. 클러스터 Operator

OpenShift Container Platform에서 모든 클러스터 기능은 일련의 기본 *클러스터 Operator*로 나뉩니다. 클러스터 Operator는 클러스터 전체 애플리케이션 로깅, Kubernetes 컨트롤 플레인 관리 또는 머신 프로비저닝 시스템과 같은 특정 클러스터 기능 영역을 관리합니다.

클러스터 Operator는 **ClusterOperator** 오브젝트로 표시됩니다. 이 오브젝트는 클러스터 관리자가 **관리 → 클러스터 설정** 페이지의 OpenShift Container Platform 웹 콘솔에서 볼 수 있습니다. 각 클러스터 Operator는 클러스터 기능을 결정하기 위한 간단한 API를 제공합니다. Operator는 해당 구성 요소의 라이프사이클 관리 세부사항을 숨깁니다. Operator는 단일 구성 요소 또는 수십 개의 구성 요소를 관리할 수 있지만 최종 목표는 항상 일반적인 작업을 자동화하여 운영 부담을 줄이는 것입니다.

## 추가 리소스

- [클러스터 Operator 참조](#)

### 6.3.2. 애드온 Operator

OLM(Operator Lifecycle Manager) 및 OperatorHub는 Kubernetes 네이티브 애플리케이션을 Operator로 관리하는 데 도움이 되는 OpenShift Container Platform의 기본 구성 요소입니다. 함께 클러스터에서 사용 가능한 선택적 애드온 Operator를 검색, 설치 및 관리하기 위한 시스템을 제공합니다.

OpenShift Container Platform 웹 콘솔의 OperatorHub를 사용하여 클러스터 관리자 및 권한이 부여된 사용자는 Operator 카탈로그에서 설치할 Operator를 선택할 수 있습니다. OperatorHub에서 Operator를 설치한 후에는 전역 또는 특정 네임스페이스에서 사용자 애플리케이션에서 실행할 수 있도록 할 수 있습니다.

Red Hat Operator, 인증된 Operator, 커뮤니티 Operator가 포함된 기본 카탈로그 소스를 사용할 수 있습니다. 클러스터 관리자는 사용자 정의 Operator 세트를 포함할 수 있는 자체 사용자 정의 카탈로그 소스를 추가할 수도 있습니다.

개발자는 Operator SDK를 사용하여 OLM 기능을 활용하는 사용자 정의 Operator를 작성할 수 있습니다. 그런 다음 Operator를 번들로 제공하고 사용자 정의 카탈로그 소스에 추가할 수 있습니다. 이 소스는 클러스터에 추가하고 사용자가 사용할 수 있도록 합니다.



#### 참고

OLM은 OpenShift Container Platform 아키텍처를 구성하는 클러스터 Operator를 관리하지 않습니다.

## 추가 리소스

- OpenShift Container Platform에서 애드온 Operator를 실행하는 방법에 대한 자세한 내용은 [OLM\(Operator Lifecycle Manager\) 및 OperatorHub에 대한 Operator](#) 가이드 섹션을 참조하십시오.
- Operator SDK에 대한 자세한 내용은 [Operator 개발을 참조하십시오](#).

### 6.3.3. 플랫폼 Operator (기술 프리뷰)



#### 중요

platform Operator 유형은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

OLM(Operator Lifecycle Manager)에는 플랫폼 Operators라는 새로운 유형의 Operator가 도입되었습니다. Platform Operator는 OpenShift Container Platform 클러스터의 Day 0 작업 중 또는 이후에 설치할 수 있고 클러스터의 라이프사이클에 참여할 수 있는 OLM 기반 Operator입니다. 클러스터 관리자는 플랫폼 Operator를 사용하여 요구 사항 및 사용 사례에 맞게 OpenShift Container Platform 설치를 추가로 사용자 지정할 수 있습니다.



클러스터 관리자는 OpenShift Container Platform의 기존 클러스터 기능을 사용하여 클러스터 설치 전에 초기 페이로드에 필수적이지 않은 것으로 간주되는 CVO(Cluster Version Operator-based) 구성 요소의 하위 세트를 이미 비활성화할 수 있습니다. Platform Operator는 추가 사용자 지정 옵션을 제공하여 이 모델을 반복합니다. RukPak 구성 요소의 리소스에 의존하는 플랫폼 Operator 메커니즘을 통해 이제 클러스터 설치 시 OLM 기반 Operator를 설치할 수 있으며 Operator가 성공적으로 설치되지 않으면 클러스터를 아웃을 차단할 수 있습니다.

OpenShift Container Platform 4.15에서 이 기술 프리뷰 릴리스는 기본 플랫폼 Operator 메커니즘에 중점을 두고 있으며 향후 릴리스에서 개념을 확장하기 위한 기반을 구축합니다. 클러스터 전체 PlatformOperator API를 사용하여 **TechPreviewNoUpgrade** 기능 세트를 활성화한 클러스터에서 클러스터 생성 후에 Operator를 구성할 수 있습니다.

#### 추가 리소스

- 플랫폼 Operator 관리
- 플랫폼 Operator의 기술 프리뷰 제한 사항
- RukPak 구성 요소 및 패키징 형식
- 클러스터 기능

## 6.4. ETCD 개요

etcd는 메모리에 완전히 들어갈 수 있는 적은 양의 데이터를 보유하는 일관된 분산 키-값 저장소입니다. etcd는 많은 프로젝트의 핵심 구성 요소이지만 Kubernetes의 기본 데이터 저장소이며 이는 컨테이너 오케스트레이션의 표준 시스템입니다.

### 6.4.1. etcd를 사용하는 이점

etcd를 사용하면 다음과 같은 여러 가지 방법으로 이점을 누릴 수 있습니다.

- 클라우드 네이티브 애플리케이션의 일관된 가동 시간을 유지하고 개별 서버가 실패하더라도 계속 작동합니다.
- Kubernetes의 모든 클러스터 상태 저장 및 복제
- 구성 데이터를 배포하여 노드 구성에 중복 및 복원력 제공

### 6.4.2. etcd 작동 방식

클러스터 구성 및 관리에 대한 안정적인 접근 방식을 보장하기 위해 etcd는 etcd Operator를 사용합니다. Operator는 OpenShift Container Platform과 같은 Kubernetes 컨테이너 플랫폼에서 etcd 사용을 간소화합니다. etcd Operator를 사용하면 etcd 멤버를 만들거나 삭제하고, 클러스터의 크기를 조정하고, 백업을 수행하고, etcd를 업그레이드할 수 있습니다.

etcd Operator는 다음을 관찰, 분석 및 작동합니다.

1. Kubernetes API를 사용하여 클러스터 상태를 관찰합니다.
2. 현재 상태와 원하는 상태 간의 차이점을 분석합니다.
3. etcd 클러스터 관리 API, Kubernetes API 또는 둘 다를 통해 차이점을 수정합니다.

etcd에는 클러스터 상태가 있으며 이는 지속적으로 업데이트됩니다. 이 상태는 지속적으로 유지되므로 높

은 빈도로 인해 작은 변화가 많이 발생합니다. 따라서 빠르고 대기 시간이 짧은 I/O로 etcd 클러스터 멤버를 백업하는 것이 중요합니다. etcd의 모범 사례에 대한 자세한 내용은 "etcd에 대한 권장 사례"를 참조하십시오.

#### 추가 리소스

- [etcd 관련 권장 사례](#)
- [etcd 백업](#)

## 6.5. 호스트된 컨트롤 플레인 소개

Red Hat OpenShift Container Platform에 호스팅되는 컨트롤 플레인을 사용하여 관리 비용을 줄이고 클러스터 배포 시간을 최적화하며 관리 및 워크로드 문제를 분리하여 애플리케이션에 집중할 수 있습니다.

호스팅되는 컨트롤 플레인은 다음 플랫폼에서 [Kubernetes Operator 버전 2.0 이상](#)에 [다중 클러스터 엔진](#)을 사용하여 사용할 수 있습니다.

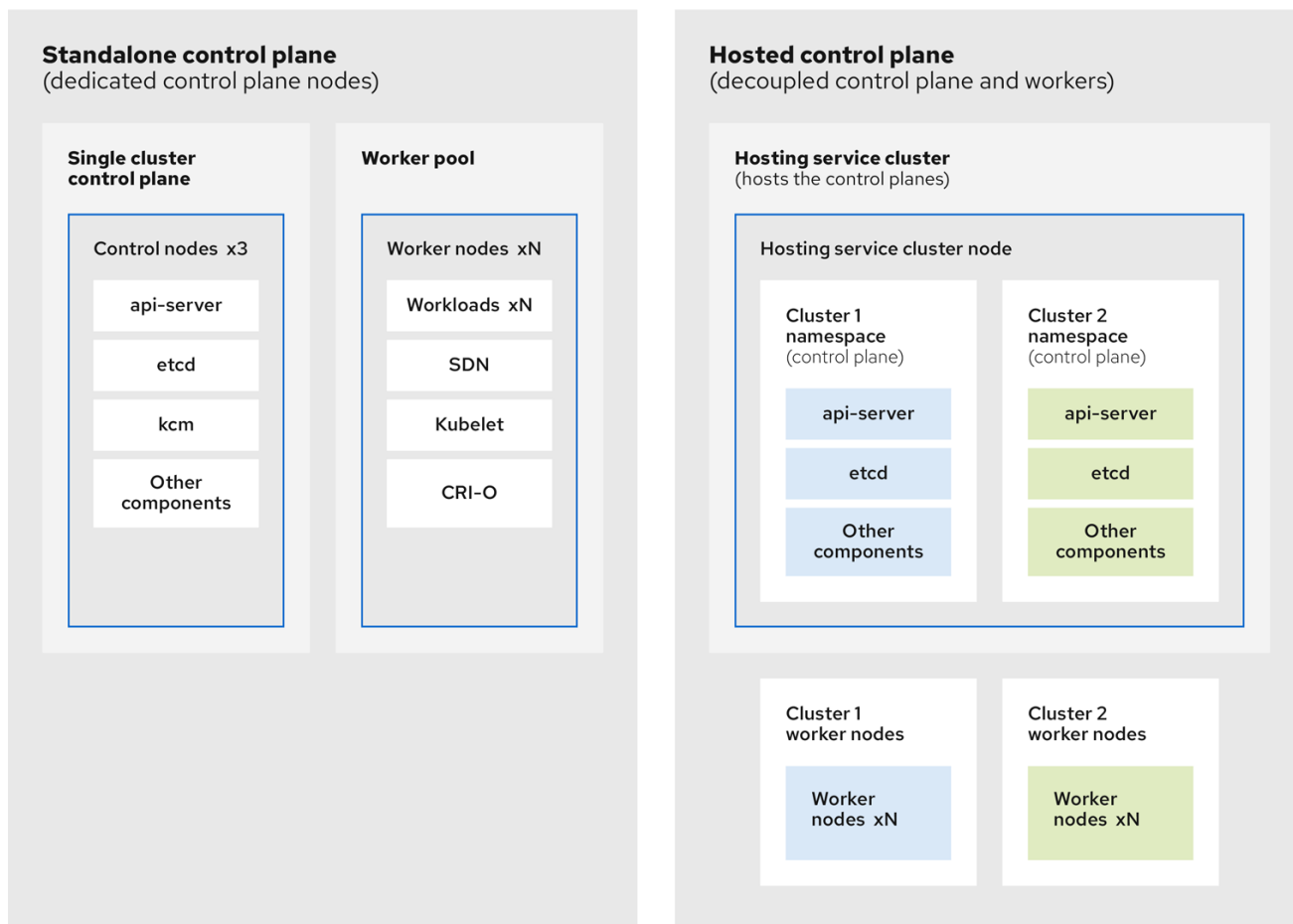
- 에이전트 공급자를 사용하여 베어 메탈
- OpenShift Virtualization: 연결된 환경에서 일반적으로 사용 가능한 기능으로, 연결이 끊긴 환경의 기술 프리뷰 기능
- AWS(Amazon Web Services) - 기술 프리뷰 기능
- IBM Z, 기술 프리뷰 기능
- IBM Power, 기술 프리뷰 기능

### 6.5.1. 호스트된 컨트롤 플레인 아키텍처

OpenShift Container Platform은 종종 클러스터가 컨트롤 플레인과 데이터 플레인으로 구성된 결합형 또는 독립 실행형 모델로 배포됩니다. 컨트롤 플레인에는 상태를 확인하는 API 끝점, 스토리지 끝점, 워크로드 스케줄러, 작업자가 포함됩니다. 데이터 플레인에는 워크로드 및 애플리케이션이 실행되는 컴퓨팅, 스토리지 및 네트워킹이 포함됩니다.

독립 실행형 컨트롤 플레인은 쿼럼을 보장하기 위해 최소 수를 사용하여 물리적 또는 가상 노드 전용 그룹에 의해 호스팅됩니다. 네트워크 스택이 공유됩니다. 클러스터에 대한 관리자 액세스는 클러스터의 컨트롤 플레인, 머신 관리 API 및 클러스터 상태에 기여하는 기타 구성 요소를 시각화할 수 있습니다.

독립 실행형 모델이 제대로 작동하지만 일부 상황에서는 컨트롤 플레인 및 데이터 플레인이 분리되는 아키텍처가 필요합니다. 이러한 경우 데이터 플레인은 전용 물리적 호스팅 환경을 사용하는 별도의 네트워크 도메인에 있습니다. 컨트롤 플레인은 Kubernetes의 네이티브 배포 및 상태 저장 세트와 같은 고급 프리미티브를 사용하여 호스팅됩니다. 컨트롤 플레인은 다른 워크로드로 처리됩니다.



272\_OpenShift\_1122

### 6.5.2. 호스팅된 컨트롤 플레인의 장점

OpenShift Container Platform의 호스팅된 컨트롤 플레인을 사용하면 진정한 하이브리드 클라우드 접근 방식을 구축하고 다른 여러 가지 이점을 누릴 수 있습니다.

- 컨트롤 플레인이 분리되고 전용 호스팅 서비스 클러스터에서 호스팅되므로 관리와 워크로드 간의 보안 경계가 더욱 강화됩니다. 결과적으로 클러스터의 인증 정보를 다른 사용자에게 유출될 가능성이 줄어듭니다. 인프라 시크릿 계정 관리도 분리되므로 클러스터 인프라 관리자는 실수로 컨트롤 플레인 인프라를 삭제할 수 없습니다.
- 호스팅된 컨트롤 플레인을 사용하면 더 적은 수의 노드에서 많은 컨트롤 플레인을 실행할 수 있습니다. 이로 인해 클러스터 비용이 더 경제적이 됩니다.
- 컨트롤 플레인 은 OpenShift Container Platform에서 시작되는 Pod로 구성되므로 컨트롤 플레인이 빠르게 시작됩니다. 모니터링, 로깅, 자동 확장과 같은 컨트롤 플레인 및 워크로드에 동일한 원칙이 적용됩니다.
- 인프라 화면에서 레지스트리, HAProxy, 클러스터 모니터링, 스토리지 노드 및 기타 인프라 구성 요소를 테넌트의 클라우드 공급자 계정으로 푸시하여 테넌트에서 사용을 분리할 수 있습니다.
- 운영 화면에서 다중 클러스터 관리는 더 중앙 집중화되어 클러스터 상태 및 일관성에 영향을 미치는 외부 요인이 줄어듭니다. 사이트 안정성 엔지니어는 문제를 디버그하고 클러스터 데이터 플레인으로 이동하여 문제 해결 시간 (TTR: Time to Resolution)이 단축되고 생산성 향상으로 이어질 수 있습니다.

- [호스팅된 컨트롤 플레인](#)

### 6.5.3. 호스팅된 컨트롤 플레인의 일반 개념 및 가상 사용자집

OpenShift Container Platform에 호스팅되는 컨트롤 플레인을 사용하는 경우 주요 개념과 관련 가상 사용자를 이해하는 것이 중요합니다.

#### 6.5.3.1. 개념

##### 호스트된 클러스터

관리 클러스터에서 호스팅되는 컨트롤 플레인 및 API 끝점이 있는 OpenShift Container Platform 클러스터입니다. 호스트된 클러스터에는 컨트롤 플레인과 해당 데이터 플레인이 포함됩니다.

##### 호스트된 클러스터 인프라

테넌트 또는 최종 사용자 클라우드 계정에 존재하는 네트워크, 컴퓨팅 및 스토리지 리소스입니다.

##### 호스트된 컨트롤 플레인

호스팅된 클러스터의 API 끝점에 의해 노출되는 관리 클러스터에서 실행되는 OpenShift Container Platform 컨트롤 플레인입니다. 컨트롤 플레인의 구성 요소에는 etcd, Kubernetes API 서버, Kubernetes 컨트롤러 관리자 및 VPN이 포함됩니다.

##### 호스트 클러스터

*관리 클러스터를 참조하십시오.*

##### 관리형 클러스터

허브 클러스터가 관리하는 클러스터입니다. 이 용어는 Kubernetes Operator의 다중 클러스터 엔진에서 Red Hat Advanced Cluster Management에서 관리하는 클러스터 라이프사이클에 따라 다릅니다. 관리형 클러스터는 *관리 클러스터*와 동일하지 않습니다. 자세한 내용은 [관리 클러스터](#)를 참조하십시오.

##### 관리 클러스터

HyperShift Operator가 배포되고 호스팅된 클러스터의 컨트롤 플레인이 호스팅되는 OpenShift Container Platform 클러스터입니다. 관리 클러스터는 *호스팅 클러스터*와 동일합니다.

##### 관리 클러스터 인프라

관리 클러스터의 네트워크, 컴퓨팅 및 스토리지 리소스입니다.

##### 노드 풀

컴퓨팅 노드가 포함된 리소스입니다. 컨트롤 플레인에는 노드 풀이 포함되어 있습니다. 컴퓨팅 노드는 애플리케이션 및 워크로드를 실행합니다.

### 6.5.3.2. 가상 사용자

#### 클러스터 인스턴스 관리자

이 역할의 사용자는 독립 실행형 OpenShift Container Platform의 관리자와 동일한 것으로 간주됩니다. 이 사용자에게는 프로비저닝된 클러스터에 **cluster-admin** 역할이 있지만 클러스터 업데이트 또는 구성 시 전원이 켜지지 않을 수 있습니다. 이 사용자는 클러스터에 예상된 일부 구성을 확인하기 위해 읽기 전용 액세스 권한이 있을 수 있습니다.

#### 클러스터 인스턴스 사용자

이 역할의 사용자는 독립 실행형 OpenShift Container Platform의 개발자와 동일한 것으로 간주됩니다. 이 사용자에게는 OperatorHub 또는 머신에 대한 보기가 없습니다.

#### 클러스터 서비스 소비자

이 역할에서 컨트롤 플레인 및 작업자 노드를 요청하거나, 드라이브 업데이트를 요청하거나, 외부화된 구성을 수정할 수 있다고 가정합니다. 일반적으로 이 사용자는 클라우드 인증 정보 또는 인프라 암호화를 관리하거나 액세스하지 않습니다. 클러스터 서비스 소비자 가상 사용자는 호스트된 클러스터를

요청하고 노드 풀과 상호 작용할 수 있습니다. 이 역할에는 논리 경계 내에서 호스팅된 클러스터 및 노드 풀을 생성, 읽기, 업데이트 또는 삭제하는 RBAC가 있다고 가정합니다.

### 클러스터 서비스 공급자

이 역할에는 일반적으로 관리 클러스터에 대한 **cluster-admin** 역할이 있고 HyperShift Operator의 가용성을 모니터링하고 소유할 수 있는 RBAC와 테넌트의 호스트 클러스터에 대한 컨트롤 플레인이 있는 사용자입니다. 클러스터 서비스 공급자 개인은 다음 예제를 포함하여 여러 활동을 담당합니다.

- 컨트롤 플레인 가용성, 가동 시간 및 안정성에 대한 서비스 수준 오브젝트 보유
- 컨트롤 플레인을 호스팅할 관리 클러스터의 클라우드 계정 구성
- 사용 가능한 컴퓨팅 리소스에 대한 호스트 인식이 포함된 사용자 프로비저닝 인프라 구성

### 6.5.4. 호스팅된 컨트롤 플레인 버전 관리

OpenShift Container Platform의 각 메이저, 마이너 또는 패치 버전 릴리스에서 호스팅되는 컨트롤 플레인의 두 가지 구성 요소가 릴리스됩니다.

- The HyperShift Operator
- **hcp** 명령줄 인터페이스(CLI)

HyperShift Operator는 **HostedCluster** API 리소스로 표시되는 호스팅 클러스터의 라이프사이클을 관리합니다. HyperShift Operator는 각 OpenShift Container Platform 릴리스와 함께 릴리스됩니다.

HyperShift Operator는 **hypershift** 네임스페이스에 **supported-versions** 구성 맵을 생성합니다. 구성 맵에는 지원되는 호스팅 클러스터 버전이 포함되어 있습니다.

동일한 관리 클러스터에서 다른 버전의 컨트롤 플레인을 호스팅할 수 있습니다.

#### supported-versions 구성 맵 오브젝트의 예

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.15"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

**hcp** CLI를 사용하여 호스팅된 클러스터를 생성할 수 있습니다.

**HostedCluster** 및 **NodePool** 과 같은 **hypershift.openshift.io** API 리소스를 사용하여 대규모로 OpenShift Container Platform 클러스터를 생성하고 관리할 수 있습니다. **HostedCluster** 리소스에는 컨트롤 플레인 및 일반 데이터 플레인 구성이 포함되어 있습니다. **HostedCluster** 리소스를 생성할 때 연결된 노드가 없는 완전히 작동하는 컨트롤 플레인이 있습니다. **NodePool** 리소스는 **HostedCluster** 리소스에 연결된 확장 가능한 작업자 노드 집합입니다.

API 버전 정책은 일반적으로 [Kubernetes API 버전 관리](#) 정책과 일치합니다.

## 7장. NVIDIA GPU 아키텍처 개요

NVIDIA는 OpenShift Container Platform에서 GPU(그래픽 처리 장치) 리소스 사용을 지원합니다. OpenShift Container Platform은 대규모 Kubernetes 클러스터를 배포 및 관리하기 위해 Red Hat에서 개발하고 지원하는 보안 중심 및 강화된 Kubernetes 플랫폼입니다. OpenShift Container Platform에는 Kubernetes에 대한 개선 사항이 포함되어 있으므로 사용자가 NVIDIA GPU 리소스를 쉽게 구성하고 사용하여 워크로드를 가속화할 수 있습니다.

NVIDIA GPU Operator는 OpenShift Container Platform 내에서 Operator 프레임워크를 활용하여 GPU 가속 워크로드를 실행하는 데 필요한 NVIDIA 소프트웨어 구성 요소의 전체 라이프사이클을 관리합니다.

이러한 구성 요소에는 NVIDIA 드라이버( CUDA 활성화), GPU용 Kubernetes 장치 플러그인, NVIDIA 컨테이너 툴킷, GPU 기능 검색을 사용한 자동 노드 태그 지정, DCGM 기반 모니터링 등이 포함됩니다.



### 참고

NVIDIA GPU Operator는 NVIDIA에서만 지원됩니다. NVIDIA에서 지원을 얻는 방법에 대한 자세한 내용은 [NVIDIA에서 지원 받기](#)를 참조하십시오.

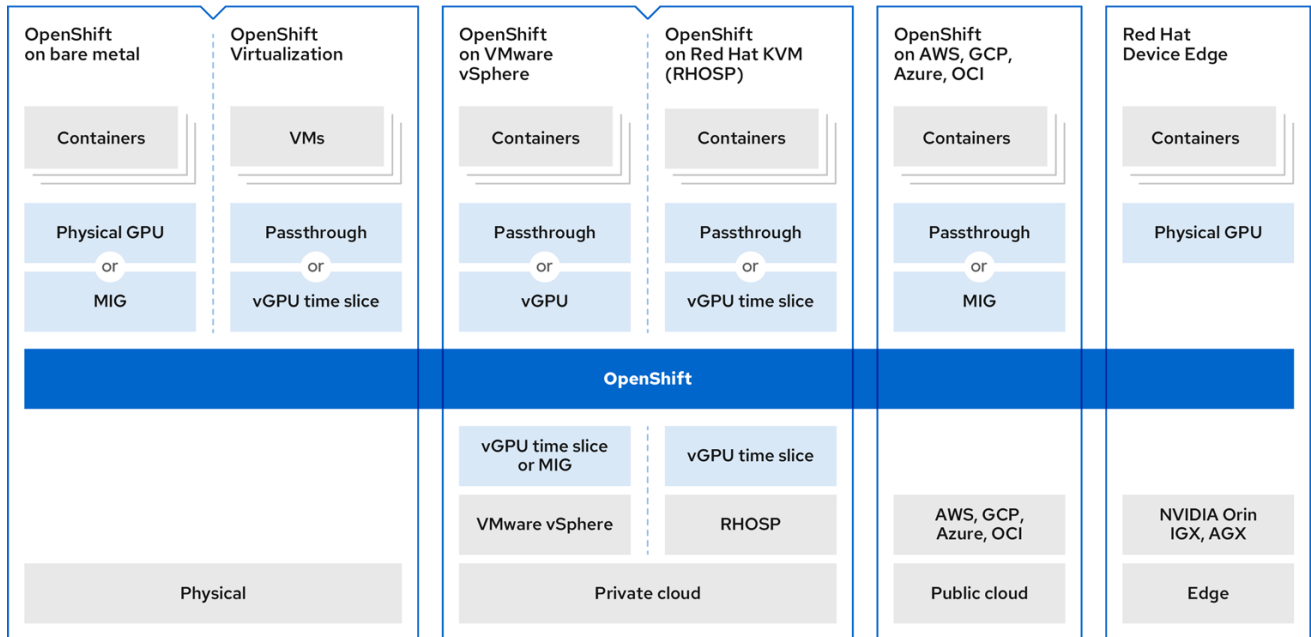
### 7.1. NVIDIA GPU 사전 요구 사항

- GPU 작업자 노드가 하나 이상 있는 작동 중인 OpenShift 클러스터입니다.
- 필요한 단계를 수행하려면 **cluster-admin** 으로 OpenShift 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있어야 합니다.
- NFD(노드 기능 검색) Operator가 설치되고 **nodefeaturediscovery** 인스턴스가 생성됩니다.

### 7.2. NVIDIA GPU 사용

다음 다이어그램은 OpenShift에 GPU 아키텍처를 활성화하는 방법을 보여줍니다.

그림 7.1. NVIDIA GPU 사용



512\_OpenShift\_I223



## 참고

Cheg는 A30, A100, A100X, A800, AX800, H100 및 H800에서만 지원됩니다.

## 7.2.1. GPU 및 베어 메탈

NVIDIA 인증 베어 메탈 서버에 OpenShift Container Platform을 배포할 수 있지만 몇 가지 제한 사항이 있습니다.

- 컨트롤 플레인 노드는 CPU 노드일 수 있습니다.
- 이러한 작업자 노드에서 AI/ML 워크로드가 실행되도록 작업자 노드는 GPU 노드여야 합니다. 또한 작업자 노드는 하나 이상의 GPU를 호스팅할 수 있지만 동일한 유형이어야 합니다. 예를 들어 노드에는 두 개의 NVIDIA A100 GPU가 있을 수 있지만 A100 GPU 1개와 T4 GPU가 1개인 노드는 지원되지 않습니다. Kubernetes용 NVIDIA 장치 플러그인은 동일한 노드에서 다른 GPU 모델 혼합을 지원하지 않습니다.
- OpenShift를 사용하는 경우 하나 이상의 서버가 필요합니다. 두 개의 서버가 있는 클러스터는 지원되지 않습니다. 단일 서버 배포는 단일 노드 openShift(SNO)라고 하며 이 구성을 사용하면 고가용성 OpenShift 환경이 생성됩니다.

컨테이너화된 GPU에 액세스하기 위해 다음 방법 중 하나를 선택할 수 있습니다.

- GPU 패스스루
- MIG (Multi-Instance GPU)

## 추가 리소스

- [베어 메탈 스택에서 Red Hat OpenShift](#)

## 7.2.2. GPU 및 가상화



많은 개발자와 기업이 컨테이너화된 애플리케이션 및 서버리스 인프라로 이동하고 있지만 VM(가상 머신)에서 실행되는 애플리케이션을 개발하고 유지 관리하는 데 많은 관심이 있습니다. Red Hat OpenShift Virtualization은 이러한 기능을 통해 기업은 클러스터 내의 컨테이너화된 워크플로우에 VM을 통합할 수 있습니다.

다음 방법 중 하나를 선택하여 작업자 노드를 GPU에 연결할 수 있습니다.

- VM(가상 머신) 내에서 GPU 하드웨어에 액세스하고 사용하기 위한 GPU 패스스루입니다.
- GPU 컴퓨팅 용량이 워크로드에 의해 포화되지 않은 경우 GPU(vGPU) 시간 분할입니다.

#### 추가 리소스

- [OpenShift Virtualization을 사용하는 NVIDIA GPU Operator](#)

### 7.2.3. GPU 및 vSphere

다양한 GPU 유형을 호스팅할 수 있는 NVIDIA 인증 VMware vSphere 서버에 OpenShift Container Platform을 배포할 수 있습니다.

VM에서 vGPU 인스턴스를 사용하는 경우 NVIDIA GPU 드라이버를 하이퍼바이저에 설치해야 합니다. VMware vSphere의 경우 이 호스트 드라이버는 VIB 파일 형식으로 제공됩니다.

작업자 노드 VM에 할당할 수 있는 최대 vGPUS 수는 vSphere 버전에 따라 다릅니다.

- vSphere 7.0: VM당 최대 4개의 vGPU
- vSphere 8.0: VM당 최대 8개의 vGPU



#### 참고

vSphere 8.0에서는 VM과 관련된 전체 또는 소수의 이기종 프로필을 여러 개 지원 합니다.

작업자 노드를 GPU에 연결할 다음 방법 중 하나를 선택할 수 있습니다.

- VM(가상 머신) 내에서 GPU 하드웨어에 액세스하고 사용하기 위한 GPU 패스스루
- GPU (vGPU) 시간 분할, 일부 GPU가 필요하지 않은 경우

베어 메탈 배포와 유사하게 하나 이상의 서버가 필요합니다. 두 개의 서버가 있는 클러스터는 지원되지 않습니다.

#### 추가 리소스

- [NVIDIA vGPU가 있는 VMware vSphere의 OpenShift Container Platform](#)

### 7.2.4. GPU 및 Red Hat KVM

NVIDIA 인증 KVM(커널 기반 가상 머신) 서버에서 OpenShift Container Platform을 사용할 수 있습니다.

베어 메탈 배포와 유사하게 하나 이상의 서버가 필요합니다. 두 개의 서버가 있는 클러스터는 지원되지 않습니다.



그러나 베어 메탈 배포와 달리 서버에서 다양한 유형의 GPU를 사용할 수 있습니다. 이는 Kubernetes 노드 역할을 하는 다른 VM에 이러한 GPU를 할당할 수 있기 때문입니다. 유일한 제한 사항은 Kubernetes 노드에 자체 수준에서 동일한 GPU 유형 세트가 있어야 한다는 것입니다.

컨테이너화된 GPU에 액세스하기 위해 다음 방법 중 하나를 선택할 수 있습니다.

- VM(가상 머신) 내에서 GPU 하드웨어에 액세스하고 사용하기 위한 GPU 패스스루
- GPU(vGPU)의 일부 GPU가 필요한 것은 아닙니다.

vGPU 기능을 활성화하려면 호스트 수준에 특수 드라이버를 설치해야 합니다. 이 드라이버는 RPM 패키지로 제공됩니다. 이 호스트 드라이버는 GPU 패스스루 할당에는 전혀 필요하지 않습니다.

#### 추가 리소스

- [KVM에 OpenShift Container Platform 4.13을 배포하는 방법](#)

### 7.2.5. GPU 및 CSP

주요 클라우드 서비스 공급자(CSP), AWS(Amazon Web Services), Google Cloud 또는 Microsoft Azure 중 하나에 OpenShift Container Platform을 배포할 수 있습니다.

완전히 관리되는 배포와 자체 관리 배포의 두 가지 작업 모드를 사용할 수 있습니다.

- 완전 관리형 배포에서 CSP와 협력하여 모든 것이 Red Hat에 의해 자동화됩니다. CSP 웹 콘솔을 통해 OpenShift 인스턴스를 요청할 수 있으며, Red Hat에서 클러스터가 자동으로 생성되고 완전히 관리됩니다. 환경의 노드 실패 또는 오류에 대해 우려할 필요가 없습니다. Red Hat은 클러스터 가동 시간을 완전히 유지합니다. 완전 관리형 서비스는 AWS, Azure 및 Google Cloud에서 사용할 수 있습니다. AWS의 경우 OpenShift 서비스를 ROSA(AWS의 Red Hat OpenShift Service)라고 합니다. Azure의 경우 이 서비스를 Azure Red Hat OpenShift라고 합니다. Google Cloud의 경우 이 서비스를 Google Cloud에서 OpenShift Dedicated라고 합니다.
- 자체 관리 배포에서 OpenShift 클러스터를 인스턴스화하고 유지보수해야 합니다. Red Hat은 이 경우 OpenShift 클러스터 배포를 지원하는 OpenShift-install 유틸리티를 제공합니다. 자체 관리 서비스는 모든 CSP에서 전역적으로 사용할 수 있습니다.

이 컴퓨팅 인스턴스는 GPU 가속 컴퓨팅 인스턴스이고 GPU 유형이 NVIDIA AI Enterprise의 지원되는 GPU 목록과 일치해야 합니다. 예를 들어 T4, V100 및 A100은 이 목록의 일부입니다.

컨테이너화된 GPU에 액세스하기 위해 다음 방법 중 하나를 선택할 수 있습니다.

- VM(가상 머신) 내에서 GPU 하드웨어에 액세스하고 사용하기 위한 GPU 패스스루입니다.
- 전체 GPU가 필요하지 않은 경우 GPU(vGPU) 시간 분할입니다.

#### 추가 리소스

- [클라우드의 Red Hat Openshift](#)

### 7.2.6. GPU 및 Red Hat Device Edge

Red Hat Device Edge는 MicroShift에 대한 액세스를 제공합니다. MicroShift는 리소스가 제한적인 (지대) 컴퓨팅에 필요한 기능 및 서비스를 통해 단일 노드 배포의 단순성을 제공합니다. Red Hat Device Edge는 리소스가 제한적인 환경에 배포된 베어 메탈, 가상, 컨테이너화된 Kubernetes 워크로드의 요구 사항을 충족합니다.

Red Hat Device Edge 환경에서 컨테이너에서 NVIDIA GPU를 활성화할 수 있습니다.

GPU 패스스루를 사용하여 컨테이너화된 GPU에 액세스합니다.

#### 추가 리소스

- [Red Hat Device Edge에서 NVIDIA GPU를 사용하여 워크로드를 가속화하는 방법](#)

## 7.3. GPU 공유 방법

Red Hat과 NVIDIA는 엔터프라이즈급 OpenShift Container Platform 클러스터에서 GPU 가속 컴퓨팅을 단순화하기 위해 GPU 동시성 및 공유 메커니즘을 개발했습니다.

일반적으로 애플리케이션에는 GPU를 사용하지 않는 다른 컴퓨팅 요구 사항이 있습니다. 각 워크로드에 적합한 양의 컴퓨팅 리소스를 제공하는 것은 배포 비용을 줄이고 GPU 사용률을 극대화하는 데 중요합니다.

가상화를 포함한 프로그래밍 모델 API에서 시스템 소프트웨어 및 하드웨어 파티셔닝에 이르기까지 다양한 GPU 사용률을 개선하기 위한 동시성 메커니즘이 존재합니다. 다음 목록은 GPU 동시성 메커니즘을 보여줍니다.

- Compute Unified Device Architecture (CUDA) 스트림
- 시간 분할
- CUDA Multi-Process Service (MPS)
- 다중 인스턴스 GPU(MIG)
- vGPU를 사용한 가상화

다른 OpenShift Container Platform 시나리오에 GPU 동시성 메커니즘을 사용할 때 다음 GPU 공유 제안을 고려하십시오.

#### 베어 메탈

vGPU를 사용할 수 없습니다. CheG 지원 카드를 사용하는 것이 좋습니다.

#### VMs

vGPU가 최선의 선택입니다.

#### 베어 메탈에서 MIG가 없는 이전 NVIDIA 카드

시간 분할을 사용하는 것이 좋습니다.

#### 여러 GPU가 있고 패스스루 및 vGPU를 원하는 VM

별도의 VM을 사용하는 것이 좋습니다.

#### OpenShift Virtualization 및 여러 GPU를 사용한 베어 메탈

호스트된 VM에 패스스루를 사용하고 컨테이너에 시간 분할을 사용하는 것이 좋습니다.

#### 추가 리소스

- [GPU 사용률 개선](#)

### 7.3.1. CUDA 스트림

CUDA(Compute Unified Device Architecture)는 GPU의 일반 컴퓨팅을 위해 NVIDIA에서 개발한 병렬 컴퓨팅 플랫폼 및 프로그래밍 모델입니다.

스트림은 GPU에서 문제순으로 실행되는 일련의 작업입니다. CUDA 명령은 일반적으로 기본 스트림에서 순차적으로 실행되며 이전 작업이 완료될 때까지 작업이 시작되지 않습니다.

다양한 스트림의 작업을 비동기적으로 처리하면 병렬 작업을 실행할 수 있습니다. 한 스트림에서 발행된 작업은 다른 작업이 다른 스트림으로 발행되기 전, 중 또는 후에 실행됩니다. 이를 통해 GPU는 정해진 순서로 여러 작업을 동시에 실행하여 성능이 향상됩니다.

#### 추가 리소스

- [비동기 동시 실행](#)

### 7.3.2. 시간 분할

GPU 시간 분할은 여러 CUDA 애플리케이션을 실행할 때 과부하된 GPU에서 예약된 워크로드를 상호 저장합니다.

GPU의 복제본 세트를 정의하여 Kubernetes에서 GPU를 시간 분할할 수 있으며 각각 워크로드를 실행하기 위해 개별적으로 Pod에 배포할 수 있습니다. MIG(Multi-instance GPU)와 달리 복제본 간 메모리 또는 내결함성이 없지만 일부 워크로드의 경우 공유하지 않는 것보다 좋습니다. 내부적으로 GPU 시간 분할은 동일한 기본 GPU의 복제본에서 멀티플렉션에 사용됩니다.

시간 분할에 대한 클러스터 전체 기본 구성을 적용할 수 있습니다. 노드별 구성을 적용할 수도 있습니다. 예를 들어 Cryostat T4 GPU가 있는 노드에만 시간 분할 구성을 적용하고 다른 GPU 모델의 노드를 수정할 수 없습니다.

클러스터 전체 기본 구성을 적용한 다음 노드에 레이블을 지정하여 해당 노드에 노드별 구성을 지정하여 이러한 두 가지 접근 방식을 결합할 수 있습니다.

### 7.3.3. CUDA 다중 프로세스 서비스

CUDA Multi-Process Service(MPS)를 사용하면 단일 GPU가 여러 CUDA 프로세스를 사용할 수 있습니다. 프로세스는 GPU에서 병렬로 실행되며 GPU 컴퓨팅 리소스의 포화 상태를 제거합니다. 또한 MPS는 다양한 프로세스에서 커널 작업 및 메모리를 복사하여 사용률을 향상시킬 수 있습니다.

#### 추가 리소스

- [CUDA MPS](#)

### 7.3.4. 다중 인스턴스 GPU

MG(Multi-instance GPU)를 사용하여 GPU 컴퓨팅 단위와 메모리를 여러 MIG 인스턴스로 분할할 수 있습니다. 이러한 각 인스턴스는 시스템 관점에서 독립 실행형 GPU 장치를 나타내며 노드에서 실행되는 모든 애플리케이션, 컨테이너 또는 가상 머신에 연결할 수 있습니다. GPU를 사용하는 소프트웨어는 이러한 기타 각 인스턴스를 개별 GPU로 처리합니다.

MIG는 전체 GPU의 전체 성능이 필요하지 않은 애플리케이션이 있는 경우에 유용합니다. 새로운 NVIDIA Ampere 아키텍처의 MIG 기능을 사용하면 하드웨어 리소스를 여러 GPU 인스턴스로 분할할 수 있으며 각 GPU 인스턴스는 운영 체제에서 독립적인 CUDA 지원 GPU로 사용할 수 있습니다.

NVIDIA GPU Operator 버전 1.7.0 이상은 A100 및 A30 Ampere 카드에 대해 MIG 지원을 제공합니다. 이러한 GPU 인스턴스는 전용 하드웨어 리소스와 완전히 격리된 최대 7개의 독립 CUDA 애플리케이션을 지원하도록 설계되었습니다.

## 추가 리소스

- [NVIDIA Multi-Instance GPU 사용자 가이드](#)

### 7.3.5. vGPU를 사용한 가상화

가상 머신(VM)은 NVIDIA vGPU를 사용하여 단일 물리적 GPU에 직접 액세스할 수 있습니다. 엔터프라이즈 전체의 VM에서 공유하여 다른 장치에서 액세스할 수 있는 가상 GPU를 생성할 수 있습니다.

이 기능은 GPU 성능의 기능과 vGPU가 제공하는 관리 및 보안 이점을 결합합니다. vGPU가 제공하는 추가 이점으로 VM 환경에 대한 사전 관리 및 모니터링, 혼합 VDI 및 컴퓨팅 워크로드를 위한 워크로드 분산, 여러 VM의 리소스 공유 등이 포함됩니다.

## 추가 리소스

- [가상 GPU](#)

## 7.4. OPENSIFT CONTAINER PLATFORM 용 NVIDIA GPU 기능

### NVIDIA 컨테이너 툴킷

NVIDIA 컨테이너 툴킷을 사용하면 GPU 가속 컨테이너를 생성하고 실행할 수 있습니다. 툴킷에는 NVIDIA GPU를 사용하도록 컨테이너를 자동으로 구성하는 컨테이너 런타임 라이브러리 및 유틸리티가 포함되어 있습니다.

### NVIDIA AI Enterprise

NVIDIA AI Enterprise는 NVIDIA 인증 시스템에서 지원, 인증 및 지원되는 포괄적인 AI 및 데이터 분석 소프트웨어 제품군입니다.

NVIDIA AI Enterprise는 Red Hat OpenShift Container Platform을 지원합니다. 지원되는 설치 방법은 다음과 같습니다.

- GPU Passthrough가 있는 베어 메탈 또는 VMware vSphere의 OpenShift Container Platform.
- NVIDIA vGPU를 사용하는 VMware vSphere의 OpenShift Container Platform.

### GPU 기능 검색

Kubernetes용 NVIDIA GPU 기능 검색은 노드에서 사용 가능한 GPU에 대한 레이블을 자동으로 생성할 수 있는 소프트웨어 구성 요소입니다. GPU 기능 검색에서는 NFD(노드 기능 검색)를 사용하여 이 레이블을 수행합니다.

NFD(Node Feature Discovery Operator)는 하드웨어 관련 정보로 노드에 레이블을 지정하여 OpenShift Container Platform 클러스터에서 하드웨어 기능 및 구성 검색을 관리합니다. NFD는 PCI 카드, 커널, OS 버전과 같은 노드별 속성을 사용하여 호스트에 레이블을 지정합니다.

"Node Feature Discovery"를 검색하여 Operator Hub에서 NFD Operator를 찾을 수 있습니다.

### OpenShift Virtualization을 사용하는 NVIDIA GPU Operator

이 시점까지 GPU Operator는 GPU 가속 컨테이너를 실행하기 위해 작업자 노드만 프로비저닝했습니다. 이제 GPU Operator를 사용하여 GPU 가속 VM(가상 머신)을 실행하기 위해 작업자 노드를 프로비저닝할 수도 있습니다.

해당 노드에서 실행되도록 구성된 GPU 워크로드에 따라 작업자 노드에 다른 소프트웨어 구성 요소를 배포하도록 GPU Operator를 구성할 수 있습니다.

### GPU 모니터링 대시보드

모니터링 대시보드를 설치하여 OpenShift Container Platform 웹 콘솔의 클러스터 **모니터링** 페이지에 대한 GPU 사용 정보를 표시할 수 있습니다. GPU 사용률 정보에는 사용 가능한 GPU 수, 전력 소비(단위: 와트), 온도(단위: 섭씨), 사용률(%) 및 각 GPU에 대한 기타 메트릭이 포함됩니다.

#### 추가 리소스

- [NVIDIA 인증 시스템](#)
- [NVIDIA AI Enterprise](#)
- [NVIDIA 컨테이너 툴킷](#)
- [GPU 모니터링 대시보드 활성화](#)
- [OpenShift Container Platform에서 MIG 지원](#)
- [OpenShift에서 NVIDIA GPU 시간 분할](#)
- [연결이 끊긴 또는 Airgapped 환경에 GPU Operator 배포](#)
- [Node Feature Discovery Operator](#)

## 8장. OPENSIFT CONTAINER PLATFORM 개발 이해

엔터프라이즈급 애플리케이션을 개발하고 실행할 때 컨테이너의 기능을 완전히 활용하려면 컨테이너가 다음을 수행할 수 있게 하는 도구를 통해 환경을 지원해야 합니다.

- 컨테이너화되었거나 그렇지 않은 다른 서비스에 연결할 수 있는 개별 마이크로서비스로 생성됩니다. 예를 들어, 애플리케이션을 데이터베이스와 결합하거나 모니터링 애플리케이션을 데이터베이스에 연결할 수 있습니다.
- 복원력이 뛰어나서 서버가 충돌하거나 유지보수 또는 서비스 해제를 위해 서버를 가동 중단해야 하는 경우 컨테이너가 또 다른 머신에서 시작될 수 있습니다.
- 코드 변경 사항을 자동으로 가져온 다음 새 버전을 자체 시작 및 배포하도록 자동화되었습니다.
- 수요가 증가하면 더 많은 인스턴스가 클라이언트에 서비스를 제공하고 수요가 감소하면 인스턴스 수가 감소하도록 확장 또는 복제됩니다.
- 애플리케이션 유형에 따라 다른 방식으로 실행할 수 있습니다. 예를 들어 한 애플리케이션이 한 달에 한 번 실행되어 보고서를 생성한 다음 종료될 수 있습니다. 또 다른 애플리케이션은 지속적으로 실행되어야 하며 클라이언트가 사용할 수 있는 가능성이 높아야 합니다.
- 애플리케이션 상태를 보고 문제가 발생했을 때 대응할 수 있도록 관리되었습니다.

컨테이너가 광범위하게 사용되고, 결과적으로 엔터프라이즈를 지원하기 위한 도구 및 방법이 필요하게 됨에 따라 많은 옵션이 제공되었습니다.

이 섹션의 나머지 부분에서는 OpenShift Container Platform에서 컨테이너화된 Kubernetes 애플리케이션을 빌드 및 배포할 때 생성할 수 있는 자산 옵션에 대해 설명합니다. 또한 다양한 종류의 애플리케이션 및 개발 요구사항에 사용할 수 있는 방법에 대해서도 설명합니다.

### 8.1. 컨테이너화된 애플리케이션 개발 정보

여러 가지 방법으로 컨테이너를 사용하여 애플리케이션 개발에 접근할 수 있으며 상황에 따라 다른 접근 방식이 더 적합할 수 있습니다. 이러한 다양성 중 일부를 설명하기 위해 제시된 일련의 접근 방식은 단일 컨테이너를 개발하는 것으로 시작하여 궁극적으로 해당 컨테이너를 대기업의 미션 크리티컬 애플리케이션으로 배포하는 것입니다. 이러한 접근 방식은 컨테이너화된 애플리케이션 개발에 사용할 수 있는 다양한 도구, 형식 및 방법을 보여줍니다. 이 주제에서는 다음을 설명합니다.

- 간단한 컨테이너를 빌드하여 레지스트리에 저장
- Kubernetes 매니페스트를 생성하여 Git 리포지토리에 저장
- Operator가 다른 사용자와 애플리케이션을 공유하게 만들기

### 8.2. 간단한 컨테이너 빌드

애플리케이션에 대한 아이디어가 있으며 컨테이너화하고 싶은 경우를 살펴보겠습니다.

먼저 buildah 또는 docker와 같은 컨테이너를 빌드하는 도구와 컨테이너에 들어가는 내용을 설명하는 파일(일반적으로 [Dockerfile](#))이 필요합니다.

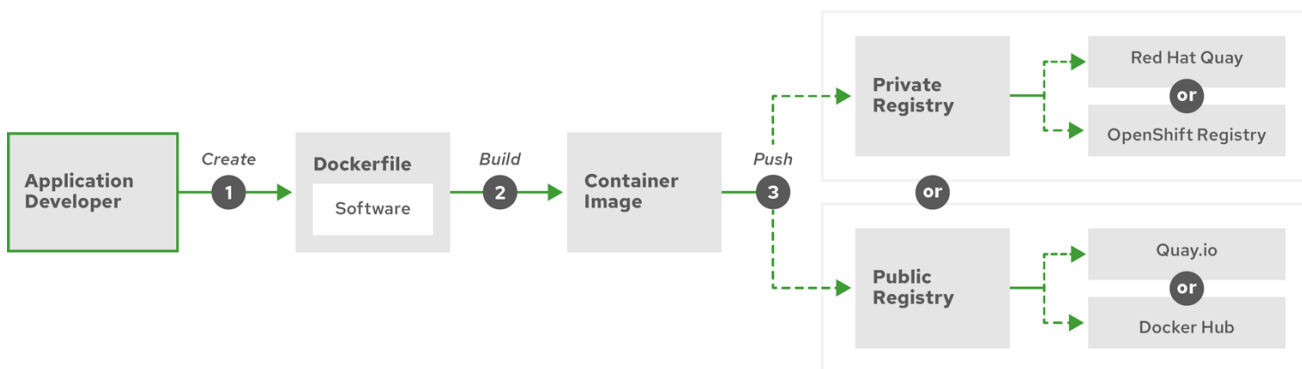
다음으로 결과 컨테이너 이미지를 넣을 위치가 필요하므로 실행하려는 위치로 끌어올 수 있습니다. 이 위치는 컨테이너 레지스트리입니다.

이러한 각 구성 요소의 몇 가지 예는 사용자가 직접 제공하는 Dockerfile을 제외하고 대부분의 Linux 운영 체제에 기본적으로 설치됩니다.



다음 다이어그램은 이미지를 작성하고 푸시하는 프로세스를 보여줍니다.

그림 8.1. 컨테이너화된 간단한 애플리케이션을 생성하여 레지스트리로 푸시



OpenShift\_25\_0519

RHEL(Red Hat Enterprise Linux)을 운영 체제로 실행하는 컴퓨터를 사용하는 경우 컨테이너화된 애플리케이션을 생성하는 프로세스에는 다음 단계가 필요합니다.

1. 컨테이너 빌드 도구 설치: RHEL에는 컨테이너를 빌드하고 관리하는 데 사용하는 podman, buildah 및 skopeo가 포함된 도구 세트가 포함되어 있습니다.
2. 기본 이미지와 소프트웨어를 결합하는 Dockerfile 생성: 컨테이너 빌드에 대한 정보는 **Dockerfile**이라는 파일에 입력됩니다. 이 파일에서 빌드한 기본 이미지, 설치한 소프트웨어 패키지 및 컨테이너에 복사한 소프트웨어를 식별합니다. 컨테이너 외부에 노출되는 네트워크 포트 및 컨테이너 내부에 마운트된 볼륨과 같은 매개변수 값도 식별합니다. RHEL 시스템의 디렉터리에 Dockerfile과 컨테이너화된 소프트웨어를 배치합니다.
3. buildah 또는 docker 빌드 실행: **buildah build-using-dockerfile** 또는 **docker build** 명령을 실행하여 선택한 기본 이미지를 가져와 로컬에 저장된 컨테이너 이미지를 생성합니다. buildah를 사용하여 Dockerfile 없이 컨테이너 이미지를 빌드할 수도 있습니다.
4. 태그 지정 및 레지스트리로 푸시: 컨테이너를 저장하고 공유할 레지스트리의 위치를 식별하는 새 컨테이너 이미지에 태그를 추가하십시오. 그런 다음 **podman push** 또는 **docker push** 명령을 실행하여 레지스트리에 이미지를 푸시합니다.
5. 이미지 가져오기 및 실행: podman 또는 docker와 같은 컨테이너 클라이언트 도구가 있는 시스템에서 새 이미지를 식별하는 명령을 실행하십시오. 예를 들어 **podman run <image\_name>** 또는 **docker run <image\_name>** 명령을 실행하십시오. 여기서 **<image\_name>**은 새 컨테이너 이미지의 이름이며, **quay.io/myrepo/myapp:latest**와 비슷합니다. 이미지를 푸시하고 가져오려면 레지스트리에 자격 증명이 필요할 수 있습니다.

컨테이너 이미지를 빌드하고 레지스트리에 푸시하는 프로세스에 대한 자세한 내용은 [Buildah를 사용한 사용자 정의 이미지 빌드](#)를 참조하십시오.

### 8.2.1. 컨테이너 빌드 도구 옵션

buildah, podman 및 skopeo를 사용하여 컨테이너를 빌드하고 관리하면 OpenShift Container Platform 또는 기타 Kubernetes 환경에서 컨테이너 배포를 위해 특별히 조정된 기능이 포함된 업계 표준 컨테이너 이미지가 생성됩니다. 이러한 틀은 데몬이 없으며 루트 권한 없이 실행할 수 있으므로 실행하기 위한 오버헤드가 줄어듭니다.



## 중요

컨테이너 런타임으로 Docker Container Engine에 대한 지원은 Kubernetes 1.20에서 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 그러나 Docker 생성 이미지는 CRI-O를 포함하여 모든 런타임에서 클러스터에서 계속 작동합니다. 자세한 내용은 [Kubernetes 블로그 공지](#)를 참조하십시오.

OpenShift Container Platform에서 컨테이너를 궁극적으로 실행할 때는 [CRI-O](#) 컨테이너 엔진을 사용합니다. CRI-O는 OpenShift Container Platform 클러스터의 모든 작업자 및 컨트롤 플레인 머신에서 실행되지만 CRI-O는 아직 OpenShift Container Platform 외부의 독립 실행형 런타임으로 지원되지 않습니다.

### 8.2.2. 기본 이미지 옵션

애플리케이션을 빌드하기 위해 선택한 기본 이미지에는 애플리케이션의 Linux 시스템과 유사한 소프트웨어 세트가 포함되어 있습니다. 고유 이미지를 빌드하면 소프트웨어가 해당 파일 시스템에 배치되고 해당 파일 시스템이 운영 체제를 보는 것처럼 인식합니다. 이 기본 이미지를 선택하면 향후 컨테이너의 안전성, 효율성 및 업그레이드 가능성에 큰 영향을 미칩니다.

Red Hat에서는 [Red Hat Universal Base Images \(UBI\)](#)라는 새로운 기본 이미지 세트를 제공합니다. 이 이미지는 Red Hat Enterprise Linux를 기반으로 하며 과거 Red Hat에서 제공한 기본 이미지와 유사하지만 한 가지 큰 차이점이 있습니다. 이는 Red Hat 서브스크립션이 없이 자유롭게 재배포할 수 있습니다. 결과적으로 공유 방법이나 환경에 따라 다른 이미지를 만드는 것에 대한 걱정 없이 UBI 이미지에서 애플리케이션을 빌드할 수 있습니다.

이 UBI 이미지에는 표준, 초기화 및 최소 버전이 있습니다. Node.js, Perl 또는 Python과 같은 특정 런타임 환경에 종속된 애플리케이션의 기초로 [Red Hat Software Collections](#) 이미지를 사용할 수도 있습니다. 이러한 런타임 기본 이미지 중 일부 특수 버전은 S2I(Source-to-Image) 이미지라고 합니다. S2I 이미지를 사용하면 해당 코드를 실행할 준비가 된 기본 이미지 환경에 코드를 삽입할 수 있습니다.

OpenShift Container Platform 웹 UI에서 직접 S2I 이미지를 사용할 수 있습니다. 개발자 화면에서 **+추가** 보기로 이동하고 **개발자 카탈로그** 타일에서 **개발자 카탈로그**에서 사용 가능한 모든 서비스를 확인합니다.

#### 그림 8.2. 특정 런타임이 필요한 앱에 맞는 S2I 기본 이미지 선택

The screenshot shows the OpenShift Developer Catalog interface. On the left is a sidebar with navigation options: Developer, +Add, Topology, Observe, Search, Builds, Helm, Project, ConfigMaps, and Secrets. The main area displays the 'Developer Catalog' for 'Project: myproject'. It includes a search bar and a list of items categorized by type (Builder Images, Helm Charts, Operator Backed, Templates). The items shown include .NET, .NET Application, a10networks-a10tkc, AlertmanagerConfig, Apache HTTP Server, and another Apache HTTP Server (httpd). Each item has a brief description and is provided by Red Hat.

### 8.2.3. 레지스트리 옵션



컨테이너 레지스트리에 컨테이너 이미지를 저장하면 다른 사용자와 공유하고 궁극적으로 실행되는 플랫폼에서 사용할 수 있게 됩니다. 무료 계정을 제공하는 대규모 공용 컨테이너 레지스트리 또는 추가 스토리지와 특수 기능을 제공하는 프리미엄 버전을 선택할 수 있습니다. 조직에 독점적이거나 다른 사용자와 선택적으로 공유할 수 있는 고유한 레지스트리도 설치할 수 있습니다.

Red Hat 이미지 및 인증된 파트너 이미지를 얻으려면 Red Hat Registry에서 그럴 수 있습니다. Red Hat Registry는 인증되지 않고 더 이상 사용되지 않는 [registry.access.redhat.com](https://registry.access.redhat.com)과 인증이 필요한 [registry.redhat.io](https://registry.redhat.io)의 두 위치로 표시됩니다. [Container images section of the Red Hat Ecosystem Catalog](#)에서 Red Hat Registry의 Red Hat 및 파트너 이미지에 관해 알아볼 수 있습니다. Red Hat 컨테이너 이미지를 나열하는 것 외에도 적용된 보안 업데이트를 기반으로 한 상태 점수를 포함하여 해당 이미지의 내용과 품질에 대한 광범위한 정보를 보여줍니다.

대규모 공개 레지스트리에는 [Docker Hub](#) 및 [Quay.io](#)가 있습니다. Quay.io 레지스트리는 Red Hat이 소유하고 관리합니다. OpenShift Container Platform에 사용되는 많은 구성 요소는 컨테이너 이미지 및 OpenShift Container Platform 자체를 배포하는 데 사용되는 Operator를 포함하여 Quay.io에 저장됩니다. Quay.io는 Helm 차트를 포함하여 다른 유형의 콘텐츠를 저장하는 수단도 제공합니다.

고유한 개인 컨테이너 레지스트리를 원하는 경우 OpenShift Container Platform 자체에는 OpenShift Container Platform과 함께 설치되고 해당 클러스터에서 실행되는 개인 컨테이너 레지스트리가 포함됩니다. Red Hat은 [Red Hat Quay](#) 라는 개인 Quay.io 레지스트리 버전도 제공합니다. Red Hat Quay에는 지역 복제, Git 빌드 트리거, Clair 이미지 스캔 및 기타 여러 기능이 포함되어 있습니다.

여기에 언급된 모든 레지스트리는 해당 레지스트리에서 이미지를 다운로드하기 위해 자격 증명이 필요할 수 있습니다. 이러한 인증서 중 일부는 OpenShift Container Platform에서 클러스터 전체에 제공되는 반면 다른 자격 증명은 개인에게 할당될 수 있습니다.

### 8.3. OPENSIFT CONTAINER PLATFORM용 KUBERNETES 매니페스트 생성

컨테이너 이미지는 컨테이너화된 애플리케이션의 기본 구성 요소이지만 OpenShift Container Platform과 같은 Kubernetes 환경에서 해당 애플리케이션을 관리하고 배포하려면 추가 정보가 필요합니다. 일반적으로 이미지를 생성한 후의 단계는 다음과 같습니다.

- Kubernetes 매니페스트에서 작업하는 데 사용할 다양한 리소스 이해
- 실행 중인 애플리케이션의 종류 결정
- 지원 구성 요소 수집
- Git 리포지토리에 매니페스트를 생성하고 저장하여 소스 버전 관리 시스템에 저장하고, 감사하며, 추적하고, 다음 환경으로 승격 및 배포하며, 필요한 경우 이전 버전으로 롤백하고 다른 사용자와 공유할 수 있습니다.

#### 8.3.1. Kubernetes 포드 및 서비스 정보

컨테이너 이미지는 docker가 있는 기본 단위이지만 Kubernetes가 작동하는 기본 단위는 [pod](#)라고 합니다. 포드는 애플리케이션을 빌드하는 다음 단계를 나타냅니다. 포드는 하나 이상의 컨테이너를 포함할 수 있습니다. 핵심은 포드가 배포, 스케일링 및 관리하는 단일 단위라는 것입니다.

확장성과 네임스페이스는 포드에 들어갈 내용을 결정할 때 고려해야 할 주요 항목입니다. 쉽게 배포하기 위해 포드에 컨테이너를 배포하고 자체 로깅 및 모니터링 컨테이너를 포드에 포함시킬 수 있습니다. 나중에 포드를 실행하고 추가 인스턴스를 스케일링해야 하는 경우 다른 컨테이너도 함께 확장됩니다. 네임스페이스의 경우 포드의 컨테이너는 동일한 네트워크 인터페이스, 공유 스토리지 볼륨 및 리소스 제한(예:

메모리 및 CPU)을 공유하므로, 포드의 콘텐츠를 단일 단위로 관리하기가 쉬워집니다. 포드의 컨테이너는 System V 세마포어 또는 POSIX 공유 메모리와 같은 표준 프로세스 간 통신을 사용하여 서로 통신할 수 있습니다.

개별 포드는 Kubernetes에서 확장 가능한 단위를 나타내지만 **서비스**는 포드 세트를 함께 그룹화하여 부하 분산과 같은 작업을 완료할 수 있는 완전하고 안정적인 애플리케이션을 생성하는 수단을 제공합니다. 또한 서비스는 삭제할 때까지 동일한 IP 주소에서 계속 사용할 수 있기 때문에 포드보다 영구적입니다. 서비스가 사용 중일 때 이름별로 요청되고 OpenShift Container Platform 클러스터는 해당 이름을 서비스를 구성하는 포드에 도달할 수 있는 IP 주소 및 포트로 해석합니다.

컨테이너화된 애플리케이션은 특성상 실행되는 운영 체제뿐 아니라 사용자와 분리됩니다. Kubernetes 매니페스트의 일부는 컨테이너화된 애플리케이션과의 통신을 세밀하게 제어할 수 있는 **네트워크 정책**을 정의하여 애플리케이션을 내부 및 외부 네트워크에 노출하는 방법을 설명합니다. HTTP, HTTPS 및 기타 서비스에 대한 수신 요청을 클러스터 외부에서 클러스터 내부 서비스로 연결하려면 **Ingress** 리소스를 사용할 수 있습니다.

컨테이너에 서비스를 통해 제공될 수 있는 데이터베이스 스토리지 대신 온-디스크 스토리지가 필요한 경우 매니페스트에 **볼륨**을 추가하여 해당 스토리지를 포드에서 사용하게 만들 수 있습니다. 영구 볼륨(PV)을 생성하거나 **포드** 정의에 추가되는 볼륨을 동적으로 생성하도록 매니페스트를 구성할 수 있습니다.

애플리케이션을 구성하는 Pod 그룹을 정의한 후 **Deployment** 및 **DeploymentConfig** 오브젝트에서 해당 Pod를 정의할 수 있습니다.

### 8.3.2. 애플리케이션 유형

다음으로 애플리케이션 유형이 애플리케이션 실행 방법에 어떤 영향을 미치는지 고려하십시오.

Kubernetes는 다양한 종류의 애플리케이션에 적합한 다양한 유형의 워크로드를 정의합니다. 애플리케이션에 적합한 워크로드를 판별하려면 애플리케이션이 다음과 같은지 고려하십시오.

- 완전히 실행되어 완료되어야 합니다. 예를 들어 보고서를 생성하기 시작하고 보고서가 완료되면 종료되는 애플리케이션이 있습니다. 그러면 애플리케이션이 한 달 동안 다시 실행되지 않을 수 있습니다. 이러한 유형의 애플리케이션에 적합한 OpenShift Container Platform 오브젝트에는 **Job** 및 **CronJob** 오브젝트가 포함됩니다.
- 지속적으로 실행될 것으로 예상됩니다. 장기 실행 애플리케이션의 경우 **배포**를 작성할 수 있습니다.
- 가용성이 높아야 합니다. 애플리케이션에 고가용성이 필요한 경우 둘 이상의 인스턴스가 있도록 배포 크기를 조정해야 합니다. **Deployment** 또는 **DeploymentConfig** 오브젝트는 해당 유형의 애플리케이션에 대한 **복제본 세트**를 통합할 수 있습니다. 복제본 세트를 사용하면 포드가 여러 노드에서 실행되어 작업자가 중단된 경우에도 애플리케이션을 항상 사용할 수 있습니다.
- 모든 노드에서 실행해야 합니다. 일부 유형의 Kubernetes 애플리케이션은 모든 마스터 또는 작업자 노드의 클러스터 자체에서 실행되도록 설계되었습니다. DNS 및 모니터링 애플리케이션은 모든 노드에서 지속적으로 실행해야 하는 애플리케이션의 예입니다. 이 유형의 애플리케이션을 **데몬 세트**로 실행할 수 있습니다. 노드 레이블을 기반으로 노드 서버 세트에서 데몬 세트를 실행할 수도 있습니다.
- 라이프사이클 관리가 필요합니다. 다른 사용자가 사용할 수 있도록 애플리케이션을 전달하려면 **Operator** 생성을 고려하십시오. Operator를 사용하면 지능적으로 빌드할 수 있으므로 백업 및 업그레이드와 같은 작업을 자동으로 처리할 수 있습니다. OLM(Operator Lifecycle Manager)과 함께 클러스터 관리자는 선택된 네임스페이스에 Operator를 노출시키므로 클러스터의 사용자가 이를 실행할 수 있습니다.
- ID 또는 번호 지정 요구사항이 있습니다. 애플리케이션에는 ID 요구사항이나 번호 지정 요구사항

이 있을 수 있습니다. 예를 들어, 정확히 3개의 애플리케이션 인스턴스를 실행하고 인스턴스 이름을 **0, 1** 및 **2**로 지정해야 할 수 있습니다. 이 애플리케이션에는 **상태 저장 세트**가 적합합니다. 상태 저장 세트는 데이터베이스 및 zookeeper 클러스터와 같은 독립 스토리지가 필요한 애플리케이션에 가장 유용합니다.

### 8.3.3. 사용 가능한 지원 구성 요소

작성하는 애플리케이션에는 데이터베이스 또는 로깅 구성 요소와 같은 지원 구성 요소가 필요할 수 있습니다. 이러한 요구를 충족시키기 위해 OpenShift Container Platform 웹 콘솔에서 사용 가능한 다음 카탈로그에서 필요한 구성 요소를 얻을 수 있습니다.

- 각 OpenShift Container Platform 4.15 클러스터에서 사용할 수 있는 OperatorHub. OperatorHub를 통해 Red Hat, 인증된 Red Hat 파트너 및 커뮤니티 멤버가 클러스터 운영자까지 Operator를 사용할 수 있습니다. 클러스터 운영자는 해당 Operator를 클러스터의 모든 네임스페이스 또는 선택된 네임스페이스에서 사용 가능하게 할 수 있으므로, 개발자가 애플리케이션을 사용하여 이를 시작하고 구성할 수 있습니다.
- 구성 요소를 설치한 후 구성 요소의 라이프사이클이 중요하지 않은 일회용 애플리케이션 유형에 유용한 템플릿. 템플릿은 최소한의 오버헤드로 Kubernetes 애플리케이션 개발을 쉽게 시작할 수 있는 방법을 제공합니다. 템플릿은 **Deployment, Service, Route** 또는 기타 오브젝트가 될 수 있는 리소스 정의 목록일 수 있습니다. 이름이나 리소스를 변경하려면 템플릿에서 이러한 값을 매개 변수로 설정할 수 있습니다.

개발 팀의 특정 요구에 맞게 지원 Operator 및 템플릿을 구성한 다음 개발자가 작업하는 네임스페이스에서 사용 가능하게 만들 수 있습니다. 많은 사용자가 다른 모든 네임스페이스에서 액세스할 수 있으므로 **openshift** 네임스페이스에 공유 템플릿을 추가합니다.

### 8.3.4. 매니페스트 적용

Kubernetes 매니페스트를 사용하면 Kubernetes 애플리케이션을 구성하는 구성 요소를 더 완벽하게 이해할 수 있습니다. 이러한 매니페스트를 YAML 파일로 작성하고 **oc apply** 명령을 실행하는 등의 작업을 통해 클러스터에 적용하여 배포합니다.

### 8.3.5. 다음 단계

이 시점에서 컨테이너 개발 프로세스를 자동화하는 방법을 고려하십시오. 이상적으로 이미지를 빌드하고 레지스트리로 푸시하는 일종의 CI 파이프라인이 있습니다. 특히, GitOps 파이프라인은 컨테이너 개발을 애플리케이션 빌드에 필요한 소프트웨어를 저장하는 데 사용하는 Git 리포지토리와 통합합니다.

이 시점의 워크플로우는 다음과 같습니다.

- 1일 차: YAML을 작성합니다. 그런 다음 **oc apply** 명령을 실행하여 해당 YAML을 클러스터에 적용하고 작동하는지 테스트합니다.
- 2일 차: YAML 컨테이너 구성 파일을 자체 Git 리포지토리에 넣습니다. 여기에서 해당 앱을 설치하거나 개선하는 데 도움이 되는 사용자는 YAML을 풀다운하고 클러스터에 적용하여 앱을 실행할 수 있습니다.
- 3일 차: 애플리케이션에 맞는 Operator 작성을 고려하십시오.

## 8.4. OPERATOR용 개발

다른 사용자가 애플리케이션을 실행할 수 있게 하려면 애플리케이션을 Operator로 패키징하고 배포하는 것이 좋습니다. 앞에서 언급했듯이 Operator는 애플리케이션을 설치하는 즉시 애플리케이션 실행 작업이 완료되지 않음을 확인하는 라이프사이클 구성 요소를 애플리케이션에 추가합니다.

Operator로 애플리케이션을 생성할 때 애플리케이션을 실행하고 유지보수하는 방법에 관한 고유 지식을 빌드할 수 있습니다. 애플리케이션 업그레이드, 백업, 스케일링 또는 지속적인 상태 추적을 위한 기능을 빌드할 수 있습니다. 애플리케이션을 올바르게 구성하면 Operator 업데이트와 같은 유지보수 작업이 Operator의 사용자에게 표시되지 않고 자동으로 발생할 수 있습니다.

유용한 Operator의 예는 특정 시간에 자동으로 데이터를 백업하도록 설정된 Operator입니다. Operator가 정해진 시간에 애플리케이션의 백업을 관리하게 하면 시스템 관리자가 이를 기억하는 데 쏟는 시간이 절약됩니다.

데이터 백업 또는 인증서 교체와 같이 일반적으로 수동으로 완료되는 모든 애플리케이션 유지보수는 Operator를 통해 자동으로 완료할 수 있습니다.

## 9장. RHCOS(RED HAT ENTERPRISE LINUX COREOS)

### 9.1. RHCOS 소개

RHCOS(Red Hat Enterprise Linux CoreOS)는 RHEL(Red Hat Enterprise Linux)의 품질 표준을 자동화된 원격 업그레이드 기능을 제공하여 차세대 단일 용도 컨테이너 운영 체제 기술을 나타냅니다.

RHCOS는 모든 OpenShift Container Platform 머신에 대해 OpenShift Container Platform 4.15의 구성 요소로만 지원됩니다. RHCOS는 OpenShift Container Platform 컨트롤 플레인 머신 또는 마스터 머신에 지원되는 유일한 운영 체제입니다. RHCOS는 모든 클러스터 머신의 기본 운영 체제이지만 RHEL을 운영 체제로 사용하는 작업자 머신이라고도 하는 컴퓨팅 머신을 생성할 수 있습니다. OpenShift Container Platform 4.15에 RHCOS를 배포하는 일반적인 방법은 다음 두 가지가 있습니다.

- 설치 프로그램이 프로비저닝하는 인프라에 클러스터를 설치하는 경우 설치 중에 RHCOS 이미지가 대상 플랫폼으로 다운로드됩니다. RHCOS 구성을 제어하는 적합한 Ignition 구성 파일도 다운로드하여 머신을 배포하는 데 사용됩니다.
- 관리하는 인프라에 클러스터를 설치하는 경우 설치 문서에 따라 RHCOS 이미지를 얻고, Ignition 구성 파일을 생성하며, Ignition 구성 파일을 사용하여 머신을 프로비저닝해야 합니다.

#### 9.1.1. 주요 RHCOS 기능

다음 목록은 RHCOS 운영 체제의 주요 기능을 설명합니다.

- **RHEL 기반:** 기본 운영 체제는 주로 RHEL 구성 요소로 구성됩니다. RHEL을 지원하는 동일한 품질, 보안 및 제어 수단도 RHCOS를 지원합니다. 예를 들어, RHCOS 소프트웨어는 RPM 패키지에 있으며 각 RHCOS 시스템은 RHEL 커널과 systemd init 시스템에서 관리하는 일련의 서비스로 시작됩니다.
- **제어된 불변성:** RHEL 구성 요소가 포함되어 있지만 RHCOS는 기본 RHEL 설치보다 더 엄격하게 관리되도록 설계되었습니다. OpenShift Container Platform 클러스터에서 원격으로 관리가 수행됩니다. RHCOS 머신을 설정할 때 몇 가지 시스템 설정만 수정할 수 있습니다. 이러한 제어된 불변성을 통해 OpenShift Container Platform이 클러스터에 최신 상태의 RHCOS 시스템을 저장할 수 있으므로, 항상 추가 머신을 생성하고 최신 RHCOS 구성에 따라 업데이트를 수행할 수 있습니다.
- **CRI-O 컨테이너 런타임:** RHCOS에는 Docker에 필요한 OCI 및 libcontainer 형식 컨테이너를 실행하는 기능이 포함되어 있지만 Docker 컨테이너 엔진 대신 CRI-O 컨테이너 엔진을 통합합니다. OpenShift Container Platform과 같은 Kubernetes 플랫폼에 필요한 기능에 중점을 두어 CRI-O는 다양한 Kubernetes 버전과의 특정 호환성을 제공할 수 있습니다. CRI-O에서는 더 큰 기능 세트를 제공하는 컨테이너 엔진으로 가능한 것보다 작은 설치 공간과 감소된 공격 면적을 제공합니다. 현재 CRI-O는 OpenShift Container Platform 클러스터 내에서만 사용할 수 있는 엔진입니다. CRI-O는 runC 또는 crun 컨테이너 런타임을 사용하여 컨테이너를 시작하고 관리할 수 있습니다. crun 활성화 방법에 대한 자세한 내용은 **ContainerRuntimeConfig** CR 생성 설명서를 참조하십시오.
- **컨테이너 도구 세트:** 컨테이너 빌드, 복사 및 관리와 같은 작업의 경우 RHCOS는 Docker CLI 도구를 호환 가능한 컨테이너 도구 세트로 대체합니다. podman CLI 도구에서는 컨테이너 및 컨테이너 이미지 실행, 시작, 중지, 나열 및 제거와 같은 많은 컨테이너 런타임 기능을 지원합니다. skopeo CLI 도구를 통해서도 이미지를 복사, 인증 및 서명할 수 있습니다. **crictl** CLI 도구를 사용하여 CRI-O 컨테이너 엔진의 컨테이너 및 포트에 대해 작업할 수 있습니다. RHCOS에서 이러한 도구를 직접 사용하는 것은 좋지 않지만 디버깅 목적으로 사용할 수 있습니다.
- **rpm-ostree upgrades:** RHCOS에서 **rpm-ostree** 시스템을 사용하여 트랜잭션 업그레이드를 제공합니다. 업데이트는 컨테이너 이미지를 통해 제공되며 OpenShift Container Platform 업데이트 프로세스의 일부입니다. 배포되면 컨테이너 이미지를 가져와서 추출하고 디스크에 쓴 다음, 부트

로더가 새 버전으로 부팅되도록 수정됩니다. 클러스터 용량에 미치는 영향을 최소화하기 위해 머신이 롤링 방식으로 업데이트되게 재부팅됩니다.

- **bootupd 펌웨어 및 부트로더 업데이트 도구.** 패키지 관리자 및 **rpm-ostree**와 같은 하이브리드 시스템은 펌웨어 또는 부트로더를 업데이트하지 않습니다. **bootupd**를 사용하는 RHCOS 사용자는 최신 아키텍처 (x86\_64, ppc64le, aarch64 등)에서 실행되는 UEFI 및 레거시 BIOS 부팅 모드 펌웨어 및 부팅 업데이트를 관리하는 교차 배포 시스템 관련 OS 업데이트 툴에 액세스할 수 있습니다.  
**bootupd**를 설치하는 방법에 대한 자세한 내용은 *bootupd를 사용하여 부트로더 업데이트* 설명서를 참조하십시오.
- **Machine Config Operator를 통해 업데이트:** OpenShift Container Platform에서 Machine Config Operator는 운영 체제 업그레이드를 처리합니다. **yum** 업그레이드와 마찬가지로 개별 패키지를 업그레이드하는 대신 **rpm-ostree**는 원자 단위로 OS 업그레이드를 제공합니다. 새 OS 배포는 업그레이드 중에 준비되며 다음에 다시 부팅할 때 적용됩니다. 업그레이드에 문제가 있으면 단일 롤백 및 재부팅을 통해 시스템을 이전 상태로 되돌립니다. OpenShift Container Platform의 RHCOS 업그레이드는 클러스터 업데이트 중에 수행됩니다.

RHCOS 시스템의 경우 **rpm-ostree** 파일 시스템의 레이아웃은 다음과 같은 특징이 있습니다.

- **/usr**은 운영 체제 바이너리 및 라이브러리가 저장되는 위치이며 읽기 전용입니다. 이 변경은 지원하지 않습니다.
- **/etc, /boot, /var**은 시스템에 쓸 수 있지만 Machine Config Operator를 통해서만 변경될 수 있습니다.
- **/var/lib/containers**는 컨테이너 이미지를 저장하기 위한 그래프 스토리지 위치입니다.

### 9.1.2. RHCOS 구성 방법 선택

RHCOS는 최소한의 사용자 구성으로 OpenShift Container Platform 클러스터에 배포하도록 설계되었습니다. 가장 기본적인 형태로 다음과 같이 구성됩니다.

- AWS와 같은 프로비저닝된 인프라로 시작하거나 인프라를 직접 프로비저닝.
- **openshift-install**을 실행할 때 **install-config.yaml** 파일에 자격 증명과 클러스터 이름과 같은 몇 가지 정보를 제공합니다.

그 이후 OpenShift Container Platform의 RHCOS 시스템은 OpenShift Container Platform 클러스터에서 완전히 관리되도록 설계되었으므로 RHCOS 머신을 직접 변경하는 것은 좋지 않습니다. 디버깅 목적으로 RHCOS 머신 클러스터에 대한 직접 액세스를 제한할 수 있지만 RHCOS 시스템을 직접 구성해서는 안 됩니다. 대신 OpenShift Container Platform 노드에서 기능을 추가하거나 변경해야 하는 경우 다음과 같은 방식으로 변경해 보십시오.

- **Kubernetes 워크로드 개체 (DaemonSet 및 Deployment 등)** 서비스 또는 기타 사용자 수준 기능을 클러스터에 추가해야 하는 경우 Kubernetes 워크로드 개체로 추가하는 것이 좋습니다. 특정 노드 구성 외부에서 이 기능을 유지하는 것이 후속 업그레이드에서 클러스터를 중단할 위험을 줄이는 가장 좋은 방법입니다.
- **2일 차 사용자 정의:** 가능하면 클러스터 노드를 사용자 정의하지 않고 클러스터를 작동시키며 클러스터가 가동된 후 필요한 노드를 변경합니다. 이와 같이 변경하면 나중에 추적하기가 쉬워지고 업데이트를 중단할 가능성이 적어집니다. 이러한 사용자 정의 방법으로 머신 구성을 생성하거나 Operator 사용자 정의 리소스를 수정할 수 있습니다.
- **1일 차 사용자 정의:** 클러스터가 처음 표시될 때 구현해야 하는 사용자 정의의 경우 최초 부팅 시 변경사항이 구현되도록 클러스터를 수정하는 방법이 있습니다. 1일 차 사용자 정의는 **openshift-**

**install** 중에 Ignition 구성 및 매니페스트 파일을 통해 수행하거나 사용자가 프로비저닝한 ISO 설치 중에 부트 옵션을 추가하여 수행할 수 있습니다.

1일 차에 수행할 수 있는 사용자 정의의 예는 다음과 같습니다.

- **커널 인수:** 클러스터가 최초 부팅될 때 노드에서 특정 커널 기능 또는 조정이 필요한 경우.
- **디스크 암호화:** 보안상 FIPS 지원과 같이 노드의 루트 파일 시스템을 암호화해야 하는 경우.
- **커널 모듈:** 네트워크 카드 또는 비디오 카드와 같은 특정 하드웨어 장치에 Linux 커널에서 기본적으로 사용 가능한 모듈이 없는 경우.
- **Chronyd:** 시간 서버의 위치와 같은 특정 시계 설정을 노드에 제공하려는 경우.

이러한 작업을 수행하려면 **MachineConfig** 오브젝트와 같은 추가 오브젝트를 포함하도록 **openshift-install** 프로세스를 보강할 수 있습니다. 머신 구성을 생성하는 절차는 클러스터가 가동된 후 Machine Config Operator에 전달될 수 있습니다.

#### 참고

- 설치 프로그램에서 생성하는 Ignition 구성 파일에 24시간 후에 만료되는 인증서가 포함되어 있습니다. 이 인증서는 그 후에 갱신됩니다. 인증서를 갱신하기 전에 클러스터가 종료되고 24시간이 지난 후에 클러스터가 다시 시작되면 클러스터는 만료된 인증서를 자동으로 복구합니다. 예외적으로 kubelet 인증서를 복구하려면 대기 중인 **node-bootstrapper** 인증서 서명 요청(CSR)을 수동으로 승인해야 합니다. 자세한 내용은 **만료된 컨트롤 플레인 인증서에서 복구** 문서를 참조하십시오.
- 24 시간 인증서는 클러스터를 설치한 후 16시간에서 22시간으로 인증서가 교체되기 때문에 생성된 후 12시간 이내에 Ignition 구성 파일을 사용하는 것이 좋습니다. 12시간 이내에 Ignition 구성 파일을 사용하면 설치 중에 인증서 업데이트가 실행되는 경우 설치 실패를 방지할 수 있습니다.

### 9.1.3. RHCOS 배포 방법 선택

OpenShift Container Platform용 RHCOS 설치 간의 차이점은 설치 프로그램이 프로비저닝한 인프라에 배포하는지 아니면 사용자가 프로비저닝한 인프라에 배포하는지에 따라 다릅니다.

- **설치 관리자 프로비저닝:** 일부 클라우드 환경은 최소한의 구성으로 OpenShift Container Platform 클러스터를 시작할 수 있는 사전 구성된 인프라를 제공합니다. 이러한 유형의 설치에서는 클러스터가 최초 부팅될 때 준비되도록 각 노드에 콘텐츠를 배치하는 Ignition 구성을 제공할 수 있습니다.
- **사용자 프로비저닝:** 자체 인프라를 프로비저닝하는 경우 RHCOS 노드에 콘텐츠를 추가하는 방법에 있어 유연성이 향상됩니다. 예를 들어, RHCOS ISO 설치 프로그램을 부팅하여 각 시스템을 설치할 때 커널 인수를 추가할 수 있습니다. 그러나 운영 체제 자체에서 구성이 필요한 대부분의 경우 Ignition 구성을 통해 해당 구성을 제공하는 것이 가장 좋습니다.

Ignition 기능은 RHCOS 시스템이 처음 설정된 경우에만 실행됩니다. 그런 다음 나중에 머신 구성을 사용하여 Ignition 구성을 제공할 수 있습니다.

### 9.1.4. Ignition 정보

Ignition은 RHCOS가 초기 구성 중에 디스크를 조작하는 데 사용하는 유틸리티입니다. 디스크 파티셔닝, 파티션 포맷, 파일 작성 및 사용자 구성을 포함한 일반적인 디스크 작업을 완료합니다. 최초 부팅 시 Ignition은 설치 미디어 또는 사용자가 지정한 위치에서 구성을 읽고 해당 구성을 머신에 적용합니다.



클러스터를 설치하든 클러스터에 머신을 추가하든 Ignition은 항상 OpenShift Container Platform 클러스터 머신의 초기 구성을 수행합니다. 실제 시스템 설정의 대부분은 각 머신 자체에서 수행됩니다. 머신마다 Ignition은 RHCOS 이미지를 가져와서 RHCOS 커널을 부팅합니다. 커널 명령줄의 옵션은 배포 유형과 Ignition 지원 초기 RAM 디스크(initramfs)의 위치를 식별합니다.

#### 9.1.4.1. Ignition 작동 방식

Ignition을 사용하여 머신을 생성하려면 Ignition 구성 파일이 필요합니다. OpenShift Container Platform 설치 프로그램은 클러스터를 배포하는 데 필요한 Ignition 구성 파일을 생성합니다. 이러한 파일은 설치 프로그램에 직접 또는 **install-config.yaml** 파일을 통해 제공한 정보를 기반으로 합니다.

Ignition이 머신을 구성하는 방법은 [cloud-init](#) 또는 Linux Anaconda [kickstart](#)와 같은 도구가 시스템을 구성하는 방법과 유사하지만 몇 가지 중요한 차이점이 있습니다.

- Ignition은 설치하려는 시스템과 별도의 초기 RAM 디스크에서 실행됩니다. 따라서 Ignition은 디스크를 다시 파티셔닝하고 파일 시스템을 설정하며 머신의 영구 파일 시스템을 변경할 수 있습니다. 반대로, cloud-init는 시스템 부팅 시 머신 init 시스템의 일부로 실행되므로 디스크 파티션과 같은 사항을 근본적으로 변경할 수는 없습니다. cloud-init를 사용하면 노드 부팅 프로세스 도중에 부팅 프로세스를 재구성하기가 어렵습니다.
- Ignition은 기존 시스템을 변경하지 않고 시스템을 초기화하기 위한 것입니다. 머신이 초기화되고 설치된 시스템에서 커널이 실행된 후 OpenShift Container Platform 클러스터의 Machine Config Operator가 향후 머신 구성을 모두 완료합니다.
- 정의된 작업 세트를 완료하는 대신 Ignition은 선언적 구성을 구현합니다. 새 머신이 시작되기 전에 모든 파티션, 파일, 서비스 및 기타 항목이 제 위치에 있는지 확인합니다. 그런 다음 새 머신이 지정된 구성을 충족시키는 데 필요한 파일을 디스크에 복사하는 등의 변경을 수행합니다.
- Ignition이 머신 구성을 마치면 커널은 계속 실행되지만 초기 RAM 디스크를 버리고 디스크의 설치된 시스템으로 전환합니다. 시스템을 재부팅하지 않아도 모든 새로운 시스템 서비스 및 기타 기능이 시작됩니다.
- Ignition은 모든 새 머신이 선언된 구성을 충족하는지 확인하므로 부분적으로 구성된 머신을 가질 수 없습니다. 머신 설정에 실패하면 초기화 프로세스가 완료되지 않고 Ignition이 새 머신을 시작하지 않습니다. 클러스터에는 부분적으로 구성된 머신이 포함되지 않습니다. Ignition을 완료할 수 없으면 머신이 클러스터에 추가되지 않습니다. 대신 새 머신을 추가해야 합니다. 그러면 구성 작업에 종속된 항목이 나중에 실패할 때까지 실패한 구성 작업의 결과를 알 수 없을 때 머신을 디버깅하기 어려운 상황이 방지됩니다.
- 머신 설정에 실패하게 만드는 Ignition 구성에 문제가 있는 경우, Ignition은 동일한 구성을 사용하여 다른 머신을 설정하려고 시도하지 않습니다. 예를 들어, 동일한 파일을 생성하려는 상위 및 하위 구성으로 구성된 Ignition 구성 때문에 실패할 수 있습니다. 이러한 경우 장애가 발생하면 문제가 해결될 때까지 다른 머신을 설정하기 위해 Ignition 구성을 다시 사용하지 못하게 됩니다.
- 여러 개의 Ignition 구성 파일이 있으면 해당 구성 세트를 통합합니다. Ignition은 선언적이므로 구성 간의 충돌 때문에 Ignition이 머신을 설정하지 못할 수 있습니다. 해당 파일의 정보 순서는 중요하지 않습니다. Ignition은 가장 적합한 방식으로 각 설정을 정렬하고 구현합니다. 예를 들어, 파일에 여러 수준의 디렉터리가 필요한 경우 다른 파일에 해당 경로를 따라 디렉터리가 필요한 경우 나중 파일이 먼저 생성됩니다. Ignition은 모든 파일, 디렉터리 및 링크를 수준별로 정렬하고 생성합니다.
- Ignition은 완전히 비어 있는 하드 디스크로 시작할 수 있으므로 cloud-init에서 수행할 수 없는 작업을 수행할 수 있습니다. PXE 부팅과 같은 기능을 사용하여 처음부터 베어 메탈에 시스템을 설정합니다. 베어 메탈의 경우 Ignition 구성이 부팅 파티션에 삽입되어 Ignition 구성이 이를 찾고 시스템을 올바르게 구성할 수 있도록 합니다.



### 9.1.4.2. Ignition 순서

OpenShift Container Platform 클러스터에 있는 RHCOS 머신의 Ignition 프로세스에는 다음 단계가 포함됩니다.

- 머신이 Ignition 구성 파일을 가져옵니다. 컨트롤 플레인 머신은 부트스트랩 머신에서 Ignition 구성 파일을 가져오고 작업자 머신은 컨트롤 플레인 머신에서 Ignition 구성 파일을 가져옵니다.
- Ignition은 머신에 디스크 파티션, 파일 시스템, 디렉토리 및 링크를 생성합니다. RAID 어레이는 지원하지만 LVM 볼륨은 지원하지 않습니다.
- Ignition은 initramfs의 **/sysroot** 디렉터리에 영구 파일 시스템의 루트를 마운트하고 해당 **/sysroot** 디렉터리에서 작동을 시작합니다.
- Ignition은 정의된 모든 파일 시스템을 구성하고 런타임 시 적절히 마운트되도록 설정합니다.
- Ignition은 **systemd** 임시 파일을 실행하여 **/var** 디렉터리에 필요한 파일을 채웁니다.
- Ignition은 Ignition 구성 파일을 실행하여 사용자, systemd 장치 파일 및 기타 구성 파일을 설정합니다.
- Ignition에서는 initramfs에 마운트된 영구 시스템의 모든 구성 요소를 마운트 해제합니다.
- Ignition은 새 머신의 init 프로세스를 시작하여 시스템 부팅 중에 실행되는 머신에서 다른 모든 서비스를 시작합니다.

이 프로세스가 끝나면 시스템이 클러스터에 참여할 준비가 되어 재부팅할 필요가 없습니다.

## 9.2. IGNITION 구성 파일 보기

부트스트랩 머신을 배포하는 데 사용된 Ignition 구성 파일을 보려면 다음 명령을 실행하십시오.

```
$ openshift-install create ignition-configs --dir $HOME/testconfig
```

몇 가지 질문에 대답하면 **bootstrap.ign**, **master.ign** 및 **worker.ign** 파일이 입력한 디렉터리에 표시됩니다.

**bootstrap.ign** 파일의 내용을 보려면 **jq** 필터를 통해 전송하십시오. 해당 파일의 스니펫은 다음과 같습니다.

```
$ cat $HOME/testconfig/bootstrap.ign | jq
{
  "ignition": {
    "version": "3.2.0"
  },
  "passwd": {
    "users": [
      {
        "name": "core",
        "sshAuthorizedKeys": [
          "ssh-rsa AAAAB3NzaC1yc...."
        ]
      }
    ]
  }
},
```

```

"storage": {
  "files": [
    {
      "overwrite": false,
      "path": "/etc/motd",
      "user": {
        "name": "root"
      },
      "append": [
        {
          "source": "data:text/plain;charset=utf-
8;base64,VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVklHdo
ZW4gdGhlIG1hc3RlciBpcyBmdWxseSB1cC4KCIRoZSBwcmltYXJ5IHNIcnZpY2VzIGFyZSByZWxlYXNl
WltYWdlLnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXI
gc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RlIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2VydmljZSAt
dSBib290a3ViZS5zZXJ2aWNlCg=="
        }
      ],
      "mode": 420
    },
  ],
  ...

```

**bootstrap.ign** 파일에 나열된 파일의 내용을 디코딩하려면 해당 파일의 내용을 나타내는 base64 인코딩 데이터 문자열을 **base64 -d** 명령으로 전송하십시오. 다음은 위의 출력에서 부트스트랩 머신에 추가된 **/etc/motd** 파일의 내용을 사용하는 예입니다.

```

$ echo
VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVklHdoZW4gdG
hlIG1hc3RlciBpcyBmdWxseSB1cC4KCIRoZSBwcmltYXJ5IHNIcnZpY2VzIGFyZSByZWxlYXNlWltYWdl
LnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXIgc3Rhd
HVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RlIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2VydmljZSAt
dSBib290a3ViZS5zZXJ2aWNlCg== | base64 --decode

```

## 출력 예

This is the bootstrap node; it will be destroyed when the master is fully up.

The primary services are release-image.service followed by bootkube.service. To watch their status, run e.g.

```
journalctl -b -f -u release-image.service -u bootkube.service
```

**master.ign** 및 **worker.ign** 파일에서 해당 명령을 반복하여 각 머신 유형에 대한 Ignition 구성 파일의 소스를 확인하십시오. **worker.ign**에 다음과 같은 행이 표시되어 부트스트랩 머신에서 Ignition 구성을 가져오는 방법을 식별해야 합니다.

```
"source": "https://api.myign.develcluster.example.com:22623/config/worker",
```

**bootstrap.ign** 파일에서 배울 수 있는 몇 가지 사항은 다음과 같습니다.

- 형식: 파일 형식은 [Ignition 구성 사양](#)에 정의되어 있습니다. MCO에서 나중에 동일한 형식의 파일을 사용하여 변경 사항을 머신 구성에 병합합니다.

- 내용: 부트스트랩 머신이 다른 머신의 Ignition 구성을 제공하므로 마스터 및 작업자 머신 Ignition 구성 정보는 부트스트랩 머신의 구성과 함께 **bootstrap.ign**에 저장됩니다.
- 크기: 파일의 길이는 1,300줄 이상이며 다양한 유형의 리소스 경로가 포함됩니다.
- 머신에 복사될 각 파일의 내용은 실제로 데이터 URL로 인코딩되므로 내용을 읽기가 다소 어려워 집니다. (내용을 더 읽기 쉽게 하려면 이전에 표시된 **jq** 및 **base64** 명령을 사용하십시오.)
- 구성: Ignition 구성 파일에 다른 섹션이 있으면 일반적으로 기존 파일을 수정하는 명령이 아니라 머신의 파일 시스템에 드롭된 파일을 포함하는 것입니다. 예를 들어, NFS에 해당 서비스를 구성하는 섹션이 두는 대신, 시스템이 시작될 때 init 프로세스에서 시작하는 NFS 구성 파일을 추가합니다.
- 사용자: ssh 키가 사용자에게 할당되어 **core**라는 이름의 사용자가 생성됩니다. 그러면 해당 사용자 이름과 인증서로 클러스터에 로그인할 수 있습니다.
- 스토리지: 스토리지 섹션에서는 각 머신에 추가된 파일을 식별합니다. 주목할 몇 가지 파일에는 **/root/.docker/config.json**(클러스터가 컨테이너 이미지 레지스트리에서 가져와야 하는 인증서 제공)과 클러스터를 구성하는 데 사용되는 **/opt/openshift/manifests**의 많은 매니페스트 파일이 포함됩니다.
- systemd: **systemd** 섹션에는 **systemd** 장치 파일을 작성하는 데 사용되는 내용이 있습니다. 이러한 파일은 부팅 시 서비스를 시작하고 실행 중인 시스템에서 해당 서비스를 관리하는 데 사용됩니다.
- 프리미티브: Ignition에서는 다른 도구를 빌드할 수 있는 하위 수준의 프리미티브도 노출합니다.

### 9.3. 설치 후 IGNITION 구성 변경

머신 구성 풀은 노드 클러스터와 해당 머신 구성을 관리합니다. 머신 구성에는 클러스터의 구성 정보가 포함됩니다. 알려진 모든 머신 구성 풀을 나열하려면 다음을 수행하십시오.

```
$ oc get machineconfigpools
```

출력 예

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-1638c1aea398413bb918e76632f20799	False	False	False
worker	worker-2feef4f8288936489a5a832ca8efe953	False	False	False

모든 머신 구성을 나열하려면 다음을 수행하십시오.

```
$ oc get machineconfig
```

출력 예

NAME	GENERATEDBY	CONTROLLER	IGNITIONVERSION	CREATED
OSIMAGEURL				
00-master	4.0.0-0.150.0.0-dirty	3.2.0		16m
00-master-ssh	4.0.0-0.150.0.0-dirty			16m
00-worker	4.0.0-0.150.0.0-dirty	3.2.0		16m
00-worker-ssh	4.0.0-0.150.0.0-dirty			16m
01-master-kubelet	4.0.0-0.150.0.0-dirty	3.2.0		16m

```
01-worker-kubelet          4.0.0-0.150.0.0-dirty 3.2.0      16m
master-1638c1aea398413bb918e76632f20799 4.0.0-0.150.0.0-dirty 3.2.0      16m
worker-2feef4f8288936489a5a832ca8efe953 4.0.0-0.150.0.0-dirty 3.2.0      16m
```

Machine Config Operator는 이러한 머신 구성을 적용할 때 Ignition과 약간 다르게 작동합니다. 머신 구성을 순서대로 읽습니다(00\*에서 99\*까지). 머신 구성 내부의 레이블을 사용하여 각 노드 유형(마스터 또는 작업자)을 식별합니다. 동일한 파일이 여러 머신 구성 파일에 표시되면 마지막 파일이 우선합니다. 예를 들어, 99\* 파일에 표시되는 파일이 00\* 파일에 표시되는 동일한 파일을 대체합니다. 입력 **MachineConfig** 오브젝트가 "렌더링된" **MachineConfig** 오브젝트로 통합되며, 이는 운영자가 대상으로 사용하고 머신 구성 풀에 표시되는 값입니다.

머신 구성에서 관리 중인 파일을 보려면 특정 **MachineConfig** 오브젝트 내에서 "Path:"를 찾으십시오. 예를 들면 다음과 같습니다.

```
$ oc describe machineconfigs 01-worker-container-runtime | grep Path:
```

#### 출력 예

```
Path:      /etc/containers/registries.conf
Path:      /etc/containers/storage.conf
Path:      /etc/crio/crio.conf
```

머신 구성 파일에 나중 파일의 이름을 지정하십시오(예: 10-worker-container-runtime). 각 파일의 내용은 URL 스타일 데이터로 되어 있습니다. 그런 다음 새 머신 구성을 클러스터에 적용하십시오.

## 10장. 승인 플러그인

승인 플러그인은 OpenShift Container Platform의 기능을 규제하는 데 사용됩니다.

### 10.1. 승인 플러그인 정보

승인 플러그인은 마스터 API에 대한 요청을 가로채어 리소스 요청을 검증합니다. 요청이 인증 및 승인되면 승인 플러그인에서 관련 정책이 준수되도록 합니다. 예를 들어 보안 정책, 리소스 제한 또는 구성 요구사항을 적용하는 데 일반적으로 사용됩니다.

승인 플러그인은 순서대로 승인 체인으로 실행됩니다. 순서의 승인 플러그인이 요청을 거부하면 전체 체인이 중단되고 오류가 반환됩니다.

OpenShift Container Platform에는 각 리소스 유형에 대해 기본 승인 플러그인 세트가 활성화되어 있습니다. 클러스터가 올바르게 작동하는 데 필요합니다. 승인 플러그인은 책임이 없는 리소스는 무시합니다.

기본 설정 외에도 사용자 정의 웹 후크 서버를 호출하는 웹 후크 승인 플러그인을 통해 승인 체인을 동적으로 확장할 수 있습니다. 웹 후크 승인 플러그인에는 변경 승인 플러그인과 검증 승인 플러그인의 두 가지 유형이 있습니다. 변경 승인 플러그인이 먼저 실행되며 리소스를 수정하고 요청을 검증할 수 있습니다. 검증 승인 플러그인은 변경 승인 플러그인에서 트리거한 수정도 검증할 수 있도록 요청을 검증하고 변경 승인 플러그인 후에 실행됩니다.

변경 승인 플러그인을 통해 웹 후크 서버를 호출하면 대상 오브젝트와 관련된 리소스에 부작용이 발생할 수 있습니다. 이러한 상황에서는 최종 결과가 예상한 대로인지 확인하는 단계를 수행해야 합니다.

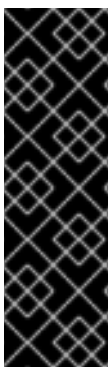


#### 주의

동적 승인은 클러스터 컨트롤 플레인 작업에 영향을 주기 때문에 주의해서 사용해야 합니다. OpenShift Container Platform 4.15에서 웹 후크 승인 플러그인을 통해 웹 후크 서버를 호출할 때 문서를 완전히 읽고 변경 부작용을 테스트했는지 확인하십시오. 요청이 전체 승인 체인을 통과하지 않는 경우 변경 이전에 리소스를 원래 상태로 복원하는 단계를 포함합니다.

### 10.2. 기본 승인 플러그인

OpenShift Container Platform 4.15에서는 기본 검증 및 승인 플러그인이 활성화됩니다. 이러한 기본 플러그인은 수신 정책, 클러스터 리소스 제한 덮어쓰기 및 할당량 정책과 같은 기본 컨트롤 플레인 기능에 기여합니다.



#### 중요

기본 프로젝트에서 워크로드를 실행하거나 기본 프로젝트에 대한 액세스를 공유하지 마세요. 기본 프로젝트는 핵심 클러스터 구성 요소를 실행하기 위해 예약되어 있습니다.

다음 기본 프로젝트는 높은 권한이 있는 것으로 간주됩니다. **default, kube-public, kube-system, openshift, openshift-infra, openshift-node** 및 **openshift.io/run-level** 레이블이 **0** 또는 **1**로 설정된 기타 시스템 생성 프로젝트입니다. Pod 보안 승인, 보안 컨텍스트 제약 조건, 클러스터 리소스 할당량 및 이미지 참조 확인과 같은 승인 플러그인에 의존하는 기능은 높은 권한 있는 프로젝트에서 작동하지 않습니다.

다음 목록에는 기본 승인 플러그인이 포함되어 있습니다.

#### 예 10.1. 검증 승인 플러그인

- **LimitRanger**
- **ServiceAccount**
- **PodNodeSelector**
- **Priority**
- **PodTolerationRestriction**
- **OwnerReferencesPermissionEnforcement**
- **PersistentVolumeClaimResize**
- **RuntimeClass**
- **CertificateApproval**
- **CertificateSigning**
- **CertificateSubjectRestriction**
- **autoscaling.openshift.io/ManagementCPUsOverride**
- **authorization.openshift.io/RestrictSubjectBindings**
- **scheduling.openshift.io/OriginPodNodeEnvironment**
- **network.openshift.io/ExternalIPRanger**
- **network.openshift.io/RestrictedEndpointsAdmission**
- **image.openshift.io/ImagePolicy**
- **security.openshift.io/SecurityContextConstraint**
- **security.openshift.io/SCCExecRestrictions**
- **route.openshift.io/IngressAdmission**
- **config.openshift.io/ValidateAPIServer**
- **config.openshift.io/ValidateAuthentication**
- **config.openshift.io/ValidateFeatureGate**
- **config.openshift.io/ValidateConsole**
- **operator.openshift.io/ValidateDNS**
- **config.openshift.io/ValidateImage**

- `config.openshift.io/ValidateOAuth`
- `config.openshift.io/ValidateProject`
- `config.openshift.io/DenyDeleteClusterConfiguration`
- `config.openshift.io/ValidateScheduler`
- `quota.openshift.io/ValidateClusterResourceQuota`
- `security.openshift.io/ValidateSecurityContextConstraints`
- `authorization.openshift.io/ValidateRoleBindingRestriction`
- `config.openshift.io/ValidateNetwork`
- `operator.openshift.io/ValidateKubeControllerManager`
- `ValidatingAdmissionWebhook`
- `resourceQuota`
- `quota.openshift.io/ClusterResourceQuota`

#### 예 10.2. 변경 승인 플러그인

- `NamespaceLifecycle`
- `LimitRanger`
- `ServiceAccount`
- `NodeRestriction`
- `TaintNodesByCondition`
- `PodNodeSelector`
- `Priority`
- `DefaultTolerationSeconds`
- `PodTolerationRestriction`
- `DefaultStorageClass`
- `StorageObjectInUseProtection`
- `RuntimeClass`
- `DefaultIngressClass`
- `autoscaling.openshift.io/ManagementCPUsOverride`
- `scheduling.openshift.io/OriginPodNodeEnvironment`

- [image.openshift.io/ImagePolicy](https://image.openshift.io/ImagePolicy)
- [security.openshift.io/SecurityContextConstraint](https://security.openshift.io/SecurityContextConstraint)
- [security.openshift.io/DefaultSecurityContextConstraints](https://security.openshift.io/DefaultSecurityContextConstraints)
- [MutatingAdmissionWebhook](#)

### 10.3. WEBHOOK 승인 플러그인

OpenShift Container Platform 기본 승인 플러그인 외에도 웹 후크 서버를 호출하는 웹 후크 승인 플러그인을 통해 동적 승인을 구현하여 승인 체인의 기능을 확장할 수 있습니다. 웹 후크 서버는 정의된 엔드포인트에서 HTTP를 통해 호출됩니다.

OpenShift Container Platform에는 다음 두 가지 유형의 웹 후크 승인 플러그인이 있습니다.

- 승인 프로세스 중에 *변경 승인 플러그인*은 선택도 레이블 삽입과 같은 작업을 수행할 수 있습니다.
- 승인 프로세스가 끝나면 *검증 승인 플러그인*을 사용하여 오브젝트가 올바르게 구성되었는지 확인할 수 있습니다(예: 선택도 레이블이 예상대로 있는지 확인). 검증이 통과되면 OpenShift Container Platform에서 구성된 대로 오브젝트를 스케줄링합니다.

API 요청이 입력되면 변경 또는 검증 승인 플러그인이 구성의 외부 웹 후크 목록을 사용하여 병렬로 호출합니다.

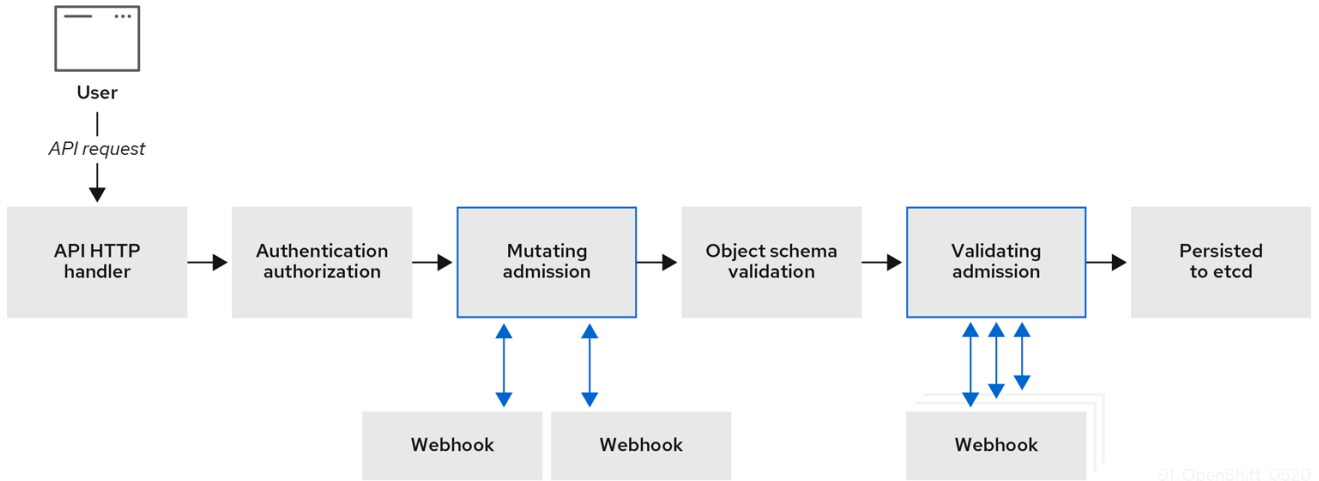
- 모든 웹 후크가 요청을 승인하면 승인 체인이 계속됩니다.
- 웹 후크 중 하나라도 요청을 거부하면 승인 요청이 거부됩니다. 그 이유는 첫 번째 거부를 기반으로 합니다.
- 둘 이상의 웹 후크가 승인 요청을 거부하면 첫 번째 거부 이유만 사용자에게 반환됩니다.
- 웹 후크를 호출할 때 오류가 발생하면 오류 정책 세트에 따라 요청이 거부되거나 웹 후크가 무시됩니다. 오류 정책이 **Ignore**로 설정되면 실패 시 요청이 무조건 수락됩니다. 정책이 **Fail**로 설정되면 실패한 요청이 거부됩니다. **Ignore**를 사용하면 모든 클라이언트에 대해 예기치 않은 동작이 발생할 수 있습니다.

웹 후크 승인 플러그인과 웹 후크 서버 간 통신은 TLS를 사용해야 합니다. CA 인증서를 생성하고 인증서를 사용하여 웹 후크 승인 서버에서 사용하는 서버 인증서에 서명하십시오. PEM으로 인코딩된 CA 인증서는 서비스 제공 인증서 보안과 같은 메커니즘을 사용하여 웹 후크 승인 플러그인에 제공됩니다.

다음 다이어그램은 여러 웹 후크 서버가 호출되는 순차적 승인 체인 프로세스를 보여줍니다.



그림 10.1. 변경 및 검증 승인 플러그인을 사용하는 API 승인 체인



91\_OpenShift\_0520

웹 후크 승인 플러그인 사용 사례의 예는 모든 Pod에 공통 레이블 세트가 있어야 하는 경우입니다. 이 예에서 변경 승인 플러그인은 레이블을 삽입할 수 있으며 검증 승인 플러그인은 레이블이 예상대로 있는지 확인할 수 있습니다. 계속하여 OpenShift Container Platform에서 필요한 레이블이 포함된 포드를 스케줄링하고 그렇지 않은 레이블은 거부합니다.

일반적인 웹 후크 승인 플러그인 사용 사례는 다음과 같습니다.

- 네임스페이스 예약.
- SR-IOV 네트워크 장치 플러그인에서 관리하는 사용자 정의 네트워크 리소스 제한.
- 테인트를 통해 노드에서 스케줄링해야 할 포드를 확인할 수 있는 허용 오차 정의.
- 포드 우선 순위 클래스 검증.



#### 참고

OpenShift Container Platform의 기본 웹 후크 시간 제한 값은 13초이며 변경할 수 없습니다.

## 10.4. 웹 후크 승인 플러그인의 유형

클러스터 관리자는 API 서버 승인 체인의 변경 승인 플러그인 또는 검증 승인 플러그인을 통해 웹 후크 서버를 호출할 수 있습니다.

### 10.4.1. 변경 승인 플러그인

변경 승인 플러그인은 승인 프로세스의 변경 단계에서 호출되므로 리소스 콘텐츠가 지속되기 전에 수정할 수 있습니다. 변경 승인 플러그인을 통해 호출할 수 있는 웹 후크의 한 예는 네임스페이스의 주석을 사용하여 라벨 선택기를 찾고 Pod 사양에 추가하는 Pod 노드 선택기 기능입니다.

#### 샘플 변경 승인 플러그인 구성

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration 1
metadata:
  name: <webhook_name> 2
  
```

```
webhooks:
- name: <webhook_name> 3
  clientConfig: 4
    service:
      namespace: default 5
      name: kubernetes 6
      path: <webhook_url> 7
    caBundle: <ca_signing_certificate> 8
  rules: 9
  - operations: 10
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - ""
    resources:
    - <resource>
  failurePolicy: <policy> 11
  sideEffects: None
```

- 1 변경 승인 플러그인 구성을 지정합니다.
- 2 **MutatingWebhookConfiguration** 오브젝트의 이름입니다. **<webhook\_name>**을 적절한 값으로 바꿉니다.
- 3 호출할 웹 후크의 이름입니다. **<webhook\_name>**을 적절한 값으로 바꿉니다.
- 4 웹 후크 서버에 연결하고 신뢰하고 데이터를 전송하는 방법에 대한 정보입니다.
- 5 프런트 엔드 서비스가 생성되는 네임스페이스입니다.
- 6 프런트 엔드 서비스의 이름입니다.
- 7 승인 요청에 사용되는 웹 후크 URL입니다. **<webhook\_url>**을 적절한 값으로 바꿉니다.
- 8 웹 후크 서버가 사용하는 서버 인증서에 서명하는 PEM 인코딩된 CA 인증서입니다. **<ca\_signing\_certificate>**를 base64 형식의 적절한 인증서로 바꿉니다.
- 9 API 서버가 이 웹 후크 승인 플러그인을 사용해야 하는 시기를 정의하는 규칙입니다.
- 10 API 서버가 이 웹 후크 승인 플러그인을 호출하도록 트리거하는 하나 이상의 작업입니다. 가능한 값은 **create, update, delete** 또는 **connect**입니다. **<operation>** 및 **<resource>**를 적절한 값으로 바꿉니다.
- 11 웹 후크 서버를 사용할 수 없는 경우 정책 진행 방법을 지정합니다. **<policy>**를 **Ignore**(실패한 경우 요청을 무조건 수락) 또는 **Fail**(실패한 요청 거부)로 바꿉니다. **Ignore**를 사용하면 모든 클라이언트에 대해 예기치 않은 동작이 발생할 수 있습니다.



### 중요

OpenShift Container Platform 4.15에서는 변경 승인 플러그인을 통해 사용자 또는 제어 루프가 생성한 오브젝트는 특히 초기 요청에 설정된 값을 덮어쓰는 경우 예기치 않은 결과를 반환할 수 있습니다. 이는 권장되지 않습니다.

### 10.4.2. 검증 승인 플러그인

승인 프로세스의 검증 단계에서 검증 승인 플러그인이 호출됩니다. 이 단계에서는 특정 API 리소스에 대한 변형을 적용하여 리소스가 다시 변경되지 않게 합니다. 포드 노드 선택기는 검증 승인 플러그인에 의해 호출되는 웹 후크의 예이며, 네임스페이스의 노드 선택기 제한에 따라 모든 **nodeSelector** 필드가 제한되도록 합니다.

#### 샘플 검증 플러그인 구성

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration ❶
metadata:
  name: <webhook_name> ❷
webhooks:
- name: <webhook_name> ❸
  clientConfig: ❹
    service:
      namespace: default ❺
      name: kubernetes ❻
      path: <webhook_url> ❼
      caBundle: <ca_signing_certificate> ❽
  rules: ❾
  - operations: ❿
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - <resource>
  failurePolicy: <policy> 11
  sideEffects: Unknown
```

- ❶ 검증 승인 플러그인 구성을 지정합니다.
- ❷ **ValidatingWebhookConfiguration** 오브젝트의 이름입니다. **<webhook\_name>**을 적절한 값으로 바꿉니다.
- ❸ 호출할 웹 후크의 이름입니다. **<webhook\_name>**을 적절한 값으로 바꿉니다.
- ❹ 웹 후크 서버에 연결하고 신뢰하고 데이터를 전송하는 방법에 대한 정보입니다.
- ❺ 프런트 엔드 서비스가 생성되는 네임스페이스입니다.
- ❻ 프런트 엔드 서비스의 이름입니다.
- ❼ 승인 요청에 사용되는 웹 후크 URL입니다. **<webhook\_url>**을 적절한 값으로 바꿉니다.
- ❽ 웹 후크 서버가 사용하는 서버 인증서에 서명하는 PEM 인코딩된 CA 인증서입니다. **<ca\_signing\_certificate>**를 base64 형식의 적절한 인증서로 바꿉니다.
- ❾ API 서버가 이 웹 후크 승인 플러그인을 사용해야 하는 시기를 정의하는 규칙입니다.
- 10

API 서버가 이 웹 후크 승인 플러그인을 호출하도록 트리거하는 하나 이상의 작업입니다. 가능한 값은 **create**, **update**, **delete** 또는 **connect**입니다. **<operation>** 및 **<resource>**를 적절한 값으로 바꿉니다.

- 11** 웹 후크 서버를 사용할 수 없는 경우 정책 진행 방법을 지정합니다. **<policy>**를 **Ignore**(실패한 경우 요청을 무조건 수락) 또는 **Fail**(실패한 요청 거부)로 바꿉니다. **Ignore**를 사용하면 모든 클라이언트에 대해 예기치 않은 동작이 발생할 수 있습니다.

## 10.5. 동적 승인 구성

이 절차에서는 동적 승인을 구성하기 위한 고급 단계를 간략하게 설명합니다. 승인 체인의 기능은 웹 후크 서버를 호출하도록 웹 후크 승인 플러그인을 구성하여 확장합니다.

웹 후크 서버는 집계 API 서버로도 구성됩니다. 그러면 다른 OpenShift Container Platform 구성 요소가 내부 인증서를 사용하여 웹 후크와 통신하고 **oc** 명령을 사용하여 테스트를 용이하게 합니다. 또한 웹 후크에 대한 역할 기반 액세스 제어(RBAC)를 가능하게 하고 다른 API 서버의 토큰 정보가 웹 후크에 공개되는 것을 방지합니다.

### 전제 조건

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정.
- OpenShift Container Platform CLI(**oc**)가 설치됨.
- 게시된 웹 후크 서버 컨테이너 이미지.

### 절차

1. 웹 후크 서버 컨테이너 이미지를 빌드하고 이미지 레지스트리를 사용하여 클러스터에서 사용 가능하게 합니다.
2. 로컬 CA 키 및 인증서를 작성한 다음 웹 후크 서버의 인증서 서명 요청(CSR)에 서명하는 데 사용합니다.
3. 웹 후크 리소스에 맞는 새 프로젝트를 생성합니다.

```
$ oc new-project my-webhook-namespace 1
```

- 1** 웹 후크 서버에서 특정 이름을 기대할 수 있습니다.

4. **rbac.yaml**이라는 파일에서 집계된 API 서비스의 RBAC 규칙을 정의합니다.

```
apiVersion: v1
kind: List
items:
- apiVersion: rbac.authorization.k8s.io/v1 1
  kind: ClusterRoleBinding
  metadata:
    name: auth-delegator-my-webhook-namespace
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:auth-delegator
```

```

subjects:
- kind: ServiceAccount
  namespace: my-webhook-namespace
  name: server

- apiVersion: rbac.authorization.k8s.io/v1 2
  kind: ClusterRole
  metadata:
    annotations:
      name: system:openshift:online:my-webhook-server
  rules:
- apiGroups:
  - online.openshift.io
  resources:
  - namespacesreservations 3
  verbs:
  - get
  - list
  - watch

- apiVersion: rbac.authorization.k8s.io/v1 4
  kind: ClusterRole
  metadata:
    name: system:openshift:online:my-webhook-requester
  rules:
- apiGroups:
  - admission.online.openshift.io
  resources:
  - namespacesreservations 5
  verbs:
  - create

- apiVersion: rbac.authorization.k8s.io/v1 6
  kind: ClusterRoleBinding
  metadata:
    name: my-webhook-server-my-webhook-namespace
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:openshift:online:my-webhook-server
  subjects:
- kind: ServiceAccount
  namespace: my-webhook-namespace
  name: server

- apiVersion: rbac.authorization.k8s.io/v1 7
  kind: RoleBinding
  metadata:
    namespace: kube-system
    name: extension-server-authentication-reader-my-webhook-namespace
  roleRef:
    kind: Role
    apiGroup: rbac.authorization.k8s.io
    name: extension-apiserver-authentication-reader
  subjects:
- kind: ServiceAccount

```

```

    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: my-cluster-role
  rules:
  - apiGroups:
    - admissionregistration.k8s.io
    resources:
    - validatingwebhookconfigurations
    - mutatingwebhookconfigurations
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - namespaces
    verbs:
    - get
    - list
    - watch

- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: my-cluster-role
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: my-cluster-role
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

```

- 1 인증 및 권한 부여를 웹 후크 서버 API에 위임합니다.
- 2 웹 후크 서버가 클러스터 리소스에 액세스할 수 있게 합니다.
- 3 리소스를 가리킵니다. 이 예는 **namespacereservations** 리소스를 가리킵니다.
- 4 집계된 API 서버를 사용하여 승인 검토를 작성할 수 있습니다.
- 5 리소스를 가리킵니다. 이 예는 **namespacereservations** 리소스를 가리킵니다.
- 6 웹 후크 서버를 사용하여 클러스터 리소스에 액세스할 수 있습니다.
- 7 인증 종료 구성을 읽기 위한 역할 바인딩.
- 8 집계된 API 서버의 기본 클러스터 역할 및 클러스터 역할 바인딩.

5. 해당 RBAC 규칙을 클러스터에 적용합니다.

```
$ oc auth reconcile -f rbac.yaml
```

6. 네임스페이스에서 웹 후크를 데몬 세트 서버로 배포하는 데 사용되는 **webhook-daemonset.yaml**이라는 YAML 파일을 생성합니다.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: my-webhook-namespace
  name: server
  labels:
    server: "true"
spec:
  selector:
    matchLabels:
      server: "true"
  template:
    metadata:
      name: server
      labels:
        server: "true"
    spec:
      serviceAccountName: server
      containers:
        - name: my-webhook-container ❶
          image: <image_registry_username>/<image_path>:<tag> ❷
          imagePullPolicy: IfNotPresent
          command:
            - <container_commands> ❸
          ports:
            - containerPort: 8443 ❹
          volumeMounts:
            - mountPath: /var/serving-cert
              name: serving-cert
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8443 ❺
              scheme: HTTPS
          volumes:
            - name: serving-cert
              secret:
                defaultMode: 420
                secretName: server-serving-cert
```

❶ 웹 후크 서버에서 특정 컨테이너 이름을 기대할 수 있습니다.

❷ 웹 후크 서버 컨테이너 이미지를 가리킵니다.  
**<image\_registry\_username>/<image\_path>:<tag>**를 적절한 값으로 바꿉니다.

❸ 웹 후크 컨테이너 실행 명령을 지정합니다. **<container\_commands>**를 적절한 값으로 바꿉니다.

- 4 포드 내에서 대상 포트를 정의합니다. 이 예에서는 8443 포트를 사용합니다.
- 5 준비 프로브가 사용하는 포트를 지정합니다. 이 예에서는 8443 포트를 사용합니다.

7. 데몬 세트를 배포합니다.

```
$ oc apply -f webhook-daemonset.yaml
```

8. **webhook-secret.yaml**이라는 YAML 파일 내에서 서비스 제공 인증서 서명자의 시크릿을 정의합니다.

```
apiVersion: v1
kind: Secret
metadata:
  namespace: my-webhook-namespace
  name: server-serving-cert
type: kubernetes.io/tls
data:
  tls.crt: <server_certificate> 1
  tls.key: <server_key> 2
```

- 1 서명된 웹 후크 서버 인증서를 참조합니다. **<server\_certificate>**를 base64 형식의 적절한 인증서로 바꿉니다.
- 2 서명된 웹 후크 서버 키를 참조합니다. **<server\_key>**를 base64 형식의 적절한 키로 바꿉니다.

9. 시크릿을 생성합니다.

```
$ oc apply -f webhook-secret.yaml
```

10. **webhook-service.yaml**이라는 YAML 파일 내에서 서비스 계정 및 서비스를 정의합니다.

```
apiVersion: v1
kind: List
items:
  - apiVersion: v1
    kind: ServiceAccount
    metadata:
      namespace: my-webhook-namespace
      name: server
  - apiVersion: v1
    kind: Service
    metadata:
      namespace: my-webhook-namespace
      name: server
    annotations:
      service.beta.openshift.io/serving-cert-secret-name: server-serving-cert
    spec:
      selector:
        server: "true"
```



```
ports:
```

```
- port: 443 ①
  targetPort: 8443 ②
```

- ① 서비스가 청취하는 포트를 정의합니다. 이 예에서는 443 포트를 사용합니다.
- ② 서비스가 연결을 전달하는 포트 내의 대상 포트를 정의합니다. 이 예에서는 8443 포트를 사용합니다.

11. 클러스터 내에 웹 후크 서버를 노출합니다.

```
$ oc apply -f webhook-service.yaml
```

12. **webhook-crd.yaml**이라는 파일에서 웹 후크 서버의 사용자 정의 리소스 정의를 정의합니다.

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: namespacesreservations.online.openshift.io ①
spec:
  group: online.openshift.io ②
  version: v1alpha1 ③
  scope: Cluster ④
  names:
    plural: namespacesreservations ⑤
    singular: namespacesreservation ⑥
    kind: NamespaceReservation ⑦
```

- ① **CustomResourceDefinition spec** 값을 반영하고 **<plural>.<group>** 형식입니다. 이 예에서는 **namespacesreservations** 리소스를 사용합니다.
- ② REST API 그룹 이름.
- ③ REST API 버전 이름.
- ④ 허용되는 값은 **Namespaced** 또는 **Cluster**입니다.
- ⑤ URL에 포함될 복수 이름입니다.
- ⑥ **oc** 출력에 표시되는 별칭입니다.
- ⑦ 리소스 매니페스트에 대한 참조입니다.

13. 사용자 정의 리소스 정의를 적용합니다.

```
$ oc apply -f webhook-crd.yaml
```

14. **webhook-api-service.yaml**이라는 파일 내에서 웹 후크 서버를 집계 API 서버로 구성합니다.

```
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
```

```

name: v1beta1.admission.online.openshift.io
spec:
  caBundle: <ca_signing_certificate> ❶
  group: admission.online.openshift.io
  groupPriorityMinimum: 1000
  versionPriority: 15
  service:
    name: server
    namespace: my-webhook-namespace
  version: v1beta1

```

- ❶ 웹 후크 서버가 사용하는 서버 인증서에 서명하는 PEM 인코딩된 CA 인증서입니다. <ca\_signing\_certificate>를 base64 형식의 적절한 인증서로 바꿉니다.

15. 집계된 API 서비스를 배포합니다.

```
$ oc apply -f webhook-api-service.yaml
```

16. **webhook-config.yaml** 이라는 파일 내에 웹 후크 승인 플러그인 구성을 정의합니다. 이 예에서는 검증 승인 플러그인을 사용합니다.

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: namespacesreservations.admission.online.openshift.io ❶
webhooks:
- name: namespacesreservations.admission.online.openshift.io ❷
  clientConfig:
    service: ❸
    namespace: default
    name: kubernetes
    path: /apis/admission.online.openshift.io/v1beta1/namespacesreservations ❹
    caBundle: <ca_signing_certificate> ❺
  rules:
  - operations:
    - CREATE
    apiGroups:
    - project.openshift.io
    apiVersions:
    - "*"
    resources:
    - projectrequests
  - operations:
    - CREATE
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - namespaces
  failurePolicy: Fail

```

- 1 **ValidatingWebhookConfiguration** 개체의 이름입니다. 이 예에서는 **namespacereservations** 리소스를 사용합니다.
- 2 호출할 웹 후크의 이름입니다. 이 예에서는 **namespacereservations** 리소스를 사용합니다.
- 3 집계된 API를 통해 웹 후크 서버에 액세스할 수 있습니다.
- 4 승인 요청에 사용되는 웹 후크 URL입니다. 이 예에서는 **namespacereservation** 리소스를 사용합니다.
- 5 웹 후크 서버가 사용하는 서버 인증서에 서명하는 PEM 인코딩된 CA 인증서입니다. **<ca\_signing\_certificate>**를 base64 형식의 적절한 인증서로 바꿉니다.

17. 웹 후크를 배포합니다.

```
$ oc apply -f webhook-config.yaml
```

18. 웹 후크가 예상대로 작동하는지 확인합니다. 예를 들어 특정 네임스페이스를 예약하기 위해 동적 승인을 구성한 경우 해당 네임스페이스를 생성하는 요청이 거부되고 예약되지 않은 네임스페이스를 생성하는 요청이 성공했는지 확인합니다.

## 10.6. 추가 리소스

- [SR-IOV Network Operator](#) 구성
- [노드 테인트를 사용하여 Pod 배치 제어](#)
- [Pod 우선순위 이름](#)