OpenShift Container Platform 4.7

아키텍처

OpenShift Container Platform의 아키텍처 개요

Last Updated: 2023-04-08

OpenShift Container Platform 4.7 아키텍처

OpenShift Container Platform의 아키텍처 개요

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

법적 공지

초록

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

차례

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 아키텍처를 참조하십시오.

1.1. 설치 및 업데이트 정보

클러스터 관리자는 OpenShift Container Platform 설치 프로그램을 사용하여 다음 방법 중 하나를 사용하여 클러스터를 설치하고 배포할 수 있습니다.

- 설치 관리자 프로비저닝 인프라
- 사용자 프로비저닝 인프라

1.2. 컨트롤 플레인 정보

컨트롤 플레인 은 작업자 노드와 클러스터의 Pod를 관리합니다. MCP(머신 구성 풀)를 사용하여 노드를 구성할 수 있습니다. MCP는 처리하는 리소스를 기반으로 하는 컨트롤 플레인 구성 요소 또는 사용자 워크로드와 같은 머신 그룹입니다. OpenShift Container Platform은 호스트에 다양한 역할을 할당합니다. 이러한 역할은 클러스터에서 시스템의 기능을 정의합니다. 클러스터에는 표준 컨트롤 플레인 및 작업자 역할 유형의 정의가 포함되어 있습니다.

Operator를 사용하여 컨트롤 플레인에서 서비스를 패키지, 배포 및 관리할 수 있습니다. Operator는 다음 서비스를 제공하기 때문에 OpenShift Container Platform의 중요한 구성 요소입니다.

- 상태 점검 수행
- 애플리케이션을 감시하는 방법 제공
- 무선(Over-The-Air) 업데이트 관리
- 애플리케이션이 지정된 상태에 유지되도록 설정

1.3. 개발자를 위한 컨테이너화된 애플리케이션 정보

개발자는 다양한 툴, 방법 및 형식을 사용하여 고유한 요구 사항에 따라 컨테이너화된 애플리케이션을 개발할 수 있습니다. 예를 들면 다음과 같습니다.

- 다양한 build-tool, base-image 및 registry 옵션을 사용하여 간단한 컨테이너 애플리케이션을 빌드합니다.
- OperatorHub 및 템플릿과 같은 지원 구성 요소를 사용하여 애플리케이션을 개발합니다.
- 애플리케이션을 Operator로 패키징하고 배포합니다.

Kubernetes 매니페스트를 생성하여 Git 리포지토리에 저장할 수도 있습니다. Kubernetes는 포드라는 기본 장치에서 작동합니다. 포드는 클러스터에서 실행 중인 프로세스의 단일 인스턴스입니다. Pod에는 하나 이상의 컨테이너가 포함될 수 있습니다. 일련의 포드 및 해당 액세스 정책을 그룹화하여 서비스를 생성할 수 있습니다. 서비스에서는 포드가 생성 및 삭제될 때 사용할 다른 애플리케이션에 영구 내부 IP 주소 및 호스트 이름을 제공합니다. Kubernetes는 애플리케이션 유형을 기반으로 워크로드를 정의합니다.

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

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

- 차세대 단일 용도의 컨테이너 운영 체제 기술에 대해 자세히 알아보기.
- RHCOS(Red Hat Enterprise Linux CoreOS) 구성 방법 선택
- RHCOS(Red Hat Enterprise Linux CoreOS) 배포 방법 선택.
  - 설치 관리자 프로비저닝 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 분할, 포맷, 파일 작성, 사용자 구성 등의 일반적인 디스크 작업을 수행합니다. 처음 부팅하는 동안 Ignition은 설치 미디어 또는 사용자가 지정한 위치에서 구성을 읽고 해당 구성을 머신에 적용합니다.

Ignition 작동 방식, OpenShift Container Platform 클러스터의 RHCOS(Red Hat Enterprise Linux CoreOS) 프로세스, Ignition 구성 파일을 보고, 설치 후 Ignition 구성을 변경할 수 있습니다.

1.5. 승인 플러그인 정보

승인 플러그인을 사용하여 OpenShift Container Platform이 작동하는 방식을 규제할 수 있습니다. 리소스 요청이 인증되고 승인되면 승인 플러그인은 마스터 API에 대한 리소스 요청을 가로채서 리소스 요청의 유효성을 검사하고 확장 정책을 준수하도록 합니다. 승인 플러그인은 보안 정책, 리소스 제한 또는 구성 요구 사항을 적용하는 데 사용됩니다.

2장. OPENSHIFT CONTAINER PLATFORM 아키텍처

2.1. OPENSHIFT 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, 뱅킹 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.

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 노드(마스터 노드라고도 함)에서 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 4.7이 제공하는 중요한 이점입니다. 다음 섹션에서는 OpenShift Container Platform의 고유한 기능과 이점에 관해 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 도구와 함께 작업하여 빠른 설치, Operator 기반 관리 및 단순화된 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 4.7에서는 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 4.7을 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 플레인은 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 4.7 코드 베이스의 기본 단위이자 애플리케이션 및 애플리케이션에서 사용할 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 Operator에 OpenShift Container Platform 클러스터를 제공하는 Quay.io 컨테이너 레지스트리입니다. Quay.io는 수백만 개의 이미지와 태그를 저장하는 Red Hat Quay의 공개 레지스트리 버전입니다.

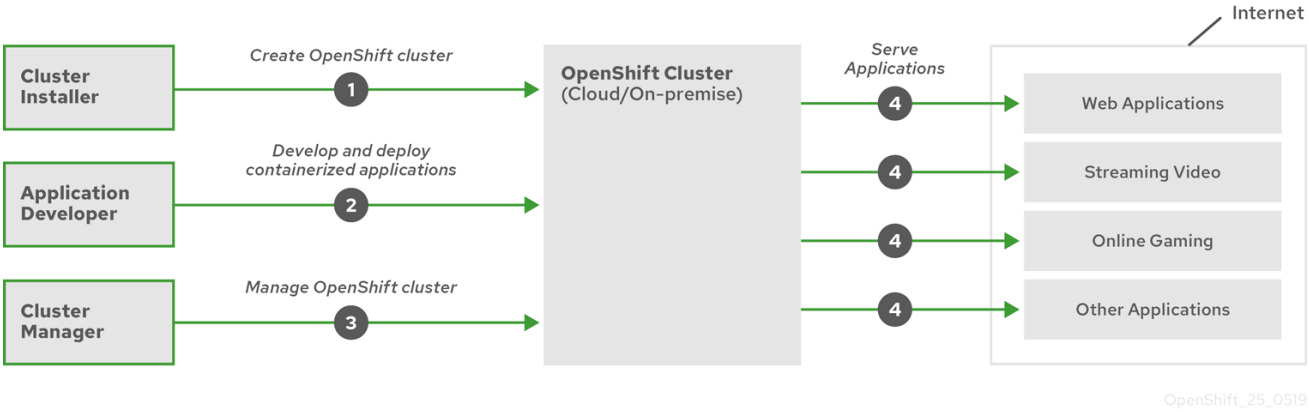
OpenShift Container Platform의 Kubernetes의 기타 향상된 기능에는 소프트웨어 정의 네트워킹(SDN), 인증, 로그 집계, 모니터링 및 라우팅 개선이 포함됩니다. OpenShift Container Platform에서는 포괄적인 웹 콘솔과 사용자 정의 OpenShift CLI(**oc**) 인터페이스도 제공합니다.

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



2.1.4. OpenShift Container Platform 용 인터넷 액세스

OpenShift Container Platform 4.7에서 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.io에 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. 설치를 수행하는 프로세스에서 필요한 내용을 다운로드한 다음, 이를 사용하여 클러스터를 설치하고 설치 프로그램을 생성하는 데 필요한 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. OPENSHIFT CONTAINER PLATFORM 설치 프로그램 개요

OpenShift Container Platform 설치 프로그램에서는 유연성을 제공합니다. 설치 프로그램을 사용하면 설치 프로그램이 프로비저닝하고 클러스터가 유지보수하는 인프라에 클러스터를 배포하거나 준비하고 유지보수하는 인프라에 클러스터를 배포할 수 있습니다.

이 두 가지 기본 유형의 OpenShift Container Platform 클러스터는 종종 설치 관리자 프로비저닝 인프라 클러스터 및 사용자 프로비저닝 인프라 클러스터라고 합니다.

두 유형의 클러스터에는 모두 다음과 같은 특징이 있습니다.

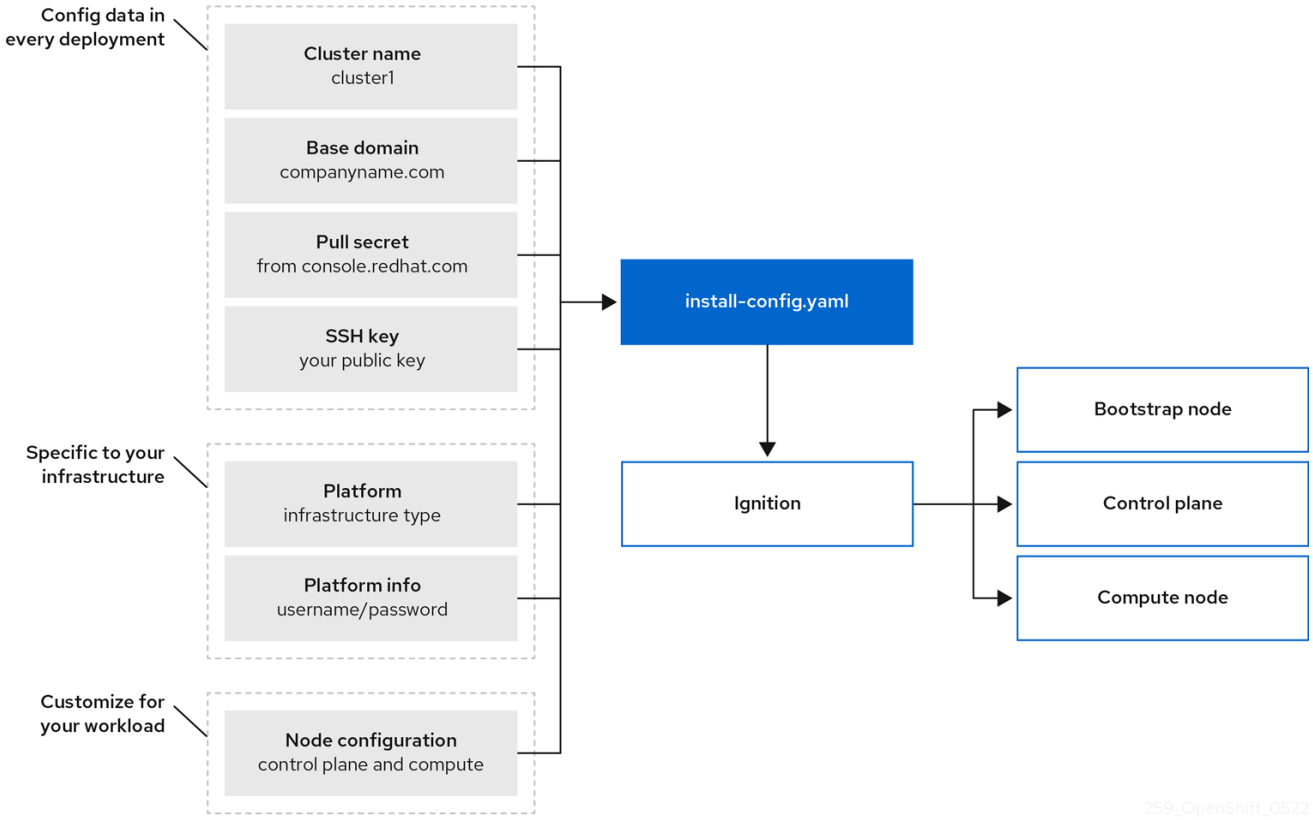
- 기본적으로 단일 장애 지점이 없는 고가용성 인프라를 사용할 수 있습니다.
- 관리자는 적용되는 업데이트 및 해당 시기에 관한 제어 권한을 유지합니다.

동일한 설치 프로그램을 사용하여 두 유형의 클러스터를 모두 배포합니다. 설치 프로그램에서 생성된 주요 자산은 부트스트랩, 마스터 및 작업자 머신의 Ignition 구성 파일입니다. 이 세 가지 구성과 올바르게 구성된 인프라를 사용하면 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 고유한 종속 항목에만 관련되므로 여러 대상이 병렬로 수행되도록 설치 프로그램이 작동할 수 있습니다. 궁극적인 목표는 실행 중인 클러스터입니다. 설치 프로그램은 명령을 실행하지 않고 종속 항목을 충족함으로써 명령을 다시 작성하기 위해 명령을 실행하는 대신 기존 구성 요소를 인식하고 사용할 수 있습니다.

다음 다이어그램에서는 설치 대상 및 종속 항목의 서브 세트를 보여줍니다.

그림 3.1. OpenShift Container Platform 설치 대상 및 종속 항목



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

OpenShift Container Platform 4.7 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 Operator가 클러스터 전체에서 롤아웃한 컨테이너 이미지에 임베드된 Atomic OSTree 리포지토리로 제공됩니다. 실제 운영 체제 변경은 rpm-ostree를 사용하여 각 머신에서 원자 작업으로 수행됩니다. 이러한 기술을 사용하면 OpenShift Container Platform에서 전체 플랫폼을 최신 상태로 유지하는 내부 업그레이드를 통해 클러스터의 다른 애플리케이션을 관리하는 것처럼 운영 체제를 관리할 수 있습니다. 이러한 내부 업데이트는 운영 팀의 부담을 줄일 수 있습니다.

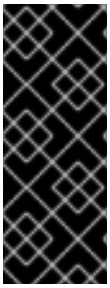
모든 클러스터 머신의 운영 체제로 RHCOS를 사용하는 경우 클러스터가 운영 체제를 포함한 구성 요소 및 머신의 모든 측면을 관리합니다. 그러면 설치 프로그램 및 Machine Config Operator만 머신을 변경할 수 있습니다. 설치 프로그램에서는 Ignition 구성 파일을 사용하여 각 머신의 정확한 상태를 설정하고 Machine Config Operator는 설치 후 새 인증서 또는 키 적용과 같은 머신에 대한 추가 변경을 완료합니다.

3.1.1. OpenShift 클러스터에서 지원되는 플랫폼

OpenShift Container Platform 4.7에서는 다음 플랫폼에서 설치 관리자 프로비저닝 인프라를 사용하는 클러스터를 설치할 수 있습니다.

- AWS(Amazon Web Services)
- GCP(Google Cloud Platform)
- Microsoft Azure
- RHOSP(Red Hat OpenStack Platform) 버전 13 및 16
    - 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 매트릭스를 참조하십시오.
- RHV(Red Hat Virtualization)
- VMware vSphere
- AWS의 VMware Cloud (VMC)
- 베어 메탈

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
- 클러스터가 설치된 플랫폼과 다른 플랫폼의 영구 스토리지 프레임워크 사용과 같은 클라우드 공급자 구성 요소 혼합

OpenShift Container Platform 4.7에서는 다음 플랫폼에서 사용자 프로비저닝 인프라를 사용하는 클러스터를 설치할 수 있습니다.

- AWS
- Azure
- GCP
- RHOSP
- RHV
- VMware vSphere
- AWS의 VMware Cloud
- 베어 메탈
- IBM Z 또는 LinuxONE
- IBM Power Systems

플랫폼의 지원 사례에 따라 사용자 프로비저닝 인프라에 을 설치하면 전체 인터넷 액세스가 가능한 머신을 실행하거나, 클러스터를 프록시 뒤에 배치하거나, *제한된 네트워크 설치*를 수행할 수 있습니다. 제한된 네트워크 설치에서는 클러스터를 설치하는 데 필요한 이미지를 다운로드하여 미러 레지스트리에 배치한 다음 해당 데이터를 사용하여 클러스터를 설치할 수 있습니다. vSphere 또는 베어 메탈 인프라에서 제한된 네트워크 설치로 플랫폼 컨테이너의 이미지를 가져오려면 인터넷에 액세스해야 하지만, 클러스터 컴퓨터는 인터넷에 직접 액세스할 필요가 없습니다.

다른 플랫폼의 통합 테스트에 대한 자세한 내용은 OpenShift Container Platform 4.x 통합 테스트 페이지를 참조하십시오.

3.1.2. 설치 프로세스

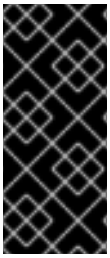
OpenShift Container Platform 클러스터를 설치할 때 OpenShift Cluster Manager 사이트의 해당 인프라 공급자 페이지에서 설치 프로그램을 다운로드합니다. 이 사이트에서는 다음을 관리합니다.

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

OpenShift Container Platform 4.7에서 설치 프로그램은 자산 세트에서 일련의 파일 변환을 수행하는 Go 바이너리 파일입니다. 설치 프로그램과 상호 작용하는 방법은 설치 유형에 따라 다릅니다.

- 설치 관리자 프로비저닝 인프라가 있는 클러스터의 경우 인프라 부트스트랩 및 프로비저닝을 직접 수행하는 대신 설치 프로그램에 위임합니다. 설치 프로그램은 클러스터를 지원하는 데 필요한 모든 네트워킹, 머신 및 운영 체제를 생성합니다.
- 클러스터의 인프라를 프로비저닝하고 관리하는 경우 부트스트랩 머신, 네트워킹, 부하 분산, 스토리지 및 개별 클러스터 머신을 포함한 모든 클러스터 인프라 및 리소스를 제공해야 합니다.

설치하는 동안 **install-config.yaml**이라는 설치 구성 파일, Kubernetes 매니페스트 및 머신 유형에 맞는 Ignition 구성 파일의 세 가지 파일 세트를 사용합니다.

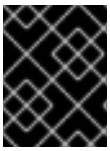


중요

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 제거되므로 다시 사용하려는 모든 구성 파일을 백업하십시오.



중요

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

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 사용하는 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.

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 다음을 포함하여 클러스터 자원을 직접 관리하고 유지보수해야 합니다.

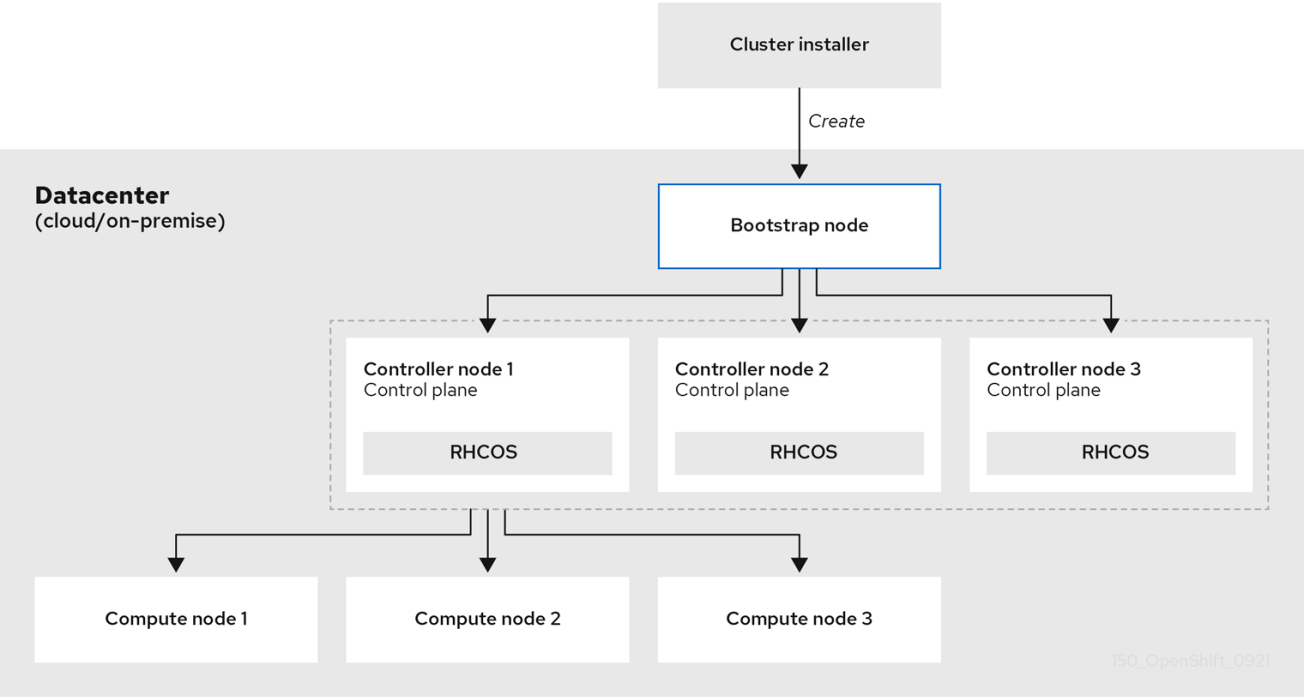
- 클러스터를 구성하는 컨트롤 플레인 및 컴퓨팅 머신의 기본 인프라
- 로드 밸런서
- DNS 레코드 및 필수 서브넷을 포함한 클러스터 네트워킹
- 클러스터 인프라 및 애플리케이션용 스토리지

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

설치 프로세스 세부사항

클러스터의 각 머신에는 프로비저닝 시 클러스터에 대한 정보가 필요하므로 OpenShift Container Platform은 초기 구성 중에 임시 *부트스트랩* 머신을 사용하여 필요한 정보를 영구 컨트롤 플레인에 제공합니다. 클러스터 생성 방법을 설명하는 Ignition 구성 파일을 사용하여 부팅됩니다. 부트스트랩 머신은 컨트롤 플레인을 구성하는 컨트롤 플레인 머신 (마스터 머신이라고도 함)을 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



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 *Recovering from expired control plane certificates* 문서를 참조하십시오.
- 클러스터를 설치한 후 24시간에서 22시간까지의 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.

클러스터 부트스트랩에는 다음 단계가 포함됩니다.

1. 부트스트랩 머신이 부팅되고 컨트롤 플레인 머신을 부팅하는 데 필요한 원격 리소스 호스팅이 시작됩니다. (인프라를 프로비저닝할 경우 수동 조작 필요)

2. 부트스트랩 머신은 단일 노드 etcd 클러스터와 임시 Kubernetes 컨트롤 플레인을 시작합니다.
3. 컨트롤 플레인 머신은 부트스트랩 머신에서 원격 리소스를 가져오고 부팅을 완료합니다. (인프라를 프로비저닝할 경우 수동 조작 필요)
4. 임시 컨트롤 플레인은 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 부트스트랩 머신을 종료합니다. (인프라를 프로비저닝할 경우 수동 조작 필요)
9. 컨트롤 플레인이 컴퓨팅 노드를 설정합니다.
10. 컨트롤 플레인은 일련의 Operator 형태로 추가 서비스를 설치합니다.

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

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. OPENSHIFT 업데이트 서비스 정보

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

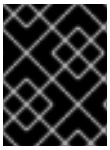
클러스터의 CVO (Cluster Version Operator)는 OpenShift Update Service를 확인하여 현재 구성 요소 버전 및 그래프의 정보를 기반으로 유효한 업데이트 및 업데이트 경로를 확인합니다. 업데이트를 요청하면 CVO는 해당 업데이트에 릴리스 이미지를 사용하여 클러스터를 업데이트합니다. 릴리스 아티팩트는 Quay에서 컨테이너 이미지로 호스팅됩니다.

OpenShift Update Service가 호환 가능한 업데이트만 제공할 수 있도록 자동화를 지원하는 버전 확인 파이프 라인이 제공됩니다. 각 릴리스 아티팩트는 지원되는 클라우드 플랫폼 및 시스템 아키텍처 및 기타 구성 요소 패키지와의 호환성 여부를 확인합니다. 파이프 라인에서 적용 가능한 버전이 있음을 확인한 후 OpenShift Update Service는 해당 버전 업데이트를 사용할 수 있음을 알려줍니다.

중요

OpenShift Update Service는 현재 클러스터에 권장되는 모든 업데이트를 표시합니다. OpenShift Update Service에서 업그레이드 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 버전으로의 업그레이드만 지원됩니다. 클러스터를 이전 버전으로 되돌리거나 롤백을 수행하는 것은 지원되지 않습니다. 업데이트에 실패하면 Red Hat 지원에 문의하십시오.

업데이트 프로세스 중에 MCO (Machine Config Operator)는 새 구성을 클러스터 머신에 적용합니다. MCO는 머신 설정 풀의 **maxUnavailable** 필드에 지정된 노드 수를 제한하고 이를 사용할 수없는 것으로 표시합니다. 기본적으로 이 값은 **1**로 설정됩니다. MCO는 새 설정을 적용하여 컴퓨터를 다시 시작합니다.

RHEL (Red Hat Enterprise Linux) 머신을 작업자로 사용하는 경우 먼저 시스템에서 OpenShift API를 업데이트해야하기 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 재정의를 설정하면 전체 클러스터가 지원되지 않는 상태가 됩니다. 지원을 계속하려면 재정의를 제거한 후 보고된 문제를 재현해야 합니다.

3.4. 다음 단계

- 클러스터 설치 방법 선택 및 사용자를 위한 준비

4장. OPENSHIFT CONTAINER PLATFORM 컨트롤 플레인

4.1. OPENSHIFT CONTAINER PLATFORM 컨트롤 플레인 이해

컨트롤 플레인 머신(마스터 머신이라고도 함)으로 구성된 컨트롤 플레인은 OpenShift Container Platform 클러스터를 관리합니다. 컨트롤 플레인 머신에서는 작업자 머신이라고도 하는 컴퓨팅 머신의 워크로드를 관리합니다. 클러스터 자체에서 Cluster Version Operator, Machine Config Operator 및 개별 Operator 세트의 작업을 통해 머신과 관련된 모든 업그레이드를 관리합니다.

4.1.1. 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.

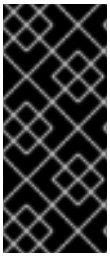
기본적으로 클러스터가 설치될 때 생성되는 **master** 및 **worker**라는 두 개의 MCP가 있습니다. 각 기본 MCP에는 MCP(Machine Config Operator)에 의해 적용되는 정의된 구성이 있으며 MCP를 관리하고 MCP 업그레이드를 가능하게 합니다. 추가 MCP 또는 사용자 지정 풀을 생성하여 기본 노드 유형 이외의 사용자 지정 사용 사례가 있는 노드를 관리할 수 있습니다.

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.



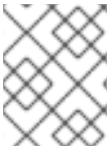
중요

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 테인트를 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.

4.1.2. OpenShift Container Platform의 머신 역할

OpenShift Container Platform은 호스트에 다른 역할을 할당합니다. 이 역할은 클러스터 내에서 머신의 기능을 정의합니다. 클러스터에는 표준 마스터 및 작업자 역할 유형의 정의가 포함됩니다.



참고

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

4.1.2.1. 컨트롤 플레인 및 노드 호스트 호환성

OpenShift Container Platform 버전은 컨트롤 플레인 호스트와 노드 호스트 간의 일치해야 합니다. 예를 들어 4.9 클러스터에서는 모든 컨트롤 플레인 호스트가 4.9이고 모든 노드가 4.9이어야 합니다.

클러스터를 업그레이드하는 동안 일시적인 불일치가 허용됩니다. 예를 들어 OpenShift Container Platform 4.8에서 4.9로 업그레이드하는 경우 일부 노드는 다른 노드보다 4.9로 업그레이드됩니다. 컨트롤 플레인 호스트와 노드 호스트를 많이 사용하면 이전 컴퓨팅 머신이 버그 및 누락된 기능에 노출될 수 있습니다. 사용자는 최대한 빨리 불일치 컨트롤 플레인 호스트와 노드 호스트를 확인해야 합니다.

kubelet 서비스는 **kube-apiserver** 보다 최신 상태가 아니어야 하며 OpenShift Container Platform 버전이 홀수인지에 따라 이전 버전 2개까지 이전할 수 있습니다. 아래 표는 적절한 버전 호환성을 보여줍니다.

OpenShift Container Platform 버전	지원되는 **kubelet** 불일치
홀수 OpenShift Container Platform 마이너 버전 [1]	이전 버전 최대 1개
OpenShift Container Platform 마이너 버전도 [2]	이전 버전 최대 2개

1. 예를 들어 OpenShift Container Platform 4.5, 4.7, 4.9입니다.
2. 예를 들면 OpenShift Container Platform 4.6, 4.8, 4.10입니다.

4.1.2.2. 클러스터 작업자

Kubernetes 클러스터에서 작업자 노드는 Kubernetes 사용자가 요청한 실제 워크로드가 실행되고 관리되는 위치입니다. 작업자 노드는 용량을 알리고 마스터 서비스의 일부인 스케줄러는 컨테이너와 포드를 시작할 노드를 결정합니다. 컨테이너 엔진인 CRI-O, 컨테이너 워크로드 실행 및 중지 요청을 수락 및 이행하는 서비스인 Kubelet 및 작업자 간의 포드 통신을 관리하는 서비스 프록시를 포함하여 각 작업자 노드에서 중요한 서비스가 실행됩니다.

OpenShift Container Platform에서 머신 세트가 작업자 머신을 제어합니다. 작업자 역할 드라이브가 있는 머신은 자동 스케일링하는 특정 머신 풀이 관리하는 워크로드를 컴퓨팅합니다. OpenShift Container Platform에는 여러 머신 유형을 지원할 수 있는 용량이 있으므로 작업자 머신은 *컴퓨팅* 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

머신 세트는 **machine-api** 네임스페이스에서 머신 리소스의 그룹입니다. 머신 세트는 특정 클라우드 공급자에서 새 머신을 시작하기 위해 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.

4.1.2.3. 클러스터 마스터

Kubernetes 클러스터에서 컨트롤 플레인 노드(마스터 노드라고도 함)는 Kubernetes 클러스터를 제어하는 데 필요한 서비스를 실행합니다. OpenShift Container Platform에서 컨트롤 플레인 시스템은 컨트롤 플레인입니다. 여기에는 OpenShift Container Platform 클러스터 관리를 위한 Kubernetes 서비스 외에도 많은 것이 포함되어 있습니다. 컨트롤 플레인 역할의 모든 머신은 컨트롤 플레인 머신이므로 *마스터* 와 *컨트롤 플레인* 이라는 용어는 서로 바꿔 사용할 수 있습니다. 컨트롤 플레인 시스템은 머신 세트로 그룹화되는 대신 일련의 독립 실행형 머신 API 리소스로 정의됩니다. 모든 컨트롤 플레인 시스템을 삭제하고 클러스터를 손상시키지 않도록 컨트롤 플레인 시스템에 추가 컨트롤이 적용됩니다.

참고

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

마스터의 Kubernetes 카테고리에 속하는 서비스에는 Kubernetes API 서버, etcd, Kubernetes 컨트롤러 관리자 및 Kuberbetes 스케줄러가 포함됩니다.

표 4.1. 컨트롤 플레인에서 실행되는 Kubernetes 서비스

구성 요소	설명
Kubernetes API 서버	Kubernetes API 서버는 포드, 서비스 및 복제 컨트롤러의 데이터를 검증하고 구성합니다. 클러스터의 공유 상태에 대한 초점도 제공합니다.
etcd	etcd는 지속적 마스터 상태를 저장하고 다른 구성 요소는 etcd가 변경사항을 감시하여 지정된 상태로 변경될 수 있게 합니다.
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 4.2. 컨트롤 플레인에서 실행되는 OpenShift 서비스

구성 요소	설명

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 시스템의 이러한 서비스 중 일부는 systemd 서비스로 실행되는 반면 다른 서비스는 정적 포드로 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 4.7에서는 Docker Container Engine 대신 CRI-O를 사용합니다.
- Kubelet(kubelet): 마스터 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**

4.1.3. OpenShift 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. 애플리케이션 모니터링, 상태 점검 수행, OTA(Over-the-air) 업데이트 관리 및 애플리케이션이 지정된 상태로 유지되도록 하는 수단을 제공합니다.

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 서로 다른 두 시스템에서 관리합니다.

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

4.1.4. 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 표시되며, 클러스터 관리자는 **관리 → 클러스터 설정** 페이지에서 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

4.1.5. 애드온 Operator

OLM(Operator Lifecycle Manager) 및 OperatorHub는 Kubernetes 네이티브 애플리케이션을 Operator로 관리하는 데 도움이 되는 OpenShift Container Platform의 기본 구성 요소입니다. 함께 사용하면 클러스터에서 사용할 수 있는 선택적 애드온 Operator를 검색, 설치 및 관리할 수 있는 시스템을 제공합니다.

OpenShift Container Platform 웹 콘솔에서 OperatorHub를 사용하여 클러스터 관리자와 권한 있는 사용자는 Operator 카탈로그에서 설치할 Operator를 선택할 수 있습니다. OperatorHub에서 Operator를 설치한 후 전역적으로 또는 특정 네임스페이스에서 사용자 애플리케이션에서 실행할 수 있습니다.

Red Hat Operator, 인증된 Operator 및 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. 그러면 Operator를 번들하고 사용자 정의 카탈로그 소스에 추가할 수 있으며 이는 클러스터에 추가되어 사용자가 사용할 수 있습니다.

참고

OLM은 OpenShift Container Platform 아키텍처를 구성하는 클러스터 Operator를 관리하지 않습니다.

추가 리소스

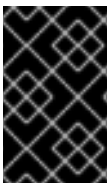
- OpenShift Container Platform에서 애드온 Operator 실행 방법에 대한 자세한 내용은 OLM(Operator Lifecycle Manager) 및 OperatorHub 의 Operator 가이드를 참조하십시오.
- *Operator SDK에 대한 자세한 내용은 Operator 개발을 참조하십시오.*

4.1.5.1. OpenShift 업데이트 서비스 정보

OSUS(OpenShift Update Service)는 Red Hat Enterprise Linux CoreOS(RHCOS)를 비롯한 OpenShift Container Platform에 대한 무선 업데이트를 제공합니다. 구성 요소 Operator의 정점과 이를 연결하는 에지를 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 업데이트에 릴리스 이미지를 사용하여 클러스터를 업데이트합니다. 릴리스 아티팩트는 Quay에서 컨테이너 이미지로 호스팅됩니다.

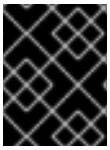
OpenShift Update Service가 호환 가능한 업데이트만 제공할 수 있도록 자동화를 지원하는 버전 확인 파이프 라인이 제공됩니다. 각 릴리스 아티팩트는 지원되는 클라우드 플랫폼 및 시스템 아키텍처 및 기타 구성 요소 패키지와의 호환성 여부를 확인합니다. 파이프 라인에서 적용 가능한 버전이 있음을 확인한 후 OpenShift Update Service는 해당 버전 업데이트를 사용할 수 있음을 알려줍니다.



중요

OpenShift Update Service는 현재 클러스터에 권장되는 모든 업데이트를 표시합니다. OpenShift Update Service에서 업그레이드 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 버전으로의 업그레이드만 지원됩니다. 클러스터를 이전 버전으로 되돌리거나 롤백을 수행하는 것은 지원되지 않습니다. 업데이트에 실패하면 Red Hat 지원에 문의하십시오.

*업데이트 프로세스 중에 MCO (Machine Config Operator)는 새 구성을 클러스터 머신에 적용합니다. MCO는 머신 설정 풀의 **maxUnavailable** 필드에 지정된 노드 수를 제한하고 이를 사용할 수없는 것으로 표시합니다. 기본적으로 이 값은 **1**로 설정됩니다. MCO는 새 설정을 적용하여 컴퓨터를 다시 시작합니다.*

RHEL (Red Hat Enterprise Linux) 머신을 작업자로 사용하는 경우 먼저 시스템에서 OpenShift API를 업데이트해야하기 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.

4.1.5.2. Machine Config Operator 이해

OpenShift Container Platform 4.7은 운영 체제와 클러스터 관리를 모두 통합합니다. 클러스터는 클러스터 노드에서 RHCOS(Red Hat Enterprise Linux CoreOS)에 대한 업데이트를 포함하여 자체 업데이트를 관리하므로 OpenShift Container Platform은 노드 업그레이드 오케스트레이션을 단순화하는 독단적인 라이프사이클 관리 환경을 제공합니다.

OpenShift Container Platform은 3개의 데몬 세트와 컨트롤러를 사용하여 노드 관리를 단순화합니다. 이러한 데몬 세트는 표준 Kubernetes 스타일 구성을 사용하여 호스트에 대한 운영 체제 업데이트 및 구성 변경을 오케스트레이션합니다. 다음이 포함됩니다.

- *컨트롤 플레인에서 머신 업그레이드를 조정하는 **machine-config-controller**. 모든 클러스터 노드를 모니터링하고 구성 업데이트를 오케스트레이션합니다.*
- *클러스터의 각 노드에서 실행되며 머신 구성에 정의된 구성으로 MachineConfigController의 지시에 따라 머신을 업데이트하는 **machine-config-daemon** 데몬 세트. 노드가 변경사항을 감지하면 포드를 비우고 업데이트를 적용한 다음 재부팅합니다. 이러한 변경사항은 지정된 머신 구성 및 제어 kubelet 구성을 적용하는 Ignition 구성 파일의 형태로 제공됩니다. 업데이트 자체는 컨테이너로 제공됩니다. 이 프로세스는 OpenShift Container Platform 및 RHCOS 업데이트를 성공적으로 관리하는 데 핵심입니다.*
- *클러스터에 참여할 때 컨트롤 플레인 노드에 Ignition 구성 파일을 제공하는 **machine-config-server** 데몬 세트.*

*머신 구성은 Ignition 구성의 서브 세트입니다. **machine-config-daemon**은 OSTree 업데이트를 수행해야 하는지 아니면 일련의 systemd kubelet 파일 변경, 구성 변경 또는 운영 체제나 OpenShift Container Platform 구성의 기타 변경 사항을 적용해야 하는지 확인하기 위해 머신 구성을 읽습니다.*

*노드 관리 작업을 수행할 때 **KubeletConfig** 사용자 정의 리소스(CR)를 생성하거나 수정합니다.*

중요

머신 구성을 변경하면 MCO(Machine Config Operator)가 변경 사항을 적용하기 위해 해당 노드를 자동으로 재부팅합니다.

*머신 구성 변경사항이 적용된 후 노드가 자동 재부팅되지 않게 하려면 해당 머신 구성 풀에서 **spec.paused** 필드를 **true**로 설정하여 자동 부팅 프로세스를 일시중지해야 합니다. 일시 정지되면 **spec.paused** 필드를 **false**로 설정하고 노드가 새 구성으로 재부팅될 때까지 머신 구성 변경 사항이 적용되지 않습니다.*

다음 수정 사항에서는 노드 재부팅이 트리거되지 않습니다.

- *MCO가 다음 변경 사항을 감지하면 노드를 드레이닝하거나 재부팅하지 않고 업데이트를 적용합니다.*
  - *머신 구성의 **spec.config.passwd.users.sshAuthorizedKeys** 매개변수에서 SSH 키 변경*
  - ***openshift-config** 네임 스페이스에서 글로벌 풀 시크릿 또는 풀 시크릿 관련 변경 사항*
  - *Kubernetes API Server Operator의 **/etc/kubernetes/kubelet-ca.crt** 인증 기관(CA) 자동 교체*
- *MCO가 **ImageContentSourcePolicy** 오브젝트 추가 또는 편집과 같은 **/etc/containers/registries.conf** 파일의 변경 사항을 감지하면 해당 노드를 비우고 변경 사항을 적용한 다음 노드를 분리합니다.*

추가 정보

Machine Config Operator가 머신 구성을 변경한 후 컨트롤 플레인 머신이 재부팅되지 않게 하는 방법에 대한 자세한 내용은 Disabling Machine Config Operator from automatically rebooting 를 참조하십시오.

5 장. OPENSHIFT 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 Platform 에서 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.

5.1. 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
- Operator 가 다른 사용자와 애플리케이션을 공유하게 만들기

5.2. 간단한 컨테이너 빌드

애플리케이션에 대한 아이디어가 있으며 컨테이너화하고 싶은 경우를 살펴보겠습니다.

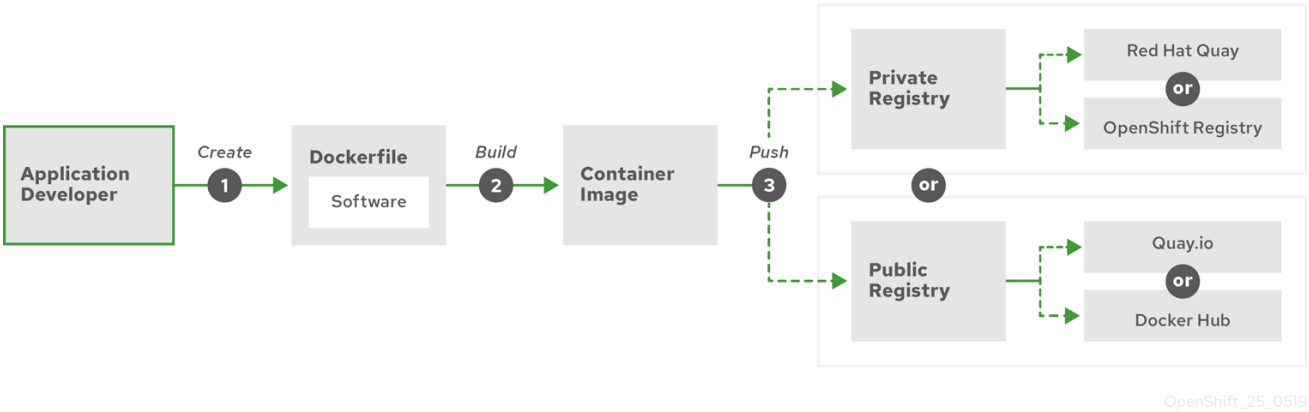
먼저 buildah 또는 docker 와 같은 컨테이너를 빌드하는 도구와 컨테이너에 들어가는 내용을 설명하는 파일(일반적으로 Dockerfile) 이 필요합니다.

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 Dockerfile 을 제외하고 대부분의 Linux 운영 체제에 기본적으로 설치됩니다.

다음 다이어그램은 이미지를 작성하고 푸시하는 프로세스를 보여줍니다.

그림 5.1. 컨테이너화된 간단한 애플리케이션을 생성하여 레지스트리로 푸시



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

1. *컨테이너 빌드 툴을 설치합니다. RHEL에는 컨테이너를 빌드하고 관리하는 데 사용하는 podman, buildah 및 skopeo가 포함된 툴 세트가 포함되어 있습니다.*

2. *기본 이미지와 소프트웨어를 결합하는 Dockerfile을 생성합니다. 컨테이너 빌드에 대한 정보는 **Dockerfile** 이라는 파일에 포함됩니다. 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 build를 실행합니다. **buildah build-using-dockerfile** 또는 **docker build** 명령을 실행하여 선택한 기본 이미지를 로컬 시스템으로 가져오고 로컬에 저장된 컨테이너 이미지를 생성합니다. buildah를 사용하여 Dockerfile 없이 컨테이너 이미지를 빌드할 수도 있습니다.*

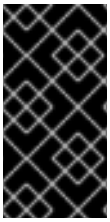
4. *태그를 지정하고 레지스트리로 푸시합니다. 컨테이너를 저장하고 공유할 레지스트리의 위치를 식별하는 새 컨테이너 이미지에 태그를 추가합니다. 그런 다음 **podman push** 또는 **docker push** 명령을 실행하여 레지스트리에 이미지를 푸시합니다.*

5. *이미지를 가져와 실행합니다. podman 또는 docker와 같은 컨테이너 클라이언트 도구가 있는 시스템에서 새 이미지를 식별하는 명령을 실행합니다. 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 푸시하며 실행하는 프로세스에 관한 자세한 내용은 Buildah로 사용자 정의 이미지 빌드를 참조하십시오.

5.2.1. 컨테이너 빌드 도구 옵션

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



중요

컨테이너 런타임으로서 Docker Container Engine 지원은 Kubernetes 1.20에서 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 그러나 Docker에서 생성된 이미지는 CRI-O를 포함한 모든 런타임과 함께 클러스터에서 계속 작동합니다. 자세한 내용은 Kubernetes 블로그 발표를 참조하십시오.

OpenShift Container Platform에서 컨테이너를 궁극적으로 실행할 때는 CRI-O 컨테이너 엔진을 사용합니다. CRI-O는 OpenShift Container Platform 클러스터의 모든 작업자 및 컨트롤 플레인 시스템(마스터 머신이라고도 함)에서 실행되지만 CRI-O는 아직 OpenShift Container Platform 외부의 독립 실행형 런타임으로 지원되지 않습니다.

5.2.2. 기본 이미지 옵션

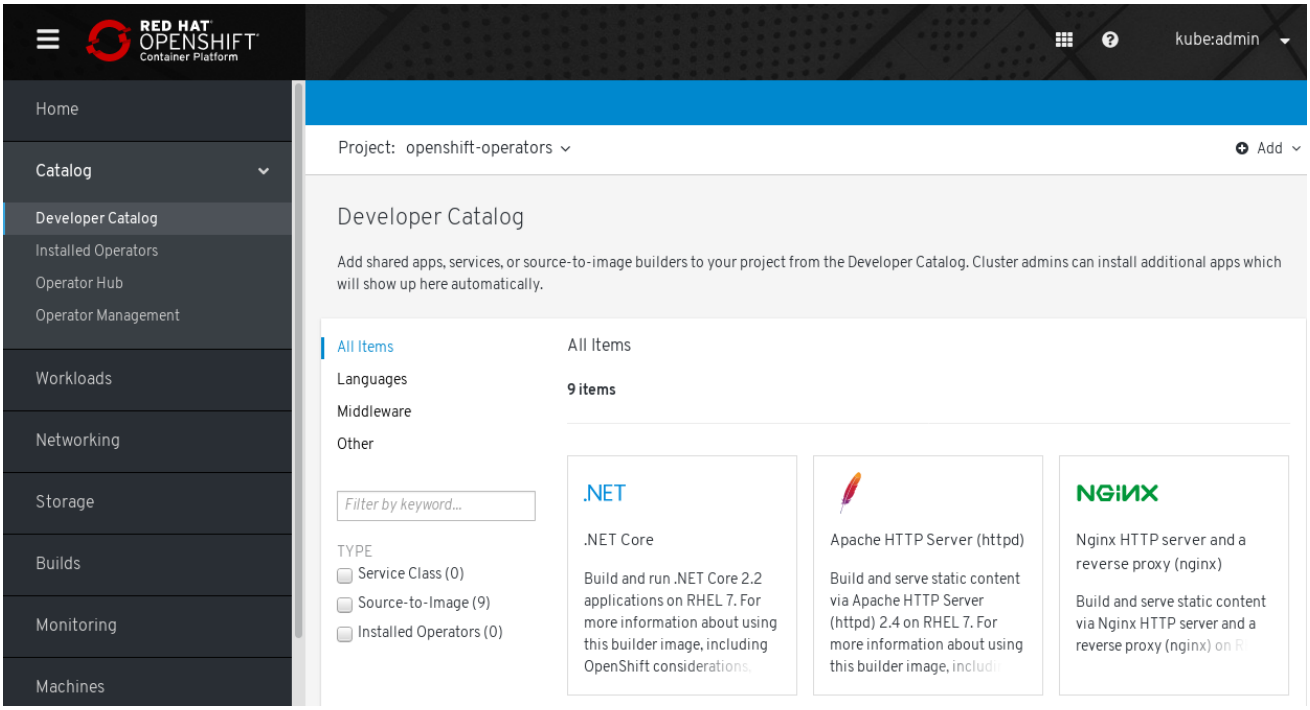
애플리케이션을 빌드하기 위해 선택한 기본 이미지에는 애플리케이션의 Linux 시스템과 유사한 소프트웨어 세트가 포함되어 있습니다. 고유 이미지를 빌드하면 소프트웨어가 해당 파일 시스템에 배치되고 해당 파일 시스템이 운영 체제를 보는 것처럼 인식합니다. 이 기본 이미지를 선택하면 향후 컨테이너의 안전성, 효율성 및 업그레이드 가능성에 큰 영향을 미칩니다.

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.

*다음 그림과 같이 **카탈로그 → 개발자 카탈로그**를 선택하여 OpenShift Container Platform 웹 UI에서 S2I 이미지를 직접 사용할 수 있습니다.*

그림 5.2. 특정 런타임이 필요한 앱에 맞는 S2I 기본 이미지 선택



5.2.3. 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 그릴 수 있습니다. Red Hat Registry는 인증되지 않고 더 이상 사용되지 않는 **registry.access.redhat.com**과 인증이 필요한 **registry.redhat.io**의 두 위치로 표시됩니다. Container images section of the Red Hat Ecosystem Catalog에서 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.io가 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 Quay라는 개인용 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.

5.3. OPENSHIFT 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.

5.3.1. Kubernetes 포드 및 서비스 정보

컨테이너 이미지는 docker가 있는 기본 단위이지만 Kubernetes가 작동하는 기본 단위는 pod라고 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 수도 있습니다.

개별 포드는 Kubernetes에서 확장 가능한 단위를 나타내지만 서비스는 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 정책을 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 볼륨을 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.

5.3.2. 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 배포를 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 세트를 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 세트로 실행할 수 있습니다. 노드 레이블을 기반으로 노드 서브 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 요구사항이 있을 수 있습니다. 예를 들어, 정확히 3개의 애플리케이션 인스턴스를 실행하고 인스턴스 이름

을 **0**, **1** 및 **2**로 지정해야 할 수 있습니다. 이 애플리케이션에는 상태 저장 세트가 적합합니다. 상태 저장 세트는 데이터베이스 및 zookeeper 클러스터와 같은 독립 스토리지가 필요한 애플리케이션에 가장 유용합니다.

5.3.3. 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 4.7 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 매개변수로 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.

5.3.4. 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.

5.3.5. 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.

- 1일차: YAML을 작성합니다. 그런 다음 **oc apply** 명령을 실행하여 해당 YAML을 클러스터에 적용하고 작동하는지 테스트합니다.

- 2일차: YAML 컨테이너 구성 파일을 자체 Git 리포지토리에 배치합니다. 여기에서 해당 앱을 설치하거나 개선하는 데 도움이 되는 사용자는 YAML을 풀다운하고 클러스터에 적용하여 앱을 실행할 수 있습니다.

- 3일차: 애플리케이션에 사용할 Operator를 작성하는 것이 좋습니다.

5.4. 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.

6장. RHCOS(RED HAT ENTERPRISE LINUX COREOS)

6.1. RHCOS 소개

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

RHCOS는 모든 OpenShift Container Platform 머신에 대해 OpenShift Container Platform 4.7의 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 4.7에 RHCOS를 배포하는 일반적인 방법에는 두 가지가 있습니다.

- 클러스터가 프로비저닝한 인프라에 클러스터를 설치하면 설치 중에 RHCOS 이미지가 대상 플랫폼으로 다운로드되고 RHCOS 구성을 제어하는 적합한 Ignition 구성 파일이 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.

6.1.1. 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.

- ***컨테이너 도구 세트***: 컨테이너 빌드, 복사 및 관리와 같은 작업의 경우 RHCOS는 Docker CLI 툴을 호환되는 컨테이너 툴 세트로 대체합니다. podman CLI 도구에서는 컨테이너 및 컨테이너 이미지 실행, 시작, 중지, 나열 및 제거와 같은 많은 컨테이너 런타임 기능을 지원합니다. skopeo CLI 도구를 통해서는 이미지를 복사, 인증 및 서명할 수 있습니다. ***crictl*** CLI 도구를 사용하여 CRI-O 컨테이너 엔진의 컨테이너 및 포드에 대해 작업할 수 있습니다. RHCOS에서 이러한 도구를 직접 사용하는 것은 좋지 않지만 디버깅 목적으로 사용할 수 있습니다.

- ***rpm-ostree upgrades***: RHCOS는 ***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:*** *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*** *설치 방법에 대한 자세한 내용은 bootupd 를 사용하여 부트로더 업데이트 설명서를 참조하십시오.*

- ***Machine Config Operator 를 통해*** *업데이트되었습니다. OpenShift Container Platform 에서 Machine Config Operator 는 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 Platform 의 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 Operator 를 통해서만 변경될 수 있습니다.*

- ***/var/lib/containers*** *는 컨테이너 이미지를 저장하기 위한 그래프 스토리지 위치입니다.*

6.1.2. RHCOS 구성 방법 선택

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

- *AWS 와 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 Platform 의 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 오브젝트(예: DaemonSet 및 Deployment)*** *: 서비스 또는 기타 사용자 수준 기능을 클러스터에 추가해야 하는 경우 Kubernetes 워크로드 오브젝트로 추가하는 것이 좋습니다. 특정 노드 구성 외부에서 이 기능을 유지하는 것이 후속 업그레이드에서 클러스터를 중단할 위험을 줄이는 가장 좋은 방법입니다.*

- ***Day-2 사용자 정의****: 가능한 경우 클러스터 노드를 사용자 정의하지 않고 클러스터를 가져오고 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.*

- ***Day-1 사용자 정의****: 클러스터가 처음 시작될 때 구현해야 하는 사용자 정의의 경우 최초 부팅 시 변경 사항이 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 처음 부팅될 때 노드에서 특정 커널 기능 또는 튜닝이 필요한 경우
- ***디스크 암호화***: 보안에 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 서버 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.
- 클러스터를 설치한 후 24시간에서 22시간까지의 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.

6.1.3. 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 환경에서는 최소한의 구성으로 OpenShift Container Platform 클러스터를 가동할 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 컨텐츠를 추가하는 방법에 대한 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.

6.1.4. 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.

6.1.4.1. 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 kickstart와 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 노드의 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, 디렉토리 및 링크를 수준별로 정렬하고 생성합니다.

- Ignition은 완전히 비어 있는 하드 디스크로 시작할 수 있기 때문에 cloud-init가 할 수 없는 작업을 수행할 수 있습니다. 즉, 처음부터 베어 메탈에서 시스템을 설정합니다(PXE 부트와 같은 기능 사용). 베어 메탈의 경우 Ignition 구성이 부트 파티션에 삽입되고 Ignition이 이 구성을 찾아 시스템을 올바르게 구성할 수 있습니다.

6.1.4.2. Ignition 순서

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

- 머신이 Ignition 구성 파일을 가져옵니다. 컨트롤 플레인 머신(마스터 머신이라고도 함)은 부트스트랩 머신에서 Ignition 구성 파일을 가져오고 작업자 머신은 마스터에서 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.
- Ignition에서 새 머신의 init 프로세스를 시작하면, 시스템 부팅 중에 실행되는 머신에서 다른 모든 서비스가 시작됩니다.

그런 다음 머신은 클러스터에 참여할 준비가 되었으며 재부팅할 필요가 없습니다.

6.2. 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,VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdo
ZW4gdGhlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2VzIGFyZSByZWxlYXNlIWltYWdlLnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXIgc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2VydmljZSAtdSBib290a3ViZS5zZXJ2aWNlCg=="
          }
        ],
        "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
VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdoZW4gdGhlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2VzIGFyZSByZWxlYXNlIWltYWdlLnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXIgc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2VydmljZSAtdSBib290a3ViZS5zZXJ2aWNlCg== | 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 사양에 정의되어 있습니다. MCO에서 나중에 동일한 형식의 파일을 사용하여 변경 사항을 머신 구성에 병합합니다.*
- *내용: 부트스트랩 머신은 다른 머신의 Ignition 구성을 제공하므로 마스터 및 작업자 머신 Ignition 구성 정보는* ***부트스트랩 머신 구성과 함께 bootstrap.ign*** *에 저장됩니다.*
- *크기: 파일은 1300줄 이상이며 다양한 유형의 리소스 경로가 있습니다.*
- *머신에 복사될 각 파일의 내용은 실제로 데이터 URL로 인코딩되므로 내용을 읽기가 다소 어려워집니다. (내용을 더 읽기 쉽게 하려면 이전에 표시된* ***jq*** *및* ***base64*** *명령을 사용하십시오.)*

- *설정: Ignition 구성 파일의 다른 섹션은 일반적으로 기존 파일을 수정하는 명령이 아니라 머신의 파일 시스템에 드롭된 파일을 포함합니다. 예를 들어, NFS에 해당 서비스를 구성하는 섹션이 두는 대신, 시스템이 시작될 때 init 프로세스에서 시작하는 NFS 구성 파일을 추가합니다.*

- *사용자: SSH 키가 해당 사용자에게 할당되어 **core** 라는 사용자가 생성됩니다. 그러면 해당 사용자 이름과 인증서로 클러스터에 로그인할 수 있습니다.*

- *storage: 스토리지 섹션에서는 각 시스템에 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 만드는 데 사용되는 콘텐츠가 있습니다. 이러한 파일은 부팅 시 서비스를 시작하고 실행 중인 시스템에서 해당 서비스를 관리하는 데 사용됩니다.*

- *프리미티브: 또한 Ignition은 다른 툴을 빌드할 수 있는 하위 수준의 프리미티브를 노출합니다.*

6.3. 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                                       GENERATEDBYCONTROLLER   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 들*

어, 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.

7장. 승인 플러그인

7.1. 승인 플러그인 정보

승인 플러그인은 OpenShift Container Platform 4.7이 작동하는 식을 규제하는 데 사용됩니다. 승인 플러그인은 요청이 인증 및 권한 부여된 후 마스터 API에 대한 요청을 인터셉트하여 리소스 요청의 유효성을 검증하고 정책이 준수되도록 합니다. 예를 들어 보안 정책, 리소스 제한 또는 구성 요구사항을 적용하는 데 일반적으로 사용됩니다.

승인 플러그인은 순서에 따라 승인 체인으로 실행됩니다. 순서에 있는 승인 플러그인이 요청을 거부하면 전체 체인이 중단되고 오류가 반환됩니다.

OpenShift Container Platform에는 각 리소스 유형에 맞게 사용되는 기본 승인 플러그인 세트가 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 플러그인이 트리거하는 수정도 검증할 수 있도록 요청을 검증하고 변경 승인 플러그인 이후에 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 4.7에서 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.

7.2. 기본 승인 플러그인

OpenShift Container Platform 4.7에서는 기본 검증 및 승인 플러그인이 활성화됩니다. 이러한 기본 플러그인은 수신 정책, 클러스터 리소스 제한 재정의 및 할당량 정책과 같은 기본 컨트롤 플레인 기능에 기여합니다. 다음 목록에는 기본 승인 플러그인이 포함되어 있습니다.

예 7.1. 검증 승인 플러그인

- **LimitRanger**
- **ServiceAccount**
- **PodNodeSelector**
- **우선 순위**
- **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 7.2. 변경 승인 플러그인

- *NamespaceLifecycle*
- *LimitRanger*
- *ServiceAccount*
- *NodeRestriction*
- *TaintNodesByCondition*
- *PodNodeSelector*
- *우선 순위*
- *DefaultTolerationSeconds*
- *PodTolerationRestriction*
- *PersistentVolumeLabel*
- *DefaultStorageClass*
- *StorageObjectInUseProtection*
- *RuntimeClass*
- *DefaultIngressClass*
- *autoscaling.openshift.io/ManagementCPUsOverride*
- *scheduling.openshift.io/OriginPodNodeEnvironment*
- *image.openshift.io/ImagePolicy*
- *security.openshift.io/SecurityContextConstraint*
- *security.openshift.io/DefaultSecurityContextConstraints*
- *MutatingAdmissionWebhook*

7.3. 웹 후크 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 두 가지 유형의 웹 후크 승인 플러그인이 있습니다.

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

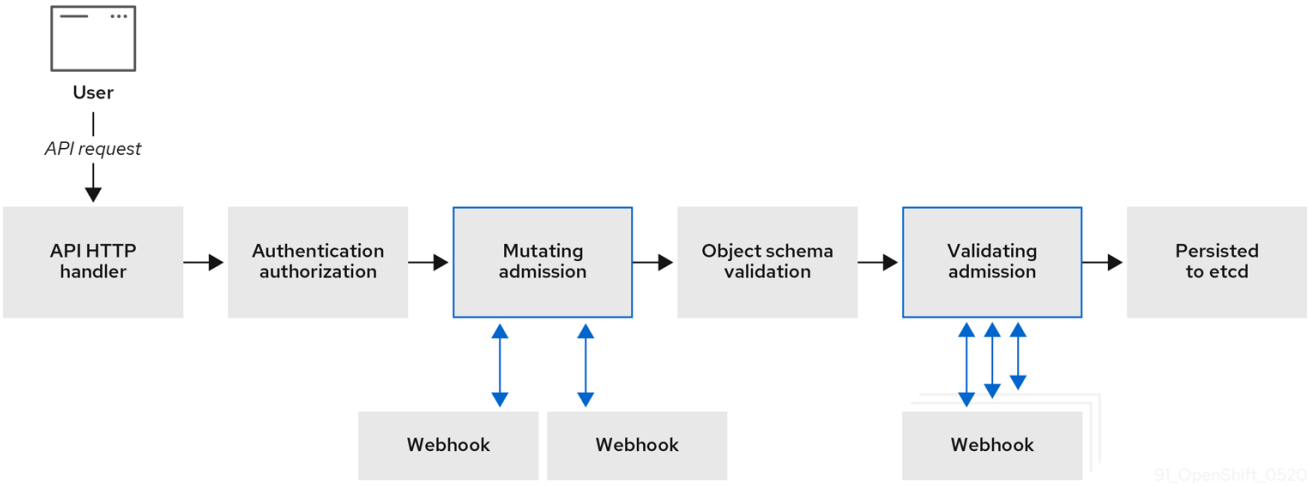
API 요청이 입력되면 변경 또는 검증 승인 플러그인에서 구성에 있는 외부 웹 후크 목록을 사용하여 병렬로 호출합니다.

- *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 간의 통신에서는 TLS를 사용해야 합니다. CA 인증서를 생성하고 인증서를 사용하여 웹 후크 승인 서버에서 사용하는 서버 인증서에 서명하십시오. PEM으로 인코딩된 CA 인증서는 서비스 제공 인증서 시크릿과 같은 메커니즘을 사용하여 웹 후크 승인 플러그인에 제공됩니다.

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

그림 7.1. 변경 및 검증 승인 플러그인을 사용하는 API 승인 체인



웹 후크 승인 플러그인 사용 사례의 예로는 모든 포드에 공통 레이블 세트가 있어야 하는 경우가 있습니다. 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.

7.4. 웹 후크 승인 플러그인의 유형

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

7.4.1. 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 레이블 선택기를 찾고 이를 포드 사양에 추가하는 포드 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 프론트 엔드 서비스의 이름입니다.

7 승인 요청에 사용되는 웹 후크 URL 입니다. **<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 4.7에서 변경 승인 플러그인을 통해 사용자 또는 제어 루프가 생성한 오브젝트는 특히 초기 요청에 설정된 값을 덮어쓰는 경우 예기치 않은 결과를 반환할 수 있습니다. 이 경우는 권장되지 않습니다.

7.4.2. 검증 승인 플러그인

승인 프로세스의 검증 단계 중에 검증 승인 플러그인이 호출됩니다. 이 단계에서는 특정 API 리소스에 대한 변형을 적용하여 리소스가 다시 변경되지 않게 합니다. 포드 노드 선택기는 검증 승인 플러그인이 호출하는 웹 후크의 예이며, 네임스페이스의 노드 선택기 제한사항을 통해 모든 **nodeSelector** 필드가 제한되게 합니다.

샘플 검증 승인 플러그인 구성

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration (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: Unknown
```

(1) 검증 승인 플러그인 구성을 지정합니다.

(2) **ValidatingWebhookConfiguration** 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) 프론트 엔드 서비스의 이름입니다.

(7) 승인 요청에 사용되는 웹 후크 URL 입니다. **<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.

7.5. 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:
    - namespacereservations (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:
```

```
    - namespacereservations 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 8
  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 1
        image: <image_registry_username>/<image_path>:<tag> 2
```

```
    imagePullPolicy: IfNotPresent
    command:
    - <container_commands> 3
    ports:
    - containerPort: 8443 4
    volumeMounts:
    - mountPath: /var/serving-cert
      name: serving-cert
    readinessProbe:
      httpGet:
        path: /healthz
        port: 8443 5
        scheme: HTTPS
  volumes:
  - name: serving-cert
    secret:
      defaultMode: 420
      secretName: server-serving-cert
```

1. 웹 후크 서버에서 특정 컨테이너 이름을 기대할 수 있습니다.
2. 웹 후크 서버 컨테이너 이미지를 가리킵니다. **<image_registry_username>/<image_path>:<tag>**를 적절한 값으로 바꿉니다.
3. 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 포드 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: namespacereservations.online.openshift.io (1)
spec:
  group: online.openshift.io (2)
  version: v1alpha1 (3)
  scope: Cluster (4)
  names:
    plural: namespacereservations (5)
    singular: namespacereservation (6)
    kind: NamespaceReservation (7)
```

-

1. ***CustomResourceDefinition spec** 값을 반영하고 **<plural>.<group>** 형식입니다. 이 예에서는 **namespacereservations** 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> (1)
  group: admission.online.openshift.io
  groupPriorityMinimum: 1000
  versionPriority: 15
  service:
    name: server
    namespace: my-webhook-namespace
  version: v1beta1
```

1. *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: namespacereservations.admission.online.openshift.io (1)
webhooks:
- name: namespacereservations.admission.online.openshift.io (2)
  clientConfig:
```

```
    service: (3)
      namespace: default
      name: kubernetes
      path: /apis/admission.online.openshift.io/v1beta1/namespacereservations (4)
    caBundle: <ca_signing_certificate> (5)
  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 URL 입니다. 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.*

7.6. 추가 리소스

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