



OpenShift Container Platform 4.12

노드

OpenShift Container Platform에서 노드 구성 및 관리

OpenShift Container Platform 4.12 노드

OpenShift Container Platform에서 노드 구성 및 관리

법적 공지

Copyright © 2024 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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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.

초록

이 문서에서는 클러스터의 노드, Pod, 컨테이너를 구성하고 관리하는 방법에 대한 지침을 제공합니다. 또한 Pod 예약 및 배치 구성, 작업 및 DaemonSet를 사용하여 작업 자동화, 클러스터를 효율적으로 유지하는 기타 작업에 대한 정보도 제공합니다.

차례

1장. 노드 개요	4
1.1. 노드 정보	4
1.2. POD 정보	6
1.3. 컨테이너 정보	8
1.4. 노드에서 POD 자동 스케일링 정보	9
1.5. OPENSIFT CONTAINER PLATFORM 노드에 대한 일반 용어집	10
2장. 노드 작업	12
2.1. POD 사용	12
2.2. POD 보기	16
2.3. POD에 대한 OPENSIFT CONTAINER PLATFORM 클러스터 구성	20
2.4. 수평 POD 자동 스케일러를 사용하여 POD 자동 스케일링	28
2.5. 수직 POD 자동 스케일러를 사용하여 POD 리소스 수준 자동 조정	55
2.6. POD에 민감한 데이터 제공	76
2.7. 구성 맵 생성 및 사용	100
2.8. 장치 플러그인을 사용하여 POD가 있는 외부 리소스에 액세스	117
2.9. POD 예약 결정에 POD 우선순위 포함	121
2.10. 노드 선택기를 사용하여 특정 노드에 POD 배치	127
3장. CUSTOM METRICS AUTOSCALER OPERATOR를 사용하여 POD 자동 스케일링	134
3.1. 릴리스 노트	134
3.2. 사용자 정의 METRICS AUTOSCALER OPERATOR 개요	143
3.3. 사용자 정의 메트릭 자동 스케일러 설치	146
3.4. 사용자 정의 메트릭 자동 스케일러 트리거 이해	150
3.5. 사용자 정의 메트릭 자동 스케일러가 인증 트리거 이해	163
3.6. 확장 오브젝트에 대한 사용자 정의 지표 자동 스케일러 일시 중지	171
3.7. 감사 로그 수집	173
3.8. 디버깅 데이터 수집	178
3.9. OPERATOR 메트릭 보기	182
3.10. 사용자 정의 메트릭 자동 스케일러 추가 방법	184
3.11. CUSTOM METRICS AUTOSCALER OPERATOR 제거	195
4장. 노드에 대한 POD 배치 제어(예약)	199
4.1. 스케줄러를 사용하여 POD 배치 제어	199
4.2. 스케줄러 프로필을 사용하여 POD 예약	201
4.3. 유사성 및 유사성 방지 규칙을 사용하여 다른 POD에 상대적인 POD 배치	203
4.4. 노드 유사성 규칙을 사용하여 노드에 대한 POD 배치 제어	217
4.5. 과다 할당된 노드에 POD 배치	230
4.6. 노드 테인트를 사용하여 POD 배치 제어	232
4.7. 노드 선택기를 사용하여 특정 노드에 POD 배치	252
4.8. POD 토폴로지 분배 제약 조건을 사용하여 POD 배치 제어	273
4.9. DESCHEDULER를 사용하여 POD 제거	278
4.10. 보조 스케줄러	290
5장. 작업 및 DAEMONSET 사용	302
5.1. 데몬 세트를 사용하여 노드에서 자동으로 백그라운드 작업 실행	302
5.2. 작업을 사용하여 POD에서 작업 실행	306
6장. 노드 작업	318
6.1. OPENSIFT CONTAINER PLATFORM 클러스터에서 노드 보기 및 나열	318
6.2. 노드 작업	326
6.3. 노드 관리	334

6.4. 노드당 최대 POD 수 관리	349
6.5. NODE TUNING OPERATOR 사용	353
6.6. 노드 수정, 펜싱 및 유지 관리	364
6.7. 노드 재부팅 이해	364
6.8. 가비지 컬렉션을 사용하여 노드 리소스 해제	370
6.9. OPENSIFT CONTAINER PLATFORM 클러스터의 노드에 리소스 할당	377
6.10. 클러스터의 노드에 특정 CPU 할당	386
6.11. KUBELET의 TLS 보안 프로필 활성화	388
6.12. 머신 구성 데몬 메트릭	392
6.13. 인프라 노드 생성	395
7장. 컨테이너 작업	400
7.1. 컨테이너 이해	400
7.2. POD를 배포하기 전에 INIT CONTAINER를 사용하여 작업 수행	402
7.3. 볼륨을 사용하여 컨테이너 데이터 유지	406
7.4. 예상된 볼륨을 사용하여 볼륨 매핑	420
7.5. 컨테이너에서 API 오브젝트를 사용하도록 허용	430
7.6. OPENSIFT CONTAINER PLATFORM 컨테이너에 또한 해당 컨테이너에서 파일 복사	442
7.7. OPENSIFT CONTAINER PLATFORM 컨테이너에서 원격 명령 실행	445
7.8. 포트 전달을 사용하여 컨테이너의 애플리케이션에 액세스	448
7.9. 컨테이너의 SYSCTL 사용	451
8장. 클러스터 작업	471
8.1. OPENSIFT CONTAINER PLATFORM 클러스터에서 시스템 이벤트 정보 보기	471
8.2. OPENSIFT CONTAINER PLATFORM 노드에서 보유할 수 있는 POD 수 추정	481
8.3. 제한 범위를 사용하여 리소스 사용 제한	488
8.4. 컨테이너 메모리 및 위험 요구 사항을 충족하도록 클러스터 메모리 구성	500
8.5. 과다 할당된 노드에 POD를 배치하도록 클러스터 구성	511
8.6. LINUX 제어 그룹 버전 2 (CGROUP V2) 활성화	531
8.7. 기능 게이트를 사용한 기능 활성화	537
8.8. 작업자 대기 시간 프로필을 사용하여 대기 시간이 높은 환경에서 클러스터 안정성 개선	547
9장. 네트워크 엣지의 원격 작업자 노드	556
9.1. 네트워크 엣지에서 원격 작업자 노드 사용	556
10장. 단일 노드 OPENSIFT 클러스터의 작업자 노드	568
10.1. 단일 노드 OPENSIFT 클러스터에 작업자 노드 추가	568

1장. 노드 개요

1.1. 노드 정보

노드는 Kubernetes 클러스터의 가상 또는 베어 메탈 시스템입니다. 작업자 노드는 애플리케이션 컨테이너를 호스팅하고 pod로 그룹화됩니다. 컨트롤 플레인 노드는 Kubernetes 클러스터를 제어하는 데 필요한 서비스를 실행합니다. OpenShift Container Platform에서 컨트롤 플레인 노드에는 OpenShift Container Platform 클러스터를 관리하기 위한 Kubernetes 서비스 외에도 많은 기능이 포함되어 있습니다.

클러스터에서 안정적이고 정상적인 노드를 보유하는 것이 호스트된 애플리케이션의 원활한 작동에 필수적입니다. OpenShift Container Platform에서는 노드를 나타내는 **Node** 오브젝트를 통해 노드에 액세스, 관리 및 모니터링할 수 있습니다. OpenShift CLI(**oc**) 또는 웹 콘솔을 사용하여 노드에서 다음 작업을 수행할 수 있습니다.

노드의 다음 구성 요소는 Pod 실행을 유지하고 Kubernetes 런타임 환경을 제공합니다.

컨테이너 런타임

컨테이너 런타임은 컨테이너 실행을 담당합니다. Kubernetes는 containerd, cri-o, rktlet, Docker와 같은 여러 런타임을 제공합니다.

kubelet

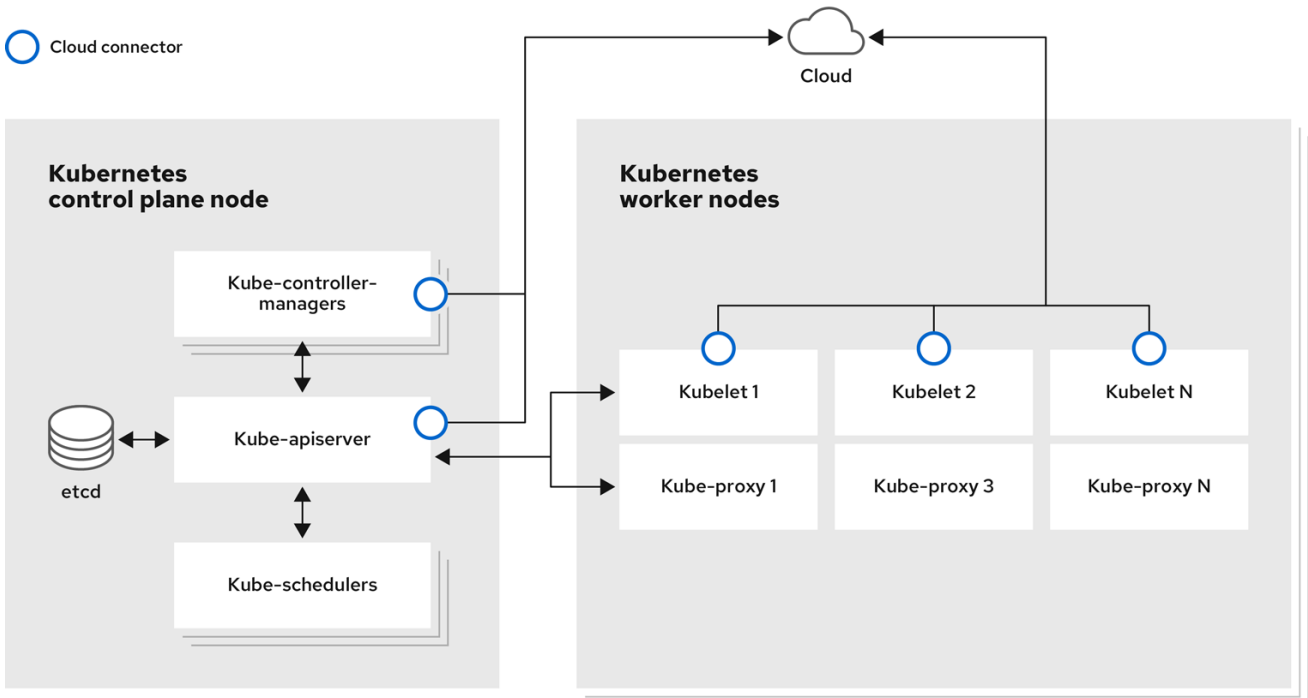
kubelet은 노드에서 실행되며 컨테이너 매니페스트를 읽습니다. 정의된 컨테이너가 시작되고 실행 중인지 확인합니다. kubelet 프로세스는 작업 상태와 노드 서버 상태를 유지합니다. kubelet은 네트워크 규칙 및 포트 전달을 관리합니다. kubelet은 Kubernetes에서 생성한 컨테이너만 관리합니다.

kube-proxy

kube-proxy는 클러스터의 모든 노드에서 실행되며 Kubernetes 리소스 간의 네트워크 트래픽을 유지합니다. Kube-proxy를 사용하면 네트워킹 환경을 분리하고 액세스할 수 있습니다.

DNS

클러스터 DNS는 Kubernetes 서비스에 대한 DNS 레코드를 제공하는 DNS 서버입니다. Kubernetes에서 시작하는 컨테이너는 DNS 검색에 이 DNS 서버를 자동으로 포함합니다.



읽기 작업

읽기 작업을 통해 관리자 또는 개발자가 OpenShift Container Platform 클러스터의 노드에 대한 정보를 가져올 수 있습니다.

- 클러스터의 모든 노드를 나열합니다.
- 메모리 및 CPU 사용량, 상태, 상태 및 경과와 같은 노드에 대한 정보를 가져옵니다.
- 노드에서 실행 중인 Pod를 나열합니다.

관리 운영

관리자는 여러 작업을 통해 OpenShift Container Platform 클러스터에서 노드를 쉽게 관리할 수 있습니다.

- **노드 레이블을 추가하거나 업데이트합니다.** 레이블은 **Node** 오브젝트에 적용되는 키-값 쌍입니다. 라벨을 사용하여 Pod 예약을 제어할 수 있습니다.
- CRD(사용자 정의 리소스 정의) 또는 **kubeletConfig** 오브젝트를 사용하여 노드 구성을 변경합니다.
- Pod 예약을 허용하거나 허용하지 않도록 노드를 구성합니다. 정상 상태의 작업자 노드는 컨트롤 플레인 노드가 없는 동안 기본적으로 Pod 배치를 허용합니다. **작업자 노드를 예약할 수 없고 컨트롤 플레인 노드를 예약할 수 없도록 구성하여** 이 기본 동작을 변경할 수 있습니다.
- **system-reserved** 설정을 사용하여 **노드에 리소스를 할당합니다.** OpenShift Container Platform에서 노드의 최적의 **system-reserved CPU** 및 메모리 리소스를 자동으로 결정하거나 노드에 가장 적합한 리소스를 수동으로 확인하고 설정할 수 있습니다.
- **노드의 프로세서 코어 수, 하드 제한 또는 둘 다에 따라 노드에서 실행할 수 있는 Pod 수를 구성합니다.**
- **Pod 유사성 방지를 사용하여 노드를 정상적으로 재부팅합니다.**
- **컴퓨팅 머신 세트를 사용하여 클러스터를 축소하여 클러스터에서 노드를 삭제합니다.** 베어 메탈 클러스터에서 노드를 삭제하려면 먼저 노드의 모든 Pod를 트레이닝한 다음 노드를 수동으로 삭제해야 합니다.

기능 개선 작업

OpenShift Container Platform을 사용하면 노드 액세스 및 관리 이상의 작업을 수행할 수 있습니다. 관리자는 노드에서 다음 작업을 수행하여 클러스터를 보다 효율적이고 애플리케이션 친화적인 상태로 만들고 개발자에게 더 나은 환경을 제공할 수 있습니다.

- **Node Tuning Operator**를 사용하여 일정 수준의 커널 튜닝이 필요한 고성능 애플리케이션에 대한 노드 수준 튜닝을 관리합니다.

- 노드에서 **TLS** 보안 프로필을 활성화하여 **kubelet**과 **Kubernetes API** 서버 간의 통신을 보호합니다.
- 데몬 세트를 사용하여 노드에서 백그라운드 작업을 자동으로 실행합니다. 데몬 세트를 생성하고 사용하여 공유 스토리지를 생성하거나 모든 노드에서 로깅 **Pod**를 실행하거나 모든 노드에 모니터링 에이전트를 배포할 수 있습니다.
- 가비지 컬렉션을 사용하여 노드 리소스를 확보합니다. 종료된 컨테이너 및 실행 중인 **Pod**에서 참조하지 않는 이미지를 제거하여 노드가 효율적으로 실행되고 있는지 확인할 수 있습니다.
- 노드 집합에 커널 인수를 추가합니다.
- 네트워크 엣지(원격 작업자 노드)의 작업자 노드를 갖도록 **OpenShift Container Platform** 클러스터를 구성합니다. **OpenShift Container Platform** 클러스터의 원격 작업자 노드 및 원격 작업자 노드에서 **Pod**를 관리하는 데 권장되는 몇 가지 접근 방식에 대한 자세한 내용은 [네트워크 에지에서 원격 작업자 노드 사용](#)을 참조하십시오.

1.2. POD 정보

Pod는 노드에 함께 배포되는 하나 이상의 컨테이너입니다. 클러스터 관리자는 **Pod**를 정의하고 예약 및 관리할 수 있는 정상 노드에서 실행되도록 할당할 수 있습니다. **Pod**는 컨테이너가 실행되는 동안 실행됩니다. **Pod**가 정의되어 있고 실행 중인 후에는 **Pod**를 변경할 수 없습니다. **Pod** 작업 시 수행할 수 있는 일부 작업은 다음과 같습니다.

읽기 작업

관리자는 다음 작업을 통해 프로젝트의 **Pod**에 대한 정보를 가져올 수 있습니다.

- 복제본 수 및 재시작, 현재 상태 및 경과와 같은 정보를 포함하여 [프로젝트와 연결된 Pod를 나열](#)합니다.
- **CPU**, 메모리, 스토리지 소비와 같은 [Pod 사용량 통계](#)를 확인합니다.

관리 운영

다음 작업 목록은 관리자가 **OpenShift Container Platform** 클러스터에서 **Pod**를 관리하는 방법에 대한 개요를 제공합니다.

- OpenShift Container Platform에서 사용할 수 있는 고급 예약 기능을 사용하여 Pod 예약을 제어합니다.
 - Pod 유사성, 노드 유사성 및 유사성 방지와 같은 **node-to-pod** 바인딩 규칙입니다.
 - 노드 레이블 및 선택기.
 - 테인트 및 허용 오차.
 - Pod 토폴로지 분배 제약 조건.
 - 보조 예약.
- 스케줄러가 더 적절한 노드로 Pod를 다시 예약하도록 특정 전략에 따라 Pod를 제거하도록 **Descheduler**를 구성합니다.
- Pod 컨트롤러를 사용하여 재시작하고 정책을 다시 시작한 후 Pod가 작동하는 방식을 구성합니다.
- Pod에서 송신 및 수신 트래픽을 모두 제한합니다.
- Pod 템플릿이 있는 모든 오브젝트에 볼륨을 추가하고 제거합니다. 볼륨은 Pod의 모든 컨테이너에서 사용할 수 있는 마운트된 파일 시스템입니다. 컨테이너 스토리지는 임시 스토리지입니다. 볼륨을 사용하여 컨테이너 데이터를 유지할 수 있습니다.

기능 개선 작업

OpenShift Container Platform에서 사용할 수 있는 다양한 툴 및 기능을 통해 Pod를 보다 쉽고 효율적으로 작업할 수 있습니다. 다음 작업에는 이러한 툴 및 기능을 사용하여 Pod를 보다 효과적으로 관리해야 합니다.

작업	사용자	더 알아보기
수평 Pod 자동 스케일러를 생성하고 사용합니다.	개발자	수평 Pod 자동 스케일러를 사용하여 실행하려는 최소 및 최대 Pod 수와 Pod에서 목표로 하는 CPU 사용률 또는 메모리 사용률을 지정할 수 있습니다. 수평 Pod 자동 스케일러를 사용하면 Pod 를 자동으로 스케일링 할 수 있습니다.
수직 Pod 자동 스케일러를 설치하고 사용합니다.	관리자 및 개발자	관리자는 수직 Pod 자동 스케일러를 사용하여 리소스와 워크로드의 리소스 요구 사항을 모니터링하여 클러스터 리소스를 더 효율적으로 사용합니다. 개발자는 수직 Pod 자동 스케일러를 사용하여 각 Pod에 충분한 리소스가 있는 노드에 Pod를 예약하여 수요가 많은 기간 동안 Pod를 계속 사용할 수 있도록 합니다.
장치 플러그인을 사용하여 외부 리소스에 대한 액세스 권한을 제공합니다.	시스템 관리자	장치 플러그인은 특정 하드웨어 리소스를 관리하는 노드(kubelet 외부)에서 실행되는 gRPC 서비스입니다. 장치 플러그인을 배포하여 클러스터 전체에서 하드웨어 장치를 소비하는 일관되고 이식 가능한 솔루션을 제공할 수 있습니다.
Secret 오브젝트를 사용하여 Pod에 민감한 데이터를 제공합니다.	시스템 관리자	일부 애플리케이션에는 암호 및 사용자 이름과 같은 중요한 정보가 필요합니다. Secret 오브젝트를 사용하여 애플리케이션 포트에 이러한 정보를 제공할 수 있습니다.

1.3. 컨테이너 정보

컨테이너는 종속 항목, 라이브러리 및 바이너리와 함께 패키징된 애플리케이션 코드로 구성된 OpenShift Container Platform 애플리케이션의 기본 단위입니다. 컨테이너에서는 물리 서버, 가상 머신 (VM) 및 프라이빗 또는 퍼블릭 클라우드와 같은 환경 및 여러 배치 대상 사이에 일관성을 제공합니다.

Linux 컨테이너 기술은 실행 중인 프로세스를 분리하고 지정된 리소스로만 액세스를 제한하는 간단한 메커니즘입니다. 관리자는 다음과 같은 Linux 컨테이너에서 다양한 작업을 수행할 수 있습니다.

- 컨테이너에서 또는 컨테이너에 파일을 복사합니다.

- 컨테이너에서 **API** 오브젝트를 사용하도록 허용합니다.
- 컨테이너에서 원격 명령을 실행합니다.
- 포트 전달을 사용하여 컨테이너의 애플리케이션에 액세스 합니다.

OpenShift Container Platform은 **Init 컨테이너**라는 특수 컨테이너를 제공합니다. **Init** 컨테이너는 애플리케이션 컨테이너 전에 실행되며 애플리케이션 이미지에 없는 유틸리티 또는 설정 스크립트를 포함할 수 있습니다. **Init** 컨테이너를 사용하여 나머지 **Pod**를 배포하기 전에 작업을 수행할 수 있습니다.

노드, **Pod** 및 컨테이너에서 특정 작업을 수행하는 것 외에도 전체 **OpenShift Container Platform** 클러스터와 함께 작업하여 클러스터와 애플리케이션 **Pod**의 고가용성을 유지할 수 있습니다.

1.4. 노드에서 **POD** 자동 스케일링 정보

OpenShift Container Platform에서는 노드의 **Pod** 수와 **Pod**에 할당된 리소스를 자동으로 확장하는데 사용할 수 있는 세 가지 틀을 제공합니다.

수평 Pod Autoscaler

HPA(horizontal Pod Autoscaler)는 복제 컨트롤러 또는 배포 구성에 속하는 **Pod**에서 수집한 메트릭에 따라 복제 컨트롤러 또는 배포 구성의 규모를 자동으로 늘리거나 줄일 수 있습니다.

자세한 내용은 [수평 Pod 자동 스케일러를 사용하여 Pod 자동 스케일링을 참조하십시오.](#)

사용자 정의 지표 자동 스케일러

사용자 정의 메트릭 자동 스케일러는 **CPU** 또는 메모리를 기반으로 하지 않는 사용자 정의 메트릭을 기반으로 배포, 상태 저장 세트, 사용자 정의 리소스 또는 작업의 **Pod** 수를 자동으로 늘리거나 줄일 수 있습니다.

자세한 내용은 [Custom Metrics Autoscaler Operator 개요](#) 를 참조하십시오.

Vertical Pod Autoscaler

VPA(Vertical Pod Autoscaler)는 **Pod**에서 컨테이너의 기록 및 현재 **CPU** 및 메모리 리소스를 자동으로 확인하고 학습한 사용량 값에 따라 리소스 제한 및 요청을 업데이트할 수 있습니다.

자세한 내용은 수직 **Pod** 자동 스케일러를 사용하여 **Pod** 리소스 수준 자동 조정을 참조하십시오.

1.5. OPENSIFT CONTAINER PLATFORM 노트에 대한 일반 용어집

이 용어집은 노트 콘텐츠에 사용되는 공통 용어를 정의합니다.

컨테이너

소프트웨어 및 모든 종속 항목으로 구성된 경량 실행 이미지입니다. 결과적으로 운영 체제를 가상화하여 데이터 센터에서 퍼블릭 또는 프라이빗 클라우드로 어디에서나 컨테이너를 실행할 수 있습니다.

데몬 세트

Pod 복제본이 **OpenShift Container Platform** 클러스터의 적격 노트에서 실행되는지 확인합니다.

egress

포드에서 네트워크 아웃바운드 트래픽을 통해 외부적으로 공유하는 데이터 프로세스입니다.

가비지 컬렉션

실행 중인 **Pod**에서 참조하지 않는 종료된 컨테이너 및 이미지와 같은 클러스터 리소스를 정리하는 프로세스입니다.

HPA(Horizontal Pod Autoscaler)

Kubernetes API 리소스 및 컨트롤러로 구현됩니다. **HPA**를 사용하여 실행하려는 최소 및 최대 **Pod** 수를 지정할 수 있습니다. **Pod**에서 대상으로 하는 **CPU** 또는 메모리 사용률을 지정할 수도 있습니다. **HPA**는 지정된 **CPU** 또는 메모리 임계값이 교차하면 **Pod**를 확장하고 축소합니다.

Ingress

포드에 들어오는 트래픽입니다.

Job

완료될 때까지 실행되는 프로세스입니다. 작업은 하나 이상의 **Pod** 오브젝트를 생성하고 지정된 **Pod**가 성공적으로 완료되었는지 확인합니다.

라벨

키-값 쌍인 레이블을 사용하여 **Pod**와 같은 오브젝트의 하위 집합을 구성하고 선택할 수 있습니다.

노드

OpenShift Container Platform 클러스터의 작업자 머신. 노드는 **VM**(가상 머신) 또는 물리적 머신일 수 있습니다.

Node Tuning Operator

Node Tuning Operator를 사용하여 **Tuned** 데몬을 사용하여 노드 수준 튜닝을 관리할 수 있습니다. 이를 통해 사용자 정의 튜닝 사양이 데몬이 이해할 수 있는 형식으로 클러스터에서 실행되는 모든 컨테이너화된 **Tuned** 데몬에 전달됩니다. 데몬은 클러스터의 모든 노드에서 노드당 하나씩 실행됩니다.

자체 노드 해결 Operator

Operator는 클러스터 노드에서 실행되며 비정상인 노드를 식별하고 재부팅합니다.

Pod

OpenShift Container Platform 클러스터에서 실행되는 볼륨 및 **IP** 주소와 같은 공유 리소스를 사용하는 하나 이상의 컨테이너입니다. **Pod**는 정의된 최소 컴퓨팅 단위, 배포, 관리입니다.

허용 오차

테인트가 일치하는 노드 또는 노드 그룹에 **Pod**를 예약할 수 있음을 나타냅니다. 허용 오차를 사용하여 스케줄러에서 일치하는 테인트로 **Pod**를 예약할 수 있습니다.

taint

키, 값 및 효과를 포함하는 코어 오브젝트입니다. 테인트(**Taints**) 및 톨러레이션(**Tolerations**)이 함께 작동하여 **Pod**가 관련 없는 노드에서 예약되지 않도록 합니다.

2장. 노트 작업

2.1. POD 사용

Pod는 하나의 호스트에 함께 배포되는 하나 이상의 컨테이너이자 정의, 배포, 관리할 수 있는 최소 컴퓨팅 단위입니다.

2.1.1. Pod 이해

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

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

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



참고

OpenShift Container Platform 노트 호스트당 최대 **Pod** 수는 클러스터 제한을 참조하십시오.



주의

복제 컨트롤러에서 관리하지 않는 배어 **Pod**는 노트 중단 시 다시 예약되지 않습니다.

2.1.2. Pod 구성의 예

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

다음은 **Rails** 애플리케이션의 **Pod** 정의 예입니다. 이 예제에서는 **Pod**의 다양한 기능을 보여줍니다. 대부분 다른 주제에서 설명하므로 여기에서는 간단히 언급합니다.

Pod 오브젝트 정의(YAML)

```

kind: Pod
apiVersion: v1
metadata:
  name: example
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/example
  uid: 5cc30063-0265780783bc
  resourceVersion: '165032'
  creationTimestamp: '2019-02-13T20:31:37Z'
  labels:
    app: hello-openshift ①
  annotations:
    openshift.io/scc: anyuid
spec:
  restartPolicy: Always ②
  serviceAccountName: default
  imagePullSecrets:
    - name: default-dockercfg-5zrhb
  priority: 0
  schedulerName: default-scheduler
  terminationGracePeriodSeconds: 30
  nodeName: ip-10-0-140-16.us-east-2.compute.internal
  securityContext: ③
    seLinuxOptions:
      level: 's0:c11,c10'
  containers: ④
    - resources: {}
      terminationMessagePath: /dev/termination-log
      name: hello-openshift
      securityContext:
        capabilities:
          drop:
            - MKNOD
        procMount: Default
      ports:
        - containerPort: 8080
          protocol: TCP
      imagePullPolicy: Always
      volumeMounts: ⑤
        - name: default-token-wbqsl
          readOnly: true

```

```

    mountPath: /var/run/secrets/kubernetes.io/serviceaccount 6
    terminationMessagePolicy: File
    image: registry.redhat.io/openshift4/ose-logging-eventrouter:v4.3 7
serviceAccount: default 8
volumes: 9
- name: default-token-wbqsl
  secret:
    secretName: default-token-wbqsl
    defaultMode: 420
dnsPolicy: ClusterFirst
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2019-02-13T20:31:37Z'
    - type: Ready
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2019-02-13T20:31:37Z'
      reason: ContainersNotReady
      message: 'containers with unready status: [hello-openshift]'
    - type: ContainersReady
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2019-02-13T20:31:37Z'
      reason: ContainersNotReady
      message: 'containers with unready status: [hello-openshift]'
    - type: PodScheduled
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2019-02-13T20:31:37Z'
  hostIP: 10.0.140.16
  startTime: '2019-02-13T20:31:37Z'
  containerStatuses:
    - name: hello-openshift
      state:
        waiting:
          reason: ContainerCreating
        lastState: {}
        ready: false
        restartCount: 0
        image: openshift/hello-openshift
        imageID: ""
  qosClass: BestEffort

```

1

Pod는 단일 작업에서 **Pod** 그룹을 선택하고 관리하는 데 사용할 수 있는 라벨을 하나 이상 사용하여 "태그를 지정"할 수 있습니다. 라벨은 **metadata** 해시의 키/값 형식으로 저장됩니다.

2

Pod는 가능한 값 **Always, OnFailure, Never**를 사용하여 정책을 재시작합니다. 기본값은 **Always**입니다.

3

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

4

containers는 하나 이상의 컨테이너 정의로 이루어진 배열을 지정합니다.

5

이 컨테이너는 컨테이너 내에서 외부 스토리지 볼륨이 마운트되는 위치를 지정합니다. 이 경우 레지스트리에서 **OpenShift Container Platform API**에 대해 요청하는 데 필요한 자격 증명 액세스 권한을 저장하는 볼륨이 있습니다.

6

Pod에 제공할 볼륨을 지정합니다. 지정된 경로에 볼륨이 마운트됩니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 **/dev/pts** 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. **/host**를 사용하여 호스트를 마운트하는 것이 안전합니다.

7

Pod의 각 컨테이너는 자체 컨테이너 이미지에서 인스턴스화됩니다.

8

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

9

Pod는 사용할 컨테이너에서 사용할 수 있는 스토리지 볼륨을 정의합니다. 이 경우 기본 서비스 계정 토큰이 포함된 보안 볼륨에 임시 볼륨을 제공합니다.

파일이 많은 영구 볼륨을 **Pod**에 연결하면 해당 **Pod**가 실패하거나 시작하는 데 시간이 오래 걸릴 수 있습니다. 자세한 내용은 **OpenShift**에서 파일 수가 많은 영구 볼륨을 사용하는 경우 포드를 시작하지 못하거나 **"Ready"** 상태를 달성하는 데 과도한 시간이 걸리는 이유를 참조하십시오.



참고

이 **Pod** 정의에는 **Pod**가 생성되고 해당 라이프사이클이 시작된 후 **OpenShift Container Platform**에 의해 자동으로 채워지는 특성은 포함되지 않습니다. [Kubernetes Pod 설명서](#)에는 **Pod**의 기능 및 용도에 대한 세부 정보가 있습니다.

2.1.3. 추가 리소스

- **Pod** 및 스토리지에 대한 자세한 내용은 [영구 스토리지 이해 및 임시 스토리지 이해](#) 를 참조하십시오.

2.2. POD 보기

관리자는 클러스터의 **Pod**를 보고 해당 **Pod** 및 클러스터의 상태를 전체적으로 확인할 수 있습니다.

2.2.1. Pod 정보

OpenShift Container Platform에서는 하나의 호스트에 함께 배포되는 하나 이상의 컨테이너이자 정의, 배포, 관리할 수 있는 최소 컴퓨팅 단위인 **Pod**의 **Kubernetes** 개념을 활용합니다. **Pod**는 컨테이너에 대한 머신 인스턴스(실제 또는 가상)와 대략적으로 동일합니다.

특정 프로젝트와 연결된 **Pod** 목록을 확인하거나 **Pod** 관련 사용량 통계를 볼 수 있습니다.

2.2.2. 프로젝트의 Pod 보기

Pod의 복제본 수, 현재 상태, 재시작 횟수, 수명 등을 포함하여 현재 프로젝트와 관련된 **Pod** 목록을 확인할 수 있습니다.

프로세스

프로젝트의 **Pod**를 보려면 다음을 수행합니다.

1. 프로젝트로 변경합니다.

```
$ oc project <project-name>
```

2.

다음 명령을 실행합니다.

```
$ oc get pods
```

예를 들면 다음과 같습니다.

```
$ oc get pods
```

출력 예

```
NAME                READY STATUS RESTARTS AGE
console-698d866b78-bnshf 1/1 Running 2      165m
console-698d866b78-m87pm 1/1 Running 2      165m
```

Pod IP 주소와 Pod가 있는 노드를 보려면 **-o wide** 플래그를 추가합니다.

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

출력 예

```
NAME                READY STATUS RESTARTS AGE IP      NODE
NOMINATED NODE
console-698d866b78-bnshf 1/1 Running 2      166m 10.128.0.24 ip-10-0-152-71.ec2.internal <none>
console-698d866b78-m87pm 1/1 Running 2      166m 10.129.0.23 ip-10-0-173-237.ec2.internal <none>
```

2.2.3. Pod 사용량 통계 보기

컨테이너의 런타임 환경을 제공하는 Pod에 대한 사용량 통계를 표시할 수 있습니다. 이러한 사용량 통계에는 CPU, 메모리, 스토리지 사용량이 포함됩니다.

사전 요구 사항

- 사용량 통계를 보려면 **cluster-reader** 권한이 있어야 합니다.
- 사용량 통계를 보려면 메트릭이 설치되어 있어야 합니다.

프로세스

사용량 통계를 보려면 다음을 수행합니다.

1. 다음 명령을 실행합니다.

```
$ oc adm top pods
```

예를 들면 다음과 같습니다.

```
$ oc adm top pods -n openshift-console
```

출력 예

```
NAME                CPU(cores) MEMORY(bytes)
console-7f58c69899-q8c8k  0m          22Mi
console-7f58c69899-xhbgg  0m          25Mi
downloads-594fccc94-bcxk8  3m          18Mi
downloads-594fccc94-kv4p6  2m          15Mi
```

2. 다음 명령을 실행하여 라벨이 있는 **Pod**의 사용량 통계를 확인합니다.

```
$ oc adm top pod --selector="
```

필터링할 선택기(라벨 쿼리)를 선택해야 합니다. =, ==, !=가 지원됩니다.

예를 들면 다음과 같습니다.

```
$ oc adm top pod --selector='name=my-pod'
```

2.2.4. 리소스 로그 보기

OpenShift CLI(oc) 및 웹 콘솔에서 다양한 리소스의 로그를 볼 수 있습니다. 로그는 로그의 말미 또는 끝에서 읽습니다.

사전 요구 사항

- **OpenShift CLI(oc)**에 액세스합니다.

프로세스(UI)

1. **OpenShift Container Platform** 콘솔에서 워크로드 → **Pod**로 이동하거나 조사하려는 리소스를 통해 **Pod**로 이동합니다.



참고

빌드와 같은 일부 리소스에는 직접 쿼리할 **Pod**가 없습니다. 이러한 인스턴스에서 리소스의 세부 정보 페이지에서 로그 링크를 찾을 수 있습니다.

2. 드롭다운 메뉴에서 프로젝트를 선택합니다.
3. 조사할 **Pod** 이름을 클릭합니다.
4. 로그를 클릭합니다.

프로세스(CLI)

- 특정 **Pod**의 로그를 확인합니다.

```
$ oc logs -f <pod_name> -c <container_name>
```

다음과 같습니다.

-f

선택 사항: 출력에서 로그에 기록되는 내용을 따르도록 지정합니다.

<pod_name>

pod 이름을 지정합니다.

<container_name>

선택 사항: 컨테이너의 이름을 지정합니다. Pod에 여러 컨테이너가 있는 경우 컨테이너 이름을 지정해야 합니다.

예를 들면 다음과 같습니다.

```
$ oc logs ruby-58cd97df55-mww7r
```

```
$ oc logs -f ruby-57f7f4855b-znl92 -c ruby
```

로그 파일의 내용이 출력됩니다.

- 특정 리소스의 로그를 확인합니다.

```
$ oc logs <object_type>/<resource_name> ①
```

①

리소스 유형 및 이름을 지정합니다.

예를 들면 다음과 같습니다.

```
$ oc logs deployment/ruby
```

로그 파일의 내용이 출력됩니다.

2.3. POD에 대한 OPENSIFT CONTAINER PLATFORM 클러스터 구성

관리자는 **Pod**에 효율적인 클러스터를 생성하고 유지 관리할 수 있습니다.

클러스터를 효율적으로 유지하면 **Pod**가 종료될 때 수행하는 작업과 같은 틀을 사용하여 개발자에게 더 나은 환경을 제공할 수 있습니다. 즉 필요한 수의 **Pod**가 항상 실행되고 있는지 확인하여 한 번만 실행되도록 설계된 **Pod**를 재시작하는 경우 **Pod**에 사용할 수 있는 대역폭을 제한하고, 중단 중 **Pod**를 계속 실행하는 방법을 제공합니다.

2.3.1. 재시작 후 Pod 작동 방식 구성

Pod 재시작 정책에 따라 해당 **Pod**의 컨테이너가 종료될 때 **OpenShift Container Platform**에서 응답하는 방법이 결정됩니다. 정책은 해당 **Pod**의 모든 컨테이너에 적용됩니다.

가능한 값은 다음과 같습니다.

- Always** - 급격한 백오프 지연(10초, 20초, 40초)을 5분으로 제한하여 **Pod**에서 성공적으로 종료된 컨테이너를 지속적으로 재시작합니다. 기본값은 **Always**입니다.
- OnFailure** - 급격한 백오프 지연(10초, 20초, 40초)을 5분으로 제한하여 **Pod**에서 실패한 컨테이너를 재시작합니다.
- Never** - **Pod**에서 종료되거나 실패한 컨테이너를 재시작하지 않습니다. **Pod**가 즉시 실패하고 종료됩니다.

Pod가 특정 노드에 바인딩된 후에는 다른 노드에 바인딩되지 않습니다. 따라서 노드 장애 시 **Pod**가 작동하려면 컨트롤러가 필요합니다.

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

Pod의 컨테이너가 실패하고 재시작 정책이 **OnFailure**로 설정된 경우 **Pod**가 노드에 남아 있고 컨테이

너가 재시작됩니다. 컨테이너를 재시작하지 않으려면 재시작 정책 **Never**를 사용하십시오.

전체 **Pod**가 실패하면 **OpenShift Container Platform**에서 새 **Pod**를 시작합니다. 개발자는 애플리케이션이 새 **Pod**에서 재시작될 수 있는 가능성을 고려해야 합니다. 특히 애플리케이션에서는 이전 실행으로 발생한 임시 파일, 잠금, 불완전한 출력 등을 처리해야 합니다.



참고

Kubernetes 아키텍처에서는 클라우드 공급자의 끝점이 안정적인 것으로 예상합니다. 클라우드 공급자가 중단되면 **kubelet**에서 **OpenShift Container Platform**이 재시작되지 않습니다.

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

OpenShift Container Platform에서 실패한 컨테이너에 재시작 정책을 사용하는 방법에 대한 자세한 내용은 **Kubernetes** 설명서의 [예제 상태](#)를 참조하십시오.

2.3.2. Pod에서 사용할 수 있는 대역폭 제한

Pod에 서비스 품질 트래픽 조절 기능을 적용하고 사용 가능한 대역폭을 효과적으로 제한할 수 있습니다. **Pod**에서 송신하는 트래픽은 구성된 속도를 초과하는 패킷을 간단히 삭제하는 정책에 따라 처리합니다. **Pod**에 수신되는 트래픽은 데이터를 효과적으로 처리하기 위해 대기 중인 패킷을 구성하여 처리합니다. 특정 **Pod**에 대한 제한 사항은 다른 **Pod**의 대역폭에 영향을 미치지 않습니다.

프로세스

Pod의 대역폭을 제한하려면 다음을 수행합니다.

1. 오브젝트 정의 **JSON** 파일을 작성하고 **kubernetes.io/ingress-bandwidth** 및 **kubernetes.io/egress-bandwidth** 주석을 사용하여 데이터 트래픽 속도를 지정합니다. 예를 들어 **Pod** 송신 및 수신 대역폭을 둘 다 **10M/s**로 제한하려면 다음을 수행합니다.

제한된 **Pod** 오브젝트 정의

```
{
```

```

"kind": "Pod",
"spec": {
  "containers": [
    {
      "image": "openshift/hello-openshift",
      "name": "hello-openshift"
    }
  ]
},
"apiVersion": "v1",
"metadata": {
  "name": "iperf-slow",
  "annotations": {
    "kubernetes.io/ingress-bandwidth": "10M",
    "kubernetes.io/egress-bandwidth": "10M"
  }
}
}

```

2.

오브젝트 정의를 사용하여 **Pod**를 생성합니다.

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

2.3.3. Pod 중단 예산을 사용하여 실행 중인 pod 수를 지정하는 방법

Pod 중단 예산을 사용하면 유지보수를 위해 노드를 드레이닝하는 등 작업 중에 **Pod**에 대한 보안 제약 조건을 지정할 수 있습니다.

PodDisruptionBudget은 동시에 작동해야 하는 최소 복제본 수 또는 백분율을 지정하는 **API** 오브젝트입니다. 프로젝트에서 이러한 설정은 노드 유지 관리 (예: 클러스터 축소 또는 클러스터 업그레이드) 중에 유용할 수 있으며 (노드 장애 시가 아니라) 자발적으로 제거된 경우에만 적용됩니다.

PodDisruptionBudget 오브젝트의 구성은 다음과 같은 주요 부분으로 구성되어 있습니다.

- 일련의 **pod**에 대한 라벨 쿼리 기능인 라벨 선택기입니다.
- 동시에 사용할 수 있어야 하는 최소 **pod** 수를 지정하는 가용성 수준입니다.

- **minAvailable**은 중단 중에도 항상 사용할 수 있어야하는 pod 수입니다.
- **maxUnavailable**은 중단 중에 사용할 수없는 pod 수입니다.



참고

Available 은 **condition Ready=True** 가 있는 **Pod** 수를 나타냅니다. **ready=True** 는 요청을 처리할 수 있는 **Pod**를 참조하며 일치하는 모든 서비스의 부하 분산 풀에 추가해야 합니다.

maxUnavailable 0 % 또는 **0**이나 **minAvailable**의 **100 %** 혹은 복제본 수와 동일한 값은 허용되지만 이로 인해 노드가 드레인되지 않도록 차단할 수 있습니다.

다음을 사용하여 모든 프로젝트에서 **pod** 중단 예산을 확인할 수 있습니다.

```
$ oc get poddisruptionbudget --all-namespaces
```

출력 예

NAMESPACE	NAME	MIN AVAILABLE	MAX UNAVAILABLE	ALLOWED DISRUPTIONS	AGE
openshift-apiserver	openshift-apiserver-pdb	N/A	1	1	121m
openshift-cloud-controller-manager	aws-cloud-controller-manager	1	N/A		125m
openshift-cloud-credential-operator	pod-identity-webhook	1	N/A		117m
openshift-cluster-csi-drivers	aws-ebs-csi-driver-controller-pdb	N/A	1		121m
openshift-cluster-storage-operator	csi-snapshot-controller-pdb	N/A	1		122m
openshift-cluster-storage-operator	csi-snapshot-webhook-pdb	N/A	1		122m
openshift-console	console	N/A	1	1	116m
#...					

PodDisruptionBudget은 시스템에서 최소 **minAvailable pod**가 실행중인 경우 정상으로 간주됩니다. 이 제한을 초과하는 모든 **pod**는 제거할 수 있습니다.



참고

Pod 우선 순위 및 선점 설정에 따라 우선 순위가 낮은 **pod**는 **pod** 중단 예산 요구 사항을 무시하고 제거될 수 있습니다.

2.3.3.1. Pod 중단 예산을 사용하여 실행해야 할 pod 수 지정

PodDisruptionBudget 오브젝트를 사용하여 동시에 가동되어야 하는 최소 복제본 수 또는 백분율을 지정할 수 있습니다.

프로세스

pod 중단 예산을 구성하려면 다음을 수행합니다.

1. 다음과 같은 오브젝트 정의를 사용하여 **YAML** 파일을 만듭니다.

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 2
  selector: 3
    matchLabels:
      name: my-pod
```

1

PodDisruptionBudget 은 **policy/v1 API** 그룹의 일부입니다.

2

동시에 사용할 수 필요가 있는 최소 **pod** 수입니다. 정수 또는 백분율 (예: **20 %**)을 지정하는 문자열을 사용할 수 있습니다.

3

리소스 집합에 대한 라벨 쿼리입니다. **matchLabels** 및 **matchExpressions**의 결과는 논리적으로 결합됩니다. 이 **parameter**(예: **selector { }**)를 비워 두고 프로젝트의 모든 포드를 선택합니다.

또는 다음을 수행합니다.

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      name: my-pod
```

1

PodDisruptionBudget 은 policy/v1 API 그룹의 일부입니다.

2

동시에 사용할 수 없는 최대 pod 수입니다. 정수 또는 백분율 (예: 20 %)을 지정하는 문자열을 사용할 수 있습니다.

3

리소스 집합에 대한 라벨 쿼리입니다. matchLabels 및 matchExpressions의 결과는 논리적으로 결합됩니다. 이 parameter(예: selector {})를 비워 두고 프로젝트의 모든 포드를 선택합니다.

2.

다음 명령을 실행하여 오브젝트를 프로젝트에 추가합니다.

```
$ oc create -f </path/to/file> -n <project_name>
```

2.3.4. 중요 Pod를 사용하여 Pod 제거 방지

완전히 작동하는 클러스터에 중요하지만 마스터가 아닌 일반 클러스터 노드에서 실행되는 다양한 핵심 구성 요소가 있습니다. 중요한 추가 기능이 제거되면 클러스터가 제대로 작동하지 않을 수 있습니다.

중요로 표시된 Pod는 제거할 수 없습니다.

프로세스

Pod를 중요로 설정하려면 다음을 수행합니다.

1. Pod 사양을 생성하거나 **system-cluster-critical** 우선순위 클래스를 포함하도록 기존 Pod를 편집합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pdb
spec:
  template:
    metadata:
      name: critical-pod
      priorityClassName: system-cluster-critical 1

```

1

노드에서 제거해서는 안 되는 Pod의 기본 우선순위 클래스입니다.

또는 클러스터에 중요한 Pod에 대해 **system-node-critical**을 지정할 수 있지만 필요한 경우 제거할 수도 있습니다.

2. Pod를 생성합니다.

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

2.3.5. 파일 수가 많은 영구 볼륨을 사용할 때 Pod 타임아웃 감소

스토리지 볼륨에 많은 파일(~1,000,000개 이상)이 포함된 경우 Pod 시간 초과가 발생할 수 있습니다.

이는 볼륨이 마운트될 때 Pod의 **securityContext**에 지정된 **fsGroup**과 일치하도록 **OpenShift Container Platform**에서 각 볼륨의 콘텐츠의 소유권 및 권한을 재귀적으로 변경하기 때문에 발생할 수 있습니다. 대규모 볼륨의 경우 소유권 및 권한을 확인하고 변경하는 데 시간이 오래 걸리므로 Pod 시작 속도가 매우 느려질 수 있습니다.

다음 해결 방법 중 하나를 적용하여 이 지연을 줄일 수 있습니다.

- SCC(보안 컨텍스트 제약 조건)를 사용하여 볼륨의 SELinux 재레이블을 건너뛸니다.
- SCC 내에서 **fsGroupChangePolicy** 필드를 사용하여 **OpenShift Container Platform**에서

볼륨의 소유권 및 권한을 확인하고 관리하는 방법을 제어합니다.

-

런타임 클래스를 사용하여 볼륨의 SELinux 재레이블을 건너뛰니다.

자세한 내용은 [OpenShift에서 파일 수가 많은 영구 볼륨을 사용할 때 Pod가 시작되지 않거나 "Ready" 상태를 달성하는 데 과도한 시간이 걸리는 이유는 무엇입니까?](#)

2.4. 수평 POD 자동 스케일러를 사용하여 POD 자동 스케일링

개발자는 HPA(수평 Pod 자동 스케일러)를 사용하여 해당 복제 컨트롤러 또는 배포 구성에 속하는 Pod에서 수집한 메트릭을 기반으로 OpenShift Container Platform에서 복제 컨트롤러 또는 배포 구성의 규모를 자동으로 늘리거나 줄이는 방법을 지정할 수 있습니다. 배포, 배포 구성, 복제본 세트, 복제 컨트롤러 또는 상태 저장 세트에 대해 HPA를 생성할 수 있습니다.

사용자 정의 메트릭을 기반으로 Pod 스케일링에 대한 자세한 내용은 사용자 정의 메트릭을 [기본으로 Pod 자동 스케일링](#)을 참조하십시오.



참고

다른 오브젝트에서 제공하는 특정 기능이나 동작이 필요하지 않은 경우 Deployment 오브젝트 또는 ReplicaSet 오브젝트를 사용하는 것이 좋습니다. 이러한 오브젝트에 대한 자세한 내용은 [Deployment 및 DeploymentConfig 오브젝트 이해](#) 를 참조하십시오.

2.4.1. 수평 Pod 자동 스케일러 이해

수평 Pod 자동 스케일러를 생성하여 실행하려는 최소 및 최대 Pod 수와 Pod에서 목표로 하는 CPU 사용률 또는 메모리 사용률을 지정할 수 있습니다.

수평 Pod 자동 스케일러를 생성하면 OpenShift Container Platform에서 Pod의 CPU 및/또는 메모리 리소스 메트릭을 쿼리합니다. 이러한 메트릭을 사용할 수 있는 경우 수평 Pod 자동 스케일러에서 현재 메트릭 사용률과 원하는 메트릭 사용률의 비율을 계산하고 그에 따라 확장 또는 축소합니다. 쿼리 및 스케일링은 정기적으로 수행되지만 메트릭을 사용할 수 있을 때까지 1~2분이 걸릴 수 있습니다.

복제 컨트롤러의 경우 이러한 스케일링은 복제 컨트롤러의 복제본과 직접적으로 일치합니다. 배포 구성의 경우 스케일링은 배포 구성의 복제본 수와 직접적으로 일치합니다. 자동 스케일링은 Complete 단계에서 최신 배포에만 적용됩니다.

OpenShift Container Platform은 리소스를 자동으로 차지하여 시작하는 동안과 같이 리소스가 급증하는 동안 불필요한 자동 스케일링을 방지합니다. **unready** 상태의 **Pod**는 확장 시 **CPU** 사용량이 **0**이고, 축소 시에는 자동 스케일러에서 **Pod**를 무시합니다. 알려진 메트릭이 없는 **Pod**는 확장 시 **CPU** 사용량이 **0%**이고, 축소 시에는 **100%**입니다. 이를 통해 **HPA**를 결정하는 동안 안정성이 향상됩니다. 이 기능을 사용하려면 준비 상태 점검을 구성하여 새 **Pod**를 사용할 준비가 되었는지 확인해야 합니다.

수평 **Pod** 자동 스케일러를 사용하려면 클러스터 관리자가 클러스터 메트릭을 올바르게 구성해야 합니다.

2.4.1.1. 지원되는 메트릭

수평 **Pod** 자동 스케일러에서는 다음 메트릭을 지원합니다.

표 2.1. 메트릭

메트릭	설명	API 버전
CPU 사용	사용되는 CPU 코어의 수입니다. Pod에서 요청하는 CPU의 백분율을 계산하는데 사용할 수 있습니다.	autoscaling/v1, autoscaling/v2
메모리 사용률	사용되는 메모리의 양입니다. Pod에서 요청하는 메모리의 백분율을 계산하는데 사용할 수 있습니다.	autoscaling/v2

중요

메모리 기반 자동 스케일링의 경우 메모리 사용량이 복제본 수에 비례하여 증가 및 감소해야 합니다. 평균적으로 다음과 같습니다.

- 복제본 수가 증가하면 **Pod**당 메모리(작업 집합) 사용량이 전반적으로 감소해야 합니다.
- 복제본 수가 감소하면 **Pod**별 메모리 사용량이 전반적으로 증가해야 합니다.

메모리 기반 자동 스케일링을 사용하기 전에 **OpenShift Container Platform** 웹 콘솔을 사용하여 애플리케이션의 메모리 동작을 확인하고 애플리케이션이 해당 요구 사항을 충족하는지 확인하십시오.

다음 예제에서는 **image-registry Deployment** 오브젝트에 대한 자동 스케일링을 보여줍니다. 초기 배

포에는 **Pod 3개**가 필요합니다. **HPA** 오브젝트는 최소 값을 **5**로 늘립니다. **Pod**의 **CPU** 사용량이 **75%**에 도달하면 **Pod**가 **7**로 증가합니다.

```
$ oc autoscale deployment/image-registry --min=5 --max=7 --cpu-percent=75
```

출력 예

```
horizontalpodautoscaler.autoscaling/image-registry autoscaled
```

minReplicas 가 **3**으로 설정된 **image-registry Deployment** 오브젝트의 샘플 **HPA**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: image-registry
  namespace: default
spec:
  maxReplicas: 7
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: image-registry
  targetCPUUtilizationPercentage: 75
status:
  currentReplicas: 5
  desiredReplicas: 0
```

1. 배포의 새 상태를 확인합니다.

```
$ oc get deployment image-registry
```

이제 배포에 **Pod 5개**가 있습니다.

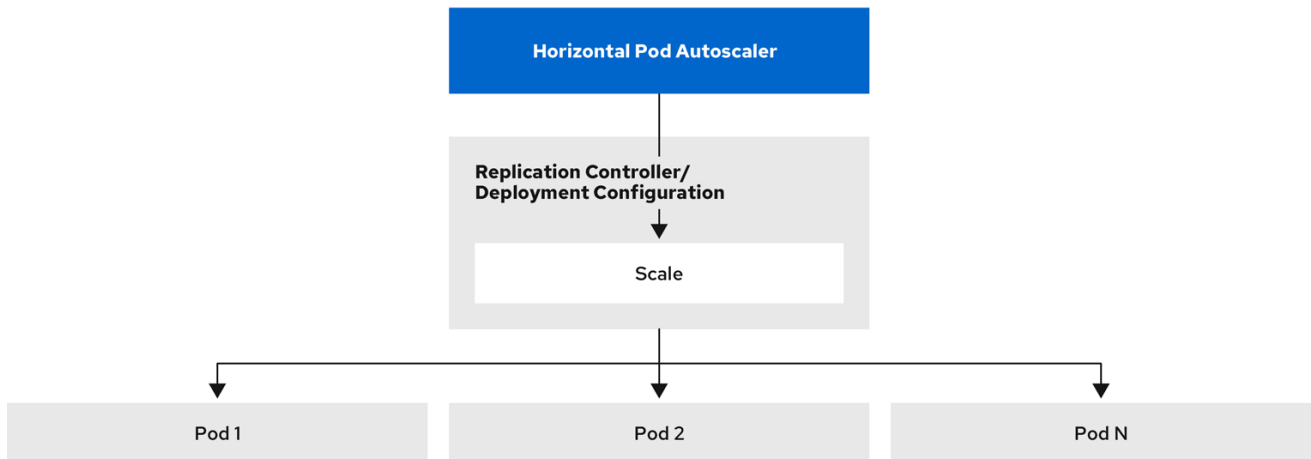
출력 예

NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
image-registry	1	5	5	config

2.4.2. HPA는 어떻게 작동합니까?

HPA(Horizontal Pod Autoscaler)는 **Pod** 자동 확장 개념을 확장합니다. **HPA**를 사용하면 부하 분산된 노드 그룹을 생성하고 관리할 수 있습니다. **HPA**는 지정된 **CPU** 또는 메모리 임계값이 교차하면 **Pod** 수를 자동으로 늘리거나 줄입니다.

그림 2.1. HPA의 높은 수준의 워크플로



223_OpenShift_0222

HPA는 **Kubernetes** 자동 스케일링 **API** 그룹의 **API** 리소스입니다. 자동 스케일러는 동기화 기간 동안 기본값이 15초인 컨트롤 루프로 작동합니다. 이 기간 동안 컨트롤러 관리자는 **HPA**의 **YAML** 파일에 정의된 **CPU**, 메모리 사용률 또는 둘 다를 쿼리합니다. 컨트롤러 관리자는 **HPA**에서 대상으로 하는 각 **Pod**에 대해 **CPU** 또는 메모리와 같은 **Pod**별 리소스 지표 **API**에서 사용률 지표를 가져옵니다.

사용률 값 타겟이 설정된 경우 컨트롤러는 사용률 값을 각 **Pod**의 컨테이너에 동일한 리소스 요청의 백분율로 계산합니다. 그러면 컨트롤러가 모든 대상 **Pod**에서 평균 사용률을 사용하고 원하는 복제본 수를 확장하는 데 사용되는 비율을 생성합니다. **HPA**는 지표 서버에서 제공하는 **metrics.k8s.io** 에서 메트릭을 가져오도록 구성됩니다. 지표 평가의 동적 특성으로 인해 복제본 그룹의 확장 중에 복제본 수를 변동할 수 있습니다.



참고

HPA를 구현하려면 모든 대상 **Pod**에 해당 컨테이너에 리소스 요청이 설정되어 있어야 합니다.

2.4.3. 요청 및 제한 정보

스케줄러는 **Pod**에서 컨테이너에 지정하는 리소스 요청을 사용하여 **Pod**를 배치할 노드를 결정합니다. **kubelet**은 컨테이너에서 지정된 제한을 초과할 수 없도록 컨테이너에 지정하는 리소스 제한을 적용합니다. **kubelet**은 해당 컨테이너가 사용할 수 있도록 특별히 해당 시스템 리소스의 요청 양을 예약합니다.

리소스 메트릭을 사용하는 방법은 무엇입니까?

Pod 사양에서 **CPU** 및 메모리와 같은 리소스 요청을 지정해야 합니다. **HPA**는 이 사양을 사용하여 리소스 사용률을 확인한 다음 대상을 **up** 또는 **down**으로 확장합니다.

예를 들어 **HPA** 오브젝트는 다음 지표 소스를 사용합니다.

```
type: Resource
resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
```

이 예에서 **HPA**는 스케일링 대상에서 **Pod**의 평균 사용률을 **60%**로 유지합니다. 사용량은 현재 리소스 사용량과 **Pod**의 요청된 리소스 간의 비율입니다.

2.4.4. 모범 사례

모든 **Pod**에 리소스 요청이 구성되어 있어야 합니다.

HPA는 **OpenShift Container Platform** 클러스터에서 모니터링되는 **Pod**의 **CPU** 또는 메모리 사용률 값을 기반으로 스케일링 결정을 내립니다. 사용률 값은 각 **Pod**의 리소스 요청의 백분율로 계산됩니다. 누락된 리소스 요청 값은 **HPA**의 최적 성능에 영향을 미칠 수 있습니다.

누적 다운 기간을 설정합니다.

수평 **Pod** 자동 스케일링 중에 시간 간격 없이 이벤트를 빠르게 스케일링할 수 있습니다. 자주 발생하는 복제본 변동을 방지하기 위해 하이크 다운 기간을 설정합니다. **stabilizationWindowSeconds** 필드를 구성하여 연속 기간을 지정할 수 있습니다. 확장에 사용되는 메트릭이 계속 변동할 때 **Stabilization** 창은

복제본 수의 변동을 제한하는 데 사용됩니다. 자동 스케일링 알고리즘은 이 값을 사용하여 이전 원하는 상태를 유추하고 워크로드 스케일링을 원하지 않는 변경을 방지합니다.

예를 들어 `scaleDown` 필드에 대해 `stabilization` 값이 지정됩니다.

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
```

위의 예에서 지난 5분 동안 원하는 모든 상태가 고려됩니다. 이렇게 하면 롤링 최대값을 반복하고 확장 알고리즘이 Pod를 자주 제거하지 않도록 하여 동등한 Pod를 즉시 다시 생성할 수 있습니다.

2.4.4.1. 스케일링 정책

`autoscaling/v2` API를 사용하면 수평 Pod 자동 스케일러에 스케일링 정책을 추가할 수 있습니다. 스케일링 정책은 OpenShift Container Platform HPA(수평 Pod 자동 스케일러)에서 Pod를 스케일링하는 방법을 제어합니다. 스케일링 정책을 사용하면 지정된 기간에 스케일링할 특정 수 또는 특정 백분율을 설정하여 HPA에서 Pod를 확장 또는 축소하는 비율을 제한할 수 있습니다. 또한 메트릭이 계속 변동하는 경우 이전에 계산한 원하는 상태를 사용하여 스케일링을 제어하는 안정화 기간을 정의할 수 있습니다. 동일한 스케일링 방향(확장 또는 축소)에 대해 여러 정책을 생성하여 변경 정도에 따라 사용할 정책을 결정할 수 있습니다. 반복 시간을 지정하여 스케일링을 제한할 수도 있습니다. HPA는 반복 중 Pod를 스케일링한 다음 필요에 따라 추가 반복에서 스케일링을 수행합니다.

스케일링 정책이 포함된 HPA 오브젝트 샘플

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
  behavior:
    scaleDown: 1
    policies: 2
    - type: Pods 3
      value: 4 4
      periodSeconds: 60 5
    - type: Percent
      value: 10 6
      periodSeconds: 60
    selectPolicy: Min 7
    stabilizationWindowSeconds: 300 8
  scaleUp: 9
  policies:
```

```
- type: Pods
  value: 5 10
  periodSeconds: 70
- type: Percent
  value: 12 11
  periodSeconds: 80
selectPolicy: Max
stabilizationWindowSeconds: 0
```

...

1

스케일링 정책의 방향, 즉 **scaleDown** 또는 **scaleUp**을 지정합니다. 이 예제에서는 축소 정책을 생성합니다.

2

스케일링 정책을 정의합니다.

3

정책이 각 반복에서 특정 **Pod** 수 또는 **Pod** 백분율로 스케일링하는지의 여부를 결정합니다. 기본값은 **pod**입니다.

4

각 반복 중에 **Pod** 수 또는 **Pod** 백분율 중 하나의 스케일링 양을 제한합니다. **Pod** 수에 따라 축소할 기본값은 없습니다.

5

스케일링 반복의 길이를 결정합니다. 기본값은 15초입니다.

6

백분율로 된 축소 기본값은 100%입니다.

7

여러 정책이 정의된 경우 먼저 사용할 정책을 결정합니다. 가장 많은 변경을 허용하는 정책을 사용하려면 **Max**를 지정하고, 최소 변경을 허용하는 정책을 사용하려면 **Min**을 지정합니다. **HPA**에서 해당 정책 방향으로 스케일링하지 않도록 하려면 **Disabled**를 지정합니다. 기본값은 **Max**입니다.

8

HPA에서 원하는 상태를 검토해야 하는 기간을 결정합니다. 기본값은 0입니다.

9

이 예제에서는 확장 정책을 생성합니다.

10

Pod 수에 따라 확장되는 양을 제한합니다. Pod 수 확장 기본값은 4%입니다.

11

Pod 백분율에 따라 확장되는 양을 제한합니다. 백분율로 된 확장 기본값은 100%입니다.

축소 정책의 예

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
  ...
  minReplicas: 20
  ...
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
    policies:
      - type: Pods
        value: 4
        periodSeconds: 30
      - type: Percent
        value: 10
        periodSeconds: 60
    selectPolicy: Max
  scaleUp:
    selectPolicy: Disabled
```

이 예제에서 Pod 수가 40개를 초과하면 `selectPolicy`에서 요구하는 대로 해당 정책으로 인해 상당한 변경이 발생하므로 축소에 백분율 기반 정책이 사용됩니다.

Pod 복제본이 80개 있는 경우 HPA는 첫 번째 반복에서 1분(`periodSeconds: 60`)에 걸쳐 (`type: Percent` 및 `value: 10` 매개변수에 따라) Pod 80개 중 10%에 해당하는 8개의 Pod를 줄입니다. 다음 반복에서는 Pod가 72개입니다. HPA는 나머지 Pod의 10%를 계산한 7.2개를 8개로 올림하여 Pod 8개를 축소합니다. 이후 반복할 때마다 나머지 Pod 수에 따라 스케일링할 Pod 수가 다시 계산됩니다. Pod 수가 40개 미만으로 줄어들면 Pod 기반 숫자가 백분율 기반 숫자보다 크기 때문에 Pod 기반 정책이 적용됩니다. HPA는 20개의 복제본(`minReplicas`)이 남아 있을 때까지 30초(`periodSeconds: 30`)에 걸쳐 한 번에 Pod를 4개씩 줄입니다(`type: Pods` 및 `value: 4`).

`selectPolicy: Disabled` 매개변수를 사용하면 HPA에서 Pod를 확장하지 못합니다. 필요한 경우 복제본 세트 또는 배포 세트의 복제본 수를 조정하여 수동으로 확장할 수 있습니다.

설정되어 있는 경우 `oc edit` 명령을 사용하여 스케일링 정책을 확인할 수 있습니다.

```
$ oc edit hpa hpa-resource-metrics-memory
```

출력 예

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  annotations:
    autoscaling.alpha.kubernetes.io/behavior:\
'{"ScaleUp":{"StabilizationWindowSeconds":0,"SelectPolicy":"Max","Policies":\
[{"Type":"Pods","Value":4,"PeriodSeconds":15},\
{"Type":"Percent","Value":100,"PeriodSeconds":15}]},\
"ScaleDown":{"StabilizationWindowSeconds":300,"SelectPolicy":"Min","Policies":\
[{"Type":"Pods","Value":4,"PeriodSeconds":60},\
{"Type":"Percent","Value":10,"PeriodSeconds":60}]}'
...
```

2.4.5. 웹 콘솔을 사용하여 수평 Pod 자동 스케일러 생성

웹 콘솔에서 `Deployment` 또는 `DeploymentConfig` 오브젝트에서 실행할 최소 및 최대 Pod 수를 지정하는 HPA(수평 Pod 자동 스케일러)를 생성할 수 있습니다. Pod에서 대상으로 하는 CPU 또는 메모리 사용량을 정의할 수도 있습니다.



참고

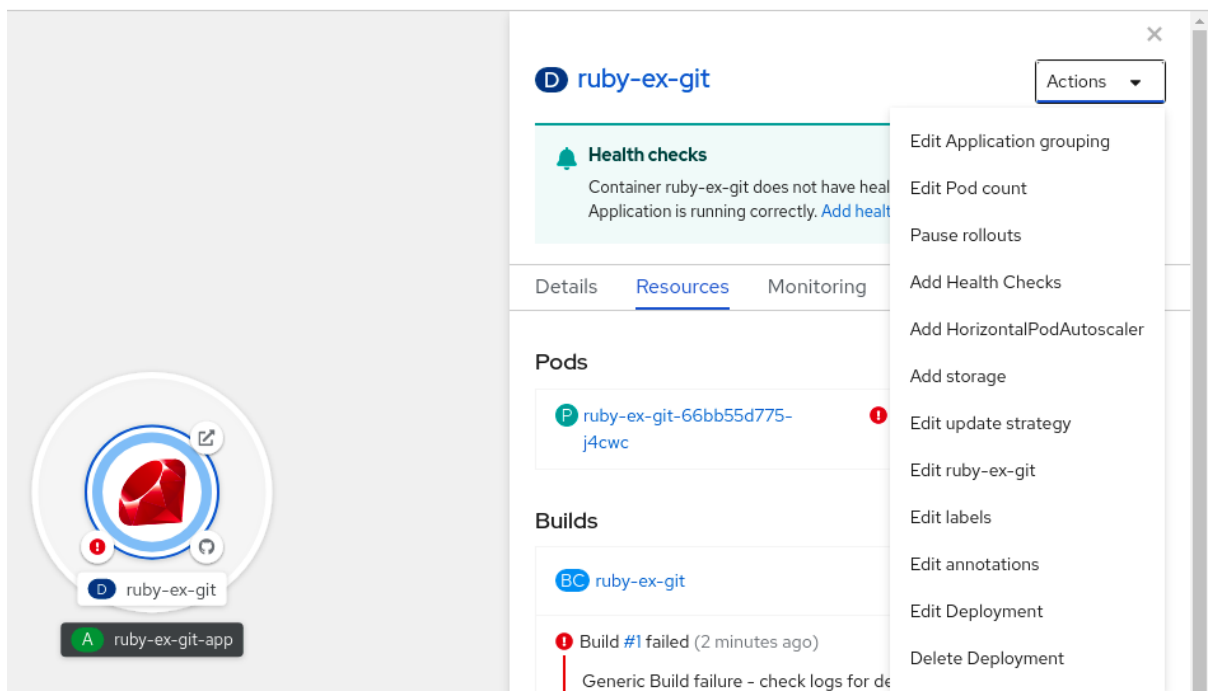
HPA는 Operator 지원 서비스, Knative 서비스 또는 Helm 차트의 일부인 배포에 추가할 수 없습니다.

절차

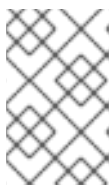
웹 콘솔에서 HPA를 생성하려면 다음을 수행합니다.

1. 토폴로지 보기에서 노드를 클릭하여 측면 창을 표시합니다.
2. 작업 드롭다운 목록에서 **HorizontalPodAutoscaler** 추가를 선택하여 **HorizontalPodAutoscaler** 추가 양식을 엽니다.

그림 2.2. Add HorizontalPodAutoscaler



3. **HorizontalPodAutoscaler** 추가 양식에서 이름, 최소 및 최대 Pod 제한, CPU 및 메모리 사용량을 정의하고 저장을 클릭합니다.



참고

CPU 및 메모리 사용량에 대한 값이 없는 경우 경고가 표시됩니다.

웹 콘솔에서 **HPA**를 편집하려면 다음을 수행합니다.

1. 토폴로지 보기에서 노드를 클릭하여 측면 창을 표시합니다.
2. 작업 드롭다운 목록에서 **HorizontalPodAutoscaler** 편집을 선택하여 **Horizontal Pod Autoscaler** 편집 양식을 엽니다.
3. **Horizontal Pod Autoscaler** 편집 양식에서 최소 및 최대 **Pod** 제한과 **CPU** 및 메모리 사용량을 편집한 다음 저장을 클릭합니다.



참고

웹 콘솔에서 수평 **Pod** 자동 스케일러를 생성하거나 편집하는 동안 양식 보기에서 **YAML** 보기로 전환할 수 있습니다.

웹 콘솔에서 **HPA**를 제거하려면 다음을 수행합니다.

1. 토폴로지 보기에서 노드를 클릭하여 측면 창을 표시합니다.
2. 작업 드롭다운 목록에서 **HorizontalPodAutoscaler** 제거를 선택합니다.
3. 확인 팝업 창에서 제거를 클릭하여 **HPA**를 제거합니다.

2.4.6. CLI를 사용하여 CPU 사용률에 대한 수평 Pod 자동 스케일러 생성

OpenShift Container Platform CLI를 사용하면 **HPA**(수평 Pod 자동 스케일러)를 생성하여 기존 **Deployment,DeploymentConfig,ReplicaSet,ReplicationController** 또는 **StatefulSet** 오브젝트를 자동으로 확장할 수 있습니다. **HPA**는 해당 오브젝트와 연결된 **Pod**를 스케일링하여 지정한 **CPU** 사용량을 유지합니다.



참고

다른 오브젝트에서 제공하는 특정 기능이나 동작이 필요하지 않은 경우 **Deployment** 오브젝트 또는 **ReplicaSet** 오브젝트를 사용하는 것이 좋습니다.

HPA는 최소 및 최대 개수 사이에서 복제본 수를 늘리거나 줄여 전체 **Pod**에서 지정된 **CPU** 사용률을 유지합니다.

CPU 사용률을 자동 스케일링할 때는 **oc autoscale** 명령을 사용하여 언제든지 실행하려는 최소 및 최대 **Pod** 수와 **Pod**에서 목표로 하는 평균 **CPU** 사용률을 지정할 수 있습니다. 최솟값을 지정하지 않으면 **Pod**에 **OpenShift Container Platform** 서버의 기본값이 지정됩니다.

특정 **CPU** 값을 자동 스케일링하려면 대상 **CPU** 및 **Pod** 제한을 사용하여 **HorizontalPodAutoscaler** 오브젝트를 생성합니다.

사전 요구 사항

수평 **Pod** 자동 스케일러를 사용하려면 클러스터 관리자가 클러스터 메트릭을 올바르게 구성해야 합니다. **oc describe PodMetrics <pod-name>** 명령을 사용하여 메트릭이 구성되어 있는지 확인할 수 있습니다. 메트릭이 구성된 경우 출력이 다음과 유사하게 표시되고 **Usage**에 **Cpu** 및 **Memory**가 표시됩니다.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

출력 예

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link:          /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-
scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

```

Timestamp:    2019-05-23T18:47:56Z
Window:      1m0s
Events:      <none>

```

프로세스

CPU 사용률에 대한 수평 Pod 자동 스케일러를 생성하려면 다음을 수행합니다.

1.

다음 중 하나를 수행합니다.

-

CPU 사용률의 백분율에 따라 스케일링하려면 기존 오브젝트에 대한 **HorizontalPodAutoscaler** 오브젝트를 생성합니다.

```

$ oc autoscale <object_type>/<name> \ 1
--min <number> \ 2
--max <number> \ 3
--cpu-percent=<percent> 4

```

1

자동 스케일링할 오브젝트의 유형 및 이름을 지정합니다. 오브젝트가 존재하고 **Deployment** **DeploymentConfig/dc**, **ReplicaSet/rs**, **ReplicationController/rc** 또는 **StatefulSet** 여야 합니다.

2

필요한 경우 축소 시 최소 복제본 수를 지정합니다.

3

확장 시 최대 복제본 수를 지정합니다.

4

요청된 CPU의 백분율로 표시되는 모든 Pod의 목표 평균 CPU 사용량을 지정합니다. 지정하지 않거나 음수가 아닌 경우 기본 자동 스케일링 정책이 사용됩니다.

예를 들어 다음 명령은 **image-registry Deployment** 오브젝트에 대한 자동 스케일링을 보여줍니다. 초기 배포에는 Pod 3개가 필요합니다. HPA 오브젝트는 최소 값을 5로 늘립니다.

다. Pod의 CPU 사용량이 75%에 도달하면 Pod가 7으로 증가합니다.

```
$ oc autoscale deployment/image-registry --min=5 --max=7 --cpu-percent=75
```

- 특정 CPU 값을 스케일링하려면 기존 오브젝트에 대해 다음과 유사한 YAML 파일을 생성합니다.

a.

다음과 유사한 YAML 파일을 생성합니다.

```
apiVersion: autoscaling/v2 1
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
    kind: Deployment 4
    name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
  - type: Resource
    resource:
      name: cpu 9
      target:
        type: AverageValue 10
        averageValue: 500m 11
```

1

autoscaling/v2 API를 사용합니다.

2

이 수평 Pod 자동 스케일러 오브젝트의 이름을 지정합니다.

3

스케일링할 오브젝트의 API 버전을 지정합니다.

o

Deployment, ReplicaSet, Statefulset 오브젝트의 경우 apps/v1 을 사용합니다.

- **ReplicationController** 의 경우 **v1** 을 사용합니다.
- **DeploymentConfig** 의 경우 **apps.openshift.io/v1** 를 사용합니다.

4

오브젝트 유형을 지정합니다. 오브젝트는 **Deployment,DeploymentConfig/dc,ReplicaSet/rs,ReplicationController/rc** 또는 **StatefulSet** 여야 합니다.

5

스케일링할 오브젝트의 이름을 지정합니다. 오브젝트가 있어야 합니다.

6

축소 시 최소 복제본 수를 지정합니다.

7

확장 시 최대 복제본 수를 지정합니다.

8

메모리 사용률에 **metrics** 매개변수를 사용합니다.

9

CPU 사용률에 **cpu**를 지정합니다.

10

AverageValue로 설정합니다.

11

대상 CPU 값을 사용하여 **averageValue**로 설정합니다.

b.

수평 Pod 자동 스케일러를 생성합니다.

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

2.

수평 Pod 자동 스케일러가 생성되었는지 확인합니다.

```
$ oc get hpa cpu-autoscale
```

출력 예

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
cpu-autoscale	Deployment/example	173m/500m	1	10	1
					20m

2.4.7. CLI를 사용하여 메모리 사용률에 대한 수평 Pod 자동 스케일러 오브젝트 생성

OpenShift Container Platform CLI를 사용하면 HPA(수평 Pod 자동 스케일러)를 생성하여 기존 Deployment, DeploymentConfig, ReplicaSet, ReplicationController 또는 StatefulSet 오브젝트를 자동으로 확장할 수 있습니다. HPA는 해당 오브젝트에 연결된 Pod를 스케일링하여 지정하는 평균 메모리 사용률(직접적인 값 또는 요청된 메모리의 백분율)을 유지합니다.



참고

다른 오브젝트에서 제공하는 특정 기능이나 동작이 필요하지 않은 경우 Deployment 오브젝트 또는 ReplicaSet 오브젝트를 사용하는 것이 좋습니다.

HPA는 최소 및 최대 개수 사이에서 복제본 수를 늘리거나 줄여 전체 Pod에서 지정된 메모리 사용률을 유지합니다.

메모리 사용률의 경우 최소 및 최대 Pod 수와 Pod에서 목표로 해야 하는 평균 메모리 사용률을 지정할 수 있습니다. 최솟값을 지정하지 않으면 Pod에 OpenShift Container Platform 서버의 기본값이 지정됩니다.

사전 요구 사항

수평 Pod 자동 스케일러를 사용하려면 클러스터 관리자가 클러스터 메트릭을 올바르게 구성해야 합니다. `oc describe PodMetrics <pod-name>` 명령을 사용하여 메트릭이 구성되어 있는지 확인할 수 있습니다. 메트릭이 구성된 경우 출력이 다음과 유사하게 표시되고 Usage에 Cpu 및 Memory가 표시됩니다.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-129-223.compute.internal -n openshift-kube-scheduler
```

출력 예

```
Name:      openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Cpu: 0
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2020-02-14T22:21:14Z
  Self Link:          /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-129-223.compute.internal
  Timestamp:          2020-02-14T22:21:14Z
  Window:             5m0s
  Events:             <none>
```

프로세스

메모리 사용률에 대한 수평 Pod 자동 스케일러를 생성하려면 다음을 수행합니다.

1. 다음 중 하나에 대한 **YAML** 파일을 생성합니다.
 - 특정 메모리 값을 스케일링하려면 기존 오브젝트에 대해 다음과 유사한 **HorizontalPodAutoscaler** 오브젝트를 생성합니다.

```
apiVersion: autoscaling/v2 1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory 2
  namespace: default
spec:
```



```

scaleTargetRef:
  apiVersion: apps/v1 3
  kind: Deployment 4
  name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
- type: Resource
  resource:
    name: memory 9
    target:
      type: AverageValue 10
      averageValue: 500Mi 11
behavior: 12
scaleDown:
  stabilizationWindowSeconds: 300
  policies:
    - type: Pods
      value: 4
      periodSeconds: 60
    - type: Percent
      value: 10
      periodSeconds: 60
  selectPolicy: Max

```

1

autoscaling/v2 API를 사용합니다.

2

이 수평 Pod 자동 스케일러 오브젝트의 이름을 지정합니다.

3

스케일링할 오브젝트의 API 버전을 지정합니다.

- Deployment, ReplicaSet, Statefulset 오브젝트의 경우 apps/v1 을 사용합니다.
- ReplicationController 의 경우 v1 을 사용합니다.
- DeploymentConfig 의 경우 apps.openshift.io/v1 를 사용합니다.

4

5

스케일링할 오브젝트의 이름을 지정합니다. 오브젝트가 있어야 합니다.

6

축소 시 최소 복제본 수를 지정합니다.

7

확장 시 최대 복제본 수를 지정합니다.

8

메모리 사용률에 **metrics** 매개변수를 사용합니다.

9

메모리 사용률에 대한 메모리를 지정합니다.

10

유형을 **AverageValue**로 설정합니다.

11

averageValue 및 특정 메모리 값을 지정합니다.

12

선택 사항: 스케일링 정책을 지정하여 확장 또는 축소비율을 제어합니다.

•

백분율로 스케일링하려면 기존 오브젝트에 대해 다음과 유사한 **HorizontalPodAutoscaler** 오브젝트를 생성합니다.

```
apiVersion: autoscaling/v2 1
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
```

```

kind: Deployment 4
name: example 5
minReplicas: 1 6
maxReplicas: 10 7
metrics: 8
- type: Resource
  resource:
    name: memory 9
    target:
      type: Utilization 10
      averageUtilization: 50 11
behavior: 12
scaleUp:
  stabilizationWindowSeconds: 180
policies:
- type: Pods
  value: 6
  periodSeconds: 120
- type: Percent
  value: 10
  periodSeconds: 120
selectPolicy: Max

```

1

autoscaling/v2 API를 사용합니다.

2

이 수평 Pod 자동 스케일러 오브젝트의 이름을 지정합니다.

3

스케일링할 오브젝트의 API 버전을 지정합니다.

- ReplicationController의 경우 v1 을 사용합니다.
- DeploymentConfig의 경우 apps.openshift.io/v1 를 사용합니다.
- Deployment, ReplicaSet, Statefulset 오브젝트의 경우 apps/v1 을 사용합니다.

4

오브젝트 유형을 지정합니다. 오브젝트는 Deployment,DeploymentConfig,ReplicaSet,ReplicationController 또는 StatefulSet

여야 합니다.

5

스케일링할 오브젝트의 이름을 지정합니다. 오브젝트가 있어야 합니다.

6

축소 시 최소 복제본 수를 지정합니다.

7

확장 시 최대 복제본 수를 지정합니다.

8

메모리 사용률에 **metrics** 매개변수를 사용합니다.

9

메모리 사용률에 대한 메모리를 지정합니다.

10

Utilization으로 설정합니다.

11

averageUtilization 및 전체 **Pod**에 대한 대상 평균 메모리 사용률(요청 메모리의 백분율로 표시)을 지정합니다. 대상 **Pod**에 메모리 요청이 구성되어 있어야 합니다.

12

선택 사항: 스케일링 정책을 지정하여 확장 또는 축소비율을 제어합니다.

2.

수평 **Pod** 자동 스케일러를 생성합니다.

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

예를 들면 다음과 같습니다.

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

출력 예

```
horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

3.

수평 Pod 자동 스케일러가 생성되었는지 확인합니다.

```
$ oc get hpa hpa-resource-metrics-memory
```

출력 예

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
REPLICAS AGE				
hpa-resource-metrics-memory	Deployment/example	2441216/500Mi	1	10
20m				1

```
$ oc describe hpa hpa-resource-metrics-memory
```

출력 예

```
Name:          hpa-resource-metrics-memory
Namespace:     default
Labels:        <none>
Annotations:   <none>
CreationTimestamp:  Wed, 04 Mar 2020 16:31:37 +0530
Reference:     Deployment/example
Metrics:       ( current / target )
  resource memory on pods: 2441216 / 500Mi
Min replicas:   1
Max replicas:   10
ReplicationController pods: 1 current / 1 desired
Conditions:
  Type           Status Reason          Message
  ----           -
  AbleToScale    True  ReadyForNewScale  recommended size matches current size
  ScalingActive  True  ValidMetricFound  the HPA was able to successfully calculate
  a replica count from memory resource
```

```

ScalingLimited False DesiredWithinRange the desired count is within the
acceptable range
Events:
  Type Reason Age From Message
  ---- -
Normal SuccessfulRescale 6m34s horizontal-pod-autoscaler New size:
1; reason: All metrics below target
    
```

2.4.8. CLI를 사용하여 수평 Pod 자동 스케일러 상태 조건 이해

일련의 상태 조건을 사용하여 HPA(수평 Pod 자동 스케일러)에서 스케일링할 수 있는지 그리고 HPA가 현재 제한되어 있는지의 여부를 결정할 수 있습니다.

HPA 상태 조건은 v2 버전의 자동 스케일링 API에서 사용할 수 있습니다.

HPA는 다음과 같은 상태 조건을 통해 응답합니다.

- **AbleToScale** 상태는 HPA에서 메트릭을 가져오고 업데이트할 수 있는지의 여부 및 백오프 관련 상태로 스케일링을 방지할 수 있는지의 여부를 나타냅니다.
 - **True** 조건은 스케일링이 허용되었음을 나타냅니다.
 - **False** 조건은 지정된 이유로 스케일링이 허용되지 않음을 나타냅니다.
- **ScalingActive** 조건은 HPA가 활성화되어 있고(예: 대상의 복제본 수가 0이 아님) 원하는 메트릭을 계산할 수 있는지의 여부를 나타냅니다.
 - **True** 조건은 메트릭이 제대로 작동함을 나타냅니다.
 - **False** 조건은 일반적으로 메트릭을 가져오는 데 문제가 있음을 나타냅니다.
- **ScalingLimited** 조건은 원하는 스케일링이 수평 Pod 자동 스케일러의 최댓값 또는 최솟값으로 제한되었음을 나타냅니다.

- **True** 조건은 스케일링을 위해 최소 또는 최대 복제본 수를 늘리거나 줄여야 함을 나타냅니다.
- **False** 조건은 요청된 스케일링이 허용됨을 나타냅니다.

```
$ oc describe hpa cm-test
```

출력 예

```
Name: cm-test
Namespace: prom
Labels: <none>
Annotations: <none>
CreationTimestamp: Fri, 16 Jun 2017 18:09:22 +0000
Reference: ReplicationController/cm-test
Metrics: ( current / target )
"http_requests" on pods: 66m / 500m
Min replicas: 1
Max replicas: 4
ReplicationController pods: 1 current / 1 desired
Conditions: 1
  Type          Status Reason          Message
  ----          -
  AbleToScale   True   ReadyForNewScale the last scale time was sufficiently
old as to warrant a new scale
  ScalingActive True   ValidMetricFound the HPA was able to successfully
calculate a replica count from pods metric http_request
  ScalingLimited False  DesiredWithinRange the desired replica count is within
the acceptable range
Events:
```

1

수평 Pod 자동 스케일러의 상태 메시지입니다.

다음은 스케일링할 수 없는 Pod의 예입니다.

출력 예

Conditions:

Type	Status	Reason	Message
AbleToScale	False	FailedGetScale	the HPA controller was unable to get the target's current scale: no matches for kind "ReplicationController" in group "apps"

Events:

Type	Reason	Age	From	Message
Warning	FailedGetScale	6s (x3 over 36s)	horizontal-pod-autoscaler	no matches for kind "ReplicationController" in group "apps"

다음은 스케일링에 필요한 메트릭을 가져올 수 없는 Pod의 예입니다.

출력 예

Conditions:

Type	Status	Reason	Message
AbleToScale	True	SucceededGetScale	the HPA controller was able to get the target's current scale
ScalingActive	False	FailedGetResourceMetric	the HPA was unable to compute the replica count: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API

다음은 요청된 자동 스케일링이 필요한 최솟값보다 적은 Pod의 예입니다.

출력 예

Conditions:

Type	Status	Reason	Message
AbleToScale	True	ReadyForNewScale	the last scale time was sufficiently old as to warrant a new scale
ScalingActive	True	ValidMetricFound	the HPA was able to successfully calculate a


```

replica count from pods metric http_request
ScalingLimited False DesiredWithinRange the desired replica count is within the
acceptable range

```

2.4.8.1. CLI를 사용하여 수평 Pod 자동 스케일러 상태 조건 보기

HPA(수평 Pod 자동 스케일러)를 통해 Pod에 설정된 상태 조건을 볼 수 있습니다.



참고

수평 Pod 자동 스케일러 상태 조건은 v2 버전의 자동 스케일링 API에서 사용할 수 있습니다.

사전 요구 사항

수평 Pod 자동 스케일러를 사용하려면 클러스터 관리자가 클러스터 메트릭을 올바르게 구성해야 합니다. `oc describe PodMetrics <pod-name>` 명령을 사용하여 메트릭이 구성되어 있는지 확인할 수 있습니다. 메트릭이 구성된 경우 출력이 다음과 유사하게 표시되고 Usage에 Cpu 및 Memory가 표시됩니다.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

출력 예

```

Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link:          /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-
scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal

```

```
Timestamp:      2019-05-23T18:47:56Z
Window:        1m0s
Events:        <none>
```

프로세스

Pod의 상태 조건을 보려면 **Pod** 이름과 함께 다음 명령을 사용합니다.

```
$ oc describe hpa <pod-name>
```

예를 들면 다음과 같습니다.

```
$ oc describe hpa cm-test
```

상태가 출력의 **Conditions** 필드에 나타납니다.

출력 예

```
Name:          cm-test
Namespace:    prom
Labels:       <none>
Annotations:  <none>
CreationTimestamp:  Fri, 16 Jun 2017 18:09:22 +0000
Reference:    ReplicationController/cm-test
Metrics:      ( current / target )
"http_requests" on pods:  66m / 500m
Min replicas:  1
Max replicas:  4
ReplicationController pods:  1 current / 1 desired
Conditions:  1
  Type          Status Reason          Message
  ----          -
  AbleToScale   True   ReadyForNewScale the last scale time was sufficiently old as to
warrant a new scale
  ScalingActive True   ValidMetricFound the HPA was able to successfully calculate a
replica count from pods metric http_request
  ScalingLimited False  DesiredWithinRange the desired replica count is within the
acceptable range
```

2.4.9. 추가 리소스

- 복제 컨트롤러 및 배포 컨트롤러에 대한 자세한 내용은 배포 및 배포 구성 이해 를 참조하십시오.
- HPA 사용 방법에 대한 예제는 Memory Utilization에 따라 Quarkus 애플리케이션 자동 스케일링 을 참조하십시오.

2.5. 수직 POD 자동 스케일러를 사용하여 POD 리소스 수준 자동 조정

OpenShift Container Platform VPA(Vertical Pod Autoscaler Operator)는 Pod의 컨테이너에 대한 과거 및 현재의 CPU 및 메모리 리소스를 자동으로 검토한 후 확인한 사용량 값에 따라 리소스 제한 및 요청을 업데이트할 수 있습니다. VPA는 개별 CR(사용자 정의 리소스)을 사용하여 Deployment, Deployment Config, StatefulSet, Job, DaemonSet, ReplicaSet 또는 ReplicationController와 같은 워크로드 오브젝트와 연결된 모든 Pod를 프로젝트에서 업데이트합니다.

VPA를 사용하면 Pod의 최적 CPU 및 메모리 사용량을 이해하고 Pod의 라이프사이클 내내 Pod 리소스를 자동으로 유지 관리할 수 있습니다.

2.5.1. Vertical Pod Autoscaler Operator 정보

VPA(Vertical Pod Autoscaler Operator)는 API 리소스 및 CR(사용자 정의 리소스)로 구현됩니다. CR에 따라 Vertical Pod Autoscaler Operator에서 데몬 세트, 복제 컨트롤러 등과 같은 특정 워크로드 오브젝트와 관련된 Pod에서 수행해야 하는 작업이 프로젝트에서 결정됩니다.

기본 권장 사항을 사용하거나 자체 알고리즘에 따라 자동 스케일링을 대신 대신 사용할 수 있습니다.

기본 권장 사항은 해당 Pod의 컨테이너에 대한 과거 및 현재 CPU 및 메모리 사용량을 자동으로 계산하고 이 데이터를 사용하여 최적화된 리소스 제한 및 요청을 확인하여 이러한 Pod가 항상 효율적으로 작동하는지 확인합니다. 예를 들어 기본 권장 방법은 사용 중인 리소스보다 더 많은 리소스를 요청하는 Pod에 대한 리소스가 감소하고 충분하지 않은 Pod에 리소스를 늘립니다.

그런 다음 VPA는 애플리케이션이 다운타임 없이 요청을 계속 제공할 수 있도록 이러한 권장 사항과 일치하지 않는 모든 Pod를 한 번에 하나씩 자동으로 삭제합니다. 그런 다음 워크로드 오브젝트는 원래 리소스 제한 및 요청을 사용하여 Pod를 재배포합니다. VPA는 변경 승인 Webhook를 사용하여 Pod가 노드에 승인되기 전에 최적화된 리소스 제한 및 요청을 사용하여 Pod를 업데이트합니다. VPA에서 Pod를 삭제하지 않으려면 필요에 따라 VPA 리소스 제한 및 요청 및 수동으로 Pod를 업데이트할 수 있습니다.



참고

기본적으로 워크로드 오브젝트는 VPA에서 Pod를 자동으로 삭제할 수 있도록 최소 두 개의 복제본을 지정해야 합니다. 이 최소보다 적은 복제본을 지정하는 워크로드 오브젝트는 삭제되지 않습니다. 이러한 Pod를 수동으로 삭제하면 워크로드 오브젝트에서 Pod를 재배포할 때 VPA는 권장 사항으로 새 Pod를 업데이트합니다. **VPA 최소 값 변경과 같이 VerticalPodAutoscalerController** 오브젝트를 수정하여 이 최소값을 변경할 수 있습니다.

예를 들어 CPU의 50%를 사용하면서 10%만 요청하는 Pod가 있는 경우, VPA는 요청하는 것보다 더 많은 CPU를 사용하고 있는 것으로 판단하고 Pod를 삭제합니다. 복제본 세트와 같은 워크로드 오브젝트는 Pod를 재시작하고 VPA는 권장 리소스로 새 Pod를 업데이트합니다.

개발자의 경우 VPA를 사용하면 각 Pod에 적절한 리소스를 제공하도록 노드에 Pod를 예약하여 수요가 많은 기간에도 Pod가 유지되도록 할 수 있습니다.

관리자는 VPA를 사용하여 Pod에서 필요 이상의 CPU 리소스를 예약하지 않도록 클러스터 리소스를 더 효율적으로 활용할 수 있습니다. VPA는 워크로드에서 실제로 사용 중인 리소스를 모니터링하고 다른 워크로드에서 용량을 사용할 수 있도록 리소스 요구 사항을 조정합니다. 또한 VPA는 초기 컨테이너 구성에 지정된 제한 및 요청 간 비율도 유지합니다.



참고

VPA 실행을 중지하거나 클러스터에서 특정 VPA CR을 삭제하는 경우 VPA에서 이미 수정한 Pod에 대한 리소스 요청은 변경되지 않습니다. 새 Pod에서는 모두 VPA에서 설정한 이전의 권장 사항 대신 워크로드 오브젝트에 정의된 리소스를 가져옵니다.

2.5.2. Vertical Pod Autoscaler Operator 설치

OpenShift Container Platform 웹 콘솔을 사용하여 VPA(Vertical Pod Autoscaler Operator)를 설치할 수 있습니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.
2. 사용 가능한 **Operator** 목록에서 **VerticalPodAutoscaler**를 선택한 다음 설치를 클릭합니다.
3. **Operator** 설치 페이지에서 **Operator** 권장 네임스페이스 옵션이 선택되어 있는지 확인합니다.

다. 그러면 필수 **openshift-vertical-pod-autoscaler** 네임스페이스에 **Operator**가 설치됩니다. 해당 네임스페이스가 존재하지 않는 경우 자동으로 생성됩니다.

4. 설치를 클릭합니다.

Verification

1. **VPA Operator** 구성 요소를 나열하여 설치를 확인합니다.
 - a. 워크로드 → **Pod**로 이동합니다.
 - b. 드롭다운 메뉴에서 **openshift-vertical-pod-autoscaler** 프로젝트를 선택하고 **Pod 4개**가 실행되고 있는지 확인합니다.
 - c. 워크로드 → 배포로 이동하여 배포 4개가 실행되고 있는지 확인합니다.
2. 선택 사항: 다음 명령을 사용하여 **OpenShift Container Platform CLI**에서 설치를 확인합니다.

```
$ oc get all -n openshift-vertical-pod-autoscaler
```

출력에는 **Pod 4개**와 **배포 4개**가 표시됩니다.

출력 예

```

NAME                                READY STATUS RESTARTS AGE
pod/vertical-pod-autoscaler-operator-85b4569c47-2gmhc 1/1 Running 0 3m13s
pod/vpa-admission-plugin-default-67644fc87f-xq7k9    1/1 Running 0 2m56s
pod/vpa-recommender-default-7c54764b59-8gckt        1/1 Running 0 2m56s
pod/vpa-updater-default-7f6cc87858-47vw9           1/1 Running 0 2m56s

NAME          TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S) AGE
service/vpa-webhook ClusterIP  172.30.53.206 <none>      443/TCP 2m56s

NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/vertical-pod-autoscaler-operator 1/1 1 1 3m13s
deployment.apps/vpa-admission-plugin-default    1/1 1 1 2m56s

```

deployment.apps/vpa-recommender-default	1/1	1	1	2m56s
deployment.apps/vpa-updater-default	1/1	1	1	2m56s
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/vertical-pod-autoscaler-operator-85b4569c47	1	1	1	3m13s
replicaset.apps/vpa-admission-plugin-default-67644fc87f	1	1	1	2m56s
replicaset.apps/vpa-recommender-default-7c54764b59	1	1	1	2m56s
replicaset.apps/vpa-updater-default-7f6cc87858	1	1	1	2m56s

2.5.3. Vertical Pod Autoscaler Operator 사용 정보

VPA(Vertical Pod Autoscaler Operator)를 사용하려면 클러스터에서 워크로드 오브젝트에 대한 VPA CR(사용자 정의 리소스)을 생성합니다. VPA는 해당 워크로드 오브젝트와 연결된 Pod에 가장 적합한 CPU 및 메모리 리소스를 확인하고 적용합니다. 배포, 상태 저장 세트, 작업, 데몬 세트, 복제본 세트 또는 복제 컨트롤러 워크로드 오브젝트에 VPA를 사용할 수 있습니다. VPA CR은 모니터링할 Pod와 동일한 프로젝트에 있어야 합니다.

VPA CR을 사용하여 워크로드 오브젝트를 연결하고 VPA가 작동하는 모드를 지정합니다.

- Auto** 및 **Recreate** 모드는 Pod 수명 동안 VPA CPU 및 메모리 권장 사항을 자동으로 적용합니다. VPA는 권장 사항과 일치하지 않는 프로젝트의 모든 Pod를 삭제합니다. 워크로드 오브젝트에서 재배포하면 VPA는 새 Pod를 권장 사항으로 업데이트합니다.
- Initial** 모드는 Pod 생성 시에만 VPA 권장 사항을 자동으로 적용합니다.
- Off** 모드는 권장되는 리소스 제한 및 요청만 제공하며 권장 사항을 수동으로 적용할 수 있습니다. off 모드에서는 Pod를 업데이트하지 않습니다.

CR을 사용하여 VPA 평가 및 업데이트에서 특정 컨테이너를 옵트아웃할 수도 있습니다.

예를 들어 Pod에 다음과 같은 제한 및 요청이 있습니다.

```
resources:
  limits:
    cpu: 1
    memory: 500Mi
```

```
requests:
  cpu: 500m
  memory: 100Mi
```

auto로 설정된 **VPA**를 생성하면 **VPA**에서 리소스 사용량을 확인하고 **Pod**를 삭제합니다. 재배포되면 **Pod**는 새 리소스 제한 및 요청을 사용합니다.

```
resources:
  limits:
    cpu: 50m
    memory: 1250Mi
  requests:
    cpu: 25m
    memory: 262144k
```

다음 명령을 사용하여 **VPA** 권장 사항을 볼 수 있습니다.

```
$ oc get vpa <vpa-name> --output yaml
```

몇 분 후 출력에는 **CPU** 및 메모리 요청에 대한 권장 사항이 표시되며 다음과 유사합니다.

출력 예

```
...
status:
...
recommendation:
  containerRecommendations:
  - containerName: frontend
    lowerBound:
      cpu: 25m
      memory: 262144k
    target:
      cpu: 25m
      memory: 262144k
    uncappedTarget:
      cpu: 25m
      memory: 262144k
    upperBound:
      cpu: 262m
      memory: "274357142"
  - containerName: backend
    lowerBound:
      cpu: 12m
      memory: 131072k
    target:
```

```

cpu: 12m
memory: 131072k
uncappedTarget:
cpu: 12m
memory: 131072k
upperBound:
cpu: 476m
memory: "498558823"
...

```

출력에는 권장 리소스(target), 최소 권장 리소스(lowerBound), 최고 권장 리소스(upperBound), 최
 신 리소스 권장 사항(uncappedTarget)이 표시됩니다.

VPA는 lowerBound 및 upperBound 값을 사용하여 Pod를 업데이트해야 하는지 확인합니다. Pod에
 lowerBound 값보다 작거나 upperBound 값을 초과하는 리소스 요청이 있는 경우 VPA는 Pod를 종료하
 고 target 값을 사용하여 Pod를 다시 생성합니다.

2.5.3.1. VPA 최소 값 변경

기본적으로 워크로드 오브젝트는 VPA에서 Pod를 자동으로 삭제하고 업데이트할 수 있도록 최소 두
 개의 복제본을 지정해야 합니다. 결과적으로 두 개 미만의 복제본을 지정하는 워크로드 오브젝트는 VPA
 에서 자동으로 작동하지 않습니다. VPA는 VPA 외부의 일부 프로세스에서 Pod를 재시작하는 경우 이러
 한 워크로드 오브젝트에서 새 Pod를 업데이트합니다. VerticalPodAutoscalerController 사용자 정의 리
 소스(CR)에서 minReplicas 매개변수를 수정하여 클러스터 전체 최소 값을 변경할 수 있습니다.

예를 들어 minReplicas 를 3 으로 설정하면 VPA에서 복제본 3개 미만을 지정하는 워크로드 오브젝트
 의 Pod를 삭제하지 않습니다.

참고

minReplicas 를 1 로 설정하면 VPA에서 하나의 복제본만 지정하는 워크로드 오브젝
 트에 대한 유일한 Pod를 삭제할 수 있습니다. VPA에서 리소스를 조정하기 위해 Pod를 삭
 제할 때마다 워크로드가 중단될 수 있는 경우에만 이 설정을 one-replica 오브젝트에 사용
 해야 합니다. one-replica 오브젝트로 원치 않는 다운타임을 방지하려면
 podUpdatePolicy 가 Initial 로 설정된 VPA CR을 구성하여 일부 프로세스에서 VPA를 재
 시작하거나 애플리케이션에 적절한 시간에 Pod를 수동으로 업데이트할 수 있는 경우에만
 Pod를 자동으로 업데이트합니다.

VerticalPodAutoscalerController 오브젝트의 예


```

apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
metadata:
  creationTimestamp: "2021-04-21T19:29:49Z"
  generation: 2
  name: default
  namespace: openshift-vertical-pod-autoscaler
  resourceVersion: "142172"
  uid: 180e17e9-03cc-427f-9955-3b4d7aeb2d59
spec:
  minReplicas: 3 1
  podMinCPUMillicores: 25
  podMinMemoryMb: 250
  recommendationOnly: false
  safetyMarginFraction: 0.15

```



VPA가 작동할 워크로드 오브젝트에서 최소 복제본 수를 지정합니다. 복제본이 거의 없는 모든 오브젝트는 VPA에서 자동으로 삭제되지 않습니다.

2.5.3.2. VPA 권장 사항 자동 적용

VPA를 사용하여 Pod를 자동으로 업데이트하려면 `updateMode`를 `Auto` 또는 `Recreate`로 설정하여 특정 워크로드 오브젝트에 대한 VPA CR을 생성합니다.

워크로드 오브젝트에 대한 Pod가 생성되면 VPA에서 컨테이너를 지속적으로 모니터링하여 CPU 및 메모리 요구 사항을 분석합니다. VPA는 CPU 및 메모리에 대한 VPA 권장 사항을 충족하지 않는 모든 Pod를 삭제합니다. 재배포되면 Pod는 VPA 권장 사항에 따라 새 리소스 제한 및 요청을 사용하여 애플리케이션에 대해 설정된 모든 Pod 중단 예산을 준수합니다. 권장 사항은 참조를 위해 VPA CR의 `status 필드`에 추가되어 있습니다.



참고

기본적으로 워크로드 오브젝트는 VPA에서 Pod를 자동으로 삭제할 수 있도록 최소 두 개의 복제본을 지정해야 합니다. 이 최소보다 적은 복제본을 지정하는 워크로드 오브젝트는 삭제되지 않습니다. 이러한 Pod를 수동으로 삭제하면 워크로드 오브젝트에서 Pod를 재배포할 때 VPA는 권장 사항으로 새 Pod를 업데이트합니다. VPA 최소 값 변경과 같이 `VerticalPodAutoscalerController` 오브젝트를 수정하여 이 최소값을 변경할 수 있습니다.

Auto 모드 VPA CR의 예

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Auto" 3

```

1

이 VPA CR에서 관리할 워크로드 오브젝트의 유형입니다.

2

이 VPA CR에서 관리할 워크로드 오브젝트의 이름입니다.

3

모드를 **Auto** 또는 **Recreate**로 설정합니다.

- **Auto.** VPA는 Pod 생성 시 리소스 요청을 할당하고 요청된 리소스가 새 권장 사항과 크게 다른 경우 기존 Pod를 종료하여 업데이트합니다.
- **Recreate.** VPA는 Pod 생성 시 리소스 요청을 할당하고 요청된 리소스가 새 권장 사항과 크게 다른 경우 기존 Pod를 종료하여 업데이트합니다. 이 모드는 리소스 요청이 변경 될 때마다 Pod를 재시작해야 하는 경우에만 사용해야 합니다.



참고

VPA에서 리소스에 대한 권장 사항을 결정하고 권장 리소스를 새 Pod에 적용하려면 작동 중인 Pod가 있어야 하며 프로젝트에서 실행 중이어야 합니다.

CPU 및 메모리와 같은 워크로드의 리소스 사용량이 일관되게 유지되면 VPA에서 몇 분 내에 리소스에 대한 권장 사항을 확인할 수 있습니다. 워크로드의 리소스 사용량이 일치하지 않는 경우 VPA는 정확한 권장 사항을 수행하기 위해 VPA의 다양한 리소스 사용량 간격으로 지표를 수집해야 합니다.

2.5.3.3. Pod 생성에 VPA 권장 사항 자동 적용

VPA를 사용하여 Pod를 처음 배포할 때만 권장 리소스를 적용하려면 `updateMode`를 `Initial`로 설정하여 특정 워크로드 오브젝트에 대한 VPA CR을 생성합니다.

그런 다음 VPA 권장 사항을 사용하려는 워크로드 오브젝트와 연결된 모든 Pod를 수동으로 삭제합니다. `Initial` 모드에서 VPA는 새 리소스 권장 사항을 확인할 때 Pod를 삭제하지 않고 Pod을 업데이트하지도 않습니다.

Initial 모드 VPA CR의 예

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Initial" 3
```

1

이 VPA CR에서 관리할 워크로드 오브젝트의 유형입니다.

2

이 VPA CR에서 관리할 워크로드 오브젝트의 이름입니다.

3

모드를 **Initial**로 설정합니다. Pod가 생성되면 VPA에서 리소스를 할당하고 Pod 수명 동안 리소스를 변경하지 않습니다.



참고

VPA에서 권장 리소스를 결정하고 새 Pod에 권장 사항을 적용하려면 운영 Pod가 있어야 하며 프로젝트에서 실행 중이어야 합니다.

VPA에서 가장 정확한 권장 사항을 얻으려면 Pod가 실행되고 VPA가 안정될 때까지 최소 8일 동안 기다립니다.

2.5.3.4. VPA 권장 사항 수동 적용

VPA를 권장 CPU 및 메모리 값을 확인하는 데에만 사용하려면 updateMode를 off로 설정하여 특정 워크로드 오브젝트에 대한 VPA CR을 생성합니다.

해당 워크로드 오브젝트에 대한 Pod가 생성되면 VPA는 컨테이너의 CPU 및 메모리 요구 사항을 분석하고 VPA CR의 status 필드에 해당 권장 사항을 기록합니다. VPA는 새 리소스 권장 사항을 확인할 때 Pod를 업데이트하지 않습니다.

Off 모드 VPA CR의 예

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Off" 3

```

1

이 **VPA CR**에서 관리할 워크로드 오브젝트의 유형입니다.

2

이 **VPA CR**에서 관리할 워크로드 오브젝트의 이름입니다.

3

모드를 **Off**로 설정합니다.

다음 명령을 사용하여 권장 사항을 볼 수 있습니다.

```
$ oc get vpa <vpa-name> --output yaml
```

권장 사항에 따라 워크로드 오브젝트를 편집하여 **CPU** 및 메모리 요청을 추가한 다음 권장 리소스를 사용하여 **Pod**를 삭제하고 재배포할 수 있습니다.



참고

VPA에서 권장 리소스를 결정하고 새 **Pod**에 권장 사항을 적용하려면 운영 **Pod**가 있어야 하며 프로젝트에서 실행 중이어야 합니다.

VPA에서 가장 정확한 권장 사항을 얻으려면 **Pod**가 실행되고 **VPA**가 안정될 때까지 최소 **8일** 동안 기다립니다.

2.5.3.5. VPA 권장 사항 적용에서 컨테이너 제외

워크로드 오브젝트에 컨테이너가 여러 개 있고 **VPA**에서 모든 컨테이너를 평가하고 해당 컨테이너에 대해 작동하지 않도록 하려면 특정 워크로드 오브젝트에 대한 **VPA CR**을 생성하고 **resourcePolicy**를 추가하여 특정 컨테이너를 옵트아웃합니다.

VPA에서 권장 리소스를 사용하여 **Pod**를 업데이트하면 **resourcePolicy**가 포함된 모든 컨테이너가 업데이트되지 않으며 **VPA**는 **Pod**의 해당 컨테이너에 대한 권장 사항을 제공하지 않습니다.

■

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Auto" 3
  resourcePolicy: 4
  containerPolicies:
    - containerName: my-opt-sidecar
      mode: "Off"

```

1

이 VPA CR에서 관리할 워크로드 오브젝트의 유형입니다.

2

이 VPA CR에서 관리할 워크로드 오브젝트의 이름입니다.

3

모드를 **Auto**, **Recreate** 또는 **Off**로 설정합니다. **Recreate** 모드는 리소스 요청이 변경될 때마다 Pod를 재시작해야 하는 경우에만 사용해야 합니다.

4

옵트아웃할 컨테이너를 지정하고 **mode**를 **Off**로 설정합니다.

예를 들면 Pod에 다음과 같이 리소스 요청 및 제한이 동일한 두 개의 컨테이너가 있습니다.

```

# ...
spec:
  containers:
    - name: frontend
      resources:
        limits:
          cpu: 1
          memory: 500Mi
        requests:
          cpu: 500m
          memory: 100Mi
    - name: backend
      resources:
        limits:

```

```

cpu: "1"
memory: 500Mi
requests:
  cpu: 500m
  memory: 100Mi
# ...

```

backend 컨테이너를 옵트아웃으로 설정하여 **VPA CR**을 시작하면 **VPA**에서 **Pod**를 종료한 후 **frontend** 컨테이너에만 적용되는 권장 리소스를 사용하여 **Pod**를 다시 생성합니다.

```

...
spec:
  containers:
    name: frontend
    resources:
      limits:
        cpu: 50m
        memory: 1250Mi
      requests:
        cpu: 25m
        memory: 262144k
  ...
  name: backend
  resources:
    limits:
      cpu: "1"
      memory: 500Mi
    requests:
      cpu: 500m
      memory: 100Mi
  ...

```

2.5.3.6. 대체 권장 사항 사용

자체 권장 사항을 사용하여 자체 알고리즘에 따라 자동 스케일링할 수 있습니다. 대체 권장 사항을 지정하지 않으면 **OpenShift Container Platform**에서는 기본 권장 사항을 사용하므로 기록 사용량에 따라 **CPU** 및 메모리 요청을 제안합니다. 모든 유형의 워크로드에 적용되는 일반적인 권장 사항 정책은 없으므로 특정 워크로드에 대해 다른 권장 사항을 생성하고 배포해야 할 수 있습니다.

예를 들어, 기본 권장 사항은 컨테이너가 애플리케이션 모니터링에 사용되는 것과 같이 사용량 급증 및 ID 간 대체 패턴 또는 딥 러닝 애플리케이션에 사용되는 반복 패턴과 같은 특정 리소스 동작을 표시하는 경우 향후 리소스 사용량을 정확하게 예측하지 않을 수 있습니다. 이러한 사용 동작에 기본 권장 사항을 사용하면 애플리케이션에 대해 과도한 프로비저닝 및 **OOM**(메모리 부족)이 종료될 수 있습니다.



참고

권장 사항을 생성하는 방법에 대한 지침은 이 문서의 범위를 벗어납니다.

절차

Pod에 대체 권장 사항을 사용하려면 다음을 수행합니다.

1. 대체 권장 사항에 대한 서비스 계정을 생성하고 해당 서비스 계정을 필요한 클러스터 역할에 바인딩합니다.

```

apiVersion: v1 1
kind: ServiceAccount
metadata:
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 2
kind: ClusterRoleBinding
metadata:
  name: system:example-metrics-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:metrics-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 3
kind: ClusterRoleBinding
metadata:
  name: system:example-vpa-actor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-actor
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 4
kind: ClusterRoleBinding
metadata:
  name: system:example-vpa-target-reader-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-target-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>

```


1

권장자가 배포된 네임스페이스에서 권장 사항에 대한 서비스 커밋을 생성합니다.

2

권장 사항 서비스 계정을 **metrics-reader** 역할에 바인딩합니다. 권장자를 배포할 네임스페이스를 지정합니다.

3

권장 사항 서비스 계정을 **vpa-actor** 역할에 바인딩합니다. 권장자를 배포할 네임스페이스를 지정합니다.

4

권장 사항 서비스 계정을 **vpa-target-reader** 역할에 바인딩합니다. 권장자를 배포할 네임스페이스를 지정합니다.

2.

대체 권장 사항을 클러스터에 추가하려면 다음과 유사한 **Deployment** 오브젝트를 생성합니다.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: alt-vpa-recommender
  namespace: <namespace_name>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alt-vpa-recommender
  template:
    metadata:
      labels:
        app: alt-vpa-recommender
    spec:
      containers: 1
      - name: recommender
        image: quay.io/example/alt-recommender:latest 2
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 200m
            memory: 1000Mi
          requests:
            cpu: 50m
            memory: 500Mi
      ports:

```

```

- name: prometheus
  containerPort: 8942
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
    seccompProfile:
      type: RuntimeDefault
  serviceAccountName: alt-vpa-recommender-sa
  securityContext:
    runAsNonRoot: true

```

1

대체 권장자를 위한 컨테이너를 만듭니다.

2

권장 이미지를 지정합니다.

3

권장자를 위해 생성한 서비스 계정을 연결합니다.

동일한 네임스페이스에서 대체 추천을 위해 새 Pod가 생성됩니다.

```
$ oc get pods
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
frontend-845d5478d-558zf	1/1	Running	0	4m25s
frontend-845d5478d-7z9gx	1/1	Running	0	4m25s
frontend-845d5478d-b7l4j	1/1	Running	0	4m25s
vpa-alt-recommender-55878867f9-6tp5v	1/1	Running	0	9s

3.

대체 권장 배포 오브젝트의 이름을 포함하는 VPA CR을 구성합니다.

대체 권장 사항을 포함하는 VPA CR의 예

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
  namespace: <namespace_name>
spec:
  recommenders:
    - name: alt-vpa-recommender ❶
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment ❷
    name: frontend

```

❶

대체 권장 배포의 이름을 지정합니다.

❷

이 VPA에서 관리할 기존 워크로드 오브젝트의 이름을 지정합니다.

2.5.4. Vertical Pod Autoscaler Operator 사용

VPA(Vertical Pod Autoscaler Operator) CR(사용자 정의 리소스)을 생성하여 VPA를 사용할 수 있습니다. CR은 VPA에서 해당 Pod에 수행할 작업을 분석하고 결정해야 하는 Pod를 나타냅니다.

사전 요구 사항

- 자동 스케일링하려는 워크로드 오브젝트가 있어야 합니다.
- 대체 권장 사항을 사용하려면 해당 권장자를 포함한 배포가 있어야 합니다.

절차

특정 워크로드 오브젝트에 대한 VPA CR을 생성하려면 다음을 수행합니다.

1. 스케일링할 워크로드 오브젝트가 있는 프로젝트로 변경합니다.

a. VPA CR YAML 파일을 생성합니다.

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Auto" 3
  resourcePolicy: 4
  containerPolicies:
    - containerName: my-opt-sidecar
      mode: "Off"
  recommenders: 5
    - name: my-recommender

```

1

이 VPA에서 관리할 워크로드 오브젝트 유형(Deployment, StatefulSet, Job, DaemonSet, ReplicaSet 또는 ReplicationController)을 지정합니다.

2

이 VPA에서 관리할 기존 워크로드 오브젝트의 이름을 지정합니다.

3

다음과 같이 VPA 모드를 지정합니다.

- **auto:** 컨트롤러와 연결된 Pod에 권장 리소스를 자동으로 적용합니다. VPA는 기존 Pod를 종료하고 권장 리소스 제한 및 요청을 사용하여 새 Pod를 생성합니다.
- **recreate:** 워크로드 오브젝트와 연결된 Pod에 권장 리소스를 자동으로 적용합니다. VPA는 기존 Pod를 종료하고 권장 리소스 제한 및 요청을 사용하여 새 Pod를 생성합니다. recreate 모드는 리소스 요청이 변경될 때마다 Pod를 재시작해야 하는 경우에만 사용해야 합니다.
-

initial: 워크로드 오브젝트와 연결된 **Pod**가 생성될 때 권장 리소스를 자동으로 적용합니다. **VPA**는 새 리소스 권장 사항을 확인할 때 **Pod**를 업데이트하지 않습니다.

•

off: 워크로드 오브젝트와 연결된 **Pod**의 리소스 권장 사항만 생성합니다. **VPA**는 새 리소스 권장 사항을 확인할 때 **Pod**를 업데이트하지 않고 해당 권장 사항을 새 **Pod**에 적용하지도 않습니다.

4

선택 사항: 옵트아웃할 컨테이너를 지정하고 모드를 **Off**로 설정합니다.

5

선택 사항: 대체 권장 사항을 지정합니다.

b.

VPA CR을 생성합니다.

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

잠시 후 **VPA**는 워크로드 오브젝트와 연결된 **Pod**에서 컨테이너의 리소스 사용량을 확인합니다.

다음 명령을 사용하여 **VPA** 권장 사항을 볼 수 있습니다.

```
$ oc get vpa <vpa-name> --output yaml
```

출력에는 **CPU** 및 메모리 요청에 대한 권장 사항이 표시되며 다음과 유사합니다.

출력 예

```
...
status:
...

recommendation:
  containerRecommendations:
    - containerName: frontend
```

```

lowerBound: ①
  cpu: 25m
  memory: 262144k
target: ②
  cpu: 25m
  memory: 262144k
uncappedTarget: ③
  cpu: 25m
  memory: 262144k
upperBound: ④
  cpu: 262m
  memory: "274357142"
- containerName: backend
  lowerBound:
    cpu: 12m
    memory: 131072k
  target:
    cpu: 12m
    memory: 131072k
  uncappedTarget:
    cpu: 12m
    memory: 131072k
  upperBound:
    cpu: 476m
    memory: "498558823"

```

...

①

lowerBound는 최소 권장 리소스 수준은 입니다.

②

target은 권장 리소스 수준입니다.

③

upperBound는 권장되는 최고 리소스 수준입니다.

④

uncappedTarget은 최신 리소스 권장 사항입니다.

2.5.5. Vertical Pod Autoscaler Operator 설치 제거

OpenShift Container Platform 클러스터에서 **VPA(Vertical Pod Autoscaler Operator)**를 제거할 수 있습니다. 설치 제거해도 기존 **VPA CR**에 의해 이미 수정된 **Pod**의 리소스 요청은 변경되지 않습니다. 새 **Pod**에서는 모두 **Vertical Pod Autoscaler Operator**에서 설정한 권장 사항 대신 워크로드 오브젝트에 정의된 리소스를 가져옵니다.



참고

`oc delete vpa <vpa-name >` 명령을 사용하여 특정 **VPA CR**을 제거할 수 있습니다. 리소스 요청에는 수직 **Pod** 자동 스케일러를 설치 제거할 때와 동일한 작업이 적용됩니다.

VPA Operator를 제거한 후 잠재적인 문제를 방지하려면 **Operator**와 관련된 다른 구성 요소를 제거하는 것이 좋습니다.

사전 요구 사항

- **Vertical Pod Autoscaler Operator**를 설치해야 합니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → 설치된 **Operator**를 클릭합니다.
2. **openshift-vertical-pod-autoscaler** 프로젝트로 전환합니다.
3. **VerticalPodAutoscaler Operator**의 경우 옵션 메뉴
 - ⋮
 를 클릭하고 **Operator** 설치 제거를 선택합니다.
4. 선택 사항: **Operator**와 연결된 모든 피연산자를 제거하려면 대화 상자에서 이 연산자의 모든 피연산자 인스턴스 삭제를 선택합니다.
5. 제거를 클릭합니다.
6. 선택 사항: **OpenShift CLI**를 사용하여 **VPA** 구성 요소를 제거합니다.

- a. VPA 네임스페이스를 삭제합니다.

```
$ oc delete namespace openshift-vertical-pod-autoscaler
```

- b. VPA CRD(사용자 정의 리소스 정의) 오브젝트를 삭제합니다.

```
$ oc delete crd verticalpodautoscalercheckpoints.autoscaling.k8s.io
```

```
$ oc delete crd verticalpodautoscalercontrollers.autoscaling.openshift.io
```

```
$ oc delete crd verticalpodautoscalers.autoscaling.k8s.io
```

CRD를 삭제하면 연결된 역할, 클러스터 역할 및 역할 바인딩이 제거됩니다.



참고

이 작업은 클러스터에서 모든 사용자가 생성한 VPA CR에서 제거됩니다. VPA를 다시 설치하는 경우 이러한 오브젝트를 다시 생성해야 합니다.

- c. 다음 명령을 실행하여 MutatingWebhookConfiguration 오브젝트를 삭제합니다.

```
$ oc delete MutatingWebhookConfiguration vpa-webhook-config
```

- d. VPA Operator를 삭제합니다.

```
$ oc delete operator/vertical-pod-autoscaler.openshift-vertical-pod-autoscaler
```

2.6. POD에 민감한 데이터 제공

일부 애플리케이션에는 개발자에게 제공하길 원하지 않는 민감한 정보(암호 및 사용자 이름 등)가 필요합니다.

관리자는 Secret 오브젝트를 사용하여 이러한 정보를 명확한 텍스트로 공개하지 않고도 제공할 수 있습니다.

2.6.1. 보안 이해

Secret 오브젝트 유형에서는 암호, **OpenShift Container Platform** 클라이언트 구성 파일, 개인 소스 리포지토리 자격 증명 등과 같은 중요한 정보를 보유하는 메커니즘을 제공합니다. 보안은 **Pod**에서 중요한 콘텐츠를 분리합니다. 볼륨 플러그인을 사용하여 컨테이너에 보안을 마운트하거나 시스템에서 시크릿을 사용하여 **Pod** 대신 작업을 수행할 수 있습니다.

주요 속성은 다음과 같습니다.

- 보안 데이터는 정의와는 별도로 참조할 수 있습니다.
- 보안 데이터 볼륨은 임시 파일 저장 기능(**tmpfs**)에 의해 지원되며 노드에 저장되지 않습니다.
- 보안 데이터는 네임스페이스 내에서 공유할 수 있습니다.

YAML Secret 오브젝트 정의

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque ①
data: ②
  username: <username> ③
  password: <password>
stringData: ④
  hostname: myapp.mydomain.com ⑤
```

①

보안의 키 이름과 값의 구조를 나타냅니다.

②

data 필드에서 허용되는 키 형식은 **Kubernetes** 구분자 용어집의 **DNS_SUBDOMAIN** 값에 있는 지침을 충족해야 합니다.

3

data 맵의 키와 관련된 값은 **base64**로 인코딩되어야 합니다.

4

stringData 맵의 항목이 **base64**로 변환된 후 해당 항목이 자동으로 **data** 맵으로 이동합니다. 이 필드는 쓰기 전용이며 값은 **data** 필드를 통해서만 반환됩니다.

5

stringData 맵의 키와 관련된 값은 일반 텍스트 문자열로 구성됩니다.

먼저 보안을 생성한 후 해당 보안을 사용하는 **Pod**를 생성해야 합니다.

보안 생성 시 다음을 수행합니다.

- 보안 데이터를 사용하여 보안 오브젝트를 생성합니다.
- **Pod** 서비스 계정을 업데이트하여 보안에 대한 참조를 허용합니다.
- 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

2.6.1.1. 보안 유형

type 필드의 값은 보안의 키 이름과 값의 구조를 나타냅니다. 유형을 사용하면 보안 오브젝트에 사용자 이름과 키를 적용할 수 있습니다. 검증을 수행하지 않으려면 기본값인 **opaque** 유형을 사용합니다.

보안 데이터에 특정 키 이름이 있는지 확인하기 위해 서버 측 최소 검증을 트리거하려면 다음 유형 중 하나를 지정합니다.

- **kubernetes.io/basic-auth**: 기본 인증으로 사용
- **kubernetes.io/dockercfg**: 이미지 풀 시크릿으로 사용

- **kubernetes.io/dockerconfigjson:** 이미지 풀 시크릿으로 사용
- **kubernetes.io/service-account-token:** 기존 서비스 계정 API 토큰을 얻으려면 사용
- **kubernetes.io/ssh-auth:** SSH 키 인증과 함께 사용
- **kubernetes.io/tls:** TLS 인증 기관과 함께 사용

검증을 수행하지 않으려면 **typ: Opaque**를 지정합니다. 즉 보안에서 키 이름 또는 값에 대한 규칙을 준수하도록 요청하지 않습니다. **opaque** 보안에는 임의의 값을 포함할 수 있는 비정형 **key:value** 쌍을 사용할 수 있습니다.



참고

example.com/my-secret-type과 같은 다른 임의의 유형을 지정할 수 있습니다. 이러한 유형은 서버 측에 적용되지 않지만 보안 생성자가 해당 유형의 키/값 요구 사항을 준수하도록 의도했음을 나타냅니다.

다양한 유형의 시크릿 생성 예를 보려면 시크릿을 생성하는 방법 이해를 참조하십시오.

2.6.1.2. 보안 데이터 키

보안키는 DNS 하위 도메인에 있어야 합니다.

2.6.1.3. 자동으로 생성된 서비스 계정 토큰 시크릿 정보

서비스 계정이 생성되면 서비스 계정 토큰 시크릿이 자동으로 생성됩니다. 이 서비스 계정 토큰 시크릿은 자동으로 생성된 Docker 구성 시크릿과 함께 내부 OpenShift Container Platform 레지스트리에 인증하는 데 사용됩니다. 자동으로 생성된 시크릿을 사용하지 마십시오. 향후 OpenShift Container Platform 릴리스에서 제거될 수 있습니다.



참고

OpenShift Container Platform 4.11 이전에는 서비스 계정이 생성될 때 두 번째 서비스 계정 토큰 시크릿이 생성되었습니다. 이 서비스 계정 토큰 시크릿은 **Kubernetes API**에 액세스하는 데 사용되었습니다.

OpenShift Container Platform 4.11부터 이 두 번째 서비스 계정 토큰 시크릿이 더 이상 생성되지 않습니다. 이는 **LegacyServiceAccountTokenNoAutoGeneration** 업스트림 **Kubernetes** 기능 게이트가 활성화되어 **Kubernetes API**에 액세스하기 위한 시크릿 기반 서비스 계정 토큰의 자동 생성을 중지하기 때문입니다.

4.12로 업그레이드한 후 기존 서비스 계정 토큰 시크릿은 삭제되지 않고 계속 작동합니다.

바인딩된 서비스 계정 토큰을 얻기 위해 예상 볼륨과 함께 워크로드가 자동으로 삽입됩니다. 워크로드에 추가 서비스 계정 토큰이 필요한 경우 워크로드 매니페스트에 예상 볼륨을 추가합니다. 바인딩된 서비스 계정 토큰은 다음과 같은 이유로 서비스 계정 토큰 시크릿보다 안전합니다.

- 바인딩된 서비스 계정 토큰은 수명이 바인딩되어 있습니다.
- 바인딩된 서비스 계정 토큰에는 대상자가 포함됩니다.
- 바인딩된 서비스 계정 토큰은 **Pod** 또는 보안에 바인딩될 수 있으며 바인딩된 오브젝트가 제거되면 바인딩된 토큰이 무효화됩니다.

자세한 내용은 볼륨 프로젝션을 사용하여 바인딩된 서비스 계정 토큰 구성을 참조하십시오.

읽을 수 있는 **API** 오브젝트에서 만료되지 않은 토큰을 보안 노출하는 경우 토큰을 얻기 위해 서비스 계정 토큰 시크릿을 수동으로 생성할 수도 있습니다. 자세한 내용은 [서비스 계정 토큰 시크릿 생성](#)을 참조하십시오.

추가 리소스

- 바인딩된 서비스 계정 토큰을 요청하는 방법에 대한 자세한 내용은 [바인딩된 서비스 계정 토큰 사용](#)을 참조하십시오.

- 서비스 계정 토큰 시크릿 생성에 대한 자세한 내용은 [서비스 계정 토큰 시크릿 생성](#) 을 참조 하십시오.

2.6.2. 보안 생성 방법 이해

관리자는 개발자가 해당 보안을 사용하는 **Pod**를 생성하기 전에 보안을 생성해야 합니다.

보안 생성 시 다음을 수행합니다.

1. 시크릿을 유지하려는 데이터가 포함된 보안 오브젝트를 생성합니다. 각 시크릿 유형에 필요한 특정 데이터는 다음 섹션에서 확인할 수 있습니다.

불투명 보안을 생성하는 **YAML** 오브젝트의 예

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
type: Opaque 1
data: 2
  username: <username>
  password: <password>
stringData: 3
  hostname: myapp.mydomain.com
secret.properties: |
  property1=valueA
  property2=valueB
```

1

보안 유형을 지정합니다.

2

인코딩된 문자열 및 데이터를 지정합니다.

3

디코딩된 문자열 및 데이터를 지정합니다.

둘 다 아닌 **data** 또는 **stringdata** 필드를 사용합니다.

- 2. **Pod**의 서비스 계정을 업데이트하여 보안을 참조합니다.

보안을 사용하는 서비스 계정의 **YAML**

```
apiVersion: v1
kind: ServiceAccount
...
secrets:
- name: test-secret
```

- 3. 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

보안 데이터로 볼륨의 파일을 채우는 **Pod**의 **YAML**

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
  - name: secret-test-container
    image: busybox
    command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
    volumeMounts: ①
    - name: secret-volume
      mountPath: /etc/secret-volume ②
      readOnly: true ③
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret ④
  restartPolicy: Never
```

1

보안이 필요한 각 컨테이너에 **volumeMounts** 필드를 추가합니다.

2

보안을 표시할 미사용 디렉터리 이름을 지정합니다. 시크릿 데이터 맵의 각 키는 **mountPath** 아래의 파일 이름이 됩니다.

3

true로 설정합니다. **true**인 경우 드라이버에 읽기 전용 볼륨을 제공하도록 지시합니다.

4

시크릿 이름을 지정합니다.

보안 데이터로 환경 변수를 채우는 Pod의 YAML

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
  - name: secret-test-container
    image: busybox
    command: [ "/bin/sh", "-c", "export" ]
    env:
    - name: TEST_SECRET_USERNAME_ENV_VAR
      valueFrom:
        secretKeyRef: 1
          name: test-secret
          key: username
    restartPolicy: Never

```

1

secret 키를 사용하는 환경 변수를 지정합니다.

보안 데이터로 환경 변수를 채우는 빌드 구성의 **YAML**

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef: ❶
            name: test-secret
            key: username
      from:
        kind: ImageStreamTag
        namespace: openshift
        name: 'cli:latest'

```

❶

secret 키를 사용하는 환경 변수를 지정합니다.

2.6.2.1. 보안 생성 제한 사항

보안을 사용하려면 **Pod**에서 보안을 참조해야 합니다. 보안은 다음 세 가지 방법으로 **Pod**에서 사용할 수 있습니다.

- 컨테이너에 환경 변수를 채우기 위해 사용.
- 하나 이상의 컨테이너에 마운트된 볼륨에서 파일로 사용.
- **Pod**에 대한 이미지를 가져올 때 **kubelet**으로 사용.

볼륨 유형 보안은 볼륨 메커니즘을 사용하여 데이터를 컨테이너에 파일로 작성합니다. 이미지 가져오기 보안은 서비스 계정을 사용하여 네임스페이스의 모든 **Pod**에 보안을 자동으로 삽입합니다.

템플릿에 보안 정의가 포함된 경우 템플릿에 제공된 보안을 사용할 수 있는 유일한 방법은 보안 볼륨 소스를 검증하고 지정된 오브젝트 참조가 **Secret** 오브젝트를 실제로 가리키는 것입니다. 따라서 보안을 생성한 후 해당 보안을 사용하는 **Pod**를 생성해야 합니다. 가장 효과적인 방법은 서비스 계정을 사용하여 자동으로 삽입되도록 하는 것입니다.

Secret API 오브젝트는 네임스페이스에 있습니다. 동일한 네임스페이스에 있는 **Pod**만 참조할 수 있습니다.

개별 보안은 **1MB**로 제한됩니다. 이는 대규모 보안이 생성되어 **apiserver** 및 **kubelet** 메모리가 소모되는 것을 막기 위한 것입니다. 그러나 작은 보안을 많이 생성해도 메모리가 소모될 수 있습니다.

2.6.2.2. 불투명 보안 생성

관리자는 임의의 값을 포함할 수 있는 비정형 **key:value** 쌍을 저장할 수 있는 불투명 보안을 생성할 수 있습니다.

절차

1. 컨트롤 플레인 노드의 **YAML** 파일에 **Secret** 오브젝트를 생성합니다.

예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque ①
data:
  username: <username>
  password: <password>
```

①

불투명 보안을 지정합니다.

2.

다음 명령을 사용하여 **Secret** 오브젝트를 생성합니다.

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

3.

Pod에서 보안을 사용하려면 다음을 수행합니다.

a.

"보안 생성 방법" 섹션에 표시된 대로 **Pod**의 서비스 계정을 업데이트하여 보안을 참조합니다.

b.

"**secret**을 보안 생성 방법" 섹션에 설명된 대로 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

추가 리소스

-

Pod에서 시크릿 사용에 대한 자세한 내용은 [시크릿 생성 방법 이해](#) 를 참조하십시오.

2.6.2.3. 서비스 계정 토큰 시크릿 생성

관리자는 **API**에 인증해야 하는 애플리케이션에 서비스 계정 토큰을 배포할 수 있는 서비스 계정 토큰 시크릿을 생성할 수 있습니다.



참고

서비스 계정 토큰 시크릿을 사용하는 대신 **TokenRequest API**를 사용하여 바인딩된 서비스 계정 토큰을 얻는 것이 좋습니다. **TokenRequest API**에서 얻은 토큰은 바인딩된 수명을 가지며 다른 **API** 클라이언트에서 읽을 수 없기 때문에 시크릿에 저장된 토큰보다 더 안전합니다.

TokenRequest API를 사용할 수 없고 읽을 수 없는 **API** 오브젝트에서의 보안 노출이 허용 가능한 경우에만 서비스 계정 토큰 시크릿을 생성해야 합니다.

바인딩된 서비스 계정 토큰 생성에 대한 정보는 다음 추가 리소스 섹션을 참조하십시오.

절차

1. 컨트롤 플레인 노드의 **YAML** 파일에 **Secret** 오브젝트를 생성합니다.

보안 오브젝트 의 예:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa-sample
  annotations:
    kubernetes.io/service-account.name: "sa-name" ①
type: kubernetes.io/service-account-token ②
```

①

기존 서비스 계정 이름을 지정합니다. **ServiceAccount** 및 **Secret** 오브젝트를 모두 생성하는 경우 **ServiceAccount** 오브젝트를 먼저 생성합니다.

②

서비스 계정 토큰 보안을 지정합니다.

2. 다음 명령을 사용하여 **Secret** 오브젝트를 생성합니다.

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

3. **Pod**에서 보안을 사용하려면 다음을 수행합니다.

- a. "보안 생성 방법" 섹션에 표시된 대로 **Pod**의 서비스 계정을 업데이트하여 보안을 참조합니다.
- b. "**secret**을 보안 생성 방법" 섹션에 설명된 대로 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

추가 리소스

- Pod에서 시크릿 사용에 대한 자세한 내용은 [시크릿 생성 방법 이해](#) 를 참조하십시오.
- 바인딩된 서비스 계정 토큰을 요청하는 방법에 대한 자세한 내용은 [바인딩된 서비스 계정 토큰 사용](#)을 참조하십시오.
- 서비스 계정 생성에 대한 자세한 내용은 서비스 계정 [이해 및 생성](#)을 참조하십시오.

2.6.2.4. 기본 인증 보안 생성

관리자는 기본 인증에 필요한 자격 증명을 저장할 수 있는 기본 인증 보안을 생성할 수 있습니다. 이 시크릿 유형을 사용하는 경우 **Secret** 오브젝트의 **data** 매개변수에 **base64** 형식으로 인코딩된 다음 키가 포함되어야 합니다.

- **Username:** 인증을 위한 사용자 이름
- **password:** 인증이 필요한 암호 또는 토큰



참고

stringData 매개변수를 사용하여 일반 텍스트 콘텐츠를 사용할 수 있습니다.

절차

1. 컨트롤 플레인 노드의 **YAML** 파일에 **Secret** 오브젝트를 생성합니다.

보안 오브젝트 의 예

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth 1
data:
stringData: 2
  username: admin
  password: <password>

```

1

기본 인증 보안을 지정합니다.

2

사용할 기본 인증 값을 지정합니다.

2.

다음 명령을 사용하여 **Secret** 오브젝트를 생성합니다.

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

3.

Pod에서 보안을 사용하려면 다음을 수행합니다.

a.

"보안 생성 방법" 섹션에 표시된 대로 **Pod**의 서비스 계정을 업데이트하여 보안을 참조합니다.

b.

"**secret**을 보안 생성 방법" 섹션에 설명된 대로 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

추가 리소스

•

Pod에서 시크릿 사용에 대한 자세한 내용은 [시크릿 생성 방법 이해](#) 를 참조하십시오.

2.6.2.5. SSH 인증 보안 생성

관리자는 **SSH** 인증에 사용되는 데이터를 저장할 수 있는 **SSH** 인증 시크릿을 생성할 수 있습니다. 이 시크릿 유형을 사용하는 경우 **Secret** 오브젝트의 **data** 매개변수에 사용할 **SSH** 자격 증명이 포함되어야 합니다.

절차

1.

컨트롤 플레인 노드의 **YAML** 파일에 **Secret** 오브젝트를 생성합니다.

보안 오브젝트 의 예:

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-ssh-auth
type: kubernetes.io/ssh-auth 1
data:
  ssh-privatekey: | 2
    MIIEpQIBAAKCAQEAlqb/Y ...

```

1

SSH 인증 보안을 지정합니다.

2

SSH 키/값 쌍을 사용할 SSH 자격 증명으로 지정합니다.

2.

다음 명령을 사용하여 **Secret** 오브젝트를 생성합니다.

```

$ oc create -f <filename>.yaml

```

3.

Pod에서 보안을 사용하려면 다음을 수행합니다.

a.

"보안 생성 방법" 섹션에 표시된 대로 **Pod**의 서비스 계정을 업데이트하여 보안을 참조합니다.

b.

"**secret**을 보안 생성 방법" 섹션에 설명된 대로 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 **Pod**를 생성합니다.

추가 리소스

-

[보안 생성 방법 이해](#)

2.6.2.6. Docker 구성 보안 생성

관리자는 컨테이너 이미지 레지스트리에 액세스하기 위한 인증 정보를 저장할 수 있는 **Docker** 구성 시크릿을 생성할 수 있습니다.

- **kubernetes.io/dockercfg.** 이 시크릿 유형을 사용하여 로컬 **Docker** 구성 파일을 저장합니다. **secret** 오브젝트의 **data** 매개변수에 **base64** 형식으로 인코딩된 **.dockercfg** 파일의 콘텐츠가 포함되어야 합니다.
- **kubernetes.io/dockerconfigjson.** 이 시크릿 유형을 사용하여 로컬 **Docker** 구성 **JSON** 파일을 저장합니다. **secret** 오브젝트의 **data** 매개변수에 **base64** 형식으로 인코딩된 **.docker/config.json** 파일의 내용이 포함되어야 합니다.

절차

1. 컨트롤 플레인 노드의 **YAML** 파일에 **Secret** 오브젝트를 생성합니다.

Docker 구성 보안 오브젝트의 예

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-cfg
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:
  .dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ
  2cgYXV0aCBrZXIzCg== 2

```

1

시크릿이 **Docker** 구성 파일을 사용하도록 지정합니다.

2

base64로 인코딩된 **Docker** 구성 파일의 출력

Docker 구성 JSON 시크릿 오브젝트의 예

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-json
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:
  .dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cgYXV0aCBrZXIzCg== 2

```

1

시크릿이 Docker 구성 JSONfile을 사용하도록 지정합니다.

2

base64로 인코딩된 Docker 구성 JSON 파일의 출력

2.

다음 명령을 사용하여 Secret 오브젝트를 생성합니다.

```

$ oc create -f <filename>.yaml

```

3.

Pod에서 보안을 사용하려면 다음을 수행합니다.

a.

"보안 생성 방법" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

b.

"secret을 보안 생성 방법" 섹션에 설명된 대로 보안을 환경 변수로 사용하거나 secret 볼륨을 사용하여 파일로 사용하는 Pod를 생성합니다.

추가 리소스

-

Pod에서 시크릿 사용에 대한 자세한 내용은 [시크릿 생성 방법 이해](#) 를 참조하십시오.

2.6.2.7. 웹 콘솔을 사용하여 보안 생성

웹 콘솔을 사용하여 보안을 생성할 수 있습니다.

절차

1. 워크로드 → 시크릿으로 이동합니다.
2. 생성 → **YAML** 을 클릭합니다.
 - a. 사양에 맞게 **YAML**을 수동으로 편집하거나 파일을 **YAML** 편집기로 끌어서 놓습니다. 예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: example
  namespace: <namespace>
type: Opaque ①
data:
  username: <base64 encoded username>
  password: <base64 encoded password>
stringData: ②
  hostname: myapp.mydomain.com
  
```

①

이 예제에서는 불투명 보안을 지정합니다. 그러나 서비스 계정 토큰 시크릿, 기본 인증 시크릿, SSH 인증 시크릿 또는 Docker 구성을 사용하는 시크릿과 같은 다른 시크릿 유형이 표시될 수 있습니다.

②

stringData 맵의 항목이 **base64**로 변환된 후 해당 항목이 자동으로 **data** 맵으로 이동합니다. 이 필드는 쓰기 전용이며 값은 **data** 필드를 통해서만 반환됩니다.

3. 생성을 클릭합니다.
4. 워크로드에 시크릿 추가를 클릭합니다.

- a. 드롭다운 메뉴에서 추가할 워크로드를 선택합니다.
- b. 저장을 클릭합니다.

2.6.3. 보안 업데이트 방법 이해

보안 값을 수정해도 이미 실행 중인 **Pod**에서 사용하는 값은 동적으로 변경되지 않습니다. 보안을 변경하려면 원래 **Pod**를 삭제하고 새 **Pod**를 생성해야 합니다(대개 동일한 **PodSpec** 사용).

보안 업데이트 작업에서는 새 컨테이너 이미지를 배포하는 것과 동일한 워크플로를 따릅니다. **kubectl rolling-update** 명령을 사용할 수 있습니다.

보안의 **resourceVersion** 값을 참조 시 지정되지 않습니다. 따라서 **Pod**가 시작되는 동시에 보안이 업데이트되는 경우 **Pod**에 사용되는 보안의 버전이 정의되지 않습니다.



참고

현재는 **Pod**가 생성될 때 사용된 보안 오브젝트의 리소스 버전을 확인할 수 없습니다. 컨트롤러에서 이전 **resourceVersion** 을 사용하여 재시작할 수 있도록 **Pod**에서 이 정보를 보고하도록 계획되어 있습니다. 그동안 기존 보안 데이터를 업데이트하지 말고 고유한 이름으로 새 보안을 생성하십시오.

2.6.4. 보안 생성 및 사용

관리자는 서비스 계정 토큰 시크릿을 생성할 수 있습니다. 이를 통해 **API**에 인증해야 하는 애플리케이션에 서비스 계정 토큰을 배포할 수 있습니다.

절차

1. 다음 명령을 실행하여 네임스페이스에 서비스 계정을 생성합니다.

```
$ oc create sa <service_account_name> -n <your_namespace>
```

2. 다음 **YAML** 예제를 **service-account-token-secret.yaml** 이라는 파일에 저장합니다. 예제에는 서비스 계정 토큰을 생성하는 데 사용할 수 있는 **Secret** 오브젝트 구성이 포함되어 있습니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: <secret_name> ❶
  annotations:
    kubernetes.io/service-account.name: "sa-name" ❷
type: kubernetes.io/service-account-token ❸

```

❶

& lt;secret_name& gt;을 서비스 토큰 시크릿의 이름으로 교체합니다.

❷

기존 서비스 계정 이름을 지정합니다. **ServiceAccount** 및 **Secret** 오브젝트를 모두 생성하는 경우 **ServiceAccount** 오브젝트를 먼저 생성합니다.

❸

서비스 계정 토큰 시크릿 유형을 지정합니다.

3.

파일을 적용하여 서비스 계정 토큰을 생성합니다.

```
$ oc apply -f service-account-token-secret.yaml
```

4.

다음 명령을 실행하여 시크릿에서 서비스 계정 토큰을 가져옵니다.

```
$ oc get secret <sa_token_secret> -o jsonpath='{.data.token}' | base64 --decode ❶
```

출력 예

```

ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpoNnh5YnZhSE9QMkk3YnRZMV
ZocIFfQTZfRFp1YlUifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50liwia3Vi
ZXJuZXRlcy5pby9zZXJ2aWNIYWVWbnJ3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0liwia3ViZX
JuZXRlcy5pby9zZXJ2aWNIYWVWbnJ3VudC9zZW5yZXQubmFtZSI6ImJ1aWxkZXltdG9rZW
4tdHZrbnliLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWVWbnJ3Vu
dC5uYW1lIjoieVpibGRlcilslmt1YmVybWV0ZXMuaW8vc2VydmliZWZjY291bnQvc2Vyd
mljZS1hY2NvdW50LnVpZCI6IjNmZGU2MGZmLTA1NGYtNDkyZi04YzhjLTNlZjE0NDk3M
mFmNyIsInN1Yil6InN5c3RlbnR1b3VudC9zZXJ2aWNIYWVWbnJ3VudDpkZWZhdWx0OmJ1aWxkZXlif
Q.OmqFTDuMHC_IYvvEUrjr1x453hIEEHYcxS9VKSzmRkP1SiVZWPNPkTWIfNRp6bIUZD
3U6aN3N7dMNS0el5hu36xPgpKTdvuckKLTcnelMx6cxOdAbrcw1mCmOCINscwjS1KO
1kzMtYnnq8rXHiMJELsNIhnRyyIXRTtNBsy4t64T3283s3SLsancyx0gy0ujx-

```

```
Ch3uKAKdZi5iT-l8jnnQ-ds5THDs2h65RJhggIQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnvi3DG
n2XI-hZxl1yD2yGH1RBpYUHA
```

1

<sa_token_secret>을 서비스 토큰 시크릿의 이름으로 바꿉니다.

5.

서비스 계정 토큰을 사용하여 클러스터의 API로 인증합니다.

```
$ curl -X GET <openshift_cluster_api> --header "Authorization: Bearer <token>" 1
```

2

1

< ;openshift_cluster_api>를 OpenShift 클러스터 API로 바꿉니다.

2

& lt;token >을 이전 명령의 출력되는 서비스 계정 토큰으로 바꿉니다.

2.6.5. 보안이 포함된 서명된 인증서 사용 정보

서비스에 대한 통신을 보호하려면 프로젝트의 보안에 추가할 수 있는 서명된 제공 인증서/키 쌍을 생성하도록 **OpenShift Container Platform**을 구성하면 됩니다.

*서비스 제공 인증서 보안*은 즉시 사용 가능한 인증서가 필요한 복잡한 미들웨어 애플리케이션을 지원하기 위한 것입니다. 해당 설정은 관리자 툴에서 노드 및 마스터에 대해 생성하는 서버 인증서와 동일합니다.

서비스 **Pod** 사양은 서비스 제공 인증서 보안에 대해 구성됩니다.

```
apiVersion: v1
kind: Service
metadata:
  name: registry
```

annotations:

service.beta.openshift.io/serving-cert-secret-name: registry-cert **1**

...

1

인증서 이름을 지정합니다.

기타 Pod는 해당 Pod에 자동으로 마운트되는 `/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt` 파일의 CA 번들을 사용하여 내부 DNS 이름에만 서명되는 클러스터 생성 인증서를 신뢰할 수 있습니다.

이 기능의 서명 알고리즘은 `x509.SHA256WithRSA`입니다. 직접 교대하려면 생성된 보안을 삭제합니다. 새 인증서가 생성됩니다.

2.6.5.1. 보안과 함께 사용할 서명된 인증서 생성

Pod에 서명된 제공 인증서/키 쌍을 사용하려면 서비스를 생성하거나 편집하여 `service.beta.openshift.io/serving-cert-secret-name` 주석을 추가한 다음 Pod에 보안을 추가합니다.

절차

서비스 제공 인증서 보안을 생성하려면 다음을 수행합니다.

1. 서비스에 대한 Pod 사양을 편집합니다.
2. 시크릿에 사용하려는 이름과 함께 `service.beta.openshift.io/serving-cert-secret-name` 주석을 추가합니다.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: my-cert 1
spec:
  selector:
    app: MyApp
```

```
ports:
- protocol: TCP
  port: 80
  targetPort: 9376
```

인증서 및 키는 PEM 형식이며 각각 `tls.crt` 및 `tls.key`에 저장됩니다.

- 3. 서비스를 생성합니다.

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

- 4. 보안이 생성되었는지 확인합니다.

- a. 모든 보안 목록을 확인합니다.

```
$ oc get secrets
```

출력 예

NAME	TYPE	DATA	AGE
my-cert	kubernetes.io/tls	2	9m

- b. 보안에 대한 세부 정보를 확인합니다.

```
$ oc describe secret my-cert
```

출력 예

```
Name:      my-cert
Namespace: openshift-console
Labels:    <none>
Annotations: service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
             service.beta.openshift.io/originating-service-name: my-service
             service.beta.openshift.io/originating-service-uid: 640f0ec3-afc2-4380-bf31-
             a8c784846a11
```

```
service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
```

```
Type: kubernetes.io/tls
```

```
Data
```

```
====
```

```
tls.key: 1679 bytes
```

```
tls.crt: 2595 bytes
```

5.

해당 보안을 사용하여 **Pod** 사양을 편집합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: my-container
      mountPath: "/etc/my-path"
  volumes:
  - name: my-volume
    secret:
      secretName: my-cert
      items:
      - key: username
        path: my-group/my-username
        mode: 511
```

사용 가능한 경우 **Pod**가 실행됩니다. 인증서는 내부 서비스 **DNS** 이름인 **<service.name>**. **<service.namespace>.svc**에 적합합니다.

인증서/키 쌍은 만료 시기가 다가오면 자동으로 교체됩니다. 보안의 **service.beta.openshift.io/expiry** 주석에서 **RFC3339** 형식으로 된 만료 날짜를 확인합니다.



참고

대부분의 경우 서비스 **DNS** 이름 **<service.name>**. **<service.namespace>.svc**는 외부에서 라우팅할 수 없습니다. **<service.name>**. **<service.namespace>.svc**는 주로 클러스터 내 또는 서비스 내 통신과 경로 재암호화에 사용됩니다.

2.6.6. 보안 문제 해결

(서비스의 `service.beta.openshift.io/serving-cert-generation-error` 주석과 함께 서비스 인증서 생성이 실패하면 다음이 포함됩니다.

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

인증서를 생성한 서비스가 더 이상 존재하지 않거나 `serviceUID`가 다릅니다. 이전 보안을 제거하고 서비스에서 `service.beta.openshift.io/serving-cert-generation-error` , `service.beta.openshift.io/serving-cert-generation-error -num` 주석을 지워 인증서를 강제로 다시 생성해야 합니다.

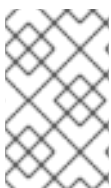
1. 보안을 삭제합니다.

```
$ oc delete secret <secret_name>
```

2. 주석을 지웁니다.

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```



참고

주석을 제거하는 명령에는 제거할 주석 이름 뒤에 `-`가 있습니다.

2.7. 구성 맵 생성 및 사용

다음 섹션에서는 구성 맵과 이를 생성하고 사용하는 방법을 정의합니다.

2.7.1. 구성 맵 이해

많은 애플리케이션에는 구성 파일, 명령줄 인수 및 환경 변수 조합을 사용하여 구성이 필요합니다. OpenShift Container Platform에서 컨테이너화된 애플리케이션을 이식하기 위해 이러한 구성 아티팩트는 이미지 콘텐츠와 분리됩니다.

ConfigMap 오브젝트는 컨테이너를 **OpenShift Container Platform**과 무관하게 유지하면서 구성 데이터를 사용하여 컨테이너를 삽입하는 메커니즘을 제공합니다. 구성 맵은 개별 속성 또는 전체 구성 파일 또는 **JSON Blob**과 같은 세분화된 정보를 저장하는 데 사용할 수 있습니다.

ConfigMap 오브젝트에는 **Pod**에서 사용하거나 컨트롤러와 같은 시스템 구성 요소의 구성 데이터를 저장하는 데 사용할 수 있는 구성 데이터의 키-값 쌍이 있습니다. 예를 들면 다음과 같습니다.

ConfigMap 오브젝트 정의

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: my-namespace
data: ①
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw ②
```

① ①

구성 데이터를 포함합니다.

②

UTF8이 아닌 데이터를 포함한 파일을 가리킵니다(예: 바이너리 **Java** 키 저장소 파일). **Base 64**에 파일 데이터를 입력합니다.



참고

이미지와 같은 바이너리 파일에서 구성 맵을 생성할 때 **binaryData** 필드를 사용할 수 있습니다.

다양한 방법으로 **Pod**에서 구성 데이터를 사용할 수 있습니다. 구성 맵을 다음과 같이 사용할 수 있습니다.

- 컨테이너에서 환경 변수 값 채우기
- 컨테이너에서 명령줄 인수 설정
- 볼륨에 구성 파일 채우기

사용자 및 시스템 구성 요소는 구성 데이터를 구성 맵에 저장할 수 있습니다.

구성 맵은 보안과 유사하지만 민감한 정보가 포함되지 않은 문자열 작업을 더 편리하게 지원하도록 설계되었습니다.

구성 맵 제한 사항

Pod에서 콘텐츠를 사용하기 전에 구성 맵을 생성해야 합니다.

컨트롤러는 누락된 구성 데이터를 허용하도록 작성할 수 있습니다. 상황에 따라 구성 맵을 사용하여 구성된 개별 구성 요소를 참조하십시오.

ConfigMap 오브젝트는 프로젝트에 있습니다.

동일한 프로젝트의 **Pod**에서만 참조할 수 있습니다.

Kubelet은 **API** 서버에서 가져오는 **Pod**에 대한 구성 맵만 지원합니다.

여기에는 **CLI**를 사용하거나 복제 컨트롤러에서 간접적으로 생성되는 모든 **Pod**가 포함됩니다. **OpenShift Container Platform** 노드의 **--manifest-url** 플래그, **--config** 플래그 또는 해당 **REST API**를 사용하여 생성한 **Pod**를 포함하지 않으며 이는 **Pod**를 생성하는 일반적인 방법이 아니기 때문입니다.

2.7.2. OpenShift Container Platform 웹 콘솔에서 구성 맵 생성

OpenShift Container Platform 웹 콘솔에서 구성 맵을 생성할 수 있습니다.

절차

- 클러스터 관리자로 구성 맵을 생성하려면 다음을 수행합니다.
 1. 관리자 관점에서 **Workloads** → **Config Maps**을 선택합니다.
 2. 페이지 오른쪽 상단에서 구성 맵 생성을 선택합니다.
 3. 구성 맵의 콘텐츠를 입력합니다.
 4. 생성을 선택합니다.
- 개발자로 구성 맵을 생성하려면 다음을 수행합니다.
 1. 개발자 관점에서 **Config Maps**을 선택합니다.
 2. 페이지 오른쪽 상단에서 구성 맵 생성을 선택합니다.
 3. 구성 맵의 콘텐츠를 입력합니다.
 4. 생성을 선택합니다.

2.7.3. CLI를 사용하여 구성 맵 생성

다음 명령을 사용하여 디렉토리, 특정 파일 또는 리터럴 값에서 구성 맵을 생성할 수 있습니다.

절차

- 구성 맵 생성:

```
$ oc create configmap <configmap_name> [options]
```

2.7.3.1. 디렉토리에서 구성 맵 생성

--from-file 플래그를 사용하여 디렉토리에서 구성 맵을 생성할 수 있습니다. 이 방법을 사용하면 디렉토리 내 여러 파일을 사용하여 구성 맵을 생성할 수 있습니다.

디렉토리의 각 파일은 구성 맵에서 키를 채우는 데 사용됩니다. 여기서 키 이름은 파일 이름이며 키 값은 파일의 내용입니다.

예를 들어 다음 명령은 **example-files** 디렉토리의 콘텐츠를 사용하여 구성 맵을 생성합니다.

```
$ oc create configmap game-config --from-file=example-files/
```

구성 맵에서 키를 표시합니다.

```
$ oc describe configmaps game-config
```

출력 예

```
Name:      game-config
Namespace: default
Labels:    <none>
Annotations: <none>

Data

game.properties: 158 bytes
ui.properties:   83 bytes
```

맵의 두 키가 명령에 지정된 디렉토리의 파일 이름에서 생성되는 것을 확인할 수 있습니다. 해당 키의 내용은 커질 수 있으므로 **oc describe**의 출력은 키와 크기의 이름만 표시합니다.

사전 요구 사항

- 구성 맵을 채우려는 데이터가 포함된 파일이 있는 디렉터리가 있어야 합니다.

다음 절차에서는 다음 예제 파일을 사용합니다. **game.properties** 및 **ui.properties**:

```
$ cat example-files/game.properties
```

출력 예

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

출력 예

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

절차

- 다음 명령을 입력하여 이 디렉토리에 있는 각 파일의 내용을 보유한 구성 맵을 생성합니다.

```
$ oc create configmap game-config \
  --from-file=example-files/
```

검증

- 키 값을 보려면 **-o** 옵션을 사용하여 오브젝트에 대한 **oc get** 명령을 입력합니다.

```
$ oc get configmaps game-config -o yaml
```

출력 예

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"
  selflink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

2.7.3.2. 파일에서 구성 맵 생성

--from-file 플래그를 사용하여 파일에서 구성 맵을 생성할 수 있습니다. **--from-file** 옵션을 CLI에 여러 번 전달할 수 있습니다.

key=value 표현식을 **--from-file** 옵션에 전달하여 파일에서 가져온 콘텐츠의 구성 맵에 설정할 키를 지정할 수도 있습니다. 예를 들면 다음과 같습니다.

```
$ oc create configmap game-config-3 --from-file=game-special-key=example-
files/game.properties
```



참고

파일에서 구성 맵을 생성하는 경우 **UTF8**이 아닌 데이터를 손상시키지 않고 이 필드에 배치된 **UTF8**이 아닌 데이터가 포함된 파일을 포함할 수 있습니다. **OpenShift Container Platform**에서는 바이너리 파일을 감지하고 파일을 **MIME**로 투명하게 인코딩합니다. 서버에서 **MIME** 페이로드는 데이터 손상 없이 디코딩되어 저장됩니다.

사전 요구 사항

- 구성 맵을 채우려는 데이터가 포함된 파일이 있는 디렉터리가 있어야 합니다.

다음 절차에서는 다음 예제 파일을 사용합니다. **game.properties** 및 **ui.properties**:

```
$ cat example-files/game.properties
```

출력 예

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UDDLRRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

출력 예

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

절차

- 특정 파일을 지정하여 구성 맵을 생성합니다.

```
$ oc create configmap game-config-2 \
  --from-file=example-files/game.properties \
  --from-file=example-files/ui.properties
```

- 키-값 쌍을 지정하여 구성 맵을 생성합니다.

```
$ oc create configmap game-config-3 \
  --from-file=game-special-key=example-files/game.properties
```

검증

- 파일에서 키 값을 확인하려면 **-o** 옵션을 사용하여 오브젝트에 대한 **oc get** 명령을 입력합니다.

```
$ oc get configmaps game-config-2 -o yaml
```

출력 예

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selflink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```


- 키-값 쌍의 키 값을 확인하려면 **-o** 옵션을 사용하여 오브젝트에 대한 **oc get** 명령을 입력합니다.

```
$ oc get configmaps game-config-3 -o yaml
```

출력 예

```
apiVersion: v1
data:
  game-special-key: |- ①
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  selflink: /api/v1/namespaces/default/configmaps/game-config-3
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985
```

①

이전 단계에서 설정한 키입니다.

2.7.3.3. 리터럴 값에서 구성 맵 생성

구성 맵에 리터럴 값을 제공할 수 있습니다.

--from-literal 옵션은 **key=value** 구문을 사용하므로 명령줄에서 직접 리터럴 값을 제공할 수 있습니다.

절차

- 리터럴 값을 지정하여 구성 맵을 생성합니다.

```
$ oc create configmap special-config \
  --from-literal=special.how=very \
  --from-literal=special.type=charm
```

검증

- 키 값을 보려면 **-o** 옵션을 사용하여 오브젝트에 대한 **oc get** 명령을 입력합니다.

```
$ oc get configmaps special-config -o yaml
```

출력 예

```
apiVersion: v1
data:
  special.how: very
  special.type: charm
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: special-config
  namespace: default
  resourceVersion: "651"
  selflink: /api/v1/namespaces/default/configmaps/special-config
  uid: dadce046-d673-11e5-8cd0-68f728db1985
```

2.7.4. 사용 사례: Pod에서 구성 맵 사용

다음 섹션에서는 Pod에서 **ConfigMap** 오브젝트를 사용할 때 몇 가지 사용 사례에 대해 설명합니다.

2.7.4.1. 구성 맵을 사용하여 컨테이너에서 환경 변수 채우기

구성 맵을 사용하여 컨테이너에서 개별 환경 변수를 채우거나 유효한 환경 변수 이름을 형성하는 모든 키에서 컨테이너의 환경 변수를 채울 수 있습니다.

예를 들어 다음 구성 맵을 고려하십시오.

두 개의 환경 변수가 있는 ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config 1
  namespace: default 2
data:
  special.how: very 3
  special.type: charm 4
```

1

구성 맵의 이름입니다.

2

구성 맵이 있는 프로젝트입니다. 구성 맵은 동일한 프로젝트의 **Pod**에서만 참조할 수 있습니다.

3 4

삽입할 환경 변수입니다.

하나의 환경 변수가 있는 ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config 1
  namespace: default
data:
  log_level: INFO 2
```

1

구성 맵의 이름입니다.

2

삽입할 환경 변수입니다.

절차

-

configMapKeyRef 섹션을 사용하여 Pod에서 이 **ConfigMap**의 키를 사용할 수 있습니다.

특정 환경 변수를 삽입하도록 구성된 샘플 Pod 사양

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env: 1
    - name: SPECIAL_LEVEL_KEY 2
      valueFrom:
        configMapKeyRef:
          name: special-config 3
          key: special.how 4
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config 5
          key: special.type 6
          optional: true 7
    envFrom: 8
    - configMapRef:
        name: env-config 9
  restartPolicy: Never

```

1

ConfigMap에서 지정된 환경 변수를 가져오는 스탠자입니다.

2

키 값을 삽입하는 **pod** 환경 변수의 이름입니다.

3 5

특정 환경 변수를 끌어올 **ConfigMap**의 이름입니다.

4 6

ConfigMap에서 가져올 환경 변수입니다.

7

환경 변수를 선택적으로 만듭니다. 선택 사항으로 지정된 **ConfigMap** 및 키가 없는 경우에도 **Pod**가 시작됩니다.

8

ConfigMap에서 모든 환경 변수를 가져오는 스탠자입니다.

9

모든 환경 변수를 가져올 **ConfigMap**의 이름입니다.

이 **Pod**가 실행되면 **Pod** 로그에 다음 출력이 포함됩니다.

```
SPECIAL_LEVEL_KEY=very
log_level=INFO
```

참고

SPECIAL_TYPE_KEY=charm은 예제 출력에 나열되지 않습니다. **optional: true**가 설정되어 있기 때문입니다.

2.7.4.2. 구성 맵을 사용하여 컨테이너 명령에 대한 명령줄 인수 설정

구성 맵을 사용하여 **Kubernetes** 대체 구문 **\$(VAR_NAME)** 을 사용하여 컨테이너에서 명령 또는 인수 값을 설정할 수 있습니다.

예를 들어 다음 구성 맵을 고려하십시오.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

절차

- 컨테이너의 명령에 값을 삽입하려면 환경 변수로 사용할 키를 사용해야 합니다. 그런 다음 `$(VAR_NAME)` 구문을 사용하여 컨테이너의 명령에서 참조할 수 있습니다.

특정 환경 변수를 삽입하도록 구성된 샘플 Pod 사양

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY)
$(SPECIAL_TYPE_KEY)" ] 1
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
      restartPolicy: Never
```

1

환경 변수로 사용할 키를 사용하여 컨테이너의 명령에 값을 삽입합니다.

이 Pod가 실행되면 **test-container** 컨테이너에서 실행되는 **echo** 명령의 출력은 다음과 같습니다.

```
very charm
```

2.7.4.3. 구성 맵을 사용하여 볼륨에 콘텐츠 삽입

구성 맵을 사용하여 볼륨에 콘텐츠를 삽입할 수 있습니다.

ConfigMap CR(사용자 정의 리소스)의 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

절차

구성 맵을 사용하여 볼륨에 콘텐츠를 삽입하는 몇 가지 다른 옵션이 있습니다.

- 구성 맵을 사용하여 콘텐츠를 볼륨에 삽입하는 가장 기본적인 방법은 키가 파일 이름이고 파일의 콘텐츠가 키의 값인 파일로 볼륨을 채우는 것입니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/special.how" ]
```

```

volumeMounts:
  - name: config-volume
    mountPath: /etc/config
volumes:
  - name: config-volume
    configMap:
      name: special-config ❶
restartPolicy: Never
    
```

❶

키가 포함된 파일입니다.

이 Pod가 실행되면 `cat` 명령의 출력은 다음과 같습니다.

very

-

구성 맵 키가 프로젝션되는 볼륨 내의 경로를 제어할 수도 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/path/to/special-key" ]
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
          - key: special.how
            path: path/to/special-key ❶
restartPolicy: Never
    
```

❶

구성 맵 키 경로입니다.

이 Pod가 실행되면 `cat` 명령의 출력은 다음과 같습니다.

very

2.8. 장치 플러그인을 사용하여 POD가 있는 외부 리소스에 액세스

장치 플러그인을 사용하면 사용자 정의 코드를 작성하지 않고도 **OpenShift Container Platform Pod**에서 특정 장치 유형(**GPU, InfiniBand** 또는 벤더별 초기화 및 설정이 필요한 기타 유사한 컴퓨팅 리소스)을 사용할 수 있습니다.

2.8.1. 장치 플러그인 이해

장치 플러그인은 클러스터 전체에서 하드웨어 장치를 소비할 수 있는 일관되고 이식 가능한 솔루션을 제공합니다. 장치 플러그인은 확장 메커니즘을 통해 이러한 장치를 지원하여 컨테이너에서 이러한 장치를 사용할 수 있게 하고 장치의 상태 점검을 제공하며 안전하게 공유합니다.



중요

OpenShift Container Platform은 장치 플러그인 **API**를 지원하지만 장치 플러그인 컨테이너는 개별 공급 업체에서 지원합니다.

장치 플러그인은 특정 하드웨어 리소스를 관리하는 노드(**kubelet**외부)에서 실행되는 **gRPC** 서비스입니다. 모든 장치 플러그인은 다음 원격 프로시저 호출 (**RPC**)을 지원해야 합니다.

```
service DevicePlugin {
  // GetDevicePluginOptions returns options to be communicated with Device
  // Manager
  rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

  // ListAndWatch returns a stream of List of Devices
  // Whenever a Device state change or a Device disappears, ListAndWatch
  // returns the new list
  rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

  // Allocate is called during container creation so that the Device
  // Plug-in can run device specific operations and instruct Kubelet
  // of the steps to make the Device available in the container
  rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

  // PreStartcontainer is called, if indicated by Device Plug-in during
  // registration phase, before each container start. Device plug-in
  // can run device specific operations such as resetting the device
  // before making devices available to the container
  rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

장치 플러그인 예

- [COS 기반 운영 체제 용 NVIDIA GPU 장치 플러그인](#)
- [NVIDIA 공식 GPU 장치 플러그인](#)
- [Solarflare 장치 플러그인](#)
- [KubeVirt 장치 플러그인: vfio 및 kvm](#)
- [Kubernetes 장치 플러그인\(CEX\) 카드용 Kubernetes 장치 플러그인](#)



참고

간편한 장치 플러그인 참조 구현을 위해 장치 관리자 코드에 [vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go](https://github.com/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go)의 스텝 장치 플러그인이 있습니다.

2.8.1.1. 장치 플러그인을 배포하는 방법

- 장치 플러그인 배포에는 데몬 세트 접근 방식을 사용하는 것이 좋습니다.
- 시작 시 장치 플러그인은 노드의 `/var/lib/kubelet/device-plugin/`에 UNIX 도메인 소켓을 만들어 장치 관리자의 **RPC**를 제공하려고 합니다.
- 장치 플러그인은 하드웨어 리소스, 호스트 파일 시스템에 대한 액세스 및 소켓 생성을 관리해야 하므로 권한 있는 보안 컨텍스트에서 실행해야 합니다.
- 배포 단계에 대한 보다 구체적인 세부 정보는 각 장치 플러그인 구현에서 확인할 수 있습니다.

2.8.2. 장치 관리자 이해

장치 관리자는 장치 플러그인이라는 플러그인을 사용하여 특수 노드 하드웨어 리소스를 알리기 위한 메커니즘을 제공합니다.

업스트림 코드 변경없이 특수 하드웨어를 공개할 수 있습니다.



중요

OpenShift Container Platform은 장치 플러그인 **API**를 지원하지만 장치 플러그인 컨테이너는 개별 공급 업체에서 지원합니다.

장치 관리자는 장치를 확장 리소스(**Extended Resources**)으로 공개합니다. 사용자 **pod**는 다른 확장 리소스를 요청하는 데 사용되는 동일한 제한/요청 메커니즘을 사용하여 장치 관리자에 의해 공개된 장치를 사용할 수 있습니다.

시작시 장치 플러그인은 `/var/lib/kubelet/device-plugins/kubelet.sock`에서 **Register**를 호출하는 장치 관리자에 직접 등록하고 장치 관리자 요청을 제공하기 위해 `/var/lib/kubelet/device-plugins/<plugin>.sock`에서 **gRPC** 서비스를 시작합니다.

장치 관리자는 새 등록 요청을 처리하는 동안 장치 플러그인 서비스에서 **ListAndWatch** 원격 프로시저 호출(**RPC**)을 호출합니다. 이에 대한 응답으로 장치 관리자는 플러그인에서 **gRPC** 스트림을 통해 장치 오브젝트 목록을 가져옵니다. 장치 관리자는 플러그인에서 새로운 업데이트를 위해 스트림을 모니터링합니다. 플러그인 측에서도 플러그인은 스트림을 열린 상태로 유지하고 장치 상태가 변경될 때마다 동일한 스트리밍 연결을 통해 새 장치 목록이 장치 관리자로 전송됩니다.

새로운 **pod** 승인 요청을 처리하는 동안 **Kubelet**은 장치 할당을 위해 요청된 **Extended Resources**를 장치 관리자에게 전달합니다. 장치 관리자는 데이터베이스에서 해당 플러그인이 존재하는지 확인합니다. 플러그인이 존재하고 로컬 캐시별로 할당 가능한 장치가 있는 경우 **Allocate RPC**가 특정 장치 플러그인에서 호출됩니다.

또한 장치 플러그인은 드라이버 설치, 장치 초기화 및 장치 재설정과 같은 몇 가지 다른 장치 관련 작업을 수행할 수도 있습니다. 이러한 기능은 구현마다 다릅니다.

2.8.3. 장치 관리자 활성화

장치 관리자는 장치 플러그인을 구현하여 업스트림 코드 변경없이 특수 하드웨어를 사용할 수 있습니다.

장치 관리자는 장치 플러그인이라는 플러그인을 사용하여 특수 노드 하드웨어 리소스를 알리기 위한 메커니즘을 제공합니다.

1. 다음 명령을 입력하여 구성할 노드 유형의 정적 **MachineConfigPool CRD**와 연관된 라벨을 가져옵니다. 다음 중 하나를 실행합니다.

- a. **Machine config**를 표시합니다:

```
# oc describe machineconfig <name>
```

예를 들면 다음과 같습니다.

```
# oc describe machineconfig 00-worker
```

출력 예

```
Name:      00-worker
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker 1
```

1

장치 관리자에 필요한 라벨입니다.

프로세스

1. 구성 변경을 위한 사용자 정의 리소스 (**CR**)를 만듭니다.

장치 관리자 **CR**의 설정 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr 2
```

```
kubeletConfig:
  feature-gates:
    - DevicePlugins=true 3
```

1

CR에 이름을 지정합니다.

2

Machine Config Pool에서 라벨을 입력합니다.

3

DevicePlugins를 'true'로 설정합니다.

2.

장치 관리자를 만듭니다.

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

출력 예

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3.

노드에서 `/var/lib/kubelet/device-plugins/kubelet.sock`이 작성되었는지 확인하여 장치 관리자가 실제로 사용 가능한지 확인합니다. 이는 장치 관리자의 gRPC 서버가 새 플러그인 등록을 수신하는 UNIX 도메인 소켓입니다. 이 소켓 파일은 장치 관리자가 활성화된 경우에만 Kubelet을 시작할 때 생성됩니다.

2.9. POD 예약 결정에 POD 우선순위 포함

클러스터에서 Pod 우선순위 및 선점을 활성화할 수 있습니다. Pod 우선순위는 다른 Pod와 관련된 Pod의 중요성을 나타내며 해당 우선순위를 기반으로 Pod를 대기열에 넣습니다. Pod 선점을 통해 클러스터에서 우선순위가 낮은 Pod를 제거하거나 선점할 수 있으므로 적절한 노드 Pod 우선순위에 사용 가능

한 공간이 없는 경우 우선순위가 높은 **Pod**를 예약할 수 있습니다. **Pod Pod** 우선순위는 **Pod**의 스케줄링 순서에도 영향을 미치지 않습니다.

우선순위 및 선점 기능을 사용하려면 **Pod**의 상대적 가중치를 정의하는 우선순위 클래스를 생성합니다. 그런 다음 **Pod** 사양의 우선순위 클래스를 참조하여 예약에 해당 값을 적용합니다.

2.9.1. Pod 우선순위 이해

Pod 우선순위 및 선점 기능을 사용하면 스케줄러에서 보류 중인 **Pod**를 우선순위에 따라 정렬하고, 보류 중인 **Pod**는 예약 큐에서 우선순위가 더 낮은 다른 대기 중인 **Pod**보다 앞에 배치됩니다. 그 결과 예약 요구 사항이 충족되는 경우 우선순위가 높은 **Pod**가 우선순위가 낮은 **Pod**보다 더 빨리 예약될 수 있습니다. **Pod**를 예약할 수 없는 경우에는 스케줄러에서 우선순위가 낮은 다른 **Pod**를 계속 예약합니다.

2.9.1.1. Pod 우선순위 클래스

네임스페이스가 지정되지 않은 오브젝트로서 이름에서 우선순위 정수 값으로의 매핑을 정의하는 우선순위 클래스를 **Pod**에 할당할 수 있습니다. 값이 클수록 우선순위가 높습니다.

우선순위 클래스 오브젝트에는 1000000000(10억)보다 작거나 같은 32비트 정수 값을 사용할 수 있습니다. 선점하거나 제거해서는 안 되는 중요한 **Pod**의 경우 10억보다 크거나 같은 숫자를 예약합니다. 기본적으로 **OpenShift Container Platform**에는 중요한 시스템 **Pod**의 예약을 보장하기 위해 우선순위 클래스가 2개 예약되어 있습니다.

```
$ oc get priorityclasses
```

출력 예

NAME	VALUE	GLOBAL-DEFAULT	AGE
system-node-critical	2000001000	false	72m
system-cluster-critical	2000000000	false	72m
openshift-user-critical	1000000000	false	3d13h
cluster-logging	1000000	false	29s

-

system-node-critical - 이 우선순위 클래스의 값은 2000001000이며 노드에서 제거해서는 안 되는 모든 **Pod**에 사용됩니다. 이 우선순위 클래스가 있는 **Pod**의 예로는 **sdn-ovs**, **sdn** 등이 있습니다. 대다수의 중요한 구성 요소에는 기본적으로 **system-node-critical** 우선순위 클래스가 포함됩니다. 예를 들면 다음과 같습니다.

- **master-api**
- **master-controller**
- **master-etcd**
- **sdn**
- **sdn-ovs**
- **sync**
- **system-cluster-critical** - 이 우선순위 클래스의 값은 2000000000(10억)이며 클러스터에 중요한 Pod에 사용합니다. 이 우선순위 클래스가 있는 Pod는 특정 상황에서 노드에서 제거할 수 있습니다. 예를 들어 **system-node-critical** 우선순위 클래스를 사용하여 구성된 Pod가 우선할 수 있습니다. 그러나 이 우선순위 클래스는 예약을 보장합니다. 이 우선순위 클래스를 사용할 수 있는 Pod의 예로는 **fluentd**, **Descheduler**와 같은 추가 기능 구성 요소 등이 있습니다. 대다수의 중요한 구성 요소에는 기본적으로 **system-cluster-critical** 우선순위 클래스가 포함됩니다. 예를 들면 다음과 같습니다.
 - **fluentd**
 - **metrics-server**
 - **Descheduler**
- **openshift-user-critical** - 리소스 사용을 바인딩할 수 없고 리소스 소비 동작이 없는 중요한 Pod와 함께 **priorityClassName** 필드를 사용할 수 있습니다. **openshift-monitoring** 및 **openshift-user-workload-monitoring** 네임스페이스의 **Prometheus Pod**는 **openshift-user-critical priorityClassName** 을 사용합니다. 워크로드 모니터링은 첫 번째 **priorityClass** 로 **system-critical** 을 사용하지만 모니터링 시 과도한 메모리를 사용하며 노드가 제거할 수 없습니다. 결과적으로 모니터링은 스케줄러 유연성을 제공하기 위해 우선 순위가 줄어들어 중요한 노드의 작동을 유지하기 위해 많은 워크로드를 이동합니다.
-

cluster-logging - 이 우선순위는 **Fluentd**에서 **Fluentd Pod**가 다른 앱보다 먼저 노드에 예약되도록 하는 데 사용됩니다.

2.9.1.2. Pod 우선순위 이름

우선순위 클래스가 한 개 이상 있으면 **Pod** 사양에서 우선순위 클래스 이름을 지정하는 **Pod**를 생성할 수 있습니다. 우선순위 승인 컨트롤러는 우선순위 클래스 이름 필드를 사용하여 정수 값으로 된 우선순위를 채웁니다. 이름이 지정된 우선순위 클래스가 없는 경우 **Pod**가 거부됩니다.

2.9.2. Pod 선점 이해

개발자가 **Pod**를 생성하면 **Pod**가 큐로 들어갑니다. 개발자가 **Pod**에 **Pod** 우선순위 또는 선점을 구성한 경우 스케줄러는 큐에서 **Pod**를 선택하고 해당 **Pod**를 노드에 예약하려고 합니다. 스케줄러가 노드에서 **Pod**의 지정된 요구 사항을 모두 충족하는 적절한 공간을 찾을 수 없는 경우 보류 중인 **Pod**에 대한 선점 논리가 트리거됩니다.

스케줄러가 노드에서 **Pod**를 하나 이상 선점하면 우선순위가 높은 **Pod** 사양의 **nominatedNodeName** 필드가 **nodename** 필드와 함께 노드의 이름으로 설정됩니다. 스케줄러는 **nominatedNodeName** 필드를 사용하여 **Pod**용으로 예약된 리소스를 계속 추적하고 클러스터의 선점에 대한 정보도 사용자에게 제공합니다.

스케줄러에서 우선순위가 낮은 **Pod**를 선점한 후에는 **Pod**의 정상 종료 기간을 따릅니다. 스케줄러에서 우선순위가 낮은 **Pod**가 종료되기를 기다리는 동안 다른 노드를 사용할 수 있게 되는 경우 스케줄러는 해당 노드에서 우선순위가 더 높은 **Pod**를 예약할 수 있습니다. 따라서 **Pod** 사양의 **nominatedNodeName** 필드 및 **nodeName** 필드가 다를 수 있습니다.

또한 스케줄러가 노드의 **Pod**를 선점하고 종료되기를 기다리고 있는 상태에서 보류 중인 **Pod**보다 우선순위가 높은 **Pod**를 예약해야 하는 경우, 스케줄러는 우선순위가 더 높은 **Pod**를 대신 예약할 수 있습니다. 이러한 경우 스케줄러는 보류 중인 **Pod**의 **nominatedNodeName**을 지워 해당 **Pod**를 다른 노드에 사용할 수 있도록 합니다.

선점을 수행해도 우선순위가 낮은 **Pod**가 노드에서 모두 제거되는 것은 아닙니다. 스케줄러는 우선순위가 낮은 **Pod**의 일부를 제거하여 보류 중인 **Pod**를 예약할 수 있습니다.

스케줄러는 노드에서 보류 중인 **Pod**를 예약할 수 있는 경우에만 해당 노드에서 **Pod** 선점을 고려합니다.

2.9.2.1. 선점되지 않은 우선 순위 클래스

선점 정책이 **Never**로 설정된 **Pod**는 예약 큐에서 우선순위가 낮은 **Pod**보다 앞에 배치되지만 다른 **Pod**는 선점할 수 없습니다. 예약 대기 중인 비 선점 **Pod**는 사용 가능한 리소스가 충분하고 해당 **Pod**를 예약할 수 있을 때까지 예약 큐에 남아 있습니다. 비 선점 **Pod**는 다른 **Pod**와 마찬가지로 스케줄러 백오프의 영향을 받습니다. 즉 스케줄러에서 이러한 **Pod**를 예약하지 못하면 더 낮은 빈도로 다시 예약을 시도하여 우선순위가 더 낮은 기타 **Pod**를 먼저 예약할 수 있습니다.

비 선점 **Pod**는 우선순위가 높은 다른 **Pod**에서 계속 선점할 수 있습니다.

2.9.2.2. Pod 선점 및 기타 스케줄러 설정

Pod 우선순위 및 선점 기능을 활성화하는 경우 다른 스케줄러 설정을 고려하십시오.

Pod 우선순위 및 Pod 중단 예산

Pod 중단 예산은 동시에 작동해야 하는 최소 복제본 수 또는 백분율을 지정합니다. **Pod** 중단 예산을 지정하면 **Pod**를 최적의 노력 수준에서 선점할 때 **OpenShift Container Platform**에서 해당 예산을 준수합니다. 스케줄러는 **Pod** 중단 예산을 위반하지 않고 **Pod**를 선점하려고 합니다. 이러한 **Pod**를 찾을 수 없는 경우 **Pod** 중단 예산 요구 사항과 관계없이 우선순위가 낮은 **Pod**를 선점할 수 있습니다.

Pod 우선순위 및 Pod 유사성

Pod 유사성을 위해서는 동일한 라벨이 있는 다른 **Pod**와 같은 노드에서 새 **Pod**를 예약해야 합니다.

노드에서 우선순위가 낮은 하나 이상의 **Pod**와 보류 중인 **Pod**에 **Pod** 간 유사성이 있는 경우 스케줄러는 선호도 요구 사항을 위반하지 않고 우선순위가 낮은 **Pod**를 선점할 수 없습니다. 이 경우 스케줄러는 보류 중인 **Pod**를 예약할 다른 노드를 찾습니다. 그러나 스케줄러에서 적절한 노드를 찾을 수 있다는 보장이 없고 보류 중인 **Pod**가 예약되지 않을 수 있습니다.

이러한 상황을 방지하려면 우선순위가 같은 **Pod**를 사용하여 **Pod** 유사성을 신중하게 구성합니다.

2.9.2.3. 선점된 Pod의 정상 종료

Pod를 선점할 때 스케줄러는 **Pod**의 정상 종료 기간이 만료될 때까지 대기하여 **Pod**가 작동을 완료하고 종료할 수 있도록 합니다. 시간이 지난 후에도 **Pod**가 종료되지 않으면 스케줄러에서 **Pod**를 종료합니다. 이러한 정상 종료 기간으로 인해 스케줄러에서 **Pod**를 선점하는 시점과 노드에서 보류 중인 **Pod**를 예약할 수 있는 시간 사이에 시차가 발생합니다.

이 간격을 최소화하려면 우선순위가 낮은 **Pod**의 정상 종료 기간을 짧게 구성하십시오.

2.9.3. 우선순위 및 선점 구성

Pod 사양에 `priorityClassName` 을 사용하여 우선순위 클래스 오브젝트를 생성하고 Pod를 우선순위에 연결하여 우선순위 및 선점을 적용합니다.



참고

예약된 기존 **Pod**에 우선순위 클래스를 직접 추가할 수 없습니다.

프로세스

우선순위 및 선점을 사용하도록 클러스터를 구성하려면 다음을 수행합니다.

1. 우선순위 클래스를 한 개 이상 생성합니다.
 - a. 다음과 유사한 **YAML** 파일을 생성합니다.

```

apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority ①
  value: 1000000 ②
  preemptionPolicy: PreemptLowerPriority ③
  globalDefault: false ④
  description: "This priority class should be used for XYZ service pods only." ⑤

```

①

우선순위 클래스 오브젝트의 이름입니다.

②

오브젝트의 우선순위 값입니다.

③

선택 사항: 이 우선순위 클래스가 선점인지 또는 선점되지 않는지 여부를 지정합니다. 선점 정책의 기본값은 **PreemptLowerPriority**로, 해당 우선순위 클래스의 **Pod**에서 우선순위가 낮은 **Pod**를 선점할 수 있습니다. 선점 정책이 **Never**로 설정된 경우 해당 우선순위 클래스의 **Pod**는 선점하지 않습니다.

④

5

선택 사항: 개발자가 이 우선순위 클래스와 함께 사용해야 하는 Pod를 설명합니다. 임의의 텍스트 문자열을 입력합니다.

b.

우선순위 클래스를 생성합니다.

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

2.

우선순위 클래스의 이름을 포함할 Pod 사양을 생성합니다.

a.

다음과 유사한 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
    priorityClassName: high-priority 1
```

1

이 Pod에 사용할 우선순위 클래스를 지정합니다.

b.

Pod를 생성합니다.

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

Pod 구성 또는 Pod 템플릿에 우선순위 이름을 직접 추가할 수 있습니다.

2.10. 노드 선택기를 사용하여 특정 노드에 POD 배치

노드 선택기는 키-값 쌍으로 구성된 맵을 지정합니다. 규칙은 노드의 사용자 정의 라벨과 Pod에 지정된 선택기를 사용하여 정의합니다.

Pod를 노드에서 실행하려면 Pod에 노드의 라벨로 표시된 키-값 쌍이 있어야 합니다.

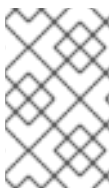
동일한 Pod 구성에서 노드 유사성 및 노드 선택기를 사용하는 경우 아래의 중요 고려 사항을 참조하십시오.

2.10.1. 노드 선택기를 사용하여 Pod 배치 제어

Pod의 노드 선택기와 노드의 라벨을 사용하여 Pod가 예약되는 위치를 제어할 수 있습니다. 노드 선택기를 사용하면 OpenShift Container Platform에서 일치하는 라벨이 포함된 노드에 Pod를 예약합니다.

노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 머신 세트에 레이블을 추가하면 노드 또는 머신이 중단되는 경우 새 노드에 라벨이 지정됩니다. 노드 또는 머신이 중단된 경우 노드 또는 머신 구성에 추가된 라벨이 유지되지 않습니다.

기존 Pod에 노드 선택기를 추가하려면 ReplicaSet 오브젝트, DaemonSet 오브젝트, StatefulSet 오브젝트, Deployment 오브젝트 또는 DeploymentConfig 오브젝트와 같이 해당 Pod의 제어 오브젝트에 노드 선택기를 추가합니다. 이 제어 오브젝트 아래의 기존 Pod는 라벨이 일치하는 노드에서 재생성됩니다. 새 Pod를 생성하는 경우 Pod 사양에 노드 선택기를 직접 추가할 수 있습니다. Pod에 제어 오브젝트가 없는 경우 Pod를 삭제하고 Pod 사양을 편집하고 Pod를 다시 생성해야 합니다.



참고

예약된 기존 Pod에 노드 선택기를 직접 추가할 수 없습니다.

사전 요구 사항

기존 Pod에 노드 선택기를 추가하려면 해당 Pod의 제어 오브젝트를 결정하십시오. 예를 들어 router-default-66d5cf9464-m2g75 Pod는 router-default-66d5cf9464 복제본 세트에서 제어합니다.

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

출력 예

```
kind: Pod
```

```

apiVersion: v1
metadata:
#...
Name:          router-default-66d5cf9464-7pwkc
Namespace:     openshift-ingress
# ...
Controlled By: ReplicaSet/router-default-66d5cf9464
# ...

```

웹 콘솔에서 Pod YAML의 `ownerReferences` 아래에 제어 오브젝트가 나열됩니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
- apiVersion: apps/v1
  kind: ReplicaSet
  name: router-default-66d5cf9464
  uid: d81dd094-da26-11e9-a48a-128e7edf0312
  controller: true
  blockOwnerDeletion: true
# ...

```

절차

1.
 1. 컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 노드에 라벨을 추가합니다.
 - 1. 노드를 생성할 때 컴퓨팅 머신 세트에서 관리하는 노드에 라벨을 추가하려면 **MachineSet** 오브젝트를 사용합니다.
 - a.
 1. 다음 명령을 실행하여 **MachineSet** 오브젝트에 라벨을 추가합니다.

```

$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"
<key>":"<value>"},"<key>":"<value>"}]}' -n openshift-machine-api

```

예를 들면 다음과 같습니다.

```
$ oc patch MachineSet abc612-msrtw-worker-us-east-1c --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":
{"type":"user-node","region":"east"}}]' -n openshift-machine-api
```

작은 정보

또는 다음 **YAML**을 적용하여 컴퓨팅 머신 세트에 라벨을 추가할 수 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: xf2bd-infra-us-east-2a
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
#...
```

- b. **oc edit** 명령을 사용하여 라벨이 **MachineSet** 오브젝트에 추가되었는지 확인합니다.

예를 들면 다음과 같습니다.

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-
api
```

MachineSet 오브젝트의 예

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet

# ...

spec:
# ...
  template:
    metadata:
# ...
    spec:
      metadata:
        labels:
```

```

region: east
type: user-node
# ...

```

- 라벨을 노드에 직접 추가합니다.

- a. 노드의 **Node** 오브젝트를 편집합니다.

```
$ oc label nodes <name> <key>=<value>
```

예를 들어 노드에 라벨을 지정하려면 다음을 수행합니다.

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

작은 정보

다음 **YAML**을 적용하여 노드에 라벨을 추가할 수 있습니다.

```

kind: Node
apiVersion: v1
metadata:
  name: hello-node-6fbccf8d9
labels:
  type: "user-node"
  region: "east"
#...

```

- b. 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc get nodes -l type=user-node,region=east
```

출력 예

```

NAME                                STATUS ROLES AGE VERSION
ip-10-0-142-25.ec2.internal Ready worker 17m v1.25.0

```

2.

Pod에 일치하는 노드 선택기를 추가합니다.

- 기존 및 향후 **Pod**에 노드 선택기를 추가하려면 **Pod**의 제어 오브젝트에 노드 선택기를 추가합니다.

라벨이 있는 **ReplicaSet** 오브젝트의 예

```
kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
  template:
    metadata:
      creationTimestamp: null
    labels:
      ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
      pod-template-hash: 66d5cf9464
    spec:
      nodeSelector:
        kubernetes.io/os: linux
        node-role.kubernetes.io/worker: "
        type: user-node ①
#...
```

①

노드 선택기를 추가합니다.

- 특정 새 **Pod**에 노드 선택기를 추가하려면 선택기를 **Pod** 오브젝트에 직접 추가합니다.

노드 선택기가 있는 **Pod** 오브젝트의 예


```
apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...
```



참고

예약된 기존 **Pod**에 노드 선택기를 직접 추가할 수 없습니다.

3장. CUSTOM METRICS AUTOSCALER OPERATOR를 사용하여 POD 자동 스케일링

3.1. 릴리스 노트

3.1.1. 사용자 정의 Metrics Autoscaler Operator 릴리스 정보

Custom Metrics Autoscaler Operator for Red Hat OpenShift의 릴리스 노트는 새로운 기능 및 개선 사항, 더 이상 사용되지 않는 기능 및 알려진 문제에 대해 설명합니다.

Custom Metrics Autoscaler Operator는 Kubernetes 기반 Event driven Autoscaler(KEDA)를 사용하며 OpenShift Container Platform HPA(수평 Pod 자동 스케일러) 상단에 빌드됩니다.



참고

Custom Metrics Autoscaler Operator for Red Hat OpenShift는 핵심 OpenShift Container Platform과 별도의 릴리스 사이클로 설치 가능한 구성 요소로 제공됩니다. [Red Hat OpenShift Container Platform 라이프 사이클 정책](#)은 릴리스 호환성에 대해 간단히 설명합니다.

3.1.1.1. 지원되는 버전

다음 표는 각 OpenShift Container Platform 버전에 대한 Custom Metrics Autoscaler Operator 버전을 정의합니다.

버전	OpenShift Container Platform 버전	정식 출시일 (GA)
2.13.1	4.16	정식 출시일 (GA)
2.13.1	4.15	정식 출시일 (GA)
2.13.1	4.14	정식 출시일 (GA)
2.13.1	4.13	정식 출시일 (GA)
2.13.1	4.12	정식 출시일 (GA)

3.1.1.2. 사용자 정의 지표 Autoscaler Operator 2.13.1 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.13.1-421 릴리스는 OpenShift Container Platform 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. [RHBA-2024:4837](#)에 대

해 다음 권고를 사용할 수 있습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 **Kubernetes** 기반 이벤트 기반 자동 스케일러(**KEDA**)의 커뮤니티 지원 버전을 제거합니다.

3.1.1.2.1. 새로운 기능 및 개선 사항

3.1.1.2.1.1. Custom Metrics Autoscaler Operator를 사용하여 사용자 정의 인증서 지원

Custom Metrics Autoscaler Operator는 사용자 정의 서비스 **CA** 인증서를 사용하여 외부 **Kafka** 클러스터 또는 외부 **Prometheus** 서비스와 같은 **TLS** 사용 지표 소스에 안전하게 연결할 수 있습니다. 기본적으로 **Operator**는 자동으로 생성된 서비스 인증서를 사용하여 클러스터 내 서비스에만 연결합니다. **KedaController** 오브젝트에는 구성 맵을 사용하여 외부 서비스에 연결하기 위한 사용자 정의 서버 **CA** 인증서를 로드할 수 있는 새 필드가 있습니다.

자세한 내용은 [사용자 정의 지표 자동 스케일러의 사용자 정의 CA 인증서를 참조하십시오.](#)

3.1.1.2.2. 버그 수정

- 이전에는 **custom-metrics-autoscaler** 및 **custom-metrics-autoscaler-adapter** 이미지에 시간대 정보가 누락되었습니다. 결과적으로 컨트롤러가 시간대 정보를 찾을 수 없기 때문에 **cron** 트리거를 사용하여 확장 개체가 작동하지 않았습니다. 이번 수정을 통해 시간대 정보를 포함하도록 이미지 빌드가 업데이트됩니다. 결과적으로 **cron** 트리거를 포함하는 확장 오브젝트가 올바르게 작동합니다. 현재 **cron** 트리거를 포함하는 확장 오브젝트는 사용자 정의 메트릭 자동 스케일러에서 지원되지 않습니다. ([OCPBUGS-34018](#))

3.1.2. Custom Metrics Autoscaler Operator의 이전 릴리스 릴리스 노트

다음 릴리스 노트는 이전 버전의 **Custom Metrics Autoscaler Operator**에 대한 것입니다.

현재 버전의 경우 [Custom Metrics Autoscaler Operator 릴리스 노트](#)를 참조하십시오.

3.1.2.1. 사용자 정의 지표 Autoscaler Operator 2.12.1-394 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.12.1-394** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 버그 수정을 제공합니다. [RHSA-2024:2901](#)에 대해 다음 권고를

사용할 수 있습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 **Kubernetes** 기반 이벤트 기반 자동 스케일러(**KEDA**)의 커뮤니티 지원 버전을 제거합니다.

3.1.2.1.1. 버그 수정

- 이전에는 `protojson.Unmarshal` 함수가 특정 양식의 잘못된 **JSON** 형식을 해제 할 때 무한 루프에 입력되었습니다. 이 조건은 `Google.protobuf.Any` 값이 포함된 메시지 또는 `UnmarshalOptions.DiscardUnknown` 옵션이 설정된 경우 발생할 수 있습니다. 이 릴리스에서는 이 문제가 해결되었습니다. ([OCBUGS-30305](#))
- 이전 버전에서는 `multipart` 양식을 구문 분석할 때 `Request.ParseMultipartForm` 메서드를 사용하여 명시적으로 또는 `Request.FormValue`, `Request.PostFormValue` 또는 `Request.FormFile` 메서드를 사용하여 암시적으로 해석될 때 구문 분석된 양식의 총 크기에 대한 제한이 사용된 메모리에 적용되지 않았습니다. 이로 인해 메모리 소진이 발생할 수 있습니다. 이번 수정을 통해 이제 구문 분석 프로세스가 단일 양식 줄을 읽는 동안 양식 행의 최대 크기를 올바르게 제한합니다. ([OCBUGS-30360](#))
- 이전 버전에서는 **HTTP** 리디렉션을 따르는 경우 일치하는 하위 도메인에 없거나 초기 도메인의 정확한 일치에 따라 **HTTP** 클라이언트가 **Authorization** 또는 **Cryostat**와 같은 중요한 헤더를 전달하지 않았습니다. 예를 들어 `example.com` 에서 `www.example.com` 로의 리디렉션은 **Authorization** 헤더를 전달하지만 `www.example.org` 로 리디렉션은 헤더를 전달하지 않습니다. 이 릴리스에서는 이 문제가 해결되었습니다. ([OCBUGS-30365](#))
- 이전에는 알 수 없는 공개 키 알고리즘이 포함된 인증서 체인을 확인하면 인증서 확인 프로세스가 패닉 상태가 되었습니다. 이 조건은 `Config.ClientAuth` 매개변수를 `VerifyClientCertIfGiven` 또는 `RequireAndVerifyClientCert` 값으로 설정하는 모든 **TLS(Transport Layer Security)** 클라이언트 및 서버에 영향을 미쳤습니다. 기본 동작은 **TLS** 서버가 클라이언트 인증서를 확인하지 않는 것입니다. 이 릴리스에서는 이 문제가 해결되었습니다. ([OCBUGS-30370](#))
- 이전 버전에서는 `MarshalJSON` 메서드에서 반환된 오류가 사용자 제어 데이터가 포함된 경우 공격자는 데이터를 사용하여 **HTML** 템플릿 패키지의 컨텍스트 자동 이스케이프 동작을 중단할 수 있었습니다. 이 조건을 사용하면 후속 작업에서 예기치 않은 콘텐츠를 템플릿에 삽입할 수 있습니다. 이 릴리스에서는 이 문제가 해결되었습니다. ([OCBUGS-30397](#))
- 이전에는 `net/http` 및 `golang.org/x/net/http2` Go 패키지에서 **HTTP/2** 요청의 **CONTINUATION** 프레임 수를 제한하지 않았습니다. 이 조건으로 인해 **CPU** 사용량이 과도해질

수 있습니다. 이 릴리스에서는 이 문제가 해결되었습니다. ([OCPBUGS-30894](#))

3.1.2.2. 사용자 정의 Metrics Autoscaler Operator 2.12.1-384 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.12.1-384** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 버그 수정을 제공합니다. [RHBA-2024:2043](#)에 대해 다음 권고를 사용할 수 있습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 **KEDA** 버전을 제거합니다.

3.1.2.2.1. 버그 수정

- 이전에는 **custom-metrics-autoscaler** 및 **custom-metrics-autoscaler-adapter** 이미지에 시간대 정보가 누락되었습니다. 결과적으로 컨트롤러가 시간대 정보를 찾을 수 없기 때문에 **cron** 트리거를 사용하여 확장 개체가 작동하지 않았습니다. 이번 수정을 통해 시간대 정보를 포함하도록 이미지 빌드가 업데이트됩니다. 결과적으로 **cron** 트리거를 포함하는 확장 오브젝트가 올바르게 작동합니다. ([OCPBUGS-32395](#))

3.1.2.3. 사용자 정의 지표 Autoscaler Operator 2.12.1-376 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.12.1-376** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 보안 업데이트 및 버그 수정을 제공합니다. [RHSA-2024:1812](#)에 대해 다음 권고를 사용할 수 있습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 **KEDA** 버전을 제거합니다.

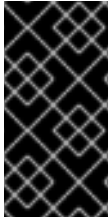
3.1.2.3.1. 버그 수정

- 이전 버전에서는 존재하지 않는 네임스페이스와 같은 잘못된 값이 확장 오브젝트 메타데이터에 지정된 경우 기본 확장기 클라이언트가 해제되거나 닫지 않아 클라이언트 설명자가 느려졌습니다. 이번 수정을 통해 오류가 있을 때 기본 클라이언트 설명자가 올바르게 닫히고 메모리가 유출되지 않습니다. ([OCPBUGS-30145](#))
- 이전 버전에서는 **CR**에서 **http**의 메트릭 포트 이름을 참조했기 때문에 **keda-metrics-**

apiserver Pod의 ServiceMonitor CR(사용자 정의 리소스)이 작동하지 않았습니다. 이번 수정에서는 메트릭의 적절한 포트 이름을 참조하도록 **ServiceMonitor CR**이 수정되었습니다. 결과적으로 서비스 모니터가 제대로 작동합니다. ([OCBUGS-25806](#))

3.1.2.4. 사용자 정의 지표 Autoscaler Operator 2.11.2-322 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.11.2-322** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 보안 업데이트 및 버그 수정을 제공합니다. [RHSA-2023:6144](#)에 대해 다음 권고를 사용할 수 있습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 **KEDA** 버전을 제거합니다.

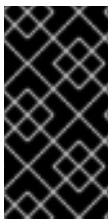
3.1.2.4.1. 버그 수정



Custom Metrics Autoscaler Operator 버전 3.11.2-311이 **Operator** 배포에 필요한 볼륨 마운트 없이 릴리스되었으므로 **Custom Metrics Autoscaler Operator Pod**가 15분마다 다시 시작됩니다. 이번 수정에서는 **Operator** 배포에 필요한 볼륨 마운트가 추가되었습니다. 결과적으로 **Operator**가 15분마다 더 이상 재시작되지 않습니다. ([OCBUGS-22361](#))

3.1.2.5. 사용자 정의 지표 Autoscaler Operator 2.11.2-311 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.11.2-311** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. **Custom Metrics Autoscaler Operator 2.11.2-311**의 구성 요소는 [RHBA-2023:5981](#)에서 릴리스되었습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 **KEDA** 버전을 제거합니다.

3.1.2.5.1. 새로운 기능 및 개선 사항

3.1.2.5.1.1. Red Hat OpenShift Service on AWS (ROSA) 및 OpenShift Dedicated 지원

Custom Metrics Autoscaler Operator 2.11.2-311은 **OpenShift ROSA** 및 **OpenShift Dedicated** 관리 클러스터에 설치할 수 있습니다. 이전 버전의 **Custom Metrics Autoscaler Operator**는 **openshift-keda** 네임스페이스에만 설치할 수 있었습니다. 이로 인해 **OpenShift ROSA** 및 **OpenShift Dedicated** 클러스터에 **Operator**가 설치되지 않았습니다. 이 **Custom Metrics Autoscaler** 버전을 사용하면

openshift-operators 또는 **keda** 와 같은 다른 네임스페이스에 설치할 수 있으므로 **ROSA** 및 **Dedicated** 클러스터에 설치할 수 있습니다.

3.1.2.5.2. 버그 수정

- 이전 버전에서는 **Custom Metrics Autoscaler Operator**를 설치 및 구성했지만 사용하지 않은 경우 **OpenShift CLI**에서 **oc** 명령을 입력한 후 **external.metrics.k8s.io/v1beta1**에 대한 **Got** 빈 응답: **external.metrics.k8s.io/v1beta1** 오류에 대한 리소스 목록을 가져올 수 없었습니다. 메시지가 무해하지만 혼동을 일으킬 수 있었습니다. 이번 수정으로 **external.metrics...** 오류에 대한 **Got** 빈 응답이 더 이상 부적절하게 표시되지 않습니다. ([OCBUGS-15779](#))
- 이전에는 **Custom Metrics Autoscaler**에서 관리하는 오브젝트에 대한 주석 또는 레이블 변경이 **Keda Controller**를 수정할 때마다 **Custom Metrics Autoscaler Operator**에 의해 취소되었습니다(예: 구성 변경 후). 이로 인해 오브젝트에서 라벨이 지속적으로 변경되었습니다. **Custom Metrics Autoscaler**는 이제 자체 주석을 사용하여 레이블 및 주석을 관리하고 주석 또는 레이블을 더 이상 잘못 되돌리지 않습니다. ([OCBUGS-15590](#))

3.1.2.6. 사용자 정의 지표 Autoscaler Operator 2.10.1-267 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.10.1-267** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. **Custom Metrics Autoscaler Operator 2.10.1-267**의 구성 요소는 [RHBA-2023:4089](#) 에서 릴리스되었습니다.



중요

이 버전의 **Custom Metrics Autoscaler Operator**를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 **KEDA** 버전을 제거합니다.

3.1.2.6.1. 버그 수정

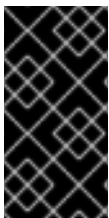
- 이전에는 **custom-metrics-autoscaler** 및 **custom-metrics-autoscaler-adapter** 이미지에 시간대 정보가 포함되지 않았습니다. 이로 인해 컨트롤러가 시간대 정보를 찾을 수 없기 때문에 **cron** 트리거를 사용하여 확장 개체가 작동하지 않았습니다. 이번 수정으로 이미지 빌드에 시간대 정보가 포함됩니다. 결과적으로 **cron** 트리거를 포함하는 확장 오브젝트가 올바르게 작동합니다. ([OCBUGS-15264](#))
- 이전에는 **Custom Metrics Autoscaler Operator**에서 다른 네임스페이스 및 클러스터 범위 오브젝트의 오브젝트를 포함하여 모든 관리 오브젝트의 소유권을 시도했습니다. 이로 인해 **Custom Metrics Autoscaler Operator**에서 **API** 서버가 되는 데 필요한 인증 정보를 읽기 위해 역할 바인딩을 생성할 수 없었습니다. 이로 인해 **kube-system** 네임 스페이스에 오류가 발생했습니다. 이번 수정을 통해 **Custom Metrics Autoscaler Operator**는 **ownerReference** 필드를 다른

네임스페이스 또는 클러스터 범위 오브젝트의 오브젝트에 추가하는 것을 건너뛸니다. 결과적으로 이제 오류 없이 역할 바인딩이 생성됩니다. ([OCBUGS-15038](#))

- 이전에는 Custom Metrics Autoscaler Operator에서 ownerReferences 필드를 openshift-keda 네임스페이스에 추가했습니다. 이로 인해 기능 문제가 발생하지 않았지만 이 필드가 있으면 클러스터 관리자에게 혼동이 발생할 수 있었습니다. 이번 수정으로 Custom Metrics Autoscaler Operator는 ownerReference 필드를 openshift-keda 네임스페이스에 추가하지 않습니다. 결과적으로 openshift-keda 네임스페이스에 더 이상 불필요한 ownerReference 필드가 없습니다. ([OCBUGS-15293](#))
- 이전 버전에서는 Pod ID 이외의 인증 방법으로 구성된 Prometheus 트리거를 사용한 후 podidentity 매개변수가 none 으로 설정된 경우 트리거를 스케일링하지 못했습니다. 이번 수정으로 OpenShift의 Custom Metrics Autoscaler에서 이제 none Pod ID 공급자 유형을 올바르게 처리합니다. 결과적으로 Prometheus 트리거는 Pod ID 이외의 인증 방법으로 구성되고 podidentity 매개변수 sset을 none 으로 올바르게 스케일링합니다. ([OCBUGS-15274](#))

3.1.2.7. 사용자 정의 Metrics Autoscaler Operator 2.10.1 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.10.1 릴리스는 OpenShift Container Platform 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. Custom Metrics Autoscaler Operator 2.10.1의 구성 요소는 [RHEA-2023:3199](#) 에서 릴리스되었습니다.



중요

이 버전의 Custom Metrics Autoscaler Operator를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 KEDA 버전을 제거합니다.

3.1.2.7.1. 새로운 기능 및 개선 사항

3.1.2.7.1.1. 사용자 정의 지표 Autoscaler Operator 일반 가용성

Custom Metrics Autoscaler Operator는 이제 일반적으로 Custom Metrics Autoscaler Operator 버전 2.10.1에서 사용할 수 있습니다.



중요

확장된 작업을 사용하여 스케일링하는 것은 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.

3.1.2.7.1.2. 성능 지표

이제 **PromQL(Prometheus Query Language)**을 사용하여 **Custom Metrics Autoscaler Operator**에서 지표를 쿼리할 수 있습니다.

3.1.2.7.1.3. 확장된 오브젝트에 대한 사용자 정의 지표 자동 스케일링 일시 중지

이제 필요에 따라 확장 오브젝트의 자동 스케일링을 일시 중지하고 준비가 되면 자동 스케일링을 재개할 수 있습니다.

3.1.2.7.1.4. 확장 오브젝트의 복제본 대체

이제 스케일링된 오브젝트가 소스에서 메트릭을 가져오지 못하는 경우 다시 대체할 복제본 수를 지정할 수 있습니다.

3.1.2.7.1.5. 확장된 오브젝트에 대해 사용자 정의 가능한 HPA 이름 지정

확장된 오브젝트에서 수평 **Pod** 자동 스케일러의 사용자 정의 이름을 지정할 수 있습니다.

3.1.2.7.1.6. 활성화 및 스케일링 임계값

HPA(수평 Pod 자동 스케일러)는 복제본 0개로 스케일링할 수 없으므로 **Custom Metrics Autoscaler Operator**는 해당 스케일링을 수행하여 **HPA**에서 스케일링을 수행합니다. 이제 **HPA**에서 복제본 수에 따라 자동 스케일링을 인수하는 시기를 지정할 수 있습니다. 이를 통해 확장 정책을 통해 유연성을 높일 수 있습니다.

3.1.2.8. 사용자 정의 메트릭 자동 스케일러 Operator 2.8.2-174 릴리스 노트

이번 **Custom Metrics Autoscaler Operator 2.8.2-174** 릴리스는 **OpenShift Container Platform** 클

러스티에서 **Operator**를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. **Custom Metrics Autoscaler Operator 2.8.2-174**의 구성 요소는 [RHEA-2023:1683](#)에서 릴리스되었습니다.



중요

Custom Metrics Autoscaler Operator 버전 2.8.2-174는 [기술 프리뷰](#) 기능입니다.

3.1.2.8.1. 새로운 기능 및 개선 사항

3.1.2.8.1.1. Operator 업그레이드 지원

이제 이전 버전의 **Custom Metrics Autoscaler Operator**에서 업그레이드할 수 있습니다. **Operator** 업그레이드에 대한 자세한 내용은 "ECDHE 리소스"의 "Operator 업데이트 채널 변경"을 참조하십시오.

3.1.2.8.1.2. must-gather 지원

OpenShift Container Platform must-gather 툴을 사용하여 **Custom Metrics Autoscaler Operator** 및 해당 구성 요소에 대한 데이터를 수집할 수 있습니다. 현재 사용자 정의 메트릭 자동 스케일러와 함께 **must-gather** 툴을 사용하는 프로세스는 다른 **Operator**와 다릅니다. 자세한 내용은 "ECDHE 리소스"의 디버깅 데이터를 참조하십시오.

3.1.2.9. 사용자 정의 Metrics Autoscaler Operator 2.8.2 릴리스 정보

이번 **Custom Metrics Autoscaler Operator 2.8.2** 릴리스는 **OpenShift Container Platform** 클러스터에서 **Operator**를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. **Custom Metrics Autoscaler Operator 2.8.2**의 구성 요소는 [RHSA-2023:1042](#)에서 릴리스되었습니다.



중요

Custom Metrics Autoscaler Operator 버전 2.8.2는 [기술 프리뷰](#) 기능입니다.

3.1.2.9.1. 새로운 기능 및 개선 사항

3.1.2.9.1.1. 감사 로깅

Custom Metrics Autoscaler Operator 및 관련 구성 요소에 대한 감사 로그를 수집하고 볼 수 있습니다. 감사 로그는 시스템의 개별 사용자, 관리자 또는 기타 구성 요소가 시스템에 영향을 준 활동 순서를 문서화하는 보안 관련 레코드 세트입니다.

3.1.2.9.1.2. Apache Kafka 메트릭을 기반으로 애플리케이션 확장

이제 **KEDA Apache kafka** 트리거/scaler를 사용하여 **Apache Kafka** 주제를 기반으로 배포를 확장할 수 있습니다.

3.1.2.9.1.3. CPU 메트릭을 기반으로 애플리케이션 스케일링

이제 **KEDA CPU** 트리거/scaler를 사용하여 **CPU** 메트릭을 기반으로 배포를 확장할 수 있습니다.

3.1.2.9.1.4. 메모리 메트릭을 기반으로 애플리케이션 스케일링

이제 **KEDA** 메모리 트리거/scaler를 사용하여 메모리 메트릭에 따라 배포를 확장할 수 있습니다.

3.2. 사용자 정의 METRICS AUTOSCALER OPERATOR 개요

개발자는 **Red Hat OpenShift**에 **Custom Metrics Autoscaler Operator**를 사용하여 **OpenShift Container Platform**에서 **CPU** 또는 메모리를 기반으로 하지 않는 사용자 정의 메트릭을 기반으로 배포, 상태 저장 세트, 사용자 정의 리소스 또는 작업의 **Pod** 수를 자동으로 늘리거나 줄이는 방법을 지정할 수 있습니다.

Custom Metrics Autoscaler Operator는 쿠버네티스 이벤트 기반 자동 스케일러(**KEDA**)를 기반으로 하는 선택적 **Operator**로, **Pod** 메트릭 이외의 추가 메트릭 소스를 사용하여 워크로드를 확장할 수 있습니다.

사용자 정의 메트릭 자동 스케일러는 현재 **Prometheus**, **CPU**, 메모리 및 **Apache Kafka** 메트릭만 지원됩니다.

Custom Metrics Autoscaler Operator는 특정 애플리케이션의 사용자 정의 외부 지표에 따라 **Pod**를 확장 및 축소합니다. 다른 애플리케이션에서는 다른 스케일링 방법을 계속 사용합니다. 사용자 지정 지표 자동 스케일러가 스케일링하는 방법을 결정하는 데 사용하는 이벤트 및 메트릭의 소스인 **scalers**라고도 하는 **트리거**를 구성합니다. 사용자 정의 지표 자동 스케일러는 메트릭 **API**를 사용하여 **OpenShift Container Platform**에서 사용할 수 있는 양식으로 외부 메트릭을 변환합니다. 사용자 정의 메트릭 자동 스케일러는 실제 스케일링을 수행하는 **HPA (Horizontal Pod Autoscaler)**를 생성합니다.

사용자 정의 지표 자동 스케일러를 사용하려면 스케일링 메타데이터를 정의하는 **CR(사용자 정의 리소스)**인 워크로드에 대해 **scaled Object** 또는 **scaled Job** 오브젝트를 만듭니다. 스케일링할 배포 또는 작업, 스케일링할 메트릭의 소스(trigger), 기타 매개변수(예: 최소 및 최대 복제본 수)를 지정합니다.



참고

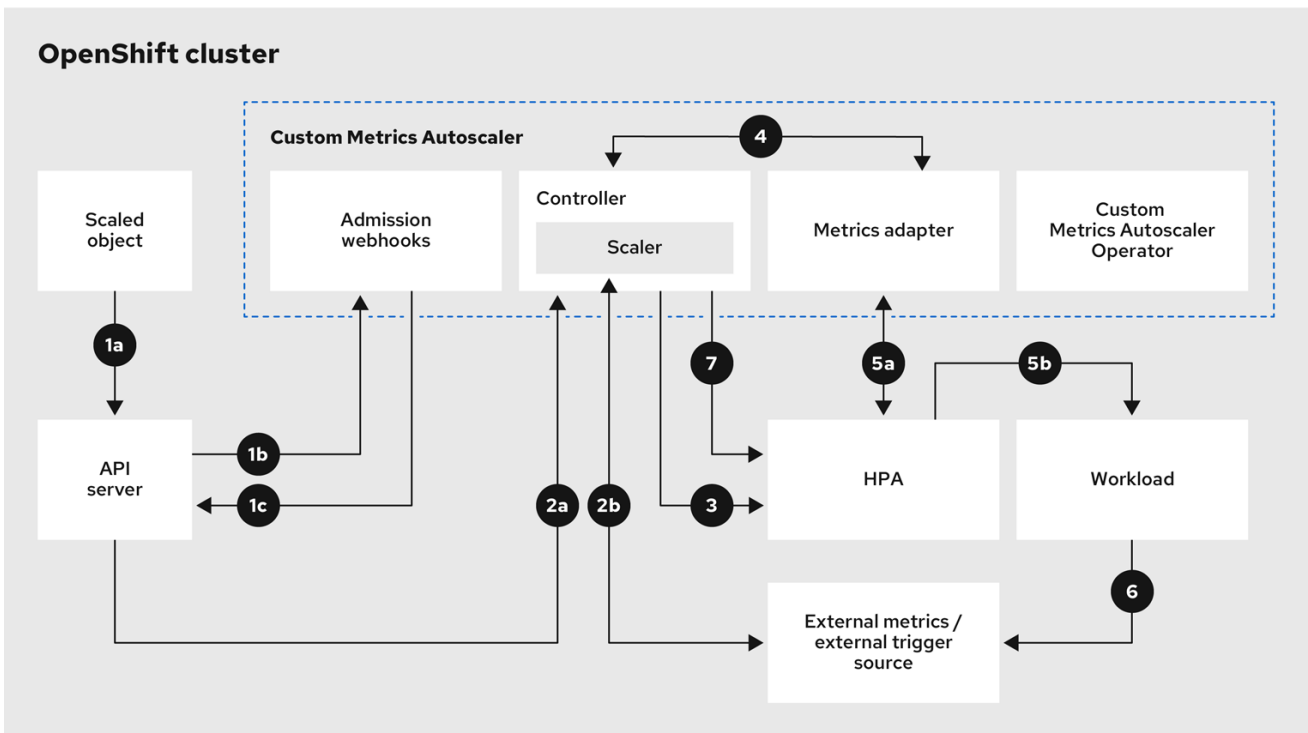
스케일링할 각 워크로드에 대해 하나의 확장 오브젝트 또는 스케일링된 작업만 생성할 수 있습니다. 또한 확장된 오브젝트 또는 스케일링된 작업과 동일한 워크로드에서 HPA(Horizontal Pod Autoscaler)를 사용할 수 없습니다.

HPA와 달리 사용자 정의 메트릭 자동 스케일러는 0으로 확장할 수 있습니다. 사용자 정의 메트릭 자동 스케일러 CR에서 minReplicaCount 값을 0으로 설정하면 사용자 정의 메트릭 자동 스케일러는 워크로드를 복제본이 0개에서 복제본 수로 또는 최대 0개의 복제본으로 축소합니다. 이를 활성화 단계라고 합니다. 최대 1개의 복제본으로 확장한 후 HPA는 스케일링을 제어합니다. 이를 스케일링 단계라고 합니다.

일부 트리거를 사용하면 클러스터 지표 자동 스케일러에서 스케일링하는 복제본 수를 변경할 수 있습니다. 모든 경우, 활성화 단계를 구성하는 매개 변수는 항상 활성화 접두사가 지정된 동일한 문구를 사용합니다. 예를 들어 threshold 매개변수가 스케일링을 구성하는 경우 activationThreshold가 활성화를 구성합니다. 활성화 및 확장 단계를 구성하면 확장 정책에 따라 유연성을 높일 수 있습니다. 예를 들어 메트릭이 특히 낮은 경우 확장 또는 축소되지 않도록 더 높은 활성화 단계를 구성할 수 있습니다.

활성화 값은 서로 다른 결정에 따라 스케일링 값보다 우선 순위가 높습니다. 예를 들어 임계값이 10으로 설정되고 활성화Threshold가 50개이면 메트릭에서 40개를 보고하면 scaler가 활성 상태가 아니며 HPA에 4개의 인스턴스가 필요한 경우에도 Pod가 0으로 확장됩니다.

그림 3.1. 사용자 정의 메트릭 자동 스케일러 워크플로



565_OpenShift_0224

클러스터의 워크로드에 대해 확장된 오브젝트 사용자 정의 리소스를 생성하거나 수정합니다. 오브젝트에는 해당 워크로드에 대한 스케일링 구성이 포함되어 있습니다. 새 오브젝트를 수락하기 전에 **OpenShift API** 서버에서 사용자 정의 지표 자동 스케일러 승인 **Webhook** 프로세스로 전송하여 오브젝트가 유효한지 확인합니다. 유효성 검사가 성공하면 **API** 서버는 오브젝트를 유지합니다.

2.

사용자 정의 메트릭 자동 스케일러 컨트롤러는 새 확장 오브젝트 또는 수정된 오브젝트를 감시합니다. **OpenShift API** 서버가 컨트롤러에게 변경 사항을 알릴 때 컨트롤러는 지표 데이터 변경을 위해 오브젝트에 지정된 데이터 소스라고도 하는 외부 트리거 소스를 모니터링합니다. 하나 이상의 스케일러는 외부 트리거 소스에서 스케일링 데이터를 요청합니다. 예를 들어 **Kafka** 트리거 유형의 경우 컨트롤러는 **Kafka** 스케일러를 사용하여 **Kafka** 인스턴스와 통신하여 트리거에서 요청한 데이터를 가져옵니다.

3.

컨트롤러는 확장된 오브젝트에 대한 수평 **Pod** 자동 스케일러 오브젝트를 생성합니다. 결과적으로 **Horizontal Pod Autoscaler (HPA) Operator**가 트리거와 연결된 스케일링 데이터 모니터링을 시작합니다. **HPA**는 클러스터 **OpenShift API** 서버 끝점에서 데이터 스케일링을 요청합니다.

4.

OpenShift API 서버 끝점은 사용자 정의 지표 자동 스케일러 지표 어댑터에서 제공합니다. 메트릭 어댑터가 사용자 지정 메트릭에 대한 요청을 수신하면 컨트롤러에 **GRPC** 연결을 사용하여 **scaler**에서 수신한 최신 트리거 데이터에 대해 요청합니다.

5.

HPA는 지표 어댑터에서 수신된 데이터를 기반으로 스케일링 결정을 내리고 복제본을 늘리거나 줄여 워크로드를 확장 또는 축소합니다.

6.

워크로드가 작동하면 스케일링 메트릭에 영향을 미칠 수 있습니다. 예를 들어 **Kafka** 큐에서 작업을 처리하도록 워크로드가 확장되면 워크로드 프로세스가 모든 작업 후에 큐 크기가 감소합니다. 결과적으로 워크로드가 축소됩니다.

7.

지표가 **minReplicaCount** 값으로 지정된 범위에 있는 경우 사용자 정의 지표 자동 스케일러 컨트롤러는 모든 스케일링을 비활성화하고 복제본 수를 고정된 수준으로 유지합니다. 메트릭이 해당 범위를 초과하면 사용자 정의 지표 자동 스케일러 컨트롤러에서 스케일링을 활성화하고 **HPA**에서 워크로드를 확장할 수 있습니다. 스케일링은 비활성화되어 있지만 **HPA**는 작업을 수행하지 않습니다.

3.2.1. 사용자 정의 지표 자동 스케일러의 사용자 정의 CA 인증서

기본적으로 **Custom Metrics Autoscaler Operator**는 자동으로 생성된 서비스 **CA** 인증서를 사용하여 클러스터 내 서비스에 연결합니다.

사용자 정의 **CA** 인증서가 필요한 클러스터 외부 서비스를 사용하려면 구성 맵에 필요한 인증서를 추가할 수 있습니다. 그런 다음 사용자 정의 지표 자동 스케일러 설치에 설명된 대로 구성 맵을 **KedaController** 사용자 정의 리소스에 추가합니다. **Operator**는 시작 시 해당 인증서를 로드하고 **Operator**에서 신뢰할 수 있는 인증서로 등록합니다.

구성 맵에는 하나 이상의 **PEM** 인코딩 **CA** 인증서가 포함된 인증서 파일이 하나 이상 포함될 수 있습니다. 또는 각 인증서 파일에 대해 별도의 구성 맵을 사용할 수 있습니다.



참고

나중에 구성 맵을 업데이트하여 인증서를 추가하는 경우 변경 사항을 적용하려면 **keda-operator-* Pod**를 다시 시작해야 합니다.

3.3. 사용자 정의 메트릭 자동 스케일러 설치

OpenShift Container Platform 웹 콘솔을 사용하여 **Custom Metrics Autoscaler Operator**를 설치할 수 있습니다.

설치 시 다음 5개의 **CRD**가 생성됩니다.

- **ClusterTriggerAuthentication**
- **KedaController**
- **ScaledJob**
- **ScaledObject**
- **TriggerAuthentication**

3.3.1. 사용자 정의 메트릭 자동 스케일러 설치

다음 절차를 사용하여 **Custom Metrics Autoscaler Operator**를 설치할 수 있습니다.

사전 요구 사항

- **Cluster Metrics Autoscaler Operator**의 이전에 설치한 기술 프리뷰 버전을 제거합니다.
- 커뮤니티 기반 **KEDA** 버전을 제거합니다.

또한 다음 명령을 실행하여 **KEDA 1.x** 사용자 지정 리소스 정의를 제거합니다.

```
$ oc delete crd scaledobjects.keda.k8s.io
```

```
$ oc delete crd triggerauthentications.keda.k8s.io
```

- **선택 사항:** 외부 **Kafka** 클러스터 또는 외부 **Prometheus** 서비스와 같은 클러스터 외부 서비스에 연결하는 데 **Custom Metrics Autoscaler Operator**가 필요한 경우 필요한 서비스 **CA** 인증서를 구성 맵에 배치합니다. 구성 맵은 **Operator**가 설치된 동일한 네임스페이스에 있어야 합니다. 예를 들면 다음과 같습니다.

```
$ oc create configmap -n openshift-keda thanos-cert --from-file=ca-cert.pem
```

절차

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.
2. 사용 가능한 **Operator** 목록에서 **Custom Metrics Autoscaler**를 선택한 다음 설치를 클릭합니다.
3. **Operator** 설치 페이지에서 설치 모드에 대해 **All namespaces on the cluster (default)** 옵션이 선택되어 있는지 확인합니다. 그러면 모든 네임스페이스에 **Operator**가 설치됩니다.
4. 설치된 네임스페이스에 대해 **openshift-keda** 네임스페이스가 선택되어 있는지 확인합니다. **OpenShift Container Platform**은 클러스터에 없는 경우 네임스페이스를 생성합니다.
5. 설치를 클릭합니다.
6. **Custom Metrics Autoscaler Operator** 구성 요소를 나열하여 설치를 확인합니다.

- a. 워크로드 → **Pod**로 이동합니다.
 - b. 드롭다운 메뉴에서 **openshift-keda** 프로젝트를 선택하고 **custom-metrics-autoscaler-operator-* Pod**가 실행 중인지 확인합니다.
 - c. **Workloads** → **Deployments** 로 이동하여 **custom-metrics-autoscaler-operator** 배포가 실행 중인지 확인합니다.
7. 선택 사항: 다음 명령을 사용하여 **OpenShift CLI**에서 설치를 확인합니다.

```
$ oc get all -n openshift-keda
```

출력은 다음과 유사합니다.

출력 예

```

NAME                                READY STATUS RESTARTS AGE
pod/custom-metrics-autoscaler-operator-5fd8d9ffd8-xt4xp 1/1 Running 0
18m

NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/custom-metrics-autoscaler-operator 1/1 1 1 18m

NAME                                DESIRED CURRENT READY AGE
replicaset.apps/custom-metrics-autoscaler-operator-5fd8d9ffd8 1 1 1
18m

```

8. 필요한 **CRD**를 생성하는 **KedaController** 사용자 정의 리소스를 설치합니다.
- a. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → 설치된 **Operator**를 클릭합니다.
 - b. **Custom Metrics Autoscaler** 를 클릭합니다.

c.

Operator 세부 정보 페이지에서 **KedaController** 탭을 클릭합니다.

d.

KedaController 탭에서 **KedaController** 생성을 클릭하고 파일을 편집합니다.

```

kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  watchNamespace: " 1
  operator:
    logLevel: info 2
    logEncoder: console 3
    caConfigMaps: 4
    - thanos-cert
    - kafka-cert
  metricsServer:
    logLevel: '0' 5
    auditConfig: 6
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
    policy:
      rules:
        - level: Metadata
      omitStages: ["RequestReceived"]
      omitManagedFields: false
    lifetime:
      maxAge: "2"
      maxBackup: "1"
      maxSize: "50"
  serviceAccount: {}

```

1

Custom Metrics Autoscaler Operator에서 애플리케이션을 스케일링해야 하는 단일 네임스페이스를 지정합니다. 비워 두거나 모든 네임스페이스의 애플리케이션을 확장하려면 비워 둡니다. 이 필드에는 네임스페이스가 있거나 비어 있어야 합니다. 기본값은 비어 있습니다.

2

Custom Metrics Autoscaler Operator 로그 메시지의 상세 수준을 지정합니다. 허용되는 값은 `debug,info,error` 입니다. 기본값은 `info` 입니다.

3

Custom Metrics Autoscaler Operator 로그 메시지의 로깅 형식을 지정합니다. 허용되는 값은 `console` 또는 `json` 입니다. 기본값은 `console` 입니다.

4

선택 사항: **Custom Metrics Autoscaler Operator**에서 **TLS** 사용 메트릭 소스에 안전하게 연결하는 데 사용할 수 있는 **CA** 인증서로 하나 이상의 구성 맵을 지정합니다.

5

Custom Metrics Autoscaler Metrics Server의 로깅 수준을 지정합니다. 허용되는 값은 **info** 및 **4** 또는 **debug** 에 대해 **0** 입니다. 기본값은 **0**입니다.

6

Custom Metrics Autoscaler Operator에 대한 감사 로깅을 활성화하고 "감사 로깅 구성" 섹션에 설명된 대로 사용할 감사 정책을 지정합니다.

e.

생성 을 클릭하여 **KEDA** 컨트롤러를 생성합니다.

3.4. 사용자 정의 메트릭 자동 스케일러 트리거 이해

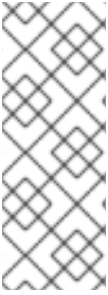
scalers라고도 하는 트리거는 **Custom Metrics Autoscaler Operator**가 **Pod**를 스케일링하는 데 사용하는 메트릭을 제공합니다.

사용자 정의 메트릭 자동 스케일러는 현재 **Prometheus**, **CPU**, 메모리 및 **Apache Kafka** 트리거만 지원됩니다.

ScaledObject 또는 **ScaledJob** 사용자 정의 리소스를 사용하여 다음 섹션에 설명된 대로 특정 개체에 대한 트리거를 구성합니다.

3.4.1. Prometheus 트리거 이해

설치된 **OpenShift Container Platform** 모니터링을 사용하거나 외부 **Prometheus** 서버를 메트릭 소스로 사용할 수 있는 **Prometheus** 메트릭을 기반으로 **Pod**를 확장할 수 있습니다. 메트릭의 소스로 **OpenShift Container Platform** 모니터링을 사용하는 데 필요한 구성에 대한 자세한 내용은 "추가 리소스"를 참조하십시오.



참고

Prometheus가 사용자 정의 지표 자동 스케일러가 스케일링하는 애플리케이션에서 지표를 수집하는 경우 사용자 정의 리소스에서 최소 복제본을 0으로 설정하지 마십시오. 애플리케이션 Pod가 없는 경우 사용자 정의 지표 자동 스케일러에는 스케일링할 메트릭이 없습니다.

Prometheus 대상이 있는 스케일링 오브젝트의 예

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prom-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
    - type: prometheus 1
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092 2
        namespace: kedatest 3
        metricName: http_requests_total 4
        threshold: '5' 5
        query: sum(rate(http_requests_total{job="test-app"}[1m])) 6
        authModes: basic 7
        cortexOrgID: my-org 8
        ignoreNullValues: false 9
        unsafeSsl: false 10

```

1

Prometheus를 트리거 유형으로 지정합니다.

2

Prometheus 서버의 주소를 지정합니다. 이 예에서는 **OpenShift Container Platform** 모니터링을 사용합니다.

3

선택 사항: 스케일링할 오브젝트의 네임스페이스를 지정합니다. **OpenShift Container Platform** 모니터링을 메트릭 소스로 사용하는 경우 이 매개변수가 필요합니다.

4

external.metrics.k8s.io API에서 메트릭을 식별하는 이름을 지정합니다. 둘 이상의 트리거를 사용하는 경우 모든 메트릭 이름을 고유해야 합니다.

5

스케일링을 트리거하는 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

6

사용할 **Prometheus** 쿼리를 지정합니다.

7

사용할 인증 방법을 지정합니다. **Prometheus** 스케일러는 전달자 인증(베어러 인증), 기본 인증(기본인증) 또는 **TLS** 인증(**tls**)을 지원합니다. 다음 섹션에서 설명하는 대로 트리거 인증에 특정 인증 매개변수를 구성합니다. 필요한 경우 시크릿을 사용할 수도 있습니다.

8

선택 사항: **Prometheus**의 **X-Scope-OrgID** 헤더를 다중 테넌트 **Cortex** 또는 **Mimir** 스토리지로 전달합니다. 이 매개변수는 반환해야 하는 데이터를 나타내기 위해 다중 테넌트 **Prometheus** 스토리지에서만 필요합니다.

9

선택 사항: **Prometheus** 대상이 손실되는 경우 트리거를 진행하는 방법을 지정합니다.

- **true** 인 경우 **Prometheus** 대상이 손실되면 트리거가 계속 작동합니다. 이는 기본 동작입니다.
- **false** 인 경우 **Prometheus** 대상이 손실되면 트리거에서 오류를 반환합니다.

10

선택 사항: 인증서 검사를 건너뛰어야 하는지 여부를 지정합니다. 예를 들어 **Prometheus** 끝점에서 자체 서명된 인증서를 사용하는 경우 검사를 건너뛸 수 있습니다.

- **true** 인 경우 인증서 검사가 수행됩니다.
- **false** 인 경우 인증서 검사가 수행되지 않습니다. 이는 기본 동작입니다.

3.4.1.1. OpenShift Container Platform 모니터링을 사용하도록 사용자 정의 메트릭 자동 스케일러 구성

설치된 **OpenShift Container Platform Prometheus** 모니터링을 사용자 정의 메트릭 자동 스케일러에서 사용하는 메트릭의 소스로 사용할 수 있습니다. 그러나 수행해야 하는 몇 가지 추가 구성이 있습니다.



참고

이러한 단계는 외부 **Prometheus** 소스에 필요하지 않습니다.

이 섹션에 설명된 대로 다음 작업을 수행해야 합니다.

- 서비스 계정을 생성하여 토큰을 가져옵니다.
- 역할을 생성합니다.
- 해당 역할을 서비스 계정에 추가합니다.
- **Prometheus**에서 사용하는 트리거 인증 오브젝트에서 토큰을 참조합니다.

사전 요구 사항

- **OpenShift Container Platform** 모니터링이 설치되어 있어야 합니다.
- 사용자 정의 워크로드 모니터링 구성 맵 생성 섹션에 설명된 대로 **OpenShift Container Platform** 모니터링에서 사용자 정의 워크로드 모니터링 이 활성화되어야 합니다.
- **Custom Metrics Autoscaler Operator**가 설치되어 있어야 합니다.

절차

1. 스케일링할 오브젝트를 사용하여 프로젝트로 변경합니다.

\$ oc project my-project

2.

클러스터에 없는 경우 다음 명령을 사용하여 서비스 계정을 생성합니다.

\$ oc create serviceaccount <service_account>

다음과 같습니다.

<service_account>

서비스 계정의 이름을 지정합니다.

3.

다음 명령을 사용하여 서비스 계정에 할당된 토큰을 찾습니다.

\$ oc describe serviceaccount <service_account>

다음과 같습니다.

<service_account>

서비스 계정의 이름을 지정합니다.

출력 예

```

Name:          thanos
Namespace:     my-project
Labels:        <none>
Annotations:   <none>
Image pull secrets: thanos-dockercfg-nnwgj
Mountable secrets: thanos-dockercfg-nnwgj
Tokens:        thanos-token-9g4n5 1
Events:        <none>

```

1

트리거 인증에 이 토큰을 사용합니다.

4.

서비스 계정 토큰을 사용하여 트리거 인증을 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-trigger-auth-prometheus
spec:
  secretTargetRef: 1
  - parameter: bearerToken 2
    name: thanos-token-9g4n5 3
    key: token 4
  - parameter: ca
    name: thanos-token-9g4n5
    key: ca.crt
```

1

이 오브젝트가 권한 부여에 보안을 사용하도록 지정합니다.

2

토큰을 사용하여 제공할 인증 매개 변수를 지정합니다.

3

사용할 토큰의 이름을 지정합니다.

4

지정된 매개변수와 함께 사용할 토큰의 키를 지정합니다.

b.

CR 오브젝트를 생성합니다.

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

5.

Thanos 메트릭을 읽는 역할을 생성합니다.

a.

다음 매개변수를 사용하여 **YAML** 파일을 생성합니다.

■

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch

```

- b. **CR** 오브젝트를 생성합니다.

```

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

```

- 6. **Thanos** 메트릭을 읽기 위한 역할 바인딩을 생성합니다.

- a. 다음과 유사한 **YAML** 파일을 생성합니다.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: thanos-metrics-reader 1
  namespace: my-project 2
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: thanos-metrics-reader
subjects:
- kind: ServiceAccount
  name: thanos 3
  namespace: my-project 4

```

1

생성한 역할의 이름을 지정합니다.

2

3

역할에 바인딩할 서비스 계정의 이름을 지정합니다.

4

스케일링할 오브젝트의 네임스페이스를 지정합니다.

b.

CR 오브젝트를 생성합니다.

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

"사용자 정의 메트릭 자동 스케일러를 추가하는 방법 이해"에 설명된 대로 확장 오브젝트 또는 확장 작업을 배포하여 애플리케이션에 대한 자동 스케일링을 활성화할 수 있습니다. **OpenShift Container Platform** 모니터링을 소스, 트리거 또는 **scaler**로 사용하려면 다음 매개변수를 포함해야 합니다.

- **triggers.type** 은 **prometheus**여야 합니다
- **triggers.metadata.serverAddress** 는 **https://thanos-querier.openshift-monitoring.svc.cluster.local:9092**이어야 합니다
- **triggers.metadata.authModes** 는 **베어러**여야 합니다.
- **Trigger.metadata.namespace** 는 스케일링할 오브젝트의 네임스페이스로 설정해야 합니다.
- **triggers.authenticationRef** 는 이전 단계에서 지정된 트리거 인증 리소스를 가리켜야 합니다.

3.4.2. CPU 트리거 이해

CPU 메트릭을 기반으로 Pod를 확장할 수 있습니다. 이 트리거는 클러스터 지표를 메트릭의 소스로 사용합니다.

사용자 정의 메트릭 자동 스케일러는 지정한 CPU 사용량을 유지하도록 오브젝트와 연결된 Pod를 스케일링합니다. 자동 스케일러는 최소 및 최대 개수 사이에서 복제본 수를 늘리거나 줄여 모든 Pod에서 지정된 CPU 사용률을 유지합니다. 메모리 트리거는 전체 Pod의 메모리 사용률을 고려합니다. Pod에 컨테이너가 여러 개 있는 경우 메모리 트리거는 Pod에 있는 모든 컨테이너의 총 메모리 사용률을 고려합니다.



참고

- 이 트리거는 **ScaledJob** 사용자 정의 리소스와 함께 사용할 수 없습니다.
- 메모리 트리거를 사용하여 오브젝트를 스케일링할 때 여러 트리거를 사용하는 경우에도 개체는 0으로 스케일링되지 않습니다.

CPU 대상이 있는 확장된 오브젝트의 예

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: cpu 1
    metricType: Utilization 2
    metadata:
      value: '60' 3
    minReplicaCount: 1 4

```

1

CPU를 트리거 유형으로 지정합니다.

2

사용할 메트릭의 유형(Utilization 또는 AverageValue)을 지정합니다.

3

스케일링을 트리거하는 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

- **Utilization** 을 사용하는 경우 **target** 값은 **Pod**에 대해 요청된 리소스의 백분율로 표시되는 모든 관련 **Pod**에서 리소스 지표의 평균값입니다.
- **AverageValue** 를 사용하는 경우 **target** 값은 모든 관련 **Pod**의 평균 지표입니다.

4

축소 시 최소 복제본 수를 지정합니다. **CPU** 트리거의 경우 **CPU** 지표만 사용하는 경우 **HPA**를 0으로 확장할 수 없기 때문에 **CPU** 트리거의 값을 1 이상 입력합니다.

3.4.3. 메모리 트리거 이해

메모리 메트릭을 기반으로 **Pod**를 확장할 수 있습니다. 이 트리거는 클러스터 지표를 메트릭의 소스로 사용합니다.

사용자 정의 메트릭 자동 스케일러는 오브젝트와 연결된 **Pod**를 스케일링하여 지정된 평균 메모리 사용량을 유지합니다. 자동 스케일러는 최소 및 최대 개수 사이에서 복제본 수를 늘리거나 줄여 모든 **Pod**에서 지정된 메모리 사용률을 유지합니다. 메모리 트리거는 전체 **Pod**의 메모리 사용률을 고려합니다. **Pod**에 컨테이너가 여러 개 있는 경우 메모리 사용률은 모든 컨테이너의 합계입니다.

참고

- 이 트리거는 **ScaledJob** 사용자 정의 리소스와 함께 사용할 수 없습니다.
- 메모리 트리거를 사용하여 오브젝트를 스케일링할 때 여러 트리거를 사용하는 경우에도 개체는 0으로 스케일링되지 않습니다.

메모리 대상이 있는 확장된 오브젝트의 예

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: memory-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
    - type: memory 1
```

```
metricType: Utilization 2
metadata:
  value: '60' 3
  containerName: api 4
```

1

memory를 트리거 유형으로 지정합니다.

2

사용할 메트릭의 유형(**Utilization** 또는 **AverageValue**)을 지정합니다.

3

스케일링을 트리거하는 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

•

Utilization 을 사용하는 경우 **target** 값은 **Pod**에 대해 요청된 리소스의 백분율로 표시되는 모든 관련 **Pod**에서 리소스 지표의 평균값입니다.

•

AverageValue 를 사용하는 경우 **target** 값은 모든 관련 **Pod**의 평균 지표입니다.

4

선택 사항: 전체 **Pod**가 아닌 해당 컨테이너의 메모리 사용률에 따라 스케일링할 개별 컨테이너를 지정합니다. 이 예제에서는 **api** 라는 컨테이너만 스케일링할 수 있습니다.

3.4.4. Kafka 트리거 이해

Apache Kafka 주제 또는 **Kafka** 프로토콜을 지원하는 기타 서비스를 기반으로 **Pod**를 확장할 수 있습니다. 사용자 정의 메트릭 자동 스케일러는 스케일링된 오브젝트 또는 스케일링 작업에서 **allowIdleConsumers** 매개변수를 **true** 로 설정하지 않는 한 **Kafka** 파티션 수보다 크게 확장되지 않습니다.

참고

소비자 그룹의 수가 주제의 파티션 수를 초과하면 추가 소비자 그룹은 유휴 상태로 유지됩니다. 이 문제를 방지하려면 기본적으로 복제본 수는 다음을 초과하지 않습니다.

- 주제가 지정된 경우 주제의 파티션 수
- 주제가 지정되지 않은 경우 소비자 그룹에 있는 모든 항목의 파티션 수
- 확장 오브젝트 또는 확장 작업 CR에 지정된 `maxReplicaCount`

`allowIdleConsumers` 매개변수를 사용하여 이러한 기본 동작을 비활성화할 수 있습니다.

Kafka 대상이 있는 확장된 오브젝트의 예

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: kafka-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: kafka ①
    metadata:
      topic: my-topic ②
      bootstrapServers: my-cluster-kafka-bootstrap.openshift-operators.svc:9092 ③
      consumerGroup: my-group ④
      lagThreshold: '10' ⑤
      activationLagThreshold: '5' ⑥
      offsetResetPolicy: latest ⑦
      allowIdleConsumers: true ⑧
      scaleToZeroOnInvalidOffset: false ⑨
      excludePersistentLag: false ⑩
      version: '1.0.0' ⑪
      partitionLimitation: '1,2,10-20,31' ⑫

```

1

Kafka를 트리거 유형으로 지정합니다.

2

Kafka가 오프셋 지연을 처리하는 **Kafka** 항목의 이름을 지정합니다.

3

연결할 **Kafka** 브로커의 쉽표로 구분된 목록을 지정합니다.

4

주제에서 오프셋을 확인하고 관련 지연을 처리하는 데 사용되는 **Kafka** 소비자 그룹의 이름을 지정합니다.

5

선택 사항: 스케일링을 트리거하는 평균 대상 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다. 기본값은 5 입니다.

6

선택 사항: 활성화 단계의 대상 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

7

선택 사항: **Kafka** 소비자에 대한 **Kafka** 오프셋 재설정 정책을 지정합니다. 사용 가능한 값은 **latest** 및 **earliest** 입니다. 기본값은 **latest** 입니다.

8

선택 사항: **Kafka** 복제본 수가 주제의 파티션 수를 초과할 수 있는지 여부를 지정합니다.

- **true** 인 경우 **Kafka** 복제본 수는 주제의 파티션 수를 초과할 수 있습니다. 이를 통해 유틸리티 상태의 **Kafka** 소비자를 사용할 수 있습니다.
- **false** 인 경우 **Kafka** 복제본 수는 주제의 파티션 수를 초과할 수 없습니다. 이는 기본값입니다.

9

Kafka 파티션에 유효한 오프셋이 없을 때 트리거가 작동하는 방법을 지정합니다.

- **true** 인 경우 소비자는 해당 파티션에 대해 0으로 확장됩니다.
- **false** 인 경우 **scaler**는 해당 파티션에 대한 단일 소비자를 유지합니다. 이는 기본값입니다.

10

선택 사항: 현재 오프셋이 이전 폴링 사이클의 현재 오프셋과 동일한 파티션에 대해 트리거에 파티션 지연을 포함하거나 제외하는지 여부를 지정합니다.

- **true** 인 경우 **scaler**는 이러한 파티션에서 파티션 지연을 제외합니다.
- **false** 인 경우 트리거는 모든 파티션에 모든 소비자 지연을 포함합니다. 이는 기본값입니다.

11

선택 사항: **Kafka** 브로커의 버전을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다. 기본값은 **1.0.0** 입니다.

12

선택 사항: 스케일링 범위를 지정할 쉼표로 구분된 파티션 **ID** 목록을 지정합니다. 설정된 경우 지연을 계산할 때 나열된 **ID**만 고려합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다. 기본값은 모든 파티션을 고려하는 것입니다.

3.5. 사용자 정의 메트릭 자동 스케일러가 인증 트리거 이해

트리거 인증을 사용하면 확장된 오브젝트 또는 관련 컨테이너에서 사용할 수 있는 스케일링된 작업에 인증 정보를 포함할 수 있습니다. 트리거 인증을 사용하여 **OpenShift Container Platform** 보안, 플랫폼 네이티브 **Pod** 인증 메커니즘, 환경 변수 등을 전달할 수 있습니다.

스케일링하려는 오브젝트와 동일한 네임스페이스에 **TriggerAuthentication** 오브젝트를 정의합니다. 이 트리거 인증은 해당 네임스페이스의 오브젝트에서만 사용할 수 있습니다.

또는 여러 네임스페이스의 오브젝트 간에 인증 정보를 공유하려면 모든 네임스페이스에서 사용할 수

있는 **ClusterTriggerAuthentication** 오브젝트를 생성할 수 있습니다.

인증 트리거 및 클러스터 트리거 인증은 동일한 구성을 사용합니다. 그러나 클러스터 트리거 인증에는 스케일링된 오브젝트의 인증 참조에 추가 **kind** 매개변수가 필요합니다.

보안이 포함된 인증 트리거 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: user-name 3
    name: my-secret 4
    key: USER_NAME 5
  - parameter: password
    name: my-secret
    key: USER_PASSWORD
```

1

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

이 트리거 인증이 권한 부여에 보안을 사용하도록 지정합니다.

3

보안을 사용하여 제공할 인증 매개 변수를 지정합니다.

4

사용할 시크릿의 이름을 지정합니다.

5

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

보안이 포함된 클러스터 트리거 인증 예

```

kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata: ❶
  name: secret-cluster-triggerauthentication
spec:
  secretTargetRef: ❷
  - parameter: user-name ❸
    name: secret-name ❹
    key: USER_NAME ❺
  - parameter: user-password
    name: secret-name
    key: USER_PASSWORD

```

❶

클러스터 트리거 인증에 네임스페이스가 사용되지 않습니다.

❷

이 트리거 인증이 권한 부여에 보안을 사용하도록 지정합니다.

❸

보안을 사용하여 제공할 인증 매개 변수를 지정합니다.

❹

사용할 시크릿의 이름을 지정합니다.

❺

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

토큰이 있는 트리거 인증의 예

```

kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:

```

```
name: token-triggerauthentication  
namespace: my-namespace 1  
spec:  
secretTargetRef: 2  
- parameter: bearerToken 3  
name: my-token-2vzfq 4  
key: token 5  
- parameter: ca  
name: my-token-2vzfq  
key: ca.crt
```

1

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

이 트리거 인증이 권한 부여에 보안을 사용하도록 지정합니다.

3

토큰을 사용하여 제공할 인증 매개 변수를 지정합니다.

4

사용할 토큰의 이름을 지정합니다.

5

지정된 매개변수와 함께 사용할 토큰의 키를 지정합니다.

환경 변수를 사용한 인증 트리거 예

```
kind: TriggerAuthentication  
apiVersion: keda.sh/v1alpha1  
metadata:  
name: env-var-triggerauthentication  
namespace: my-namespace 1  
spec:  
env: 2
```

```
- parameter: access_key 3
  name: ACCESS_KEY 4
  containerName: my-container 5
```

1

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

이 트리거 인증이 권한 부여에 환경 변수를 사용하도록 지정합니다.

3

이 변수로 설정할 매개변수를 지정합니다.

4

환경 변수의 이름을 지정합니다.

5

선택 사항: 인증이 필요한 컨테이너를 지정합니다. 컨테이너는 스케일링 오브젝트에서 `scaleTargetRef` 에서 참조하는 리소스와 동일한 리소스에 있어야 합니다.

Pod 인증 공급자를 사용한 트리거 인증 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: pod-id-triggerauthentication
  namespace: my-namespace 1
spec:
  podIdentity: 2
  provider: aws-eks 3
```

1

2

이 트리거 인증이 승인을 위해 플랫폼 네이티브 Pod 인증 방법을 사용하도록 지정합니다.

3

Pod ID를 지정합니다. 지원되는 값은 none,azure,aws-eks, aws-kiam. 기본값은 None 입니다.

추가 리소스

- **OpenShift Container Platform** 보안에 대한 자세한 내용은 [Pod에 민감한 데이터 제공을 참조하십시오.](#)

3.5.1. 트리거 인증 사용

사용자 정의 리소스를 사용하여 인증을 생성한 후 확장된 오브젝트 또는 확장된 작업에 대한 참조를 추가하여 인증 트리거 및 클러스터 트리거 인증을 사용합니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator**가 설치되어 있어야 합니다.
- 보안을 사용하는 경우 **Secret** 오브젝트가 있어야 합니다. 예를 들면 다음과 같습니다.

보안 예

```

apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  user-name: <base64_USER_NAME>
  password: <base64_USER_PASSWORD>

```

절차

1.

TriggerAuthentication 또는 **ClusterTriggerAuthentication** 오브젝트를 생성합니다.

a.

오브젝트를 정의하는 **YAML** 파일을 생성합니다.

보안이 포함된 인증 트리거 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: prom-triggerauthentication
  namespace: my-namespace
spec:
  secretTargetRef:
    - parameter: user-name
      name: my-secret
      key: USER_NAME
    - parameter: password
      name: my-secret
      key: USER_PASSWORD
```

b.

TriggerAuthentication 오브젝트를 생성합니다.

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

2.

트리거 인증을 사용하는 **scaledObject** **YAML** 파일을 생성하거나 편집합니다.

a.

다음 명령을 실행하여 오브젝트를 정의하는 **YAML** 파일을 생성합니다.

트리거 인증이 있는 확장된 오브젝트의 예

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
```

```

scaleTargetRef:
  name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
  metadata:
    serverAddress: https://thanos-querier.openshift-
monitoring.svc.cluster.local:9092
    namespace: kedatest # replace <NAMESPACE>
    metricName: http_requests_total
    threshold: '5'
    query: sum(rate(http_requests_total{job="test-app"}[1m]))
    authModes: "basic"
  authenticationRef:
    name: prom-triggerauthentication 1
    kind: TriggerAuthentication 2

```

1

트리거 인증 오브젝트의 이름을 지정합니다.

2

`TriggerAuthentication` 을 지정합니다. `TriggerAuthentication` 이 기본값입니다.

클러스터 트리거 인증이 있는 확장 오브젝트의 예

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
  metadata:
    serverAddress: https://thanos-querier.openshift-
monitoring.svc.cluster.local:9092

```

```

namespace: kedatest # replace <NAMESPACE>
metricName: http_requests_total
threshold: '5'
query: sum(rate(http_requests_total{job="test-app"}[1m]))
authModes: "basic"
authenticationRef:
  name: prom-cluster-triggerauthentication 1
  kind: ClusterTriggerAuthentication 2

```

1

트리거 인증 오브젝트의 이름을 지정합니다.

2

`ClusterTriggerAuthentication` 을 지정합니다.

b.

다음 명령을 실행하여 확장 오브젝트를 생성합니다.

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

3.6. 확장 오브젝트에 대한 사용자 정의 지표 자동 스케일러 일시 중지

필요에 따라 워크로드 자동 스케일링을 일시 중지하고 다시 시작할 수 있습니다.

예를 들어 클러스터 유지 관리를 수행하기 전에 자동 스케일링을 중지하거나 누락되지 않은 워크로드를 제거하여 리소스 부족을 방지할 수 있습니다.

3.6.1. 사용자 정의 메트릭 자동 스케일러 일시 중지

확장된 오브젝트의 사용자 정의 지표 자동 스케일러에 `autoscaling.keda.sh/paused-replicas` 주석을 추가하여 확장 오브젝트의 자동 스케일링을 일시 중지할 수 있습니다. 사용자 정의 메트릭 자동 스케일러는 해당 워크로드의 복제본을 지정된 값으로 스케일링하고 주석이 제거될 때까지 자동 스케일링을 일시 중지합니다.

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:

```

```

annotations:
  autoscaling.keda.sh/paused-replicas: "4"
# ...

```

절차

1. 다음 명령을 사용하여 워크로드에 대한 **scaledObject CR**을 편집합니다.

```
$ oc edit ScaledObject scaledobject
```

2. 다음 값을 사용하여 **autoscaling.keda.sh/paused-replicas** 주석을 추가합니다.

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
    creationTimestamp: "2023-02-08T14:41:01Z"
    generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: '65729'
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0

```

1

Custom Metrics Autoscaler Operator가 복제본을 지정된 값으로 스케일링하고 자동 스케일링을 중지하도록 지정합니다.

3.6.2. 확장 오브젝트의 사용자 정의 지표 자동 스케일러를 다시 시작

해당 **scaled Object**에 대한 **autoscaling.keda.sh/paused-replicas** 주석을 제거하여 일시 중지된 사용자 정의 지표 자동 스케일러를 다시 시작할 수 있습니다.

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...

```

절차

1. 다음 명령을 사용하여 워크로드에 대한 **scaledObject CR**을 편집합니다.

```
$ oc edit ScaledObject scaledobject
```

2. **autoscaling.keda.sh/paused-replicas** 주석을 제거합니다.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: '65729'
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

1

일시 중지된 사용자 정의 메트릭 자동 스케일러를 다시 시작하려면 이 주석을 제거합니다.

3.7. 감사 로그 수집

시스템의 개별 사용자, 관리자 또는 기타 구성 요소가 시스템에 영향을 준 활동 순서를 문서화하는 보안 관련 레코드 세트인 감사 로그를 수집할 수 있습니다.

예를 들어 감사 로그는 자동 스케일링 요청이 발생하는 위치를 이해하는 데 도움이 될 수 있습니다. 이는 사용자 애플리케이션에서 수행한 자동 확장 요청에 의해 백엔드가 과부하되는 경우 주요 정보입니다. 이로 인해 문제가 발생한 애플리케이션을 결정해야 합니다.

3.7.1. 감사 로깅 구성

CloudEventda Controller 사용자 정의 리소스를 편집하여 **Custom Metrics Autoscaler Operator**에 대한 감사를 구성할 수 있습니다. 로그는 **CloudEventda Controller CR**의 영구 볼륨 클레임을 사용하여 보안되는 볼륨의 감사 로그 파일로 전송됩니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator가 설치되어 있어야 합니다.**

절차

1. **CloudEventda Controller 사용자 지정 리소스를 편집하여 auditConfig 스텐자를 추가합니다.**

```

kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  # ...
  metricsServer:
  # ...
  auditConfig:
    logFormat: "json" 1
    logOutputVolumeClaim: "pvc-audit-log" 2
    policy:
      rules: 3
      - level: Metadata
      omitStages: "RequestReceived" 4
      omitManagedFields: false 5
    lifetime: 6
    maxAge: "2"
    maxBackup: "1"
    maxSize: "50"

```

1

audit 로그의 출력 형식(legacy 또는 json)을 지정합니다.

2

로그 데이터를 저장하기 위한 기존 영구 볼륨 클레임을 지정합니다. API 서버에 들어오는 모든 요청은 이 영구 볼륨 클레임에 기록됩니다. 이 필드를 비워 두면 로그 데이터가 **stdout**으로 전송됩니다.

3

기록해야 하는 이벤트와 어떤 데이터를 포함해야 하는지 지정합니다.

- **none:** 이벤트를 기록하지 않습니다.
-

metadata: 사용자, 타임스탬프 등과 같은 요청에 대한 메타데이터만 기록합니다. 요청 텍스트와 응답 텍스트를 기록하지 마십시오. 이는 기본값입니다.

- **request:** 메타데이터와 요청 텍스트만 기록하지만 응답 텍스트는 기록하지 않습니다. 이 옵션은 리소스가 아닌 요청에는 적용되지 않습니다.

- **RequestResponse:** 이벤트 메타데이터, 요청 텍스트 및 응답 텍스트를 기록합니다. 이 옵션은 리소스가 아닌 요청에는 적용되지 않습니다.

4

이벤트가 생성되지 않는 단계를 지정합니다.

5

요청 및 응답 본문의 관리 필드를 API 감사 로그에 작성하지 못하도록 하려면 **true** 로 필드를 생략할지 아니면 필드를 생략할 **false** 를 생략할지 여부를 지정합니다.

6

감사 로그의 크기 및 수명을 지정합니다.

- **maxAge:** 파일 이름에 인코딩된 타임스탬프에 따라 감사 로그 파일을 유지하는 최대 일 수입니다.

- **maxBackup:** 유지할 감사 로그 파일의 최대 수입니다. 모든 감사 로그 파일을 유지하려면 **0** 으로 설정합니다.

- **maxSize:** 감사 로그 파일의 최대 크기(MB)가 순환되기 전에 이루어집니다.

검증

1.

감사 로그 파일을 직접 확인합니다.

a.

keda-metrics-apiserver-* Pod 이름을 가져옵니다.

```
oc get pod -n openshift-keda
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
custom-metrics-autoscaler-operator-5cb44cd75d-9v4lv	1/1	Running	0	8m20s
keda-metrics-apiserver-65c7cc44fd-rrl4r	1/1	Running	0	2m55s
keda-operator-776cbb6768-zpj5b	1/1	Running	0	2m55s

b.

다음과 유사한 명령을 사용하여 로그 데이터를 확인합니다.

```
$ oc logs keda-metrics-apiserver-<hash>|grep -i metadata 1
```

1

선택 사항: **grep** 명령을 사용하여 표시할 로그 수준을 지정할 수 있습니다. 메타 데이터, 요청 ,Request Response.

예를 들면 다음과 같습니다.

```
$ oc logs keda-metrics-apiserver-65c7cc44fd-rrl4r|grep -i metadata
```

출력 예

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"4c81d41b-3dab-4675-90ce-20b87ce24013","stage":"ResponseComplete","requestURI":"/healthz","verb":"get","user":{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":["10.131.0.1"],"userAgent":"kube-probe/1.26","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2023-02-16T13:00:03.554567Z","stageTimestamp":"2023-02-16T13:00:03.555032Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
...
```

2.

또는 특정 로그를 볼 수 있습니다.

a.

keda-metrics-apiserver- Pod에 로그인하려면 다음과 유사한 명령을 사용합니다.*

```
$ oc rsh pod/keda-metrics-apiserver-<hash> -n openshift-keda
```

예를 들면 다음과 같습니다.

```
$ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n openshift-keda
```

b.

/var/audit-policy/ 디렉터리로 변경합니다.

```
sh-4.4$ cd /var/audit-policy/
```

c.

사용 가능한 로그를 나열합니다.

```
sh-4.4$ ls
```

출력 예

```
log-2023.02.17-14:50 policy.yaml
```

d.

필요한 경우 로그를 확인합니다.

```
sh-4.4$ cat <log_name>/<pvc_name>/grep -i <log_level> 1
```

1

선택 사항: grep 명령을 사용하여 표시할 로그 수준을 지정할 수 있습니다. 메타 데이터, 요청 ,Request Response.

예를 들면 다음과 같습니다.

```
sh-4.4$ cat log-2023.02.17-14:50/pvc-audit-log|grep -i Request
```

출력 예

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"63e7f68c-04ec-4f4d-8749-bf1656572a41","stage":"ResponseComplete","requestURI":"/openapi/v2","verb":"get","user":{"username":"system:aggregator","groups":["system:authenticated"]},"sourceIPs":["10.128.0.1"],"responseStatus":{"metadata":{"code":304},"requestReceivedTimestamp":"2023-02-17T13:12:55.035478Z","stageTimestamp":"2023-02-17T13:12:55.038346Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group \"system:authenticated\"}}}
...
```

3.8. 디버깅 데이터 수집

지원 사례를 여는 경우 클러스터에 대한 디버깅 정보를 **Red Hat** 지원에 제공하면 도움이 됩니다.

문제를 해결하려면 다음 정보를 입력합니다.

- **must-gather** 툴을 사용하여 수집된 데이터입니다.
- 고유한 클러스터 ID입니다.

must-gather 툴을 사용하여 다음 항목을 포함하여 **Custom Metrics Autoscaler Operator** 및 해당 구성 요소에 대한 데이터를 수집할 수 있습니다.

- **openshift-keda** 네임스페이스와 해당 하위 오브젝트입니다.

- **Custom Metric Autoscaler Operator 설치 오브젝트입니다.**
- **Custom Metric Autoscaler Operator CRD 오브젝트입니다.**

3.8.1. 디버깅 데이터 수집

다음 명령은 **Custom Metrics Autoscaler Operator**에 대한 **must-gather** 툴을 실행합니다.

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')
```

참고

표준 **OpenShift Container Platform must-gather** 명령 **oc adm must-gather**, **Custom Metrics Autoscaler Operator** 데이터를 수집하지 않습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **OpenShift Container Platform CLI(oc)**가 설치되어 있어야 합니다.

절차

1. **must-gather** 데이터를 저장하려는 디렉터리로 이동합니다.

참고

클러스터에서 제한된 네트워크를 사용하는 경우 추가 단계를 수행해야 합니다. 미리 레지스트리에 신뢰할 수 있는 **CA**가 있는 경우 먼저 신뢰할 수 있는 **CA**를 클러스터에 추가해야 합니다. 제한된 네트워크의 모든 클러스터에 대해 다음 명령을 실행하여 기본 **must-gather** 이미지를 이미지 스트림으로 가져와야 합니다.

```
$ oc import-image is/must-gather -n openshift
```

2.

다음 중 하나를 수행합니다.

- Custom Metrics Autoscaler Operator must-gather** 데이터만 가져오려면 다음 명령을 사용합니다.

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

must-gather 명령의 사용자 정의 이미지는 **Operator** 패키지 매니페스트에서 직접 가져와서 **Custom Metric Autoscaler Operator**를 사용할 수 있는 모든 클러스터에서 작동합니다.

- Custom Metric Autoscaler Operator** 정보 외에 기본 **must-gather** 데이터를 수집하려면 다음을 수행합니다.

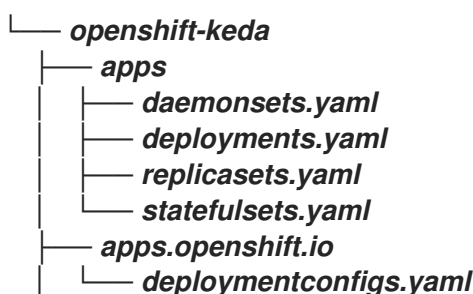
- 다음 명령을 사용하여 **Custom Metrics Autoscaler Operator** 이미지를 가져와서 환경 변수로 설정합니다.

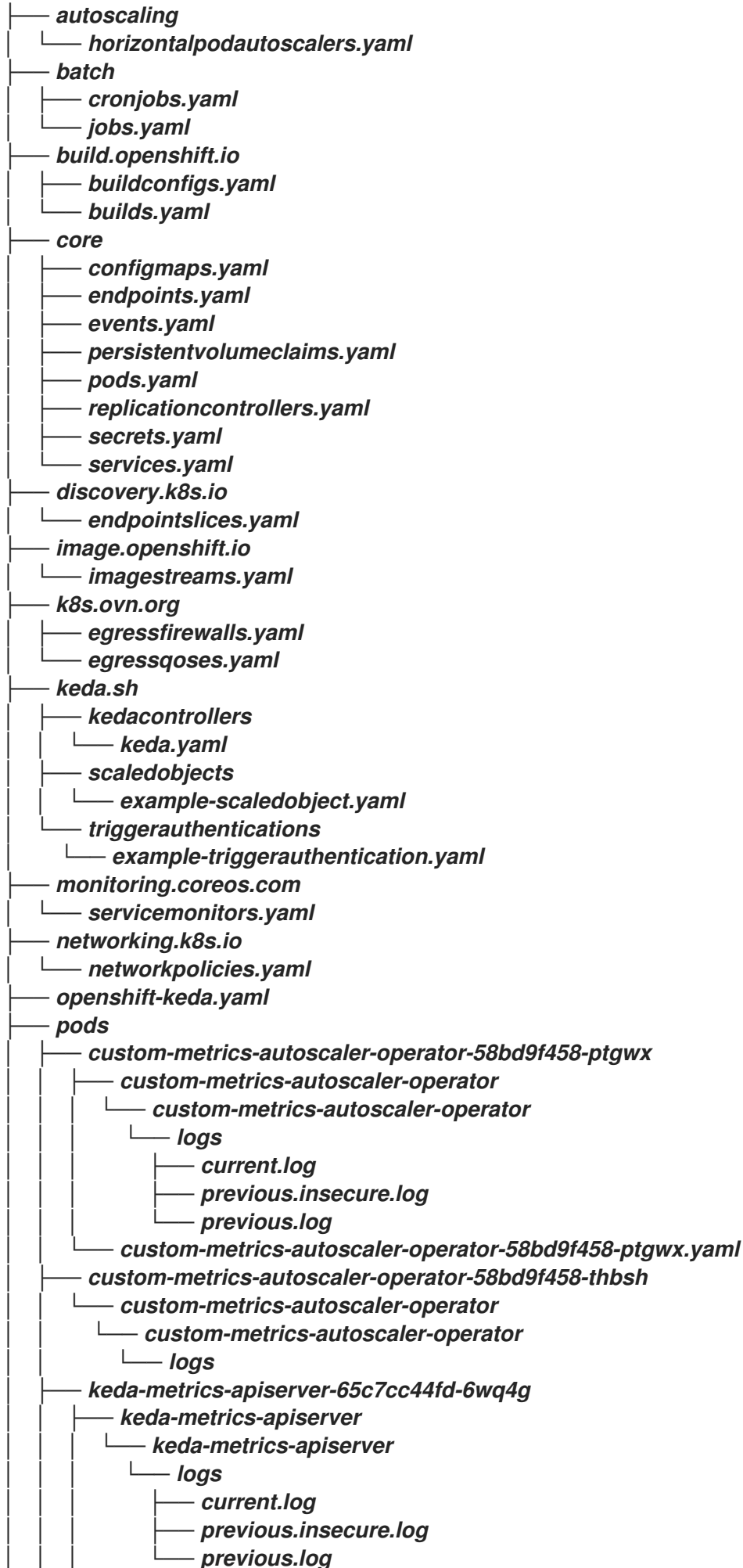
```
$ IMAGE="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

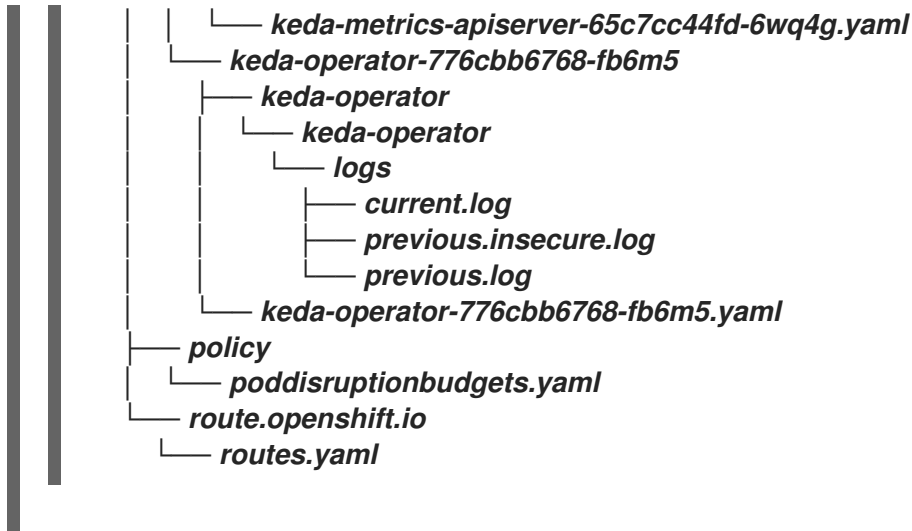
- 사용자 정의 **Metrics Autoscaler Operator** 이미지와 함께 **oc adm must-gather**를 사용합니다.

```
$ oc adm must-gather --image-stream=openshift/must-gather --image=${IMAGE}
```

예 3.1. Custom Metric Autoscaler의 must-gather 출력 예:







3. 작업 디렉터리에 생성된 **must-gather** 디렉터리의 압축 파일을 생성합니다. 예를 들어 Linux 운영 체제를 사용하는 컴퓨터에서 다음 명령을 실행합니다.

```
$ tar cvaf must-gather.tar.gz must-gather-local.5421342344627712289/ 1
```

1

must-gather-local.5421342344627712289/를 실제 디렉터리 이름으로 바꿉니다.

4. [Red Hat Customer Portal](#)에서 해당 지원 사례에 압축 파일을 첨부합니다.

3.9. OPERATOR 메트릭 보기

Custom Metrics Autoscaler Operator는 클러스터 내 모니터링 구성 요소에서 가져오는 즉시 사용 가능한 지표를 표시합니다. **PromQL(Prometheus Query Language)**을 사용하여 메트릭을 쿼리하여 문제를 분석하고 진단할 수 있습니다. 컨트롤러 Pod가 다시 시작되면 모든 메트릭이 재설정됩니다.

3.9.1. 성능 지표 액세스

OpenShift Container Platform 웹 콘솔을 사용하여 메트릭에 액세스하고 쿼리를 실행할 수 있습니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에서 관리자 화면을 선택합니다.

2. **Observe** → **Metrics** 를 선택합니다.
3. 사용자 지정 쿼리를 만들려면 **Expression** 필드에 **PromQL** 쿼리를 추가합니다.
4. 여러 쿼리를 추가하려면 **쿼리 추가**를 선택합니다.

3.9.1.1. 제공된 Operator 지표

Custom Metrics Autoscaler Operator는 **OpenShift Container Platform** 웹 콘솔을 사용하여 볼 수 있는 다음 메트릭을 노출합니다.

표 3.1. 사용자 정의 메트릭 자동 스케일러 Operator 메트릭

메트릭 이름	설명
keda_scaler_activity	특정 스케일러가 활성 상태인지 아니면 비활성인지 여부입니다. 값 1 은 scaler가 활성 상태임을 나타냅니다. 값 0 은 스케일러가 비활성 상태임을 나타냅니다.
keda_scaler_metrics_value	대상 평균을 계산하는 HPA(Horizontal Pod Autoscaler)에서 사용하는 각 스케일러의 메트릭의 현재 값입니다.
keda_scaler_metrics_latency	각 스케일러에서 현재 메트릭을 검색하는 대기 시간입니다.
keda_scaler_errors	각 확장기에 발생한 오류 수입니다.
keda_scaler_errors_total	모든 scalers에 대해 발생한 총 오류 수입니다.
keda_scaled_object_errors	스케일링된 각 object에 대해 발생한 오류 수입니다.
keda_resource_totals	각 사용자 정의 리소스 유형의 각 네임스페이스에 있는 총 사용자 정의 메트릭 자동 스케일러 사용자 정의 리소스의 수입니다.
keda_trigger_totals	트리거 유형별 총 트리거 수입니다.

사용자 정의 메트릭 자동 스케일러 Admission Webhook 메트릭

사용자 정의 지표 자동 스케일러 **Admission Webhook**에서는 다음 **Prometheus** 메트릭도 노출합니다.

메트릭 이름	설명
keda_scaled_object_validation_total	스케일링된 오브젝트 검증 수입니다.
keda_scaled_object_validation_errors	검증 오류 수입니다.

3.10. 사용자 정의 메트릭 자동 스케일러 추가 방법

사용자 지정 지표 자동 스케일러를 추가하려면 배포, 상태 저장 세트 또는 사용자 정의 리소스에 대한 **ScaledObject** 사용자 정의 리소스를 생성합니다. 작업에 대한 **scaledJob** 사용자 정의 리소스를 생성합니다.

스케일링할 각 워크로드에 대해 하나의 확장 오브젝트만 생성할 수 있습니다. 또한 동일한 워크로드에서 스케일링된 오브젝트와 **HPA**(수평 Pod 자동 스케일러)를 사용할 수 없습니다.

3.10.1. 워크로드에 사용자 정의 메트릭 자동 스케일러 추가

Deployment, StatefulSet 또는 사용자 정의 리소스 오브젝트로 생성된 워크로드에 대한 사용자 정의 지표 자동 스케일러를 생성할 수 있습니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator**가 설치되어 있어야 합니다.
- **CPU** 또는 메모리를 기반으로 스케일링에 사용자 정의 메트릭 자동 스케일러를 사용하는 경우:
 - 클러스터 관리자가 클러스터 메트릭을 올바르게 구성해야 합니다. **oc describe PodMetrics <pod-name>** 명령을 사용하여 메트릭이 구성되어 있는지 확인할 수 있습니다. 메트릭이 구성된 경우 출력이 다음과 유사하게 표시되고 **Usage**에 **CPU** 및 메모리가 표시됩니다.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

출력 예

```

Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind: PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link: /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-
scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp: 2019-05-23T18:47:56Z
  Window: 1m0s
  Events: <none>

```

○

스케일링하려는 오브젝트와 연결된 Pod에는 지정된 메모리 및 CPU 제한이 포함되어야 합니다. 예를 들면 다음과 같습니다.

Pod 사양의 예

```

apiVersion: v1
kind: Pod
# ...
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
# ...

```

1.

다음과 유사한 **YAML** 파일을 생성합니다. **<2>** 이름, 오브젝트 이름 **<4>** 및 오브젝트 종류 **<5>** 만 필요합니다.

스케일링된 오브젝트의 예

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "0" 1
  name: scaledobject 2
  namespace: my-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
    name: example-deployment 4
    kind: Deployment 5
    envSourceContainerName: .spec.template.spec.containers[0] 6
  cooldownPeriod: 200 7
  maxReplicaCount: 100 8
  minReplicaCount: 0 9
  metricsServer: 10
  auditConfig:
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
    policy:
      rules:
        - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
      lifetime:
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
  fallback: 11
  failureThreshold: 3
  replicas: 6
  pollingInterval: 30 12
  advanced:
    restoreToOriginalReplicaCount: false 13
    horizontalPodAutoscalerConfig:
      name: keda-hpa-scale-down 14
    behavior: 15
      scaleDown:
        stabilizationWindowSeconds: 300
        policies:
          - type: Percent
            value: 100
            periodSeconds: 15

```

triggers:

- type: prometheus **16**

metadata:

serverAddress: <https://thanos-querier.openshift-monitoring.svc.cluster.local:9092>

namespace: kedatest

metricName: http_requests_total

threshold: '5'

query: `sum(rate(http_requests_total{job="test-app"})[1m])`

authModes: basic

authenticationRef: **17**

name: prom-triggerauthentication

kind: TriggerAuthentication

1

선택 사항: **Custom Metrics Autoscaler Operator**가 "작업에 대한 사용자 정의 지표 자동 스케일러 사용" 섹션에 설명된 대로 복제본을 지정된 값으로 스케일링하고 자동 스케일링을 중지하도록 지정합니다.

2

이 사용자 정의 메트릭 자동 스케일러의 이름을 지정합니다.

3

선택 사항: 대상 리소스의 **API** 버전을 지정합니다. 기본값은 **apps/v1** 입니다.

4

스케일링할 개체의 이름을 지정합니다.

5

kind 를 **Deployment**, **StatefulSet** 또는 **CustomResource** 로 지정합니다.

6

선택 사항: 사용자 정의 메트릭 자동 스케일러가 시크릿을 보유하는 환경 변수를 가져 오는 대상 리소스의 컨테이너 이름을 지정합니다. 기본값은 **.spec.template.spec.containers[0].spec**입니다.

7

선택 사항: **minReplicaCount** 가 **0** 으로 설정된 경우 배포를 **0** 으로 스케일링하기 전에 마지막 트리거가 보고될 때까지 대기하는 기간(초)을 지정합니다. 기본값은 **300** 입니다.

8

선택 사항: 확장 시 최대 복제본 수를 지정합니다. 기본값은 **100**입니다.

9

선택 사항: 축소 시 최소 복제본 수를 지정합니다.

10

선택 사항: "감사 로깅 구성" 섹션에 설명된 대로 감사 로그의 매개변수를 지정합니다.

11

선택 사항: 스케일링기가 **failureThreshold** 매개변수에 정의된 횟수에 대한 소스에서 메트릭을 가져오지 못하는 경우 대체할 복제본 수를 지정합니다. 대체 동작에 대한 자세한 내용은 [KEDA 설명서](#) 를 참조하십시오.

12

선택 사항: 각 트리거를 확인하는 간격(초)을 지정합니다. 기본값은 **30** 입니다.

13

선택 사항: 스케일링된 오브젝트가 삭제된 후 대상 리소스를 원래 복제본 수로 확장할 지 여부를 지정합니다. 기본값은 **false** 이며 스케일링된 오브젝트가 삭제될 때 복제본 수를 유지합니다.

14

선택 사항: 수평 **Pod** 자동 스케일러의 이름을 지정합니다. 기본값은 **keda-hpa-{scaled-object-name}** 입니다.

15

선택 사항: "스케이딩 정책" 섹션에 설명된 대로 **Pod**를 확장 또는 확장하는 데 사용할 스케일링 정책을 지정합니다.

16

"사용자 정의 지표 자동 스케일러 트리거" 섹션에 설명된 대로 스케일링 기반으로 사용할 트리거를 지정합니다. 이 예에서는 **OpenShift Container Platform** 모니터링을 사용합니다.

17

선택 사항: 트리거 인증 또는 클러스터 트리거 인증을 지정합니다. 자세한 내용은 추가 리소스 섹션에서 사용자 정의 메트릭 자동 스케일러 트리거 인증 이해 를 참조하십시오.

- 트리거 인증을 사용하려면 **TriggerAuthentication** 을 입력합니다. 이는 기본값입니다.
- 클러스터 트리거 인증을 사용하려면 **ClusterTriggerAuthentication** 을 입력합니다.

2.

다음 명령을 실행하여 사용자 정의 메트릭 자동 스케일러를 생성합니다.

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

검증

- 명령 출력을 보고 사용자 정의 지표 자동 스케일러가 생성되었는지 확인합니다.

```
$ oc get scaledobject <scaled_object_name>
```

출력 예

```

NAME          SCALETARGETKIND  SCALETARGETNAME  MIN  MAX  TRIGGERS
AUTHENTICATION  READY  ACTIVE  FALLBACK  AGE
scaledobject  apps/v1.Deployment  example-deployment  0  50  prometheus
prom-triggerauthentication  True  True  True  17s

```

출력에서 다음 필드를 기록해 둡니다.

- 사용 중인 트리거 또는 축소자를 나타냅니다. **Indicates the trigger, or scaler, that is being used.**
- **AUTHENTICATION:** 사용 중인 트리거 인증의 이름을 표시합니다.

- 확장된 개체가 스케일링을 시작할 준비가 되었는지 여부를 나타냅니다.
 - **True** 이면 확장된 오브젝트가 준비됩니다.
 - **False** 인 경우 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 확장된 오브젝트가 준비되지 않습니다.

- **Net Namespace** : 스케일링이 발생했는지를 나타냅니다.
 - **True** 가 되면 확장이 수행됩니다.
 - **False** 인 경우 지표가 없거나 사용자가 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 스케일링이 발생하지 않습니다.

- **FALLBACK**: 사용자 정의 메트릭 자동 스케일러가 소스에서 메트릭을 가져올 수 있는지 여부를 나타냅니다.
 - **False** 인 경우 사용자 지정 지표 자동 스케일러에 메트릭이 수신됩니다.
 - **True** 인 경우 사용자 지정 지표 자동 스케일러는 메트릭이 없거나 사용자가 생성한 하나 이상의 오브젝트에 문제가 있기 때문에 메트릭이 수신됩니다.

3.10.2. 작업에 사용자 정의 메트릭 자동 스케일러 추가

모든 **Job** 오브젝트에 대한 사용자 지정 지표 자동 스케일러를 생성할 수 있습니다.

중요

확장된 작업을 사용하여 스케일링하는 것은 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.

사전 요구 사항

-

Custom Metrics Autoscaler Operator가 설치되어 있어야 합니다.

프로세스

- 1.

다음과 유사한 YAML 파일을 생성합니다.

```
kind: ScaledJob
apiVersion: keda.sh/v1alpha1
metadata:
  name: scaledjob
  namespace: my-namespace
spec:
  failedJobsHistoryLimit: 5
  jobTargetRef:
    activeDeadlineSeconds: 600 1
    backoffLimit: 6 2
    parallelism: 1 3
    completions: 1 4
    template: 5
      metadata:
        name: pi
      spec:
        containers:
          - name: pi
            image: perl
            command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
  maxReplicaCount: 100 6
  pollingInterval: 30 7
  successfulJobsHistoryLimit: 5 8
  failedJobsHistoryLimit: 5 9
  envSourceContainerName: 10
  rolloutStrategy: gradual 11
  scalingStrategy: 12
```

```

strategy: "custom"
customScalingQueueLengthDeduction: 1
customScalingRunningJobPercentage: "0.5"
pendingPodConditions:
  - "Ready"
  - "PodScheduled"
  - "AnyOtherCustomPodCondition"
multipleScalersCalculation : "max"
triggers:
- type: prometheus 13
  metadata:
    serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
    namespace: kedatest
    metricName: http_requests_total
    threshold: '5'
    query: sum(rate(http_requests_total{job="test-app"})[1m]))
    authModes: "bearer"
  authenticationRef: 14
    name: prom-cluster-triggerauthentication

```

1

작업을 실행할 수 있는 최대 기간을 지정합니다.

2

작업 재시도 횟수를 지정합니다. 기본값은 6 입니다.

3

선택 사항: 작업에서 병렬로 실행해야 하는 Pod 복제본 수를 지정합니다. 기본값은 1 입니다.

-

비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않으면 기본값은 1 입니다.

4

선택 사항: 작업이 완료된 것으로 표시하는 데 필요한 성공적인 Pod 완료 횟수를 지정합니다.

-

비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않으면 기본값은 1 입니다.

-

완료 횟수가 고정된 병렬 작업의 경우 완료 횟수를 지정합니다.

- 작업 큐가 있는 병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 **parallelism** 매개변수의 값입니다.

5

컨트롤러에서 생성하는 **Pod**에 대한 템플릿을 지정합니다.

6

선택 사항: 확장 시 최대 복제본 수를 지정합니다. 기본값은 **100**입니다.

7

선택 사항: 각 트리거를 확인하는 간격(초)을 지정합니다. 기본값은 **30**입니다.

8

선택 사항: 유지해야 하는 성공적으로 완료된 작업의 수를 지정합니다. 기본값은 **100**입니다.

9

선택 사항: 유지해야 하는 실패한 작업의 수를 지정합니다. 기본값은 **100**입니다.

10

선택 사항: 사용자 정의 자동 스케일러가 시크릿을 포함하는 환경 변수를 가져오는 데 상 리소스의 컨테이너 이름을 지정합니다. 기본값은 **.spec.template.spec.containers[0].spec**입니다.

11

선택 사항: 스케일링된 작업이 업데이트될 때마다 기존 작업이 종료되는지 여부를 지정합니다.

- **Default:** 자동 스케일러는 연결된 스케일링된 작업이 업데이트되면 기존 작업을 종료합니다. 자동 스케일러는 최신 사양을 사용하여 작업을 다시 생성합니다.

- **gradual:** 연결된 확장된 작업이 업데이트되면 자동 스케일러가 기존 작업을 종료하지 않습니다. 자동 스케일러는 최신 사양을 사용하여 새 작업을 생성합니다.

12

13

14

선택 사항: 트리거 인증 또는 클러스터 트리거 인증을 지정합니다. 자세한 내용은 추가 리소스 섹션에서 사용자 정의 메트릭 자동 스케일러 트리거 인증 이해 를 참조하십시오.

- 트리거 인증을 사용하려면 **TriggerAuthentication** 을 입력합니다. 이는 기본값입니다.
- 클러스터 트리거 인증을 사용하려면 **ClusterTriggerAuthentication** 을 입력합니다.

2.

다음 명령을 실행하여 사용자 정의 메트릭 자동 스케일러를 생성합니다.

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

검증

- 명령 출력을 보고 사용자 정의 지표 자동 스케일러가 생성되었는지 확인합니다.

```
$ oc get scaledjob <scaled_job_name>
```

출력 예

```
NAME          MAX TRIGGERS  AUTHENTICATION  READY ACTIVE AGE
scaledjob    100 prometheus  prom-triggerauthentication  True  True  8s
```

출력에서 다음 필드를 기록해 둡니다.

- 사용 중인 트리거 또는 축소자를 나타냅니다. *Indicates the trigger, or scaler, that is being used.*

- **AUTHENTICATION:** 사용 중인 트리거 인증의 이름을 표시합니다.
- 확장된 개체가 스케일링을 시작할 준비가 되었는지 여부를 나타냅니다.
 - **True** 이면 확장된 오브젝트가 준비됩니다.
 - **False** 인 경우 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 확장된 오브젝트가 준비되지 않습니다.
- **Net Namespace :** 스케일링이 발생했는지를 나타냅니다.
 - **True** 가 되면 확장이 수행됩니다.
 - **False** 인 경우 지표가 없거나 사용자가 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 스케일링이 발생하지 않습니다.

3.10.3. 추가 리소스

- [사용자 정의 메트릭 자동 스케일러 트리거 인증 이해](#)

3.11. CUSTOM METRICS AUTOSCALER OPERATOR 제거

OpenShift Container Platform 클러스터에서 사용자 정의 메트릭 자동 스케일러를 제거할 수 있습니다. **Custom Metrics Autoscaler Operator**를 제거한 후 **Operator**와 관련된 다른 구성 요소를 제거하여 잠재적인 문제를 방지합니다.



참고

KedaController CR(사용자 정의 리소스)을 먼저 삭제합니다. **KedaController CR**을 삭제하지 않으면 **openshift-keda** 프로젝트를 삭제할 때 **OpenShift Container Platform**이 중단될 수 있습니다. **CR**을 삭제하기 전에 **Custom Metrics Autoscaler Operator**를 삭제하는 경우 **CR**을 삭제할 수 없습니다.


3.11.1. Custom Metrics Autoscaler Operator 설치 제거

다음 절차에 따라 OpenShift Container Platform 클러스터에서 사용자 정의 메트릭 자동 스케일러를 제거합니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator가 설치되어 있어야 합니다.**

절차

1. **OpenShift Container Platform 웹 콘솔에서 Operator → 설치된 Operator를 클릭합니다.**
2. **openshift-keda 프로젝트로 전환합니다.**
3. **KedaController 사용자 정의 리소스를 제거합니다.**
 - a. **CustomMetricsAutoscaler Operator를 찾아 KedaController 탭을 클릭합니다.**
 - b. **사용자 정의 리소스를 찾은 다음 KedaController 삭제를 클릭합니다.**
 - c. **제거를 클릭합니다.**
4. **Custom Metrics Autoscaler Operator를 제거합니다.**
 - a. **Operators → 설치된 Operators를 클릭합니다.**
 - b. **CustomMetricsAutoscaler Operator를 찾아 옵션 메뉴**

를 클릭하고 Operator 설치 제거를 선택합니다.

c.

제거를 클릭합니다.

5.

선택 사항: **OpenShift CLI**를 사용하여 사용자 정의 지표 자동 스케일러 구성 요소를 제거합니다.

a.

사용자 정의 지표 자동 스케일러 **CRD**를 삭제합니다.

- `clustertriggerauthentications.keda.sh`
- `kedacontrollers.keda.sh`
- `scaledjobs.keda.sh`
- `scaledobjects.keda.sh`
- `triggerauthentications.keda.sh`

```
$ oc delete crd clustertriggerauthentications.keda.sh kedacontrollers.keda.sh
scaledjobs.keda.sh scaledobjects.keda.sh triggerauthentications.keda.sh
```

CRD를 삭제하면 연결된 역할, 클러스터 역할 및 역할 바인딩이 제거됩니다. 그러나 수동으로 삭제해야 하는 몇 가지 클러스터 역할이 있을 수 있습니다.

b.

사용자 정의 지표 자동 스케일러 클러스터 역할을 나열합니다.

```
$ oc get clusterrole | grep keda.sh
```

c.

나열된 사용자 정의 지표 자동 스케일러 클러스터 역할을 삭제합니다. 예를 들면 다음과 같습니다.

```
$ oc delete clusterrole.keda.sh-v1alpha1-admin
```

d.

사용자 정의 메트릭 자동 스케일러 클러스터 역할 바인딩을 나열합니다.

```
$ oc get clusterrolebinding | grep keda.sh
```

e.

나열된 사용자 정의 지표 자동 스케일러 클러스터 역할 바인딩을 삭제합니다. 예를 들면 다음과 같습니다.

```
$ oc delete clusterrolebinding.keda.sh-v1alpha1-admin
```

6.

사용자 정의 지표 자동 스케일러 프로젝트를 삭제합니다.

```
$ oc delete project openshift-keda
```

7.

Cluster Metric Autoscaler Operator를 삭제합니다.

```
$ oc delete operator/openshift-custom-metrics-autoscaler-operator.openshift-keda
```

4장. 노드에 대한 POD 배치 제어(예약)

4.1. 스케줄러를 사용하여 POD 배치 제어

Pod 예약은 클러스터 내 노드에 대한 새 Pod 배치를 결정하는 내부 프로세스입니다.

스케줄러 코드는 새 Pod가 생성될 때 해당 Pod를 감시하고 이를 호스팅하는 데 가장 적합한 노드를 확인할 수 있도록 깔끔하게 분리되어 있습니다. 그런 다음 마스터 API를 사용하여 Pod에 대한 바인딩(Pod와 노드의 바인딩)을 생성합니다.

기본 Pod 예약

OpenShift Container Platform에는 대부분의 사용자의 요구 사항을 충족하는 기본 스케줄러가 제공됩니다. 기본 스케줄러는 고유 톨과 사용자 정의 톨을 모두 사용하여 Pod에 가장 적합한 항목을 결정합니다.

고급 Pod 예약

새 Pod가 배치되는 위치를 더 많이 제어해야 하는 경우 **OpenShift Container Platform** 고급 예약 기능을 사용하여 Pod가 필요하거나 특정 노드에서 또는 특정 Pod와 함께 실행되도록 Pod를 구성할 수 있습니다.

다음 예약 기능을 사용하여 Pod 배치를 제어할 수 있습니다.

- [스케줄러 프로파일](#)
- [Pod 유사성 및 유사성 방지 규칙](#)
- [노드 유사성](#)
- [노드 선택기](#)
- [테인트\(Taints\) 및 톨러레이션\(Tolerations\)](#)
- [노드 과다 할당](#)

4.1.1. 기본 스케줄러 정보

기본 **OpenShift Container Platform Pod** 스케줄러는 클러스터 내의 노드에 새 **Pod** 배치를 결정합니다. **Pod**에서 데이터를 읽고 구성된 프로필을 기반으로 하는 노드를 찾습니다. 완전히 독립적이며 독립 실행형 솔루션으로 존재합니다. **Pod**를 수정하지 않습니다. **Pod**를 특정 노드에 연결하는 **Pod** 바인딩이 생성됩니다.

4.1.1.1. 기본 예약 이해

기본 스케줄러는 3단계 작업에서 **Pod**를 호스팅할 노드를 선택하는 기본 플랫폼 제공 스케줄러 엔진에 해당합니다.

노드를 필터링합니다.

사용 가능한 노드를 지정된 제약 조건 또는 요구 사항에 따라 필터링합니다. 이는 서술자 라는 필터 함수 목록을 통해 각 노드를 실행하거나 필터를 통해 수행됩니다.

필터링된 노드 목록의 우선순위를 지정합니다.

이 작업은 일련의 우선 순위 또는 점수 점수를 통해 0 - 10 사이의 점수를 할당하는 함수를 통해 달성되며, 0은 **Pod**를 호스팅하는 데 적합하지 않음을 나타내고 10은 **Pod**를 호스팅하는 데 적합합니다. 스케줄러 구성은 각 점수 기능에 대해 간단한 가중치 (정의 숫자 값)를 사용할 수도 있습니다. 각 점수 함수에서 제공하는 노드 점수에 가중치(대부분 점수의 기본 가중치는 1임)를 곱한 다음 모든 점수에서 제공하는 각 노드의 점수를 추가하여 결합합니다. 관리자는 이 **weight** 속성을 사용하여 일부 점수에 높은 중요성을 부여할 수 있습니다.

최적의 노드 선택

노드는 해당 점수에 따라 정렬되며 점수가 가장 높은 노드가 **Pod**를 호스팅하도록 선택됩니다. 여러 노드의 점수가 동일한 경우 해당 노드 중 하나가 무작위로 선택됩니다.

4.1.2. 스케줄러 사용 사례

OpenShift Container Platform 내에서 예약하는 중요 사용 사례 중 하나는 유연한 유사성 및 유사성 방지 정책을 지원하는 것입니다.

4.1.2.1. 인프라 토폴로지 수준

관리자는 노드에 라벨을 지정하여 인프라(노드)에 다양한 토폴로지 수준을 정의할 수 있습니다. 예를 들면 **region=r1, zone=z1, rack=s1**과 같습니다.

이러한 라벨 이름에는 특별한 의미가 없으며 관리자는 도시/빌딩/방과 같은 인프라 수준의 이름을 자유롭게 지정할 수 있습니다. 또한 관리자는 인프라 토폴로지에 원하는 수의 수준을 정의할 수 있으며 일반

적으로 세 가지 수준이 적합합니다(예: **region** → **zones** → **racks**). 관리자는 모든 조합에 유사성 및 유사성 방지 규칙을 지정할 수 있습니다.

4.1.2.2. 유사성

관리자는 임의의 토폴로지 수준 또는 여러 수준에도 유사성을 지정하도록 스케줄러를 구성할 수 있어야 합니다. 특정 수준의 유사성은 동일한 서비스에 속하는 모든 **Pod**가 동일한 수준에 속하는 노드에 예약됨을 나타냅니다. 이렇게 하면 관리자가 피어 **Pod**가 지리적으로 너무 멀리 떨어져 있지 않도록 할 수 있어 애플리케이션의 대기 시간 요구 사항이 처리됩니다. 동일한 유사성 그룹 내에서 **Pod**를 호스팅할 수 있는 노드가 없는 경우 **Pod**를 예약하지 않습니다.

Pod가 예약되는 위치를 더 잘 제어해야 하는 경우 노드 유사성 규칙을 사용하여 노드에서 **Pod** 배치 제어 및 유사성 및 유사성 방지 규칙을 사용하여 다른 **Pod**와 관련된 **Pod** 배치를 참조하십시오.

관리자는 이러한 고급 예약 기능을 사용하여 **Pod**를 예약할 수 있는 노드를 지정하고 기타 **Pod**와 관련된 예약을 강제 적용하거나 거부할 수 있습니다.

4.1.2.3. 유사성 방지

관리자는 임의의 토폴로지 수준 또는 여러 수준에도 유사성 방지를 지정하도록 스케줄러를 구성할 수 있어야 합니다. 특정 수준의 유사성 방지(또는 '분배')는 동일한 서비스에 속하는 모든 **Pod**가 해당 수준에 속하는 노드에 분배되어 있음을 나타냅니다. 이 경우 고가용성을 위해 애플리케이션이 잘 분배됩니다. 스케줄러는 적용 가능한 모든 노드에서 가능한 한 균등하게 서비스 **Pod**의 균형을 맞추려고 합니다.

Pod가 예약되는 위치를 더 잘 제어해야 하는 경우 노드 유사성 규칙을 사용하여 노드에서 **Pod** 배치 제어 및 유사성 및 유사성 방지 규칙을 사용하여 다른 **Pod**와 관련된 **Pod** 배치를 참조하십시오.

관리자는 이러한 고급 예약 기능을 사용하여 **Pod**를 예약할 수 있는 노드를 지정하고 기타 **Pod**와 관련된 예약을 강제 적용하거나 거부할 수 있습니다.

4.2. 스케줄러 프로필을 사용하여 POD 예약

예약 프로필을 사용하여 클러스터 내의 노드에 **Pod**를 예약하도록 **OpenShift Container Platform**을 구성할 수 있습니다.

4.2.1. 스케줄러 프로필 정보

스케줄러 프로필을 지정하여 노드에 **Pod**를 예약하는 방법을 제어할 수 있습니다.

다음 스케줄러 프로필을 사용할 수 있습니다.

LowNodeUtilization

이 프로필은 여러 노드에 **Pod**를 균등하게 분배하여 노드당 리소스 사용량을 줄입니다. 이 프로필은 기본 스케줄러 동작을 제공합니다.

HighNodeUtilization

이 프로필은 가능한 한 많은 **Pod**를 가능한 한 소수의 노드에 배치하려고 합니다. 이렇게 하면 노드 수가 최소화되고 노드당 리소스 사용량이 늘어납니다.

NoScoring

모든 점수 플러그인을 비활성화하여 가장 빠른 스케줄링 주기를 위해 대기 시간이 짧은 프로필입니다. 이렇게 하면 보다 신속하게 더 나은 예약 결정을 내릴 수 있습니다.

4.2.2. 스케줄러 프로필 구성

스케줄러 프로필을 사용하도록 스케줄러를 구성할 수 있습니다.

사전 요구 사항

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

프로세스

1. **Scheduler** 오브젝트를 편집합니다.

```
$ oc edit scheduler cluster
```

2. **spec.profile** 필드에 사용할 프로필을 지정합니다.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
#...
```

```
spec:
  mastersSchedulable: false
  profile: HighNodeUtilization 1
#...
```

1

LowNodeUtilization, HighNodeUtilization 또는 **NoScoring**으로 설정합니다.

3.

파일을 저장하여 변경 사항을 적용합니다.

4.3. 유사성 및 유사성 방지 규칙을 사용하여 다른 POD에 상대적인 POD 배치

유사성은 예약할 노드를 제어하는 Pod의 속성입니다. 유사성 방지는 Pod가 노드에서 예약되지 않도록 하는 Pod의 속성입니다.

OpenShift Container Platform에서 Pod 유사성 및 Pod 유사성 방지를 사용하면 다른 Pod의 키 값 라벨에 따라 Pod를 예약할 수 있는 노드를 제한할 수 있습니다.

4.3.1. Pod 유사성 이해

Pod 유사성 및 Pod 유사성 방지를 사용하면 다른 Pod의 키/값 라벨에 따라 Pod를 예약할 수 있는 노드를 제한할 수 있습니다.

•

Pod 유사성을 사용하면 새 Pod의 라벨 선택기가 현재 Pod의 라벨과 일치하는 경우 다른 Pod와 동일한 노드에서 새 Pod를 찾도록 스케줄러에 지시할 수 있습니다.

•

Pod 유사성 방지를 사용하면 새 Pod의 라벨 선택기가 현재 Pod의 라벨과 일치하는 경우 스케줄러에서 동일한 라벨을 사용하여 Pod와 동일한 노드에서 새 Pod를 찾지 않도록 할 수 있습니다.

예를 들어 유사성 규칙을 사용하여 서비스 내에서 또는 다른 서비스의 Pod와 관련하여 Pod를 분배하거나 패키징할 수 있습니다. 유사성 방지 규칙을 사용하면 특정 서비스의 Pod가 첫 번째 서비스의 Pod 성능을 방해하는 것으로 알려진 다른 서비스의 Pod와 동일한 노드에 예약되지 않도록 할 수 있습니다. 또는 서비스의 Pod를 노드, 가용성 영역 또는 가용성 집합에 분배하여 관련 오류를 줄일 수 있습니다.



참고

라벨 선택기는 여러 Pod 배포가 있는 Pod와 일치할 수 있습니다. 일치하는 Pod를 방지하도록 유사성 방지 규칙을 구성할 때 라벨의 고유한 조합을 사용합니다.

Pod 유사성 규칙에는 필수 및 기본 두 가지의 유형이 있습니다.

노드에 Pod를 예약하려면 먼저 필수 규칙을 충족해야 합니다. 기본 규칙은 규칙이 충족되는 경우 스케줄러가 규칙을 적용하려고 하지만 반드시 적용되는 것은 아닙니다.



참고

Pod 우선순위 및 선점 설정에 따라 유사성 요구 사항을 위반하지 않으면 스케줄러에서 Pod에 적절한 노드를 찾지 못하는 경우가 있습니다. 이 경우 Pod를 예약하지 못할 수 있습니다.

이러한 상황을 방지하려면 우선순위가 같은 Pod를 사용하여 Pod 유사성을 신중하게 구성합니다.

Pod 사양 파일을 통해 Pod 유사성/유사성 방지를 구성합니다. 필수 규칙, 기본 규칙 또는 둘 다 지정할 수 있습니다. 둘 다 지정하는 경우 노드는 먼저 필수 규칙을 충족한 다음 기본 규칙을 충족하려고 합니다.

다음 예제에서는 Pod 유사성 및 유사성 방지를 위해 구성된 Pod 사양을 보여줍니다.

이 예제에서 Pod 유사성 규칙은 노드에 이미 실행 중인 Pod가 한 개 이상 있고 키가 security이고 값이 S1인 라벨이 있는 경우에만 노드에 Pod를 예약할 수 있음을 나타냅니다. Pod 유사성 방지 규칙은 노드에서 이미 Pod를 실행 중이고 키가 security이고 값이 S2인 라벨이 있는 경우 Pod를 노드에 예약하지 않는 것을 선호함을 나타냅니다.

Pod 유사성이 포함된 샘플 Pod 구성 파일

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
```



```

affinity:
  podAffinity: ①
    requiredDuringSchedulingIgnoredDuringExecution: ②
      - labelSelector:
          matchExpressions:
            - key: security ③
              operator: In ④
              values:
                - S1 ⑤
          topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod

```

①

Pod 유사성을 구성하는 스탠자입니다.

②

필요한 규칙을 정의합니다.

③ ⑤

규칙을 적용하려면 일치해야 하는 키 및 값(라벨)입니다.

④

이 연산자는 기존 Pod의 라벨과 새 Pod 사양에 있는 `matchExpression` 매개변수의 값 집합 간의 관계를 나타냅니다. `In`, `NotIn`, `Exists` 또는 `DoesNotExist`일 수 있습니다.

Pod 유사성 방지가 포함된 샘플 Pod 구성 파일

```

apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: ①
      preferredDuringSchedulingIgnoredDuringExecution: ②
        - weight: 100 ③
      podAffinityTerm:

```

```

labelSelector:
  matchExpressions:
    - key: security 4
      operator: In 5
      values:
        - S2
    topologyKey: kubernetes.io/hostname
containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod

```

1

Pod 유사성 방지를 구성하는 스탠자입니다.

2

기본 규칙을 정의합니다.

3

기본 규칙의 가중치를 지정합니다. 가중치가 가장 높은 노드가 우선합니다.

4

유사성 방지 규칙이 적용되는 시기를 결정하는 Pod 라벨에 대한 설명입니다. 라벨의 키와 값을 지정합니다.

5

이 연산자는 기존 Pod의 라벨과 새 Pod 사양에 있는 matchExpression 매개변수의 값 집합 간의 관계를 나타냅니다. In, NotIn, Exists 또는 DoesNotExist일 수 있습니다.



참고

런타임 시 노드의 라벨이 변경되어 Pod의 유사성 규칙이 더 이상 충족되지 않는 경우 Pod가 노드에서 계속 실행됩니다.

4.3.2. Pod 유사성 규칙 구성

다음 단계에서는 라벨이 있는 Pod 및 유사성을 사용하여 해당 Pod에 예약할 수 있는 Pod를 생성하는 간단한 2-Pod 구성을 보여줍니다.



참고

예약된 Pod에 선호도를 직접 추가할 수 없습니다.

프로세스

1.

Pod 사양에서 특정 라벨을 사용하여 Pod를 생성합니다.

a.

다음 콘텐츠를 사용하여 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

b.

Pod를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

2.

다른 Pod를 생성할 때 유사성을 추가하도록 다음 매개변수를 구성합니다.

a.

다음 콘텐츠를 사용하여 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1-east
#...
spec
  affinity 1
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: 2
      - labelSelector:
          matchExpressions:
            - key: security 3
```

```

values:
- S1
operator: In 4
topologyKey: topology.kubernetes.io/zone 5
#...

```

1

Pod 유사성을 추가합니다.

2

`requiredDuringSchedulingIgnoredDuringExecution` 매개변수 또는 `preferredDuringSchedulingIgnoredDuringExecution` 매개변수를 구성합니다.

3

충족해야 하는 키와 값을 지정합니다. 새 Pod를 다른 Pod와 함께 예약하려면 첫 번째 Pod의 라벨과 동일한 key 및 values 매개변수를 사용합니다.

4

연산자를 지정합니다. 연산자는 In, NotIn, Exists 또는 DoesNotExist일 수 있습니다. 예를 들어 노드에 라벨이 있어야 하는 경우 연산자 In을 사용합니다.

5

이러한 토폴로지 도메인을 나타내기 위해 사용하며 미리 채워져 있는 **Kubernetes** 라벨인 `topologyKey`를 지정합니다.

b.

Pod를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

4.3.3. Pod 유사성 방지 규칙 구성

다음 단계에서는 라벨이 있는 Pod 및 유사성 방지 기본 규칙을 사용하여 해당 Pod에 예약하지 않는 Pod를 생성하는 간단한 2-Pod 구성을 보여줍니다.



참고

예약된 Pod에 선호도를 직접 추가할 수 없습니다.

프로세스

1. **Pod 사양에서 특정 라벨을 사용하여 Pod를 생성합니다.**

- a. 다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

- b. **Pod**를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

2. 다른 **Pod**를 생성할 때 다음 매개변수를 구성합니다.

- a. 다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s2-east
#...
spec
  affinity 1
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: security 4
                values:
                  - S1
            operator: In 5
          topologyKey: kubernetes.io/hostname 6
#...
```

1

Pod 유사성 방지를 추가합니다.

2

`requiredDuringSchedulingIgnoredDuringExecution` 매개변수 또는 `preferredDuringSchedulingIgnoredDuringExecution` 매개변수를 구성합니다.

3

기본 규칙의 경우 노드의 가중치 **1-100**을 지정합니다. 가중치가 높은 노드가 우선합니다.

4

충족해야 하는 키와 값을 지정합니다. 새 **Pod**를 다른 **Pod**와 함께 예약하지 않으려면 첫 번째 **Pod**의 라벨과 동일한 **key** 및 **values** 매개변수를 사용합니다.

5

연산자를 지정합니다. 연산자는 **In**, **NotIn**, **Exists** 또는 **DoesNotExist**일 수 있습니다. 예를 들어 노드에 라벨이 있어야 하는 경우 연산자 **In**을 사용합니다.

6

이러한 토폴로지 도메인을 나타내는 데 사용하는 미리 채워진 **Kubernetes** 라벨인 **topologyKey**를 지정합니다.

b.

Pod를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

4.3.4. 샘플 Pod 유사성 및 유사성 방지 규칙

다음 예제에서는 **Pod** 유사성 및 **Pod** 유사성 방지를 보여줍니다.

4.3.4.1. Pod 유사성

다음 예제에서는 일치하는 라벨 및 라벨 선택기가 있는 **Pod**의 **Pod** 유사성을 보여줍니다.

- Pod team4에는 라벨 team:4가 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
  #...
spec:
  containers:
    - name: ocp
      image: docker.io/ocpqe/hello-pod
  #...

```

- Pod team4a에는 podAffinity 아래에 라벨 선택기 team:4가 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: team4a
  #...
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: team
                operator: In
                values:
                  - "4"
            topologyKey: kubernetes.io/hostname
        containers:
          - name: pod-affinity
            image: docker.io/ocpqe/hello-pod
  #...

```

- team4a Pod는 team4 Pod와 동일한 노드에 예약됩니다.

4.3.4.2. Pod 유사성 방지

다음 예제에서는 일치하는 라벨 및 라벨 선택기가 있는 Pod의 Pod 유사성 방지를 보여줍니다.

- Pod pod-s1에는 라벨 security:s1이 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
  #...
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
  #...

```

- Pod pod-s2에는 podAntiAffinity 아래에 라벨 선택기 **security:s1**이 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
  #...
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - s1
        topologyKey: kubernetes.io/hostname
      containers:
      - name: pod-antiaffinity
        image: docker.io/ocpqe/hello-pod
      #...

```

- Pod pod-s2는 pod-s1과 동일한 노드에 예약할 수 없습니다.

4.3.4.3. 일치하는 라벨이 없는 Pod 유사성

다음 예제에서는 일치하는 라벨 및 라벨 선택기가 없는 Pod의 Pod 유사성을 보여줍니다.

- Pod pod-s1에는 라벨 **security:s1**이 있습니다.

```

apiVersion: v1
kind: Pod
metadata:

```



```

name: pod-s1
labels:
  security: s1
#...
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
#...

```

•

Pod pod-s2에는 라벨 선택기 **security:s2**가 있습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
#...
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
#...

```

•

security:s2 라벨이 있는 Pod가 포함된 노드가 없는 경우 Pod pod-s2는 예약되지 않습니다. 해당 라벨이 있는 기타 Pod가 없는 경우 새 Pod는 보류 중인 상태로 유지됩니다.

출력 예

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s2	0/1	Pending	0	32s	<none>	

4.3.5. Pod 유사성 및 유사성 방지를 사용하여 Operator 설치 위치 제어

기본적으로 **Operator**를 설치할 때 **OpenShift Container Platform**은 **Operator Pod**를 작업자 노드 중 하나에 무작위로 설치합니다. 그러나 해당 **Pod**를 특정 노드 또는 노드 세트에 예약하려는 경우가 있을 수 있습니다.

다음 예제에서는 **Operator Pod**를 특정 노드 또는 노드 세트에 예약할 수 있는 상황을 설명합니다.

- **Operator**에 **amd64** 또는 **ARM64**와 같은 특정 플랫폼이 필요한 경우
- **Operator**에 **Linux** 또는 **Windows**와 같은 특정 운영 체제가 필요한 경우
- 동일한 호스트 또는 동일한 랙에 있는 호스트에서 함께 작동하는 **Operator**가 필요한 경우
- 네트워크 또는 하드웨어 문제로 인해 다운타임을 방지하기 위해 **Operator**가 인프라 전체에 분산되도록 하려면

Operator의 **Subscription** 오브젝트에 **Pod** 유사성 또는 유사성 방지를 추가하여 **Operator Pod** 설치 위치를 제어할 수 있습니다.

다음 예제에서는 **Pod** 유사성 방지를 사용하여 특정 라벨이 있는 **Pod**가 있는 모든 노드에서 **Custom Metrics Autoscaler Operator**를 설치하지 않도록 하는 방법을 보여줍니다.

Operator Pod를 하나 이상의 특정 노드에 배치하는 **Pod** 유사성 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
```

```

operator: In
values:
- test
topologyKey: kubernetes.io/hostname
#...

```

1

app=test 라벨이 있는 Pod가 있는 노드에 Operator의 Pod를 배치하는 Pod 유사성입니다.

Operator Pod가 하나 이상의 특정 노드에서 방지되는 Pod 유사성 방지 예제

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    podAntiAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: cpu
          operator: In
          values:
          - high
      topologyKey: kubernetes.io/hostname
#...

```

1

cpu=high 라벨이 있는 Pod가 있는 노드에서 Operator의 Pod를 예약하지 못하도록 하는 Pod 유사성 방지입니다.

프로세스

Operator Pod 배치를 제어하려면 다음 단계를 완료합니다.

1. **Operator**를 정상적으로 설치합니다.
2. 필요한 경우 노드에 유사성에 올바르게 응답하도록 레이블이 지정되어 있는지 확인합니다.
3. **Operator Subscription** 오브젝트를 편집하여 유사성을 추가합니다.

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    podAntiAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
      podAffinityTerm:
        labelSelector:
          matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - ip-10-0-185-229.ec2.internal
          topologyKey: topology.kubernetes.io/zone
  #...

```

1

podAffinity 또는 **podAntiAffinity** 를 추가합니다.

검증

•

Pod가 특정 노드에 배포되었는지 확인하려면 다음 명령을 실행합니다.

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

출력 예

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED	NODE	READINESS	GATES	
custom-metrics-autoscaler-operator-5dcc45d656-bhshg	1/1	Running	0	50s	
10.131.0.20	ip-10-0-185-229.ec2.internal	<none>	<none>		

4.4. 노드 유사성 규칙을 사용하여 노드에 대한 POD 배치 제어

유사성은 예약할 노드를 제어하는 Pod의 속성입니다.

OpenShift Container Platform 노드 유사성은 스케줄러에서 Pod를 배치할 수 있는 위치를 결정하는 데 사용하는 규칙 집합입니다. 규칙은 노드의 사용자 정의 라벨과 Pod에 지정된 라벨 선택기를 사용하여 정의합니다.

4.4.1. 노드 유사성 이해

노드 유사성을 사용하면 Pod에서 Pod를 배치할 수 있는 노드 그룹에 대한 유사성을 지정할 수 있습니다. 노드는 배치를 제어할 수 없습니다.

예를 들어 특정 CPU 또는 특정 가용성 영역이 있는 노드에서만 실행하도록 Pod를 구성할 수 있습니다.

노드 유사성 규칙에는 필수 및 기본 두 가지의 유형이 있습니다.

노드에 Pod를 예약하려면 먼저 필수 규칙을 충족해야 합니다. 기본 규칙은 규칙이 충족되는 경우 스케줄러가 규칙을 적용하려고 하지만 반드시 적용되는 것은 아닙니다.



참고

노드의 라벨이 런타임에 변경되어 Pod에 대한 노드 유사성 규칙이 더 이상 충족되지 않으면 Pod가 해당 노드에서 계속 실행됩니다.

노드 유사성은 Pod 사양 파일을 통해 구성합니다. 필수 규칙, 기본 규칙 또는 둘 다 지정할 수 있습니다.

다. 둘 다 지정하는 경우 노드는 먼저 필수 규칙을 충족한 다음 기본 규칙을 충족하려고 합니다.

다음 예제는 키가 `e2e-az-NorthSouth`이고 값이 `e2e-az-North` 또는 `e2e-az-South`인 라벨이 있는 노드에 Pod를 배치해야 하는 규칙이 있는 Pod 사양입니다.

노드 유사성 필수 규칙이 있는 Pod 구성 파일의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: ①
      requiredDuringSchedulingIgnoredDuringExecution: ②
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-NorthSouth ③
                operator: In ④
                values:
                  - e2e-az-North ⑤
                  - e2e-az-South ⑥
    containers:
      - name: with-node-affinity
        image: docker.io/ocpqe/hello-pod
#...
```

①

노드 유사성을 구성하는 스탠자입니다.

②

필요한 규칙을 정의합니다.

③ ⑤ ⑥

규칙을 적용하려면 일치해야 하는 키/값(라벨)입니다.

④

연산자는 노드의 라벨과 Pod 사양에 있는 `matchExpression` 매개변수의 값 집합 간의 관계를 나타냅니다. 이 값은 `In`, `NotIn`, `Exists`, `DoesNotExist`, `Lt` 또는 `Gt`일 수 있습니다.

다음 예제는 Pod에 대해 키가 **e2e-az-EastWest**이고 값이 **e2e-az-East** 또는 **e2e-az-West**인 라벨이 있는 노드를 선호하는 기본 규칙이 있는 노드 사양입니다.

노드 유사성 기본 규칙이 있는 Pod 구성 파일의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
        - weight: 1 3
          preference:
            matchExpressions:
              - key: e2e-az-EastWest 4
                operator: In 5
                  values:
                    - e2e-az-East 6
                    - e2e-az-West 7
            containers:
              - name: with-node-affinity
                image: docker.io/ocpqe/hello-pod
#...
```

1

노드 유사성을 구성하는 스탠자입니다.

2

기본 규칙을 정의합니다.

3

기본 규칙의 가중치를 지정합니다. 가중치가 높은 노드가 우선합니다.

4

6

7

5

연산자는 노드의 라벨과 Pod 사양에 있는 `matchExpression` 매개변수의 값 집합 간의 관계를 나타냅니다. 이 값은 `In`, `NotIn`, `Exists`, `DoesNotExist`, `Lt` 또는 `Gt`일 수 있습니다.

평시적인 노드 유사성 방지 개념은 없지만 `NotIn` 또는 `DoesNotExist` 연산자를 사용하여 해당 동작을 복제합니다.

참고

노드 유사성 및 노드 선택기를 동일한 Pod 구성으로 사용하는 경우 다음 사항에 유의하십시오.

- `nodeSelector`와 `nodeAffinity`를 둘 다 구성하는 경우 Pod를 후보 노드에 예약하기 위해서는 두 상태를 모두 충족해야 합니다.
- `nodeAffinity` 유형과 연결된 `nodeSelectorTerms`를 여러 개 지정하는 경우 `nodeSelectorTerms` 중 하나를 충족하면 Pod를 노드에 예약할 수 있습니다.
- `nodeSelectorTerms`와 연결된 `matchExpressions`를 여러 개 지정하는 경우 모든 `matchExpressions`를 충족할 때만 Pod를 노드에 예약할 수 있습니다.

4.4.2. 필수 노드 유사성 규칙 구성

노드에 Pod를 예약하려면 먼저 필수 규칙을 충족해야 합니다.

프로세스

다음 단계에서는 하나의 노드 및 스케줄러에서 해당 노드에 배치해야 하는 하나의 Pod를 생성하는 간단한 구성을 보여줍니다.

1. `oc label node` 명령을 사용하여 노드에 라벨을 추가합니다.

```
$ oc label node node1 e2e-az-name=e2e-az1
```


작은 정보

다음 **YAML**을 적용하여 레이블을 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    e2e-az-name: e2e-az1
#...
```

2.

Pod 사양에서 특정 라벨을 사용하여 **Pod**를 생성합니다.

a.

다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.



참고

예약된 **Pod**에 선호도를 직접 추가할 수 없습니다.

출력 예

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
spec:
  affinity: ①
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution: ②
    nodeSelectorTerms:
      - matchExpressions:
          - key: e2e-az-name ③
            values:
              - e2e-az1
              - e2e-az2
            operator: In ④
#...
```

1

Pod 유사성을 추가합니다.

2

requiredDuringSchedulingIgnoredDuringExecution 매개변수를 구성합니다.

3

충족해야 하는 키와 값을 지정합니다. 편집한 노드에 새 **Pod**를 예약하려면 노드의 라벨과 동일한 **key** 및 **values** 매개변수를 사용합니다.

4

연산자를 지정합니다. 연산자는 **In**, **NotIn**, **Exists** 또는 **DoesNotExist**일 수 있습니다. 예를 들어 노드에 라벨이 있어야 하는 경우 연산자 **In**을 사용합니다.

b.

Pod를 생성합니다.

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

4.4.3. 기본 노드 유사성 규칙 구성

기본 규칙은 규칙이 충족되는 경우 스케줄러가 규칙을 적용하려고 하지만 반드시 적용되는 것은 아닙니다.

프로세스

다음 단계에서는 하나의 노드 및 스케줄러에서 해당 노드에 배치하려고 하는 하나의 **Pod**를 생성하는 간단한 구성을 보여줍니다.

1.

oc label node 명령을 사용하여 노드에 라벨을 추가합니다.

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2.

특정 라벨을 사용하여 **Pod**를 생성합니다.

a.

다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.



참고

예약된 Pod에 선호도를 직접 추가할 수 없습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: s1
spec:
  affinity: ①
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: ②
        - weight: ③
          preference:
            matchExpressions:
              - key: e2e-az-name ④
                values:
                  - e2e-az3
                operator: In ⑤
#...

```

①

Pod 유사성을 추가합니다.

②

`preferredDuringSchedulingIgnoredDuringExecution` 매개변수를 구성합니다.

③

노드의 가중치를 숫자 1~100으로 지정합니다. 가중치가 높은 노드가 우선합니다.

④

충족해야 하는 키와 값을 지정합니다. 편집한 노드에 새 Pod를 예약하려면 노드의 라벨과 동일한 `key` 및 `values` 매개변수를 사용합니다.

⑤

연산자를 지정합니다. 연산자는 `In`, `NotIn`, `Exists` 또는 `DoesNotExist`일 수 있습니다. 예를 들어 노드에 라벨이 있어야 하는 경우 연산자 `In`을 사용합니다.

b.

Pod를 생성합니다.

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

4.4.4. 노드 유사성 규칙 샘플

다음 예제에서는 노드 유사성을 보여줍니다.

4.4.4.1. 일치하는 라벨이 있는 노드 유사성

다음 예제에서는 일치하는 라벨이 있는 노드 및 Pod의 노드 유사성을 보여줍니다.

- **Node1** 노드에는 라벨 **zone:us**가 있습니다.

```
$ oc label node node1 zone=us
```

작은 정보

다음 **YAML**을 적용하여 레이블을 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: us
#...
```

- **pod-s1** Pod에는 필수 노드 유사성 규칙에 따라 **zone** 및 **us** 키/값 쌍이 있습니다.

```
$ cat pod-s1.yaml
```

출력 예

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
```

```

name: hello-pod
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: "zone"
              operator: In
              values:
                - us
#...

```

- *pod-s1 Pod를 Node1에 예약할 수 있습니다.*

```
$ oc get pod -o wide
```

출력 예

```

NAME     READY   STATUS    RESTARTS   AGE   IP      NODE
pod-s1   1/1     Running   0           4m   IP1     node1

```

4.4.4.2. 일치하는 라벨이 없는 노드 유사성

다음 예제에서는 일치하는 라벨이 없는 노드 및 Pod의 노드 유사성을 보여줍니다.

- *Node1 노드에는 라벨 zone:emea가 있습니다.*

```
$ oc label node node1 zone=emea
```

작은 정보

다음 **YAML**을 적용하여 레이블을 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: emea
#...
```

•

pod-s1 Pod에는 필수 노드 유사성 규칙에 따라 **zone** 및 **us** 키/값 쌍이 있습니다.

```
$ cat pod-s1.yaml
```

출력 예

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
                operator: In
                values:
                  - us
#...
```

•

pod-s1 Pod는 **Node1**에 예약할 수 없습니다.

```
$ oc describe pod pod-s1
```

출력 예

...

Events:

FirstSeen	LastSeen	Count	From	SubObjectPath	Type	Reason
1m	33s	8	default-scheduler	Warning	FailedScheduling	No nodes are available that match all of the following predicates:: MatchNodeSelector (1).

4.4.5. 노드 유사성을 사용하여 Operator 설치 위치 제어

기본적으로 **Operator**를 설치할 때 **OpenShift Container Platform**은 **Operator Pod**를 작업자 노드 중 하나에 무작위로 설치합니다. 그러나 해당 **Pod**를 특정 노드 또는 노드 세트에 예약하려는 경우가 있을 수 있습니다.

다음 예제에서는 **Operator Pod**를 특정 노드 또는 노드 세트에 예약할 수 있는 상황을 설명합니다.

- **Operator**에 **amd64** 또는 **ARM64**와 같은 특정 플랫폼이 필요한 경우
- **Operator**에 **Linux** 또는 **Windows**와 같은 특정 운영 체제가 필요한 경우
- 동일한 호스트 또는 동일한 랙에 있는 호스트에서 함께 작동하는 **Operator**가 필요한 경우
- 네트워크 또는 하드웨어 문제로 인해 다운타임을 방지하기 위해 **Operator**가 인프라 전체에 분산되도록 하려면

Operator의 **Subscription** 오브젝트에 노드 유사성 제약 조건을 추가하여 **Operator Pod** 설치 위치를 제어할 수 있습니다.

다음 예제에서는 노드 유사성을 사용하여 **Custom Metrics Autoscaler Operator** 인스턴스를 클러스터의 특정 노드에 설치하는 방법을 보여줍니다.

특정 노드에 **Operator Pod**를 배치하는 노드 유사성 예

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    nodeAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - ip-10-0-163-94.us-west-2.compute.internal
#...

```

1

ip-10-0-163-94.us-west-2.compute.internal 노드에 **Operator Pod**를 예약해야 하는 노드 유사성입니다.

특정 플랫폼이 있는 노드에 **Operator Pod**를 배치하는 노드 유사성 예

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    nodeAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:

```



```

- matchExpressions:
- key: kubernetes.io/arch
  operator: In
  values:
- arm64
- key: kubernetes.io/os
  operator: In
  values:
- linux

```

#...

1

kubernetes.io/arch=arm64 및 **kubernetes.io/os=linux** 라벨을 사용하여 노드에 Operator Pod를 예약해야 하는 노드 유사성입니다.

프로세스

Operator Pod 배치를 제어하려면 다음 단계를 완료합니다.

1. **Operator**를 정상적으로 설치합니다.
2. 필요한 경우 노드에 유사성에 올바르게 응답하도록 레이블이 지정되어 있는지 확인합니다.
3. **Operator Subscription** 오브젝트를 편집하여 유사성을 추가합니다.

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity: 1
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/hostname

```

```
operator: In
values:
- ip-10-0-185-229.ec2.internal
#...
```

1

nodeAffinity 를 추가합니다.

검증

- Pod가 특정 노드에 배포되었는지 확인하려면 다음 명령을 실행합니다.

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

출력 예

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED NODE	READINESS GATES			
custom-metrics-autoscaler-operator-5dcc45d656-bhshg	1/1	Running	0	50s	
10.131.0.20	ip-10-0-185-229.ec2.internal	<none>	<none>		

4.4.6. 추가 리소스

- [노드에서 라벨을 업데이트하는 방법 이해](#)

4.5. 과다 할당된 노드에 POD 배치

과다 할당 상태에서는 컨테이너 컴퓨팅 리소스 요청과 제한의 합이 시스템에서 사용 가능한 리소스를 초과합니다. 용량에 맞게 보장된 성능을 절충할 수 있는 개발 환경에서는 과다 할당이 바람직할 수 있습니다.

관리자는 요청 및 제한을 통해 노드의 리소스 과다 할당을 허용하고 관리할 수 있습니다. 스케줄러는 요청을 사용하여 컨테이너를 예약하고 최소 서비스 보장 기능을 제공합니다. 제한은 노드에서 사용할 수 있는 컴퓨팅 리소스의 양을 제한합니다.

4.5.1. 과다 할당 이해

관리자는 요청 및 제한을 통해 노드의 리소스 과다 할당을 허용하고 관리할 수 있습니다. 스케줄러는 요청을 사용하여 컨테이너를 예약하고 최소 서비스 보장 기능을 제공합니다. 제한은 노드에서 사용할 수 있는 컴퓨팅 리소스의 양을 제한합니다.

OpenShift Container Platform 관리자는 개발자 컨테이너에 설정된 요청과 제한 사이의 비율을 덮어쓰도록 마스터를 구성하여 노드에서 과다 할당 수준을 제어하고 컨테이너 밀도를 관리할 수 있습니다. 제한 및 기본값을 지정하는 프로젝트별 **LimitRange** 오브젝트와 함께 컨테이너 제한 및 요청을 조정하여 원하는 수준의 과다 할당을 구현합니다.



참고

컨테이너에 제한이 설정되어 있지 않은 경우 이러한 덮어쓰기가 적용되지 않습니다. 덮어쓰기를 적용하려면 개별 프로젝트별로 또는 프로젝트 템플릿에 기본 제한을 사용하여 **LimitRange** 오브젝트를 생성합니다.

이러한 덮어쓰기 이후에도 프로젝트의 모든 **LimitRange** 오브젝트에서 컨테이너 제한 및 요청의 유효성을 검사해야 합니다. 예를 들어 개발자가 최소 제한에 가까운 제한을 지정하고 요청에서 최소 제한 미만을 덮어쓰도록 하여 **Pod**를 금지할 수 있습니다. 이처럼 잘못된 사용자 경험은 향후 작업에서 해결해야 하지만 현재는 이 기능과 **LimitRange** 오브젝트를 주의해서 구성하십시오.

4.5.2. 노드 과다 할당 이해

오버 커밋된 환경에서는 최상의 시스템 동작을 제공하도록 노드를 올바르게 구성하는 것이 중요합니다.

노드가 시작되면 메모리 관리를 위한 커널 조정 가능한 플래그가 올바르게 설정됩니다. 커널은 실제 메모리가 소진되지 않는 한 메모리 할당에 실패해서는 안 됩니다.

이 동작을 확인하기 위해 **OpenShift Container Platform**은 **vm.overcommit_memory** 매개변수를 1로 설정하여 기본 운영 체제 설정을 재정의하여 커널이 항상 메모리를 오버 커밋하도록 구성합니다.

OpenShift Container Platform은 **vm.panic_on_oom** 매개변수를 0으로 설정하여 메모리 부족시 커널이 패닉 상태가 되지 않도록 구성합니다. 0으로 설정하면 커널에서 **OOM** (메모리 부족) 상태일 때 **oom_killer**를 호출하여 우선 순위에 따라 프로세스를 종료합니다.

노드에서 다음 명령을 실행하여 현재 설정을 볼 수 있습니다.

```
$ sysctl -a |grep commit
```

출력 예

```
#...  
vm.overcommit_memory = 0  
#...
```

```
$ sysctl -a |grep panic
```

출력 예

```
#...  
vm.panic_on_oom = 0  
#...
```



참고

위의 플래그는 이미 노드에 설정되어 있어야 하며 추가 조치가 필요하지 않습니다.

각 노드에 대해 다음 구성을 수행할 수도 있습니다.

- **CPU CFS** 할당량을 사용하여 CPU 제한 비활성화 또는 실행
- 시스템 프로세스의 리소스 예약
- **Quality of Service (QoS)** 계층에서의 메모리 예약

4.6. 노드 테인트를 사용하여 POD 배치 제어

테인트 및 허용 오차를 사용하면 노드에서 예약해야 하는 (또는 예약해서는 안 되는) Pod를 제어할 수 있습니다.

4.6.1. 테인트(Taints) 및 톨러레이션(Tolerations)의 이해

테인트를 사용하면 Pod에 일치하는 허용 오차가 없는 경우 노드에서 Pod 예약을 거부할 수 있습니다.

Node 사양(NodeSpec)을 통해 노드에 테인트를 적용하고 Pod 사양(PodSpec)을 통해 Pod에 허용 오차를 적용합니다. 노드에 테인트를 적용할 때 Pod에서 테인트를 허용할 수 없는 경우 스케줄러에서 해당 노드에 Pod를 배치할 수 없습니다.

노드 사양의 테인트 예

```
apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Pod 사양의 허용 오차 예

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

테인트 및 톨러레이션은 **key, value** 및 **effect**로 구성되어 있습니다.

표 4.1. 테인트 및 톨러레이션 구성 요소

매개변수	설명						
key	key 는 최대 253 자의 문자열입니다. 키는 문자 또는 숫자로 시작해야 하며 문자, 숫자, 하이픈, 점, 밑줄을 포함할 수 있습니다.						
value	value 는 최대 63 자의 문자열입니다. 값은 문자 또는 숫자로 시작해야 하며 문자, 숫자, 하이픈, 점, 밑줄을 포함할 수 있습니다.						
effect	다음 명령 중 하나를 실행합니다. <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">NoSchedule ^[1]</td> <td> <ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약되지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. </td> </tr> <tr> <td>PreferNoSchedule</td> <td> <ul style="list-style-type: none"> 테인트와 일치하지 않는 새 pod는 해당 노드에 예약할 수 있지만 스케줄러는 그렇게 하지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. </td> </tr> <tr> <td>NoExecute</td> <td> <ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약할 수 없습니다. 일치하는 톨러레이션이 없는 노드의 기존 pod는 제거됩니다. </td> </tr> </table>	NoSchedule ^[1]	<ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약되지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. 	PreferNoSchedule	<ul style="list-style-type: none"> 테인트와 일치하지 않는 새 pod는 해당 노드에 예약할 수 있지만 스케줄러는 그렇게 하지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. 	NoExecute	<ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약할 수 없습니다. 일치하는 톨러레이션이 없는 노드의 기존 pod는 제거됩니다.
NoSchedule ^[1]	<ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약되지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. 						
PreferNoSchedule	<ul style="list-style-type: none"> 테인트와 일치하지 않는 새 pod는 해당 노드에 예약할 수 있지만 스케줄러는 그렇게 하지 않습니다. 노드의 기존 pod는 그대로 유지됩니다. 						
NoExecute	<ul style="list-style-type: none"> 테인트에 일치하지 않는 새 pod는 해당 노드에 예약할 수 없습니다. 일치하는 톨러레이션이 없는 노드의 기존 pod는 제거됩니다. 						
operator	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Equal</td> <td>key/value/effect 매개변수가 일치해야 합니다. 이는 기본값입니다.</td> </tr> <tr> <td>Exists</td> <td>key/effect 매개변수가 일치해야 합니다. 일치하는 빈 value 매개변수를 남겨 두어야 합니다.</td> </tr> </table>	Equal	key/value/effect 매개변수가 일치해야 합니다. 이는 기본값입니다.	Exists	key/effect 매개변수가 일치해야 합니다. 일치하는 빈 value 매개변수를 남겨 두어야 합니다.		
Equal	key/value/effect 매개변수가 일치해야 합니다. 이는 기본값입니다.						
Exists	key/effect 매개변수가 일치해야 합니다. 일치하는 빈 value 매개변수를 남겨 두어야 합니다.						

1. 컨트롤 플레인 노드에 **NoSchedule** 테인트를 추가하는 경우 노드에 기본적으로 추가되는 **node-role.kubernetes.io/master=:NoSchedule** 테인트가 있어야 합니다.

예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-
master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
    name: my-node
  #...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  #...

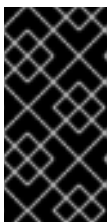
```

톨러레이션은 테인트와 일치합니다.

- **operator** 매개변수가 **Equal**로 설정된 경우:
 - **key** 매개변수는 동일합니다.
 - **value** 매개변수는 동일합니다.
 - **effect** 매개변수는 동일합니다.
- **operator** 매개변수가 **Exists**로 설정된 경우:
 - **key** 매개변수는 동일합니다.
 - **effect** 매개변수는 동일합니다.

다음 테인트는 **OpenShift Container Platform**에 빌드됩니다.

- **node.kubernetes.io/not-ready:** 노드가 준비 상태에 있지 않습니다. 이는 노드 조건 **Ready=False**에 해당합니다.
- **node.kubernetes.io/unreachable:** 노드가 노드 컨트롤러에서 연결할 수 없습니다. 이는 노드 조건 **Ready=Unknown**에 해당합니다.
- **node.kubernetes.io/memory-pressure:** 노드에 메모리 부족 문제가 있습니다. 이는 노드 조건 **MemoryPressure=True**에 해당합니다.
- **node.kubernetes.io/disk-pressure:** 노드에 디스크 부족 문제가 있습니다. 이는 노드 조건 **DiskPressure=True**에 해당합니다.
- **node.kubernetes.io/network-unavailable:** 노드 네트워크를 사용할 수 없습니다.
- **node.kubernetes.io/unschedulable:** 노드를 예약할 수 없습니다.
- **node.cloudprovider.kubernetes.io/uninitialized:** 노드 컨트롤러가 외부 클라우드 공급자로 시작되면 이 테인트 노드에 사용 불가능으로 표시됩니다. **cloud-controller-manager**의 컨트롤러가 이 노드를 초기화하면 **kubelet**이 이 테인트를 제거합니다.
- **node.kubernetes.io/pid-pressure:** 노드에 **pid** 압력이 있습니다. 이는 노드 조건 **PIDPressure=True**에 해당합니다.



중요

OpenShift Container Platform에서는 기본 **pid.available** 제거 **Hard** 를 설정하지 않습니다.

4.6.1.1. **tolerationSeconds**를 사용하여 **pod** 제거를 지연하는 방법

Pod 사양 또는 **MachineSet** 오브젝트에 **tolerationSeconds** 매개변수를 지정하면 **Pod**를 제거하기 전에 노드에 바인딩되는 시간을 지정할 수 있습니다. **NoExecute** 효과가 있는 테인트가 **tolerationSeconds** 매개변수가 있는 테인트를 허용하는 **Pod**인 노드에 추가되면 해당 기간이 만료될 때까지 **Pod**가 제거되지 않습니다.

출력 예


```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...

```

여기에서 이 **Pod**가 실행 중이지만 일치하는 허용 오차가 없으면 **Pod**는 **3,600**초 동안 노드에 바인딩된 후 제거됩니다. 이 시간 이전에 테인트가 제거되면 **pod**가 제거되지 않습니다.

4.6.1.2. 여러 테인트를 사용하는 방법

동일한 노드에 여러 테인트를 배치하고 동일한 **pod**에 여러 톨러레이션을 배치할 수 있습니다. **OpenShift Container Platform**은 다음과 같이 여러 테인트 및 톨러레이션을 처리합니다.

1. **Pod**에 일치하는 톨러레이션이 있는 테인트를 처리합니다.
2. 나머지 일치하지 않는 테인트는 **pod**에서 다음 **effect**를 갖습니다.
 - **effect**가 **NoSchedule**인 일치하지 않는 테인트가 하나 이상 있는 경우 **OpenShift Container Platform**은 해당 노드에 **pod**를 예약할 수 없습니다.
 - **effect**가 **NoSchedule**인 일치하지 않는 테인트가 없지만 **effect**가 **PreferNoSchedule**인 일치하지 않는 테인트가 하나 이상 있는 경우, **OpenShift** 컨테이너 플랫폼은 노드에 **pod**를 예약 시도하지 않습니다.
 - 효과가 **NoExecute**인 일치하지 않는 테인트가 하나 이상 있는 경우 **OpenShift Container Platform**은 **Pod**가 노드에서 이미 실행되고 있으면 노드에서 **Pod**를 제거합니다.

Pod가 노드에서 아직 실행되고 있지 않으면 **Pod**가 노드에 예약되지 않습니다.

- 테인트를 허용하지 **Pod**는 즉시 제거됩니다.
- **Pod** 사양에 **tolerationSeconds**를 지정하지 않은 테인트를 허용하는 **Pod**는 영구적으로 바인딩된 상태를 유지합니다.
- **tolerationSeconds**가 지정된 테인트를 허용하는 **Pod**는 지정된 시간 동안 바인딩된 상태로 유지됩니다.

예를 들면 다음과 같습니다.

- 노드에 다음 테인트를 추가합니다.

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
$ oc adm taint nodes node1 key1=value1:NoExecute
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- **Pod**에는 다음과 같은 톨러레이션이 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoExecute"
#...
```

이 경우 세 번째 테인트와 일치하는 톨러레이션이 없기 때문에 **pod**를 노드에 예약할 수 없습니다. 세 번째 테인트는 **pod**에서 허용되지 않는 세 번째 테인트 중 하나이기 때문에 테인트가 추가될 때 노드에서

이미 실행되고 있는 경우 pod가 계속 실행됩니다.

4.6.1.3. Pod 예약 및 노드 상태 (taint node by condition)

상태별 노드 테인트 기능은 기본적으로 활성화되어 있으며 메모리 부족 및 디스크 부족과 같은 상태를 보고하는 노드를 자동으로 테인트합니다. 노드가 상태를 보고하면 상태가 해제될 때까지 테인트가 추가됩니다. 테인트에는 **NoSchedule effect**가 있습니다. 즉, pod에 일치하는 톨러레이션이 없으면 노드에서 pod를 예약할 수 없습니다.

스케줄러는 pod를 예약하기 전에 노드에서 이러한 테인트를 확인합니다. 테인트가 있는 경우 pod는 다른 노드에 예약됩니다. 스케줄러는 실제 노드 상태가 아닌 테인트를 확인하기 때문에 적절한 pod 톨러레이션을 추가하여 이러한 노드 상태 중 일부를 무시하도록 스케줄러를 구성합니다.

이전 버전과의 호환성을 보장하기 위해 데몬 세트 컨트롤러는 모든 데몬에 다음과 같은 허용 오차를 자동으로 추가합니다.

- `node.kubernetes.io/memory-pressure`
- `node.kubernetes.io/disk-pressure`
- `node.kubernetes.io/unschedulable` (1.10 이상)
- `node.kubernetes.io/network-unavailable` (호스트 네트워크 만)

데몬 세트에 임의의 허용 오차를 추가할 수 있습니다.



참고

컨트롤 플레인에는 QoS 클래스가 있는 Pod에 `node.kubernetes.io/memory-pressure` 허용 오차를 추가합니다. 이는 Kubernetes가 **Guaranteed** 또는 **Burstable QoS** 클래스에서 Pod를 관리하기 때문입니다. 새 **BestEffort Pod**가 영향을 받는 노드에 예약되지 않습니다.

4.6.1.4. 상태 별 pod 제거 (taint-based evictions)

기본적으로 활성화된 **Taint-Based Evictions** 기능은 **not-ready** 및 **unreachable**과 같은 특정 상태에 있는 노드에서 **Pod**를 제거합니다. 노드에 이러한 상태 중 하나가 발생하면 **OpenShift Container Platform**은 자동으로 노드에 테인트를 추가하고 **pod**를 제거하여 다른 노드에서 다시 예약하기 시작합니다.

Taint Based Evictions에는 **NoExecute** 효과가 있으며, 여기서 테인트를 허용하지 않는 **Pod**는 즉시 제거되고 테인트를 허용하는 모든 **Pod**는 **tolerationSeconds** 매개변수를 사용하지 않는 한 제거되지 않습니다.

tolerationSeconds 매개변수를 사용하면 노드 조건이 설정된 노드에 **Pod**가 바인딩되는 시간을 지정할 수 있습니다. **tolerationSeconds** 기간 후에도 이 상태가 계속되면 테인트가 노드에 남아 있고 허용 오차가 일치하는 **Pod**가 제거됩니다. **tolerationSeconds** 기간 전에 상태 조건이 지워지면 허용 오차가 일치하는 **Pod**가 제거되지 않습니다.

값이 없는 **tolerationSeconds** 매개변수를 사용하는 경우 준비되지 않고 연결할 수 없는 노드 상태로 인해 **Pod**가 제거되지 않습니다.

참고

OpenShift Container Platform은 속도가 제한된 방식으로 **pod**를 제거하여 마스터가 노드에서 분할되는 등의 시나리오에서 대규모 **pod** 제거를 방지합니다.

기본적으로 지정된 영역의 노드 **55%** 이상이 비정상인 경우 노드 라이프사이클 컨트롤러는 해당 영역의 상태를 **PartialDisruption** 으로 변경하고 **Pod** 제거 속도가 줄어듭니다. 이 상태의 소규모 클러스터(기본적으로 **50**개 이하)의 경우 이 영역의 노드는 테인트되지 않으며 제거가 중지됩니다.

자세한 내용은 **Kubernetes** 설명서의 **제거에 대한 속도 제한**을 참조하십시오.

Pod 구성에서 허용 오차를 지정하지 않는 경우 **OpenShift Container Platform**은 자동으로 **node.kubernetes.io/not-ready** 및 **node.kubernetes.io/unreachable**의 허용 오차를 **tolerationSeconds=300**으로 추가합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
```

```

- key: node.kubernetes.io/not-ready
  operator: Exists
  effect: NoExecute
  tolerationSeconds: 300 1
- key: node.kubernetes.io/unreachable
  operator: Exists
  effect: NoExecute
  tolerationSeconds: 300
#...

```

1

이러한 톨러레이션은 이러한 노드 상태 문제 중 하나가 감지된 후 기본 **pod** 동작을 5 분 동안 바인딩된 상태로 유지할 수 있도록 합니다.

필요에 따라 이러한 톨러레이션을 구성할 수 있습니다. 예를 들어 애플리케이션에 다수의 로컬 상태가 있는 경우 네트워크 파티션 등에 따라 **pod**를 노드에 더 오래 바인딩하여 파티션을 복구하고 **pod** 제거를 방지할 수 있습니다.

데몬 세트에 의해 생성된 **Pod**는 **tolerationSeconds**가 없는 다음 테인트의 **NoExecute** 허용 오차를 사용하여 생성됩니다.

- `node.kubernetes.io/unreachable`
- `node.kubernetes.io/not-ready`

결과적으로 이러한 노드 상태로 인해 데몬 세트 **Pod**가 제거되지 않습니다.

4.6.1.5. 모든 테인트 허용

key 및 **values** 매개변수 없이 **operator: "Exists"** 허용 오차를 추가하여 모든 테인트를 허용하도록 **Pod**를 구성할 수 있습니다. 이 허용 오차가 있는 **Pod**는 테인트가 있는 노드에서 제거되지 않습니다.

모든 테인트를 허용하는 **Pod** 사양

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod

```

```
#...
spec:
  tolerations:
    - operator: "Exists"
#...
```

4.6.2. 테인트 및 톨러레이션 추가

Pod에 허용 오차를 추가하고 노드에 테인트를 추가하면 노드에 예약하거나 예약하지 않아야 하는 Pod를 노드에서 제어할 수 있습니다. 기존 Pod 및 노드의 경우 먼저 Pod에 허용 오차를 추가한 다음 노드에 테인트를 추가하여 허용 오차를 추가하기 전에 노드에서 Pod가 제거되지 않도록 합니다.

프로세스

1. **tolerations** 스탠자를 포함하도록 Pod 사양을 편집하여 Pod에 허용 오차를 추가합니다.

Equal 연산자가 있는 Pod 구성 파일 샘플

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1" ①
      value: "value1"
      operator: "Equal"
      effect: "NoExecute"
      tolerationSeconds: 3600 ②
#...
```

①

테인트 및 허용 오차 구성 요소 테이블에 설명된 허용 오차 매개변수입니다.

②

tolerationSeconds 매개변수를 지정하여 pod가 제거되기 전까지 노드에 바인딩되는 시간을 설정합니다.

예를 들면 다음과 같습니다.

Exists 연산자가 있는 Pod 구성 파일 샘플

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  #...
spec:
  tolerations:
    - key: "key1"
      operator: "Exists" 1
      effect: "NoExecute"
      tolerationSeconds: 3600
    #...

```

1

Exists 연산자는 **value**를 사용하지 않습니다.

이 예에서는 **key key1**, **value value1**, 테인트 **effect NoExecute**를 갖는 **node1**에 테인트를 배치합니다.

2.

테인트 및 허용 오차 구성 요소 테이블에 설명된 매개변수로 다음 명령을 사용하여 노드에 테인트를 추가합니다.

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

이 명령은 키가 **key1**, 값이 **value1**, 효과가 **NoExecute**인 **node1**에 테인트를 배치합니다.



참고

컨트롤 플레인 노드에 **NoSchedule** 테인트를 추가하는 경우 노드에 기본적으로 추가되는 **node-role.kubernetes.io/master=:NoSchedule** 테인트가 있어야 합니다.

예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-
f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
    name: my-node
  #...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  #...

```

Pod의 허용 오차가 노드의 테인트와 일치합니다. 허용 오차 중 하나가 있는 **Pod**를 **node1**에 예약할 수 있습니다.

4.6.2.1. 컴퓨팅 머신 세트를 사용하여 테인트 및 허용 오차 추가

컴퓨팅 머신 세트를 사용하여 노드에 테인트를 추가할 수 있습니다. **MachineSet** 오브젝트와 연결된 모든 노드는 테인트를 사용하여 업데이트됩니다. 허용 오차는 노드에 직접 추가된 테인트와 동일한 방식으로 컴퓨팅 머신 세트에 의해 추가된 테인트에 응답합니다.

프로세스

1. **tolerations** 스태ن자를 포함하도록 **Pod** 사양을 편집하여 **Pod**에 허용 오차를 추가합니다.

Equal 연산자가 있는 **Pod** 구성 파일의 예


```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1" ❶
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 ❷
#...

```

❶

테인트 및 허용 오차 구성 요소 테이블에 설명된 허용 오차 매개변수입니다.

❷

tolerationSeconds 매개변수는 Pod가 제거될 때까지 노드에 바인딩되는 시간을 지정합니다.

예를 들면 다음과 같습니다.

Exists 연산자가 있는 pod 구성 파일의 예

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...

```

2.

MachineSet 오브젝트에 테인트를 추가합니다.

a.

테인트할 노드의 **MachineSet** YAML을 편집하거나 새 **MachineSet** 오브젝트를 생성할 수 있습니다.

```
$ oc edit machineset <machineset>
```

b.

spec.template.spec 섹션에 테인트를 추가합니다.

컴퓨팅 머신 세트 사양의 테인트에

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: my-machineset
#...
spec:
#...
  template:
#...
    spec:
      taints:
      - effect: NoExecute
        key: key1
        value: value1
#...
```

이 예제에서는 키가 **key1**, 값이 **value1**, 테인트 효과가 **NoExecute**인 테인트를 노드에 배치합니다.

c.

컴퓨팅 머신 세트를 **0**으로 축소합니다.

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

작은 정보

다음 **YAML**을 적용하여 컴퓨팅 머신 세트를 확장할 수 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

머신이 제거될 때까지 기다립니다.

d.

필요에 따라 컴퓨팅 머신 세트를 확장합니다.

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

또는 다음을 수행합니다.

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

머신이 시작될 때까지 기다립니다. 테인트는 **MachineSet** 오브젝트와 연결된 노드에 추가됩니다.

4.6.2.2. 테인트 및 톨러레이션을 사용하여 사용자를 노드에 바인딩

특정 사용자 집합에서 독점적으로 사용하도록 노드 세트를 전용으로 지정하려면 해당 **Pod**에 허용 오차를 추가합니다. 그런 다음 해당 노드에 해당 테인트를 추가합니다. 허용 오차가 있는 **Pod**는 테인트된 노드 또는 클러스터의 다른 노드를 사용할 수 있습니다.

이렇게 테인트된 노드에만 **Pod**를 예약하려면 동일한 노드 세트에도 라벨을 추가하고 해당 라벨이 있는 노드에만 **Pod**를 예약할 수 있도록 **Pod**에 노드 유사성을 추가합니다.

프로세스

사용자가 해당 노드 만 사용할 수 있도록 노드를 구성하려면 다음을 수행합니다.

1.

해당 노드에 해당 테인트를 추가합니다.

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

작은 정보

다음 **YAML**을 적용하여 테인트를 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
#...
```

2.

사용자 정의 승인 컨트롤러를 작성하여 **Pod**에 허용 오차를 추가합니다.

4.6.2.3. 노드 선택기 및 허용 오차를 사용하여 프로젝트 생성

노드 선택기 및 허용 오차를 사용하여 특정 노드에 **Pod** 배치를 제어하는 프로젝트를 생성할 수 있습니다. 그런 다음 프로젝트에서 생성된 후속 리소스는 허용 오차와 일치하는 테인트가 있는 노드에 예약됩니다.

사전 요구 사항

- 컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 노드 선택의 레이블이 하나 이상의 노드에 추가되었습니다.
- 컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 테인트가 하나 이상의 노드에 추가되어 있습니다.

절차

1. **metadata.annotations** 섹션에서 노드 선택기 및 허용 오차를 지정하여 **Project** 리소스 정의를 생성합니다.

project.yaml 파일 예

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: <project_name> ①
  annotations:
    openshift.io/node-selector: '<label>' ②
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
        "<key_name>"}] ③
  ]
```

①

프로젝트 이름입니다.

②

기본 노드 선택기 레이블입니다.

③

테인트 및 허용 오차 구성 요소 테이블에 설명된 허용 오차 매개변수입니다. 이 예에서는 노드의 기존 pod를 유지할 수 있는 **NoSchedule** 효과와 값을 사용하지 않는 **Exists** 연산자를 사용합니다.

2. **oc apply** 명령을 사용하여 프로젝트를 생성합니다.

```
$ oc apply -f project.yaml
```

<project_name> 네임스페이스에서 생성된 후속 리소스는 이제 지정된 노드에서 예약해야 합니다.

추가 리소스

- [노드에 수동으로 또는 컴퓨팅 머신 세트를 사용하여 테인트 및 허용 오차 추가](#)
- [프로젝트 수준 노드 선택기 생성](#)
- [Operator 워크로드의 Pod 배치](#)

4.6.2.4. 테인트 및 톨러레이션을 사용하여 특수 하드웨어로 노드 제어

소규모 노드 하위 집합에 특수 하드웨어가 있는 클러스터에서는 테인트 및 허용 오차를 사용하여 특수 하드웨어가 필요하지 않은 Pod를 해당 노드에서 분리하여 특수 하드웨어가 필요한 Pod를 위해 노드를 남겨 둘 수 있습니다. 또한 특정 노드를 사용하기 위해 특수 하드웨어가 필요한 Pod를 요청할 수도 있습니다.

이 작업은 특수 하드웨어가 필요한 Pod에 허용 오차를 추가하고 특수 하드웨어가 있는 노드를 테인트 하여 수행할 수 있습니다.

프로세스

특수 하드웨어가 있는 노드를 특정 Pod용으로 예약하려면 다음을 수행합니다.

1. 특수 하드웨어가 필요한 Pod에 허용 오차를 추가합니다.

예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  #...
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
    #...

```

2. 다음 명령 중 하나를 사용하여 특수 하드웨어가 있는 노드에 테인트를 설정합니다.

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

또는 다음을 수행합니다.

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

작은 정보

다음 **YAML**을 적용하여 테인트를 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
#...
```

4.6.3. 테인트 및 톨러레이션 제거

필요에 따라 노드에서 테인트를 제거하고 Pod에서 톨러레이션을 제거할 수 있습니다. 허용 오차를 추가하려면 먼저 Pod에 허용 오차를 추가한 다음 노드에서 Pod가 제거되지 않도록 노드에 테인트를 추가해야 합니다.

프로세스

테인트 및 톨러레이션을 제거하려면 다음을 수행합니다.

1. 노드에서 테인트를 제거하려면 다음을 수행합니다.

```
$ oc adm taint nodes <node-name> <key>-
```

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

출력 예

```
node/ip-10-0-132-248.ec2.internal untainted
```

2.

Pod에서 허용 오차를 제거하려면 **Pod** 사양을 편집하여 허용 오차를 제거합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key2"
      operator: "Exists"
      effect: "NoExecute"
      tolerationSeconds: 3600
#...
```

4.7. 노드 선택기를 사용하여 특정 노드에 **POD** 배치

노드 선택기는 노드의 사용자 정의 라벨 및 **Pod**에 지정된 선택기를 사용하여 정의한 키/값 쌍으로 구성된 맵을 지정합니다.

노드에서 **Pod**를 실행하려면 노드의 라벨과 동일한 키/값 노드 선택기가 **Pod**에 있어야 합니다.

4.7.1. 노드 선택기 정보

Pod의 노드 선택기와 노드의 라벨을 사용하여 **Pod**가 예약되는 위치를 제어할 수 있습니다. 노드 선택기를 사용하면 **OpenShift Container Platform**에서 일치하는 라벨이 포함된 노드에 **Pod**를 예약합니다.

노드 선택기를 사용하여 특정 노드에 특정 **Pod**를 배치하고, 클러스터 수준 노드 선택기를 사용하여 클러스터의 특정 노드에 새 **Pod**를 배치하고, 프로젝트 노드 선택기를 사용하여 특정 노드의 프로젝트에 새 **Pod**를 배치할 수 있습니다.

예를 들어 클러스터 관리자는 애플리케이션 개발자가 생성하는 모든 **Pod**에 노드 선택기를 포함하여 지리적으로 가장 가까운 노드에만 **Pod**를 배포할 수 있는 인프라를 생성할 수 있습니다. 이 예제에서 클러스터는 두 지역에 분배된 데이터센터 5개로 구성됩니다. 미국에서는 노드의 라벨을 **us-east**, **us-central**

또는 **us-west**로 지정합니다. 아시아 태평양 지역(**APAC**)에서는 노드의 라벨을 **apac-east** 또는 **apac-west**로 지정합니다. 개발자는 생성한 **Pod**에 노드 선택기를 추가하여 해당 노드에 **Pod**가 예약되도록 할 수 있습니다.

Pod 오브젝트에 노드 선택기가 포함되어 있지만 일치하는 라벨이 있는 노드가 없는 경우 **Pod**를 예약하지 않습니다.

중요

동일한 **Pod** 구성의 노드 선택기 및 노드 유사성을 사용 중인 경우 다음 규칙에서 노드에 대한 **Pod** 배치를 제어합니다.

- **nodeSelector**와 **nodeAffinity**를 둘 다 구성하는 경우 **Pod**를 후보 노드에 예약하기 위해서는 두 상태를 모두 충족해야 합니다.
- **nodeAffinity** 유형과 연결된 **nodeSelectorTerms**를 여러 개 지정하는 경우 **nodeSelectorTerms** 중 하나를 충족하면 **Pod**를 노드에 예약할 수 있습니다.
- **nodeSelectorTerms**와 연결된 **matchExpressions**를 여러 개 지정하는 경우 모든 **matchExpressions**를 충족할 때만 **Pod**를 노드에 예약할 수 있습니다.

특정 Pod 및 노드의 노드 선택기

노드 선택기 및 라벨을 사용하여 특정 **Pod**가 예약된 노드를 제어할 수 있습니다.

노드 선택기와 라벨을 사용하려면 먼저 **Pod**의 일정이 조정되지 않도록 노드에 라벨을 지정한 다음 노드 선택기를 **Pod**에 추가합니다.

참고

예약된 기존 **Pod**에 노드 선택기를 직접 추가할 수 없습니다. 배포 구성과 같이 **Pod**를 제어하는 오브젝트에 라벨을 지정해야 합니다.

예를 들어 다음 **Node** 오브젝트에는 **region: east** 라벨이 있습니다.

라벨이 있는 **Node** 오브젝트 샘플

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
labels:
  kubernetes.io/os: linux
  failure-domain.beta.kubernetes.io/zone: us-east-1a
  node.openshift.io/os_version: '4.5'
  node-role.kubernetes.io/worker: ""
  failure-domain.beta.kubernetes.io/region: us-east-1
  node.openshift.io/os_id: rhcos
  beta.kubernetes.io/instance-type: m4.large
  kubernetes.io/hostname: ip-10-0-131-14
  beta.kubernetes.io/arch: amd64
  region: east 1
  type: user-node
#...
```

1

Pod 노드 선택기와 일치해야 하는 라벨입니다.

Pod에는 `type: user-node,region: east` 노드 선택기가 있습니다.

노드 선택기가 있는 Pod 오브젝트 샘플

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector: 1
    region: east
    type: user-node
#...
```

1

노드 라벨과 일치해야 하는 노드 선택기입니다. 노드에는 각 노드 선택기에 대한 레이블이 있어야 합니다.

예제 Pod 사양을 사용하여 Pod를 생성하면 예제 노드에 예약할 수 있습니다.

기본 클러스터 수준 노드 선택기

기본 클러스터 수준 노드 선택기를 사용하면 해당 클러스터에서 Pod를 생성할 때 **OpenShift Container Platform**에서 기본 노드 선택기를 Pod에 추가하고 일치하는 라벨을 사용하여 노드에서 Pod를 예약합니다.

예를 들어 다음 **Scheduler** 오브젝트에는 기본 클러스터 수준 **region=east** 및 **type=user-node** 노드 선택기가 있습니다.

스케줄러 Operator 사용자 정의 리소스의 예

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
  #...
spec:
  defaultNodeSelector: type=user-node,region=east
  #...
```

해당 클러스터의 노드에는 **type=user-node,region=east** 라벨이 있습니다.

Node 오브젝트의 예

```
apiVersion: v1
kind: Node
metadata:
```

```

name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
labels:
  region: east
  type: user-node
#...

```

노드 선택기가 있는 Pod 오브젝트의 예

```

apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector:
    region: east
#...

```

예제 클러스터에서 예제 Pod 사양을 사용하여 Pod를 생성하면 Pod가 클러스터 수준 노드 선택기와 함께 생성되어 라벨이 지정된 노드에 예약됩니다.

Pod가 라벨이 지정된 노드에 있는 Pod 목록의 예

NAME	READY	STATUS	RESTARTS	AGE	IP	NOMINATED NODE	STATUS	RESTARTS	AGE	IP	NODE
pod-s1	1/1	Running	0	20s	10.131.2.6	ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4	<none>	<none>	<none>	<none>	<none>



참고

Pod를 생성하는 프로젝트에 프로젝트 노드 선택기가 있는 경우 해당 선택기가 클러스터 수준 노드 선택기보다 우선합니다. Pod에 프로젝트 노드 선택기가 없으면 Pod가 생성되거나 예약되지 않습니다.

프로젝트 노드 선택기

프로젝트 노드 선택기를 사용하면 이 프로젝트에서 Pod를 생성할 때 OpenShift Container Platform에서 Pod에 노드 선택기를 추가하고 라벨이 일치하는 노드에 Pod를 예약합니다. 클러스터 수준 기본 노드 선택기가 있는 경우 프로젝트 노드 선택기가 우선합니다.

예를 들어 다음 프로젝트에는 `region=east` 노드 선택기가 있습니다.

Namespace 오브젝트의 예

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
  annotations:
    openshift.io/node-selector: "region=east"
#...
```

다음 노드에는 `type=user-node,region=east` 라벨이 있습니다.

Node 오브젝트의 예

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
labels:
  region: east
  type: user-node
#...
```

이 예제 프로젝트에서 예제 Pod 사양을 사용하여 Pod를 생성하면 Pod가 프로젝트 노드 선택기와 함께 생성되어 라벨이 지정된 노드에 예약됩니다.

Pod 오브젝트의 예

```

apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
  #...
spec:
  nodeSelector:
    region: east
    type: user-node
  #...

```

Pod가 라벨이 지정된 노드에 있는 **Pod** 목록의 예

```

NAME READY STATUS RESTARTS AGE IP NODE
NOMINATED NODE READINESS GATES
pod-s1 1/1 Running 0 20s 10.131.2.6 ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none> <none>

```

Pod에 다른 노드 선택기가 포함된 경우 프로젝트의 **Pod**가 생성되거나 예약되지 않습니다. 예를 들어 다음 **Pod**를 예제 프로젝트에 배포하면 **Pod**가 생성되지 않습니다.

노드 선택기가 유효하지 않은 **Pod** 오브젝트의 예

```

apiVersion: v1
kind: Pod
metadata:
  name: west-region
  #...
spec:
  nodeSelector:
    region: west
  #...

```

4.7.2. 노드 선택기를 사용하여 Pod 배치 제어

Pod의 노드 선택기와 노드의 라벨을 사용하여 **Pod**가 예약되는 위치를 제어할 수 있습니다. 노드 선택기를 사용하면 **OpenShift Container Platform**에서 일치하는 라벨이 포함된 노드에 **Pod**를 예약합니다.

노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 머신 세트에 레이블을 추가하면 노드 또는 머신이 중단되는 경우 새 노드에 라벨이 지정됩니다. 노드 또는 머신이 중단된 경우 노드 또는 머신 구성에 추가된 라벨이 유지되지 않습니다.

기존 **Pod**에 노드 선택기를 추가하려면 **ReplicaSet** 오브젝트, **DaemonSet** 오브젝트, **StatefulSet** 오브젝트, **Deployment** 오브젝트 또는 **DeploymentConfig** 오브젝트와 같이 해당 **Pod**의 제어 오브젝트에 노드 선택기를 추가합니다. 이 제어 오브젝트 아래의 기존 **Pod**는 라벨이 일치하는 노드에서 재생성됩니다. 새 **Pod**를 생성하는 경우 **Pod** 사양에 노드 선택기를 직접 추가할 수 있습니다. **Pod**에 제어 오브젝트가 없는 경우 **Pod**를 삭제하고 **Pod** 사양을 편집하고 **Pod**를 다시 생성해야 합니다.



참고

예약된 기존 **Pod**에 노드 선택기를 직접 추가할 수 없습니다.

사전 요구 사항

기존 **Pod**에 노드 선택기를 추가하려면 해당 **Pod**의 제어 오브젝트를 결정하십시오. 예를 들어 **router-default-66d5cf9464-m2g75** **Pod**는 **router-default-66d5cf9464** 복제본 세트에서 제어합니다.

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

출력 예

```
kind: Pod
apiVersion: v1
metadata:
#...
Name:      router-default-66d5cf9464-7pwkc
Namespace: openshift-ingress
# ...
Controlled By:  ReplicaSet/router-default-66d5cf9464
# ...
```

웹 콘솔에서 Pod YAML의 `ownerReferences` 아래에 제어 오브젝트가 나열됩니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: router-default-66d5cf9464
    uid: d81dd094-da26-11e9-a48a-128e7edf0312
    controller: true
    blockOwnerDeletion: true
# ...
```

절차

1.

컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 노드에 라벨을 추가합니다.

-

노드를 생성할 때 컴퓨팅 머신 세트에서 관리하는 노드에 라벨을 추가하려면 **MachineSet** 오브젝트를 사용합니다.

a.

다음 명령을 실행하여 **MachineSet** 오브젝트에 라벨을 추가합니다.

```
$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"
<key>":"<value>","<key>":"<value>"}}]' -n openshift-machine-api
```

예를 들면 다음과 같습니다.

```
$ oc patch MachineSet abc612-msrtw-worker-us-east-1c --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":
{"type":"user-node","region":"east"}}]' -n openshift-machine-api
```


작은 정보

또는 다음 **YAML**을 적용하여 컴퓨팅 머신 세트에 라벨을 추가할 수 있습니다.

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: xf2bd-infra-us-east-2a
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
#...

```

b.

oc edit 명령을 사용하여 라벨이 **MachineSet** 오브젝트에 추가되었는지 확인합니다.

예를 들면 다음과 같습니다.

```

$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api

```

MachineSet 오브젝트의 예

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet

# ...

spec:
# ...
  template:
    metadata:
# ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
# ...

```

- 라벨을 노드에 직접 추가합니다.

- a. 노드의 **Node** 오브젝트를 편집합니다.

```
$ oc label nodes <name> <key>=<value>
```

예를 들어 노드에 라벨을 지정하려면 다음을 수행합니다.

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

작은 정보

다음 **YAML**을 적용하여 노드에 라벨을 추가할 수 있습니다.

```
kind: Node  
apiVersion: v1  
metadata:  
  name: hello-node-6fbccf8d9  
  labels:  
    type: "user-node"  
    region: "east"  
  #...
```

- b. 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc get nodes -l type=user-node,region=east
```

출력 예

```
NAME                                STATUS ROLES AGE VERSION  
ip-10-0-142-25.ec2.internal Ready worker 17m v1.25.0
```

2.

Pod에 일치하는 노드 선택기를 추가합니다.

•

기존 및 향후 **Pod**에 노드 선택기를 추가하려면 **Pod**의 제어 오브젝트에 노드 선택기를 추가합니다.

라벨이 있는 **ReplicaSet** 오브젝트의 예

```

kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
template:
  metadata:
    creationTimestamp: null
  labels:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
    pod-template-hash: 66d5cf9464
  spec:
    nodeSelector:
      kubernetes.io/os: linux
      node-role.kubernetes.io/worker: "
      type: user-node ①
#...
```

①

노드 선택기를 추가합니다.

•

특정 새 **Pod**에 노드 선택기를 추가하려면 선택기를 **Pod** 오브젝트에 직접 추가합니다.

노드 선택기가 있는 **Pod** 오브젝트의 예

```

apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
```

```
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...
```



참고

예약된 기존 Pod에 노드 선택기를 직접 추가할 수 없습니다.

4.7.3. 기본 클러스터 수준 노드 선택기 생성

Pod의 기본 클러스터 수준 노드 선택기와 노드의 라벨을 함께 사용하면 클러스터에 생성되는 모든 Pod를 특정 노드로 제한할 수 있습니다.

클러스터 수준 노드 선택기를 사용하여 해당 클러스터에서 Pod를 생성하면 OpenShift Container Platform에서 기본 노드 선택기를 Pod에 추가하고 라벨이 일치하는 노드에 Pod를 예약합니다.

Scheduler Operator CR(사용자 정의 리소스)을 편집하여 클러스터 수준 노드 선택기를 구성합니다. 노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 머신 세트에 레이블을 추가하면 노드 또는 머신이 중단되는 경우 새 노드에 라벨이 지정됩니다. 노드 또는 머신이 중단된 경우 노드 또는 머신 구성에 추가된 라벨이 유지되지 않습니다.



참고

Pod에 키/값 쌍을 추가할 수 있습니다. 그러나 기본 키에는 다른 값을 추가할 수 없습니다.

프로세스

기본 클러스터 수준 노드 선택기를 추가하려면 다음을 수행합니다.

1. Scheduler Operator CR을 편집하여 기본 클러스터 수준 노드 선택기를 추가합니다.

```
$ oc edit scheduler cluster
```

노드 선택기를 사용하는 Scheduler Operator CR의 예

```

apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: type=user-node,region=east ❶
  mastersSchedulable: false

```

❶

적절한 <key>:<value> 쌍을 사용하여 노드 선택기를 추가합니다.

변경 후 `openshift-kube-apiserver` 프로젝트의 pod가 재배포될 때까지 기다립니다. 이 작업은 몇 분 정도 걸릴 수 있습니다. 기본 클러스터 수준 노드 선택기는 Pod가 재배포된 후 적용됩니다.

2.

컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 노드에 라벨을 추가합니다.

•

노드를 생성할 때 컴퓨팅 머신 세트에서 관리하는 노드에 라벨을 추가하려면 컴퓨팅 머신 세트를 사용합니다.

a.

다음 명령을 실행하여 `MachineSet` 오브젝트에 라벨을 추가합니다.

```

$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"
<key>":"<value>","key>":"<value>"}}]' -n openshift-machine-api ❶

```

❶

각 라벨에 <key>/<value> 쌍을 추가합니다.

예를 들면 다음과 같습니다.

-

```
$ oc patch MachineSet ci-ln-l8nry52-f76d1-h17m7-worker-c --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":
{"type":"user-node","region":"east"}}] -n openshift-machine-api
```

작은 정보

또는 다음 **YAML**을 적용하여 컴퓨팅 머신 세트에 라벨을 추가할 수 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b.

`oc edit` 명령을 사용하여 라벨이 **MachineSet** 오브젝트에 추가되었는지 확인합니
다.

예를 들면 다음과 같습니다.

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-
api
```

MachineSet 오브젝트의 예

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
...
spec:
...
  template:
    metadata:
...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
...

```

- c. 0으로 축소하고 노드를 확장하여 해당 컴퓨팅 머신 세트와 관련된 노드를 재배포합니다.

예를 들면 다음과 같습니다.

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

- d. 노드가 준비되고 사용 가능한 경우 `oc get` 명령을 사용하여 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>
```

예를 들면 다음과 같습니다.

```
$ oc get nodes -l type=user-node
```

출력 예

```
NAME                                STATUS ROLES  AGE  VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp Ready  worker  61s  v1.25.0
```

- 라벨을 노드에 직접 추가합니다.

- a. 노드의 `Node` 오브젝트를 편집합니다.

```
$ oc label nodes <name> <key>=<value>
```

예를 들어 노드에 라벨을 지정하려면 다음을 수행합니다.

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node
region=east
```

작은 정보

다음 **YAML**을 적용하여 노드에 라벨을 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

b.

oc get 명령을 사용하여 노드에 라벨이 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>,<key>=<value>
```

예를 들면 다음과 같습니다.

```
$ oc get nodes -l type=user-node,region=east
```

출력 예

```
NAME                                STATUS ROLES AGE VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 Ready worker 17m v1.25.0
```

4.7.4. 프로젝트 수준 노드 선택기 생성

프로젝트의 노드 선택기와 함께 노드의 라벨을 사용하여 해당 프로젝트에서 생성되는 모든 **Pod**를 라벨이 지정된 노드로 제한할 수 있습니다.

이 프로젝트에서 Pod를 생성하면 OpenShift Container Platform에서 프로젝트의 Pod에 노드 선택기를 추가하고 프로젝트의 라벨이 일치하는 노드에 Pod를 예약합니다. 클러스터 수준 기본 노드 선택기가 있는 경우 프로젝트 노드 선택기가 우선합니다.

`openshift.io/node-selector` 매개변수를 추가하도록 Namespace 오브젝트를 편집하여 프로젝트에 노드 선택기를 추가합니다. 노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 머신 세트에 레이블을 추가하면 노드 또는 머신이 중단되는 경우 새 노드에 라벨이 지정됩니다. 노드 또는 머신이 중단된 경우 노드 또는 머신 구성에 추가된 라벨이 유지되지 않습니다.

Pod 오브젝트에 노드 선택기가 포함되어 있지만 일치하는 노드 선택기가 있는 프로젝트가 없는 경우 Pod를 예약하지 않습니다. 해당 사양에서 Pod를 생성하면 다음 메시지와 유사한 오류가 발생합니다.

오류 메시지의 예

Error from server (Forbidden): error when creating "pod.yaml": pods "pod-4" is forbidden: pod node label selector conflicts with its project node label selector



참고

Pod에 키/값 쌍을 추가할 수 있습니다. 그러나 프로젝트 키에는 다른 값을 추가할 수 없습니다.

프로세스

기본 프로젝트 노드 선택기를 추가하려면 다음을 수행합니다.

1.

네임스페이스를 생성하거나 기존 네임스페이스를 편집하여 `openshift.io/node-selector` 매개변수를 추가합니다.

```
$ oc edit namespace <name>
```

출력 예

```
apiVersion: v1
```

```

kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "type=user-node,region=east" 1
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: kube:admin
    openshift.io/sa.scc.mcs: s0:c30,c5
    openshift.io/sa.scc.supplemental-groups: 1000880000/10000
    openshift.io/sa.scc.uid-range: 1000880000/10000
  creationTimestamp: "2021-05-10T12:35:04Z"
  labels:
    kubernetes.io/metadata.name: demo
  name: demo
  resourceVersion: "145537"
  uid: 3f8786e3-1fcb-42e3-a0e3-e2ac54d15001
spec:
  finalizers:
    - kubernetes

```

1

적절한 `<key>:<value>` 쌍이 포함된 `openshift.io/node-selector`를 추가합니다.

2.

컴퓨팅 머신 세트를 사용하거나 노드를 직접 편집하여 노드에 라벨을 추가합니다.

•

노드를 생성할 때 컴퓨팅 머신 세트에서 관리하는 노드에 라벨을 추가하려면 `MachineSet` 오브젝트를 사용합니다.

a.

다음 명령을 실행하여 `MachineSet` 오브젝트에 라벨을 추가합니다.

```

$ oc patch MachineSet <name> --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"
<key>":"<value>","<key>":"<value>"}]}] -n openshift-machine-api

```

예를 들면 다음과 같습니다.

```

$ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":
{"type":"user-node","region":"east"}}] -n openshift-machine-api

```

작은 정보

또는 다음 **YAML**을 적용하여 컴퓨팅 머신 세트에 라벨을 추가할 수 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

- b. *oc edit* 명령을 사용하여 라벨이 **MachineSet** 오브젝트에 추가되었는지 확인합니다.

예를 들면 다음과 같습니다.

```
$ oc edit MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

출력 예

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  ...
spec:
  ...
  template:
    metadata:
      ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
```

c.

해당 컴퓨팅 머신 세트와 연결된 노드를 재배포합니다.

예를 들면 다음과 같습니다.

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

d.

노드가 준비되고 사용 가능한 경우 `oc get` 명령을 사용하여 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>
```

예를 들면 다음과 같습니다.

```
$ oc get nodes -l type=user-node,region=east
```

출력 예

```
NAME                                STATUS ROLES AGE VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp Ready worker 61s v1.25.0
```

•

라벨을 노드에 직접 추가합니다.

a.

라벨을 추가할 `Node` 오브젝트를 편집합니다.

```
$ oc label <resource> <name> <key>=<value>
```

예를 들어 노드에 라벨을 지정하려면 다음을 수행합니다.

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-c-tgq49 type=user-node region=east
```

작은 정보

다음 **YAML**을 적용하여 노드에 라벨을 추가할 수 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
labels:
  type: "user-node"
  region: "east"
```

b.

oc get 명령을 사용하여 **Node** 오브젝트에 라벨이 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>
```

예를 들면 다음과 같습니다.

```
$ oc get nodes -l type=user-node,region=east
```

출력 예

```
NAME                                STATUS ROLES AGE VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 Ready worker 17m v1.25.0
```

추가 리소스

-

[노드 선택기 및 허용 오차를 사용하여 프로젝트 생성](#)

4.8. POD 토폴로지 분배 제약 조건을 사용하여 POD 배치 제어

Pod 토폴로지 분배 제약 조건을 사용하여 노드, 영역, 지역 또는 기타 사용자 정의 토폴로지 도메인에서 포드 배치를 세밀하게 제어할 수 있습니다. 장애 도메인에 **Pod**를 분산하면 고가용성과 리소스 활용도를 높일 수 있습니다.

4.8.1. 사용 사례 예

- 관리자는 내 워크로드가 자동으로 2~15개의 Pod로 확장되기를 바랍니다. 단일 장애 지점을 방지하기 위해 두 개의 Pod만 있는 경우 동일한 노드에 배치되지 않도록 합니다.
- 관리자는 대기 시간과 네트워크 비용을 줄이기 위해 포드를 여러 인프라 영역에 균등하게 배포하고 싶습니다. 문제가 발생할 경우 클러스터가 자동으로 복구될 수 있도록 하고 싶습니다.

4.8.2. 중요한 고려 사항

- OpenShift Container Platform 클러스터의 Pod는 배포, 상태 저장 세트 또는 데몬 세트와 같은 워크로드 컨트롤러에서 관리합니다. 이러한 컨트롤러는 클러스터의 노드에 분산 및 스케일링하는 방법을 포함하여 Pod 그룹에 대해 원하는 상태를 정의합니다. 혼동을 방지하려면 그룹의 모든 Pod에 동일한 Pod 토폴로지 분배 제약 조건을 설정해야 합니다. 배포와 같은 워크로드 컨트롤러를 사용하는 경우 Pod 템플릿에서 일반적으로 이를 처리합니다.
- 다른 Pod 토폴로지 분배 제약 조건을 혼합하면 OpenShift Container Platform 동작이 혼동되고 문제 해결이 더 어려워질 수 있습니다. 토폴로지 도메인의 모든 노드에 일관되게 레이블이 지정되도록 하여 이 문제를 방지할 수 있습니다. OpenShift Container Platform은 `kubernetes.io/hostname` 과 같은 잘 알려진 레이블을 자동으로 채웁니다. 이렇게 하면 노드에 수동으로 레이블을 지정할 필요가 없습니다. 이러한 레이블은 필수 토폴로지 정보를 제공하여 클러스터 전체에서 일관된 노드 레이블을 지정할 수 있습니다.
- 제약 조건으로 인해 분배 시 동일한 네임스페이스 내의 Pod만 일치하고 함께 그룹화됩니다.
- 여러 Pod 토폴로지 분배 제약 조건을 지정할 수 있지만 서로 충돌하지 않도록 해야 합니다. Pod를 배치하려면 모든 Pod 토폴로지 분배 제약 조건을 충족해야 합니다.

4.8.3. Skew 및 maxSkew 이해

skew는 영역 또는 노드와 같이 다양한 토폴로지 도메인의 지정된 라벨 선택기와 일치하는 Pod 수의 차이점을 나타냅니다.

skew는 해당 도메인의 Pod 수와 예약된 Pod 양이 가장 적은 Pod 수 간의 절대 차이를 사용하여 각 도메인에 대해 계산됩니다. **maxSkew** 값을 설정하면 균형 있는 pod 배포를 유지 관리할 수 있는 스케줄러가 안내됩니다.

4.8.3.1. skew 계산 예

세 개의 영역(A, B, C)이 있으며 이러한 영역에 Pod를 균등하게 배포하려고 합니다. 영역 A에 Pod가 5개, 영역 B에 3개의 Pod가 있고 영역 C에 포드가 2개 있는 경우 각 영역에 현재 Pod 수에서 예약된 Pod 수가 가장 낮은 도메인의 Pod 수를 차감할 수 있습니다. 즉, A 영역의 스큐는 3이고, 영역 B의 스큐는 1이고, C 영역의 스큐는 0입니다.

4.8.3.2. maxSkew 매개변수

maxSkew 매개변수는 두 토폴로지 도메인 간의 Pod 수에서 허용되는 최대 차이점 또는 skew를 정의합니다. maxSkew 가 1 로 설정된 경우 토폴로지 도메인의 Pod 수는 다른 도메인과 1 이상 달라서는 안 됩니다. skew가 maxSkew 를 초과하면 스케줄러는 스큐를 줄여 제약 조건을 준수하는 방식으로 새 Pod 를 배치하려고 합니다.

이전 예제 skew 계산을 사용하면 skew 값이 기본 maxSkew 값 1 을 초과합니다. 스케줄러는 새 Pod 를 영역 B 및 영역 C에 배치하여 스큐를 줄이고 더 균형 있는 배포를 수행하여 토폴로지 도메인이 1의 스큐를 초과하지 않도록 합니다.

4.8.4. Pod 토폴로지 분배 제약 조건에 대한 구성 예

함께 그룹화할 Pod, 분산되는 토폴로지 도메인 및 허용 가능한 불일치를 지정할 수 있습니다.

다음 예제에서는 Pod 토폴로지 분배 제약 조건 구성을 보여줍니다.

영역에 따라 지정된 라벨과 일치하는 Pod를 배포하는 예

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1 ①
    topologyKey: topology.kubernetes.io/zone ②
    whenUnsatisfiable: DoNotSchedule ③
    labelSelector: ④
      matchLabels:
        region: us-east ⑤

```

```

matchLabelKeys:
  - my-pod-label 6
containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]

```

1

두 토폴로지 도메인 간 최대 Pod 수 차이입니다. 기본값은 1이고 값 0은 지정할 수 없습니다.

2

노드 라벨의 키입니다. 이 키 및 동일한 값이 있는 노드는 동일한 토폴로지에 있는 것으로 간주됩니다.

3

분배 제약 조건을 충족하지 않는 경우 Pod를 처리하는 방법입니다. 기본값은 스케줄러에 Pod를 예약하지 않도록 지시하는 **DoNotSchedule**입니다. Pod를 계속 예약하기 위해 **ScheduleAnyway**로 설정하지만 스케줄러는 클러스터의 불균형이 더 심해지지 않도록 불일치에 따라 우선순위를 부여합니다.

4

이 라벨 선택기와 일치하는 Pod는 제약 조건을 충족하기 위해 분배될 때 계산되고 그룹으로 인식됩니다. 라벨 선택기를 지정해야 합니다. 그렇지 않으면 일치하는 Pod가 없습니다.

5

이 Pod 사양도 나중에 올바르게 계산되도록 하려면 해당 라벨을 이 라벨 선택기와 일치하도록 설정해야 합니다.

6

분배를 계산할 Pod를 선택하는 Pod 레이블 키 목록입니다.

단일 Pod 토폴로지 분배 제약 조건의 예


```

kind: Pod
apiVersion: v1
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod

```

이전 예제에서는 하나의 Pod 토폴로지 분배 제약 조건을 사용하여 Pod 사양을 정의합니다. **region** 레이블이 지정된 Pod와 일치합니다. **us-east** 는 영역에 배포되고, 1의 스큐를 지정하고, 이러한 요구 사항을 충족하지 않는 경우 Pod를 예약하지 않습니다.

여러 Pod 토폴로지 분배 제약 조건을 보여주는 예

```

kind: Pod
apiVersion: v1
metadata:
  name: my-pod-2
  labels:
    region: us-east
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: node
    whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
  - maxSkew: 1
    topologyKey: rack
    whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod

```

이전 예제에서는 두 개의 Pod 토폴로지 분배 제약 조건이 있는 Pod 사양을 정의합니다. 둘 다 **region: us-east** 레이블이 지정된 Pod와 일치하고, 불일치를 1로 지정하고, 이러한 요구 사항을 충족하지 않는 경우 Pod를 예약하지 않습니다.

첫 번째 제약 조건에서는 사용자 정의 라벨 **node**를 기반으로 Pod를 배포하고, 두 번째는 제약 조건에서는 사용자 정의 라벨 **rack**을 기반으로 Pod를 배포합니다. Pod를 예약하려면 두 제약 조건을 모두 충족해야 합니다.

4.8.5. 추가 리소스

- [노드에서 라벨을 업데이트하는 방법 이해](#)

4.9. DESCHEDULER를 사용하여 POD 제거

스케줄러는 새 Pod를 호스팅하는 데 가장 적합한 노드를 결정하는 데 사용되지만 **Descheduler**는 더 적합한 노드에 Pod를 다시 예약할 수 있도록 실행 중인 Pod를 제거하는 데 사용할 수 있습니다.

4.9.1. Descheduler 정보

Descheduler를 사용하면 특정 전략에 따라 Pod를 제거하여 Pod를 더 적절한 노드에 다시 예약할 수 있습니다.

다음과 같은 상황에서 실행 중인 Pod의 일정을 조정하면 이점을 누릴 수 있습니다.

- 노드가 충분히 사용되지 않았거나 너무 많이 사용되었습니다.
- 오염 또는 라벨과 같은 Pod 및 노드 선호도 요구 사항이 변경되었으며, 원래 일정 결정이 더 이상 특정 노드에 적합하지 않습니다.
- 노드 장애로 Pod를 이동해야 합니다.

- 새 노드가 클러스터에 추가되었습니다.
- Pod가 너무 많이 재시작되었습니다.



중요

Descheduler는 제거된 Pod의 교체를 예약하지 않습니다. 제거된 Pod에 대한 이러한 작업은 스케줄러에서 자동으로 수행합니다.

Descheduler가 노드에서 Pod를 제거하도록 결정하는 경우 다음과 같은 일반 메커니즘을 사용합니다.

- **openshift-*** 및 **kube-system** 네임스페이스의 Pod는 제거되지 않습니다.
- **priorityClassName**이 **system-cluster-critical** 또는 **system-node-critical**로 설정된 중요 Pod는 제거되지 않습니다.
- 복제 컨트롤러, 복제본 세트, 배포, **StatefulSet** 또는 작업에 포함되지 않은 정적, 미러링 또는 독립형 Pod는 다시 생성되지 않으므로 제거되지 않습니다.
- 데몬 세트와 연결된 Pod는 제거되지 않습니다.
- 로컬 스토리지가 있는 Pod는 제거되지 않습니다.
- 최상의 Pod가 버스트 가능 Pod 및 보장된 Pod보다 먼저 제거됩니다.
- **descheduler.alpha.kubernetes.io/evict** 주석이 있는 모든 Pod 유형을 제거할 수 있습니다. 이 주석은 제거를 방지하는 검사를 덮어쓰는 데 사용되며 사용자는 제거할 Pod를 선택할 수 있습니다. 사용자는 Pod를 다시 생성하는 방법과 다시 생성되는지의 여부를 알아야 합니다.
- **PDB(Pod 중단 예산)**가 적용되는 Pod는 일정 조정에서 해당 PDB를 위반하는 경우 제거되지 않습니다. Pod는 PDB를 처리하는 제거 하위 리소스를 사용하여 제거합니다.

4.9.2. Descheduler 프로필

다음 **Descheduler** 프로필을 사용할 수 있습니다.

AffinityAndTaints

이 프로필은 **Pod** 간 유사성 방지, 노드 유사성, 노드 테인트를 위반하는 **Pod**를 제거합니다.

다음과 같은 전략을 활성화합니다.

- **RemovePodsViolatingInterPodAntiAffinity:** **Pod** 간 유사성 방지를 위반하는 **Pod**를 제거합니다.
- **RemovePodsViolatingNodeAffinity:** 노드 유사성을 위반하는 **Pod**를 제거합니다.
- **RemovePodsViolatingNodeTaints:** 노드에서 **NoSchedule** 테인트를 위반하는 **Pod**를 제거합니다.

노드 유사성 유형이 **requiredDuringSchedulingIgnoredDuringExecution**인 **Pod**가 제거됩니다.

TopologyAndDuplicates

이 프로필은 노드 간에 유사한 **Pod** 또는 동일한 토폴로지 도메인의 **Pod**를 균등하게 분배하기 위해 **Pod**를 제거합니다.

다음과 같은 전략을 활성화합니다.

- **RemovePodsViolatingTopologySpreadConstraint:** **DoNotSchedule** 제약 조건을 위반하는 경우 균형이 맞지 않는 토폴로지 도메인을 찾아 더 큰 도메인에서 **Pod**를 제거합니다.
- **RemoveDuplicates:** 동일한 노드에서 실행 중인 복제본 세트, 복제 컨트롤러, 배포 또는 작업과 연결된 **Pod**가 하나뿐인지 확인합니다. **Pod**가 두 개 이상인 경우 클러스터에서 **Pod**를 더 잘 배포하기 위해 이러한 중복 **Pod**를 제거합니다.

LifecycleAndUtilization

이 프로파일은 장기 실행 Pod를 제거하고 노드 간 리소스 사용량의 균형을 조정합니다.

다음과 같은 전략을 활성화합니다.

- **RemovePodsHavingTooManyRestarts:** 컨테이너가 너무 여러 번 재시작된 Pod를 제거합니다.

Init Container를 비롯한 모든 컨테이너를 통해 재시작하는 총 Pod는 100개 이상입니다.

- **LowNodeUtilization:** 활용도가 낮은 노드를 찾고, Pod 재생성 시 Pod가 이처럼 활용도가 낮은 노드에 예약되도록 가능하면 과도하게 사용된 노드에서 Pod를 제거합니다.

모든 임계값(CPU, 메모리, Pod 수)에서 사용량이 20% 미만인 경우 노드는 활용도가 낮은 것으로 간주됩니다.

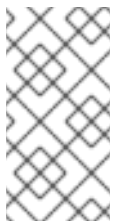
모든 임계값(CPU, 메모리, Pod 수)에서 사용량이 50%를 초과하면 노드는 과도하게 사용되는 것으로 간주됩니다.

- **PodLifeTime:** 너무 오래된 Pod를 제거합니다.

기본적으로 24시간이 지난 Pod는 제거됩니다. Pod 수명 값을 사용자 지정할 수 있습니다.

SoftTopologyAndDuplicates

이 프로파일은 **whenUnsatisfiable: ScheduleAnyway** 와 같은 소프트 토폴로지 제약 조건이 있는 Pod도 제거됨을 제외하고 **TopologyAndDuplicates** 와 동일합니다.



참고

VDDK TopologyAndDuplicates 및 **TopologyAndDuplicates** 둘 다 활성화하지 마십시오. 두 가지를 모두 활성화하면 충돌이 발생합니다.

EvictPodsWithLocalStorage

이 프로필을 사용하면 로컬 스토리지가 있는 Pod를 제거할 수 있습니다.

EvictPodsWithPVC

이 프로필을 사용하면 영구 볼륨 클레임이 있는 Pod를 제거할 수 있습니다. **Kubernetes NFS Subdir External Provisioner** 를 사용하는 경우 프로비저너가 설치된 네임스페이스에 대해 제외된 네임스페이스를 추가해야 합니다.

4.9.3. Descheduler 설치

Descheduler는 기본적으로 사용할 수 없습니다. **Descheduler**를 활성화하려면 **OperatorHub**에서 **Kube Descheduler Operator**를 설치하고 **Descheduler** 프로필을 한 개 이상 활성화해야 합니다.

기본적으로 **Descheduler**는 예측 모드에서 실행되므로 Pod 제거만 시뮬레이션합니다. **Descheduler**가 Pod 제거를 수행할 수 있도록 모드를 자동으로 변경해야 합니다.



중요

클러스터에서 호스팅된 컨트롤 플레인을 활성화한 경우 사용자 정의 우선순위 임계값을 설정하여 호스팅된 컨트롤 플레인 네임스페이스의 Pod가 제거될 가능성을 낮춥니다. 호스팅된 컨트롤 플레인 우선 순위 클래스 클래스의 가장 낮은 우선 순위 값(100000000)이 있으므로 우선순위 임계값 클래스 이름을 **hypershift-control-plane** 으로 설정합니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스합니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Kube Descheduler Operator**에 필요한 네임스페이스를 생성합니다.
 - a. 관리 → 네임스페이스로 이동하여 네임스페이스 생성을 클릭합니다.

- b. 이름 필드에 **openshift-kube-descheduler-operator** 를 입력하고 **Labels** 필드에 **openshift.io/cluster-monitoring=true** 를 입력하여 **Descheduler** 지표를 활성화한 다음 생성을 클릭합니다.
3. **Kube Descheduler Operator**를 설치합니다.
 - a. **Operators** → **OperatorHub**로 이동합니다.
 - b. 필터 박스에 **Kube Descheduler Operator**를 입력합니다.
 - c. **Kube Descheduler Operator**를 선택하고 설치를 클릭합니다.
 - d. **Operator** 설치 페이지에서 클러스터의 특정 네임스페이스를 선택합니다. 드롭다운 메뉴에서 **openshift-kube-descheduler-operator**를 선택합니다.
 - e. 업데이트 채널 및 승인 전략 값을 원하는 값으로 조정합니다.
 - f. 설치를 클릭합니다.
 4. **Descheduler** 인스턴스를 생성합니다.
 - a. **Operator** → 설치된 **Operator** 페이지에서 **Kube Descheduler Operator**를 클릭합니다.
 - b. **Kube Descheduler** 탭을 선택하고 **KubeDescheduler** 생성을 클릭합니다.
 - c. 필요에 따라 설정을 편집합니다.
 - i. 제거를 시뮬레이션하는 대신 **Pod**를 제거하려면 **Mode** 필드를 자동으로 변경합니다.

ii.

프로필 섹션을 확장하여 활성화할 하나 이상의 프로필을 선택합니다. **AffinityAndTaints** 프로필은 기본적으로 활성화되어 있습니다. 프로필 추가를 클릭하여 추가 프로필을 선택합니다.



참고

TopologyAndDuplicates 및 **softTopologyAndDuplicates** 를 모두 활성화하지 마십시오. 두 가지를 모두 활성화하면 충돌이 발생합니다.

iii.

선택 사항: 프로필 사용자 지정 섹션을 확장하여 **Descheduler**에 대한 선택적 구성을 설정합니다.

-

LifecycleAndUtilization 프로필에 사용자 정의 Pod 수명 값을 설정합니다. **podLifetime** 필드를 사용하여 숫자 값과 유효한 단위(s,m 또는 h)를 설정합니다. 기본 Pod 수명은 24시간(24시간)입니다.

-

우선순위가 지정된 우선순위 수준보다 낮은 경우에만 Pod를 제거하도록 사용자 정의 우선순위 임계값을 설정합니다. **thresholdPriority** 필드를 사용하여 숫자 우선순위 임계값을 설정하거나 **thresholdPriorityClassName** 필드를 사용하여 특정 우선순위 클래스 이름을 지정합니다.



참고

Descheduler에 **thresholdPriority** 및 **thresholdPriorityClassName** 을 모두 지정하지 마십시오.

-

Descheduler 작업에서 제외하거나 포함할 특정 네임스페이스를 설정합니다. 네임스페이스 필드를 확장하고 제외되거나 포함된 목록에 네임스페이스를 추가합니다. 제외할 네임스페이스 목록 또는 포함할 네임스페이스 목록만 설정할 수 있습니다. 보호된 네임스페이스(**openshift-***, **kube-system**,**hypershift**)는 기본적으로 제외됩니다.



중요

LowNodeUtilization 전략에서는 네임스페이스 제외를 지원하지 않습니다. **LowNodeUtilization** 전략을 활성화하는 **LifecycleAndUtilization** 프로필이 설정된 경우 보호된 네임스페이스도 제외되지 않습니다. **LowNodeUtilization** 전략이 활성화된 동안 보호된 네임스페이스에서 제거를 방지하려면 우선순위 클래스 이름을 **system-cluster-critical** 또는 **system-node-critical** 로 설정합니다.



실험적: **LowNodeUtilization** 전략의 하위 사용률 및 과다 사용률에 대한 임계값을 설정합니다. **devLowNodeUtilizationThresholds** 필드를 사용하여 다음 값 중 하나를 설정합니다.



낮음: 10%가 낮은 사용량 및 30%의 활용도



중간: 20%가 사용되고 50%의 활용도가 높은(기본값)



높음: 40% 이상 사용되지 않고 70%가 높은 사용



참고

이 설정은 실험적이므로 프로덕션 환경에서 사용해서는 안 됩니다.

iv.

선택 사항: **Descheduling Interval Seconds** 필드를 사용하여 **Descheduler** 실행 사이의 초 수를 변경합니다. 기본값은 3600 초입니다.

d.

생성을 클릭합니다.

나중에 **OpenShift CLI(oc)**를 사용하여 **Descheduler**에 대한 프로필 및 설정을 구성할 수도 있습니다. 웹 콘솔에서 **Descheduler** 인스턴스를 생성할 때 프로필을 조정하지 않은 경우 기본적으로 **AffinityAndTaints** 프로필이 활성화됩니다.

4.9.4. Descheduler 프로필 구성

Descheduler에서 **Pod**를 제거하는 데 사용하는 프로필을 구성할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 권한

프로세스

1. **KubeDescheduler** 오브젝트를 편집합니다.

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. **spec.profiles** 섹션에 하나 이상의 프로필을 지정합니다.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
  mode: Predictive
  profileCustomizations:
    namespaces:
      excluded:
        - my-namespace
  podLifetime: 48h
  thresholdPriorityClassName: my-priority-class-name
  profiles:
    - AffinityAndTaints
    - TopologyAndDuplicates
    - LifecycleAndUtilization
    - EvictPodsWithLocalStorage
    - EvictPodsWithPVC
```

1

선택사항: 기본적으로 **Descheduler**는 **Pod**를 제거하지 않습니다. **Pod**를 제거하려면 **mode** 를 **Automatic** 으로 설정합니다.

2



중요

LowNodeUtilization 전략에서는 네임스페이스 제외를 지원하지 않습니다. **LowNodeUtilization** 전략을 활성화하는 **LifecycleAndUtilization** 프로필이 설정된 경우 보호된 네임스페이스도 제외되지 않습니다. **LowNodeUtilization** 전략이 활성화된 동안 보호된 네임스페이스에서 제거를 방지하려면 우선순위 클래스 이름을 **system-cluster-critical** 또는 **system-node-critical** 로 설정합니다.

3

선택 사항: **LifecycleAndUtilization** 프로필에 사용자 정의 Pod 수명 값을 활성화합니다. 유효한 단위는 s,m 또는 h 입니다. 기본 Pod 수명은 24시간입니다.

4

선택 사항: 우선순위가 지정된 수준보다 낮은 경우에만 제거할 Pod를 고려하도록 우선순위 임계값을 지정합니다. **thresholdPriority** 필드를 사용하여 숫자 우선 순위 임계값(예: 10000)을 설정하거나 **thresholdPriorityClassName** 필드를 사용하여 특정 우선순위 클래스 이름(예: my-priority-class-name)을 지정합니다. 우선순위 클래스 이름을 지정하는 경우 이미 있어야 합니다. 그렇지 않으면 **Descheduler**에서 오류가 발생합니다. **thresholdPriority** 및 **thresholdPriorityClassName** 을 모두 설정하지 마십시오.

5

활성화할 하나 이상의 프로필을 추가합니다. 사용 가능한 프로필: **AffinityAndTaints, TopologyAndDuplicates, LifecycleAndUtilization, softTopologyAndDuplicates, EvictPodsWithLocalStorage, EvictPodsWithPVC.**

6

TopologyAndDuplicates 및 **softTopologyAndDuplicates** 를 모두 활성화하지 마십시오. 두 가지를 모두 활성화하면 충돌이 발생합니다.

여러 프로필을 활성화할 수 있으며 프로필을 지정하는 순서는 중요하지 않습니다.

3.

파일을 저장하여 변경 사항을 적용합니다.

4.9.5. Descheduler 간격 구성

Descheduler 실행 간격을 구성할 수 있습니다. 기본값은 **3600초(1시간)**입니다.

사전 요구 사항

- 클러스터 관리자 권한

프로세스

1. **KubeDescheduler** 오브젝트를 편집합니다.

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. **deschedulingIntervalSeconds** 필드를 원하는 값으로 업데이트합니다.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600 ①
  ...
```

①

Descheduler 실행 간격을 초 단위로 설정합니다. 이 필드 값이 0이면 **Descheduler**가 한 번 실행되고 종료됩니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

4.9.6. Descheduler 설치 제거


Descheduler 인스턴스를 제거하고 **Kube Descheduler Operator**를 설치 제거하여 클러스터에서 **Descheduler**를 제거할 수 있습니다. 이 프로세스는 **KubeDescheduler CRD** 및 **openshift-kube-descheduler-operator** 네임스페이스도 정리합니다.

사전 요구 사항


- 클러스터 관리자 권한이 있어야 합니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스합니다.

프로세스


1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Descheduler** 인스턴스를 삭제합니다.
 - a. **Operator** → 설치된 **Operator** 페이지에서 **Kube Descheduler Operator**를 클릭합니다.
 - b. **Kube Descheduler** 탭을 선택합니다.
 - c. 클러스터 항목 옆에 있는 옵션 메뉴




 를 클릭하고 **KubeDescheduler** 삭제 를 선택합니다.
 - d. 확인 대화 상자에서 삭제를 클릭합니다.
3. **Kube Descheduler Operator**를 설치 제거합니다.
 - a. **Operators** → 설치된 **Operator**로 이동합니다.
 - b. **Kube Descheduler Operator** 옆에 있는 옵션 메뉴



 를 클릭하고 **Operator** 설치 제거를 선택합니다.
 - c. 확인 대화 상자에서 설치 제거를 클릭합니다.

4. **openshift-kube-descheduler-operator** 네임스페이스를 삭제합니다.
 - a. 관리 → 네임스페이스로 이동합니다.
 - b. 필터 박스에 **openshift-kube-descheduler-operator**를 입력합니다.
 - c. **openshift-kube-descheduler-operator** 항목 옆에 있는 옵션 메뉴  를 클릭하고 네임스페이스 삭제 를 선택합니다.
 - d. 확인 대화 상자에서 **openshift-kube-descheduler-operator**를 입력하고 삭제를 클릭합니다.

5. **KubeDescheduler CRD**를 삭제합니다.
 - a. Administration → Custom Resource Definitions로 이동합니다.
 - b. 필터 박스에 **KubeDescheduler**를 입력합니다.
 - c. **KubeDescheduler** 항목 옆에 있는 옵션 메뉴  를 클릭하고 **CustomResourceDefinition** 삭제 를 선택합니다.
 - d. 확인 대화 상자에서 삭제를 클릭합니다.

4.10. 보조 스케줄러

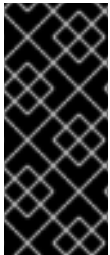
4.10.1. 보조 스케줄러 개요

Secondary Scheduler Operator를 설치하여 기본 스케줄러와 함께 사용자 정의 보조 스케줄러를 실행하여 **Pod**를 예약할 수 있습니다.

4.10.1.1. Secondary Scheduler Operator 정보

Secondary Scheduler Operator for Red Hat OpenShift는 **OpenShift Container Platform**에 사용자 지정 보조 스케줄러를 배포할 수 있는 방법을 제공합니다. 보조 스케줄러는 기본 스케줄러와 함께 실행되어 **Pod**를 예약합니다. **Pod** 구성은 사용할 스케줄러를 지정할 수 있습니다.

사용자 정의 스케줄러에는 `/bin/kube-scheduler` 바이너리가 있어야 하며 **Kubernetes** 스케줄링 프레임워크를 기반으로 합니다.



중요

Secondary Scheduler Operator를 사용하여 **OpenShift Container Platform**에 사용자 정의 보조 스케줄러를 배포할 수 있지만 **Red Hat**은 사용자 정의 보조 스케줄러의 기능을 직접 지원하지 않습니다.

Secondary Scheduler Operator는 보조 스케줄러에 필요한 기본 역할 및 역할 바인딩을 생성합니다. 보조 스케줄러에 대해 **KubeSchedulerConfiguration** 리소스를 구성하여 활성화 또는 비활성화할 스케줄링 플러그인을 지정할 수 있습니다.

4.10.2. Secondary Scheduler Operator for Red Hat OpenShift 릴리스 노트

Secondary Scheduler Operator for Red Hat OpenShift를 사용하면 **OpenShift Container Platform** 클러스터에 사용자 지정 보조 스케줄러를 배포할 수 있습니다.

이 릴리스 노트에서는 **Red Hat OpenShift**용 **Secondary Scheduler Operator**의 개발을 추적합니다.

자세한 내용은 [Secondary Scheduler Operator](#) 정보를 참조하십시오.

4.10.2.1. Secondary Scheduler Operator for Red Hat OpenShift 1.1.3 릴리스 노트

출시 날짜: 2023-10-26

다음 권고는 **Red Hat OpenShift 1.1.3**용 **Secondary Scheduler Operator**에 제공됩니다.

- **RHSA-2023:5933**

4.10.2.1.1. 버그 수정

- 이번 **Secondary Scheduler Operator** 릴리스는 **CVE(Common Vulnerabilities and Exposures)**를 처리합니다.

4.10.2.1.2. 확인된 문제

- 현재 **Secondary Scheduler Operator**를 통해 구성 맵, **CRD** 또는 **RBAC** 정책과 같은 추가 리소스를 배포할 수 없습니다. 사용자 지정 보조 스케줄러에 필요한 역할 및 역할 바인딩 이외의 리소스는 외부에 적용해야 합니다. (**WRKLDS-645**)

4.10.2.2. Secondary Scheduler Operator for Red Hat OpenShift 1.1.2 릴리스 노트

출시 날짜: 2023-8-23

다음 권고는 **Red Hat OpenShift 1.1.2용 Secondary Scheduler Operator**에 제공됩니다.

- **RHSA-2023:4657**

4.10.2.2.1. 버그 수정

- 이번 **Secondary Scheduler Operator** 릴리스는 여러 가지 **CVE(Common Vulnerabilities and Exposures)**를 처리합니다.

4.10.2.2.2. 확인된 문제

- 현재 **Secondary Scheduler Operator**를 통해 구성 맵, **CRD** 또는 **RBAC** 정책과 같은 추가 리소스를 배포할 수 없습니다. 사용자 지정 보조 스케줄러에 필요한 역할 및 역할 바인딩 이외의 리소스는 외부에 적용해야 합니다. (**WRKLDS-645**)

4.10.2.3. Secondary Scheduler Operator for Red Hat OpenShift 1.1.0 릴리스 노트

출시 날짜: 2022-9-1

Secondary Scheduler Operator for Red Hat OpenShift 1.1.0에 다음 권고를 사용할 수 있습니다.

- [RHSA-2022:6152](#)

4.10.2.3.1. 새로운 기능 및 개선 사항

- **Secondary Scheduler Operator** 보안 컨텍스트 구성이 **Pod 보안 승인 시행** 을 준수하도록 업데이트되었습니다.

4.10.2.3.2. 확인된 문제

- 현재 **Secondary Scheduler Operator**를 통해 구성 맵, **CRD** 또는 **RBAC** 정책과 같은 추가 리소스를 배포할 수 없습니다. 사용자 지정 보조 스케줄러에 필요한 역할 및 역할 바인딩 이외의 리소스는 외부에 적용해야 합니다. ([BZ#2071684](#))

4.10.3. 보조 스케줄러를 사용하여 Pod 예약

Secondary Scheduler Operator를 설치하고 보조 스케줄러를 배포하고 **Pod** 정의에서 보조 스케줄러를 설정하여 **OpenShift Container Platform**에서 사용자 지정 보조 스케줄러를 실행할 수 있습니다.

4.10.3.1. Secondary Scheduler Operator 설치

웹 콘솔을 사용하여 **Red Hat OpenShift**용 **Secondary Scheduler Operator**를 설치할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 클러스터에 액세스할 수 있습니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스할 수 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Secondary Scheduler Operator for Red Hat OpenShift**에 필요한 네임스페이스를 생성합니다.

- a. 관리 → 네임스페이스로 이동하여 네임스페이스 생성을 클릭합니다.
 - b. 이름 필드에 **openshift-secondary-scheduler-operator** 를 입력하고 생성을 클릭합니다.
3. **Secondary Scheduler Operator for Red Hat OpenShift 설치.**
- a. **Operators** → **OperatorHub**로 이동합니다.
 - b. 필터 박스에 **Secondary Scheduler Operator for Red Hat OpenShift** 를 입력합니다.
 - c. **Secondary Scheduler Operator for Red Hat OpenShift** 를 선택하고 설치를 클릭합니다.
 - d. **Operator** 설치 페이지에서 다음을 수행합니다.
 - i. **Update** 채널은 Red Hat OpenShift용 **Secondary Scheduler Operator**의 안정적인 최신 릴리스를 설치하는 **stable** 로 설정됩니다.
 - ii. 클러스터의 특정 네임스페이스를 선택하고 드롭다운 메뉴에서 **openshift-secondary-scheduler-operator** 를 선택합니다.
 - iii. 승인 전략을 업데이트합니다.
 - 자동 전략을 사용하면 **Operator** 새 버전이 준비될 때 **OLM(Operator Lifecycle Manager)**이 자동으로 **Operator**를 업데이트할 수 있습니다.
 - 수동 전략을 사용하려면 적절한 자격 증명을 가진 사용자가 **Operator** 업데이트를 승인해야 합니다.

iv.

설치를 클릭합니다.

검증

1. **Operators** → 설치된 **Operator**로 이동합니다.
2. **Secondary Scheduler Operator for Red Hat OpenShift** 가 성공 상태로 나열되어 있는지 확인합니다.

4.10.3.2. 보조 스케줄러 배포

Secondary Scheduler Operator를 설치한 후 보조 스케줄러를 배포할 수 있습니다.

전제 조건

- **cluster-admin** 권한이 있는 클러스터에 액세스할 수 있습니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스할 수 있습니다.
- **Secondary Scheduler Operator for Red Hat OpenShift**가 설치되어 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 보조 스케줄러에 대한 구성을 유지할 구성 맵을 생성합니다.
 - a. 워크로드 → **ConfigMap**으로 이동합니다.
 - b. **ConfigMap** 생성을 클릭합니다.
 - c. **YAML** 편집기에서 필요한 **KubeSchedulerConfiguration** 구성이 포함된 구성 맵 정의를 입력합니다. 예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: "secondary-scheduler-config"
  namespace: "openshift-secondary-scheduler-operator"
data:
  "config.yaml": |
    apiVersion: kubescheduler.config.k8s.io/v1beta3
    kind: KubeSchedulerConfiguration
    leaderElection:
      leaderElect: false
    profiles:
      - schedulerName: secondary-scheduler
    plugins:
      score:
        disabled:
          - name: NodeResourcesBalancedAllocation
          - name: NodeResourcesLeastAllocated
  
```

1

구성 맵의 이름입니다. 이는 **SecondaryScheduler CR**을 생성할 때 **Scheduler Config** 필드에서 사용됩니다.

2

구성 맵은 **openshift-secondary-scheduler-operator** 네임스페이스에서 생성해야 합니다.

3

보조 스케줄러의 **KubeSchedulerConfiguration** 리소스입니다. 자세한 내용은 **Kubernetes API 문서의 KubeSchedulerConfiguration** 을 참조하십시오.

4

보조 스케줄러의 이름입니다. **spec.schedulerName** 필드를 이 값으로 설정하는 **Pod**는 이 보조 스케줄러로 예약됩니다.

5

보조 스케줄러를 활성화하거나 비활성화할 플러그인입니다. 목록의 기본 스케줄링 플러그인은 **Kubernetes 문서의 스케줄링 플러그인** 을 참조하십시오.

d.

생성을 클릭합니다.

3.

SecondaryScheduler CR을 생성합니다.

a.

Operators → 설치된 **Operator**로 이동합니다.

b.

Secondary Scheduler Operator for Red Hat OpenShift 를 선택합니다.

c.

Secondary Scheduler 탭을 선택하고 **Create SecondaryScheduler** 생성을 클릭합니다.

d.

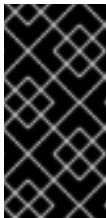
Name 필드는 기본적으로 **cluster** 입니다. 이 이름을 변경하지 마십시오.

e.

Scheduler Config 필드는 기본적으로 **secondary-scheduler-config** 로 설정됩니다. 이 값이 이 절차의 앞부분에서 생성된 구성 맵의 이름과 일치하는지 확인합니다.

f.

Scheduler Image 필드에 사용자 정의 스케줄러의 이미지 이름을 입력합니다.



중요

Red Hat은 사용자 정의 보조 스케줄러의 기능을 직접 지원하지 않습니다.

g.

생성을 클릭합니다.

4.10.3.3. 보조 스케줄러를 사용하여 Pod 예약

보조 스케줄러를 사용하여 **Pod**를 예약하려면 **Pod** 정의에서 **schedulerName** 필드를 설정합니다.

전제 조건

•

cluster-admin 권한이 있는 클러스터에 액세스할 수 있습니다.

- **OpenShift Container Platform** 웹 콘솔에 액세스할 수 있습니다.
- **Secondary Scheduler Operator for Red Hat OpenShift**가 설치되어 있습니다.
- 보조 스케줄러가 구성되어 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 워크로드 → **Pod**로 이동합니다.
3. **Pod** 생성을 클릭합니다.
4. **YAML** 편집기에서 원하는 **Pod** 구성을 입력하고 **schedulerName** 필드를 추가합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80
  schedulerName: secondary-scheduler ①

```

①

schedulerName 필드는 보조 스케줄러를 구성할 때 구성 맵에 정의된 이름과 일치해야 합니다.

5. 생성을 클릭합니다.

검증

1. **OpenShift CLI에 로그인합니다.**
2. **다음 명령을 사용하여 Pod를 설명합니다.**

```
$ oc describe pod nginx -n default
```

출력 예

```
Name:      nginx
Namespace: default
Priority:   0
Node:      ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp/10.0.128.3
...
Events:
  Type     Reason          Age   From              Message
  ----     -
  Normal   Scheduled       12s   secondary-scheduler Successfully assigned
  default/nginx to ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp
...
```

3. **events 표에서 Successfully assigned <namespace>/<pod_name> to <node_name >과 유사한 메시지가 있는 이벤트를 찾습니다.**
4. **"From" 열에서 기본 스케줄러가 아닌 보조 스케줄러에서 이벤트가 생성되었는지 확인합니다.**



참고

openshift-secondary-scheduler-namespace 에서 **secondary-scheduler-*** Pod 로그를 확인하여 보조 스케줄러에서 Pod를 예약했는지 확인할 수도 있습니다.

4.10.4. Secondary Scheduler Operator 설치 제거

Operator를 제거하고 관련 리소스를 제거하여 **OpenShift Container Platform**에서 **Secondary Scheduler Operator for Red Hat OpenShift**를 제거할 수 있습니다.

4.10.4.1. Secondary Scheduler Operator 설치 제거

웹 콘솔을 사용하여 **Secondary Scheduler Operator for Red Hat OpenShift**를 설치 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 클러스터에 액세스할 수 있습니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스할 수 있습니다.
- **Secondary Scheduler Operator for Red Hat OpenShift**가 설치되어 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Secondary Scheduler Operator for Red Hat OpenShift Operator** 설치 제거.
 - a. **Operators** → 설치된 **Operator**로 이동합니다.
 - b. **Secondary Scheduler Operator** 항목 옆에 있는 옵션 메뉴  를 클릭하고 **Operator** 설치 제거를 클릭합니다.
 - c. 확인 대화 상자에서 설치 제거를 클릭합니다.

4.10.4.2. Secondary Scheduler Operator 리소스 제거

필요한 경우 **Secondary Scheduler Operator for Red Hat OpenShift**를 설치 제거한 후 클러스터에서 관련 리소스를 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 클러스터에 액세스할 수 있습니다.
- **OpenShift Container Platform** 웹 콘솔에 액세스할 수 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Secondary Scheduler Operator**가 설치한 **CRD**를 제거합니다.
 - a. **Administration** → **CustomResourceDefinitions** 로 이동합니다.
 - b. **Name** 필드에 **SecondaryScheduler** 를 입력하여 **CRD**를 필터링합니다.
 - c. **SecondaryScheduler CRD** 옆에 있는 옵션 메뉴
 - ⋮
 - 를 클릭하고 사용자 정의 리소스 정의 삭제 를 선택합니다.
3. **openshift-secondary-scheduler-operator** 네임스페이스를 제거합니다.
 - a. **관리** → **네임스페이스**로 이동합니다.
 - b. **openshift-secondary-scheduler-operator** 옆에 있는 옵션 메뉴
 - ⋮
 - 를 클릭하고 네임스페이스 삭제 를 선택합니다.
 - c. 확인 대화 상자에서 필드에 **openshift-secondary-scheduler-operator** 를 입력하고 삭제 를 클릭합니다.

5장. 작업 및 DAEMONSET 사용

5.1. 데몬 세트를 사용하여 노드에서 자동으로 백그라운드 작업 실행

관리자는 데몬 세트를 생성 및 사용하여 **OpenShift Container Platform** 클러스터의 특정 노드 또는 모든 노드에서 **Pod** 복제본을 실행할 수 있습니다.

데몬 세트를 사용하면 모든(또는 일부) 노드에서 하나의 **Pod** 복사본을 실행할 수 있습니다. 노드가 클러스터에 추가되면 **Pod**도 해당 클러스터에 추가됩니다. 노드가 클러스터에서 제거되면 해당 **Pod**도 가비지 컬렉션을 통해 제거됩니다. 데몬 세트를 삭제하면 데몬 세트에서 생성한 **Pod**가 정리됩니다.

데몬 세트를 사용하여 공유 스토리지를 생성하거나 클러스터의 모든 노드에서 로깅 **Pod**를 실행하거나 모든 노드에 모니터링 에이전트를 배포할 수 있습니다.

보안상의 이유로 클러스터 관리자와 프로젝트 관리자가 데몬 세트를 생성할 수 있습니다.

데몬 세트에 대한 자세한 내용은 [Kubernetes 설명서](#)를 참조하십시오.



중요

데몬 세트 예약은 프로젝트의 기본 노드 선택기와 호환되지 않습니다. 비활성화하지 못하면 기본 노드 선택기와 병합되어 데몬 세트가 제한됩니다. 이로 인해 병합된 노드 선택기에서 선택하지 않은 노드에 **Pod**가 자주 다시 생성되고 이로 인해 클러스터에 불필요한 부하가 발생합니다.

5.1.1. 기본 스케줄러로 예약

데몬 세트를 사용하면 모든 적격 노드에서 하나의 **Pod** 복사본을 실행할 수 있습니다. 일반적으로 **Pod**가 실행되는 노드는 **Kubernetes** 스케줄러에서 선택합니다. 그러나 데몬 세트 **Pod**는 데몬 세트 컨트롤러에서 생성하고 예약합니다. 이 경우 다음과 같은 문제가 발생할 수 있습니다.

- 일관되지 않는 **Pod** 동작: 예약 대기 중인 정상 **Pod**가 생성되고 보류 중 상태이지만 데몬 세트 **Pod**는 **Pending** 상태가 아닙니다. 이로 인해 사용자가 혼란스러울 수 있습니다.
- **Pod** 선점을 기본 스케줄러에서 처리합니다. 선점 기능을 활성화하면 데몬 세트 컨트롤러에서 **Pod** 우선순위 및 선점을 고려하지 않고 예약을 결정합니다.

OpenShift Container Platform에서 기본적으로 활성화된 **ScheduleDaemonSetPods** 기능을 사용하면 **spec.nodeName** 용어 대신 **NodeAffinity** 용어를 데몬 세트 Pod에 추가하여 데몬 세트 컨트롤러 대신 기본 스케줄러를 사용하여 데몬 세트를 예약할 수 있습니다. 그런 다음 기본 스케줄러는 Pod를 대상 호스트에 바인딩하는 데 사용됩니다. 데몬 세트 Pod의 노드 유사성이 이미 존재하는 경우 교체됩니다. 데몬 세트 컨트롤러는 데몬 세트 Pod를 생성하거나 수정할 때만 이러한 작업을 수행하며 데몬 세트의 **spec.template**는 변경되지 않습니다.

```
kind: Pod
apiVersion: v1
metadata:
  name: hello-node-6fbccf8d9-9tmzr
#...
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchFields:
            - key: metadata.name
              operator: In
              values:
                - target-host-name
#...
```

또한 데몬 세트 Pod에 **node.kubernetes.io/unschedulable:NoSchedule** 허용 오차가 자동으로 추가됩니다. 기본 스케줄러는 데몬 세트 Pod를 예약할 때 예약할 수 없는 노드를 무시합니다.

5.1.2. 데몬 세트 생성

데몬 세트를 생성할 때 **nodeSelector** 필드는 데몬 세트에서 복제본을 배포해야 하는 노드를 나타내는 데 사용됩니다.

사전 요구 사항

- 데몬 세트를 사용하기 전에 네임스페이스 주석 **openshift.io/node-selector**를 빈 문자열로 설정하여 네임스페이스에서 기본 프로젝트 수준 노드 선택기를 비활성화하십시오.

```
$ oc patch namespace myproject -p \
  '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

작은 정보

다음 **YAML**을 적용하여 네임스페이스의 기본 프로젝트 수준 노드 선택기를 비활성화할 수 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/node-selector: "
#...
```

•

새 프로젝트를 생성하는 경우 기본 노드 선택기를 덮어씁니다.

```
$ oc adm new-project <name> --node-selector=""
```

프로세스

데몬 세트를 생성하려면 다음을 수행합니다.

1.

데몬 세트 **yaml** 파일을 정의합니다.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-daemonset
spec:
  selector:
    matchLabels:
      name: hello-daemonset ①
  template:
    metadata:
      labels:
        name: hello-daemonset ②
    spec:
      nodeSelector: ③
        role: worker
      containers:
        - image: openshift/hello-openshift
          imagePullPolicy: Always
          name: registry
          ports:
            - containerPort: 80
              protocol: TCP
          resources: {}
```

```

terminationMessagePath: /dev/termination-log
serviceAccount: default
terminationGracePeriodSeconds: 10

```

#...

1

데몬 세트에 속하는 **Pod**를 결정하는 라벨 선택기입니다.

2

Pod 템플릿의 라벨 선택기입니다. 위의 라벨 선택기와 일치해야 합니다.

3

배포해야 하는 노드 **Pod** 복제본을 결정하는 노드 선택기입니다. 노드에 일치하는 라벨이 있어야 합니다.

2.

데몬 세트 오브젝트를 생성합니다.

```

$ oc create -f daemonset.yaml

```

3.

Pod가 생성되었고 각 노드에 **Pod** 복제본이 있는지 확인하려면 다음을 수행합니다.

a.

daemonset Pod를 찾습니다.

```

$ oc get pods

```

출력 예

```

hello-daemonset-cx6md 1/1 Running 0 2m
hello-daemonset-e3md9 1/1 Running 0 2m

```

b.

Pod를 보고 **Pod**가 노드에 배치되었는지 확인합니다.

```

$ oc describe pod/hello-daemonset-cx6md|grep Node

```

출력 예

Node: openshift-node01.hostname.com/10.14.20.134

\$ oc describe pod/hello-daemonset-e3md9|grep Node

출력 예

Node: openshift-node02.hostname.com/10.14.20.137

중요

- 데몬 세트 Pod 템플릿을 업데이트해도 기존 Pod 복제본은 영향을 받지 않습니다.
- 데몬 세트를 삭제한 다음 다른 템플릿과 동일한 라벨 선택기를 사용하여 새 데몬 세트를 생성하면 기존 Pod 복제본을 일치하는 라벨이 있는 복제본으로 인식하므로 Pod 템플릿에 일치하지 않는 항목이 있더라도 복제본을 업데이트하거나 새 복제본을 생성하지 않습니다.
- 노드 라벨을 변경하면 데몬 세트에서 새 라벨과 일치하는 노드에 Pod를 추가하고 새 라벨과 일치하지 않는 노드에서 Pod를 삭제합니다.

데몬 세트를 업데이트하려면 이전 복제본 또는 노드를 삭제하여 새 Pod 복제본을 강제로 생성합니다.

5.2. 작업을 사용하여 POD에서 작업 실행

작업은 OpenShift Container Platform 클러스터에서 작업을 실행합니다.

작업에서는 작업의 전반적인 진행률을 추적하고 활성 상태에 있거나 성공 또는 실패한 **Pod**에 대한 정보를 사용하여 해당 상태를 업데이트합니다. 작업을 삭제하면 생성된 **Pod** 복제본이 모두 정리됩니다. 작업은 **Kubernetes API**의 일부이며 다른 오브젝트 유형과 같이 **oc** 명령으로 관리할 수 있습니다.

작업 사양 샘플

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 ①
  completions: 1 ②
  activeDeadlineSeconds: 1800 ③
  backoffLimit: 6 ④
  template: ⑤
    metadata:
      name: pi
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: OnFailure ⑥
#...
```

①

Pod 복제본은 병렬로 실행해야 하는 작업입니다.

②

작업이 완료된 것으로 표시하려면 **Pod**가 완료되어야 합니다.

③

작업을 실행할 수 있는 최대 기간입니다.

④

작업 재시도 횟수입니다.

5

컨트롤러에서 생성하는 **Pod**에 사용할 템플릿입니다.

6

Pod의 재시작 정책입니다.

추가 리소스

- **Kubernetes 문서의 [작업](#)**

5.2.1. 작업 및 cron 작업 이해

작업에서는 작업의 전반적인 진행률을 추적하고 활성 상태에 있거나 성공 또는 실패한 **Pod**에 대한 정보를 사용하여 해당 상태를 업데이트합니다. 작업을 삭제하면 작업에서 생성한 **Pod**가 모두 정리됩니다. 작업은 **Kubernetes API**의 일부이며 다른 오브젝트 유형과 같이 **oc** 명령으로 관리할 수 있습니다.

OpenShift Container Platform에는 한 번 실행 오브젝트를 생성할 수 있는 두 가지 리소스 유형이 있습니다.

Job

일반적인 작업은 작업을 생성하고 작업이 완료되는지 확인하는 한 번 실행 오브젝트입니다.

다음은 작업으로 실행하는 데 적합한 세 가지 주요 작업 유형입니다.

- **비병렬 작업:**
 - **Pod**가 실패하지 않는 한 하나의 **Pod**만 시작하는 작업입니다.
 - **Pod**가 성공적으로 종료되면 작업이 완료됩니다.
- **완료 횟수가 고정된 병렬 작업:**
 - 여러 **Pod**를 시작하는 작업입니다.

- 이 작업은 전체 작업을 나타내며 1에서 **completions** 값 사이의 각 값에 대해 하나의 성공적인 **Pod**가 있을 때 완료됩니다.
- 작업 큐가 있는 병렬 작업:
 - 지정된 **Pod**에 여러 병렬 작업자 프로세스가 있는 작업입니다.
 - **OpenShift Container Platform**은 **Pod**를 조정하여 각각의 작업을 결정하거나 외부 대기열 서비스를 사용합니다.
 - 각 **Pod**는 모든 피어 **Pod**가 완료되었는지 및 전체 작업이 수행되었는지를 독립적으로 확인할 수 있습니다.
 - 작업에서 성공적으로 종료된 **Pod**가 있는 경우 새 **Pod**가 생성되지 않습니다.
 - 하나 이상의 **Pod**가 성공으로 종료되고 모든 **Pod**가 종료되면 작업이 성공적으로 완료됩니다.
 - 성공으로 종료된 **Pod**가 있는 경우 다른 **Pod**에서 이 작업에 대해 작업을 수행하거나 출력을 작성해서는 안 됩니다. **Pod**는 모두 종료 프로세스에 있어야 합니다.
- 다양한 유형의 작업을 사용하는 방법에 대한 자세한 내용은 **Kubernetes** 설명서의 [작업 패턴](#)을 참조하십시오.

cron 작업

cron 작업을 사용하여 작업이 여러 번 실행되도록 예약할 수 있습니다.

cron 작업은 작업 실행 방법을 지정할 수 있도록 허용하여 일반 작업을 기반으로 빌드됩니다. **cron** 작업은 **Kubernetes API**의 일부이며 다른 오브젝트 유형과 같이 **oc** 명령으로 관리할 수 있습니다.

cron 작업은 백업 실행 또는 이메일 전송과 같은 주기적이고 반복적인 작업을 생성하는 데 유용합니다. **cron** 작업에서는 활동이 적은 기간에 작업을 예약하려는 경우와 같이 개별 작업을 특정 시간

에 예약할 수도 있습니다. **cron** 작업은 **cronjob** 컨트롤러를 실행하는 컨트롤 플레인 노드에 구성된 시간대에 따라 **Job** 오브젝트를 생성합니다.



주의

cron 작업에서는 **Job** 오브젝트를 대략적으로 일정 실행 시간당 한 번 생성하지만, 작업을 생성하지 못하거나 두 개의 작업이 생성되는 상황이 있습니다. 따라서 작업이 **idempotent**여야 하고 기록 제한을 구성해야 합니다.

5.2.1.1. 작업 생성 방법 이해

두 리소스 유형 모두 다음 주요 부분으로 구성되는 작업 구성이 필요합니다.

- **Pod 템플릿:** OpenShift Container Platform에서 생성하는 Pod를 설명합니다.
- **parallelism 매개변수:** 작업을 실행해야 하는 임의의 시점에 병렬로 실행되는 Pod 수를 지정합니다.
 - 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 1입니다.
- **completions 매개변수:** 작업을 완료하는 데 필요한 성공적인 Pod 완료 횟수를 지정합니다.
 - 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 1입니다.
 - 완료 횟수가 고정된 병렬 작업의 경우 값을 지정합니다.
 - 작업 큐가 있는 병렬 작업의 경우 설정되지 않은 상태로 둡니다. 값을 설정하지 않는 경우 기본값은 **parallelism**입니다.

5.2.1.2. 최대 작업 기간 설정 방법 이해

작업을 정의할 때 `activeDeadlineSeconds` 필드를 설정하여 최대 기간을 정의할 수 있습니다. 이는 초 단위로 지정되며 기본적으로 설정되어 있지 않습니다. 설정하지 않으면 최대 기간이 적용되지 않습니다.

최대 기간은 시스템에서 첫 번째 **Pod**가 예약되는 시점부터 계산되며 작업을 활성 상태로 유지할 수 있는 기간을 정의합니다. 전체 실행 시간을 추적합니다. 지정된 타임아웃에 도달하면 **OpenShift Container Platform**에서 작업을 종료합니다.

5.2.1.3. Pod가 실패하는 경우 작업 백오프 정책을 설정하는 방법 이해

구성의 논리적 오류 또는 기타 유사한 이유로 인해 설정된 재시도 횟수를 초과하면 작업이 실패한 것으로 간주될 수 있습니다. 작업과 연결된 실패한 **Pod**는 급격한 백오프 지연(10s, 20s, 40s ...)을 6분으로 제한하여 컨트롤러에서 다시 생성합니다. 컨트롤러 확인 중 실패한 새 **Pod**가 표시되지 않으면 제한이 재설정됩니다.

`spec.backoffLimit` 매개변수를 사용하여 작업의 재시도 횟수를 설정합니다.

5.2.1.4. 아티팩트를 제거하도록 cron 작업을 구성하는 방법

`cron` 작업에서는 작업 또는 **Pod**와 같은 아티팩트 리소스를 남겨 둘 수 있습니다. 사용자는 이전 작업과 **Pod**가 올바르게 정리되도록 기록 제한을 구성하는 것이 중요합니다. `cron` 작업 사양 내에는 이러한 리소스를 담당하는 다음 두 필드가 있습니다.

- `.spec.successfulJobsHistoryLimit`. 유지해야 하는 성공적으로 완료한 작업의 수입니다(기본값: 3).
- `.spec.failedJobsHistoryLimit`. 유지해야 하는 실패한 작업의 수입니다(기본값: 1).

작은 정보

- 더 이상 필요하지 않은 **cron** 작업을 삭제합니다.

```
$ oc delete cronjob/<cron_job_name>
```

이렇게 하면 불필요한 아티팩트가 생성되지 않습니다.

- **spec.suspend**를 **true**로 설정하여 추가 실행을 중단할 수 있습니다. 모든 후속 실행은 **false**로 재설정할 때까지 일시 중단됩니다.

5.2.1.5. 알려진 제한 사항

작업 사양 재시작 정책은 작업 컨트롤러가 아닌 **Pod**에만 적용됩니다. 그러나 작업 컨트롤러는 작업을 완료할 때까지 계속 재시도하도록 하드 코딩되어 있습니다.

따라서 **restartPolicy: Never** 또는 **--restart=Never**는 **restartPolicy: OnFailure** 또는 **--restart=OnFailure**와 동일하게 작동합니다. 즉 작업이 실패하면 작업이 성공할 때까지 (또는 작업을 수동으로 삭제할 때까지) 자동으로 재시작됩니다. 이 정책은 재시작을 수행하는 하위 시스템만 설정합니다.

Never 정책에서는 작업 컨트롤러에서 재시작을 수행합니다. 시도할 때마다 작업 컨트롤러에서 작업 상태의 실패 수를 늘리고 새 **Pod**를 생성합니다. 즉 실패할 때마다 **Pod** 수가 증가합니다.

OnFailure 정책에서는 **kubelet**에서 재시작을 수행합니다. 시도할 때마다 작업 상태의 실패 횟수가 늘어난다는 것은 아닙니다. 또한 **kubelet**은 동일한 노드에서 **Pod**를 시작하여 실패한 작업을 재시도합니다.

5.2.2. 작업 생성

작업 오브젝트를 생성하여 **OpenShift Container Platform**에서 작업을 생성합니다.

프로세스

작업을 생성하려면 다음을 수행합니다.

1. 다음과 유사한 **YAML** 파일을 생성합니다.

```

apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 ①
  completions: 1 ②
  activeDeadlineSeconds: 1800 ③
  backoffLimit: 6 ④
  template: ⑤
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: OnFailure ⑥
#...

```

①

선택 사항: 작업에서 병렬로 실행해야 하는 Pod 복제본 수를 지정합니다. 기본값은 1입니다.

•

비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 1입니다.

②

선택 사항: 작업이 완료된 것으로 표시하는 데 필요한 성공적인 Pod 완료 수를 지정합니다.

•

비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 1입니다.

•

완료 횟수가 고정된 병렬 작업의 경우 완료 횟수를 지정합니다.

•

작업 큐가 있는 병렬 작업의 경우 설정되지 않은 상태로 둡니다. 값을 설정하지 않는 경우 기본값은 **parallelism**입니다.

③

선택 사항: 작업을 실행할 수 있는 최대 기간을 지정합니다.

4

선택 사항: 작업 재시도 횟수를 지정합니다. 이 필드의 기본값은 6입니다.

5

컨트롤러에서 생성하는 Pod에 사용할 템플릿을 지정합니다.

6

Pod 재시작 정책을 지정합니다.

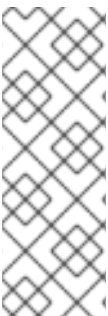
- **Never.** 작업을 재시작하지 않습니다.
- **OnFailure.** 실패하는 경우에만 작업을 재시작합니다.
- **Always** 작업을 항상 재시작합니다.

OpenShift Container Platform에서 실패한 컨테이너에 재시작 정책을 사용하는 방법에 대한 자세한 내용은 **Kubernetes** 설명서의 [예제 상태](#)를 참조하십시오.

2.

작업을 생성합니다.

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



참고

`oc create job`을 사용하여 단일 명령으로 작업을 생성하고 시작할 수도 있습니다. 다음 명령에서는 이전 예제에서 지정한 것과 유사한 작업을 생성하고 시작합니다.

`$ oc create job pi --image=perl -- perl -Mbignum=bpi -wle 'print bpi(2000)'`

5.2.3. cron 작업 생성

작업 오브젝트를 생성하여 OpenShift Container Platform에서 cron 작업을 생성합니다.

프로세스

cron 작업을 생성하려면 다음을 수행합니다.

1.

다음과 유사한 **YAML** 파일을 생성합니다.

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *" 1
  timeZone: Etc/UTC 2
  concurrencyPolicy: "Replace" 3
  startingDeadlineSeconds: 200 4
  suspend: true 5
  successfulJobsHistoryLimit: 3 6
  failedJobsHistoryLimit: 1 7
  jobTemplate: 8
    spec:
      template:
        metadata:
          labels: 9
            parent: "cronjobpi"
          spec:
            containers:
              - name: pi
                image: perl
                command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
            restartPolicy: OnFailure 10
#...
```

1

cron 형식에 지정된 작업의 스케줄입니다. 이 예제에서는 작업은 분마다 실행됩니다.

2

스케줄에 대한 선택적 시간대입니다. 유효한 옵션에 대해서는 **tz 데이터베이스 표준 시간대 목록**을 참조하십시오. 지정하지 않으면 **Kubernetes** 컨트롤러 관리자는 로컬 시간대를 기준으로 일정을 해석합니다. 이 설정은 **기술 프리뷰**로 제공됩니다.

3

cron 작업 내의 동시 작업 처리 방법을 지정하는 선택적 동시성 정책입니다. 다음 동시성 정책 중 하나만 지정할 수 있습니다. 지정하지 않는 경우 기본값은 동시 실행을 허용하는 것입니다.

- **Allow**는 **cron** 작업이 동시에 실행되도록 허용합니다.
- **Forbid**는 동시 실행을 금지하고 이전 실행이 아직 완료되지 않은 경우 다음 실행을 건너뛵니다.
- **Replace**는 현재 실행 중인 작업을 취소하고 새 작업으로 교체합니다.

4

어떠한 이유로 예약된 시간을 놓치는 경우 작업을 시작하는 선택적 데드라인(초)입니다. 누락된 작업 실행은 실패한 작업으로 간주됩니다. 지정하지 않으면 데드라인이 없습니다.

5

cron 작업의 중지를 허용하는 선택적 플래그입니다. **true**로 설정하면 이후의 모든 실행이 일시 중지됩니다.

6

유지해야 하는 성공적으로 완료한 작업의 수입입니다(기본값: 3).

7

유지해야 하는 실패한 작업의 수입입니다(기본값: 1).

8

작업 템플릿입니다. 이 템플릿은 작업 예제와 유사합니다.

9

이 **cron** 작업에서 생성한 작업에 라벨을 설정합니다.

10

Pod의 재시작 정책입니다. 이 정책은 작업 컨트롤러에 적용되지 않습니다.



참고

.spec.successfulJobsHistoryLimit 및 **.spec.failedJobsHistoryLimit** 필드는 선택 사항입니다. 해당 필드는 유지해야 하는 완료된 작업 수 및 실패한 작업 수를 지정합니다. 기본적으로 각각 3과 1로 설정됩니다. 제한을 0으로 설정하면 해당 종류의 작업을 완료한 후 작업을 유지하지 않습니다.

2.

cron 작업을 생성합니다.

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



참고

oc create cronjob을 사용하여 단일 명령으로 **cron** 작업을 생성하고 시작할 수도 있습니다. 다음 명령에서는 이전 예제에서 지정한 것과 유사한 **cron** 작업을 생성하고 시작합니다.

```
$ oc create cronjob pi --image=perl --schedule='*/1 * * * *' -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

oc create cronjob을 사용하면 **--schedule** 옵션에서 **cron** 형식으로 된 일정을 수락합니다.

6장. 노드 작업

6.1. OPENSIFT CONTAINER PLATFORM 클러스터에서 노드 보기 및 나열

클러스터의 모든 노드를 나열하여 노드의 상태, 수명, 메모리 사용량, 세부 정보와 같은 정보를 가져올 수 있습니다.

노드 관리 작업을 수행할 때 CLI는 실제 노드 호스트를 나타내는 노드 오브젝트와 상호 작용합니다. 마스터는 노드 오브젝트의 정보를 사용하여 상태 점검에서 노드를 검증합니다.

6.1.1. 클러스터의 모든 노드 나열 정보

클러스터의 노드에 대한 세부 정보를 가져올 수 있습니다.

- 다음 명령을 실행하면 모든 노드가 나열됩니다.

```
$ oc get nodes
```

다음 예제는 정상 노드가 있는 클러스터입니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
master.example.com	Ready	master	7h	v1.25.0
node1.example.com	Ready	worker	7h	v1.25.0
node2.example.com	Ready	worker	7h	v1.25.0

다음 예제는 하나의 비정상 노드가 있는 클러스터입니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION	
master.example.com	Ready	master	7h	v1.25.0	
node1.example.com	NotReady,SchedulingDisabled	worker	7h	v1.25.0	v1.25.0
node2.example.com	Ready	worker	7h	v1.25.0	

NotReady 상태를 트리거하는 조건은 이 섹션의 뒷부분에 나와 있습니다.

- **-wide** 옵션은 노드에 대한 추가 정보를 제공합니다.

```
$ oc get nodes -o wide
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
master.example.com	Ready	master	171m	v1.25.0	10.0.129.108	<none>	Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)	4.18.0-240.15.1.el8_3.x86_64	cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev
node1.example.com	Ready	worker	72m	v1.25.0	10.0.129.222	<none>	Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)	4.18.0-240.15.1.el8_3.x86_64	cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev
node2.example.com	Ready	worker	164m	v1.25.0	10.0.142.150	<none>	Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)	4.18.0-240.15.1.el8_3.x86_64	cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev

- 다음 명령에서는 단일 노드에 대한 정보를 나열합니다.

```
$ oc get node <node>
```

예를 들면 다음과 같습니다.

```
$ oc get node node1.example.com
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
node1.example.com	Ready	worker	7h	v1.25.0

다음 명령은 현재 조건의 이유를 포함하여 특정 노드에 대한 세부 정보를 제공합니다.

`$ oc describe node <node>`

예를 들면 다음과 같습니다.

`$ oc describe node node1.example.com`

출력 예

```

Name:          node1.example.com 1
Roles:         worker 2
Labels:        beta.kubernetes.io/arch=amd64 3
               beta.kubernetes.io/instance-type=m4.large
               beta.kubernetes.io/os=linux
               failure-domain.beta.kubernetes.io/region=us-east-2
               failure-domain.beta.kubernetes.io/zone=us-east-2a
               kubernetes.io/hostname=ip-10-0-140-16
               node-role.kubernetes.io/worker=
Annotations:   cluster.k8s.io/machine: openshift-machine-api/ahardin-worker-us-
east-2a-q5dzc 4
               machineconfiguration.openshift.io/currentConfig: worker-
309c228e8b3a92e2235edd544c62fea8
               machineconfiguration.openshift.io/desiredConfig: worker-
309c228e8b3a92e2235edd544c62fea8
               machineconfiguration.openshift.io/state: Done
               volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Wed, 13 Feb 2019 11:05:57 -0500
Taints:        <none> 5
Unschedulable: false
Conditions:    6
  Type          Status LastHeartbeatTime           LastTransitionTime        Reason
  Message
  ----          -

```

OutOfDisk *False* *Wed, 13 Feb 2019 15:09:42 -0500* *Wed, 13 Feb 2019 11:05:57 -0500* **KubeletHasSufficientDisk** *kubelet has sufficient disk space available*
MemoryPressure *False* *Wed, 13 Feb 2019 15:09:42 -0500* *Wed, 13 Feb 2019 11:05:57 -0500* **KubeletHasSufficientMemory** *kubelet has sufficient memory available*
DiskPressure *False* *Wed, 13 Feb 2019 15:09:42 -0500* *Wed, 13 Feb 2019 11:05:57 -0500* **KubeletHasNoDiskPressure** *kubelet has no disk pressure*
PIDPressure *False* *Wed, 13 Feb 2019 15:09:42 -0500* *Wed, 13 Feb 2019 11:05:57 -0500* **KubeletHasSufficientPID** *kubelet has sufficient PID available*
Ready *True* *Wed, 13 Feb 2019 15:09:42 -0500* *Wed, 13 Feb 2019 11:07:09 -0500* **KubeletReady** *kubelet is posting ready status*

Addresses: 7

InternalIP: 10.0.140.16
InternalDNS: ip-10-0-140-16.us-east-2.compute.internal
Hostname: ip-10-0-140-16.us-east-2.compute.internal

Capacity: 8

attachable-volumes-aws-ebs: 39
cpu: 2
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 8172516Ki
Pods: 250

Allocatable:

attachable-volumes-aws-ebs: 39
cpu: 1500m
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 7558116Ki
Pods: 250

System Info: 9

Machine ID: 63787c9534c24fde9a0cde35c13f1f66
System UUID: EC22BF97-A006-4A58-6AF8-0A38DEEA122A
Boot ID: f24ad37d-2594-46b4-8830-7f7555918325
Kernel Version: 3.10.0-957.5.1.el7.x86_64
OS Image: Red Hat Enterprise Linux CoreOS 410.8.20190520.0

(Ootpa)

Operating System: linux
Architecture: amd64
Container Runtime Version: cri-o://1.25.0-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
Kubelet Version: v1.25.0
Kube-Proxy Version: v1.25.0
PodCIDR: 10.128.4.0/24
ProviderID: aws:///us-east-2a/i-04e87b31dc6b3e171

Non-terminated Pods: (12 in total) 10

Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits
openshift-cluster-node-tuning-operator	tuned-hdl5q	0 (0%)	0 (0%)	0 (0%)	0 (0%)
openshift-dns	dns-default-l69zr	0 (0%)	0 (0%)	0 (0%)	0 (0%)
openshift-image-registry	node-ca-9hmcg	0 (0%)	0 (0%)	0 (0%)	0 (0%)
openshift-ingress	router-default-76455c45c-c5ptv	0 (0%)	0 (0%)	0 (0%)	0 (0%)

```

openshift-machine-config-operator    machine-config-daemon-cvqw9    20m
(1%)  0 (0%)  50Mi (0%)  0 (0%)
openshift-marketplace                community-operators-f67fh      0 (0%)  0 (0%)
0 (0%)  0 (0%)
openshift-monitoring                 alertmanager-main-0           50m (3%)  50m
(3%)  210Mi (2%)  10Mi (0%)
openshift-monitoring                 node-exporter-l7q8d           10m (0%)  20m
(1%)  20Mi (0%)  40Mi (0%)
openshift-monitoring                 prometheus-adapter-75d769c874-hvb85  0 (0%)
0 (0%)  0 (0%)  0 (0%)
openshift-multus                      multus-kw8w5                  0 (0%)  0 (0%)  0
(0%)  0 (0%)
openshift-sdn                         ovs-t4dsn                     100m (6%)  0 (0%)  300Mi
(4%)  0 (0%)
openshift-sdn                         sdn-g79hg                     100m (6%)  0 (0%)
200Mi (2%)  0 (0%)
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests  Limits
-----
cpu                 380m (25%)  270m (18%)
memory              880Mi (11%)  250Mi (3%)
attachable-volumes-aws-efs 0          0
Events:  11
Type     Reason           Age          From          Message
----
Normal   NodeHasSufficientPID  6d (x5 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal   NodeAllocatableEnforced  6d          kubelet, m01.example.com Updated
Node Allocatable limit across pods
Normal   NodeHasSufficientMemory  6d (x6 over 6d)  kubelet, m01.example.com
Node m01.example.com status is now: NodeHasSufficientMemory
Normal   NodeHasNoDiskPressure  6d (x6 over 6d)  kubelet, m01.example.com
Node m01.example.com status is now: NodeHasNoDiskPressure
Normal   NodeHasSufficientDisk  6d (x6 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientDisk
Normal   NodeHasSufficientPID  6d          kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal   Starting          6d          kubelet, m01.example.com Starting kubelet.
#...

```

1

노드의 이름입니다.

2

노드의 역할(master 또는 worker)입니다.

3

4

노드에 적용되는 주석입니다.

5

노드에 적용되는 테인트입니다.

6

노드 조건 및 상태입니다. **conditions** 스탠자는 **Ready, PIDPressure, PIDPressure, MemoryPressure, DiskPressure OutOfDisk** 상태를 나열합니다. 이러한 조건은 이 섹션의 뒷부분에 설명되어 있습니다.

7

노드의 IP 주소 및 호스트 이름입니다.

8

Pod 리소스 및 할당 가능한 리소스입니다.

9

노드 호스트에 대한 정보입니다.

10

노드의 Pod입니다.

11

노드에서 보고한 이벤트입니다.



참고

컨트롤 플레인 레이블은 새로 생성되거나 업데이트된 마스터 노드에 자동으로 추가되지 않습니다. 노드에 컨트롤 플레인 레이블을 사용하려면 레이블을 수동으로 구성할 수 있습니다. 자세한 내용은 추가 리소스 섹션의 노드에서 라벨을 업데이트하는 방법 이해를 참조하십시오.

노드에 표시된 정보 중에 다음 노드 상태가 이 섹션에 표시된 명령의 출력에 표시됩니다.

표 6.1. 노드 상태

상태	설명
Ready	true 인 경우 노드가 정상이고 Pod를 허용할 준비가 되었습니다. false 인 경우 노드에서 정상이 아니며 Pod를 허용하지 않습니다. unknown 인 경우 노드 컨트롤러에서 node-monitor-grace-period (기본값: 40초)에 대해 노드에서 하트비트를 수신하지 못했습니다.
DiskPressure	true 인 경우 디스크 용량이 적습니다.
MemoryPressure	true 인 경우 노드 메모리가 부족합니다.
PIDPressure	true 인 경우 노드에 너무 많은 프로세스가 있습니다.
OutOfDisk	true 인 경우 노드에 pod를 추가하는 노드의 여유 공간이 충분하지 않습니다.
NetworkUnavailable	true 인 경우 노드의 네트워크가 올바르게 구성되지 않습니다.
NotReady	true 인 경우 컨테이너 런타임 또는 네트워크와 같은 기본 구성 요소 중 하나에 문제가 있거나 이러한 구성 요소가 아직 구성되지 않았습니다.
SchedulingDisabled	Pod는 노드에 배치하도록 예약할 수 없습니다.

추가 리소스

- [노드에서 라벨을 업데이트하는 방법 이해](#)

6.1.2. 클러스터의 노드에 있는 Pod 나열

특정 노드의 모든 Pod를 나열할 수 있습니다.

프로세스

- [하나 이상의 노드에 있는 모든 Pod 또는 선택한 Pod를 나열하려면 다음을 수행합니다.](#)

```
$ oc describe node <node1> <node2>
```

예를 들면 다음과 같습니다.


```
$ oc describe node ip-10-0-128-218.ec2.internal
```

- 선택한 노드에서 모든 Pod 또는 선택한 Pod를 나열하려면 다음을 수행합니다.

```
$ oc describe node --selector=<node_selector>
```

```
$ oc describe node --selector=kubernetes.io/os
```

또는 다음을 수행합니다.

```
$ oc describe node -l=<pod_selector>
```

```
$ oc describe node -l node-role.kubernetes.io/worker
```

- 종료된 Pod를 포함하여 특정 노드의 모든 Pod를 나열하려면 다음을 수행합니다.

```
$ oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename>
```

6.1.3. 노드의 메모리 및 CPU 사용량 통계 보기

컨테이너에 런타임 환경을 제공하는 노드에 대한 사용량 통계를 표시할 수 있습니다. 이러한 사용량 통계에는 CPU, 메모리, 스토리지 사용량이 포함됩니다.

사전 요구 사항

- 사용량 통계를 보려면 **cluster-reader** 권한이 있어야 합니다.
- 사용량 통계를 보려면 메트릭이 설치되어 있어야 합니다.

프로세스

- 사용량 통계를 보려면 다음을 수행합니다.

```
$ oc adm top nodes
```

출력 예

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
ip-10-0-12-143.ec2.compute.internal	1503m	100%	4533Mi	61%
ip-10-0-132-16.ec2.compute.internal	76m	5%	1391Mi	18%
ip-10-0-140-137.ec2.compute.internal	398m	26%	2473Mi	33%
ip-10-0-142-44.ec2.compute.internal	656m	43%	6119Mi	82%
ip-10-0-146-165.ec2.compute.internal	188m	12%	3367Mi	45%
ip-10-0-19-62.ec2.compute.internal	896m	59%	5754Mi	77%
ip-10-0-44-193.ec2.compute.internal	632m	42%	5349Mi	72%

라벨을 사용하여 노드의 사용량 통계를 보려면 다음을 실행합니다.

```
$ oc adm top node --selector=
```

필터링할 선택기(라벨 쿼리)를 선택해야 합니다. =, ==, !=가 지원됩니다.

6.2. 노드 작업

관리자는 여러 작업을 수행하여 클러스터의 효율성을 높일 수 있습니다.

6.2.1. 노드에서 Pod를 비우는 방법 이해

Pod를 비우면 하나 이상의 지정된 노드에서 모든 **Pod** 또는 선택한 **Pod**를 마이그레이션할 수 있습니다.

복제 컨트롤러에서 지원하는 **Pod**만 비울 수 있습니다. 복제 컨트롤러는 다른 노드에서 새 **Pod**를 생성하고 지정된 노드에서 기존 **Pod**를 제거합니다.

복제 컨트롤러에서 지원하지 않는 배어 **Pod**는 기본적으로 영향을 받지 않습니다. **Pod** 선택기를 지정하여 **Pod**의 하위 집합을 비울 수 있습니다. **Pod** 선택기는 라벨을 기반으로 하므로 라벨이 지정된 **Pod**를 모두 비웁니다.

프로세스

1.

Pod 비우기를 수행하기 전에 노드를 예약 불가능으로 표시합니다.

a.

노드를 예약 불가능으로 표시합니다.

```
$ oc adm cordon <node1>
```

출력 예

```
node/<node1> cordoned
```

b.

노드 상태가 **Ready,SchedulingDisabled** 인지 확인합니다.

```
$ oc get node <node1>
```

출력 예

```
NAME          STATUS          ROLES    AGE    VERSION
<node1>      Ready,SchedulingDisabled  worker   1d     v1.25.0
```

2.

다음 메서드 중 하나를 사용하여 **Pod**를 비웁니다.

•

하나 이상의 노드에 있는 모든 **Pod** 또는 선택한 **Pod**를 비웁니다.

```
$ oc adm drain <node1> <node2> [--pod-selector=<pod_selector>]
```

•

--force 옵션을 사용하여 배어 **Pod**를 강제 삭제합니다. **true**로 설정하면 복제 컨트롤러, 복제본 세트, 작업, 데몬 세트 또는 상태 저장 세트에서 관리하지 않는 **Pod**가 있는 경우에도 계속 삭제합니다.

```
$ oc adm drain <node1> <node2> --force=true
```

- 각 Pod가 정상적으로 종료되는 시간(초)을 설정하고 `--grace-period` 를 사용합니다. 음수인 경우 Pod에 지정된 기본값이 사용됩니다.

```
$ oc adm drain <node1> <node2> --grace-period=-1
```

- `true`로 설정된 `--ignore-daemonsets` 플래그를 사용하여 데몬 세트에서 관리하는 Pod를 무시합니다.

```
$ oc adm drain <node1> <node2> --ignore-daemonsets=true
```

- `--timeout` 플래그를 사용하여 종료하기 전에 대기하는 시간을 설정합니다. 값이 0이면 시간이 제한되지 않습니다.

```
$ oc adm drain <node1> <node2> --timeout=5s
```

- `--delete-emptydir-data` 플래그를 `true` 로 설정하여 `emptyDir` 볼륨을 사용하는 Pod가 있는 경우에도 Pod를 삭제합니다. 노드가 드레인되면 로컬 데이터가 삭제됩니다.

```
$ oc adm drain <node1> <node2> --delete-emptydir-data=true
```

- `--dry-run` 옵션을 `true`로 설정하여 실제로 비우기를 수행하지 않고 마이그레이션할 오브젝트를 나열합니다.

```
$ oc adm drain <node1> <node2> --dry-run=true
```

특정 노드 이름(예: `<node1> <node2>`)을 지정하는 대신 `--selector=<node_selector>` 옵션을 사용하여 선택한 노드에서 Pod를 비울 수 있습니다.

3. 완료되면 노드를 예약 가능으로 표시합니다.

```
$ oc adm unccordon <node1>
```

6.2.2. 노드에서 라벨을 업데이트하는 방법 이해

노드에서 임의의 라벨을 업데이트할 수 있습니다.

머신에서 노드를 백업해도 노드를 삭제하면 노드 라벨이 유지되지 않습니다.



참고

MachineSet 오브젝트의 변경 사항은 컴퓨팅 머신 세트가 소유한 기존 머신에는 적용되지 않습니다. 예를 들어 기존 **MachineSet** 오브젝트에서 편집 또는 추가된 레이블은 컴퓨팅 머신 세트와 연결된 기존 머신 및 노드로 전달되지 않습니다.

- 다음 명령은 노드에서 라벨을 추가하거나 업데이트합니다.

```
$ oc label node <node> <key_1>=<value_1> ... <key_n>=<value_n>
```

예를 들면 다음과 같습니다.

```
$ oc label nodes webconsole-7f7f6 unhealthy=true
```

작은 정보

다음 **YAML** 을 적용하여 레이블을 적용할 수도 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: webconsole-7f7f6
  labels:
    unhealthy: 'true'
#...
```

- 다음 명령은 네임스페이스의 모든 **Pod**를 업데이트합니다.

```
$ oc label pods --all <key_1>=<value_1>
```

예를 들면 다음과 같습니다.

```
$ oc label pods --all status=unhealthy
```

6.2.3. 노드를 예약 가능 또는 예약 불가능으로 표시하는 방법 이해

기본적으로 **Ready** 상태의 정상 노드는 예약 가능으로 표시됩니다. 즉 노드에 새 **Pod**를 배치할 수 있습니다. 수동으로 노드를 예약 불가로 표시하면 새 **Pod**가 노드에 예약되지 않도록 차단됩니다. 노드의 기존 **Pod**에는 영향을 미치지 않습니다.

- 다음 명령은 하나 이상의 노드를 예약 불가로 표시합니다.

출력 예

```
$ oc adm cordon <node>
```

예를 들면 다음과 같습니다.

```
$ oc adm cordon node1.example.com
```

출력 예

```
node/node1.example.com cordoned
```

NAME	LABELS	STATUS
node1.example.com	kubernetes.io/hostname=node1.example.com	Ready,SchedulingDisabled

- 다음 명령은 현재 예약할 수 없는 하나 이상의 노드를 예약 가능으로 표시합니다.

```
$ oc adm uncordon <node1>
```

또는 특정 노드 이름(예: <node>)을 지정하는 대신 **--selector=<node_selector>** 옵션을 사용하여 선택한 노드를 예약 가능 또는 예약 불가로 표시할 수 있습니다.

6.2.4. 애플리케이션 Pod를 드레이닝하지 않고 노드가 재부팅될 때 단일 노드 OpenShift 클러스터에서 오류 처리

단일 노드 OpenShift 클러스터 및 일반적으로 OpenShift Container Platform 클러스터에서는 먼저 노드를 드레이닝하지 않고 노드 재부팅이 발생하는 상황이 발생할 수 있습니다. 이는 UnexpectedAdmissionError 오류로 장치를 요청하는 애플리케이션 Pod가 실패하는 경우가 발생할 수 있습니다. 이러한 장치를 제공해야 하는 애플리케이션 Pod가 시작되므로 배포, ReplicaSet 또는 DaemonSet 오류가 보고됩니다. Pod 재시작 순서를 제어할 수 없습니다.

이 동작이 예상되지만 배포에 실패한 경우에도 Pod가 클러스터에 남아 있을 수 있습니다. Pod는 UnexpectedAdmissionError 를 계속 보고합니다. 이 문제는 애플리케이션 Pod가 일반적으로 Deployment, ReplicaSet 또는 DaemonSet 에 포함되어 있다는 사실에 의해 완화됩니다. Pod가 이 오류 상태에 있는 경우 다른 인스턴스를 실행해야 하므로 문제가 거의 없습니다. Deployment, ReplicaSet 또는 DaemonSet 에 속하는 경우 후속 Pod를 성공적으로 생성 및 실행하고 애플리케이션을 성공적으로 배포할 수 있습니다.

이러한 Pod가 정상적으로 종료되도록 작업 업스트림이 진행 중입니다. 작업이 해결될 때까지 단일 노드 OpenShift 클러스터에 대해 다음 명령을 실행하여 실패한 포드를 제거합니다.

```
$ oc delete pods --field-selector status.phase=Failed -n <POD_NAMESPACE>
```



참고

단일 노드 OpenShift 클러스터에서 노드를 드레이닝하는 옵션을 사용할 수 없습니다.

추가 리소스

- [노드에서 Pod를 비우는 방법 이해](#)

6.2.5. 노드 삭제

6.2.5.1. 클러스터에서 노드 삭제

OpenShift Container Platform 클러스터에서 노드를 삭제하려면 적절한 MachineSet 오브젝트를 축소합니다.



중요

클러스터가 클라우드 공급자와 통합되면 해당 시스템을 삭제하여 노드를 삭제해야 합니다. 이 작업에 `oc delete node` 명령을 사용하지 마십시오.

CLI를 사용하여 노드를 삭제하면 **Kubernetes**에서 노드 오브젝트가 삭제되지만 노드에 존재하는 **Pod**는 삭제되지 않습니다. 복제 컨트롤러에서 지원하지 않는 모든 베어 **Pod**는 **OpenShift Container Platform**에서 액세스할 수 없습니다. 복제 컨트롤러에서 지원하는 **Pod**는 사용 가능한 다른 노드로 다시 예약됩니다. 로컬 매니페스트 **Pod**를 삭제해야 합니다.



참고

베어 메탈에서 클러스터를 실행 중인 경우 **MachineSet** 오브젝트를 편집하여 노드를 삭제할 수 없습니다. 컴퓨팅 머신 세트는 클러스터가 클라우드 공급자와 통합된 경우에만 사용할 수 있습니다. 대신 수동으로 삭제하기 전에 스케줄 예약을 취소하고 노드를 드레인해야 합니다.

프로세스

1.

다음 명령을 실행하여 클러스터에 있는 컴퓨팅 머신 세트를 확인합니다.

```
$ oc get machinesets -n openshift-machine-api
```

컴퓨팅 머신 세트는 `<cluster-id>-worker-<aws-region-az>` 형식으로 나열됩니다.

2.

다음 방법 중 하나를 사용하여 컴퓨팅 머신 세트를 축소합니다.

-

다음 명령을 실행하여 축소할 복제본 수를 지정합니다.

```
$ oc scale --replicas=2 machineset <machine-set-name> -n openshift-machine-api
```

-

다음 명령을 실행하여 컴퓨팅 머신 세트 사용자 정의 리소스를 편집합니다.

```
$ oc edit machineset <machine-set-name> -n openshift-machine-api
```

출력 예


```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  # ...
  name: <machine-set-name>
  namespace: openshift-machine-api
  # ...
spec:
  replicas: 2 1
  # ...

```

1

축소할 복제본 수를 지정합니다.

추가 리소스

•

컴퓨팅 머신 세트 수동 스케일링

6.2.5.2. 베어 메탈 클러스터에서 노드 삭제

CLI를 사용하여 노드를 삭제하면 **Kubernetes**에서 노드 오브젝트가 삭제되지만 노드에 존재하는 **Pod**는 삭제되지 않습니다. 복제 컨트롤러에서 지원하지 않는 기본 **Pod**는 **OpenShift Container Platform**에 액세스할 수 없습니다. 복제 컨트롤러에서 지원하는 **Pod**는 사용 가능한 다른 노드로 다시 예약됩니다. 로컬 매니페스트 **Pod**를 삭제해야 합니다.

절차

다음 단계를 완료하여 베어 메탈에서 실행 중인 **OpenShift Container Platform** 클러스터에서 노드를 삭제합니다.

1.

노드를 예약 불가능으로 표시합니다.

```
$ oc adm cordon <node_name>
```

2.

노드의 모든 **Pod**를 드레이닝합니다.

```
$ oc adm drain <node_name> --force=true
```

노드가 오프라인 상태이거나 응답하지 않는 경우 이 단계가 실패할 수 있습니다. 노드가 응답하지 않더라도 공유 스토리지에 쓰는 워크로드를 계속 실행되고 있을 수 있습니다. 데이터 손상을 방지하려면 계속하기 전에 물리적 하드웨어의 전원을 끕니다.

- 3. 클러스터에서 노드를 삭제합니다.

```
$ oc delete node <node_name>
```

노드 오브젝트가 클러스터에서 삭제되어도 재부팅 후 또는 kubelet 서비스가 재시작되면 클러스터에 다시 참여할 수 있습니다. 노드와 노드의 모든 데이터를 영구적으로 삭제하려면 노드를 해제해야 합니다.

- 4. 물리 하드웨어의 전원을 끈 경우 노드가 클러스터에 다시 참여할 수 있도록 해당 하드웨어를 다시 켭니다.

6.3. 노드 관리

OpenShift Container Platform에서는 KubeletConfig CR(사용자 정의 리소스)을 사용하여 노드의 구성을 관리합니다. KubeletConfig 오브젝트의 인스턴스를 생성하면 관리형 머신 구성이 생성되어 노드의 설정을 덮어씁니다.



참고

구성을 변경하기 위해 원격 머신에 로그인하는 것은 지원되지 않습니다.

6.3.1. 노드 수정

클러스터 또는 머신 풀을 구성하려면 CRD(사용자 정의 리소스 정의) 또는 kubeletConfig 오브젝트를 생성해야 합니다. OpenShift Container Platform에서는 CRD를 통해 도입된 변경 사항을 확인하는 데 Machine Config Controller를 사용하여 클러스터에 변경 사항을 적용합니다.



참고

kubeletConfig 오브젝트의 필드가 업스트림 Kubernetes에서 kubelet으로 직접 전달되므로 kubelet 자체에서 해당 필드의 유효성 검사가 직접 처리됩니다. 이러한 필드에 유효한 값은 관련 Kubernetes 설명서를 참조하십시오. kubeletConfig 오브젝트에서 유효하지 않은 값은 클러스터 노드를 사용할 수 없게 렌더링할 수 있습니다.

절차

1. 구성하려는 노드 유형의 정적 **CRD**인 머신 구성 풀과 연결된 라벨을 가져옵니다. 다음 중 하나를 실행합니다.

- a. 원하는 머신 구성 풀의 현재 라벨을 확인합니다.

예를 들면 다음과 같습니다.

```
$ oc get machineconfigpool --show-labels
```

출력 예

```
NAME CONFIG UPDATED UPDATING DEGRADED
LABELS
master rendered-master-e05b81f5ca4db1d249a1bf32f9ec24fd True False
False operator.machineconfiguration.openshift.io/required-for-upgrade=
worker rendered-worker-f50e78e1bc06d8e82327763145bfcf62 True False
False
```

- b. 원하는 머신 구성 풀에 사용자 정의 라벨을 추가합니다.

예를 들면 다음과 같습니다.

```
$ oc label machineconfigpool worker custom-kubelet=enabled
```

2. 구성 변경에 대한 **kubeletconfig CR**(사용자 정의 리소스)을 생성합니다.

예를 들면 다음과 같습니다.

custom-config CR 구성 샘플

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
```

```

metadata:
  name: custom-config ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled ❷
  kubeletConfig: ❸
    podsPerCore: 10
    maxPods: 250
    systemReserved:
      cpu: 2000m
      memory: 1Gi
#...

```

❶

CR에 이름을 지정합니다.

❷

구성 변경 사항을 적용하려면 라벨을 지정합니다. 이 라벨은 머신 구성 풀에 추가한 라벨입니다.

❸

변경할 새 값을 지정합니다.

3.

CR 오브젝트를 생성합니다.

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

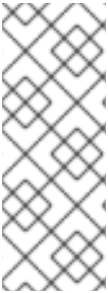
예를 들면 다음과 같습니다.

```
$ oc create -f master-kube-config.yaml
```

대부분의 **Kubelet** 구성 옵션은 사용자가 설정할 수 있습니다. 다음 옵션은 덮어쓸 수 없습니다.

- **CgroupDriver**

- **ClusterDNS**
- **ClusterDomain**
- **StaticPodPath**



참고

단일 노드에 50개 이상의 이미지가 포함된 경우 노드 간에 Pod 예약이 불균형될 수 있습니다. 이는 노드의 이미지 목록이 기본적으로 50으로 단축되기 때문입니다. **KubeletConfig** 오브젝트를 편집하고 **nodeStatusMaxImages** 값을 -1로 설정하여 이미지 제한을 비활성화할 수 있습니다.

6.3.2. 컨트롤 플레인 노드를 예약 가능으로 구성

컨트롤 플레인 노드를 예약할 수 있도록 구성할 수 있습니다. 즉, 마스터 노드에 새 Pod를 배치할 수 있습니다. 기본적으로 컨트롤 플레인 노드는 예약할 수 없습니다.

마스터는 예약 가능으로 설정할 수 있지만 작업자 노드는 유지해야 합니다.



참고

베어 메탈 클러스터에 작업자 노드 없이 **OpenShift Container Platform**을 배포할 수 있습니다. 이 경우 컨트롤 플레인 노드는 기본적으로 예약 가능으로 표시됩니다.

mastersSchedulable 필드를 구성하여 컨트롤 플레인 노드를 예약할 수 없도록 허용하거나 허용하지 않을 수 있습니다.



중요

기본 예약 불가에서 예약 가능으로 컨트롤 플레인 노드를 구성하면 추가 서브스크립션이 필요합니다. 그 이유는 컨트롤 플레인 노드가 작업자 노드가 되기 때문입니다.

프로세스

1. `schedulers.config.openshift.io` 리소스를 편집합니다.

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

2. `mastersSchedulable` 필드를 구성합니다.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: "2019-09-10T03:04:05Z"
  generation: 1
  name: cluster
  resourceVersion: "433"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: a636d30a-d377-11e9-88d4-0a60097bee62
spec:
  mastersSchedulable: false 1
status: {}
#...
```

1

컨트롤 플레인 노드를 예약할 수 없도록 하려면 `true` 로 설정하거나 `false` 로 설정하여 컨트롤 플레인 노드를 예약할 수 없습니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

6.3.3. SELinux 부울 설정

OpenShift Container Platform을 사용하면 RHCOS(Red Hat Enterprise Linux CoreOS) 노드에서 SELinux 부울을 활성화 및 비활성화할 수 있습니다. 다음 절차에서는 MCO(Machine Config Operator)를 사용하여 노드에서 SELinux 부울을 수정하는 방법을 설명합니다. 이 절차에서는 `container_manage_cgroup` 을 예제 부울로 사용합니다. 이 값을 필요한 부울로 수정할 수 있습니다.

사전 요구 사항

- OpenShift CLI(`oc`)가 설치되어 있습니다.

프로세스

1. 다음 예에 표시된 `MachineConfig` 오브젝트를 사용하여 새 `YAML` 파일을 만듭니다.

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Set SELinux booleans
            Before=kubelet.service

            [Service]
            Type=oneshot
            ExecStart=/sbin/setsebool container_manage_cgroup=on
            RemainAfterExit=true

            [Install]
            WantedBy=multi-user.target graphical.target
            enabled: true
            name: setsebool.service
#...

```

2.

다음 명령을 실행하여 새 **MachineConfig** 오브젝트를 만듭니다.

```
$ oc create -f 99-worker-setsebool.yaml
```



참고

MachineConfig 오브젝트에 변경 사항을 적용하면 변경 사항이 적용된 후 영향을 받는 모든 노드가 정상적으로 재부팅됩니다.

6.3.4. 노드에 커널 인수 추가

특별한 경우에는 클러스터 노드 세트에 커널 인수를 추가해야 할 수 있습니다. 이 작업을 수행할 때 주의해야 하며 먼저 설정된 인수의 영향을 명확하게 이해하고 있어야 합니다.



주의

커널 인수를 잘못 사용하면 시스템이 부팅되지 않을 수 있습니다.

설정할 수 있는 커널 인수의 예는 다음과 같습니다.

- nosmt:** 커널에서 대칭 멀티 스레딩 (SMT)을 비활성화합니다. 멀티 스레딩은 각 CPU마다 여러 개의 논리 스레드를 허용합니다. 멀티 테넌트 환경에서 **nosmt**를 사용하여 잠재적인 크로스 스레드 공격 위험을 줄일 수 있습니다. SMT를 비활성화하는 것은 기본적으로 성능보다는 보안을 중요시하여 선택하는 것과 같습니다.
- systemd.unified_cgroup_hierarchy:** [Linux 제어 그룹 버전 2 \(cgroup v2\)](#)를 활성화합니다. **cgroup v2**는 커널 [제어 그룹](#)의 다음 버전입니다.

중요

OpenShift Container Platform cgroups 버전 2는 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.

- enforcing=0:** SELinux(Security Enhanced Linux)를 허용 모드에서 실행하도록 구성합니다. 허용 모드에서는 SELinux가 개체에 레이블을 지정하고 로그에 액세스 거부 항목을 내보내는 등 로드된 보안 정책을 적용하는 것처럼 동작하지만 실제로는 어떤 작업도 거부하지 않습니다. 프로덕션 시스템에는 지원되지 않지만 허용 모드는 디버깅에 유용할 수 있습니다.



주의

프로덕션에서 RHCOS에서 SELinux를 비활성화하는 것은 지원되지 않습니다. 노드에서 SELinux를 비활성화한 후에는 프로덕션 클러스터에서 다시 프로비저닝해야 합니다.

커널 인수 목록 및 설명은 [Kernel.org](https://www.kernel.org) 커널 매개변수에서 참조하십시오.

다음 프로세스에서는 다음을 식별하는 MachineConfig를 만듭니다.

- 커널 인수를 추가하려는 머신 세트입니다. 이 경우 작업자 역할을 갖는 머신입니다.
- 기존 커널 인수 끝에 추가되는 커널 인수입니다.
- 머신 구성 목록에서 변경 사항이 적용되는 위치를 나타내는 라벨입니다.

사전 요구 사항

- OpenShift Container Platform 클러스터에 대한 관리자 권한을 보유하고 있어야 합니다.

절차

1.

OpenShift Container Platform 클러스터의 기존 MachineConfig 오브젝트를 나열하고 머신 구성에 라벨을 지정하는 방법을 결정합니다.

```
$ oc get MachineConfig
```

출력 예

NAME	GENERATEDBYCONTROLLER
IGNITIONVERSION AGE	
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9

```

3.2.0      33m
00-worker                                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
01-master-container-runtime
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
01-master-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
01-worker-container-runtime
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
01-worker-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
99-master-generated-registries
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
99-master-ssh                            3.2.0      40m
99-worker-generated-registries
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
99-worker-ssh                            3.2.0      40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
    
```

2.

커널 인수를 식별하는 **MachineConfig** 파일을 만듭니다 (예: **05-worker-kernelarg-selinuxpermissive.yaml**).

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 05-worker-kernelarg-selinuxpermissive 2
spec:
  kernelArguments:
    - enforcing=0 3
    
```

1

새 커널 인수를 작업자 노드에만 적용합니다.

2

머신 구성(05) 중 적합한 위치와 어떤 기능 (**SELinux** 허용 모드를 구성하기 위해 커널 매개변수 추가)을 하는지 식별하기 위해 이름이 지정됩니다.

3

정확한 커널 인수를 **enforcing=0**으로 식별합니다.

3. 새 머신 구성을 생성합니다.

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. 머신 구성에서 새 구성이 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

출력 예

NAME	GENERATEDBY	CONTROLLER
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
00-worker	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
05-worker-kernelarg-selinuxpermissive		3.2.0
105s		
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m
99-master-ssh		3.2.0 40m
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m
99-worker-ssh		3.2.0 40m
rendered-master-23e785de7587df95a4b517e0647e5ab7	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0 33m

5. 노드를 확인합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-136-161.ec2.internal	Ready	worker	28m	v1.25.0
ip-10-0-136-243.ec2.internal	Ready	master	34m	v1.25.0
ip-10-0-141-105.ec2.internal	Ready,SchedulingDisabled	worker	28m	v1.25.0
ip-10-0-142-249.ec2.internal	Ready	master	34m	v1.25.0
ip-10-0-153-11.ec2.internal	Ready	worker	28m	v1.25.0
ip-10-0-153-150.ec2.internal	Ready	master	34m	v1.25.0

변경 사항이 적용되어 있기 때문에 각 작업자 노드의 예약이 비활성화되어 있음을 알 수 있습니다.

6.

작업자 노드 중 하나로 이동하여 커널 명령 행 인수 (호스트의 `/proc/cmdline` 에 있음)를 나열하여 커널 인수가 작동하는지 확인합니다.

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

출력 예

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

`enforcing=0` 인수가 다른 커널 인수에 추가된 것을 확인할 수 있습니다.

6.3.5. 노드에서 스왑 메모리 사용 활성화

중요

노드에서 스왑 메모리 사용을 활성화하는 것은 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.

노드별로 **OpenShift Container Platform** 워크로드에 대한 스왑 메모리 사용을 활성화할 수 있습니다.



주의

스왑 메모리를 활성화하면 워크로드 성능 및 리소스 부족 처리에 부정적인 영향을 미칠 수 있습니다. 컨트롤 플레인 노드에서 스왑 메모리를 활성화하지 마십시오.

스왑 메모리를 활성화하려면 **kubeletconfig CR**(사용자 정의 리소스)을 생성하여 **swapbehavior** 매개 변수를 설정합니다. 제한되거나 무제한 스왑 메모리를 설정할 수 있습니다.

- 제한: **LimitedSwap** 값을 사용하여 스왑 메모리 워크로드를 사용할 수 있는 양을 제한합니다. **OpenShift Container Platform**에서 관리하지 않는 노드의 워크로드는 여전히 스왑 메모리를 사용할 수 있습니다. 제한적Swap 동작은 노드가 **Linux** 제어 그룹 **1(cgroups v1)** 또는 버전 **2(cgroup v 2)**에서 실행 중인지에 따라 달라집니다.
 - **cgroup v1: OpenShift Container Platform** 워크로드는 설정된 경우 Pod의 메모리 제한까지 메모리와 스왑의 조합을 사용할 수 있습니다.
 - **cgroup v2: OpenShift Container Platform** 워크로드는 스왑 메모리를 사용할 수 없습니다.
- 무제한: **restrictions Swap** 값을 사용하여 워크로드가 요청 시 시스템 제한까지 많은 스왑 메모리를 사용할 수 있도록 합니다.

kubelet은 이 구성 없이 스왑 메모리가 없으면 시작되지 않으므로 노드에서 스왑 메모리를 활성화하기 전에 **OpenShift Container Platform**에서 스왑 메모리를 활성화해야 합니다. 노드에 스왑 메모리가 없는 경우 **OpenShift Container Platform**에서 스왑 메모리를 활성화해도 적용되지 않습니다.

사전 요구 사항

- 버전 4.10 이상을 사용하는 **OpenShift Container Platform** 클러스터가 실행 중입니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- 클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화했습니다(노드 → 클러스터 작업 참조 → 기능 게이트를 사용하여 기능활성화).



참고

TechPreviewNoUpgrade 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 이러한 기능 세트는 프로덕션 클러스터에서 권장되지 않습니다.

- 노드에서 **cgroup v2**가 활성화된 경우 **swapaccount=1** 커널 인수를 설정하여 노드에서 스왑 계정을 활성화해야 합니다.

절차

1. 스왑 메모리를 허용하려는 머신 구성 풀에 사용자 정의 레이블을 적용합니다.

```
$ oc label machineconfigpool worker kubelet-swap=enabled
```

2. 스왑 설정을 활성화하고 구성할 사용자 정의 리소스(CR)를 만듭니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: swap-config
spec:
  machineConfigPoolSelector:
```

```

matchLabels:
  kubelet-swap: enabled
kubeletConfig:
  failSwapOn: false ①
  memorySwap:
    swapBehavior: LimitedSwap ②
#...

```

①

관련 노드에서 스왑 메모리 사용을 활성화하려면 **false** 로 설정합니다. 스왑 메모리 사용을 비활성화하려면 **true** 로 설정합니다.

②

스왑 메모리 동작을 지정합니다. 지정되지 않은 경우 기본값은 **LimitedSwap** 입니다.

3.

시스템에서 스왑 메모리를 활성화합니다.

6.3.6. 하나의 RHOSP 호스트에서 다른 RHOSP 호스트로 컨트롤 플레인 노드를 마이그레이션

하나의 RHOSP(Red Hat OpenStack Platform) 노드에서 다른 노드로 컨트롤 플레인 노드를 이동하는 스크립트를 실행할 수 있습니다.

사전 요구 사항

- 환경 변수 **OS_CLOUD** 는 **clouds.yaml** 파일에 관리 인증 정보가 있는 클라우드 항목을 나타냅니다.
- 환경 변수 **KUBECONFIG** 는 관리 **OpenShift Container Platform** 인증 정보가 포함된 구성을 나타냅니다.

절차

- 명령줄에서 다음 스크립트를 실행합니다.

```

#!/usr/bin/env bash

set -Eeuo pipefail

if [ $# -lt 1 ]; then
  echo "Usage: '$0 node_name'"

```

```

exit 64
fi

# Check for admin OpenStack credentials
openstack server list --all-projects >/dev/null || { >&2 echo "The script needs OpenStack admin
credentials. Exiting"; exit 77; }

# Check for admin OpenShift credentials
oc adm top node >/dev/null || { >&2 echo "The script needs OpenShift admin credentials.
Exiting"; exit 77; }

set -x

declare -r node_name="$1"
declare server_id
server_id="$(openstack server list --all-projects -f value -c ID -c Name | grep "$node_name" |
cut -d' ' -f1)"
readonly server_id

# Drain the node
oc adm cordon "$node_name"
oc adm drain "$node_name" --delete-emptydir-data --ignore-daemonsets --force

# Power off the server
oc debug "node/${node_name}" -- chroot /host shutdown -h 1

# Verify the server is shut off
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5;
done

# Migrate the node
openstack server migrate --wait "$server_id"

# Resize the VM
openstack server resize confirm "$server_id"

# Wait for the resize confirm to finish
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5;
done

# Restart the VM
openstack server start "$server_id"

# Wait for the node to show up as Ready:
until oc get node "$node_name" | grep -q "^${node_name}[:space:]+\+Ready"; do sleep 5;
done

# Uncordon the node
oc adm uncordon "$node_name"

# Wait for cluster operators to stabilize
until oc get co -o go-template='status: { { range .items } } { { range .status.conditions } } { { if eq
.type "Degraded" } } { { if ne .status "False" } } DEGRADED { { end } } { { else if eq .type
"Progressing" } } { { if ne .status "False" } } PROGRESSING { { end } } { { else if eq .type "Available" } }
{ { if ne .status "True" } } NOTAVAILABLE { { end } } { { end } } { { end } } { { end } } | grep -qv \
(DEGRADED|PROGRESSING|NOTAVAILABLE)'; do sleep 5; done

```


스크립트가 완료되면 컨트롤 플레인 시스템이 새 **RHOSP** 노드로 마이그레이션됩니다.

6.4. 노드당 최대 **POD** 수 관리

OpenShift Container Platform에서는 노드의 프로세서 코어 수, 하드 제한 또는 둘 다에 따라 노드에서 실행할 수 있는 **Pod** 수를 구성할 수 있습니다. 두 옵션을 모두 사용하는 경우 두 옵션 중 더 낮은 값이 노드의 **Pod** 수를 제한합니다.

옵션을 둘 다 사용하는 경우 한 노드의 **Pod** 수는 두 값 중 작은 값으로 제한됩니다. 이 값을 초과하면 다음과 같은 결과가 발생할 수 있습니다.

- CPU 사용률 증가
- **Pod** 예약 속도 저하.
- 노드의 메모리 크기에 따라 메모리 부족 시나리오 발생.
- IP 주소 모두 소진
- 리소스 과다 할당으로 인한 사용자 애플리케이션 성능 저하.

중요

Kubernetes의 경우 단일 컨테이너를 보유한 하나의 **Pod**에서 실제로 두 개의 컨테이너가 사용됩니다. 두 번째 컨테이너는 실제 컨테이너 시작 전 네트워킹 설정에 사용됩니다. 따라서 10개의 **Pod**를 실행하는 시스템에서는 실제로 20개의 컨테이너가 실행됩니다.

참고

클라우드 공급자의 디스크 **IOPS** 제한이 **CRI-O** 및 **kubelet**에 영향을 미칠 수 있습니다. 노드에서 다수의 **I/O** 집약적 **Pod**가 실행되고 있는 경우 오버로드될 수 있습니다. 노드에서 디스크 **I/O**를 모니터링하고 워크로드에 대해 처리량이 충분한 볼륨을 사용하는 것이 좋습니다.

podsWithCore 매개변수는 노드의 프로세서 코어 수에 따라 노드에서 실행할 수 있는 Pod 수를 설정합니다. 예를 들어 프로세서 코어가 4개인 노드에서 **podsWithCore**가 10으로 설정된 경우 노드에 허용되는 최대 Pod 수는 40이 됩니다.

```
kubeletConfig:
  podsWithCore: 10
```

podsWithCore를 0으로 설정하면 이 제한이 비활성화됩니다. 기본값은 0입니다. **podsWithCore** 매개변수 값은 **maxPods** 매개변수 값을 초과할 수 없습니다.

maxPods 매개변수는 노드의 속성에 관계없이 노드가 실행할 수 있는 Pod 수를 고정된 값으로 설정합니다.

```
kubeletConfig:
  maxPods: 250
```

6.4.1. 노드 당 최대 pod 수 구성

podsWithCore 및 **maxPods**는 노드에 예약할 수 있는 최대 Pod 수를 제어합니다. 두 옵션을 모두 사용하는 경우 두 옵션 중 더 낮은 값이 노드의 Pod 수를 제한합니다.

예를 들어 4 개의 프로세서 코어가 있는 노드에서 **podsWithCore**가 10으로 설정된 경우 노드에서 허용되는 최대 Pod 수는 40입니다.

사전 요구 사항

1. 다음 명령을 입력하여 구성할 노드 유형의 정적 **MachineConfigPool CRD**와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...

```

1

레이블이 **Labels** 아래에 표시됩니다.

작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

절차

1.

구성 변경을 위한 사용자 정의 리소스 (**CR**)를 만듭니다.

max-pods CR의 설정 예

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    podsWithCore: 10 3
    maxPods: 250 4
#...

```

1

CR에 이름을 지정합니다.

2

머신 구성 풀에서 라벨을 지정합니다.

3

노드의 프로세서 코어 수에 따라 노드가 실행할 수 있는 Pod 수를 지정합니다.

4

노드의 속성에 관계없이 노드가 고정 값으로 실행할 수 있는 Pod 수를 지정합니다.



참고

podsWithCore를 0으로 설정하면 이 제한이 비활성화됩니다.

위의 예에서 **podsWithCore**의 기본값은 10이며 **maxPods**의 기본값은 250입니다. 즉, 노드에 25 개 이상의 코어가 없으면 기본적으로 **podsWithCore**가 제한 요소가 됩니다.

2.

다음 명령을 실행하여 CR을 생성합니다.

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

검증

1.

MachineConfigPool CRD를 나열하여 변경 사항이 적용되는지 확인합니다. **Machine Config Controller**에서 변경 사항을 선택하면 **UPDATING** 열에 **True**가 보고됩니다.

```
$ oc get machineconfigpools
```

출력 예

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	False	False
worker	worker-8cecd1236b33ee3f8a5e	False	True	False

변경이 완료되면 **UPDATED** 열에 **True**가 보고됩니다.

```
$ oc get machineconfigpools
```

출력 예

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	True	False
worker	worker-8cecd1236b33ee3f8a5e	True	False	False

6.5. NODE TUNING OPERATOR 사용

Node Tuning Operator에 대해 알아보고, **Node Tuning Operator**를 사용하여 **Tuned** 데몬을 오케스트레이션하고 노드 수준 튜닝을 관리하는 방법도 알아봅니다.

Node Tuning Operator는 **TuneD** 데몬을 오케스트레이션하여 노드 수준 튜닝을 관리하고 **Performance Profile** 컨트롤러를 사용하여 짧은 대기 시간 성능을 달성하는 데 도움이 됩니다. 대부분의 고성능 애플리케이션에는 일정 수준의 커널 튜닝이 필요합니다. **Node Tuning Operator**는 노드 수준 **sysctls** 사용자에게 통합 관리 인터페이스를 제공하며 사용자의 필요에 따라 지정되는 사용자 정의 튜닝을 추가할 수 있는 유연성을 제공합니다.

Operator는 **OpenShift Container Platform**의 컨테이너화된 **TuneD** 데몬을 **Kubernetes** 데몬 세트 관리합니다. 클러스터에서 실행되는 모든 컨테이너화된 **TuneD** 데몬에 사용자 정의 튜닝 사양이 데몬이 이해할 수 있는 형식으로 전달되도록 합니다. 데몬은 클러스터의 모든 노드에서 노드당 하나씩 실행됩니다.

컨테이너화된 **TuneD** 데몬을 통해 적용되는 노드 수준 설정은 프로필 변경을 트리거하는 이벤트 시 또는 컨테이너화된 **TuneD** 데몬이 종료 신호를 수신하고 처리하여 정상적으로 종료될 때 롤백됩니다.

Node Tuning Operator는 **Performance Profile** 컨트롤러를 사용하여 자동 튜닝을 구현하여 **OpenShift Container Platform** 애플리케이션에 대해 짧은 대기 시간 성능을 실현합니다. 클러스터 관리자는 다음과 같은 노드 수준 설정을 정의하도록 성능 프로필을 구성합니다.

- 커널을 **kernel-rt**로 업데이트
- 하우스키핑을 위해 **CPU**를 선택합니다.
- 워크로드 실행을 위한 **CPU** 선택.



참고

현재 **cgroup v2**에서는 **CPU** 부하 분산을 비활성화하지 않습니다. 따라서 **cgroup v2**가 활성화된 경우 성능 프로필에서 원하는 동작을 얻지 못할 수 있습니다. 성능 프로필을 사용하는 경우에는 **cgroup v2**를 활성화하는 것은 권장되지 않습니다.

버전 4.1 이상에서는 **Node Tuning Operator**가 표준 **OpenShift Container Platform** 설치에 포함되어 있습니다.



참고

이전 버전의 **OpenShift Container Platform**에서는 **OpenShift** 애플리케이션에 대해 짧은 대기 시간 성능을 실현하기 위해 자동 튜닝을 구현하는 데 **Performance Addon Operator**를 사용했습니다. **OpenShift Container Platform 4.11** 이상에서는 이 기능은 **Node Tuning Operator**의 일부입니다.

6.5.1. Node Tuning Operator 사양 예에 액세스

이 프로세스를 사용하여 **Node Tuning Operator** 사양 예에 액세스하십시오.

절차

- 다음 명령을 실행하여 **Node Tuning Operator** 사양 예에 액세스합니다.

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

기본 CR은 OpenShift Container Platform 플랫폼의 표준 노드 수준 튜닝을 제공하기 위한 것이며 Operator 관리 상태를 설정하는 경우에만 수정할 수 있습니다. Operator는 기본 CR에 대한 다른 모든 사용자 정의 변경사항을 덮어씁니다. 사용자 정의 튜닝의 경우 고유한 Tuned CR을 생성합니다. 새로 생성된 CR은 노드 또는 Pod 라벨 및 프로필 우선 순위에 따라 OpenShift Container Platform 노드에 적용된 기본 CR 및 사용자 정의 튜닝과 결합됩니다.



주의

특정 상황에서는 Pod 라벨에 대한 지원이 필요한 튜닝을 자동으로 제공하는 편리한 방법일 수 있지만 이러한 방법은 권장되지 않으며 특히 대규모 클러스터에서는 이러한 방법을 사용하지 않는 것이 좋습니다. 기본 Tuned CR은 Pod 라벨이 일치되지 않은 상태로 제공됩니다. Pod 라벨이 일치된 상태로 사용자 정의 프로필이 생성되면 해당 시점에 이 기능이 활성화됩니다. Pod 라벨 기능은 Node Tuning Operator의 향후 버전에서 더 이상 사용되지 않습니다.

6.5.2. 사용자 정의 튜닝 사양

Operator의 CR(사용자 정의 리소스)에는 두 가지 주요 섹션이 있습니다. 첫 번째 섹션인 **profile:**은 TuneD 프로필 및 해당 이름의 목록입니다. 두 번째인 **recommend:**은 프로필 선택 논리를 정의합니다.

여러 사용자 정의 튜닝 사양은 Operator의 네임스페이스에 여러 CR로 존재할 수 있습니다. 새로운 CR의 존재 또는 오래된 CR의 삭제는 Operator에서 탐지됩니다. 기존의 모든 사용자 정의 튜닝 사양이 병합되고 컨테이너화된 TuneD 데몬의 해당 오브젝트가 업데이트됩니다.

관리 상태

Operator 관리 상태는 기본 Tuned CR을 조정하여 설정됩니다. 기본적으로 Operator는 **Managed** 상태이며 기본 Tuned CR에는 **spec.managementState** 필드가 없습니다. Operator 관리 상태에 유효한 값은 다음과 같습니다.

- **Managed:** 구성 리소스가 업데이트되면 Operator가 해당 피연산자를 업데이트합니다.
- **Unmanaged:** Operator가 구성 리소스에 대한 변경을 무시합니다.

- **Removed:** Operator가 프로비저닝한 해당 피연산자 및 리소스를 Operator가 제거합니다.

프로필 데이터

profile: 섹션에는 TuneD 프로필 및 해당 이름이 나열됩니다.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

권장 프로필

profile: 선택 논리는 CR의 **recommend:** 섹션에 의해 정의됩니다. **recommend:** 섹션은 선택 기준에 따라 프로필을 권장하는 항목의 목록입니다.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

목록의 개별 항목은 다음과 같습니다.

```
- machineConfigLabels: ①
  <mcLabels> ②
  match: ③
  <match> ④
```



```

priority: <priority> 5
profile: <tuned_profile_name> 6
operand: 7
debug: <bool> 8
tunedConfig:
  reapply_sysctl: <bool> 9

```

1

선택 사항입니다.

2

키/값 **MachineConfig** 라벨 사전입니다. 키는 고유해야 합니다.

3

생략하면 우선 순위가 높은 프로필이 먼저 일치되거나 **machineConfigLabels**가 설정되어 있지 않으면 프로필이 일치하는 것으로 가정합니다.

4

선택사항 목록입니다.

5

프로필 순서 지정 우선 순위입니다. 숫자가 작을수록 우선 순위가 높습니다(0이 가장 높은 우선 순위임).

6

일치에 적용할 **TuneD** 프로필입니다. 예를 들어 **tuned_profile_1**이 있습니다.

7

선택적 피연산자 구성입니다.

8

TuneD 데몬의 디버깅을 켜거나 비활성화합니다. **off**의 경우 **on** 또는 **false**의 경우 옵션이 **true**입니다. 기본값은 **false**입니다.

9

TuneD 데몬에 대해 **reapply_sysctl** 기능을 켜거나 **off**합니다. **off**의 경우 **on** 및 **false**의 경우 옵션이 **true**입니다.

`<match>`는 다음과 같이 재귀적으로 정의되는 선택사항 목록입니다.

```
- label: <label_name> ①
  value: <label_value> ②
  type: <label_type> ③
  <match> ④
```

1

노드 또는 Pod 라벨 이름입니다.

2

선택사항 노드 또는 Pod 라벨 값입니다. 생략하면 `<label_name>`이 있기 때문에 일치 조건을 충족합니다.

3

선택사항 오브젝트 유형(`node` 또는 `pod`)입니다. 생략하면 `node`라고 가정합니다.

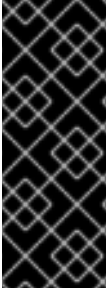
4

선택사항 `<match>` 목록입니다.

`<match>`를 생략하지 않으면 모든 중첩 `<match>` 섹션도 `true`로 평가되어야 합니다. 생략하면 `false`로 가정하고 해당 `<match>` 섹션이 있는 프로필을 적용하지 않거나 권장하지 않습니다. 따라서 중첩(하위 `<match>` 섹션)은 논리 AND 연산자 역할을 합니다. 반대로 `<match>` 목록의 항목이 일치하면 전체 `<match>` 목록이 `true`로 평가됩니다. 따라서 이 목록이 논리 OR 연산자 역할을 합니다.

`machineConfigLabels`가 정의되면 지정된 `recommend:` 목록 항목에 대해 머신 구성 풀 기반 일치가 설정됩니다. `<mcLabels>`는 머신 구성의 라벨을 지정합니다. 머신 구성은 `<tuned_profile_name>` 프로필에 대해 커널 부팅 매개변수와 같은 호스트 설정을 적용하기 위해 자동으로 생성됩니다. 여기에는 `<mcLabels>`와 일치하는 머신 구성 선택기가 있는 모든 머신 구성 풀을 찾고 머신 구성 풀이 할당된 모든 노드에서 `<tuned_profile_name>` 프로필을 설정하는 작업이 포함됩니다. 마스터 및 작업자 역할이 모두 있는 노드를 대상으로 하려면 마스터 역할을 사용해야 합니다.

목록 항목 `match` 및 `machineConfigLabels`는 논리 OR 연산자로 연결됩니다. `match` 항목은 단락 방식으로 먼저 평가됩니다. 따라서 `true`로 평가되면 `machineConfigLabels` 항목이 고려되지 않습니다.



중요

머신 구성 풀 기반 일치를 사용하는 경우 동일한 하드웨어 구성을 가진 노드를 동일한 머신 구성 풀로 그룹화하는 것이 좋습니다. 이 방법을 따르지 않으면 TuneD 피연산자가 동일한 머신 구성 풀을 공유하는 두 개 이상의 노드에 대해 충돌하는 커널 매개변수를 계산할 수 있습니다.

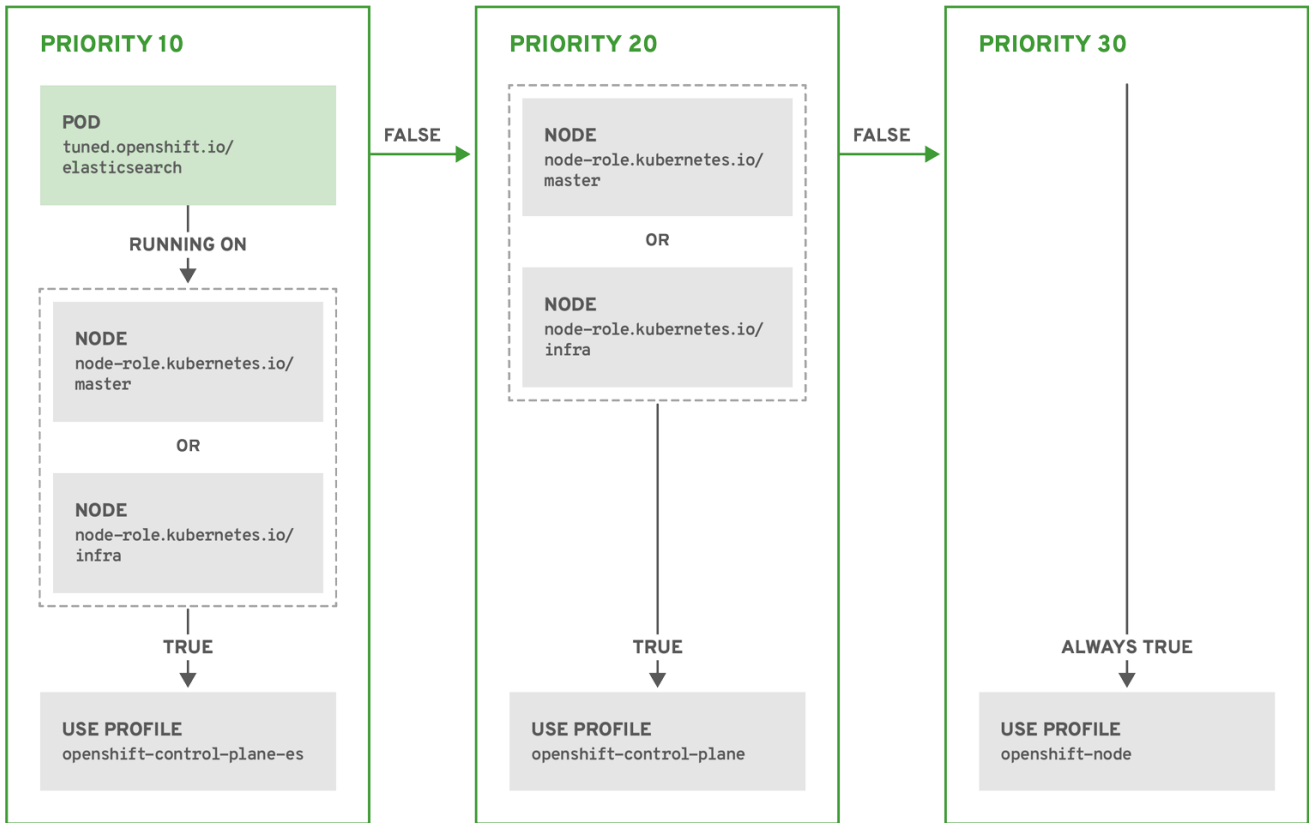
예: 노드 또는 Pod 라벨 기반 일치

```
- match:
- label: tuned.openshift.io/elasticsearch
  match:
- label: node-role.kubernetes.io/master
- label: node-role.kubernetes.io/infra
  type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
- label: node-role.kubernetes.io/master
- label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

위의 CR은 컨테이너화된 TuneD 데몬의 프로필 우선 순위에 따라 `recommended.conf` 파일로 변환됩니다. 우선 순위가 가장 높은 프로필(10)이 `openshift-control-plane-es`이므로 이 프로필을 첫 번째로 고려합니다. 지정된 노드에서 실행되는 컨테이너화된 TuneD 데몬은 `tuned.openshift.io/elasticsearch` 라벨이 설정된 동일한 노드에서 실행되는 Pod가 있는지 확인합니다. 없는 경우 전체 `<match>` 섹션이 `false`로 평가됩니다. 라벨이 있는 Pod가 있는 경우 `<match>` 섹션을 `true`로 평가하려면 노드 라벨도 `node-role.kubernetes.io/master` 또는 `node-role.kubernetes.io/infra`여야 합니다.

우선 순위가 10인 프로필의 라벨이 일치하면 `openshift-control-plane-es` 프로필이 적용되고 다른 프로필은 고려되지 않습니다. 노드/Pod 라벨 조합이 일치하지 않으면 두 번째로 높은 우선 순위 프로필(`openshift-control-plane`)이 고려됩니다. 컨테이너화된 TuneD Pod가 `node-role.kubernetes.io/master` 또는 `node-role.kubernetes.io/infra` 라벨이 있는 노드에서 실행되는 경우 이 프로필이 적용됩니다.

마지막으로, `openshift-node` 프로필은 우선 순위가 가장 낮은 30입니다. 이 프로필에는 `<match>` 섹션이 없으므로 항상 일치합니다. 지정된 노드에서 우선 순위가 더 높은 다른 프로필이 일치하지 않는 경우 `openshift-node` 프로필을 설정하는 데 `catch-all` 프로필 역할을 합니다.



OPENSIFT_10_0319

예: 머신 구성 플 기반 일치

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
    [main]
    summary=Custom OpenShift node profile with an additional kernel parameter
    include=openshift-node
    [bootloader]
    cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

  recommend:
  - machineConfigLabels:
    machineconfiguration.openshift.io/role: "worker-custom"
    priority: 20
    profile: openshift-node-custom
    
```

노드 재부팅을 최소화하려면 머신 구성 풀의 노드 선택기와 일치하는 라벨로 대상 노드에 라벨을 지정한 후 위의 **Tuned CR**을 생성하고 마지막으로 사용자 정의 머신 구성 풀을 생성합니다.

클라우드 공급자별 Tuned 프로필

이 기능을 사용하면 모든 클라우드 공급자별 노드에 **OpenShift Container Platform** 클러스터의 지정된 클라우드 공급자에 맞게 특별히 조정된 **Tuned** 프로필이 편리하게 할당될 수 있습니다. 이 작업은 노드 레이블을 추가하거나 머신 구성 풀에 노드를 그룹화하지 않고 수행할 수 있습니다.

이 기능은 `<cloud-provider>://<cloud-provider-specific-id>` 형식의 **spec.provider ID** 노드 오브젝트 값을 사용하고 **NTO** 퍼연산자 컨테이너에서 `<cloud-provider>` 값으로 `/var/lib/tuned/provider` 파일을 씁니다. 그런 다음 **Tuned**에서 해당 프로필이 있는 경우 `provider-<cloud-provider>` 프로필을 로드하는 데 이 파일의 콘텐츠를 사용합니다.

openshift-control-plane 및 **openshift-node** 프로필 모두의 설정을 상속하는 **openshift** 프로필은 조건부 프로필 로드를 사용하여 이 기능을 사용하도록 업데이트되었습니다. **NTO**와 **Tuned**는 현재 클라우드 공급자별 프로필을 제공하지 않습니다. 그러나 모든 클라우드 공급자별 클러스터 노드에 적용할 사용자 지정 프로필 `provider-<cloud-provider>`를 생성할 수 있습니다.

GCE 클라우드 공급자 프로필의 예

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
      name: provider-gce
```



참고

프로필 상속으로 인해 **provider-< cloud-provider > 프로필에 지정된 모든 설정은 openshift 프로필 및 해당 하위 프로필에서 덮어씁니다.**

6.5.3. 클러스터에 설정된 기본 프로필

다음은 클러스터에 설정된 기본 프로필입니다.

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider- $\{f:exec:cat:/var/lib/tuned/provider\}$ ,openshift
      name: openshift
    recommend:
    - profile: openshift-control-plane
      priority: 30
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    - profile: openshift-node
      priority: 40
  
```

OpenShift Container Platform 4.9부터 모든 OpenShift TuneD 프로필이 TuneD 패키지와 함께 제공됩니다. **oc exec** 명령을 사용하여 이러한 프로필의 내용을 볼 수 있습니다.

```

$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find
/usr/lib/tuned/openshift $\{-control-plane,-node\}$  -name tuned.conf -exec grep -H ^ {} \;
  
```

6.5.4. 지원되는 TuneD 데몬 플러그인

Tuned CR의 **profile:** 섹션에 정의된 사용자 정의 프로필을 사용하는 경우 **[main]** 섹션을 제외한 다음 TuneD 플러그인이 지원됩니다.

- **audio**

- *cpu*
- *disk*
- *eeepc_she*
- *modules*
- *mounts*
- *net*
- *scheduler*
- *scsi_host*
- *selinux*
- *sysctl*
- *sysfs*
- *usb*
- *video*
- *vm*

- **bootloader**

일부 플러그인에서 제공하는 동적 튜닝 기능은 지원되지 않습니다. 다음 **TuneD** 플러그인은 현재 지원되지 않습니다.

- **script**

- **systemd**



참고

TuneD 부트로더 플러그인은 **RHCOS(Red Hat Enterprise Linux CoreOS)** 작업자 노드만 지원합니다.

추가 리소스

- [사용 가능한 TuneD 플러그인](#)
- [TuneD 시작하기](#)

6.6. 노드 수정, 펜싱 및 유지 관리

커널 중단 또는 **NIC(네트워크 인터페이스 컨트롤러)**와 같은 노드 수준 오류가 발생하면 클러스터에서 필요한 작업이 감소되지 않으며 영향을 받는 노드의 워크로드를 다른 곳에서 다시 시작해야 합니다. 이러한 워크로드에 영향을 미치는 경우 데이터 손실, 손상 또는 둘 다 위험합니다. 워크로드 복구(수정 이라고 함) 및 노드 복구를 시작하기 전에 펜싱이라는 노드를 격리하는 것이 중요합니다.

노드 수정, 펜싱 및 유지 관리에 대한 자세한 내용은 [Workload Availability for Red Hat OpenShift](#) 설명서를 참조하십시오.

6.7. 노드 재부팅 이해

플랫폼에서 실행 중인 애플리케이션을 중단하지 않고 노드를 재부팅하려면 먼저 **Pod**를 비워야 합니다. 라우팅 계층에서 가용성이 높은 **Pod**의 경우 다른 작업을 수행할 필요가 없습니다. 스토리지(일반적으로 데이터베이스)가 필요한 기타 **Pod**의 경우 특정 **Pod**가 일시적으로 오프라인으로 전환된 상태에서도 계속 작동하는지 확인하는 것이 중요합니다. 상태 저장 **Pod**에 대한 복원력을 구현하는 방법은 애플리케이션마

다 다르지만 어떠한 경우에도 노드 유사성 방지를 사용하여 Pod가 사용 가능한 노드에 적절히 분배되도록 스케줄러를 구성하는 것이 중요합니다.

또 다른 문제는 라우터 또는 레지스트리와 같은 중요한 인프라를 실행하는 노드를 처리하는 방법입니다. 동일한 노드 비우기 프로세스가 적용되지만 특정 엣지 케이스를 이해하는 것이 중요합니다.

6.7.1. 중요한 인프라를 실행하는 노드 재부팅 정보

라우터 Pod, 레지스트리 Pod, 모니터링 Pod와 같은 중요한 OpenShift Container Platform 인프라 구성 요소를 호스팅하는 노드를 재부팅할 때는 이러한 구성 요소를 실행하는 데 사용 가능한 노드가 세 개 이상 있는지 확인하십시오.

다음 시나리오에서는 두 개의 노드만 사용할 수 있을 때 OpenShift Container Platform에서 실행되는 애플리케이션에서 서비스 중단이 발생하는 방식을 보여줍니다.

- 노드 A가 예약 불가로 표시되고 모든 Pod가 비어 있습니다.
- 이제 해당 노드에서 실행 중인 레지스트리 Pod가 노드 B에 다시 배포됩니다. 그러면 노드 B는 두 레지스트리 Pod를 모두 실행합니다.
- 이제 노드 B가 예약 불가로 표시되고 비어 있습니다.
- 노드 B에 Pod 끝점 두 개를 노출하는 서비스에서는 해당 끝점이 노드 A에 다시 배포될 때까지 잠시 모든 끝점이 손실됩니다.

인프라 구성 요소로 노드 세 개를 사용하는 경우 이 프로세스에서는 서비스가 중단되지 않습니다. 그러나 Pod 예약으로 인해 비워진 후 다시 제공된 마지막 노드에는 레지스트리 Pod가 없습니다. 기타 노드 중 하나에는 레지스트리 Pod가 두 개 있습니다. 마지막 노드에 세 번째 레지스트리 Pod를 예약하려면 Pod 유사성 방지를 사용하여 스케줄러에서 동일한 노드에 두 레지스트리 Pod를 배치하지 않도록 합니다.

추가 정보

- Pod 유사성 방지에 대한 자세한 내용은 유사성 및 유사성 방지 규칙을 사용하여 다른 Pod를 기준으로 Pod 배치를 참조하십시오.

6.7.2. Pod 유사성 방지를 사용하여 노드 재부팅

Pod 유사성 방지는 노드 유사성 방지와 약간 다릅니다. **Pod**를 배포할 다른 적절한 위치가 없는 경우 노드 유사성 방지를 위반할 수 있습니다. **Pod** 유사성 방지를 필수 또는 기본으로 설정할 수 있습니다.

이 규칙에서는 두 개의 인프라 노드만 사용할 수 있는 경우 한 개를 재부팅하면 컨테이너 이미지 레지스트리 **Pod**가 다른 노드에서 실행되지 않습니다. **oc get pods**는 적절한 노드가 제공될 때까지 **Pod**를 준비되지 않은 것으로 보고합니다. 노드를 사용할 수 있고 모든 **Pod**가 준비 상태가 되면 다음 노드를 재시작할 수 있습니다.

프로세스

Pod 유사성 방지를 사용하여 노드를 재부팅하려면 다음을 수행합니다.

1. 노드 사양을 편집하여 **Pod** 유사성 방지를 구성합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
    preferredDuringSchedulingIgnoredDuringExecution: 2
    - weight: 100 3
    podAffinityTerm:
      labelSelector:
        matchExpressions:
          - key: registry 4
            operator: In 5
            values:
              - default
      topologyKey: kubernetes.io/hostname
  #...

```

1 **Pod** 유사성 방지를 구성하는 스태ن자입니다.

2 기본 규칙을 정의합니다.

3 기본 규칙의 가중치를 지정합니다. 가중치가 가장 높은 노드가 우선합니다.

4

유사성 방지 규칙이 적용되는 시기를 결정하는 **Pod** 라벨에 대한 설명입니다. 라벨의 키와 값을 지정합니다.

5

이 연산자는 기존 **Pod**의 라벨과 새 **Pod** 사양에 있는 **matchExpression** 매개변수의 값 집합 간의 관계를 나타냅니다. **In**, **NotIn**, **Exists** 또는 **DoesNotExist**일 수 있습니다.

이 예제에서는 컨테이너 이미지 레지스트리 **Pod**에 **registry=default** 라벨이 있다고 가정합니다. **Pod** 유사성 방지에서는 모든 **Kubernetes** 일치 표현식을 사용할 수 있습니다.

2.

예약 정책 파일에서 **MatchInterPodAffinity** 스케줄러 서술자를 활성화합니다.

3.

노드를 정상 재시작합니다.

6.7.3. 라우터를 실행하는 노드를 재부팅하는 방법 이해

대부분의 경우 **OpenShift Container Platform** 라우터를 실행하는 **Pod**에서는 호스트 포트를 노출합니다.

PodFitsPorts 스케줄러 서술자를 사용하면 동일한 포트를 사용하는 라우터 **Pod**가 동일한 노드에서 실행되지 않고 **Pod** 유사성 방지를 구현할 수 있습니다. 라우터에서 고가용성을 위해 **IP** 장애 조치를 사용하는 경우 추가로 필요한 조치는 없습니다.

고가용성을 위해 **AWS Elastic Load Balancing**과 같은 외부 서비스를 사용하는 라우터 **Pod**의 경우 해당 서비스에서 라우터 **Pod** 재시작에 대응해야 합니다.

드물지만 라우터 **Pod**에 호스트 포트가 구성되어 있지 않은 경우가 있습니다. 이러한 경우 인프라 노드에 권장되는 재시작 프로세스를 따라야 합니다.

6.7.4. 노드를 정상적으로 재부팅

노드를 재부팅하기 전에 노드에서 데이터 손실을 방지하려면 **etcd** 데이터를 백업하는 것이 좋습니다.



참고

사용자가 클러스터를 관리하기 위해 **kubeconfig** 파일에 인증서를 두지 않고 **oc login** 명령을 수행해야 하는 단일 노드 OpenShift 클러스터의 경우, 노드를 차단하고 드레이닝 한 후 **oc adm** 명령을 사용할 수 없습니다. 이는 **openshift-oauth-apiserver** 포드가 **cordons**으로 인해 실행되지 않기 때문입니다. 다음 절차에 표시된 대로 **SSH**를 사용하여 노드에 액세스할 수 있습니다.

단일 노드 OpenShift 클러스터에서는 차단 및 드레이닝 시 **Pod**를 다시 예약할 수 없습니다. 그러나 이렇게 하면 특히 워크로드 **Pod**와 **Pod**가 적절하게 중지되고 관련 리소스를 해제할 수 있습니다.

절차

노드의 정상 재시작을 수행하려면 다음을 수행합니다.

1. 노드를 예약 불가능으로 표시합니다.

```
$ oc adm cordon <node1>
```

2. 노드를 드레이닝하여 실행 중인 모든 **Pod**를 삭제합니다.

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force
```

사용자 정의 **PDB**(**Pod** 중단 예산)와 연결된 **Pod**를 제거할 수 없는 오류가 표시될 수 있습니다.

오류 예

```
error when evicting pods/"rails-postgresql-example-1-72v2w" -n "rails" (will retry after 5s): Cannot evict pod as it would violate the pod's disruption budget.
```

이 경우 **drain** 명령을 다시 실행하여 **PDB** 검사를 우회하는 **disable-eviction** 플래그를 추가합니다.

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force --disable-
eviction
```

3. 디버그 모드에서 노드에 액세스합니다.

```
$ oc debug node/<node1>
```

4. 루트 디렉토리를 `/host` 로 변경합니다.

```
$ chroot /host
```

5. 노드를 재시작합니다.

```
$ systemctl reboot
```

잠시 후 노드가 **NotReady** 상태가 됩니다.

참고

일부 단일 노드 OpenShift 클러스터에서는 `openshift-oauth-apiserver` 포트가 실행되지 않기 때문에 차단 및 노드 드레이닝 후 `oc` 명령을 사용할 수 없습니다. SSH를 사용하여 노드에 연결하고 재부팅을 수행할 수 있습니다.

```
$ ssh core@<master-node>.<cluster_name>.<base_domain>
```

```
$ sudo systemctl reboot
```

6. 재부팅이 완료되면 다음 명령을 실행하여 노드를 예약 가능으로 표시합니다.

```
$ oc adm uncordon <node1>
```



참고

일부 단일 노드 OpenShift 클러스터에서는 `openshift-oauth-apiserver` 포트가 실행되지 않기 때문에 차단 및 노드 트레이닝 후 `oc` 명령을 사용할 수 없습니다. SSH를 사용하여 노드에 연결하고 연결을 해제할 수 있습니다.

```
$ ssh core@<target_node>
```

```
$ sudo oc adm uncordon <node> --kubeconfig /etc/kubernetes/static-pod-resources/kube-apiserver-certs/secrets/node-kubeconfigs/localhost.kubeconfig
```

7.

노드가 준비되었는지 확인합니다.

```
$ oc get node <node1>
```

출력 예

```
NAME STATUS ROLES AGE VERSION
<node1> Ready worker 6d22h v1.18.3+b0068a8
```

추가 정보

`etcd` 데이터 백업에 대한 자세한 내용은 [etcd 데이터 백업](#)을 참조하십시오.

6.8. 가비지 컬렉션을 사용하여 노드 리소스 해제

관리자는 OpenShift Container Platform에서 가비지 컬렉션을 통해 리소스를 확보함으로써 노드가 효율적으로 실행되도록 할 수 있습니다.

OpenShift Container Platform 노드는 두 가지 유형의 가비지 컬렉션을 수행합니다.

-

컨테이너 가비지 컬렉션: 종료된 컨테이너를 제거합니다.

- 이미지 가비지 컬렉션: 실행 중인 Pod에서 참조하지 않는 이미지를 제거합니다.

6.8.1. 가비지 컬렉션을 통해 종료된 컨테이너를 제거하는 방법 이해

컨테이너 가비지 컬렉션에서는 제거 임계 값을 사용하여 종료된 컨테이너를 제거합니다.

가비지 컬렉션에 제거 임계 값이 설정되어 있으면 노드는 API에서 액세스 가능한 모든 pod의 컨테이너를 유지하려고 합니다. pod가 삭제된 경우 컨테이너도 삭제됩니다. pod가 삭제되지 않고 제거 임계 값에 도달하지 않는 한 컨테이너는 보존됩니다. 노드가 디스크 부족 (disk pressure) 상태가 되면 컨테이너가 삭제되고 oc logs를 사용하여 해당 로그에 더 이상 액세스할 수 없습니다.

- **eviction-soft** - 소프트 제거 임계 값은 관리자가 지정한 필수 유예 기간이 있는 제거 임계 값과 일치합니다.
- **eviction-hard** - 하드 제거 임계 값에 대한 유예 기간이 없으며 감지되는 경우 OpenShift Container Platform은 즉시 작업을 수행합니다.

다음 표에는 제거 임계 값이 나열되어 있습니다.

표 6.2. 컨테이너 가비지 컬렉션 구성을 위한 변수

노드 상태	제거 신호	설명
MemoryPressure	memory.available	노드에서 사용 가능한 메모리입니다.
DiskPressure	<ul style="list-style-type: none"> ● nodefs.available ● nodefs.inodesFree ● imagefs.available ● imagefs.inodesFree 	노드 루트 파일 시스템, nodefs 또는 이미지 파일 시스템, imagefs 에서 사용 가능한 디스크 공간 또는 inode.



참고

evictionHard의 경우 이러한 모든 매개변수를 지정해야 합니다. 모든 매개변수를 지정하지 않으면 지정된 매개변수만 적용되고 가비지 컬렉션이 제대로 작동하지 않습니다.

노드가 소프트웨어 제거 임계 값 상한과 하한 사이에서 변동하고 연관된 유예 기간이 만료되지 않은 경우 해당 노드는 지속적으로 **true** 와 **false** 사이에서 변동합니다. 결과적으로 스케줄러는 잘못된 스케줄링 결정을 내릴 수 있습니다.

이러한 변동을 방지하려면 **eviction-pressure-transition-period** 플래그를 사용하여 **OpenShift Container Platform**이 부족 상태에서 전환하기 전에 대기해야 하는 시간을 제어합니다. **OpenShift Container Platform**은 **false** 상태로 전환되기 전에 지정된 기간에 지정된 부족 상태에 대해 제거 임계 값을 충족하도록 설정하지 않습니다.

6.8.2. 가비지 컬렉션을 통해 이미지를 제거하는 방법 이해

이미지 가비지 컬렉션에서는 실행 중인 **Pod**에서 참조하지 않는 이미지를 제거합니다.

OpenShift Container Platform은 **cAdvisor** 에서 보고하는 디스크 사용량을 기반으로 노드에서 삭제할 이미지를 결정합니다.

이미지 가비지 컬렉션 정책은 다음 두 가지 조건을 기반으로 합니다.

- 이미지 가비지 컬렉션을 트리거하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 85입니다.
- 이미지 가비지 컬렉션이 해제하려고 하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 80입니다.

이미지 가비지 컬렉션의 경우 사용자 지정 리소스를 사용하여 다음 변수를 수정할 수 있습니다.

표 6.3. 이미지 가비지 컬렉션 구성을 위한 변수

설정	설명
imageMinimumGauge	가비지 컬렉션에 의해 이미지가 제거되기 전에 사용되지 않은 이미지의 최소 보존 기간입니다. 기본값은 2m입니다.
imageGCHighThresholdPercent	이미지 가비지 컬렉션을 트리거하는 정수로 표시되는 디스크 사용량의 백분율입니다. 기본값은 85입니다.
imageGCLowThresholdPercent	이미지 가비지 컬렉션이 해제하려고 하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 80입니다.

각 가비지 컬렉터 실행으로 두 개의 이미지 목록이 검색됩니다.

1. 하나 이상의 **Pod**에서 현재 실행 중인 이미지 목록입니다.
2. 호스트에서 사용 가능한 이미지 목록입니다.

새로운 컨테이너가 실행되면 새로운 이미지가 나타납니다. 모든 이미지에는 타임 스탬프가 표시됩니다. 이미지가 실행 중이거나 (위의 첫 번째 목록) 새로 감지된 경우 (위의 두 번째 목록) 현재 시간으로 표시됩니다. 나머지 이미지는 이미 이전 실행에서 표시됩니다. 모든 이미지는 타임 스탬프별로 정렬됩니다.

컬렉션이 시작되면 중지 기준이 충족될 때까지 가장 오래된 이미지가 먼저 삭제됩니다.

6.8.3. 컨테이너 및 이미지의 가비지 컬렉션 구성

관리자는 각 **machine config pool**마다 **kubeletConfig** 오브젝트를 생성하여 **OpenShift Container Platform**이 가비지 컬렉션을 수행하는 방법을 구성할 수 있습니다.



참고

OpenShift Container Platform은 각 머신 구성 풀에 대해 하나의 **kubeletConfig** 오브젝트만 지원합니다.

다음 중 하나의 조합을 구성할 수 있습니다.

- 소프트웨어 컨테이너 제거
- 컨테이너 하드 제거
- 이미지 제거

컨테이너 가비지 컬렉션에서는 종료된 컨테이너를 제거합니다. 이미지 가비지 컬렉션에서는 실행 중인 **Pod**에서 참조하지 않는 이미지를 제거합니다.

사전 요구 사항

1.

다음 명령을 입력하여 구성할 노드 유형의 정적 **MachineConfigPool CRD**와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

1

레이블이 **Labels** 아래에 표시됩니다.

작은 정보

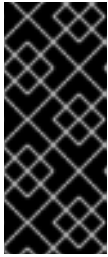
라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

절차

1.

구성 변경을 위한 사용자 정의 리소스 (CR)를 만듭니다.



중요

파일 시스템이 하나 있거나 `/var/lib/kubelet` 및 `/var/lib/containers/`가 동일한 파일 시스템에 있는 경우 가장 높은 값의 트리거 제거 설정이 먼저 충족됩니다. 파일 시스템이 제거를 트리거합니다.

컨테이너 가비지 컬렉션 CR의 설정 예:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: worker-kubeconfig 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    evictionSoft: 3
      memory.available: "500Mi" 4
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: 5
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard: 6
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s 7
    imageMinimumGCAge: 5m 8
    imageGCHighThresholdPercent: 80 9
    imageGCLowThresholdPercent: 75 10
#...

```

1

오브젝트의 이름입니다.

2

머신 구성 풀에서 라벨을 지정합니다.

3

컨테이너 가비지 컬렉션의 경우 **eviction: evictionSoft** 또는 **evictionHard** 입니다.

4

컨테이너 가비지 컬렉션의 경우: 특정 제거 트리거 신호를 기반으로 하는 제거 임계값입니다.

5

컨테이너 가비지 컬렉션의 경우: 소프트 제거에 대한 **grace** 기간입니다. 이 매개변수는 **eviction-hard**에는 적용되지 않습니다.

6

컨테이너 가비지 컬렉션의 경우: 특정 제거 트리거 신호를 기반으로 하는 제거 임계값입니다. **evictionHard**의 경우 이러한 모든 매개변수를 지정해야 합니다. 모든 매개변수를 지정하지 않으면 지정된 매개변수만 적용되고 가비지 컬렉션이 제대로 작동하지 않습니다.

7

컨테이너 가비지 컬렉션의 경우: 제거 부족 상태에서 전환되기 전에 대기하는 시간입니다.

8

이미지 가비지 컬렉션의 경우 이미지 가비지 수집에 의해 이미지가 제거되기 전에 사용되지 않은 이미지의 최소 수명입니다.

9

이미지 가비지 컬렉션의 경우: 이미지 가비지 컬렉션을 트리거하는 디스크 사용량의 백분율(정수로 표시)입니다.

10

이미지 가비지 컬렉션의 경우 이미지 가비지 컬렉션이 해제하려고 하는 디스크 사용량의 백분율(정수로 표시)입니다.

2.

다음 명령을 실행하여 **CR**을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f gc-container.yaml
```

출력 예

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

검증

1.

다음 명령을 입력하여 가비지 컬렉션이 활성화되어 있는지 확인합니다. 사용자 지정 리소스에 지정한 **Machine Config Pool**은 변경 사항이 완전히 구현될 때까지 **UPDATING**과 함께 **'true'**로 표시됩니다.

```
$ oc get machineconfigpool
```

출력 예

```
NAME CONFIG UPDATED UPDATING
master rendered-master-546383f80705bd5aeaba93 True False
worker rendered-worker-b4c51bb33ccaae6fc4a6a5 False True
```

6.9. OPENSIFT CONTAINER PLATFORM 클러스터의 노드에 리소스 할당

더 안정적인 예약 기능을 제공하고 노드 리소스 과다 할당을 최소화하려면 기본 노드 구성 요소(예: **kubelet**, **kube-proxy**) 및 나머지 시스템 구성 요소(예: **sshd**, **NetworkManager**)에서 사용할 **CPU** 및 메모리 리소스의 일부를 예약하십시오. 예약할 리소스를 지정하면 **Pod**에서 사용할 수 있는 노드의 나머지 **CPU** 및 메모리 리소스에 대한 세부 정보가 스케줄러에 제공됩니다. **OpenShift Container Platform**에서

노드에 대해 최적의 **system-reserved CPU** 및 메모리 리소스를 자동으로 결정하도록 허용하거나 수동으로 노드에 가장 적합한 리소스를 결정하고 설정할 수 있습니다.



중요

리소스 값을 수동으로 설정하려면 **kubelet** 구성 **CR**을 사용해야 합니다. 머신 구성 **CR**을 사용할 수 없습니다.

6.9.1. 노드에 리소스를 할당하는 방법 이해

OpenShift Container Platform에서 노드 구성 요소용으로 예약된 **CPU** 및 메모리 리소스는 다음 두 노드 설정을 기반으로 합니다.

설정	설명
kube-reserved	이 설정은 OpenShift Container Platform과 함께 사용되지 않습니다. system-reserved 설정에 예약할 CPU 및 메모리 리소스를 추가합니다.
system-reserved	이 설정은 노드 구성 요소 및 시스템 구성 요소(예: CRI-O 및 Kubelet)에 예약할 리소스를 식별합니다. 기본 설정은 OpenShift Container Platform 및 Machine Config Operator 버전에 따라 다릅니다. machine-config-operator 리포지토리에서 기본 systemReserved 매개변수를 확인합니다.

플래그를 설정하지 않으면 기본값이 사용됩니다. 플래그를 설정하지 않은 경우 할당된 리소스는 할당 가능 리소스를 도입하기 전과 마찬가지로 노드의 용량으로 설정됩니다.



참고

reservedSystemCPUs 매개변수를 사용하여 특별히 예약한 **CPU**는 **kube-reserved** 또는 **system-reserved**를 사용하여 할당할 수 없습니다.

6.9.1.1. OpenShift Container Platform에서 할당된 리소스를 계산하는 방법

할당된 리소스 양은 다음 공식에 따라 계산됩니다.

$$[Allocatable] = [Node Capacity] - [system-reserved] - [Hard-Eviction-Thresholds]$$



참고

Allocatable에서 **Hard-Eviction-Thresholds**를 보유하면 **Allocatable** 값이 노드 수준에서 **Pod**에 적용되므로 시스템 신뢰도가 향상됩니다.

Allocatable이 음수인 경우 0으로 설정됩니다.

각 노드는 컨테이너 런타임 및 kubelet에서 사용하는 시스템 리소스를 보고합니다. **system-reserved** 매개변수 구성을 단순화하려면 **Node Summary API**를 사용하여 노드의 리소스 사용량을 확인합니다. 노드 요약은 `/api/v1/nodes/<node>/proxy/stats/summary`에 제공됩니다.

6.9.1.2. 노드에서 리소스 제약 조건을 적용하는 방법

노드는 구성된 할당 가능 값을 기반으로 **Pod**에서 사용할 수 있는 총 리소스 양을 제한할 수 있습니다. 이 기능을 사용하면 컨테이너 런타임 및 노드 에이전트와 같은 시스템 서비스에 필요한 **CPU** 및 메모리 리소스를 **Pod**에서 사용하지 못하도록 하여 노드의 안정성이 크게 향상됩니다. 관리자는 노드 안정성을 개선하기 위해 리소스 사용량 목표에 따라 리소스를 예약해야 합니다.

노드는 서비스 품질을 적용하는 새 **cgroup** 계층을 사용하여 리소스 제약 조건을 적용합니다. 모든 **Pod**는 시스템 데몬과는 별도의 전용 **cgroup** 계층에서 시작됩니다.

관리자는 서비스 품질이 보장된 **Pod**와 비슷한 시스템 데몬을 처리해야 합니다. 시스템 데몬은 바인딩 제어 그룹 내에서 버스트될 수 있으며 이 동작은 클러스터 배포의 일부로 관리해야 합니다. **system-reserved**에 **CPU** 및 메모리 리소스를 지정하여 시스템 데몬을 위한 **CPU** 및 메모리 리소스를 예약합니다.

system-reserved 제한을 강제 적용하여 중요한 시스템 서비스에서 **CPU** 및 메모리 리소스를 수신하지 못하도록 할 수 있습니다. 그 결과 메모리 부족 종료자에서 중요한 시스템 서비스를 종료할 수 있습니다. 정확한 추정치를 결정하기 위해 노드를 철저히 프로파일링하고 메모리 부족 종료자에서 해당 그룹의 프로세스를 종료할 때 중요한 시스템 서비스를 복구할 수 있다고 확인하는 경우에만 **system-reserved**를 강제 적용하는 것이 좋습니다.

6.9.1.3. 제거 임계값 이해

노드가 메모리 부족 상태에 있는 경우 전체 노드와 해당 노드에서 실행 중인 모든 **Pod**에 영향을 미칠 수 있습니다. 예를 들어 시스템 데몬에서 예약된 메모리보다 많은 양을 사용하면 메모리 부족 이벤트가 트리거될 수 있습니다. 노드에서는 시스템 메모리 부족 이벤트를 방지하거나 줄이기 위해 리소스 부족 처리 기능을 제공합니다.

--eviction-hard 플래그를 사용하여 일부 메모리를 예약할 수 있습니다. 노드는 노드의 메모리 가용성이 이 절대값 또는 백분을 아래로 떨어지면 **Pod**를 제거하려고 합니다. 노드에 시스템 데몬이 없는 경우 **Pod**는 메모리 **capacity - eviction-hard**로 제한됩니다. 이로 인해 메모리 부족 상태에 도달하기 전에 제거 버퍼로 따로 설정된 리소스를 **Pod**에 사용할 수 없습니다.

다음은 메모리에 할당 가능한 노드의 영향을 보여주는 예입니다.

- 노드 용량이 **32Gi**입니다.
- **--system-reserved**가 **3Gi**입니다.
- **--eviction-hard**가 **100Mi**로 설정되어 있습니다.

이 노드의 경우 유효 노드 할당 가능 값은 **28.9Gi**입니다. 노드 및 시스템 구성 요소에서 예약된 용량을 모두 사용하는 경우 **Pod**에 사용 가능한 메모리는 **28.9Gi**이고 이 임계값을 초과하는 경우 **Kubelet**은 **Pod**를 제거합니다.

노드 할당 가능 **28.9Gi**를 최상위 **cgroups**와 함께 적용하면 **Pod**에서 **28.9Gi**를 초과하지 않습니다. 시스템 데몬의 메모리 사용량이 **3.1Gi**를 초과하면 제거 작업이 수행됩니다.

위 예에서 시스템 데몬이 예약된 용량을 모두 사용하지 않는 경우 노드 제거가 시작되기 전에 **Pod**의 바인딩 **cgroup**에서 **memcg OOM**이 종료됩니다. 이러한 상황에서 **QoS**를 더 잘 적용하기 위해 노드는 모든 **Pod**가 **Node Allocatable + Eviction Hard Thresholds**가 되도록 최상위 **cgroup**에 하드 제거 임계값을 적용합니다.

시스템 데몬에서 예약된 용량을 모두 사용하지 않는 경우 노드는 **Pod**의 메모리 사용량이 **28.9Gi**를 초과할 때마다 **Pod**를 제거합니다. 제거 작업이 제시간에 수행되지 않아 **Pod**에서 **29Gi**의 메모리를 사용하면 **Pod**가 **OOM** 종료됩니다.

6.9.1.4. 스케줄러에서 리소스 가용성을 결정하는 방법

스케줄러는 **node.Status.Capacity**가 아닌 **node.Status.Allocatable**의 값을 사용하여 노드가 **Pod** 예약 후보가 될지 결정합니다.

기본적으로 노드는 클러스터에서 전체 머신 용량을 예약할 수 있는 것으로 보고합니다.

6.9.2. 노드에 리소스 자동 할당

OpenShift Container Platform은 특정 머신 구성 풀과 연결된 노드의 최적의 **system-reserved CPU** 및 메모리 리소스를 자동으로 확인하고 노드가 시작될 때 해당 값으로 노드를 업데이트할 수 있습니다. 기본적으로 **system-reserved CPU**는 **500m** 이고 **system-reserved** 메모리는 **1Gi** 입니다.

노드에서 **system-reserved** 리소스를 자동으로 결정하고 할당하려면 **KubeletConfig CR**(사용자 정의 리소스)을 생성하여 **autoSizingReserved: true** 매개변수를 설정합니다. 각 노드의 스크립트는 각 노드의 설치된 **CPU** 및 메모리 용량을 기반으로 예약된 각 리소스에 대한 최적 값을 계산합니다. 이 스크립트는 용량을 늘릴수록 예약된 리소스를 늘려야 합니다.

최적의 **system-reserved** 설정을 자동으로 확인하면 클러스터가 효율적으로 실행되고 **CRI-O** 및 **kubelet**과 같은 시스템 구성 요소의 리소스 부족으로 인해 값을 수동으로 계산하고 업데이트할 필요가 없습니다.

이 기능은 기본적으로 비활성화되어 있습니다.

사전 요구 사항

1.

다음 명령을 입력하여 구성할 노드 유형의 정적 **MachineConfigPool** 오브젝트와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

1

레이블은 **Labels** 아래에 표시됩니다.

작은 정보

적절한 라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

절차

1. 구성 변경을 위한 사용자 정의 리소스(CR)를 생성합니다.

리소스 할당 CR 구성 샘플

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: dynamic-node 1
spec:
  autoSizingReserved: true 2
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 3
  #...

```

1

CR에 이름을 지정합니다.

2

OpenShift Container Platform에서 지정된 레이블과 연결된 노드에서 **system-reserved** 리소스를 자동으로 결정하고 할당할 수 있도록 **autoSizingReserved** 매개변수 세트를 **true** 로 추가합니다. 해당 노드에서 자동 할당을 비활성화하려면 이 매개변수를 **false** 로 설정합니다.

3

"사전 요구 사항" 섹션에 구성된 머신 구성 풀에서 라벨을 지정합니다. **custom-kubelet: small-pods, default** 레이블, **pool.operator.machineconfiguration.openshift.io/worker: ""** 와 같은 머신 구성 풀에 대해 원하는 레이블을 선택할 수 있습니다.

이전 예제에서는 모든 작업자 노드에서 자동 리소스 할당을 활성화합니다. **OpenShift Container Platform**은 노드를 비우고 **kubelet** 구성을 적용한 다음 노드를 다시 시작합니다.

2. 다음 명령을 입력하여 **CR**을 생성합니다.

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

검증

1. 다음 명령을 입력하여 구성된 노드에 로그인합니다.

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

2. 디버그 셸 내에서 **/host**를 **root** 디렉터리로 설정합니다.

```
# chroot /host
```

3. **/etc/node-sizing.env** 파일을 확인합니다.

출력 예

```
SYSTEM_RESERVED_MEMORY=3Gi
SYSTEM_RESERVED_CPU=0.08
```

kubelet은 **/etc/node-sizing.env** 파일의 **system-reserved** 값을 사용합니다. 이전 예에서 작업자 노드에 **0.08 CPU** 및 메모리 **3Gi**가 할당됩니다. 최적 값이 표시되는데 몇 분이 걸릴 수 있습니다.

니다.

6.9.3. 노드에 수동으로 리소스 할당

OpenShift Container Platform은 할당을 위해 CPU 및 메모리 리소스 유형을 지원합니다. ephemeral-resource 리소스 유형도 지원됩니다. cpu 유형의 경우 200m,0.5 또는 1 과 같은 코어 단위로 리소스 수량을 지정합니다. memory 및 ephemeral-storage 의 경우 200Ki,50Mi 또는 5Gi 와 같은 바이트 단위로 리소스 수량을 지정합니다. 기본적으로 system-reserved CPU는 500m 이고 system-reserved 메모리는 1Gi 입니다.

관리자는 <resource_type>=<resource_quantity> 쌍 집합(예: cpu=200m,memory=512Mi)을 통해 kubelet 구성 CR(사용자 정의 리소스)을 사용하여 이러한 값을 설정할 수 있습니다.



중요

리소스 값을 수동으로 설정하려면 kubelet 구성 CR을 사용해야 합니다. 머신 구성 CR 을 사용할 수 없습니다.

권장되는 system-reserved 값에 대한 자세한 내용은 권장되는 system-reserved 값을 참조하십시오.

사전 요구 사항

1. 다음 명령을 입력하여 구성할 노드 유형의 정적 MachineConfigPool CRD와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
```

```

labels:
  pools.operator.machineconfiguration.openshift.io/worker: "" 1
name: worker
#...

```

1

레이블이 **Labels** 아래에 표시됩니다.

작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

절차

1.

구성 변경을 위한 사용자 정의 리소스 (**CR**)를 만듭니다.

리소스 할당 **CR** 구성 샘플

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-allocatable 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    systemReserved: 3
    cpu: 1000m
    memory: 1Gi
#...

```

1

CR에 이름을 지정합니다.

2

머신 구성 풀에서 라벨을 지정합니다.

3

노드 구성 요소 및 시스템 구성 요소를 위해 예약할 리소스를 지정합니다.

2.

다음 명령을 실행하여 CR을 생성합니다.

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

6.10. 클러스터의 노드에 특정 CPU 할당

정적 CPU 관리자 정책을 사용하는 경우 클러스터의 특정 노드에서 사용할 특정 CPU를 예약할 수 있습니다. 예를 들어 CPU가 24개인 시스템에서 컨트롤 플레인에 0~3번 CPU를 예약하여 컴퓨팅 노드에서 4~23번 CPU를 사용하도록 할 수 있습니다.

6.10.1. 노드의 CPU 예약

특정 노드에 예약된 CPU 목록을 명시적으로 정의하려면 KubeletConfig CR(사용자 정의 리소스)을 생성하여 reservedSystemCPUs 매개변수를 정의합니다. 이 목록은 systemReserved 매개변수를 사용하여 예약할 수 있는 CPU를 대체합니다.

프로세스

1.

구성하려는 노드 유형의 MCP(머신 구성 풀)와 연결된 라벨을 가져옵니다.

```
$ oc describe machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc describe machineconfigpool worker
```

출력 예

```

Name:      worker
Namespace:
Labels:    machineconfiguration.openshift.io/mco-built-in=
           pools.operator.machineconfiguration.openshift.io/worker= 1
Annotations: <none>
API Version: machineconfiguration.openshift.io/v1
Kind:      MachineConfigPool
#...

```

1

MCP 라벨을 가져옵니다.

2.

KubeletConfig CR의 YAML 파일을 생성합니다.

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-reserved-cpus 1
spec:
  kubeletConfig:
    reservedSystemCPUs: "0,1,2,3" 2
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 3
#...

```

1

CR 이름을 지정합니다.

2

MCP와 연결된 노드에 예약할 CPU의 코어 ID를 지정합니다.

3

MCP에서 라벨을 지정합니다.

- 3. **CR 오브젝트를 생성합니다.**

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

추가 리소스

- **systemReserved** 매개변수에 대한 자세한 내용은 [OpenShift Container Platform 클러스터의 노드의 리소스 할당](#) 을 참조하십시오.

6.11. KUBELET의 TLS 보안 프로파일 활성화

TLS(Transport Layer Security) 보안 프로필을 사용하여 **HTTP** 서버 역할을 할 때 **kubelet**에 필요한 **TLS** 암호를 정의할 수 있습니다. **kubelet**은 **HTTP/GRPC** 서버를 사용하여 명령을 **Pod**에 전송하고 로그를 수집하며 **kubelet**을 통해 **Pod**에서 **exec** 명령을 실행하는 **Kubernetes API** 서버와 통신합니다.

TLS 보안 프로파일은 **kubelet**과 **Kubernetes API** 서버 간의 통신을 보호하기 위해 **Kubernetes API** 서버와 연결할 때 사용해야 하는 **TLS** 암호를 정의합니다.



참고

기본적으로 **kubelet**이 **Kubernetes API** 서버의 클라이언트 역할을 할 때 **API** 서버와 **TLS** 매개변수를 자동으로 협상합니다.



6.11.1. TLS 보안 프로파일 이해

TLS(Transport Layer Security) 보안 프로필을 사용하여 다양한 **OpenShift Container Platform** 구성 요소에 필요한 **TLS** 암호를 정의할 수 있습니다. **OpenShift Container Platform TLS** 보안 프로파일은 **Mozilla** 권장 구성을 기반으로 합니다.

각 구성 요소에 대해 다음 **TLS** 보안 프로파일 중 하나를 지정할 수 있습니다.

표 6.4. TLS 보안 프로파일

Profile	설명
---------	----

Profile	설명
Old	<p>이 프로파일은 레거시 클라이언트 또는 라이브러리와 함께 사용하기 위한 것입니다. 프로파일은 이전 버전과의 호환성 권장 구성을 기반으로 합니다.</p> <p>Old 프로파일에는 최소 TLS 버전 1.0이 필요합니다.</p> <div style="display: flex; align-items: center;">  <div> <p>참고</p> <p>Ingress 컨트롤러의 경우 최소 TLS 버전이 1.0에서 1.1로 변환됩니다.</p> </div> </div>
Intermediate	<p>이 프로파일은 대부분의 클라이언트에서 권장되는 구성입니다. Ingress 컨트롤러, kubelet 및 컨트롤 플레인의 기본 TLS 보안 프로파일입니다. 프로파일은 중간 호환성 권장 구성을 기반으로 합니다.</p> <p>Intermediate 프로파일에는 최소 TLS 버전이 1.2가 필요합니다.</p>
Modern	<p>이 프로파일은 이전 버전과의 호환성이 필요하지 않은 최신 클라이언트와 사용하기 위한 것입니다. 이 프로파일은 최신 호환성 권장 구성을 기반으로 합니다.</p> <p>Modern 프로파일에는 최소 TLS 버전 1.3이 필요합니다.</p>
사용자 지정	<p>이 프로 파일을 사용하면 사용할 TLS 버전과 암호를 정의할 수 있습니다.</p> <div style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <div style="display: flex; align-items: center;">  <div> <p>주의</p> <p>Custom 프로파일을 사용할 때는 잘못된 구성으로 인해 문제가 발생할 수 있으므로 주의해야 합니다.</p> </div> </div> </div>



참고

미리 정의된 프로파일 유형 중 하나를 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어 릴리스 X.Y.Z에 배포된 중간 프로파일을 사용하는 사양이 있는 경우 릴리스 X.Y.Z+1로 업그레이드하면 새 프로파일 구성이 적용되어 롤아웃이 발생할 수 있습니다.

6.11.2. kubelet의 TLS 보안 프로파일 구성

HTTP 서버 역할을 할 때 kubelet에 대한 TLS 보안 프로파일을 구성하려면 KubeletConfig CR(사용자 정의 리소스)을 생성하여 특정 노드에 대해 사전 정의 또는 사용자 지정 TLS 보안 프로파일을 지정합니다. TLS 보안 프로파일이 구성되지 않은 경우 기본 TLS 보안 프로파일은 **Intermediate**입니다.

작업자 노드에서 **Old TLS** 보안 프로필을 구성하는 샘플 **KubeletConfig CR**

```

apiVersion: config.openshift.io/v1
kind: KubeletConfig
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
#...

```

구성된 노드의 `kubelet.conf` 파일에서 구성된 **TLS** 보안 프로필의 암호 및 최소 **TLS** 버전을 확인할 수 있습니다.

사전 요구 사항

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

절차

1. **KubeletConfig CR**을 생성하여 **TLS** 보안 프로필을 구성합니다.

Custom 프로파일의 샘플 **KubeletConfig CR**

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
      ciphers: 3
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256

```

```

- ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion: VersionTLS11
machineConfigPoolSelector:
  matchLabels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 4
#...

```

1

TLS 보안 프로파일 유형(**Old**, **Intermediate** 또는 **Custom**)을 지정합니다. 기본값은 **Intermediate**입니다.

2

선택한 유형의 적절한 필드를 지정합니다.

- `old: {}`
- `intermediate: {}`
- `custom:`

3

custom 유형의 경우 **TLS** 암호화 목록 및 최소 허용된 **TLS** 버전을 지정합니다.

4

선택 사항: **TLS** 보안 프로파일을 적용하려는 노드의 머신 구성 풀 레이블을 지정합니다.

2.

KubeletConfig 오브젝트를 생성합니다.

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

클러스터의 작업자 노드 수에 따라 구성된 노드가 하나씩 재부팅될 때까지 기다립니다.

검증

프로필이 설정되었는지 확인하려면 노드가 **Ready** 상태가 된 후 다음 단계를 수행합니다.

1. 구성된 노드의 디버그 세션을 시작합니다.

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

2. 디버그 셸 내에서 `/host`를 `root` 디렉터리로 설정합니다.

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

3. `kubelet.conf` 파일을 확인합니다.

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

출력 예

```
"kind": "KubeletConfiguration",
"apiVersion": "kubelet.config.k8s.io/v1beta1",
#...
"tlsCipherSuites": [
  "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
  "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
  "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
  "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
  "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
  "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
],
"tlsMinVersion": "VersionTLS12",
#...
```

6.12. 머신 구성 데몬 메트릭

머신 구성 데몬은 **Machine Config Operator**의 일부입니다. 클러스터의 모든 노드에서 실행됩니다. 머신 구성 데몬은 각 노드의 구성 변경 및 업데이트를 관리합니다.

6.12.1. 머신 구성 데몬 메트릭

OpenShift Container Platform 4.3부터는 머신 구성 데몬에서 일련의 메트릭을 제공합니다. 이러한 메트릭은 **Prometheus** 클러스터 모니터링 스택을 사용하여 액세스할 수 있습니다.

다음 테이블에 이러한 메트릭 집합이 설명되어 있습니다. 일부 항목에는 특정 로그를 가져오는 명령이 포함되어 있습니다. **oc adm must-gather** 명령을 사용하여 가장 포괄적인 로그 세트인 **Hpwever**를 사용할 수 있습니다.



참고

Name 및 **Description** 열에서 *로 표시된 지표는 성능 문제가 발생할 수 있는 심각한 오류를 나타냅니다. 이러한 문제가 발생하면 업데이트 및 업그레이드가 진행되지 않을 수 있습니다.

표 6.5. MCO 메트릭

이름	형식	설명	참고
mcd_host_os_and_version	<code>[]string{"os", "version"}</code>	MCD가 실행 중인 OS(예: RHCOS 또는 RHEL)를 표시합니다. RHCOS의 경우 버전이 제공됩니다.	
mcd_drain_error*		드레이닝 실패 중 수신한 오류를 기록합니다. *	<p>드레이닝에 성공하려면 여러 번 시도해야 할 수 있지만 터미널에서 드레이닝이 실패하면 업데이트가 진행되지 않습니다. drain_time 메트릭은 드레이닝에 걸린 시간을 표시하며 문제 해결에 도움이 될 수 있습니다.</p> <p>추가 조사가 필요한 경우 다음을 실행하여 로그를 참조하십시오.</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon- <hash> -c machine-config-daemon</pre>

이름	형식	설명	참고
mcd_pivot_err*	<code>[]string{"err", "node", "pivot_target"}</code>	피벗 중 로그 오류가 발생했습니다. *	<p>피벗 오류로 인해 OS 업그레이드가 진행되지 않을 수 있습니다.</p> <p>추가 조사를 수행하려면 다음 명령을 실행하여 machine-config-daemon 컨테이너의 로그를 확인합니다.</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre>
mcd_state	<code>[]string{"state", "reason"}</code>	표시된 노드의 머신 구성 데몬 상태입니다. 가능한 상태는 "완료", "작업 중", "저하됨"입니다. "저하됨"의 경우 이유가 포함됩니다.	<p>추가 조사가 필요한 경우 다음을 실행하여 로그를 참조하십시오.</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre>
mcd_kubelet_state*		kubelet 상태 장애를 기록합니다. *	<p>이 값은 비어 있고 실패 횟수가 0이어야 합니다. 실패 횟수가 2를 초과하면 임계값이 초과되었음을 나타내는 오류입니다. 이는 kubelet 상태에 문제가 있을 수 있음을 나타냅니다.</p> <p>추가 조사를 수행하려면 다음 명령을 실행하여 노드에 액세스한 후 해당 로그를 모두 확인합니다.</p> <pre>\$ oc debug node/<node> — chroot /host journalctl -u kubelet</pre>
mcd_reboot_err*	<code>[]string{"message", "err", "node"}</code>	실패한 재부팅 및 해당 오류를 기록합니다. *	<p>이 값은 비어 있을 것으로 예상되며, 재부팅에 성공했음을 나타냅니다.</p> <p>추가 조사가 필요한 경우 다음을 실행하여 로그를 참조하십시오.</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre>

이름	형식	설명	참고
mcd_update_state	<code>[]string{"config", "err"}</code>	구성 업데이트의 성공 또는 실패와 해당 오류를 기록합니다.	<p>예상 값은 rendered-master/rendered-worker-XXXX입니다. 업데이트에 실패하면 오류가 발생합니다.</p> <p>추가 조사가 필요한 경우 다음을 실행하여 로그를 참조하십시오.</p> <pre>\$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon</pre>

추가 리소스

- [모니터링 개요](#)
- [클러스터에 대한 데이터 수집](#)

6.13. 인프라 노드 생성

중요

머신 API가 작동하는 클러스터에서만 고급 머신 관리 및 스케일링 기능을 사용할 수 있습니다. 사용자 프로비저닝 인프라가 있는 클러스터에는 **Machine API**를 사용하려면 추가 검증 및 구성이 필요합니다.

인프라 플랫폼 유형의 클러스터가 **Machine API**를 사용할 수 없습니다. 이 제한은 클러스터에 연결된 컴퓨팅 시스템이 기능을 지원하는 플랫폼에 설치된 경우에도 적용됩니다. 이 매개변수는 설치 후 변경할 수 없습니다.

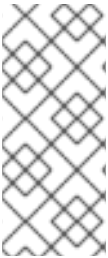
클러스터의 플랫폼 유형을 보려면 다음 명령을 실행합니다.

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

인프라 머신 세트를 사용하여 기본 라우터, 통합 컨테이너 이미지 레지스트리, 클러스터 지표 및 모니터링 구성 요소와 같은 인프라 구성 요소만 호스팅하는 머신을 생성할 수 있습니다. 이러한 인프라 시스템은

환경을 실행하는 데 필요한 총 서브스크립션 수에 포함되지 않습니다.

프로덕션 배포에서는 인프라 구성 요소를 유지하기 위해 세 개 이상의 머신 세트를 배포하는 것이 좋습니다. **OpenShift Logging** 및 **Red Hat OpenShift Service Mesh** 둘 다 다른 노드에 3개의 인스턴스를 설치해야 하는 **Elasticsearch**를 배포합니다. 이러한 각 노드는 고가용성을 위해 다른 가용성 영역에 배포할 수 있습니다. 이 구성에는 가용성 영역마다 하나씩 3개의 서로 다른 머신 세트가 필요합니다. 여러 가용성 영역이 없는 글로벌 **Azure** 리전에서는 가용성 세트를 사용하여 고가용성을 보장할 수 있습니다.



참고

인프라 노드에 **NoSchedule** 테인트를 추가하면 해당 노드에서 실행 중인 기존 **DNS Pod**가 잘못된 일정 으로 표시됩니다. 잘못된 예약 **DNS Pod**에서 허용 오차를 삭제하거나 추가해야 합니다.

6.13.1. OpenShift Container Platform 인프라 구성 요소

각 자체 관리형 **Red Hat OpenShift** 서브스크립션에는 **OpenShift Container Platform** 및 기타 **OpenShift** 관련 구성 요소에 대한 권한이 포함되어 있습니다. 이러한 권한은 **OpenShift Container Platform** 컨트롤 플레인 및 인프라 워크로드를 실행하는 데 포함되며 크기 조정 중에 고려하지 않아도 됩니다.

인프라 노드로 자격을 부여하고 포함된 인타이틀먼트를 사용하려면 최종 사용자 애플리케이션의 일부가 아닌 클러스터를 지원하는 구성 요소만 해당 인스턴스에서 실행할 수 있습니다. 예를 들면 다음과 같은 구성 요소가 있습니다.

- **Kubernetes** 및 **OpenShift Container Platform** 컨트롤 플레인 서비스
- 기본 라우터
- 통합된 컨테이너 이미지 레지스트리
- **HAProxy** 기반 **Ingress** 컨트롤러
- 사용자 정의 프로젝트를 모니터링하기 위한 구성 요소를 포함한 클러스터 메트릭 수집 또는 모니터링 서비스

- *클러스터 집계 로깅*
- *Red Hat Quay*
- *Red Hat OpenShift Data Foundation*
- *Red Hat Advanced Cluster Security for Kubernetes*
- *Red Hat Advanced Cluster Security for Kubernetes*
- *Red Hat OpenShift GitOps*
- *Red Hat OpenShift Pipelines*
- *Red Hat OpenShift Service Mesh*

다른 컨테이너, **Pod** 또는 구성 요소를 실행하는 모든 노드는 서브스크립션을 적용해야 하는 작업자 노드입니다.

인프라 노드 및 인프라 노드에서 실행할 수 있는 구성 요소에 대한 자세한 내용은 [엔터프라이즈 Kubernetes용 OpenShift 크기 조정 및 서브스크립션 가이드의 "Red Hat OpenShift 컨트롤 플레인 및 인프라 노드"](#) 섹션을 참조하십시오.

인프라 노드를 생성하려면 머신 세트를 사용하거나 노드에 레이블을 지정하거나 머신 구성 풀을 사용할 수 있습니다.

6.13.1.1. 인프라 노드 생성



중요

설치 관리자 프로비저닝 인프라 환경 또는 머신 API에서 컨트롤 플레인 노드를 관리하는 클러스터의 인프라 머신 세트 생성을 참조하십시오.

클러스터의 요구 사항은 **infra** 노드라고도 불리는 인프라를 프로비저닝해야 합니다. 설치 프로그램은 컨트롤 플레인 및 작업자 노드에 대한 프로비저닝만 제공합니다. 작업자 노드는 레이블을 통해 인프라 노드 또는 애플리케이션 (**app**, **app** 이라고도 함)으로 지정할 수 있습니다.

절차

1. 애플리케이션 노드 역할을 수행할 작업자 노드에 레이블을 추가합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/app=""
```

2. 인프라 노드 역할을 수행할 작업자 노드에 레이블을 추가합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

3. 해당 노드에 **infra** 역할 및 **app** 역할이 있는지 확인합니다.

```
$ oc get nodes
```

4. 기본 클러스터 수준 노드 선택기를 생성합니다. 기본 노드 선택기는 모든 네임스페이스에서 생성된 **Pod**에 적용됩니다. 이렇게 하면 **Pod**의 기존 노드 선택기와 교차점이 생성되어 **Pod**의 선택기가 추가로 제한됩니다.

중요

기본 노드 선택기 키가 **Pod** 라벨 키와 충돌하는 경우 기본 노드 선택기가 적용되지 않습니다.

그러나 **Pod**를 예약할 수 없게 만들 수 있는 기본 노드 선택기를 설정하지 마십시오. 예를 들어 **Pod**의 라벨이 `node-role.kubernetes.io/master=""`와 같은 다른 노드 역할로 설정된 경우 기본 노드 선택기를 `node-role.kubernetes.io/infra=""`와 같은 특정 노드 역할로 설정하면 **Pod**를 예약할 수 없게 될 수 있습니다. 따라서 기본 노드 선택기를 특정 노드 역할로 설정할 때 주의해야 합니다.

또는 프로젝트 노드 선택기를 사용하여 클러스터 수준 노드 선택기 키 충돌을 방지할 수 있습니다.

- a. **Scheduler** 오브젝트를 편집합니다.

```
$ oc edit scheduler cluster
```

- b. 적절한 노드 선택기를 사용하여 **defaultNodeSelector** 필드를 추가합니다.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec:
  defaultNodeSelector: node-role.kubernetes.io/infra="" 1
# ...
```

1

이 예제 노드 선택기는 기본적으로 인프라 노드에 **Pod**를 배포합니다.

- c. 파일을 저장하여 변경 사항을 적용합니다.

이제 인프라 리소스를 새로 레이블이 지정된 인프라 노드로 이동할 수 있습니다.

추가 리소스

- [인프라 머신 세트로 리소스 이동](#)

7장. 컨테이너 작업

7.1. 컨테이너 이해

OpenShift Container Platform 애플리케이션의 기본 단위는 컨테이너라고 합니다. **Linux 컨테이너 기술**은 실행 중인 프로세스를 격리하여 지정된 리소스와만 상호 작용하도록 제한하기 위한 간단한 메커니즘입니다.

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

Linux 커널은 수년간 컨테이너 기술을 위한 기능을 통합해 왔습니다. **OpenShift Container Platform** 및 **Kubernetes**는 다중 호스트 설치에서 컨테이너를 오케스트레이션하는 기능을 추가합니다.

7.1.1. 컨테이너 및 RHEL 커널 메모리 정보

RHEL(Red Hat Enterprise Linux) 동작으로 인해 **CPU** 사용량이 많은 노드의 컨테이너에서 예상보다 많은 메모리를 사용하는 것처럼 보일 수 있습니다. 메모리 사용량 증가는 **RHEL** 커널의 **kmem_cache**로 인한 것일 수 있습니다. **RHEL** 커널은 각 **cgroup**에 **kmem_cache**를 생성합니다. 성능 향상을 위해 **kmem_cache**에는 **cpu_cache**와 모든 **NUMA** 노드에 대한 노드 캐시가 포함됩니다. 이러한 캐시는 모두 커널 메모리를 사용합니다.

이러한 캐시에 저장된 메모리 양은 시스템에서 사용하는 **CPU** 수에 비례합니다. 결과적으로 **CPU** 수가 많을수록 이러한 캐시에 더 많은 양의 커널 메모리가 저장됩니다. 이러한 캐시의 커널 메모리 양이 늘어나면 **OpenShift Container Platform** 컨테이너가 구성된 메모리 제한을 초과하여 컨테이너가 종료될 수 있습니다.

커널 메모리 문제로 인한 컨테이너 손실을 방지하려면 컨테이너에서 메모리를 충분히 요청해야 합니다. 다음 공식을 사용하여 **kmem_cache**에서 사용하는 메모리의 양을 추정할 수 있습니다. 여기서 **nproc**는 **nproc** 명령에서 보고하는 처리 단위의 수입니다. 컨테이너 요청의 하한은 이 값에 컨테이너 메모리 요구 사항을 더한 값이어야 합니다.

\$(nproc) X 1/2 MiB

7.1.2. 컨테이너 엔진 및 컨테이너 런타임 정보

컨테이너 엔진은 명령줄 옵션 및 이미지 가져오기를 포함하여 사용자 요청을 처리하는 소프트웨어입니다. 컨테이너 엔진은 하위 수준 컨테이너 런타임 라고도 하는 컨테이너 런타임을 사용하여 컨테이너 배

포 및 운영에 필요한 구성 요소를 실행하고 관리합니다. 컨테이너 엔진 또는 컨테이너 런타임과 상호 작용할 필요가 없습니다.



참고

OpenShift Container Platform 문서에서는 컨테이너 런타임이라는 용어를 사용하여 하위 수준 컨테이너 런타임을 참조합니다. 기타 문서는 컨테이너 엔진을 컨테이너 런타임으로 참조할 수 있습니다.

OpenShift Container Platform은 컨테이너 엔진으로 **CRI-O**를 사용하고 컨테이너 런타임으로 **runC** 또는 **crun**을 사용합니다. 기본 컨테이너 런타임은 **runC**입니다. 두 컨테이너 런타임 모두 **OCI(Open Container Initiative)** 런타임 사양을 준수합니다.

CRI-O는 운영 체제와 긴밀하게 통합되어 효율적이고 최적화된 **Kubernetes** 환경을 제공하는 **Kubernetes** 네이티브 컨테이너 엔진 구현입니다. **CRI-O** 컨테이너 엔진은 각 **OpenShift Container Platform** 클러스터 노드에서 **systemd** 서비스로 실행됩니다.

Docker에서 개발했으며 **Open Container Project**에서 유지 관리하는 **runc**는 **Go**로 작성된 경량의 이식 가능한 컨테이너 런타임입니다. **Red Hat**에서 개발한 **crun**은 **C**. **OpenShift Container Platform 4.12**에서 완전히 작성된 빠르고 낮은 메모리 컨테이너 런타임입니다. **OpenShift Container Platform 4.12**에서는 둘 중 하나를 선택할 수 있습니다.



중요

crun 컨테이너 런타임 지원은 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.

crun에는 다음과 같은 **runC**에 대한 몇 가지 개선 사항이 있습니다.

-

작은 바이너리

- 더 빠른 처리
- 메모리 공간 감소

runc는 **crun**에 비해 다음과 같은 몇 가지 이점이 있습니다.

- 가장 많이 사용되는 **OCI** 컨테이너 런타임.
- 프로덕션에서 더 긴 용량입니다.
- **CRI-O**의 기본 컨테이너 런타임.

필요에 따라 두 컨테이너 런타임 간에 이동할 수 있습니다.

사용할 컨테이너 런타임 설정에 대한 자세한 내용은 [CRI-O 매개변수를 편집하기 위해 ContainerRuntimeConfig CR 생성](#) 을 참조하십시오.

7.2. POD를 배포하기 전에 **INIT CONTAINER**를 사용하여 작업 수행

OpenShift Container Platform에서는 애플리케이션 컨테이너보다 먼저 실행되고 앱 이미지에 없는 유틸리티 또는 설정 스크립트를 포함할 수 있는 특수 컨테이너인 **Init Container**를 제공합니다.

7.2.1. **Init Container** 이해

Init Container 리소스를 사용하여 나머지 **Pod**를 배포하기 전에 작업을 수행할 수 있습니다.

Pod에는 애플리케이션 컨테이너 외에도 **Init Container**가 있을 수 있습니다. **Init Container**를 사용하면 설정 스크립트 및 바인딩 코드를 재구성할 수 있습니다.

Init Container는 다음을 수행할 수 있습니다.

- 보안상의 이유로 앱 컨테이너 이미지에 포함하지 않는 유틸리티를 포함 및 실행합니다.
- 앱 이미지에 없는 설정에 대한 유틸리티 또는 사용자 정의 코드를 포함합니다. 예를 들어, 설치 중에 `sed`, `awk`, `python` 또는 `dig`와 같은 툴을 사용하기 위해 다른 이미지에서 이미지를 만들 필요가 없습니다.
- 애플리케이션 컨테이너에서 액세스할 수 없는 보안에 대한 액세스 권한과 같이 파일 시스템 보기가 앱 컨테이너와 다르게 표시되도록 **Linux** 네임스페이스를 사용합니다.

각 **Init Container**는 다음 컨테이너를 시작하기 전에 성공적으로 완료해야 합니다. 이를 위해 **Init Container**에서는 몇 가지 사전 조건 집합이 충족될 때까지 앱 컨테이너의 시작을 차단하거나 지연할 수 있는 쉬운 방법을 제공합니다.

예를 들어 다음은 **Init Container**를 사용할 수 있는 몇 가지 방법입니다.

- 다음과 같은 셸 명령을 사용하여 서비스가 생성될 때까지 기다립니다.

```
for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; done; exit 1
```

- 다음과 같은 명령을 사용하여 **Downward API**에서 원격 서버에 이 **Pod**를 등록합니다.

```
$ curl -X POST
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d
'instance=${}&ip=${}'
```

- `sleep 60`과 같은 명령을 사용하여 앱 컨테이너를 시작하기 전에 잠시 기다립니다.
- **Git** 리포지토리를 볼륨에 복제합니다.
- 구성 파일에 값을 저장하고 템플릿 툴을 실행하여 기본 앱 컨테이너에 대한 구성 파일을 동적으로 생성합니다. 예를 들어 `POD_IP` 값을 구성에 저장하고 **Jinja**를 사용하여 기본 앱 구성 파일을 생성합니다.

자세한 내용은 [Kubernetes 설명서](#)를 참조하십시오.

7.2.2. Init Container 생성

다음 예제에서는 **Init Container**가 두 개 있는 간단한 **Pod**를 간략하게 설명합니다. 첫 번째 컨테이너는 **myservice**를 기다리고 두 번째 컨테이너는 **mydb**를 기다립니다. 두 컨테이너가 모두 완료되면 **Pod**가 시작됩니다.

절차

1.

Init Container의 **Pod**를 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for
myservice; sleep 2; done;']
  - name: init-mydb
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep
2; done;']
# ...
```

b.

Pod를 생성합니다.

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

c.

Pod 상태를 확인합니다.

```
$ oc get pods
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	0/1	Init:0/2	0	5s

포드 상태 **Init:0/2** 는 두 서비스를 대기 중임을 나타냅니다.

2.

myservice 서비스를 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```

kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376

```

b.

Pod를 생성합니다.

```

$ oc create -f myservice.yaml

```

c.

Pod 상태를 확인합니다.

```

$ oc get pods

```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	0/1	Init:1/2	0	5s

Pod 상태 *Init:1/2* 는 하나의 서비스(이 경우 *mydb* 서비스)를 대기 중임을 나타냅니다.

3.

mydb 서비스를 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```

kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9377

```

b.

Pod를 생성합니다.

```

$ oc create -f mydb.yaml

```

c.

Pod 상태를 확인합니다.

```

$ oc get pods

```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	1/1	Running	0	2m

Pod 상태는 더 이상 서비스를 기다리지 않고 실행 중임을 나타냅니다.

7.3. 볼륨을 사용하여 컨테이너 데이터 유지

컨테이너의 파일은 임시 파일입니다. 컨테이너가 충돌하거나 중지되면 데이터가 손실됩니다. 볼륨을

사용하여 **Pod**의 컨테이너에서 사용하는 데이터를 유지할 수 있습니다. 볼륨은 **Pod** 수명 동안 데이터가 저장되는 **Pod**의 컨테이너에 액세스할 수 있는 디렉터리입니다.

7.3.1. 볼륨 이해

볼륨은 **Pod** 및 해당 컨테이너에서 사용할 수 있는 마운트된 파일 시스템으로, 다수의 호스트-로컬 또는 네트워크 연결 스토리지 끝점에서 지원할 수 있습니다. 컨테이너는 기본적으로 영구적이 아니며 재시작 시 저장된 콘텐츠가 지워집니다.

볼륨의 파일 시스템에 오류가 없는지 확인한 후 오류가 있고 가능한 경우 오류를 복구하기 위해 **OpenShift Container Platform**에서는 **mount** 유틸리티 이전에 **fsck** 유틸리티를 호출합니다. 이러한 작업은 볼륨을 추가하거나 기존 볼륨을 업데이트할 때 수행됩니다.

가장 간단한 볼륨 유형은 단일 머신의 임시 디렉터리인 **emptyDir**입니다. 관리자는 **Pod**에 자동으로 연결된 영구 볼륨을 요청할 수도 있습니다.



참고

클러스터 관리자가 **FSGroup** 매개변수를 활성화하면 **Pod**의 **FSGroup**을 기반으로 한 할당량에 따라 **emptyDir** 볼륨 스토리지가 제한될 수 있습니다.

7.3.2. OpenShift Container Platform CLI를 사용하여 볼륨 작업

CLI 명령 **oc set volume**을 사용하여 복제 컨트롤러 또는 배포 구성과 같은 **Pod** 템플릿이 있는 오브젝트의 볼륨 및 볼륨 마운트를 추가하고 제거할 수 있습니다. **Pod**의 볼륨 또는 **Pod** 템플릿이 있는 오브젝트도 나열할 수 있습니다.

oc set volume 명령에서는 다음과 같은 일반 구문을 사용합니다.

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

오브젝트 선택

oc set volume 명령에서 **object_selection** 매개변수에 다음 중 하나를 지정합니다.

표 7.1. 오브젝트 선택

구문	설명	예
<code><object_type> <name></code>	유형 <code><object_type></code> 의 <code><name></code> 을 선택합니다.	<code>deploymentConfig registry</code>
<code><object_type>/<name></code>	유형 <code><object_type></code> 의 <code><name></code> 을 선택합니다.	<code>deploymentConfig/registry</code>
<code><object_type>--selector=<object_label_selector></code>	지정된 라벨 선택기와 일치하는 유형 <code><object_type></code> 의 리소스를 선택합니다.	<code>deploymentConfig--selector="name=registry"</code>
<code><object_type> --all</code>	유형 <code><object_type></code> 의 모든 리소스를 선택합니다.	<code>deploymentConfig --all</code>
<code>-f 또는 --filename=<file_name></code>	리소스를 편집하는 데 사용할 파일 이름, 디렉터리 또는 URL입니다.	<code>-f registry-deployment-config.json</code>

작업

`oc set volume` 명령의 `operation` 매개변수에 `--add` 또는 `--remove`를 지정합니다.

필수 매개변수

모든 필수 매개변수는 선택한 작업에 한정되며 뒤에 나오는 섹션에서 설명합니다.

옵션

모든 옵션은 선택한 작업에 한정되며 뒤에 나오는 섹션에서 설명합니다.

7.3.3. Pod의 볼륨 및 볼륨 마운트 나열

`Pod` 또는 `Pod` 템플릿의 볼륨 및 볼륨 마운트를 나열할 수 있습니다.

프로세스

볼륨 목록을 표시하려면 다음을 수행합니다.

```
$ oc set volume <object_type>/<name> [options]
```

볼륨에서 지원하는 옵션을 나열합니다.

옵션	설명	기본
--name	볼륨 이름입니다.	
-c, --containers	이름으로 컨테이너를 선택합니다. 문자와 일치하는 와일드카드 '*'를 사용할 수도 있습니다.	'*'

예를 들면 다음과 같습니다.

- **Pod p1에 대한 모든 볼륨을 나열하려면 다음을 실행합니다.**

```
$ oc set volume pod/p1
```

- **모든 배포 구성에 정의된 볼륨 v1을 나열하려면 다음을 실행합니다.**

```
$ oc set volume dc --all --name=v1
```

7.3.4. Pod에 볼륨 추가

Pod에 볼륨 및 볼륨 마운트를 추가할 수 있습니다.

프로세스

볼륨, 볼륨 마운트 또는 둘 다를 Pod 템플릿에 추가하려면 다음을 실행합니다.

```
$ oc set volume <object_type>/<name> --add [options]
```

표 7.2. 볼륨 추가에 지원되는 옵션

옵션	설명	기본
--name	볼륨 이름입니다.	지정하지 않으면 자동으로 생성됩니다.
-t, --type	볼륨 소스의 이름입니다. 지원되는 값은 emptyDir , hostPath , secret , configmap , persistentVolumeClaim 또는 projected 입니다.	emptyDir

옵션	설명	기본
-c, --containers	이름으로 컨테이너를 선택합니다. 문자와 일치하는 와일드카드 '*'를 사용할 수도 있습니다.	'*'
-m, --mount-path	선택한 컨테이너 내부의 마운트 경로입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 /dev/pts 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. /host를 사용하여 호스트를 마운트하는 것이 안전합니다.	
--path	호스트 경로입니다. -- type=hostPath 에 대한 필수 매개변수입니다. 컨테이너 루트, / 또는 호스트와 컨테이너에서 동일한 경로에 마운트하지 마십시오. 컨테이너가 호스트 /dev/pts 파일과 같이 충분한 권한이 있는 경우 호스트 시스템이 손상될 수 있습니다. /host를 사용하여 호스트를 마운트하는 것이 안전합니다.	
--secret-name	보안의 이름입니다. -- type=secret 에 대한 필수 매개변수입니다.	
--configmap-name	구성 맵의 이름입니다. -- type=configmap 에 대한 필수 매개변수입니다.	
--claim-name	영구 볼륨 클레임의 이름입니다. -- type=persistentVolumeClaim 에 대한 필수 매개변수입니다.	
--source	JSON 문자열로 된 볼륨 소스 세부 정보입니다. 필요한 볼륨 소스를 -- type 에서 지원하지 않는 경우 사용하는 것이 좋습니다.	
-o, --output	수정된 오브젝트를 서버에서 업데이트하는 대신 표시합니다. 지원되는 값은 json, yaml 입니다.	
--output-version	지정된 버전으로 수정된 오브젝트를 출력합니다.	api-version

예를 들면 다음과 같습니다.

- 레지스트리 **DeploymentConfig** 오브젝트에 새 볼륨 소스 **emptyDir**을 추가하려면 다음을 실행합니다.

```
$ oc set volume dc/registry --add
```

작은 정보

다음 **YAML**을 적용하여 볼륨을 추가할 수 있습니다.

예 7.1. 볼륨이 추가된 배포 구성 샘플

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: registry
  namespace: registry
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes: 1
        - name: volume-pppsw
          emptyDir: {}
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
```

1

볼륨 소스 **emptyDir**을 추가합니다.

- 복제 컨트롤러 **r1**에 대해 보안 **secret1**을 사용하여 볼륨 **v1**을 추가하고 **/data**:의 컨테이너 내부에 마운트하려면 다음을 실행합니다.

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='secret1' --mount-path=/data
```

작은 정보

다음 **YAML**을 적용하여 볼륨을 추가할 수 있습니다.

예 7.2. 볼륨 및 보안이 추가된 샘플 복제 컨트롤러

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      creationTimestamp: null
    labels:
      app: httpd
      deployment: example-1
      deploymentconfig: example
    spec:
      volumes: ①
      - name: v1
        secret:
          secretName: secret1
          defaultMode: 420
      containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        volumeMounts: ②
      - name: v1
        mountPath: /data
```

①

볼륨과 시크릿을 추가합니다.

②

컨테이너 마운트 경로를 추가합니다.

- *클레임 이름이 pvc1인 기존 영구 볼륨 v1을 디스크의 배포 구성 dc.json에 추가하려면 /data의 컨테이너 c1에 볼륨을 마운트하고 서버에서 DeploymentConfig 오브젝트를 업데이트합니다.*

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \  
--claim-name=pvc1 --mount-path=/data --containers=c1
```

작은 정보

다음 **YAML**을 적용하여 볼륨을 추가할 수 있습니다.

예 7.3. 영구 볼륨이 추가된 배포 구성 샘플

```

kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 2
            - name: v1
              mountPath: /data

```

1

'pvc1'이라는 영구 볼륨 클레임을 추가합니다.

2

컨테이너 마운트 경로를 추가합니다.

-

모든 복제 컨트롤러에서 수정 버전이 5125c45f9f563인 **Git** 리포지토리 <https://github.com/namespace1/project1>을 기반으로 볼륨 v1을 추가하려면 다음을 실행합니다.



```
$ oc set volume rc --all --add --name=v1 \
  --source='{ "gitRepo": {
    "repository": "https://github.com/namespace1/project1",
    "revision": "5125c45f9f563"
  } }'
```

7.3.5. Pod의 볼륨 및 볼륨 마운트 업데이트

Pod에서 볼륨 및 볼륨 마운트를 수정할 수 있습니다.

프로세스

`--overwrite` 옵션을 사용하여 기존 볼륨을 업데이트합니다.

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

예를 들면 다음과 같습니다.

- 복제 컨트롤러 `r1`의 기존 볼륨 `v1`을 기존 영구 볼륨 클레임 `pvc1`로 교체하려면 다음을 실행합니다.

```
$ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --
claim-name=pvc1
```

작은 정보

다음 **YAML**을 적용하여 볼륨을 교체할 수 있습니다.

예 7.4. **pvc1**이라는 영구 볼륨 클레임이 있는 샘플 복제 컨트롤러

```

kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes:
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - name: v1
              mountPath: /data

```

1

영구 볼륨 클레임을 **pvc1** 로 설정합니다.

• **DeploymentConfig** 오브젝트 **d1** 마운트 옵션을 볼륨 **v1**의 **/opt**로 변경하려면 다음을 실행합니다.

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

작은 정보

다음 YAML을 적용하여 마운트 지점을 변경할 수 있습니다.

예 7.5. 선택으로 설정된 마운트 지점이 설정된 샘플 배포 구성입니다.

```

kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v2
          persistentVolumeClaim:
            claimName: pvc1
        - name: v1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: ①
            - name: v1
              mountPath: /opt

```

①

마운트 지점을 /opt 로 설정합니다.

7.3.6. Pod에서 볼륨 및 볼륨 마운트 제거

Pod에서 볼륨 또는 볼륨 마운트를 제거할 수 있습니다.

Pod 템플릿에서 볼륨을 제거하려면 다음을 수행합니다.

```
$ oc set volume <object_type>/<name> --remove [options]
```

표 7.3. 볼륨 제거에 지원되는 옵션

옵션	설명	기본
--name	볼륨 이름입니다.	
-c, --containers	이름으로 컨테이너를 선택합니다. 문자와 일치하는 와일드카드 '*'를 사용할 수도 있습니다.	'*'
--confirm	한 번에 여러 개의 볼륨을 제거할 것임을 나타냅니다.	
-o, --output	수정된 오브젝트를 서버에서 업데이트하는 대신 표시합니다. 지원되는 값은 json, yaml 입니다.	
--output-version	지정된 버전으로 수정된 오브젝트를 출력합니다.	api-version

예를 들면 다음과 같습니다.

- **DeploymentConfig** 오브젝트 **d1**에서 볼륨 **v1**을 제거하려면 다음을 실행합니다.

```
$ oc set volume dc/d1 --remove --name=v1
```

- **DeploymentConfig** 오브젝트 **d1**의 컨테이너에서 참조하지 않는 경우 **d1**의 컨테이너 **c1**에서 볼륨 **v1**을 마운트 해제하고 볼륨 **v1**을 제거하려면 다음을 실행합니다.

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- 복제 컨트롤러 **r1**의 모든 볼륨을 제거하려면 다음을 실행합니다.

```
$ oc set volume rc/r1 --remove --confirm
```

7.3.7. Pod에서 다양한 용도의 볼륨 구성

볼륨의 루트 대신 볼륨 내부에 **subPath** 값을 지정하는 **volumeMounts.subPath** 속성을 사용하여 단일 Pod에서 다양한 용도로 하나의 볼륨을 공유하도록 볼륨을 구성할 수 있습니다.



참고

예약된 기존 Pod에 **subPath** 매개변수를 추가할 수 없습니다.

프로세스

1. 볼륨의 파일 목록을 보려면 **oc rsh** 명령을 실행합니다.

```
$ oc rsh <pod>
```

출력 예

```
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. **subPath**를 지정합니다.

subPath 매개변수가 포함된 Pod 사양의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
  containers:
    - name: mysql
      image: mysql
      volumeMounts:
        - mountPath: /var/lib/mysql
          name: site-data
          subPath: mysql 1
    - name: php
      image: php
      volumeMounts:
```

```

- mountPath: /var/www/html
  name: site-data
  subPath: html 2
volumes:
- name: site-data
  persistentVolumeClaim:
    claimName: my-site-data

```

1

데이터베이스는 **mysql** 폴더에 저장됩니다.

2

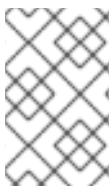
HTML 콘텐츠는 **html** 폴더에 저장됩니다.

7.4. 예상된 볼륨을 사용하여 볼륨 매핑

예상 볼륨은 여러 개의 기존 볼륨 소스를 동일한 디렉터리에 매핑합니다.

다음 유형의 볼륨 소스를 예상할 수 있습니다.

- 보안
- **Config Map**
- **Downward API**



참고

모든 소스는 **Pod**와 동일한 네임스페이스에 있어야 합니다.

7.4.1. 예상 볼륨 이해

예상 볼륨은 해당 볼륨 소스의 조합을 단일 디렉터리에 매핑할 수 있어 사용자는 다음을 수행할 수 있

습니다.

- 여러 보안의 키, 구성 맵, **Downward API** 정보를 사용하여 단일 볼륨을 자동으로 채워 단일 디렉터리를 다양한 정보 소스와 합성할 수 있습니다.
- 여러 보안의 키, 구성 맵, **Downward API** 정보를 사용하여 단일 볼륨을 채우고 각 항목의 경로를 명시적으로 지정하여 해당 볼륨의 콘텐츠를 완전히 제어할 수 있습니다.

중요

Linux 기반 Pod의 보안 컨텍스트에서 **RunAsUser** 권한이 설정된 경우 예상된 파일에는 컨테이너 사용자 소유권을 포함하여 올바른 권한이 설정됩니다. 그러나 **Windows Pod**에 해당하는 **Windows** 동등한 **RunAsUsername** 권한이 설정된 경우 **kubelet**은 예상 볼륨의 파일에 대한 소유권을 올바르게 설정할 수 없습니다.

따라서 **Windows Pod**의 보안 컨텍스트에서 설정된 **RunAsUsername** 권한은 **OpenShift Container Platform**에서 실행되는 **Windows** 예상 볼륨에 대해 지원되지 않습니다.

다음 일반 시나리오에서는 예상 볼륨을 사용하는 방법을 보여줍니다.

구성 맵, 보안, Downward API

예상 볼륨을 사용하여 암호가 포함된 구성 데이터가 있는 컨테이너를 배포할 수 있습니다. 이러한 리소스를 사용하는 애플리케이션은 **Kubernetes**에 **RHOSP(Red Hat OpenStack Platform)**를 배포할 수 있습니다. 서비스를 프로덕션 또는 테스트에 사용하는지에 따라 구성 데이터를 다르게 어셈블해야 할 수 있습니다. Pod에 프로덕션 또는 테스트로 라벨이 지정되면 **Downward API** 선택기 **metadata.labels**를 사용하여 올바른 **RHOSP** 구성을 생성할 수 있습니다.

구성 맵 + 보안

예상 볼륨을 사용하면 구성 데이터 및 암호와 관련된 컨테이너를 배포할 수 있습니다. 예를 들면 **Vault** 암호 파일을 사용하여 암호를 해독하는 몇 가지 민감한 암호화된 작업에서 구성 맵을 실행할 수 있습니다.

구성 맵 + Downward API

예상 볼륨을 사용하면 Pod 이름(**metadata.name** 선택기를 통해 제공)을 포함하여 구성을 생성할 수 있습니다. 그러면 이 애플리케이션에서 IP 추적을 사용하지 않고 소스를 쉽게 확인할 수 있도록 요청과 함께 Pod 이름을 전달할 수 있습니다.

보안 + Downward API

예상 볼륨을 사용하면 보안을 공개키로 사용하여 Pod의 네임스페이스(metadata.namespace 선택기를 통해 제공)를 암호화할 수 있습니다. 이 예제를 사용하면 Operator에서 애플리케이션을 사용하여 암호화된 전송을 사용하지 않고도 네임스페이스 정보를 안전하게 전달할 수 있습니다.

7.4.1.1. Pod 사양의 예

다음은 예상되는 볼륨을 생성하는 Pod 사양의 예입니다.

보안, Downward API, 구성 맵이 있는 Pod

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts: ①
    - name: all-in-one
      mountPath: "/projected-volume" ②
      readOnly: true ③
  volumes: ④
  - name: all-in-one ⑤
    projected:
      defaultMode: 0400 ⑥
      sources:
      - secret:
          name: mysecret ⑦
          items:
          - key: username
            path: my-group/my-username ⑧
      - downwardAPI: ⑨
          items:
          - path: "labels"
            fieldRef:
              fieldPath: metadata.labels
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: container-test
              resource: limits.cpu
      - configMap: ⑩
          name: myconfigmap
          items:
          - key: config
            path: my-group/my-config
            mode: 0777 ⑪
  
```

1

보안이 필요한 각 컨테이너에 **volumeMounts** 섹션을 추가합니다.

2

보안이 표시될 미사용 디렉터리의 경로를 지정합니다.

3

readOnly를 **true**로 설정합니다.

4

volumes 블록을 추가하여 각 예상 볼륨 소스를 나열합니다.

5

볼륨에 이름을 지정합니다.

6

파일에 대한 실행 권한을 설정합니다.

7

보안을 추가합니다. 보안 오브젝트의 이름을 입력합니다. 사용하려는 모든 시크릿을 나열해야 합니다.

8

mountPath 아래에 보안 파일의 경로를 지정합니다. 여기에서 보안 파일은 **/projected-volume/my-group/my-username**에 있습니다.

9

Downward API 소스를 추가합니다.

10

구성 맵 소스를 추가합니다.

11



참고

Pod에 컨테이너가 여러 개 있는 경우 각 컨테이너에 **volumeMounts** 섹션이 있어야 하지만 **volumes** 섹션은 하나만 있으면 됩니다.

기본이 아닌 권한 모드가 설정된 보안이 여러 개인 **Pod**

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
    defaultMode: 0755
    sources:
    - secret:
      name: mysecret
      items:
      - key: username
        path: my-group/my-username
    - secret:
      name: mysecret2
      items:
      - key: password
        path: my-group/my-password
        mode: 511

```



참고

defaultMode는 각 볼륨 소스가 아닌 예상 수준에서만 지정할 수 없습니다. 하지만 위에서 설명한 대로 개별 예상마다 **mode**를 명시적으로 설정할 수 있습니다.

7.4.1.2. 경로 지정 고려 사항

구성된 경로가 동일할 때 키 간 충돌

동일한 경로를 사용하여 여러 개의 키를 구성하면 **Pod 사양이 유효한 것으로 승인되지 않습니다**. 다음 예제에서 **mysecret** 및 **myconfigmap**에 지정된 경로는 동일합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
          - key: username
            path: my-group/data
      - configMap:
          name: myconfigmap
          items:
          - key: config
            path: my-group/data
  
```

블록 파일 경로와 관련된 다음 상황을 고려하십시오.

구성된 경로가 없는 키 간 충돌

발생할 수 있는 유일한 런타임 검증은 **Pod** 생성 시 모든 경로가 알려진 경우이며 위의 시나리오와 유사합니다. 이 경우에 해당하지 않으면 충돌이 발생할 때 최근 지정된 리소스가 이전의 모든 리소스를 덮어씁니다(**Pod** 생성 후 업데이트된 리소스 포함).

하나의 경로는 명시적이고 다른 경로는 자동으로 예상될 때의 충돌

사용자가 지정한 경로가 자동 예상 데이터와 일치하여 충돌이 발생하는 경우 위에서와 마찬가지로 자동 예상 데이터가 이전의 모든 리소스를 덮어씁니다.

7.4.2. Pod의 예상 블록 구성

예상 볼륨을 생성할 때는 예상 볼륨 이해에 설명된 볼륨 파일 경로 상황을 고려하십시오.

다음 예제에서는 예상 볼륨을 사용하여 기존 보안 볼륨 소스를 마운트하는 방법을 보여줍니다. 이 단계를 사용하여 로컬 파일에서 사용자 이름 및 암호 보안을 생성할 수 있습니다. 그런 다음 예상 볼륨을 사용하여 하나의 컨테이너를 실행하는 Pod를 생성하여 동일한 공유 디렉터리에 보안을 마운트합니다.

사용자 이름 및 암호 값은 **base64** 로 인코딩된 유효한 문자열일 수 있습니다.

다음 예제에서는 **admin**을 **base64** 형식으로 보여줍니다.

```
$ echo -n "admin" | base64
```

출력 예

```
YWRtaW4=
```

다음 예제에서는 암호 **1f2d1e2e67df** 를 **base64**로 보여줍니다.

```
$ echo -n "1f2d1e2e67df" | base64
```

출력 예

```
MWYyZDFIMmU2N2Rm
```

프로세스

예상 볼륨을 사용하여 기존 보안 볼륨 소스를 마운트합니다.

1.

시크릿을 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성하고 암호 및 사용자 정보를 적절하게 교체합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
```

b.

다음 명령을 사용하여 보안을 생성합니다.

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

예를 들면 다음과 같습니다.

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

출력 예

```
secret "mysecret" created
```

c.

다음 명령을 사용하여 보안이 생성되었는지 확인할 수 있습니다.

```
$ oc get secret <secret-name>
```

예를 들면 다음과 같습니다.

```
$ oc get secret mysecret
```

출력 예

```

NAME      TYPE      DATA      AGE
mysecret  Opaque    2          17h

```

```
$ oc get secret <secret-name> -o yaml
```

예를 들면 다음과 같습니다.

```
$ oc get secret mysecret -o yaml
```

```

apiVersion: v1
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
  name: mysecret
  namespace: default
  resourceVersion: "2107"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque

```

2.

예상 볼륨이 포함된 Pod를 생성합니다.

a.

volumes 섹션을 포함하여 다음과 유사한 **YAML** 파일을 생성합니다.

```

kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true

```



```

securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop:
    - ALL
volumes:
- name: all-in-one
  projected:
    sources:
    - secret:
        name: mysecret ①

```

①

생성한 보안의 이름입니다.

b.

구성 파일에서 **Pod**를 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f secret-pod.yaml
```

출력 예

```
pod "test-projected-volume" created
```

3.

Pod 컨테이너가 실행 중인지 확인한 후 **Pod** 변경 사항을 확인합니다.

```
$ oc get pod <name>
```

예를 들면 다음과 같습니다.

```
$ oc get pod test-projected-volume
```

출력은 다음과 유사합니다.

출력 예

```

NAME                READY  STATUS   RESTARTS  AGE
test-projected-volume 1/1    Running  0          14s

```

4.

다른 터미널에서 `oc exec` 명령을 사용하여 실행 중인 컨테이너에 셸을 엽니다.

```
$ oc exec -it <pod> <command>
```

예를 들면 다음과 같습니다.

```
$ oc exec -it test-projected-volume -- /bin/sh
```

5.

셸에서 `projected-volumes` 디렉터리에 예상 소스가 포함되어 있는지 확인합니다.

```
/# ls
```

출력 예

```

bin          home        root        tmp
dev          proc        run         usr
etc          projected-volume sys         var

```

7.5. 컨테이너에서 API 오브젝트를 사용하도록 허용

Downward API는 OpenShift Container Platform에 결합하지 않고도 컨테이너에서 API 오브젝트에 대한 정보를 사용할 수 있는 메커니즘입니다. 이러한 정보에는 Pod 이름, 네임스페이스, 리소스 값이 포함

됩니다. 컨테이너는 환경 변수 또는 볼륨 플러그인을 사용하여 **Downward API**의 정보를 사용할 수 있습니다.

7.5.1. Downward API를 사용하여 컨테이너에 Pod 정보 노출

Downward API에는 Pod 이름, 프로젝트, 리소스 값과 같은 정보가 포함됩니다. 컨테이너는 환경 변수 또는 볼륨 플러그인을 사용하여 **Downward API**의 정보를 사용할 수 있습니다.

Pod 내 필드는 **FieldRef API** 유형을 사용하여 선택합니다. **FieldRef**에는 두 개의 필드가 있습니다.

필드	설명
fieldPath	Pod와 관련하여 선택할 필드의 경로입니다.
apiVersion	fieldPath 선택기를 해석할 API 버전입니다.

현재 **v1 API**에서 유효한 선택기는 다음과 같습니다.

선택기	설명
metadata.name	Pod의 이름입니다. 이는 환경 변수와 볼륨 모두에서 지원됩니다.
metadata.namespace	Pod의 네임스페이스입니다. 환경 변수와 볼륨 모두에서 지원됩니다.
metadata.labels	Pod의 라벨입니다. 볼륨에서만 지원되며 환경 변수에서는 지원되지 않습니다.
metadata.annotations	Pod의 주석입니다. 볼륨에서만 지원되며 환경 변수에서는 지원되지 않습니다.
status.podIP	Pod의 IP입니다. 환경 변수에서만 지원되며 볼륨에서는 지원되지 않습니다.

apiVersion 필드가 지정되지 않은 경우 기본값은 포함된 **Pod** 템플릿의 **API** 버전입니다.

7.5.2. Downward API를 사용하여 컨테이너 값을 사용하는 방법 이해

컨테이너는 환경 변수 또는 볼륨 플러그인을 사용하여 **API** 값을 사용할 수 있습니다. 선택하는 메서드

에 따라 컨테이너에서 다음을 사용할 수 있습니다.

- **Pod 이름**
- **Pod 프로젝트/네임스페이스**
- **Pod 주석**
- **Pod 라벨**

블록 플러그인만 사용하여 주석 및 레이블을 사용할 수 있습니다.

7.5.2.1. 환경 변수를 사용하여 컨테이너 값 사용

컨테이너의 환경 변수를 사용할 때는 변수 값을 **value** 필드에서 지정하는 리터럴 값 대신 **FieldRef** 소스에서 제공하도록 **EnvVar** 유형의 **valueFrom** 필드(**EnvVarSource** 유형)를 사용합니다.

프로세스에 변수 값이 변경되었음을 알리는 방식으로 프로세스를 시작한 후에는 환경 변수를 업데이트할 수 없으므로 **Pod**의 상수 특성만 이러한 방식으로 사용할 수 있습니다. 환경 변수를 사용하여 지원되는 필드는 다음과 같습니다.

- **Pod 이름**
- **Pod 프로젝트/네임스페이스**

프로세스

1. 컨테이너에서 사용할 환경 변수가 포함된 새 **Pod** 사양을 생성합니다.
 - a. 다음과 유사한 **pod.yaml** 파일을 생성합니다.

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

```

metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
  restartPolicy: Never
# ...

```

b.

`pod.yaml` 파일에서 Pod를 생성합니다.

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

검증

- 컨테이너의 로그에 `MY_POD_NAME` 및 `MY_POD_NAMESPACE` 값이 있는지 확인합니다.

```
$ oc logs -p dapi-env-test-pod
```

7.5.2.2. 볼륨 플러그인을 사용하여 컨테이너 값 사용

컨테이너는 볼륨 플러그인을 사용하여 API 값을 사용할 수 있습니다.

컨테이너는 다음을 사용할 수 있습니다.

- Pod 이름
- Pod 프로젝트/네임스페이스
- Pod 주석

- **Pod 라벨**

프로세스

볼륨 플러그인을 사용하려면 다음을 수행합니다.

1. 컨테이너에서 사용할 환경 변수가 포함된 새 **Pod** 사양을 생성합니다.
 - a. 다음과 유사한 **volume-pod.yaml** 파일을 생성합니다.

```

kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  containers:
    - name: volume-test-container
      image: gcr.io/google_containers/busybox
      command: ["sh", "-c", "cat /tmp/etc/pod_labels/tmp/etc/pod_annotations"]
      volumeMounts:
        - name: podinfo
          mountPath: /tmp/etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        defaultMode: 420
        items:
          - fieldRef:
              fieldPath: metadata.name
              path: pod_name
          - fieldRef:
              fieldPath: metadata.namespace
              path: pod_namespace
          - fieldRef:
              fieldPath: metadata.labels
              path: pod_labels
          - fieldRef:
              fieldPath: metadata.annotations
              path: pod_annotations
      restartPolicy: Never
# ...

```

- b. **volume-pod.yaml** 파일에서 **Pod**를 생성합니다.

```
$ oc create -f volume-pod.yaml
```

검증

- 컨테이너의 로그를 확인하고 구성된 필드가 있는지 확인합니다.

```
$ oc logs -p dapi-volume-test-pod
```

출력 예

```
cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api
```

7.5.3. Downward API를 사용하여 컨테이너 리소스를 사용하는 방법 이해

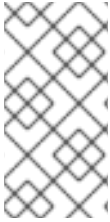
Pod를 생성할 때 이미지 및 애플리케이션 작성자가 특정 환경에 대한 이미지를 올바르게 생성할 수 있도록 **Downward API**를 사용하여 컴퓨팅 리소스 요청 및 제한에 대한 정보를 삽입할 수 있습니다.

환경 변수 또는 볼륨 플러그인을 사용하여 이 작업을 수행할 수 있습니다.

7.5.3.1. 환경 변수를 사용하여 컨테이너 리소스 사용

Pod를 생성할 때는 **Downward API**에서 환경 변수를 사용하여 컴퓨팅 리소스 요청 및 제한에 대한 정보를 삽입할 수 있습니다.

Pod 구성을 생성할 때 **spec.container** 필드의 **resources** 필드에 해당하는 환경 변수를 지정합니다.



참고

리소스 제한이 컨테이너 구성에 포함되지 않은 경우 **Downward API**의 기본값은 노드의 **CPU** 및 메모리 할당 가능 값으로 설정됩니다.

프로세스

1.

삽입할 리소스가 포함된 새 **Pod** 사양을 생성합니다.

a.

다음과 유사한 **pod.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox:1.24
    command: [ "/bin/sh", "-c", "env" ]
    resources:
      requests:
        memory: "32Mi"
        cpu: "125m"
      limits:
        memory: "64Mi"
        cpu: "250m"
    env:
    - name: MY_CPU_REQUEST
      valueFrom:
        resourceFieldRef:
          resource: requests.cpu
    - name: MY_CPU_LIMIT
      valueFrom:
        resourceFieldRef:
          resource: limits.cpu
    - name: MY_MEM_REQUEST
      valueFrom:
        resourceFieldRef:
          resource: requests.memory
    - name: MY_MEM_LIMIT
      valueFrom:
        resourceFieldRef:
          resource: limits.memory
# ...
```

b.

pod.yaml 파일에서 **Pod**를 생성합니다.


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

7.5.3.2. 볼륨 플러그인을 사용하여 컨테이너 리소스 사용

Pod를 생성할 때 Downward API를 사용하여 볼륨 플러그인을 사용하여 컴퓨팅 리소스 요청 및 제한에 대한 정보를 삽입할 수 있습니다.

Pod 구성을 생성할 때 `spec.volumes.downwardAPI.items` 필드를 사용하여 `spec.resources` 필드에 해당하는 원하는 리소스를 설명합니다.



참고

리소스 제한이 컨테이너 구성에 포함되지 않은 경우 Downward API의 기본값은 노드의 CPU 및 메모리 할당 가능 값으로 설정됩니다.

프로세스

1.

삽입할 리소스가 포함된 새 Pod 사양을 생성합니다.

a.

다음과 유사한 `pod.yaml` 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat /etc/cpu_limit; fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e /etc/mem_limit ]]; then cat /etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat /etc/mem_request; fi; sleep 5; done"]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
```

```

- name: podinfo
  downwardAPI:
    items:
      - path: "cpu_limit"
        resourceFieldRef:
          containerName: client-container
          resource: limits.cpu
      - path: "cpu_request"
        resourceFieldRef:
          containerName: client-container
          resource: requests.cpu
      - path: "mem_limit"
        resourceFieldRef:
          containerName: client-container
          resource: limits.memory
      - path: "mem_request"
        resourceFieldRef:
          containerName: client-container
          resource: requests.memory
# ...

```

b.

`volume-pod.yaml` 파일에서 Pod를 생성합니다.

```
$ oc create -f volume-pod.yaml
```

7.5.4. Downward API를 사용하여 보안 사용

Pod를 생성할 때 이미지 및 애플리케이션 작성자가 특정 환경에 대한 이미지를 생성할 수 있도록 Downward API를 사용하여 보안을 삽입할 수 있습니다.

프로세스

1.

삽입할 시크릿을 생성합니다.

a.

다음과 유사한 `secret.yaml` 파일을 생성합니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: <password>
  username: <username>
type: kubernetes.io/basic-auth

```

b.

secret.yaml 파일에서 보안 오브젝트를 생성합니다.

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

2.

위 **Secret** 오브젝트의 **username** 필드를 참조하는 **Pod**를 생성합니다.

a.

다음과 유사한 **pod.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: username
  restartPolicy: Never
# ...
```

b.

pod.yaml 파일에서 **Pod**를 생성합니다.

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

검증

•

컨테이너의 로그에 **MY_SECRET_USERNAME** 값이 있는지 확인합니다.

```
$ oc logs -p dapi-env-test-pod
```

7.5.5. Downward API를 사용하여 구성 맵 사용

Pod를 생성할 때 이미지 및 애플리케이션 작성자가 특정 환경에 대한 이미지를 생성할 수 있도록 **Downward API**를 사용하여 구성 맵 값을 삽입할 수 있습니다.

프로세스

1.

삽입할 값으로 구성 맵을 생성합니다.

a.

다음과 유사한 **configmap.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

b.

configmap.yaml 파일에서 구성 맵을 생성합니다.

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

2.

위 구성 맵을 참조하는 **Pod**를 생성합니다.

a.

다음과 유사한 **pod.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: env-test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: MY_CONFIGMAP_VALUE
      valueFrom:
        configMapKeyRef:
          name: myconfigmap
          key: mykey
    restartPolicy: Always
# ...
```

b.

pod.yaml 파일에서 **Pod**를 생성합니다.

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

- 컨테이너의 로그에 **MY_CONFIGMAP_VALUE** 값이 있는지 확인합니다.

```
$ oc logs -p dapi-env-test-pod
```

7.5.6. 환경 변수 참조

Pod를 생성할 때 **\$()** 구문을 사용하여 이전에 정의한 환경 변수의 값을 참조할 수 있습니다. 환경 변수 참조를 확인할 수 없는 경우에는 값이 제공된 문자열로 그대로 유지됩니다.

프로세스

1. 기존 환경 변수를 참조하는 **Pod**를 생성합니다.
 - a. 다음과 유사한 **pod.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_EXISTING_ENV
          value: my_value
        - name: MY_ENV_VAR_REF_ENV
          value: $(MY_EXISTING_ENV)
      restartPolicy: Never
  # ...
```

- b. **pod.yaml** 파일에서 **Pod**를 생성합니다.

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

검증

- 컨테이너의 로그에 **MY_ENV_VAR_REF_ENV** 값이 있는지 확인합니다.

```
$ oc logs -p dapi-env-test-pod
```

7.5.7. 환경 변수 참조 이스케이프

Pod를 생성할 때 이중 달러 기호를 사용하여 환경 변수 참조를 이스케이프할 수 있습니다. 그러면 해당 값이 제공된 값의 단일 달러 기호 버전으로 설정됩니다.

프로세스

1. 기존 환경 변수를 참조하는 **Pod**를 생성합니다.
 - a. 다음과 유사한 **pod.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$SOME_OTHER_ENV
      restartPolicy: Never
# ...
```

- b. **pod.yaml** 파일에서 **Pod**를 생성합니다.

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

검증

- 컨테이너의 로그에 **MY_NEW_ENV** 값이 있는지 확인합니다.

```
$ oc logs -p dapi-env-test-pod
```

7.6. OPENSIFT CONTAINER PLATFORM 컨테이너에 또한 해당 컨테이너에서 파일 복사

CLI에서 **rsync** 명령을 사용하여 컨테이너의 원격 디렉터리에서 또는 원격 디렉터리로 로컬 파일을 복사할 수 있습니다.

7.6.1. 파일을 복사하는 방법 이해

oc rsync 명령 또는 원격 동기화는 백업 및 복원을 위해 **Pod**에서 및 **Pod**로 데이터베이스 아카이브를 복사하는 유용한 툴입니다. 실행 중인 **Pod**에서 소스 파일의 핫 리로드를 지원하는 경우 개발 디버깅을 위해 **oc rsync**를 사용하여 소스 코드 변경 사항을 실행 중인 **Pod**에 복사할 수도 있습니다.

```
$ oc rsync <source> <destination> [-c <container>]
```

7.6.1.1. 요구사항

복사 소스 지정

oc rsync 명령의 소스 인수는 로컬 디렉터리 또는 **pod** 디렉터를 가리켜야 합니다. 개별 파일은 지원되지 않습니다.

Pod 디렉터를 지정할 때는 디렉터리 이름 앞에 **Pod** 이름을 붙여야 합니다.

```
<pod name>:<dir>
```

디렉터리 이름이 경로 구분자(/)로 끝나는 경우 디렉터리의 콘텐츠만 대상에 복사됩니다. 그렇지 않으면 디렉터리 및 해당 콘텐츠가 대상에 복사됩니다.

복사 대상 지정

oc rsync 명령의 대상 인수는 디렉터를 가리켜야 합니다. 해당 디렉터리가 존재하지 않지만 **rsync**가 복사에 사용되는 경우 사용자를 위해 디렉터리가 생성됩니다.

대상의 파일 삭제

--delete 플래그는 로컬 디렉터리에 없는 파일을 원격 디렉터리에서 삭제하는 데 사용할 수 있습니다.

파일 변경 시 연속 동기화

--watch 옵션을 사용하면 명령에서 파일 시스템 변경의 소스 경로를 모니터링하고 변경이 발생하면 변경 사항을 동기화합니다. 이 인수를 사용하면 명령이 영구적으로 실행됩니다.

빠르게 변경되는 파일 시스템으로 인해 동기화를 연속으로 호출하지 않도록 동기화는 잠시 후에 수행됩니다.

--watch 옵션을 사용할 때의 동작은 일반적으로 **oc rsync**에 전달되는 인수를 포함하여 **oc rsync**를 수동으로 반복해서 호출하는 것과 사실상 동일합니다. 따라서 **-delete**와 같이 **oc rsync**를 수

동으로 호출하는 데 사용하는 것과 같은 플래그를 통해 해당 동작을 제어할 수 있습니다.

7.6.2. 컨테이너에서 또는 컨테이너에 파일 복사

컨테이너에서 또는 컨테이너에 로컬 파일 복사 지원 기능은 CLI에 빌드됩니다.

사전 요구 사항

oc rsync로 작업할 때 다음 사항에 유의하십시오.

- **rsync**가 설치되어 있어야 합니다. **oc rsync** 명령은 클라이언트 머신 및 원격 컨테이너에 있는 경우 로컬 **rsync** 툴을 사용합니다.

rsync가 로컬이나 원격 컨테이너에 없는 경우 **tar** 아카이브는 로컬에 생성된 후 컨테이너로 전송되며, 여기에서 **tar** 유틸리티를 통해 파일이 추출됩니다. 원격 컨테이너에서 **tar**를 사용할 수 없는 경우 복사가 실패합니다.

tar 복사 방법에서는 **oc rsync**와 동일한 기능을 제공하지 않습니다. 예를 들어 **oc rsync**는 대상 디렉터리가 존재하지 않는 경우 대상 디렉터를 생성하고 소스와 대상 간에 다른 파일만 보냅니다.



참고

Windows에서는 **oc rsync** 명령과 함께 사용할 수 있도록 **cwRsync** 클라이언트를 설치하고 **PATH**에 추가해야 합니다.

프로세스

- 로컬 디렉터를 Pod 디렉터리에 복사하려면 다음을 수행합니다.

```
$ oc rsync <local-dir> <pod-name>:/<remote-dir> -c <container-name>
```

예를 들면 다음과 같습니다.

```
$ oc rsync /home/user/source devpod1234:/src -c user-container
```


- Pod 디렉토리를 로컬 디렉토리에 복사하려면 다음을 수행합니다.

```
$ oc rsync devpod1234:/src /home/user/source
```

출력 예

```
$ oc rsync devpod1234:/src/status.txt /home/user/
```

7.6.3. 고급 Rsync 기능 사용

`oc rsync` 명령은 표준 `rsync`보다 적은 수의 명령줄 옵션을 표시합니다. `oc rsync`에서 사용할 수 없는 표준 `rsync` 명령줄 옵션(예: `--exclude-from=FILE` 옵션)을 사용하려는 경우 표준 `rsync`의 `--rsh(-e)` 옵션 또는 `RSYNC_RSH` 환경 변수를 다음과 같이 해결 방법으로 사용할 수 있습니다.

```
$ rsync --rsh='oc rsh' --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

또는 다음을 수행합니다.

`RSYNC_RSH` 변수를 내보냅니다.

```
$ export RSYNC_RSH='oc rsh'
```

그런 다음 `rsync` 명령을 실행합니다.

```
$ rsync --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

위의 두 가지 예 모두 `oc rsh`를 원격 셸 프로그램으로 사용하여 원격 Pod에 연결할 수 있도록 표준 `rsync`를 구성하고 `oc rsync`를 실행하는 대신 사용할 수 있습니다.

7.7. OPENSIFT CONTAINER PLATFORM 컨테이너에서 원격 명령 실행

CLI를 사용하여 OpenShift Container Platform 컨테이너에서 원격 명령을 실행할 수 있습니다.

7.7.1. 컨테이너에서 원격 명령 실행

원격 컨테이너 명령 실행을 위한 지원은 **CLI**에 빌드됩니다.

프로세스

컨테이너에서 명령을 실행하려면 다음을 수행합니다.

```
$ oc exec <pod> [-c <container>] -- <command> [<arg_1> ... <arg_n>]
```

예를 들면 다음과 같습니다.

```
$ oc exec mypod date
```

출력 예

```
Thu Apr 9 02:21:53 UTC 2015
```



중요

보안상의 이유로 cluster-admin 사용자가 명령을 실행하는 경우를 제외하고 권한 있는 컨테이너에 액세스하면 **oc exec** 명령이 작동하지 않습니다.

7.7.2. 클라이언트에서 원격 명령을 시작하는 프로토콜

클라이언트는 **Kubernetes API** 서버에 대한 요청을 발행하여 컨테이너에서 원격 명령 실행을 시작합니다.

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

위 URL에서

- **<node_name>**은 노드의 **FQDN**입니다.

- **<namespace>**는 대상 Pod의 프로젝트입니다.
- **<pod>**는 대상 Pod의 이름입니다.
- **<container>**는 대상 컨테이너의 이름입니다.
- **<command>**는 실행하기를 원하는 명령입니다.

예를 들면 다음과 같습니다.

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

또한 클라이언트는 요청에 매개변수를 추가하여 다음에 대한 여부를 표시할 수 있습니다.

- 클라이언트에서 원격 컨테이너의 명령(**stdin**)에 입력을 보내야 합니다.
- 클라이언트의 터미널이 **TTY**입니다.
- 원격 컨테이너의 명령에서 **stdout**의 출력을 클라이언트로 보내야 합니다.
- 원격 컨테이너의 명령에서 **stderr**의 출력을 클라이언트로 보내야 합니다.

클라이언트는 **API** 서버로 **exec** 요청을 보낸 후 다중 스트림을 지원하는 연결로 연결을 업그레이드합니다. 현재 구현에서는 **HTTP/2** 를 사용합니다.

클라이언트는 **stdin**, **stdout**, **stderr**에 대해 각각 하나의 스트림을 생성합니다. 클라이언트는 스트림을 구분하기 위해 스트림의 **streamType** 헤더를 **stdin**, **stdout**, **stderr** 중 하나로 설정합니다.

클라이언트는 원격 명령 실행 요청을 완료하면 모든 스트림, 업그레이드된 연결, 기본 연결을 종료합니다.

7.8. 포트 전달을 사용하여 컨테이너의 애플리케이션에 액세스

OpenShift Container Platform에서는 Pod로 포트를 전달할 수 있습니다.

7.8.1. 포트 전달 이해

CLI를 사용하여 하나 이상의 로컬 포트를 Pod로 전달할 수 있습니다. 이 경우 지정된 포트 또는 임의의 포트에서 로컬로 수신 대기하고 Pod의 지정된 포트와 데이터를 주고받을 수 있습니다.

포트 전달 기능을 위한 지원은 CLI에 빌드되어 있습니다.

```
$ oc port-forward <pod> [<local_port>:<remote_port> [...<local_port_n>:<remote_port_n>]
```

CLI는 사용자가 지정한 각 로컬 포트에서 수신 대기하고 아래에 설명된 프로토콜을 사용하여 전달합니다.

포트는 다음 형식을 사용하여 지정할 수 있습니다.

5000	클라이언트는 포트 5000에서 로컬로 수신 대기하고 Pod의 5000으로 전달합니다.
6000:5000	클라이언트는 포트 6000에서 로컬로 수신 대기하고 Pod의 5000으로 전달합니다.
:5000 또는 0:5000	클라이언트는 사용 가능한 로컬 포트를 선택하고 Pod의 5000으로 전달합니다.

OpenShift Container Platform은 클라이언트의 포트 전달 요청을 처리합니다. 요청이 수신되면 OpenShift Container Platform에서 응답을 업그레이드하고 클라이언트에서 포트 전달 스트림을 생성할 때까지 기다립니다. OpenShift Container Platform에서 새 스트림을 수신하면 스트림과 Pod의 포트 간 데이터를 복사합니다.

구조적으로 Pod의 포트로 전달할 수 있는 옵션이 있습니다. 지원되는 OpenShift Container Platform 구현에서는 노드 호스트에서 직접 nsenter를 호출하여 Pod의 네트워크 네임스페이스에 입력한 다음 socat을 호출하여 스트림과 Pod 포트 사이의 데이터를 복사합니다. 그러나 사용자 정의 구현에는 nsenter 및 socat을 실행하는 helper Pod 실행을 포함할 수 있으므로 이러한 바이너리를 호스트에 설치할 필요가 없습니다.

7.8.2. 포트 전달 사용

CLI를 사용하여 하나 이상의 로컬 포트를 **Pod**로 포트 전달할 수 있습니다.

프로세스

다음 명령을 사용하여 **Pod**의 지정된 포트에서 수신 대기합니다.

```
$ oc port-forward <pod> [<local_port>:<remote_port> [...<local_port_n>:<remote_port_n>]
```

예를 들면 다음과 같습니다.

- 다음 명령을 사용하여 포트 **5000** 및 **6000**에서 로컬로 수신 대기하고 **Pod**의 포트 **5000** 및 **6000**에서 또는 해당 포트로 데이터를 전달합니다.

```
$ oc port-forward <pod> 5000 6000
```

출력 예

```
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
Forwarding from 127.0.0.1:6000 -> 6000
Forwarding from [::1]:6000 -> 6000
```

- 다음 명령을 사용하여 포트 **8888**에서 로컬로 수신 대기하고 **Pod**의 **5000**으로 전달합니다.

```
$ oc port-forward <pod> 8888:5000
```

출력 예

```
Forwarding from 127.0.0.1:8888 -> 5000
Forwarding from [::1]:8888 -> 5000
```

- 다음 명령을 사용하여 사용 가능한 포트에서 로컬로 수신 대기하고 Pod의 5000으로 전달합니다.

```
$ oc port-forward <pod> :5000
```

출력 예

```
Forwarding from 127.0.0.1:42390 -> 5000  
Forwarding from [::1]:42390 -> 5000
```

또는 다음을 수행합니다.

```
$ oc port-forward <pod> 0:5000
```

7.8.3. 클라이언트에서 포트 전달을 시작하는 프로토콜

클라이언트는 **Kubernetes API** 서버에 대한 요청을 발행하여 Pod로의 포트 전달을 시작합니다.

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

위 URL에서

- <node_name>은 노드의 FQDN입니다.
- <namespace>는 대상 Pod의 네임스페이스입니다.
- <pod>는 대상 Pod의 이름입니다.

예를 들면 다음과 같습니다.

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

클라이언트는 **API** 서버로 포트 전달 요청을 보낸 후 다중 스트림을 지원하는 연결로 연결을 업그레이드합니다. 현재 구현에서는 **Hyptertext Transfer Protocol 2(HTTP/2)** 를 사용합니다.

클라이언트는 **Pod**에 대상 포트가 포함된 **port** 헤더를 사용하여 스트림을 생성합니다. 스트림에 기록된 모든 데이터는 **kubelet**을 통해 대상 **Pod** 및 포트에 전달됩니다. 마찬가지로 이렇게 전달된 연결에 대해 **Pod**에서 전송되는 모든 데이터는 클라이언트의 동일한 스트림으로 다시 전달됩니다.

클라이언트는 포트 전달 요청을 완료하면 모든 스트림, 업그레이드된 연결, 기본 연결을 종료합니다.

7.9. 컨테이너의 SYSCTL 사용

sysctl 설정은 **Kubernetes**를 통해 노출되므로 사용자가 런타임에 특정 커널 매개변수를 수정할 수 있습니다. 네임스페이스가 지정된 **sysctl**만 **Pod**에 독립적으로 설정할 수 있습니다. **sysctl**에 노드 수준이라는 네임스페이스가 지정되지 않은 경우 **Node Tuning Operator**를 사용하여 와 같은 **sysctl**을 설정하는 다른 방법을 사용해야 합니다.

네트워크 **sysctl**은 **sysctl**의 특수 범주입니다. 네트워크 **sysctl**은 다음과 같습니다.

- 시스템 전체 **sysctl**(예: **net.ipv4.ip_local_port_range**)은 모든 네트워킹에 유효합니다. 노드의 각 **Pod**에 대해 개별적으로 이러한 값을 설정할 수 있습니다.
- 지정된 **Pod**의 특정 추가 네트워크 인터페이스에만 적용되는 인터페이스별 **sysctl**(예: **net.ipv4.conf.IFNAME.accept_local**)입니다. 각 추가 네트워크 구성에 대해 독립적으로 설정할 수 있습니다. 네트워크 인터페이스가 생성된 후 **tuning-cni**의 구성을 사용하여 이러한 설정을 설정합니다.

또한 기본적으로 안전한 것으로 간주되는 **sysctl**만 허용 목록에 포함됩니다. 노드의 다른 안전하지 않은 **sysctl**을 사용자에게 제공하도록 수동으로 활성화할 수 있습니다.

추가 리소스

sysctl을 설정하고 노드 수준이 아닌 경우 **Node Tuning Operator**를 사용하여 섹션에서 이 절차에 대한 정보를 찾을 수 있습니다.

7.9.1. sysctl 정보

Linux에서 **sysctl** 인터페이스를 사용하면 관리자가 런타임에 커널 매개변수를 수정할 수 있습니다. 매개변수는 `/proc/sys/` 가상 프로세스 파일 시스템에서 사용할 수 있습니다. 해당 매개변수는 다음과 같이 다양한 하위 시스템에 적용됩니다.

- **kernel**(공용 접두사: **kernel**.)
- **네트워킹**(공용 접두사: **net**.)
- **가상 메모리**(공용 접두사: **vm**.)
- **MDADM**(공용 접두사: **dev**.)

[커널 설명서](#)에 더 많은 하위 시스템이 설명되어 있습니다. 모든 매개변수 목록을 가져오려면 다음을 실행합니다.

```
$ sudo sysctl -a
```

7.9.2. 네임스페이스 및 노드 수준 **sysctls**

대다수의 **sysctl**은 Linux 커널에 네임스페이스가 지정됩니다. 즉 노드의 각 **Pod**에 개별적으로 설정할 수 있습니다. Kubernetes 내의 **Pod** 컨텍스트에서 **sysctl**에 액세스하려면 네임스페이스를 지정해야 합니다.

다음 **sysctl**은 네임스페이스로 알려져 있습니다.

- **kernel.shm***
- **kernel.msg***
- **kernel.sem**
- **FS.mqueue.***

또한 **net.*** 그룹의 대부분의 **sysctl**은 네임스페이스로 알려져 있습니다. 해당 네임스페이스 채택은 커널 버전 및 배포자에 따라 다릅니다.

네임스페이스가 지정되지 않은 **sysctl**은 노드 수준 이라고 하며 노드의 기본 **Linux** 배포(예: **/etc/sysctls.conf** 파일을 수정하거나 권한 있는 컨테이너에 데몬 세트를 사용하여) 클러스터 관리자가 수동으로 설정해야 합니다. **Node Tuning Operator**를 사용하여 노드 수준 **sysctl**을 설정할 수 있습니다.



참고

특수 **sysctl**이 있는 노드를 테인트로 표시하는 것이 좋습니다. 이러한 **sysctl** 설정이 필요한 노드에만 **Pod**를 예약하십시오. 테인트 및 허용 오차 기능을 사용하여 노드를 표시합니다.

7.9.3. 안전한 **sysctl** 및 안전하지 않은 **sysctl**

sysctl은 안전한 **sysctl** 및 안전하지 않은 **sysctl**로 그룹화됩니다.

시스템 전체 **sysctl**을 안전한 것으로 간주하려면 네임스페이스를 지정해야 합니다. 네임스페이스가 지정된 **sysctl**을 사용하면 네임스페이스와 **Pod** 간에 격리됩니다. 하나의 **Pod**에 **sysctl**을 설정하는 경우 다음 항목을 추가할 수 없습니다.

- 노드의 다른 **Pod**에 영향을 미침
- 노드 상태 손상
- **Pod**의 리소스 제한을 벗어나는 **CPU** 또는 메모리 리소스 확보



참고

sysctl이 안전한 것으로 간주되기 위해서는 네임스페이스만으로는 충분하지 않습니다.

OpenShift Container Platform의 허용 목록에 추가되지 않은 **sysctl**은 **OpenShift Container Platform**에서 안전하지 않은 것으로 간주됩니다.

안전하지 않은 **sysctl**은 기본적으로 허용되지 않습니다. 시스템 전체 **sysctl**의 경우 클러스터 관리자가 노트별로 수동으로 활성화해야 합니다. 안전하지 않은 **sysctl**이 비활성화된 **Pod**는 예약은 되지만 시작되지 않습니다.



참고

인터페이스별 안전하지 않은 **sysctl**을 수동으로 활성화할 수 없습니다.

OpenShift Container Platform은 다음과 같은 시스템 수준 및 인터페이스 관련 안전한 **sysctl**을 허용된 안전 목록에 추가합니다.

표 7.4. 시스템 전체의 안전한 **sysctl**

sysctl	설명
kernel.shm_rmid_forced	1 로 설정하면 현재 IPC 네임스페이스의 모든 공유 메모리 개체가 자동으로 IPC_RMID 를 사용해야 합니다. 자세한 내용은 shm_rmid_forced 를 참조하십시오.
net.ipv4.ip_local_port_range	TCP 및 UDP에서 로컬 포트를 선택하는 데 사용하는 로컬 포트 범위를 정의합니다. 첫 번째 숫자는 첫 번째 포트 번호이고 두 번째 숫자는 마지막 로컬 포트 번호입니다. 가능한 경우 이 숫자에 다른 패리티(하나 및 홀수 값)가 있는 경우 더 좋습니다. ip_unprivileged_port_start 보다 크거나 같아야 합니다. 기본 값은 각각 32768 및 60999 입니다. 자세한 내용은 ip_local_port_range 를 참조하십시오.
net.ipv4.tcp_syncookies	net.ipv4.tcp_syncookies 가 설정된 경우 커널은 일반적으로 half-open 연결 대기열이 가득 찰 때까지 TCP-O2-패킷을 처리하며, 이 시점에는 cryptsetup 쿠키 기능이 시작됩니다. 이 기능을 사용하면 서비스 거부 공격에서도 시스템이 유효한 연결을 계속 허용할 수 있습니다. 자세한 내용은 tcp_syncookies 를 참조하십시오.
net.ipv4.ping_group_range	이렇게 하면 ICMP_PROTO 데이터그램 소켓이 그룹 범위의 사용자로 제한됩니다. 기본 값은 1 0 이며 이는 root가 아닌 nobody가 ping 소켓을 생성할 수 없음을 의미합니다. 자세한 내용은 ping_group_range 를 참조하십시오.
net.ipv4.ip_unprivileged_port_start	이는 네트워크 네임스페이스에서 권한이 없는 첫 번째 포트를 정의합니다. 권한 있는 모든 포트를 비활성화하려면 이를 0 으로 설정합니다. 권한이 있는 포트는 ip_local_port_range 와 겹치지 않아야 합니다. 자세한 내용은 ip_unprivileged_port_start 를 참조하십시오.

표 7.5. 인터페이스별 안전한 **sysctl**

sysctl	설명
net.ipv4.conf.IFNAME.accept_redirects	IPv4 ICMP 리디렉션 메시지를 수락합니다.
net.ipv4.conf.IFNAME.accept_source_route	엄격한 소스 경로(SRR) 옵션을 사용하여 IPv4 패킷을 수락합니다.
net.ipv4.conf.IFNAME.arp_accept	ARP 표에 없는 IPv4 주소를 사용하여 무독한 ARP 프레임에 대한 동작을 정의합니다. <ul style="list-style-type: none"> ● 0 - ARP 테이블에 새 항목을 생성하지 마십시오. ● 1 - ARP 테이블에 새 항목을 만듭니다.
net.ipv4.conf.IFNAME.arp_notify	IPv4 주소 및 장치 변경 사항에 대한 알림 모드를 정의합니다.
net.ipv4.conf.IFNAME.disable_policy	이 IPv4 인터페이스에 대해 IPSEC 정책(SPD)을 비활성화합니다.
net.ipv4.conf.IFNAME.secure_redirects	ICMP는 인터페이스의 현재 게이트웨이 목록에 나열된 게이트웨이로만 메시지를 리디렉션합니다.
net.ipv4.conf.IFNAME.send_redirects	send redirects는 노드가 라우터 역할을 하는 경우에만 활성화됩니다. 즉 호스트는 ICMP 리디렉션 메시지를 보낼 수 없습니다. 라우터에서 호스트에 특정 대상에 사용 가능한 더 나은 라우팅 경로를 알리는 데 사용됩니다.
net.ipv6.conf.IFNAME.accept_ra	IPv6 라우터 알림을 수락하고 이를 사용하여 자동으로 구성합니다. 또한 라우터 요청을 전송할지 여부를 결정합니다. 라우터 요청은 기능 설정이 라우터 알림을 수락하는 경우에만 전송됩니다.
net.ipv6.conf.IFNAME.accept_redirects	IPv6 ICMP 리디렉션 메시지를 수락합니다.
net.ipv6.conf.IFNAME.accept_source_route	SRR 옵션을 사용하여 IPv6 패킷을 수락합니다.
net.ipv6.conf.IFNAME.arp_accept	ARP 표에 없는 IPv6 주소를 사용하여 무독한 ARP 프레임에 대한 동작을 정의합니다. <ul style="list-style-type: none"> ● 0 - ARP 테이블에 새 항목을 생성하지 마십시오. ● 1 - ARP 테이블에 새 항목을 만듭니다.
net.ipv6.conf.IFNAME.arp_notify	IPv6 주소 및 장치 변경 사항에 대한 알림 모드를 정의합니다.

sysctl	설명
net.ipv6.neigh.IFNAME.base_reachable_time_ms	이 매개 변수는 IPv6용 주변 테이블에서 IP 매핑 수명에 대한 하드웨어 주소를 제어합니다.
net.ipv6.neigh.IFNAME.retans_time_ms	주변 검색 메시지의 다시 전송 타이머를 설정합니다.



참고

튜닝 CNI 플러그인을 사용하여 이러한 값을 설정할 때 문자 그대로 **IFNAME** 값을 사용합니다. 인터페이스 이름은 **IFNAME** 토큰으로 표시되고 런타임 시 인터페이스의 실제 이름으로 교체됩니다.

7.9.4. 인터페이스별 안전한 sysctl 목록 업데이트

OpenShift Container Platform에는 사전 정의된 안전한 인터페이스별 **sysctl** 목록이 포함되어 있습니다. **openshift-multus** 네임스페이스에서 **cni-sysctl-allowlist** 를 업데이트하여 이 목록을 수정할 수 있습니다.



중요

인터페이스별 안전한 **sysctl** 목록 업데이트 지원은 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.

사전 정의된 안전한 **sysctl** 목록을 수정하려면 다음 절차를 따르십시오. 다음 절차에서는 기본 허용 목록을 확장하는 방법을 설명합니다.

절차

1. 다음 명령을 실행하여 사전 정의된 기존 목록을 확인합니다.

```
$ oc get cm -n openshift-multus cni-sysctl-allowlist -oyaml
```

예상 출력

```

apiVersion: v1
data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
kind: ConfigMap
metadata:
  annotations:
    kubernetes.io/description: |
      Sysctl allowlist for nodes.
    release.openshift.io/version: 4.12.0-0.nightly-2022-11-16-003434
    creationTimestamp: "2022-11-17T14:09:27Z"
    name: cni-sysctl-allowlist
    namespace: openshift-multus
    resourceVersion: "2422"
    uid: 96d138a3-160e-4943-90ff-6108fa7c50c3

```

2.

다음 명령을 사용하여 목록을 편집합니다.

```
$ oc edit cm -n openshift-multus cni-sysctl-allowlist -oyaml
```

예를 들어, 보다 엄격한 역방향 경로를 구현할 수 있도록 하려면 다음과 같이 `^net.ipv4.conf.IFNAME.rp_filter$` 및 `^net.ipv6.conf.conf.IFNAME.rp_filter$` 를 추가해야 합니다.

```

# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1

```

```

data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv4.conf.IFNAME.rp_filter$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
    ^net.ipv6.conf.IFNAME.rp_filter$
    
```

3. 파일의 변경 사항을 저장하고 종료합니다.



참고

sysctl 제거도 지원됩니다. 파일을 편집하고 **sysctl** 또는 **sysctl** 을 제거한 다음 변경 사항을 저장하고 종료합니다.

검증

IPv4에 대해 더 엄격한 역방향 경로 전달을 적용하려면 다음 절차를 따르십시오. 역방향 경로 전달에 대한 자세한 내용은 [역방향 경로 전달](#) 을 참조하십시오.

1. 다음 콘텐츠를 사용하여 **reverse-path-fwd-example.yaml** 과 같은 네트워크 연결 정의를 생성합니다.

```

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
    }],
    {
    
```

```

    "type": "tuning",
    "sysctl": {
      "net.ipv4.conf.IFNAME.rp_filter": "1"
    }
  }
]
}'

```

2.

다음 명령을 실행하여 **yaml**을 적용합니다.

```
$ oc apply -f reverse-path-fwd-example.yaml
```

출력 예

```
networkattachmentdefinition.k8s.cni.cncf.io/tuningnad created
```

3.

다음 **YAML**을 사용하여 **examplepod.yaml** 과 같은 **Pod**를 생성합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: example
  labels:
    app: httpd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad ❶
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: httpd
      image: 'image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest'
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL

```

❶

4. 다음 명령을 실행하여 **yaml**을 적용합니다.

```
$ oc apply -f examplepod.yaml
```

5. 다음 명령을 실행하여 **Pod**가 생성되었는지 확인합니다.

```
$ oc get pod
```

출력 예

```
NAME      READY STATUS  RESTARTS  AGE
example  1/1   Running  0         47s
```

6. 다음 명령을 실행하여 **Pod**에 로그인합니다.

```
$ oc rsh example
```

7. 구성된 **sysctl** 플래그의 값을 확인합니다. 예를 들어 다음 명령을 실행하여 **net.ipv4.conf.net1.rp_filter** 값을 찾습니다.

```
sh-4.4# sysctl net.ipv4.conf.net1.rp_filter
```

예상 출력

```
net.ipv4.conf.net1.rp_filter = 1
```

추가 리소스

- 튜닝 CNI 구성
- Linux 네트워킹 설명서

7.9.5. 안전한 `sysctl`을 사용하여 Pod 시작

Pod의 `securityContext`를 사용하여 Pod에 `sysctl`을 설정할 수 있습니다. `securityContext`는 동일한 Pod의 모든 컨테이너에 적용됩니다.

안전한 `sysctl`은 기본적으로 허용됩니다.

이 예제에서는 Pod `securityContext` 를 사용하여 다음과 같은 안전한 `sysctl`을 설정합니다.

- `kernel.shm_rmid_forced`
- `net.ipv4.ip_local_port_range`
- `net.ipv4.tcp_syncookies`
- `net.ipv4.ping_group_range`



주의

운영 체제가 불안정해지는 것을 방지하기 위해 `sysctl` 매개변수 수정이 미치는 영향을 파악한 후에만 수정하십시오.

다음 절차에 따라 설정된 `sysctl` 설정으로 Pod를 시작합니다.



참고

대부분의 경우 기존 **Pod** 정의를 수정하고 **securityContext** 사양을 추가합니다.

절차

1.

다음 예와 같이 예제 **Pod**를 정의하고 **securityContext** 사양을 추가하는 **YAML** 파일 **sysctl_pod.yaml**을 생성합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  namespace: default
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 ①
      runAsGroup: 3000 ②
      allowPrivilegeEscalation: false ③
      capabilities: ④
        drop: ["ALL"]
    securityContext:
      runAsNonRoot: true ⑤
      seccompProfile: ⑥
        type: RuntimeDefault
    sysctls:
      - name: kernel.shm_rmid_forced
        value: "1"
      - name: net.ipv4.ip_local_port_range
        value: "32770 60666"
      - name: net.ipv4.tcp_syncookies
        value: "0"
      - name: net.ipv4.ping_group_range
        value: "0 200000000"

```

①

Run AsUser는 컨테이너가 실행되는 사용자 **ID**를 제어합니다.

②

runAsGroup은 컨테이너가 실행되는 기본 그룹 **ID**를 제어합니다.

③

4

기능을 사용하면 전체 루트 액세스 권한을 부여하지 않고 권한 있는 작업을 수행할 수 있습니다. 이 정책은 모든 기능이 Pod에서 삭제되도록 합니다.

5

`runAsNonRoot: true` 를 사용하려면 컨테이너가 0이 아닌 다른 UID와 함께 컨테이너를 실행해야 합니다.

6

`RuntimeDefault` 는 Pod 또는 컨테이너 워크로드에 대한 기본 `seccomp` 프로필을 활성화합니다.

2.

다음 명령을 실행하여 Pod를 생성합니다.

```
$ oc apply -f sysctl_pod.yaml
```

3.

다음 명령을 실행하여 Pod가 생성되었는지 확인합니다.

```
$ oc get pod
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
sysctl-example	1/1	Running	0	14s

4.

다음 명령을 실행하여 Pod에 로그인합니다.

```
$ oc rsh sysctl-example
```

5.

구성된 `sysctl` 플래그 값을 확인합니다. 예를 들어 다음 명령을 실행하여 `kernel.shm_rmid_forced` 값을 찾습니다.

```
sh-4.4# sysctl kernel.shm_rmid_forced
```

예상 출력

```
kernel.shm_rmid_forced = 1
```

7.9.6. 안전하지 않은 `sysctl`을 사용하여 Pod 시작

안전하지 않은 `sysctl`이 있는 Pod는 클러스터 관리자가 해당 노드에 대해 안전하지 않은 `sysctl`을 명시적으로 활성화하지 않는 한 어떠한 노드에서도 시작되지 않습니다. 노드 수준 `sysctl`과 마찬가지로 노드에 테인트 및 허용 오차 기능을 사용하여 해당 Pod를 올바른 노드에 예약합니다.

다음 예제에서는 Pod `securityContext` 를 사용하여 안전한 `sysctl`인 `kernel.shm_rmid_forced` 와 안전하지 않은 `sysctl`인 `net.core.somaxconn` 및 `kernel.msgmaxconn`을 설정합니다. 사양에서는 안전 및 안전하지 않은 `sysctl`이 구분되지 않습니다.



주의

운영 체제가 불안정해지는 것을 방지하기 위해 `sysctl` 매개변수 수정이 미치는 영향을 파악한 후에만 수정하십시오.

다음 예제에서는 Pod 사양에 안전 및 안전하지 않은 `sysctl`을 추가할 때 발생하는 상황을 보여줍니다.

절차

1. 다음 예와 같이 예제 Pod를 정의하는 YAML 파일 `sysctl-example-unsafe.yaml` 을 생성하고 `securityContext` 사양을 추가합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-unsafe
spec:
  containers:
  - name: podexample
```

```

image: centos
command: ["bin/bash", "-c", "sleep INF"]
securityContext:
  runAsUser: 2000
  runAsGroup: 3000
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["ALL"]
securityContext:
  runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
sysctls:
- name: kernel.shm_rmid_forced
  value: "0"
- name: net.core.somaxconn
  value: "1024"
- name: kernel.msgmax
  value: "65536"

```

2.

다음 명령을 사용하여 Pod를 생성합니다.

```
$ oc apply -f sysctl-example-unsafe.yaml
```

3.

안전하지 않은 **sysctl**이 다음 명령을 사용하여 노드에 허용되지 않으므로 Pod가 예약되었지만 배포되지 않는지 확인합니다.

```
$ oc get pod
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
sysctl-example-unsafe	0/1	SysctlForbidden	0	14s

7.9.7. 안전하지 않은 sysctl 활성화

클러스터 관리자는 고성능 또는 실시간 애플리케이션 튜닝과 같이 매우 특별한 상황에 대해 안전하지 않은 특정 **sysctl**을 허용할 수 있습니다.

안전하지 않은 **sysctl**을 사용하려면 클러스터 관리자가 특정 유형의 노드에 대해 개별적으로 활성화해

야 합니다. **sysctl**에 네임스페이스가 지정되어 있어야 합니다.

보안 컨텍스트 제약 조건의 **allowedUnsafeSysctls** 필드에 **sysctls** 목록 또는 **sysctl** 패턴 목록을 지정하여 **Pod**에 설정된 **sysctl**을 추가로 제어할 수 있습니다.

- **allowedUnsafeSysctls** 옵션은 고성능 또는 실시간 애플리케이션 튜닝과 같은 특정 요구 사항을 제어합니다.



주의

안전하지 않은 상태의 특성으로 인해 안전하지 않은 **sysctl**을 사용하는 경우 사용자가 위험을 감수해야 하고 부적절한 컨테이너 동작, 리소스 부족 또는 노드 중단과 같은 심각한 문제가 발생할 수 있습니다.

절차

1. **OpenShift Container Platform** 클러스터의 기존 **MachineConfig** 오브젝트를 나열하여 다음 명령을 실행하여 머신 구성에 레이블을 지정하는 방법을 결정합니다.

```
$ oc get machineconfigpool
```

출력 예

NAME	CONFIG	UPDATED	UPDATING	DEGRADED	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT	DEGRADEDMACHINECOUNT	AGE
master	rendered-master-bfb92f0cd1684e54d8e234ab7423cc96	True	False	False	3	3	3	0	42m
worker	rendered-worker-21b6cb9a0f8919c88caf39db80ac1fce	True	False	False	3	3	3	0	42m

2. 다음 명령을 실행하여 안전하지 않은 **sysctl**이 있는 컨테이너가 실행될 머신 구성 풀에 레이블을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=sysctl
```

3.

KubeletConfig CR(사용자 정의 리소스)을 정의하는 **YAML 파일 set-sysctl-worker.yaml**을 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl 1
  kubeletConfig:
    allowedUnsafeSysctls: 2
    - "kernel.msg*"
    - "net.core.somaxconn"
```

1

머신 구성 풀에서 라벨을 지정합니다.

2

허용할 안전하지 않은 **sysctl**을 나열합니다.

4.

다음 명령을 실행하여 오브젝트를 생성합니다.

```
$ oc apply -f set-sysctl-worker.yaml
```

5.

Machine Config Operator가 렌더링된 새 구성을 생성하고 다음 명령을 실행하여 머신에 적용할 때까지 기다립니다.

```
$ oc get machineconfigpool worker -w
```

잠시 후 **UPDATING** 상태가 **True**에서 **False**로 변경됩니다.

	NAME	CONFIG	UPDATED	UPDATING	DEGRADED					
	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT							
	DEGRADEDMACHINECOUNT	AGE								
	worker	rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800	3	2	2	0	71m	False	True	False
	worker	rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800						False	True	False

3	2	3	0	72m			
worker	rendered-worker-0188658afe1f3a183ec8c4f14186f4d5	True	False	False			
3	3	3	0	72m			

6.

다음 예와 같이 예제 Pod를 정의하는 YAML 파일 `sysctl-example-safe-unsafe.yaml` 을 생성하고 `securityContext` 사양을 추가합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-safe-unsafe
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.core.somaxconn
      value: "1024"
    - name: kernel.msgmax
      value: "65536"

```

7.

다음 명령을 실행하여 Pod를 생성합니다.

```

$ oc apply -f sysctl-example-safe-unsafe.yaml

```

예상 출력

```

Warning: would violate PodSecurity "restricted:latest": forbidden sysctls
(net.core.somaxconn, kernel.msgmax)
pod/sysctl-example-safe-unsafe created

```


8. 다음 명령을 실행하여 Pod가 생성되었는지 확인합니다.

```
$ oc get pod
```

출력 예

```
NAME                READY STATUS RESTARTS AGE
sysctl-example-safe-unsafe 1/1 Running 0      19s
```

9. 다음 명령을 실행하여 Pod에 로그인합니다.

```
$ oc rsh sysctl-example-safe-unsafe
```

10. 구성된 `sysctl` 플래그 값을 확인합니다. 예를 들어 다음 명령을 실행하여 `net.core.somaxconn` 값을 찾습니다.

```
sh-4.4# sysctl net.core.somaxconn
```

예상 출력

```
net.core.somaxconn = 1024
```

안전하지 않은 `sysctl`이 허용되고 업데이트된 Pod 사양의 `securityContext` 사양에 값이 정의된 대로 설정됩니다.

7.9.8. 추가 리소스

- [인터페이스 수준 네트워크 sysctl 설정](#)

- [Node Tuning Operator 사용](#)

8장. 클러스터 작업

8.1. OPENSIFT CONTAINER PLATFORM 클러스터에서 시스템 이벤트 정보 보기

OpenShift Container Platform의 이벤트는 OpenShift Container Platform 클러스터의 API 오브젝트에 발생하는 이벤트를 기반으로 모델링됩니다.

8.1.1. 이벤트 이해

OpenShift Container Platform은 이벤트를 통해 리소스와 관계없이 실제 이벤트에 대한 정보를 기록할 수 있습니다. 또한 개발자와 관리자는 통합된 방식으로 시스템 구성 요소에 대한 정보를 사용할 수 있습니다.

8.1.2. CLI를 사용하여 이벤트 보기

CLI를 사용하여 지정된 프로젝트의 이벤트 목록을 가져올 수 있습니다.

프로세스

- 프로젝트의 이벤트를 보려면 다음 명령을 사용합니다.

```
$ oc get events [-n <project>] 1
```

1

프로젝트 이름입니다.

예를 들면 다음과 같습니다.

```
$ oc get events -n openshift-config
```

출력 예

```
LAST SEEN TYPE REASON OBJECT MESSAGE
97m Normal Scheduled pod/dapi-env-test-pod Successfully
assigned openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
97m Normal Pulling pod/dapi-env-test-pod pulling image
"gcr.io/google_containers/busybox"
```

```

97m      Normal  Pulled           pod/dapi-env-test-pod  Successfully pulled
image "gcr.io/google_containers/busybox"
97m      Normal  Created          pod/dapi-env-test-pod  Created container
9m5s     Warning  FailedCreatePodSandBox pod/dapi-volume-test-pod Failed
create pod sandbox: rpc error: code = Unknown desc = failed to create pod network
sandbox k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9
e22): Multus: Err adding pod to network "openshift-sdn": cannot set "openshift-sdn"
ifname to "eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or
directory
8m31s   Normal  Scheduled        pod/dapi-volume-test-pod Successfully
assigned openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal
    
```

-

OpenShift Container Platform 콘솔에서 프로젝트의 이벤트를 보려면 다음을 수행합니다.

1. **OpenShift Container Platform** 콘솔을 시작합니다.
2. 홈 → 이벤트를 클릭하고 프로젝트를 선택합니다.
3. 이벤트를 표시할 리소스로 이동합니다. 예를 들면 홈 → 프로젝트 → <프로젝트 이름> → <리소스 이름>과 같습니다.

Pod 및 배포와 같이 많은 오브젝트에는 자체 이벤트 탭도 있으며 해당 오브젝트와 관련된 이벤트가 표시됩니다.

8.1.3. 이벤트 목록

이 섹션에서는 OpenShift Container Platform의 이벤트에 대해 설명합니다.

표 8.1. 구성 이벤트

이름	설명
FailedValidation	Pod 구성 검증에 실패했습니다.

표 8.2. 컨테이너 이벤트

이름	설명
BackOff	백오프로 컨테이너를 재시작하지 못했습니다.
Created	컨테이너가 생성되었습니다.
Failed	가져오기/생성/시작이 실패했습니다.
Killing	컨테이너를 종료합니다.
Started	컨테이너가 시작되었습니다.
Preempting	다른 Pod를 선점합니다.
ExceededGrace Period	컨테이너 런타임이 지정된 유예 기간 내에 Pod를 중지하지 않았습니다.

표 8.3. 상태 이벤트

이름	설명
Unhealthy	컨테이너 상태가 비정상입니다.

표 8.4. 이미지 이벤트

이름	설명
BackOff	Ctrl Start를 백오프하고 이미지를 가져옵니다.
ErrImageNeverPull	이미지의 NeverPull Policy 를 위반했습니다.
Failed	이미지를 가져오지 못했습니다.
InspectFailed	이미지를 검사하지 못했습니다.
Pulled	이미지를 가져왔거나 컨테이너 이미지가 머신에 이미 있습니다.
Pulling	이미지를 가져오는 중입니다.

표 8.5. 이미지 관리자 이벤트

이름	설명
FreeDiskSpaceFailed	디스크 공간을 비우지 못했습니다.
InvalidDiskCapacity	디스크 용량이 유효하지 않습니다.

표 8.6. 노드 이벤트

이름	설명
FailedMount	볼륨을 마운트하지 못했습니다.
HostNetworkNotSupported	호스트 네트워크가 지원되지 않습니다.
HostPortConflict	호스트/포트가 충돌합니다.
KubeletSetupFailed	kubelet 설정에 실패했습니다.
NilShaper	정의되지 않은 셰이퍼입니다.
NodeNotReady	노드가 준비되지 않았습니다.
NodeNotSchedulable	노드를 예약할 수 없습니다.
NodeReady	노드가 준비되었습니다.
NodeSchedulable	노드를 예약할 수 있습니다.
NodeSelectorMismatching	노드 선택기가 일치하지 않습니다.
OutOfDisk	디스크가 없습니다.
Rebooted	노드가 재부팅되었습니다.
Starting	kubelet을 시작합니다.
FailedAttachVolume	볼륨을 연결할 수 없습니다.

이름	설명
FailedDetachVolume	볼륨을 분리할 수 없습니다.
VolumeResizeFailed	볼륨을 확장/축소할 수 없습니다.
VolumeResizeSuccessful	볼륨을 확장/축소했습니다.
FileSystemResizeFailed	파일 시스템을 확장/축소하지 못했습니다.
FileSystemResizeSuccessful	파일 시스템을 확장/축소했습니다.
FailedUnmount	볼륨을 마운트 해제하지 못했습니다.
FailedMapVolume	볼륨을 매핑하지 못했습니다.
FailedUnmapDevice	장치를 매핑 해제하지 못했습니다.
AlreadyMountedVolume	볼륨이 이미 마운트되어 있습니다.
SuccessfulDetachVolume	볼륨이 분리되었습니다.
SuccessfulMountVolume	볼륨을 마운트했습니다.
SuccessfulUnmountVolume	볼륨을 마운트 해제했습니다.
ContainerGCFailed	컨테이너 가비지 컬렉션에 실패했습니다.
ImageGCFailed	이미지 가비지 컬렉션에 실패했습니다.
FailedNodeAllocatableEnforcement	시스템 예약 Cgroup 제한을 적용하지 못했습니다.

이름	설명
NodeAllocatableEnforced	시스템 예약 Cgroup 제한을 적용했습니다.
UnsupportedMountOption	지원되지 않는 마운트 옵션입니다.
SandboxChanged	Pod 샌드박스가 변경되었습니다.
FailedCreatePodSandbox	Pod 샌드박스를 생성하지 못했습니다.
FailedPodSandboxStatus	실패한 Pod 샌드박스 상태입니다.

표 8.7. Pod 작업자 이벤트

이름	설명
FailedSync	Pod 동기화에 실패했습니다.

표 8.8. 시스템 이벤트

이름	설명
SystemOOM	클러스터에 OOM(메모리 부족) 상황이 있습니다.

표 8.9. Pod 이벤트

이름	설명
FailedKillPod	Pod를 중지하지 못했습니다.
FailedCreatePodContainer	Pod 컨테이너를 생성하지 못했습니다.
Failed	Pod 데이터 디렉토리를 생성하지 못했습니다.
NetworkNotReady	네트워크가 준비되지 않았습니다.
FailedCreate	생성하는 동안 오류가 발생했습니다(<error-msg>).
SuccessfulCreate	Pod가 생성되었습니다(<pod-name>).

이름	설명
FailedDelete	삭제하는 동안 오류가 발생했습니다(<error-msg>).
SuccessfulDelete	Pod가 삭제되었습니다(<pod-id>).

표 8.10. 수평 Pod 자동 스케일러 이벤트

이름	설명
SelectorRequired	선택기가 필요합니다.
InvalidSelector	선택기를 해당 내부 선택기 오브젝트로 변환할 수 없습니다.
FailedGetObjectMetric	HPA에서 복제본 수를 계산할 수 없었습니다.
InvalidMetricSourceType	알 수 없는 메트릭 소스 유형입니다.
ValidMetricFound	HPA에서 복제본 수를 성공적으로 계산할 수 있었습니다.
FailedConvertHPA	지정된 HPA를 변환하지 못했습니다.
FailedGetScale	HPA 컨트롤러에서 대상의 현재 규모를 가져올 수 없었습니다.
SucceededGetScale	HPA 컨트롤러에서 대상의 현재 규모를 가져올 수 있었습니다.
FailedComputeMetricsReplicas	나열된 메트릭을 기반으로 원하는 복제본 수를 계산하지 못했습니다.
FailedRescale	새 크기: <size>, 이유: <msg>, 오류: <error-msg>
SuccessfulRescale	새 크기: <size>, 이유: <msg>
FailedUpdateStatus	상태를 업데이트하지 못했습니다.

표 8.11. 네트워크 이벤트(openshift-sdn)

이름	설명
Starting	OpenShift SDN 시작.
NetworkFailed	Pod의 네트워크 인터페이스가 손실되어 Pod가 중지됩니다.

표 8.12. 네트워크 이벤트(*kube-proxy*)

이름	설명
NeedPods	서비스 포트 <code><serviceName>:<port></code> 에 Pod가 필요합니다.

표 8.13. 볼륨 이벤트

이름	설명
FailedBinding	사용 가능한 영구 볼륨이 없으며 스토리지 클래스가 설정되지 않았습니다.
VolumeMismatch	볼륨 크기 또는 클래스가 클레임에서 요청한 것과 다릅니다.
VolumeFailedRecycle	재생기 Pod를 생성하는 동안 오류가 발생했습니다.
VolumeRecycled	볼륨이 재생될 때 발생합니다.
RecyclerPod	Pod가 재생될 때 발생합니다.
VolumeDelete	볼륨이 삭제될 때 발생합니다.
VolumeFailedDelete	볼륨을 삭제할 때 오류가 발생했습니다.
ExternalProvisioning	클레임에 대한 볼륨이 수동으로 또는 외부 소프트웨어를 통해 프로비저닝되는 경우 발생합니다.
ProvisioningFailed	볼륨을 프로비저닝하지 못했습니다.
ProvisioningCleanupFailed	프로비저닝된 볼륨을 정리하는 동안 오류가 발생했습니다.
ProvisioningSucceeded	볼륨이 성공적으로 프로비저닝될 때 발생합니다.

이름	설명
WaitForFirstConsumer	Pod가 예약될 때까지 바인딩이 지연됩니다.

표 8.14. 라이프사이클 후크

이름	설명
FailedPostStartHook	핸들러에서 Pod를 시작하지 못했습니다.
FailedPreStopHook	핸들러에서 사전 정지하지 못했습니다.
UnfinishedPreStopHook	사전 정지 후크가 완료되지 않았습니다.

표 8.15. 배포

이름	설명
DeploymentCancellationFailed	배포를 취소하지 못했습니다.
DeploymentCancelled	배포가 취소되었습니다.
DeploymentCreated	새 복제 컨트롤러가 생성되었습니다.
IngressIPRangeFull	서비스에 할당할 수 있는 Ingress IP가 없습니다.

표 8.16. 스케줄러 이벤트

이름	설명
FailedScheduling	Pod(<pod-namespace>/<pod-name>)를 예약하지 못했습니다. 이 이벤트는 AssumePodVolumes 실패, 바인딩 거부 등과 같은 다양한 이유로 발생합니다.
Preempted	<node-name> 노드의 <preemptor-namespace>/<preemptor-name>에 의해 발생합니다.
Scheduled	<pod-name>을(를) <node-name>에 할당했습니다.

표 8.17. 데몬 세트 이벤트

이름	설명
SelectingAll	이 데몬 세트는 모든 Pod를 선택합니다. 비어 있지 않은 선택기가 필요합니다.
FailedPlacement	<node-name>에 Pod를 배치하지 못했습니다.
FailedDaemonPod	<node-name> 노드에 실패한 데몬 Pod<pod-name>이(가) 있어 종료하려고 합니다.

표 8.18. LoadBalancer 서비스 이벤트

이름	설명
CreatingLoadBalancerFailed	로드 밸런서 생성 중 오류가 발생했습니다.
DeletingLoadBalancer	로드 밸런서를 삭제하는 중입니다.
EnsuringLoadBalancer	로드 밸런서를 확인하는 중입니다.
EnsuredLoadBalancer	로드 밸런서를 확인했습니다.
UnavailableLoadBalancer	LoadBalancer 서비스에 사용 가능한 노드가 없습니다.
LoadBalancerSourceRanges	새 LoadBalancerSourceRanges 를 나열합니다. 예를 들면 <old-source-range> → <new-source-range>입니다.
LoadbalancerIP	새 IP 주소를 나열합니다. 예를 들면 <old-ip> → <new-ip>입니다.
ExternalIP	외부 IP 주소를 나열합니다. 예를 들면 Added: <external-ip> 입니다.
UID	새 UID를 나열합니다. 예를 들면 <old-service-uid> → <new-service-uid>입니다.
ExternalTrafficPolicy	새 ExternalTrafficPolicy 를 나열합니다. 예를 들면 <old-policy> → <new-policy>입니다.
HealthCheckNodePort	새 HealthCheckNodePort 를 나열합니다. 예를 들면 <old-node-port> → <new-node-port>입니다.
UpdatedLoadBalancer	새 호스트로 로드 밸런서를 업데이트했습니다.

이름	설명
LoadBalancerUpdateFailed	새 호스트로 로드 밸런서를 업데이트하는 동안 오류가 발생했습니다.
DeletingLoadBalancer	로드 밸런서를 삭제하는 중입니다.
DeletingLoadBalancerFailed	로드 밸런서를 삭제하는 동안 오류가 발생했습니다.
DeletedLoadBalancer	로드 밸런서를 삭제했습니다.

8.2. OPENSIFT CONTAINER PLATFORM 노드에서 보유할 수 있는 POD 수 추정

클러스터 관리자는 **OpenShift Cluster Capacity Tool**을 사용하여 현재 리소스가 소진되기 전에 현재 리소스를 늘리기 전에 예약할 수 있는 Pod 수를 확인하고 향후 포드를 예약할 수 있습니다. 이러한 용량은 클러스터의 개별 노드 호스트에서 제공하며 CPU, 메모리, 디스크 공간 등을 포함합니다.

8.2.1. OpenShift Cluster Capacity 툴 이해

OpenShift Cluster Capacity Tool은 보다 정확한 추정을 제공하기 위해 리소스가 소진되기 전에 클러스터에서 예약할 수 있는 입력 포드의 인스턴스 수를 결정하기 위해 일련의 스케줄링 결정을 시뮬레이션합니다.

참고

나머지 할당 가능 용량은 여러 노드에 배포되는 모든 리소스를 계산하지 않기 때문에 대략적인 추정치입니다. 남은 리소스만 분석하고 클러스터에서 예약할 수 있는 지정된 요구 사항이 포함된 Pod의 여러 인스턴스 측면에서 여전히 사용할 수 있는 가용 용량을 추정합니다.

또한 Pod는 선택 및 유사성 기준에 따라 특정 노드 집합에서만 예약 기능이 지원될 수 있습니다. 이로 인해 클러스터에서 예약할 수 있는 나머지 Pod를 추정하기 어려울 수 있습니다.

OpenShift Cluster Capacity Tool을 명령줄에서 독립형 유틸리티로 실행하거나 **OpenShift Container Platform** 클러스터 내부의 Pod에서 작업으로 실행할 수 있습니다. Pod 내에서 툴을 작업으로 실행하면 개입 없이 여러 번 실행할 수 있습니다.

8.2.2. 명령줄에서 OpenShift Cluster Capacity Tool 실행

명령줄에서 **OpenShift Cluster Capacity Tool**을 실행하여 클러스터에 예약할 수 있는 포드 수를 추정할 수 있습니다.

틀에서 리소스 사용량을 추정하는 데 사용하는 샘플 **Pod** 사양 파일을 생성합니다. **Pod** 사양은 리소스 요구 사항을 제한 또는 요청으로 지정합니다. 클러스터 용량 틀에서는 추정 분석에 **Pod**의 리소스 요구 사항을 고려합니다.

사전 요구 사항

1. **Red Hat Ecosystem Catalog**의 컨테이너 이미지로 사용할 수 있는 **OpenShift Cluster Capacity Tool** 을 실행합니다.
2. 샘플 **Pod** 사양 파일을 생성합니다.
 - a. 다음과 유사한 **YAML** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
  resources:
    limits:
      cpu: 150m
      memory: 100Mi
    requests:
      cpu: 150m
      memory: 100Mi
```

- b. 클러스터 역할을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f pod-spec.yaml
```

절차

명령줄에서 클러스터 용량 툴을 사용하려면 다음을 수행합니다.

1. 터미널에서 **Red Hat Registry**에 로그인합니다.

```
$ podman login registry.redhat.io
```

2. 클러스터 용량 툴 이미지를 가져옵니다.

```
$ podman pull registry.redhat.io/openshift4/ose-cluster-capacity
```

3. 클러스터 용량 툴을 실행합니다.

```
$ podman run -v $HOME/.kube:/kube:Z -v $(pwd):/cc:Z ose-cluster-capacity \
/bin/cluster-capacity --kubeconfig /kube/config --<pod_spec>.yaml \
/cc/<pod_spec>.yaml \
--verbose
```

다음과 같습니다.

```
<pod_spec>.yaml
```

사용할 **Pod** 사양을 지정합니다.

상세 정보

클러스터의 각 노드에서 예약할 수 있는 **Pod** 수에 대한 자세한 설명을 출력합니다.

출력 예

```
small-pod pod requirements:
- CPU: 150m
- Memory: 100Mi
```

The cluster can schedule 88 instance(s) of the pod small-pod.

Termination reason: Unschedulable: 0/5 nodes are available: 2 Insufficient cpu, 3 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't tolerate.

Pod distribution among nodes:

small-pod

- 192.168.124.214: 45 instance(s)*
- 192.168.124.120: 43 instance(s)*

위 예에서 클러스터에 예약할 수 있는 예상 Pod 수는 88입니다.

8.2.3. Pod 내에서 OpenShift Cluster Capacity Tool을 작업으로 실행

포드 내에서 OpenShift Cluster Capacity Tool을 작업으로 실행하면 사용자 개입 없이도 툴을 여러 번 실행할 수 있습니다. ConfigMap 오브젝트를 사용하여 OpenShift Cluster Capacity Tool을 작업으로 실행합니다.

사전 요구 사항

[OpenShift Cluster Capacity 툴](#) 을 다운로드하여 설치합니다.

프로세스

클러스터 용량 툴을 실행하려면 다음을 수행합니다.

1. 클러스터 역할을 생성합니다.
 - a. 다음과 유사한 YAML 파일을 생성합니다.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: ["" ]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services", "replicationcontrollers"]
```



```

verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["replicasets", "statefulsets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["policy"]
  resources: ["poddisruptionbudgets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "watch", "list"]

```

b.

다음 명령을 실행하여 클러스터 역할을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create sa cluster-capacity-sa
```

2.

서비스 계정을 생성합니다.

```
$ oc create sa cluster-capacity-sa -n default
```

3.

서비스 계정에 역할을 추가합니다.

```
$ oc adm policy add-cluster-role-to-user cluster-capacity-role \
system:serviceaccount:<namespace>:cluster-capacity-sa
```

다음과 같습니다.

<namespace>

Pod가 있는 네임스페이스를 지정합니다.

4.

Pod 사양을 정의하고 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```
apiVersion: v1
```

```

kind: Pod
metadata:
  name: small-pod
labels:
  app: guestbook
  tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
  resources:
    limits:
    cpu: 150m
    memory: 100Mi
    requests:
    cpu: 150m
    memory: 100Mi

```

b.

다음 명령을 실행하여 **Pod**를 생성합니다.

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

예를 들면 다음과 같습니다.

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

5.

다음 명령을 실행하여 구성 맵 오브젝트를 생성했습니다.

```
$ oc create configmap cluster-capacity-configmap \
--from-file=pod.yaml=pod.yaml
```

클러스터 용량 분석은 입력 **Pod** 사양 파일 **pod.yaml** 을 경로 **/test-pod** 의 볼륨 **test-volume** 에 마운트하기 위해 **cluster-capacity-configmap** 이라는 구성 맵 오브젝트를 사용하여 볼륨에 마운트됩니다.

6.

아래의 작업 사양 파일 예제를 사용하여 작업을 생성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```

apiVersion: batch/v1
kind: Job
metadata:

```

```

name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
      containers:
      - name: cluster-capacity
        image: openshift/origin-cluster-capacity
        imagePullPolicy: "Always"
        volumeMounts:
        - mountPath: /test-pod
          name: test-volume
      env:
      - name: CC_INCLUSTER 1
        value: "true"
      command:
      - "/bin/sh"
      - "-ec"
      - |
        /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
      restartPolicy: "Never"
      serviceAccountName: cluster-capacity-sa
      volumes:
      - name: test-volume
        configMap:
          name: cluster-capacity-configmap

```

1

클러스터 용량 틀에 클러스터 내에서 Pod로 실행되고 있음을 알리는 필수 환경 변수입니다.

ConfigMap 오브젝트의 `pod.yaml` 키는 필수는 아니지만 Pod 사양 파일의 이름과 동일합니다. 이렇게 하면 Pod 내부에서 `/test-pod/pod.yaml`로 입력 Pod 사양 파일에 액세스할 수 있습니다.

b.

다음 명령을 실행하여 Pod에서 클러스터 용량 이미지를 작업으로 실행합니다.

```
$ oc create -f cluster-capacity-job.yaml
```

검증

1.

작업 로그를 확인하여 클러스터에서 예약할 수 있는 Pod 수를 찾습니다.

```
$ oc logs jobs/cluster-capacity-job
```

출력 예

small-pod pod requirements:

- CPU: 150m
- Memory: 100Mi

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: **Unschedulable: No nodes are available that match all of the following predicates:: Insufficient cpu (2).**

Pod distribution among nodes:**small-pod**

- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)

8.3. 제한 범위를 사용하여 리소스 사용 제한

기본적으로 컨테이너는 **OpenShift Container Platform** 클러스터에서 바인딩되지 않은 컴퓨팅 리소스와 함께 실행됩니다. 제한 범위를 사용하면 프로젝트에서 특정 오브젝트에 대한 리소스 사용을 제한할 수 있습니다.

- **Pod 및 컨테이너:** Pod 및 해당 컨테이너의 CPU 및 메모리에 대한 최소 및 최대 요구사항을 설정할 수 있습니다.
- **이미지 스트림:** ImageStream 오브젝트에서 이미지 및 태그 수에 대한 제한을 설정할 수 있습니다.
- **이미지:** 내부 레지스트리로 내보낼 수 있는 이미지 크기를 제한할 수 있습니다.
- **PVC(영구 볼륨 클레임):** 요청할 수 있는 PVC 크기를 제한할 수 있습니다.

Pod가 제한 범위에 따라 적용된 제약 조건을 충족하지 않는 경우에는 네임스페이스에 Pod를 생성할 수 없습니다.

8.3.1. 제한 범위 정보

LimitRange 오브젝트에서 정의하는 제한 범위는 프로젝트의 리소스 사용을 제한합니다. 프로젝트에서는 **Pod**, 컨테이너, 이미지 스트림 또는 **PVC**(영구 볼륨 클레임)에 대한 특정 리소스 제한을 설정할 수 있습니다.

리소스 생성 및 수정을 위한 모든 요청은 프로젝트의 각 **LimitRange** 오브젝트에 대해 평가됩니다. 리소스가 열거된 제약 조건을 위반하는 경우 해당 리소스는 거부됩니다.

다음은 모든 구성 요소의 제한 범위 오브젝트(**Pod**, 컨테이너, 이미지, 이미지 스트림 또는 **PVC**)를 보여줍니다. 동일한 오브젝트에서 이러한 구성 요소의 일부 또는 모두에 대한 제한을 구성할 수 있습니다. 리소스를 제어하려는 각 프로젝트에 대해 서로 다른 제한 범위 오브젝트를 생성합니다.

컨테이너의 제한 범위 오브젝트 샘플

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

8.3.1.1. 구성 요소 제한 정보

다음 예제에서는 각 구성 요소에 대한 제한 범위 매개변수를 보여줍니다. 해당 예제는 명확성을 위해 분류되어 있습니다. 필요에 따라 일부 또는 모든 구성 요소에 대해 단일 **LimitRange** 오브젝트를 생성할 수 있습니다.

8.3.1.1.1. 컨테이너 제한

제한 범위를 사용하면 **Pod**의 각 컨테이너에서 특정 프로젝트에 대해 요청할 수 있는 최소 및 최대 **CPU** 및 메모리를 지정할 수 있습니다. 프로젝트에서 컨테이너가 생성되면 **Pod** 사양의 컨테이너 **CPU** 및 메모리 요청이 **LimitRange** 오브젝트에 설정된 값을 준수해야 합니다. 그러지 않으면 **Pod**가 생성되지 않습니다.

- 컨테이너 **CPU** 또는 메모리에 대한 요청 및 제한이 **LimitRange** 오브젝트에 지정된 컨테이너의 **min** 리소스 제약 조건보다 크거나 같아야 합니다.
 - 컨테이너 **CPU** 또는 메모리 요청 및 제한이 **LimitRange** 오브젝트에 지정된 컨테이너의 **max** 리소스 제약 조건보다 작거나 같아야 합니다.
- LimitRange** 오브젝트에서 **max CPU**를 정의하는 경우 **Pod** 사양에 **CPU request** 값을 정의할 필요가 없습니다. 그러나 제한 범위에 지정된 최대 **CPU** 제약 조건을 충족하는 **CPU limit** 값을 지정해야 합니다.
- 요청에 대한 컨테이너 제한 비율은 **LimitRange** 오브젝트에 지정된 컨테이너의 **maxLimitRequestRatio** 값보다 작거나 같아야 합니다.

LimitRange 오브젝트에서 **maxLimitRequestRatio** 제약 조건을 정의하는 경우 새 컨테이너에 **request** 및 **limit** 값이 모두 있어야 합니다. **OpenShift Container Platform**은 **limit** 값을 **request** 값으로 나눠 제한 대 요청 비율을 계산합니다. 이 값은 음수가 아닌 1보다 큰 정수여야 합니다.

예를 들어 컨테이너의 **limit** 값이 **cpu: 500**이고 **request** 값이 **cpu: 100**인 경우 **cpu**의 제한 대 요청 비율은 5입니다. 이 비율은 **maxLimitRequestRatio**보다 작거나 같아야 합니다.

Pod 사양에서 컨테이너 리소스 메모리 또는 제한을 지정하지 않으면 제한 범위 오브젝트에 지정된 컨테이너의 **default** 또는 **defaultRequest CPU** 및 메모리 값이 컨테이너에 할당됩니다.

컨테이너 **LimitRange** 오브젝트 정의

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
```

```

- type: "Container"
  max:
    cpu: "2" 2
    memory: "1Gi" 3
  min:
    cpu: "100m" 4
    memory: "4Mi" 5
  default:
    cpu: "300m" 6
    memory: "200Mi" 7
  defaultRequest:
    cpu: "200m" 8
    memory: "100Mi" 9
  maxLimitRequestRatio:
    cpu: "10" 10

```

1

LimitRange 오브젝트의 이름입니다.

2

Pod의 단일 컨테이너에서 요청할 수 있는 최대 **CPU** 양입니다.

3

Pod의 단일 컨테이너에서 요청할 수 있는 최대 메모리 양입니다.

4

Pod의 단일 컨테이너에서 요청할 수 있는 최소 **CPU** 양입니다.

5

Pod의 단일 컨테이너에서 요청할 수 있는 최소 메모리 양입니다.

6

Pod 사양에 지정되지 않은 경우 컨테이너에서 사용할 수 있는 기본 **CPU** 양입니다.

7

Pod 사양에 지정되지 않은 경우 컨테이너에서 사용할 수 있는 기본 메모리 양입니다.

8

Pod 사양에 지정되지 않은 경우 컨테이너에서 요청할 수 있는 기본 CPU 양입니다.

9

Pod 사양에 지정되지 않은 경우 컨테이너에서 요청할 수 있는 기본 메모리 양입니다.

10

컨테이너에 대한 최대 제한 대 요청 비율입니다.

8.3.1.1.2. Pod 제한

제한 범위를 사용하면 지정된 프로젝트의 Pod에서 모든 컨테이너에 대해 최소 및 최대 CPU 및 메모리 제한을 지정할 수 있습니다. 프로젝트에서 컨테이너를 생성하려면 Pod 사양의 컨테이너 CPU 및 메모리 요청이 LimitRange 오브젝트에 설정된 값을 준수해야 합니다. 그렇지 않으면 Pod가 생성되지 않습니다.

Pod 사양에서 컨테이너 리소스 메모리 또는 제한을 지정하지 않으면 제한 범위 오브젝트에 지정된 컨테이너의 default 또는 defaultRequest CPU 및 메모리 값이 컨테이너에 할당됩니다.

Pod의 모든 컨테이너에서 다음 사항이 충족되어야 합니다.

- 컨테이너 CPU 또는 메모리에 대한 요청 및 제한이 LimitRange 오브젝트에 지정된 Pod의 min 리소스 제약 조건보다 크거나 같아야 합니다.
- 컨테이너 CPU 또는 메모리에 대한 요청 및 제한이 LimitRange 오브젝트에 지정된 Pod의 max 리소스 제약 조건보다 작거나 같아야 합니다.
- 요청에 대한 컨테이너 제한 대 요청 비율이 LimitRange 오브젝트에 지정된 maxLimitRequestRatio 제약 조건보다 작거나 같아야 합니다.

Pod LimitRange 오브젝트 정의

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
```



```

name: "resource-limits" ❶
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" ❷
        memory: "1Gi" ❸
      min:
        cpu: "200m" ❹
        memory: "6Mi" ❺
    maxLimitRequestRatio:
      cpu: "10" ❻

```

❶

제한 범위 오브젝트의 이름입니다.

❷

Pod에서 모든 컨테이너에 요청할 수 있는 최대 CPU 양입니다.

❸

Pod에서 모든 컨테이너에 요청할 수 있는 최대 메모리 양입니다.

❹

Pod에서 모든 컨테이너에 요청할 수 있는 최소 CPU 양입니다.

❺

Pod에서 모든 컨테이너에 요청할 수 있는 최소 메모리 양입니다.

❻

컨테이너에 대한 최대 제한 대 요청 비율입니다.

8.3.1.1.3. 이미지 제한

LimitRange 오브젝트를 사용하면 **OpenShift** 이미지 레지스트리로 내보낼 수 있는 이미지의 최대 크기를 지정할 수 있습니다.

OpenShift 이미지 레지스트리로 이미지를 내보내는 경우 다음 사항이 충족되어야 합니다.

- 이미지 크기가 **LimitRange** 오브젝트에 지정된 이미지의 **max** 크기보다 작거나 같아야 합니다.

이미지 **LimitRange** 오브젝트 정의

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi 2

```

1

LimitRange 오브젝트의 이름입니다.

2

OpenShift 이미지 레지스트리로 내보낼 수 있는 이미지의 최대 크기입니다.



참고

제한을 초과하는 **Blob**이 레지스트리에 업로드되지 않도록 하려면 할당량을 적용하도록 레지스트리를 구성해야 합니다.



주의

업로드된 이미지의 매니페스트에서 이미지 크기를 항상 사용할 수 있는 것은 아닙니다. 특히 **Docker 1.10** 이상으로 빌드하여 **v2** 레지스트리로 내보낸 이미지의 경우 그러합니다. 이전 **Docker** 데몬을 사용하여 이러한 이미지를 가져오면 레지스트리에서 이미지 매니페스트를 모든 크기 정보가 없는 스키마 **v1**로 변환합니다. 이미지에 스토리지 제한이 설정되어 있지 않아 업로드할 수 없습니다.

문제가 처리되고 있습니다.

8.3.1.1.4. 이미지 스트림 제한

LimitRange 오브젝트를 사용하면 이미지 스트림에 대한 제한을 지정할 수 있습니다.

각 이미지 스트림에서 다음 사항이 충족되어야 합니다.

- **ImageStream** 사양의 이미지 태그 수가 **LimitRange** 오브젝트의 **openshift.io/image-tags** 제약 조건보다 작거나 같아야 합니다.
- **ImageStream** 사양의 이미지에 대한 고유 참조 수가 제한 범위 오브젝트의 **openshift.io/images** 제약 조건보다 작거나 같아야 합니다.

이미지 스트림 **LimitRange** 오브젝트 정의

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 2
        openshift.io/images: 30 3
```

1

LimitRange 오브젝트의 이름입니다.

2

imagestream 사양의 **imagestream.spec.tags** 매개변수에 있는 최대 고유 이미지 태그 수입니다.

3

imagestream 사양의 **imagestream.status.tags** 매개변수에 있는 최대 고유 이미지 참조 수입니다.

openshift.io/image-tags 리소스는 고유 이미지 참조를 나타냅니다. 사용 가능한 참조는 **ImageStreamTag**, **ImageStreamImage**, **DockerImage**입니다. 태그는 **oc tag** 및 **oc import-image** 명령을 사용하여 생성할 수 있습니다. 내부 참조와 외부 참조는 구분되지 않습니다. 그러나 **ImageStream** 사양에 태그된 각각의 고유 참조는 한 번만 계산됩니다. 내부 컨테이너 이미지 레지스트리에 대한 내보내기는 어떤 방식으로든 제한하지 않지만 태그 제한에 유용합니다.

openshift.io/images 리소스는 이미지 스트림 상태에 기록된 고유 이미지 이름을 나타냅니다. 이를 통해 **OpenShift** 이미지 레지스트리로 내보낼 수 있는 여러 이미지를 제한할 수 있습니다. 내부 및 외부 참조는 구분되지 않습니다.

8.3.1.1.5. 영구 볼륨 클레임 제한

LimitRange 오브젝트를 사용하여 **PVC(영구 볼륨 클레임)**에 요청된 스토리지를 제한할 수 있습니다.

프로젝트의 모든 영구 볼륨 클레임에서 다음 사항이 충족되어야 합니다.

- **PVC(영구 볼륨 클레임)**의 리소스 요청이 **LimitRange** 오브젝트에 지정된 **PVC**의 **min** 제약 조건보다 크거나 같아야 합니다.
- **PVC(영구 볼륨 클레임)**의 리소스 요청이 **LimitRange** 오브젝트에 지정된 **PVC**의 **max** 제약 조건보다 작거나 같아야 합니다.

PVC LimitRange 오브젝트 정의

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" ❶
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi" ❷
      max:
        storage: "50Gi" ❸

```

❶

LimitRange 오브젝트의 이름입니다.

❷

영구 볼륨 클레임에서 요청할 수 있는 최소 스토리지 양입니다.

❸

영구 볼륨 클레임에서 요청할 수 있는 최대 스토리지 양입니다.

8.3.2. 제한 범위 생성

프로젝트에 제한 범위를 적용하려면 다음을 수행합니다.

1.

필요한 사양을 사용하여 **LimitRange** 오브젝트를 생성합니다.

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" ❶
spec:
  limits:
    - type: "Pod" ❷
      max:
        cpu: "2"
        memory: "1Gi"

```

```

min:
  cpu: "200m"
  memory: "6Mi"
- type: "Container" 3
max:
  cpu: "2"
  memory: "1Gi"
min:
  cpu: "100m"
  memory: "4Mi"
default: 4
  cpu: "300m"
  memory: "200Mi"
defaultRequest: 5
  cpu: "200m"
  memory: "100Mi"
maxLimitRequestRatio: 6
  cpu: "10"
- type: openshift.io/Image 7
max:
  storage: 1Gi
- type: openshift.io/ImageStream 8
max:
  openshift.io/image-tags: 20
  openshift.io/images: 30
- type: "PersistentVolumeClaim" 9
min:
  storage: "2Gi"
max:
  storage: "50Gi"

```

1

LimitRange 오브젝트의 이름을 지정합니다.

2

Pod에 제한을 설정하려면 필요에 따라 최소 및 최대 **CPU** 및 메모리 요청을 지정합니다.

3

컨테이너에 제한을 설정하려면 필요에 따라 최소 및 최대 **CPU** 및 메모리 요청을 지정합니다.

4

선택 사항입니다. 컨테이너의 경우 **Pod** 사양에 지정하지 않는 경우 컨테이너에서 사용할 수 있는 기본 **CPU** 또는 메모리 양을 지정합니다.

5

6

선택 사항입니다. 컨테이너의 경우 **Pod** 사양에 지정할 수 있는 최대 제한 대 요청 비율을 지정합니다.

7

이미지 오브젝트에 대한 제한을 설정하려면 **OpenShift** 이미지 레지스트리로 내보낼 수 있는 최대 이미지 크기를 설정합니다.

8

이미지 스트림에 대한 제한을 설정하려면 필요에 따라 **ImageStream** 오브젝트 파일에 있을 수 있는 최대 이미지 태그 및 참조 수를 설정합니다.

9

영구 볼륨 클레임에 대한 제한을 설정하려면 요청할 수 있는 최소 및 최대 스토리지 양을 설정합니다.

2.

오브젝트를 생성합니다.

```
$ oc create -f <limit_range_file> -n <project> 1
```

1

생성한 **YAML** 파일의 이름과 제한을 적용할 프로젝트를 지정합니다.

8.3.3. 제한 보기

웹 콘솔에서 프로젝트의 할당량 페이지로 이동하면 프로젝트에 정의된 제한을 확인할 수 있습니다.

CLI를 사용하여 제한 범위 세부 정보를 볼 수도 있습니다.

1.

프로젝트에 정의된 **LimitRange** 오브젝트 목록을 가져옵니다. 예를 들어 **demoproject**라는 프로젝트의 경우 다음과 같습니다.

```
$ oc get limits -n demoproject
```

-

```
NAME          CREATED AT
resource-limits 2020-07-15T17:14:23Z
```

2.

관심 있는 **LimitRange** 오브젝트를 설명합니다(예: **resource-limits** 제한 범위).

```
$ oc describe limits resource-limits -n demoproject
```

```
Name:          resource-limits
Namespace:     demoproject
Type          Resource      Min   Max   Default Request Default Limit
Max Limit/Request Ratio
-----
Pod           cpu                200m  2    -     -     -
Pod           memory            6Mi   1Gi  -     -     -
Container    cpu               100m  2    200m  300m  10
Container    memory           4Mi   1Gi  100Mi 200Mi  -
openshift.io/Image      storage          -     1Gi  -     -     -
openshift.io/ImageStream openshift.io/image -     12  -     -     -
openshift.io/ImageStream openshift.io/image-tags -    10  -     -     -
PersistentVolumeClaim  storage          -    50Gi -     -     -
```

8.3.4. 제한 범위 삭제

활성 **LimitRange** 오브젝트를 제거하여 더 이상 프로젝트에 제한을 적용하지 않으려면 다음을 수행합니다.

- 다음 명령을 실행합니다.

```
$ oc delete limits <limit_name>
```

8.4. 컨테이너 메모리 및 위험 요구 사항을 충족하도록 클러스터 메모리 구성

클러스터 관리자는 다음과 같은 방법으로 애플리케이션 메모리를 관리하여 클러스터를 효율적으로 작동할 수 있습니다.

- 컨테이너화된 애플리케이션 구성 요소의 메모리 및 위험 요구 사항을 확인하고 해당 요구 사항에 맞게 컨테이너 메모리 매개변수를 구성합니다.
- 구성된 컨테이너 메모리 매개변수를 최적으로 준수하도록 컨테이너화된 애플리케이션 런타임(예: **OpenJDK**)을 구성합니다.

-

컨테이너에서 실행과 연결된 메모리 관련 오류 조건을 진단 및 해결합니다.

8.4.1. 애플리케이션 메모리 관리 이해

계속하기 전에 **OpenShift Container Platform**에서 컴퓨팅 리소스를 관리하는 방법에 대한 개요를 꼼꼼히 확인하는 것이 좋습니다.

각 유형의 리소스(메모리, **CPU**, 스토리지)에 대해 **OpenShift Container Platform**에서는 선택적인 요청 및 제한 값을 **Pod**의 각 컨테이너에 배치할 수 있습니다.

메모리 요청 및 메모리 제한에 대해 다음 사항에 유의하십시오.

-

메모리 요청

-

메모리 요청 값을 지정하면 **OpenShift Container Platform** 스케줄러에 영향을 미칩니다. 스케줄러는 노드에 컨테이너를 예약할 때 메모리 요청을 고려한 다음 컨테이너 사용을 위해 선택한 노드에서 요청된 메모리를 차단합니다.

-

노드의 메모리가 소모되면 **OpenShift Container Platform**에서 메모리 사용량이 메모리 요청을 가장 많이 초과하는 컨테이너를 제거하는 작업에 우선순위를 부여합니다. 메모리 소모가 심각한 경우 노드 **OOM** 종료자는 유사한 메트릭을 기반으로 컨테이너에서 프로세스를 선택하고 종료할 수 있습니다.

-

클러스터 관리자는 메모리 요청 값에 할당량을 할당하거나 기본값을 할당할 수 있습니다.

-

클러스터 관리자는 클러스터 과다 할당을 관리하기 위해 개발자가 지정하는 메모리 요청 값을 덮어쓸 수 있습니다.

-

메모리 제한

-

메모리 제한 값을 지정하면 컨테이너의 모든 프로세스에 할당될 수 있는 메모리에 대한 하드 제한을 제공합니다.

-

컨테이너의 모든 프로세스에서 할당된 메모리가 메모리 제한을 초과하면 노드의 OOM(Out of Memory) 종료자에서 즉시 컨테이너의 프로세스를 선택하여 종료합니다.

- 메모리 요청 및 제한을 둘 다 지정하면 메모리 제한 값이 메모리 요청보다 크거나 같아야 합니다.
- 클러스터 관리자는 메모리 제한 값에 할당량을 할당하거나 기본값을 할당할 수 있습니다.
- 최소 메모리 제한은 12MB입니다. 메모리를 할당할 수 없음 Pod 이벤트로 인해 컨테이너가 시작되지 않으면 메모리 제한이 너무 낮은 것입니다. 메모리 제한을 늘리거나 제거합니다. 제한을 제거하면 Pod에서 바인딩되지 않은 노드 리소스를 사용할 수 있습니다.

8.4.1.1. 애플리케이션 메모리 전략 관리

OpenShift Container Platform에서 애플리케이션 메모리 크기를 조정하는 단계는 다음과 같습니다.

1. 예상되는 컨테이너 메모리 사용량 확인

필요한 경우 경험적으로 예상되는 평균 및 최대 컨테이너 메모리 사용량을 결정합니다(예: 별도의 부하 테스트를 통해). 컨테이너에서 잠재적으로 병렬로 실행될 수 있는 모든 프로세스를 고려해야 합니다(예: 기본 애플리케이션에서 보조 스크립트를 생성하는지의 여부).

2. 위험 유형 확인

제거와 관련된 위험 유형을 확인합니다. 위험 성향이 낮으면 컨테이너는 예상되는 최대 사용량과 백분율로 된 안전 범위에 따라 메모리를 요청해야 합니다. 위험 성향이 높으면 예상되는 사용량에 따라 메모리를 요청하는 것이 더 적합할 수 있습니다.

3. 컨테이너 메모리 요청 설정

위 내용에 따라 컨테이너 메모리 요청을 설정합니다. 요청이 애플리케이션 메모리 사용량을 더 정확하게 나타낼수록 좋습니다. 요청이 너무 높으면 클러스터 및 할당량 사용이 비효율적입니다. 요청이 너무 낮으면 애플리케이션 제거 가능성이 커집니다.

4. 필요한 경우 컨테이너 메모리 제한 설정

필요한 경우 컨테이너 메모리 제한을 설정합니다. 제한을 설정하면 컨테이너에 있는 모든 프로세스의 메모리 사용량 합계가 제한을 초과하는 경우 컨테이너 프로세스가 즉시 종료되는 효과가 있어 이로 인한 장단점이 발생합니다. 다른 한편으로는 예상치 못한 과도한 메모리 사용을 조기에 확인할 수 있습니다(“빠른 실패”). 그러나 이로 인해 프로세스가 갑자기 종료됩니다.

일부 **OpenShift Container Platform** 클러스터에는 제한 값을 설정해야 할 수 있습니다. 일부는 제한에 따라 요청을 덮어쓸 수 있습니다. 일부 애플리케이션 이미지에서는 요청 값보다 탐지하기 쉬운 설정된 제한 값을 사용합니다.

메모리 제한을 설정하는 경우 예상되는 최대 컨테이너 메모리 사용량과 백분율로 된 안전 범위 이상으로 설정해야 합니다.

5. 애플리케이션이 튜닝되었는지 확인

적절한 경우 구성된 요청 및 제한 값과 관련하여 애플리케이션이 튜닝되었는지 확인합니다. 이 단계는 특히 JVM과 같이 메모리를 풀링하는 애플리케이션과 관련이 있습니다. 이 페이지의 나머지 부분에서는 이 작업에 대해 설명합니다.

추가 리소스

- [컴퓨팅 리소스 및 컨테이너 이해](#)

8.4.2. OpenShift Container Platform에 대한 OpenJDK 설정 이해

기본 **OpenJDK** 설정은 컨테이너화된 환경에서 제대로 작동하지 않습니다. 따라서 컨테이너에서 **OpenJDK**를 실행할 때마다 몇 가지 추가 **Java** 메모리 설정을 항상 제공해야 합니다.

JVM 메모리 레이아웃은 복잡하고 버전에 따라 다르며 자세한 설명은 이 문서의 범위를 벗어납니다. 그러나 최소한 다음 세 가지 메모리 관련 작업은 컨테이너에서 **OpenJDK**를 실행하기 위한 시작점으로서 중요합니다.

1. **JVM** 최대 힙 크기를 덮어씁니다.
2. 적절한 경우 **JVM**에서 사용하지 않는 메모리를 운영 체제에 제공하도록 유도합니다.

3.

컨테이너 내의 모든 JVM 프로세스가 적절하게 구성되었는지 확인합니다.

컨테이너에서 실행하기 위해 JVM 워크로드를 최적으로 튜닝하는 것은 이 문서의 범위를 벗어나며 다양한 JVM 옵션을 추가로 설정하는 작업이 포함될 수 있습니다.

8.4.2.1. JVM 최대 힙 크기를 덮어쓰는 방법 이해

대다수의 Java 워크로드에서 JVM 힙은 메모리를 가장 많이 사용하는 단일 소비 항목입니다. 현재 OpenJDK는 기본적으로 OpenJDK가 컨테이너에서 실행되는지의 여부와 관계없이 컴퓨팅 노드 메모리의 최대 1/4(1/-XX:MaxRAMFraction)을 힙에 사용할 수 있도록 허용합니다. 따라서 특히 컨테이너 메모리 제한도 설정되어 있는 경우 이 동작을 덮어쓰는 것이 중요합니다.

위 작업은 두 가지 이상의 방법으로 수행할 수 있습니다.

- 컨테이너 메모리 제한이 설정되어 있고 JVM에서 실험 옵션을 지원하는 경우 -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap 을 설정합니다.



참고

UseCGroupMemoryLimitForHeap 옵션이 JDK 11에서 제거되었습니다. 대신 -XX:+UseContainerSupport를 사용합니다.

이 명령은 -XX:MaxRAM을 컨테이너 메모리 제한으로 설정하고 최대 힙 크기(-XX:MaxHeapSize / -Xmx)를 1/-XX:MaxRAMFraction(기본값: 1/4)으로 설정합니다.

- -XX:MaxRAM, -XX:MaxHeapSize 또는 -Xmx 중 하나를 직접 덮어씁니다.

이 옵션을 수행하려면 값을 하드 코딩해야 하지만 안전한 여백을 계산할 수 있다는 장점이 있습니다.

8.4.2.2. JVM에서 사용하지 않는 메모리를 운영 체제에 제공하도록 유도하는 방법 이해

기본적으로 OpenJDK는 사용하지 않는 메모리를 운영 체제에 적극적으로 반환하지 않습니다. 이는 대다수의 컨테이너화된 Java 워크로드에 적합할 수 있습니다. 그러나 추가 프로세스가 네이티브인지 추가 JVM인지 또는 이 둘의 조합인지와 관계없이 컨테이너 내에서 추가 활성 프로세스가 JVM과 공존하는 워크로드는 주목할 만한 예외입니다.

Java 기반 에이전트는 다음 JVM 인수를 사용하여 JVM에서 사용하지 않는 메모리를 운영 체제에 제공하도록 유도할 수 있습니다.

```
-XX:+UseParallelGC  
-XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4  
-XX:AdaptiveSizePolicyWeight=90.
```

이러한 인수는 할당된 메모리가 사용 중인 메모리의 110%(-XX:MaxHeapFreeRatio)를 초과할 때마다 힙 메모리를 운영 체제에 반환하기 위한 것으로, 가비지 수집기에서 최대 20%(-XX:GCTimeRatio)의 CPU 시간을 사용합니다. 애플리케이션 힙 할당은 항상 초기 힙 할당(-XX:InitialHeapSize / -Xms로 덮어 씌움)보다 적지 않습니다. 자세한 내용은 [OpenShift에서 Java 플랫폼 튜닝\(1부\)](#), [OpenShift에서 Java 플랫폼 튜닝\(2부\)](#), [OpenJDK 및 컨테이너에서 확인할 수 있습니다](#).

8.4.2.3. 컨테이너 내의 모든 JVM 프로세스를 적절하게 구성하는 방법 이해

동일한 컨테이너에서 여러 개의 JVM이 실행되는 경우 모든 JVM이 올바르게 구성되어 있는지 확인해야 합니다. 워크로드가 많은 경우 각 JVM에 백분율로 된 메모리 예산을 부여하여 추가 안전 범위를 충분히 유지해야 합니다.

대부분의 Java 틀에서는 다양한 환경 변수(JAVA_OPTS, GRADLE_OPTS 등)를 사용하여 JVM을 구성하고 올바른 설정을 올바른 JVM으로 전달하는 것이 어려울 수 있습니다.

OpenJDK는 항상 JAVA_TOOL_OPTIONS 환경 변수를 준수하고 JAVA_TOOL_OPTIONS에 지정된 값은 JVM 명령줄에 지정된 다른 옵션에서 덮어씌웁니다. 기본적으로 이러한 옵션이 Java 기반 에이전트 이미지에서 실행되는 모든 JVM 워크로드에 기본적으로 사용되도록 OpenShift Container Platform Jenkins Maven 에이전트 이미지 세트는 다음과 같습니다.

```
JAVA_TOOL_OPTIONS="-XX:+UnlockExperimentalVMOptions  
-XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true"
```



참고

UseCGroupMemoryLimitForHeap 옵션이 JDK 11에서 제거되었습니다. 대신 -XX:+UseContainerSupport를 사용합니다.

이러한 설정을 통해 추가 옵션이 필요하지 않다고 보장할 수는 없지만 유용한 시작점이 될 수 있습니다.

8.4.3. Pod 내에서 메모리 요청 및 제한 찾기

Pod 내에서 메모리 요청 및 제한을 동적으로 검색하려는 애플리케이션에서는 **Downward API**를 사용해야 합니다.

프로세스

1.

MEMORY_REQUEST 및 **MEMORY_LIMIT** 스탠자를 추가하도록 **Pod**를 구성합니다.

a.

다음과 유사한 **YAML** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: test
    image: fedora:latest
    command:
    - sleep
    - "3600"
    env:
    - name: MEMORY_REQUEST 1
      valueFrom:
        resourceFieldRef:
          containerName: test
          resource: requests.memory
    - name: MEMORY_LIMIT 2
      valueFrom:
        resourceFieldRef:
          containerName: test
          resource: limits.memory
  resources:
    requests:
      memory: 384Mi
    limits:
      memory: 512Mi
```

1

이 스탠자를 추가하여 애플리케이션 메모리 요청 값을 검색합니다.

2

이 스탠자를 추가하여 애플리케이션 메모리 제한 값을 검색합니다.

b.

다음 명령을 실행하여 Pod를 생성합니다.

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

검증

1.

원격 셸을 사용하여 Pod에 액세스합니다.

```
$ oc rsh test
```

2.

요청된 값이 적용되었는지 확인합니다.

```
$ env | grep MEMORY | sort
```

출력 예

```
MEMORY_LIMIT=536870912
MEMORY_REQUEST=402653184
```



참고

메모리 제한 값은 `/sys/fs/cgroup/memory/memory.limit_in_bytes` 파일을 통해 컨테이너 내부에서도 확인할 수 있습니다.

8.4.4. OOM 종료 정책 이해

컨테이너에 있는 모든 프로세스의 총 메모리 사용량이 메모리 제한을 초과하거나 노드의 메모리 부족 상태가 심각한 경우에는 **OpenShift Container Platform**에서 컨테이너의 프로세스를 종료할 수 있습니다.

프로세스가 **OOM(Out of Memory)** 종료되면 컨테이너가 즉시 종료될 수 있습니다. 컨테이너 **PID 1** 프로세스에서 **SIGKILL**을 수신하면 컨테이너가 즉시 종료됩니다. 그 외에는 컨테이너 동작이 기타 프로세스의 동작에 따라 달라집니다.

예를 들어 컨테이너 프로세스가 코드 137로 종료되면 **SIGKILL** 신호가 수신되었음을 나타냅니다.

컨테이너가 즉시 종료되지 않으면 다음과 같이 **OOM** 종료를 탐지할 수 있습니다.

1. 원격 셸을 사용하여 Pod에 액세스합니다.

```
# oc rsh test
```

2. 다음 명령을 실행하여 `/sys/fs/cgroup/memory/memory.oom_control`에서 현재 **OOM** 종료 수를 확인합니다.

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

출력 예

```
oom_kill 0
```

3. 다음 명령을 실행하여 **OOM** 종료를 유도합니다.

```
$ sed -e " </dev/zero
```

출력 예

```
Killed
```

4. 다음 명령을 실행하여 **sed** 명령의 종료 상태를 확인합니다.

```
$ echo $?
```


출력 예

137

137 코드는 컨테이너 프로세스가 코드 137로 종료되었음을 나타냅니다. 이 코드는 **SIGKILL** 신호가 수신되었음을 나타냅니다.

5.

다음 명령을 실행하여 `/sys/fs/cgroup/memory/memory.oom_control`에서 **OOM** 종료 카운터가 증가했는지 확인합니다.

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

출력 예

oom_kill 1

Pod에서 하나 이상의 프로세스가 **OOM** 종료된 경우 나중에 **Pod**가 종료되면(즉시 여부와 관계없이) 단계는 실패, 이유는 **OOM** 종료가 됩니다. **restartPolicy** 값에 따라 **OOM** 종료된 **Pod**가 다시 시작될 수 있습니다. 재시작되지 않는 경우 복제 컨트롤러와 같은 컨트롤러는 **Pod**의 실패 상태를 확인하고 새 **Pod**를 생성하여 이전 **Pod**를 교체합니다.

다음 명령을 사용하여 **Pod** 상태를 가져옵니다.

```
$ oc get pod test
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
test	0/1	OOMKilled	0	1m

- *Pod가 재시작되지 않은 경우 다음 명령을 실행하여 Pod를 확인합니다.*

```
$ oc get pod test -o yaml
```

출력 예

```
...  
status:  
  containerStatuses:  
  - name: test  
    ready: false  
    restartCount: 0  
  state:  
    terminated:  
      exitCode: 137  
      reason: OOMKilled  
  phase: Failed
```

- *재시작된 경우 다음 명령을 실행하여 Pod를 확인합니다.*

```
$ oc get pod test -o yaml
```

출력 예

```
...  
status:  
  containerStatuses:  
  - name: test  
    ready: true  
    restartCount: 1  
  lastState:  
    terminated:  
      exitCode: 137  
      reason: OOMKilled  
  state:  
    running:  
  phase: Running
```

8.4.5. Pod 제거 이해

노드의 메모리가 소모되면 **OpenShift Container Platform**은 해당 노드에서 **Pod**를 제거할 수 있습니다. 메모리 소모 범위에 따라 제거가 정상적으로 수행되지 않을 수 있습니다. 정상적인 제거에서는 프로세스가 아직 종료되지 않은 경우 각 컨테이너의 기본 프로세스(**PID 1**)에서 **SIGTERM** 신호를 수신한 다음 잠시 후 **SIGKILL** 신호를 수신합니다. 비정상적인 제거에서는 각 컨테이너의 기본 프로세스에서 **SIGKILL** 신호를 즉시 수신합니다.

제거된 **Pod**의 단계는 실패, 이유는 제거됨입니다. **restartPolicy** 값과 관계없이 재시작되지 않습니다. 그러나 복제 컨트롤러와 같은 컨트롤러는 **Pod**의 실패 상태를 확인하고 새 **Pod**를 생성하여 이전 **Pod**를 교체합니다.

```
$ oc get pod test
```

출력 예

```
NAME      READY   STATUS    RESTARTS  AGE
test      0/1     Evicted  0          1m
```

```
$ oc get pod test -o yaml
```

출력 예

```
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
```

```
  phase: Failed
  reason: Evicted
```

8.5. 과다 할당된 노드에 **POD**를 배치하도록 클러스터 구성

과다 할당 상태에서는 컨테이너 컴퓨팅 리소스 요청 및 제한의 합계가 시스템에서 사용 가능한 리소스를 초과합니다. 예를 들어 용량에 맞게 보장된 성능을 절충할 수 있는 개발 환경에서는 과다 할당을 사용할 수 있습니다.

컨테이너는 컴퓨팅 리소스 요청 및 제한을 지정할 수 있습니다. 요청은 컨테이너 예약에 사용되며 최소 서비스 보장을 제공합니다. 제한은 노드에서 사용할 수 있는 컴퓨팅 리소스의 양을 제한합니다.

스케줄러는 클러스터의 모든 노드에서 컴퓨팅 리소스 사용을 최적화합니다. Pod의 컴퓨팅 리소스 요청 및 노드의 사용 가능한 용량을 고려하여 특정 노드에 Pod를 배치합니다.

OpenShift Container Platform 관리자는 노드에서 과다 할당 수준을 제어하고 컨테이너 밀도를 관리할 수 있습니다. [ClusterResourceOverrideOperator](#)를 사용하여 클러스터 수준 과다 할당을 구성하면 개발자 컨테이너에 설정된 요청과 제한 사이의 비율을 덮어쓸 수 있습니다. [노드 과다 할당 및 프로젝트 메모리 및 CPU 제한 및 기본값과 함께 리소스 제한 및 요청을 조정하여 원하는 수준의 과다 할당을 수행할 수 있습니다.](#)



참고

OpenShift Container Platform에서는 클러스터 수준 과다 할당을 활성화해야 합니다. 노드 과다 할당은 기본적으로 활성화되어 있습니다. [노드의 과다 할당 비활성화](#)를 참조하십시오.

8.5.1. 리소스 요청 및 과다 할당

각 컴퓨팅 리소스에 대해 컨테이너는 리소스 요청 및 제한을 지정할 수 있습니다. 노드에 요청된 값을 충족할 수 있는 충분한 용량을 확보하기 위한 요청에 따라 스케줄링 결정이 내려집니다. 컨테이너가 제한을 지정하지만 요청을 생략하면 요청은 기본적으로 제한 값으로 설정됩니다. 컨테이너가 노드에서 지정된 제한을 초과할 수 없습니다.

제한 적용은 컴퓨팅 리소스 유형에 따라 다릅니다. 컨테이너가 요청하거나 제한하지 않으면 컨테이너는 리소스 보장이 없는 상태에서 노드로 예약됩니다. 실제로 컨테이너는 가장 낮은 로컬 우선 순위로 사용 가능한 만큼의 지정된 리소스를 소비할 수 있습니다. 리소스가 부족한 상태에서는 리소스 요청을 지정하지 않는 컨테이너에 가장 낮은 수준의 QoS (Quality of Service)가 설정됩니다.

예약은 요청된 리소스를 기반으로 하는 반면 할당량 및 하드 제한은 리소스 제한을 나타내며 이는 요청된 리소스보다 높은 값으로 설정할 수 있습니다. 요청과 제한의 차이에 따라 오버 커밋 수준이 결정됩니다. 예를 들어, 컨테이너에 1Gi의 메모리 요청과 2Gi의 메모리 제한이 지정되면 노드에서 사용 가능한 1Gi 요청에 따라 컨테이너가 예약되지만 최대 2Gi를 사용할 수 있습니다. 따라서 이 경우 200% 오버 커밋되는 것입니다.

8.5.2. Cluster Resource Override Operator를 사용한 클러스터 수준 오버 커밋

Cluster Resource Override Operator는 클러스터의 모든 노드에서 오버 커밋 수준을 제어하고 컨테이너 밀도를 관리할 수 있는 승인 **Webhook**입니다. **Operator**는 특정 프로젝트의 노드가 정의된 메모리 및 **CPU** 한계를 초과하는 경우에 대해 제어합니다.

다음 섹션에 설명된대로 **OpenShift Container Platform** 콘솔 또는 **CLI**를 사용하여 **Cluster Resource Override Operator**를 설치해야 합니다. 설치하는 동안 다음 예에 표시된 것처럼 오버 커밋 수준을 설정하는 **ClusterResourceOverride** 사용자 지정 리소스 (**CR**)를 만듭니다.

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster ①
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ②
      cpuRequestToLimitPercent: 25 ③
      limitCPUMemoryPercent: 200 ④
# ...
```

1

이름은 **instance**이어야 합니다.

2

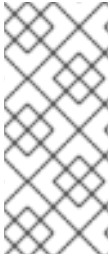
선택 사항입니다. 컨테이너 메모리 제한이 지정되어 있거나 기본값으로 설정된 경우 메모리 요청이 제한 백분율 (1-100)로 덮어 쓰기됩니다. 기본값은 **50**입니다.

3

선택 사항입니다. 컨테이너 **CPU** 제한이 지정되어 있거나 기본값으로 설정된 경우 **CPU** 요청이 1-100 사이의 제한 백분율로 덮어 쓰기됩니다. 기본값은 **25**입니다.

4

선택 사항입니다. 컨테이너 메모리 제한이 지정되어 있거나 기본값으로 설정된 경우, **CPU** 제한이 지정되어 있는 경우 메모리 제한의 백분율로 덮어 쓰기됩니다. **1Gi**의 **RAM**을 **100 %**로 스케일링하는 것은 **1** 개의 **CPU** 코어와 같습니다. **CPU** 요청을 재정의하기 전에 처리됩니다 (설정된 경우). 기본값은 **200**입니다.



참고

컨테이너에 제한이 설정되어 있지 않은 경우 **Cluster Resource Override Operator** 덮어쓰기가 적용되지 않습니다. 프로젝트별 기본 제한이 있는 **LimitRange** 오브젝트를 생성하거나 **Pod** 사양에 제한을 구성하여 덮어쓰기를 적용하십시오.

각 프로젝트의 네임 스페이스 오브젝트에 다음 라벨을 적용하여 프로젝트별로 덮어쓰기를 활성화할 수 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
# ...

labels:
  clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"
# ...
```

Operator는 **ClusterResourceOverride CR**을 감시하고 **ClusterResourceOverride** 승인 **Webhook**가 **operator**와 동일한 네임 스페이스에 설치되어 있는지 확인합니다.

8.5.2.1. 웹 콘솔을 사용하여 Cluster Resource Override Operator 설치

OpenShift Container Platform 웹 콘솔을 사용하여 **Cluster Resource Override Operator**를 설치하여 클러스터의 오버 커밋을 제어할 수 있습니다.

사전 요구 사항

- 컨테이너에 제한이 설정되어 있지 않은 경우 **Cluster Resource Override Operator**에 영향을 주지 않습니다. 덮어쓰기를 적용하려면 **LimitRange** 오브젝트를 사용하여 프로젝트의 기본 제한을 지정하거나 **Pod** 사양에 제한을 구성해야 합니다.

프로세스

OpenShift Container Platform 웹 콘솔을 사용하여 **Cluster Resource Override Operator**를 설치합니다.

1. **OpenShift Container Platform** 웹 콘솔에서 **Home** → **Projects**로 이동합니다.

- a. **Create Project**를 클릭합니다.
 - b. **clusterresourceoverride-operator**를 프로젝트 이름으로 지정합니다.
 - c. **Create**를 클릭합니다.
2. **Operators** → **OperatorHub**로 이동합니다.
- a. 사용 가능한 **Operator** 목록에서 **ClusterResourceOverride Operator**를 선택한 다음 **Install**을 클릭합니다.
 - b. **Operator** 설치 페이지에서 설치 모드에 대해 클러스터의 특정 네임스페이스가 선택되어 있는지 확인합니다.
 - c. **Installed Namespace**에 대해 **clusterresourceoverride-operator**가 선택되어 있는지 확인합니다.
 - d. **Update Channel** 및 **Approval Strategy**를 선택합니다.
 - e. 설치를 클릭합니다.
3. **Installed Operators** 페이지에서 **ClusterResourceOverride**를 클릭합니다.
- a. **ClusterResourceOverride Operator** 세부 정보 페이지에서 **Create ClusterResourceOverride** 를 클릭합니다.
 - b. **Create ClusterResourceOverride** 페이지에서 **YAML** 보기를 클릭하고 **YAML** 템플릿을 편집하여 필요에 따라 오버 커밋 값을 설정합니다.

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster ①

```

```
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 2
      cpuRequestToLimitPercent: 25 3
      limitCPUMemoryPercent: 200 4
# ...
```

1

이름은 **instance** 이어야 합니다.

2

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 1-100 사이의 백분율로 지정합니다. 기본값은 50입니다.

3

선택 사항입니다. 컨테이너 CPU 제한을 덮어 쓰기하는 경우 1-100 사이의 백분율로 지정합니다. 기본값은 25입니다.

4

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 백분율로 지정합니다 (사용되는 경우). 1Gi의 RAM을 100 %로 스케일링하는 것은 1 개의 CPU 코어와 같습니다. CPU 요청을 덮어 쓰기하기 전에 처리됩니다 (설정된 경우). 기본값은 200입니다.

c.

Create를 클릭합니다.

4.

클러스터 사용자 정의 리소스 상태를 확인하여 승인 **Webhook**의 현재 상태를 확인합니다.

a.

ClusterResourceOverride Operator 페이지에서 **cluster**를 클릭합니다.

b.

ClusterResourceOverride Details 페이지에서 **YAML** 을 클릭합니다. **webhook** 호출 시 **mutatingWebhookConfigurationRef** 섹션이 표시됩니다.

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
```



```

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride",
"metadata":{"annotations":{},"name":"cluster"},"spec":
{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
creationTimestamp: "2019-12-18T22:35:02Z"
generation: 1
name: cluster
resourceVersion: "127622"
selfLink:
/apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
podResourceOverride:
spec:
cpuRequestToLimitPercent: 25
limitCPUToMemoryPercent: 200
memoryRequestToLimitPercent: 50
status:

# ...

mutatingWebhookConfigurationRef: 1
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
name: clusterresourceoverrides.admission.autoscaling.openshift.io
resourceVersion: "127621"
uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...

```

1

ClusterResourceOverride 승인 Webhook 참조

8.5.2.2. CLI를 사용하여 Cluster Resource Override Operator 설치

OpenShift Container Platform CLI를 사용하여 Cluster Resource Override Operator를 설치하면 클러스터의 오버 커밋을 제어할 수 있습니다.

사전 요구 사항

- 컨테이너에 제한이 설정되어 있지 않은 경우 Cluster Resource Override Operator에 영향을 주지 않습니다. 덮어쓰기를 적용하려면 LimitRange 오브젝트를 사용하여 프로젝트의 기본 제한을 지정하거나 Pod 사양에 제한을 구성해야 합니다.

프로세스

CLI를 사용하여 **Cluster Resource Override Operator**를 설치하려면 다음을 수행합니다.

1.

Cluster Resource Override Operator의 네임스페이스를 생성합니다.

a.

Cluster Resource Override Operator의 **Namespace** 오브젝트 **YAML** 파일(예: **cro-namespace.yaml**)을 생성합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: clusterresourceoverride-operator
```

b.

네임스페이스를 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f cro-namespace.yaml
```

2.

Operator 그룹을 생성합니다.

a.

Cluster Resource Override Operator의 **OperatorGroup** 오브젝트 **YAML** 파일(예: **cro-og.yaml**)을 생성합니다.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: clusterresourceoverride-operator
  namespace: clusterresourceoverride-operator
spec:
  targetNamespaces:
    - clusterresourceoverride-operator
```

b.

Operator 그룹을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f cro-og.yaml
```

3.

서브스크립션을 생성합니다.

a.

Cluster Resource Override Operator의 **Subscription** 오브젝트 **YAML** 파일(예: **cro-sub.yaml**)을 생성합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "4.12"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b.

서브스크립션을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f cro-sub.yaml
```

4.

clusterresourceoverride-operator 네임 스페이스에서 **ClusterResourceOverride** 사용자 지정 리소스 (CR) 오브젝트를 만듭니다.

a.

clusterresourceoverride-operator 네임 스페이스로 변경합니다.

```
$ oc project clusterresourceoverride-operator
```

b.

Cluster Resource Override Operator의 **ClusterResourceOverride** 오브젝트 **YAML** 파일 (예: **cro-cr.yaml**)을 만듭니다.

```
apiVersion: operator.autoscaling.openshift.io/v1
```

```
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 2
      cpuRequestToLimitPercent: 25 3
      limitCPUMemoryPercent: 200 4
```

1

이름은 **instance** 이어야 합니다.

2

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 **1-100** 사이의 백분율로 지정합니다. 기본값은 **50**입니다.

3

선택 사항입니다. 컨테이너 **CPU** 제한을 덮어 쓰기하는 경우 **1-100** 사이의 백분율로 지정합니다. 기본값은 **25**입니다.

4

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 백분율로 지정합니다 (사용되는 경우). **1Gi**의 **RAM**을 **100 %**로 스케일링하는 것은 **1** 개의 **CPU** 코어와 같습니다. **CPU** 요청을 덮어 쓰기하기 전에 처리됩니다 (설정된 경우). 기본값은 **200**입니다.

c.

ClusterResourceOverride 오브젝트를 만듭니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f cro-cr.yaml
```

5.

클러스터 사용자 정의 리소스의 상태를 확인하여 승인 **Webhook**의 현재 상태를 확인합니다.

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

webhook 호출 시 mutatingWebhookConfigurationRef 섹션이 표시됩니다.

출력 예

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride",
"metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":
{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
creationTimestamp: "2019-12-18T22:35:02Z"
generation: 1
name: cluster
resourceVersion: "127622"
selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

mutatingWebhookConfigurationRef: 1
  apiVersion: admissionregistration.k8s.io/v1
  kind: MutatingWebhookConfiguration
  name: clusterresourceoverrides.admission.autoscaling.openshift.io
  resourceVersion: "127621"
  uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...

```

1

ClusterResourceOverride 승인 Webhook 참조

Cluster Resource Override Operator에는 **Operator**가 오버 커밋을 제어해야 하는 각 프로젝트에 대한 라벨 및 **ClusterResourceOverride** 사용자 지정 리소스 (CR)가 필요합니다.

사전 요구 사항

- 컨테이너에 제한이 설정되어 있지 않은 경우 **Cluster Resource Override Operator**에 영향을 주지 않습니다. 덮어쓰기를 적용하려면 **LimitRange** 오브젝트를 사용하여 프로젝트의 기본 제한을 지정하거나 **Pod** 사양에 제한을 구성해야 합니다.

프로세스

클러스터 수준 오버 커밋을 변경하려면 다음을 수행합니다.

1. **ClusterResourceOverride CR**을 편집합니다.

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ①
      cpuRequestToLimitPercent: 25 ②
      limitCPUMemoryPercent: 200 ③
# ...
```

①

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 1-100 사이의 백분율로 지정합니다. 기본값은 50입니다.

②

선택 사항입니다. 컨테이너 CPU 제한을 덮어 쓰기하는 경우 1-100 사이의 백분율로 지정합니다. 기본값은 25입니다.

③

선택 사항입니다. 컨테이너 메모리 제한을 덮어 쓰기하는 경우 백분율로 지정합니다 (사용되는 경우). 1Gi의 RAM을 100 %로 스케일링하는 것은 1 개의 CPU 코어와 같습니다. CPU 요청을 덮어 쓰기하기 전에 처리됩니다 (설정된 경우). 기본값은 200입니다.

- 2.

Cluster Resource Override Operator가 오버 커밋을 제어해야 하는 각 프로젝트의 네임 스페이스 오브젝트에 다음 라벨이 추가되었는지 확인합니다.

```
apiVersion: v1
kind: Namespace
metadata:
# ...

labels:
  clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" 1
# ...
```

1

이 라벨을 각 프로젝트에 추가합니다.

8.5.3. 노드 수준 오버 커밋

QoS (Quality of Service) 보장, **CPU** 제한 또는 리소스 예약과 같은 다양한 방법으로 특정 노드에서 오버 커밋을 제어할 수 있습니다. 특정 노드 및 특정 프로젝트의 오버 커밋을 비활성화할 수도 있습니다.

8.5.3.1. 컴퓨팅 리소스 및 컨테이너 이해

컴퓨팅 리소스에 대한 노드 적용 동작은 리소스 유형에 따라 다릅니다.

8.5.3.1.1. 컨테이너의 CPU 요구 이해

컨테이너에 요청된 **CPU**의 양이 보장되며 컨테이너에서 지정한 한도까지 노드에서 사용 가능한 초과 **CPU**를 추가로 소비할 수 있습니다. 여러 컨테이너가 초과 **CPU**를 사용하려고 하면 각 컨테이너에서 요청된 **CPU** 양에 따라 **CPU** 시간이 분배됩니다.

예를 들어, 한 컨테이너가 **500m**의 **CPU** 시간을 요청하고 다른 컨테이너가 **250m**의 **CPU** 시간을 요청한 경우 노드에서 사용 가능한 추가 **CPU** 시간이 **2:1** 비율로 컨테이너간에 분배됩니다. 컨테이너가 제한을 지정한 경우 지정된 한도를 초과하는 많은 **CPU**를 사용하지 않도록 제한됩니다. **CPU** 요청은 **Linux** 커널에서 **CFS** 공유 지원을 사용하여 적용됩니다. 기본적으로 **CPU** 제한은 **Linux** 커널에서 **CFS** 할당량 지원을 사용하여 **100ms** 측정 간격으로 적용되지만 이 기능은 비활성화할 수 있습니다.

8.5.3.1.2. 컨테이너의 메모리 요구 이해

컨테이너에 요청된 메모리 양이 보장됩니다. 컨테이너는 요청된 메모리보다 많은 메모리를 사용할 수 있지만 요청된 양을 초과하면 노드의 메모리 부족 상태에서 종료될 수 있습니다. 컨테이너가 요청된 메모리

리보다 적은 메모리를 사용하는 경우 시스템 작업 또는 데몬이 노드의 리소스 예약에 확보된 메모리 보다 더 많은 메모리를 필요로하지 않는 한 컨테이너는 종료되지 않습니다. 컨테이너가 메모리 제한을 지정할 경우 제한 양을 초과하면 즉시 종료됩니다.

8.5.3.2. 오버커밋 및 QoS (Quality of Service) 클래스 이해

요청이 없는 pod가 예약되어 있거나 해당 노드의 모든 pod에서 제한의 합계가 사용 가능한 머신 용량을 초과하면 노드가 오버 커밋됩니다.

오버 커밋된 환경에서는 노드의 pod가 특정 시점에서 사용 가능한 것보다 더 많은 컴퓨팅 리소스를 사용하려고 할 수 있습니다. 이 경우 노드는 각 pod에 우선 순위를 지정해야 합니다. 이러한 결정을 내리는데 사용되는 기능을 QoS (Quality of Service) 클래스라고 합니다.

Pod는 우선순위가 감소하는 세 가지 QoS 클래스 중 하나로 지정됩니다.

표 8.19. QoS (Quality of Service) 클래스

우선 순위	클래스 이름	설명
1(가장 높음)	Guaranteed	모든 리소스에 대해 제한 및 요청(선택 사항)이 설정되어 있고(0이 아님) 동일한 경우 Pod는 Guaranteed 로 분류됩니다.
2	Burstable	모든 리소스에 대해 요청 및 제한(선택 사항)이 설정되어 있고(0이 아님) 동일하지 않은 경우 Pod는 Burstable 로 분류됩니다.
3(가장 낮음)	BestEffort	리소스에 대한 요청 및 제한이 설정되지 않은 경우 Pod는 BestEffort 로 분류됩니다.

메모리는 압축할 수 없는 리소스이므로 메모리가 부족한 경우 우선 순위가 가장 낮은 컨테이너가 먼저 종료됩니다.

- Guaranteed** 컨테이너는 우선 순위가 가장 높은 컨테이너로 간주되며 제한을 초과하거나 시스템의 메모리가 부족하고 제거할 수 있는 우선 순위가 낮은 컨테이너가 없는 경우에만 종료됩니다.
- 시스템 메모리 부족 상태에 있는 **Burstable** 컨테이너는 제한을 초과하고 다른 **BestEffort** 컨테이너가 없으면 종료될 수 있습니다.
- BestEffort** 컨테이너는 우선 순위가 가장 낮은 컨테이너로 처리됩니다. 시스템에 메모리가 부족한 경우 이러한 컨테이너의 프로세스가 먼저 종료됩니다.

8.5.3.2.1. Quality of Service (QoS) 계층에서 메모리 예약 방법

qos-reserved 매개변수를 사용하여 특정 QoS 수준에서 pod에 예약된 메모리의 백분율을 지정할 수 있습니다. 이 기능은 요청된 리소스를 예약하여 하위 OoS 클래스의 pod가 고급 QoS 클래스의 pod에서 요청한 리소스를 사용하지 못하도록 합니다.

OpenShift Container Platform은 다음과 같이 **qos-reserved** 매개변수를 사용합니다.

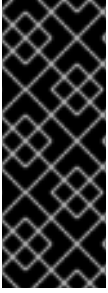
- qos-reserved=memory=100%** 값은 **Burstable** 및 **BestEffort QoS** 클래스가 더 높은 QoS 클래스에서 요청한 메모리를 소비하지 못하도록 합니다. 이를 통해 **BestEffort** 및 **Burstable** 워크로드에서 OOM이 발생할 위험이 증가되어 **Guaranteed** 및 **Burstable** 워크로드에 대한 메모리 리소스의 보장 수준을 높이는 것이 우선됩니다.
- qos-reserved=memory=50%** 값은 **Burstable** 및 **BestEffort QoS** 클래스가 더 높은 QoS 클래스에서 요청한 메모리의 절반을 소비하는 것을 허용합니다.
- qos-reserved=memory=0%** 값은 **Burstable** 및 **BestEffort QoS** 클래스가 사용 가능한 경우 할당 가능한 최대 노드 양까지 소비하는 것을 허용하지만 **Guaranteed** 워크로드가 요청된 메모리에 액세스하지 못할 위험이 높아집니다. 이로 인해 이 기능은 비활성화되어 있습니다.

8.5.3.3. 스왑 메모리 및 QOS 이해

QoS (Quality of Service) 보장을 유지하기 위해 노드에서 기본적으로 스왑을 비활성화할 수 있습니다. 그렇지 않으면 노드의 물리적 리소스를 초과 구독하여 Pod 배포 중에 Kubernetes 스케줄러가 만드는 리소스에 영향을 미칠 수 있습니다.

예를 들어 2 개의 **Guaranteed pod**가 메모리 제한에 도달하면 각 컨테이너가 스왑 메모리를 사용할 수 있습니다. 결국 스왑 공간이 충분하지 않으면 시스템의 초과 구독으로 인해 Pod의 프로세스가 종료될 수 있습니다.

스왑을 비활성화하지 못하면 노드에서 **MemoryPressure**가 발생하고 있음을 인식하지 못하여 Pod가 스케줄링 요청에서 만든 메모리를 받지 못하게 됩니다. 결과적으로 메모리 Pod를 추가로 늘리기 위해 추가 Pod가 노드에 배치되어 궁극적으로 시스템 메모리 부족 (OOM) 이벤트가 발생할 위험이 높아집니다.



중요

스왑이 활성화되면 사용 가능한 메모리에 대한 리소스 부족 처리 제거 임계 값이 예상대로 작동하지 않을 수 있습니다. 리소스 부족 처리를 활용하여 메모리 부족 상태에서 Pod를 노드에서 제거하고 메모리 부족 상태가 아닌 다른 노드에서 일정을 재조정할 수 있도록 합니다.

8.5.3.4. 노드 과다 할당 이해

오버 커밋된 환경에서는 최상의 시스템 동작을 제공하도록 노드를 올바르게 구성하는 것이 중요합니다.

노드가 시작되면 메모리 관리를 위한 커널 조정 가능한 플래그가 올바르게 설정됩니다. 커널은 실제 메모리가 소진되지 않는 한 메모리 할당에 실패해서는 안 됩니다.

이 동작을 확인하기 위해 OpenShift Container Platform은 `vm.overcommit_memory` 매개변수를 1로 설정하여 기본 운영 체제 설정을 재정의하여 커널이 항상 메모리를 오버 커밋하도록 구성합니다.

OpenShift Container Platform은 `vm.panic_on_oom` 매개변수를 0으로 설정하여 메모리 부족시 커널이 패닉 상태가 되지 않도록 구성합니다. 0으로 설정하면 커널에서 OOM (메모리 부족) 상태일 때 `oom_killer`를 호출하여 우선 순위에 따라 프로세스를 종료합니다.

노드에서 다음 명령을 실행하여 현재 설정을 볼 수 있습니다.

```
$ sysctl -a |grep commit
```

출력 예

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

출력 예

```
#...
vm.panic_on_oom = 0
#...
```



참고

위의 플래그는 이미 노드에 설정되어 있어야 하며 추가 조치가 필요하지 않습니다.

각 노드에 대해 다음 구성을 수행할 수도 있습니다.

- **CPU CFS** 할당량을 사용하여 **CPU** 제한 비활성화 또는 실행
- 시스템 프로세스의 리소스 예약
- **Quality of Service (QoS)** 계층에서의 메모리 예약

8.5.3.5. CPU CFS 할당량을 사용하여 CPU 제한 비활성화 또는 실행

기본적으로 노드는 **Linux** 커널에서 **CFS (Completely Fair Scheduler)** 할당량 지원을 사용하여 지정된 **CPU** 제한을 실행합니다.

CPU 제한 적용을 비활성화한 경우 노드에 미치는 영향을 이해해야 합니다.

- 컨테이너에 **CPU** 요청이 있는 경우 요청은 **Linux** 커널의 **CFS** 공유를 통해 계속 강제 적용됩니다.
- 컨테이너에 **CPU** 요청은 없지만 **CPU** 제한이 있는 경우 **CPU** 요청 기본값이 지정된 **CPU** 제한으로 설정되며 **Linux** 커널의 **CFS** 공유를 통해 강제 적용됩니다.
-

컨테이너에 CPU 요청 및 제한이 모두 있는 경우 Linux 커널의 CFS 공유를 통해 CPU 요청이 강제 적용되며 CPU 제한은 노드에 영향을 미치지 않습니다.

사전 요구 사항

- 다음 명령을 입력하여 구성할 노드 유형의 정적 MachineConfigPool CRD와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```



레이블이 Labels 아래에 표시됩니다.

작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

절차

1.

구성 변경을 위한 사용자 정의 리소스 (CR)를 만듭니다.

CPU 제한 비활성화를 위한 설정 예

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    cpuCfsQuota: false ❸

```

❶

CR에 이름을 지정합니다.

❷

머신 구성 풀에서 라벨을 지정합니다.

❸

cpuCfsQuota 매개변수를 false로 설정합니다.

2.

다음 명령을 실행하여 CR을 생성합니다.

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

8.5.3.6. 시스템 프로세스의 리소스 예약

보다 안정적인 스케줄링을 제공하고 노드 리소스 오버 커밋을 최소화하기 위해 각 노드는 클러스터가 작동할 수 있도록 노드에서 실행하는 데 필요한 시스템 데몬에서 사용할 리소스의 일부를 예약할 수 있습니다. 특히 메모리와 같은 압축 불가능한 리소스의 경우 리소스를 예약하는 것이 좋습니다.

프로세스

pod가 아닌 프로세스의 리소스를 명시적으로 예약하려면 스케줄링에서 사용 가능한 리소스를 지정하여 노드 리소스를 할당합니다. 자세한 내용은 노드의 리소스 할당을 참조하십시오.

8.5.3.7. 노드의 오버 커밋 비활성화

이를 활성화하면 각 노드에서 오버 커밋을 비활성화할 수 있습니다.

프로세스

노드에서 오버 커밋을 비활성화하려면 해당 노드에서 다음 명령을 실행합니다.

```
$ sysctl -w vm.overcommit_memory=0
```

8.5.4. 프로젝트 수준 제한

오버 커밋을 제어하기 위해 오버 커밋을 초과할 수 없는 프로젝트의 메모리 및 CPU 제한과 기본값을 지정하여 프로젝트 별 리소스 제한 범위를 설정할 수 있습니다.

프로젝트 수준 리소스 제한에 대한 자세한 내용은 추가 리소스를 참조하십시오.

또는 특정 프로젝트의 오버 커밋을 비활성화할 수 있습니다.

8.5.4.1. 프로젝트의 오버 커밋 비활성화

이를 활성화하면 프로젝트 별 오버 커밋을 비활성화할 수 있습니다. 예를 들어, 오버 커밋과 독립적으로 인프라 구성 요소를 구성할 수 있습니다.

프로세스

프로젝트에서 오버 커밋을 비활성화하려면 다음을 실행합니다.

1. 네임스페이스 오브젝트를 편집하여 다음 주석을 추가합니다.

```
apiVersion: v1
kind: Namespace
metadata:
```

annotations:

quota.openshift.io/cluster-resource-override-enabled: "false" **1**

...

1

이 주석을 **false** 로 설정하면 이 네임스페이스에 대한 오버 커밋이 비활성화됩니다.

8.5.5. 추가 리소스

- [배포 리소스 설정.](#)
- [노드에 리소스 할당.](#)

8.6. LINUX 제어 그룹 버전 2 (CGROUP V2) 활성화

`node.config` 오브젝트를 편집하여 클러스터에서 [Linux 제어 그룹 버전 2 \(cgroup v2\)](#)를 활성화할 수 있습니다. OpenShift Container Platform에서 `cgroup v2`를 활성화하면 클러스터의 모든 `cgroup` 버전 1 컨트롤러 및 계층 구조를 비활성화합니다. `cgroup v1`은 기본적으로 활성화되어 있습니다.

`cgroup v2`는 Linux `cgroup API`의 현재 버전입니다. `cgroup v2`는 통합 계층 구조, 더 안전한 하위 트리 위임, [pressure stall information](#) 과 같은 새로운 기능, 향상된 리소스 관리 및 격리를 포함하여 `cgroup v1`에 비해 몇 가지 개선 사항을 제공합니다. 그러나 `cgroup v2`에는 `cgroup v1`과 다른 CPU, 메모리, I/O 관리 특성이 있습니다. 따라서 일부 워크로드는 `cgroup v2`를 실행하는 클러스터의 메모리 또는 CPU 사용량에 약간의 차이가 있을 수 있습니다.



참고

- **cgroup** 파일 시스템에 의존하는 타사 모니터링 및 보안 에이전트를 실행하는 경우 에이전트를 **cgroup v2**를 지원하는 버전으로 업데이트합니다.
- **cgroup v2**를 구성하고 **Pod** 및 컨테이너 모니터링을 위한 독립 실행형 데몬 세트인 **cAdvisor**를 실행하는 경우 **cAdvisor**를 **v0.43.0** 이상으로 업데이트합니다.
- **Java** 애플리케이션을 배포하는 경우 다음 패키지과 같이 **cgroup v2**를 완전히 지원하는 버전을 사용합니다.
 - **OpenJDK / HotSpot: jdk8u372, 11.0.16, 15 이상**
 - **IBM Semeru Runtimes: jdk8u345-b01, 11.0.16.0, 17.0.4.0, 18.0.2.0 이상**
 - **IBM SDK Java Technology Edition 버전(IBM Java): 8.0.7.15 이상**



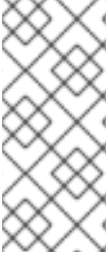
중요

OpenShift Container Platform cgroups 버전 2는 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.

8.6.1. Linux cgroup v2 구성

`node.config` 오브젝트를 편집하여 **cgroup v2**를 활성화합니다.



참고

현재 **cgroup v2**에서는 CPU 부하 분산을 비활성화하지 않습니다. 따라서 **cgroup v2**가 활성화된 경우 성능 프로필에서 원하는 동작을 얻지 못할 수 있습니다. 성능 프로필을 사용하는 경우에는 **cgroup v2**를 활성화하는 것은 권장되지 않습니다.

사전 요구 사항

- 버전 4.12 이상을 사용하는 **OpenShift Container Platform** 클러스터가 실행 중입니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- 기능 게이트를 사용하여 설정된 **TechPreviewNoUpgrade** 기능을 활성화했습니다.

절차

1. 노드에서 **cgroup v2**를 활성화합니다.
 - a. **node.config** 오브젝트를 편집합니다.

```
$ oc edit nodes.config/cluster
```

- b. **spec.cgroupMode: "v2"**: 추가

node.config 오브젝트의 예

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
    - apiVersion: config.openshift.io/v1
```

```

kind: ClusterVersion
name: version
uid: 36282574-bf9f-409e-a6cd-3032939293eb
resourceVersion: "1865"
uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  cgroupMode: "v2" 1
...

```

1

cgroup v2를 활성화합니다.

검증

1. 머신 구성에서 새 머신 구성이 추가되었는지 확인합니다.

```
$ oc get mc
```

출력 예

NAME	IGNITIONVERSION	AGE	GENERATEDBY	CONTROLLER
00-master	3.2.0	33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
00-worker	3.2.0	33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	
97-master-generated-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	3m	1
99-worker-generated-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	3m	
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	
99-master-ssh	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	40m	
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m	

99-worker-ssh	3.2.0	40m
rendered-master-23e785de7587df95a4b517e0647e5ab7 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m
worker-enable-cgroups-v2	3.2.0	10s

1

예상대로 새 머신 구성이 생성됩니다.

2.

새 `kernelArguments` 가 새 머신 구성에 추가되었는지 확인합니다.

```
$ oc describe mc <name>
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-worker-kernelarg-selinuxpermissive
spec:
  kernelArguments:
    - systemd_unified_cgroup_hierarchy=1 1
    - cgroup_no_v1="all" 2
    - psi=1 3
```

1

`systemd`에서 `cgroup v2`를 활성화합니다.

2

`cgroups v1`을 비활성화합니다.

3

3. 노트에서 노트에서 예약이 비활성화되었는지 확인합니다. 변경 사항이 적용 중임을 나타냅니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-fm1qnwt-72292-99kt6-master-0	Ready		58m	v1.25.0
ci-ln-fm1qnwt-72292-99kt6-master-1	Ready		58m	v1.25.0
ci-ln-fm1qnwt-72292-99kt6-master-2	Ready		58m	v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-a-h5gt4	Ready,SchedulingDisabled	worker	48m	v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-b-7vtmd	Ready	worker	48m	v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-c-rhzkv	Ready	worker	48m	v1.25.0

4. 노트가 **Ready** 상태가 되면 해당 노트의 디버그 세션을 시작합니다.

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

5. 디버그 셸 내에서 /host를 root 디렉터리로 설정합니다.

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

6. **sys/fs/cgroup/cgroup2fs** 파일이 노트에 있는지 확인합니다. 이 파일은 **cgroup v2**에 의해 생성됩니다.

```
$ stat -c %T -f /sys/fs/cgroup
```

출력 예

```
cgroup2fs
```

추가 리소스

- [FeatureGates를 사용하여 OpenShift Container Platform 기능 활성화](#)
- [OpenShift Container Platform 설치 프로그램 개요](#)

8.7. 기능 게이트를 사용한 기능 활성화

관리자는 **Feature Gate** 를 사용하여 기본 기능 세트의 일부가 아닌 기능을 활성화할 수 있습니다.

8.7.1. FeatureGate 이해

FeatureGate 사용자 정의 리소스 (CR)를 사용하여 클러스터에서 특정 기능 세트를 활성화할 수 있습니다. 기능 세트는 기본적으로 활성화되어 있지 않은 **OpenShift Container Platform** 기능 컬렉션입니다.

FeatureGate CR을 사용하여 다음 기능을 활성화할 수 있습니다.

- **TechPreviewNoUpgrade.** 이 기능 세트는 현재 기술 프리뷰 기능의 서브 세트입니다. 이 기능 세트를 사용하면 프로덕션 클러스터에서 비활성화된 기능을 유지하면서 테스트 클러스터에서 이러한 기술 프리뷰 기능을 완전히 테스트할 수 있습니다.



주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

기능 세트를 통해 다음과 같은 기술 프리뷰 기능을 활성화할 수 있습니다.

-

CSI 자동 마이그레이션. 지원되는 **in-tree** 볼륨 플러그인을 동등한 **CSI(Container Storage Interface)** 드라이버로 자동 마이그레이션을 활성화합니다. 지원 대상:

- **Azure File (CSIMigrationAzureFile)**
- **VMware vSphere (CSIMigrationvSphere)**
- **OpenShift 빌드의 공유 리소스 CSI 드라이버 및 CSI 볼륨 빌드. CSI(Container Storage Interface)를 활성화합니다. (CSIDriverSharedResource)**
- **CSI 볼륨. OpenShift Container Platform 빌드 시스템에 대한 CSI 볼륨 지원을 활성화합니다. (빌드CSIVolumes)**
- **노드의 스왑 메모리입니다. 노드별로 OpenShift Container Platform 워크로드에 대한 스왑 메모리 사용을 활성화합니다. (NodeSwap)**
- **cgroups v2. 다음 버전의 Linux cgroup API를 cgroup v2를 활성화합니다. (CGroupsV2)**
- **Crun. crun 컨테이너 런타임을 활성화합니다. (crun)**
- **Insights Operator. OpenShift Container Platform 구성 데이터를 수집하여 Red Hat 으로 전송하는 Insights Operator를 활성화합니다. (InsightsConfigAPI)**
- **외부 클라우드 공급자. vSphere, AWS, Azure 및 GCP에서 클러스터에 대한 외부 클라우드 공급자를 지원합니다. OpenStack에 대한 지원은 GA입니다. (ExternalCloudProvider)**
- **Pod 토폴로지 분배 제약 조건입니다. Pod 토폴로지 제약 조건에 대한 matchLabelKeys 매개변수를 활성화합니다. 매개변수는 분배를 계산할 Pod를 선택하는 Pod 라벨 키 목록입니다. (MatchLabelKeysInPodTopologySpread)**
- **Pod 보안 승인 적용 Pod 보안 승인에 대한 제한 적용을 활성화합니다. 경고만 로깅하는 대신 Pod가 Pod 보안 표준을 위반하는 경우 거부됩니다. (OpenShiftPodSecurityAdmission)**



참고

Pod 보안 승인 제한 적용은 OpenShift Container Platform 클러스터가 설치된 후 TechPreviewNoUpgrade 기능을 활성화하는 경우에만 활성화됩니다. 클러스터 설치 중에 TechPreviewNoUpgrade 기능 세트를 활성화하면 활성화되지 않습니다.

TechPreviewNoUpgrade 기능 게이트에서 활성화한 기능에 대한 자세한 내용은 다음 항목을 참조하십시오.

- [CSI 인라인 임시 블록](#)
- [CSI 자동 마이그레이션](#)
- [CSI\(Container Storage Interface\) 사용](#)
- [S2I\(Source-to-Image\) 빌드 블록 및 Docker 빌드 블록](#)
- [노드의 메모리 스왑](#)
- [클러스터 API를 사용하여 머신 관리](#)
- [Linux 제어 그룹 버전 2 활성화 \(cgroup v2\)](#)
- [컨테이너 엔진 및 컨테이너 런타임 정보](#)
- [Insights Operator 사용](#)
- [Pod 토폴로지 분배 제약 조건을 사용하여 Pod 배치 제어](#)

- **Kubernetes 문서** 의 **Pod 보안 승인** 및 **Pod 보안 승인 이해 및 관리**

8.7.2. 설치 시 기능 세트 활성화


클러스터를 배포하기 전에 **install-config.yaml** 파일을 편집하여 클러스터의 모든 노드에 대해 기능 세트를 활성화할 수 있습니다.

사전 요구 사항

- **install-config.yaml** 파일이 있습니다.

절차

1. **featureSet** 매개변수를 사용하여 활성화할 기능 세트의 이름 (예: **TechPreviewNoUpgrade**)을 지정합니다.



주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

활성화된 기능 세트가 있는 샘플 **install-config.yaml** 파일

```

compute:
- hyperthreading: Enabled
  name: worker
platform:
  aws:
    rootVolume:
      iops: 2000
      size: 500
      type: io1
    metadataService:
      authentication: Optional
    type: c5.4xlarge
    zones:

```



```
- us-west-2c
replicas: 3
featureSet: TechPreviewNoUpgrade
```

2. 설치 프로그램을 사용하여 클러스터를 배포할 때 파일을 저장하고 참조합니다.

검증

노드가 **ready** 상태로 돌아간 후 노드의 **kubelet.conf** 파일을 보고 기능 게이트가 활성화되었는지 확인할 수 있습니다.

1. 웹 콘솔의 관리자 화면에서 **Compute** → **Nodes** 로 이동합니다.
2. 노드를 선택합니다.
3. 노드 세부 정보 페이지에서 터미널 을 클릭합니다.
4. 터미널 창에서 **root** 디렉토리를 **/host:**로 변경합니다.

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

5. **kubelet.conf** 파일을 확인합니다.

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

샘플 출력

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...
```

true 로 나열된 기능은 클러스터에서 활성화되어 있습니다.



참고

나열된 기능은 **OpenShift Container Platform** 버전에 따라 다릅니다.

8.7.3. 웹 콘솔을 사용하여 기능 세트 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 **FeatureGate CR**(사용자 정의 리소스)을 편집하여 클러스터의 모든 노드에 대해 기능 세트를 활성화할 수 있습니다.

절차

기능 세트를 활성화하려면 다음을 수행합니다.

1. **OpenShift Container Platform** 웹 콘솔에서 **관리** → **사용자 지정 리소스 정의** 페이지로 전환합니다.
2. **사용자 지정 리소스 정의** 페이지에서 **FeatureGate**를 클릭합니다.
3. **사용자 정의 리소스 정의 세부 정보** 페이지에서 **인스턴스** 탭을 클릭합니다.
4. **클러스터 기능 게이트**를 클릭한 다음 **YAML** 탭을 클릭합니다.
5. 특정 기능 세트를 추가하려면 **클러스터 인스턴스**를 편집합니다.



주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

FeatureGate 사용자 지정 리소스 샘플

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster 1
# ...
spec:
  featureSet: TechPreviewNoUpgrade 2
```

1

FeatureGate CR의 이름은 **cluster**이어야 합니다.

2

활성화할 기능 세트를 추가합니다.

- **TechPreviewNoUpgrade**를 사용하면 특정 기술 프리뷰 기능을 사용할 수 있습니다.

변경 사항을 저장하면 새 머신 구성이 생성되면 머신 구성 풀이 업데이트되고 변경 사항이 적용되는 동안 각 노드에 대한 스케줄링이 비활성화됩니다.

검증

노드가 **ready** 상태로 돌아간 후 노드의 **kubelet.conf** 파일을 보고 기능 게이트가 활성화되었는지 확인할 수 있습니다.

1. 웹 콘솔의 관리자 화면에서 **Compute** → **Nodes** 로 이동합니다.
2. 노드를 선택합니다.
3. 노드 세부 정보 페이지에서 터미널 을 클릭합니다.
4. 터미널 창에서 **root** 디렉토리를 **/host:**로 변경합니다.

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

5. **kubelet.conf** 파일을 확인합니다.

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

샘플 출력

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...
```

true 로 나열된 기능은 클러스터에서 활성화되어 있습니다.



참고

나열된 기능은 **OpenShift Container Platform** 버전에 따라 다릅니다.

8.7.4. CLI를 사용하여 기능 세트 활성화

OpenShift CLI(oc)를 사용하여 **FeatureGate CR**(사용자 정의 리소스)을 편집하여 클러스터의 모든 노드에 대해 기능 세트를 활성화할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)가 설치되어 있습니다.**

절차

기능 세트를 활성화하려면 다음을 수행합니다.

1. **cluster**라는 **FeatureGate CR**을 편집합니다.

```
$ oc edit featuregate cluster
```



주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

FeatureGate 사용자 지정 리소스 샘플

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster ①
# ...
spec:
  featureSet: TechPreviewNoUpgrade ②
```

①

FeatureGate CR의 이름은 **cluster**이어야 합니다.

2

활성화할 기능 세트를 추가합니다.

- **TechPreviewNoUpgrade**를 사용하면 특정 기술 프리뷰 기능을 사용할 수 있습니다.

변경 사항을 저장하면 새 머신 구성이 생성되면 머신 구성 풀이 업데이트되고 변경 사항이 적용되는 동안 각 노드에 대한 스케줄링이 비활성화됩니다.

검증

노드가 **ready** 상태로 돌아간 후 노드의 **kubelet.conf** 파일을 보고 기능 게이트가 활성화되었는지 확인할 수 있습니다.

1. 웹 콘솔의 관리자 화면에서 **Compute** → **Nodes** 로 이동합니다.
2. 노드를 선택합니다.
3. 노드 세부 정보 페이지에서 터미널 을 클릭합니다.
4. 터미널 창에서 **root** 디렉토리를 **/host:**로 변경합니다.

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

5. **kubelet.conf** 파일을 확인합니다.

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

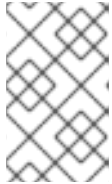
샘플 출력

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
```

```
LegacyNodeRoleBehavior: false
```

```
# ...
```

`true` 로 나열된 기능은 클러스터에서 활성화되어 있습니다.



참고

나열된 기능은 **OpenShift Container Platform** 버전에 따라 다릅니다.

8.8. 작업자 대기 시간 프로필을 사용하여 대기 시간이 높은 환경에서 클러스터 안정성 개선

클러스터 관리자가 플랫폼 확인을 위해 대기 시간 테스트를 수행한 경우 대기 시간이 긴 경우 안정성을 보장하기 위해 클러스터의 작동을 조정해야 할 수 있습니다. 클러스터 관리자는 파일에 기록된 하나의 매개 변수만 변경해야 합니다. 이 매개 변수는 감독자 프로세스가 상태를 읽고 클러스터의 상태를 해석하는 방법에 영향을 미치는 매개 변수 4개를 제어합니다. 하나의 매개 변수만 변경하면 지원 가능한 방식으로 클러스터 튜닝이 제공됩니다.

Kubelet 프로세스는 클러스터 상태를 모니터링하기 위한 시작점을 제공합니다. **Kubelet** 은 **OpenShift Container Platform** 클러스터의 모든 노드에 대한 상태 값을 설정합니다. **Kubernetes Controller Manager(kube 컨트롤러)**는 기본적으로 10초마다 상태 값을 읽습니다. **kube** 컨트롤러에서 노드 상태 값을 읽을 수 없는 경우 구성된 기간이 지난 후 해당 노드와의 연결이 끊어집니다. 기본 동작은 다음과 같습니다.

1. 컨트롤 플레인의 노드 컨트롤러는 노드 상태를 **Unhealthy** 로 업데이트하고 노드 **Ready** 조건 **'Unknown'**을 표시합니다.
2. 스케줄러는 이에 대한 응답으로 해당 노드에 대한 **Pod** 예약을 중지합니다.
3. **Node Lifecycle Controller**는 **NoExecute** 효과가 있는 **node.kubernetes.io/unreachable** 테인트를 노드에 추가하고 기본적으로 5분 후에 제거하도록 노드에 **Pod**를 예약합니다.

이 동작은 특히 네트워크 엣지에 노드가 있는 경우 네트워크가 대기 시간 문제가 발생하는 경우 문제가 발생할 수 있습니다. 경우에 따라 네트워크 대기 시간으로 인해 **Kubernetes** 컨트롤러 관리자에서 정상적인 노드에서 업데이트를 수신하지 못할 수 있습니다. **Kubelet** 은 노드가 정상이지만 노드에서 **Pod**를 제거합니다.

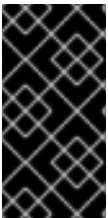
이 문제를 방지하려면 작업자 대기 시간 프로필을 사용하여 **Kubelet** 및 **Kubernetes** 컨트롤러 관리자가 작업을 수행하기 전에 상태 업데이트를 기다리는 빈도를 조정할 수 있습니다. 이러한 조정은 컨트롤 플레인과 작업자 노드 간의 네트워크 대기 시간이 최적이지 아닌 경우 클러스터가 올바르게 실행되도록 하는데 도움이 됩니다.

이러한 작업자 대기 시간 프로필에는 대기 시간을 높이기 위해 클러스터의 응답을 제어하기 위해 신중하게 조정된 값으로 미리 정의된 세 가지 매개변수 세트가 포함되어 있습니다. 실험적으로 최상의 값을 수동으로 찾을 필요가 없습니다.

클러스터를 설치할 때 또는 언제든지 클러스터 네트워크에서 대기 시간을 늘릴 때 작업자 대기 시간 프로필을 구성할 수 있습니다.

8.8.1. 작업자 대기 시간 프로필 이해

작업자 대기 시간 프로필은 신중하게 조정된 매개변수의 네 가지 범주입니다. 이러한 값을 구현하는 4개의 매개변수는 **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** 및 **default-unreachable-toleration-seconds**입니다. 이러한 매개변수는 수동 방법을 사용하여 최상의 값을 결정할 필요 없이 대기 시간 문제에 대한 클러스터의 대응을 제어할 수 있는 값을 사용할 수 있습니다.



중요

이러한 매개변수를 수동으로 설정하는 것은 지원되지 않습니다. 잘못된 매개변수 설정은 클러스터 안정성에 부정적인 영향을 미칩니다.

모든 작업자 대기 시간 프로필은 다음 매개변수를 구성합니다.

node-status-update-frequency

kubelet이 API 서버에 노드 상태를 게시하는 빈도를 지정합니다.

node-monitor-grace-period

노드를 비정상적으로 표시하고 **node.kubernetes.io/not-ready** 또는 **node.kubernetes.io/unreachable** 테인트를 노드에 추가하기 전에 **Kubernetes** 컨트롤러 관리자가 **kubelet**에서 업데이트를 기다리는 시간(초)을 지정합니다.

default-not-ready-toleration-seconds

해당 노드에서 **Pod**를 제거하기 전에 **Kube API Server Operator**가 기다리는 비정상적인 노드를 표시한 후 시간(초)을 지정합니다.

default-unreachable-toleration-seconds

해당 노드에서 Pod를 제거하기 전에 Kube API Server Operator가 대기할 수 없는 노드를 표시한 후 시간(초)을 지정합니다.

다음 Operator는 작업자 대기 시간 프로필에 대한 변경 사항을 모니터링하고 그에 따라 응답합니다.

- **MCO(Machine Config Operator)**는 작업자 노드에서 **node-status-update-frequency** 매개 변수를 업데이트합니다.
- **Kubernetes** 컨트롤러 관리자는 컨트롤 플레인 노드에서 **node-monitor-grace-period** 매개 변수를 업데이트합니다.
- **Kubernetes API Server Operator**는 컨트롤 플레인 노드에서 **default-not-ready-toleration-seconds** 및 **default-unreachable-toleration-seconds** 매개 변수를 업데이트합니다.

기본 구성은 대부분의 경우 작동하지만 OpenShift Container Platform에서는 네트워크에서 평소보다 대기 시간이 더 높은 상황에 대해 두 개의 다른 작업자 대기 시간 프로필을 제공합니다. 세 개의 작업자 대기 시간 프로필은 다음 섹션에 설명되어 있습니다.

기본 작업자 대기 시간 프로필

Default 프로필을 사용하면 각 Kubelet 이 10초마다 상태를 업데이트합니다(**node-status-update-frequency**). Kube Controller Manager 는 5초마다 Kubelet 의 상태를 확인합니다(**node-monitor-grace-period**).

Kubernetes 컨트롤러 관리자는 Kubelet 비정상을 고려하기 전에 Kubelet 에서 상태 업데이트를 40초 동안 기다립니다. **Kubernetes** 컨트롤러 관리자에서 사용할 수 없는 상태가 없는 경우 노드를 **node.kubernetes.io/not-ready** 또는 **node.kubernetes.io/unreachable** 테인트로 표시하고 해당 노드에서 Pod를 제거합니다.

해당 노드의 Pod에 NoExecute 테인트가 있는 경우 **tolerationSeconds** 에 따라 Pod가 실행됩니다. Pod에 테인트가 없는 경우 300초(Kube API서버의**default-not-ready-toleration-seconds** 및 **default-unreachable-toleration-seconds** 설정)가 제거됩니다.

Profile

구성 요소

매개 변수

값

Profile	구성 요소	매개 변수	값
기본값	kubelet	node-status-update-frequency	10s
	kubelet 컨트롤러 관리자	node-monitor-grace-period	40s
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	300s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	300s

중간 작업자 대기 시간 프로파일

네트워크 대기 시간이 평소보다 약간 높은 경우 **MediumUpdateAverageReaction** 프로필을 사용하십시오.

MediumUpdateAverageReaction 프로파일은 kubelet 업데이트 빈도를 20초로 줄이고 Kubernetes 컨트롤러 관리자가 해당 업데이트를 2분으로 기다리는 기간을 변경합니다. 해당 노드의 Pod 제거 기간이 60초로 단축됩니다. Pod에 **tolerationSeconds** 매개변수가 있는 경우 제거는 해당 매개변수로 지정된 기간 동안 기다립니다.

Kubernetes 컨트롤러 관리자는 노드의 비정상적인 것으로 간주하기 위해 2분 정도 기다립니다. 또 다른 순간에 제거 프로세스가 시작됩니다.

Profile	구성 요소	매개 변수	값
MediumUpdateAverageReaction	kubelet	node-status-update-frequency	20s
	kubelet 컨트롤러 관리자	node-monitor-grace-period	2m
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	60s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	60s

낮은 작업자 대기 시간 프로파일

네트워크 대기 시간이 매우 높은 경우 **LowUpdateSlowReaction** 프로필을 사용합니다.

LowUpdateSlowReaction 프로필은 **kubelet** 업데이트 빈도를 1분으로 줄이고 **Kubernetes** 컨트롤러 관리자가 해당 업데이트를 5분으로 기다리는 기간을 변경합니다. 해당 노드의 **Pod** 제거 기간이 60초로 단축됩니다. **Pod**에 **tolerationSeconds** 매개변수가 있는 경우 제거는 해당 매개변수로 지정된 기간 동안 기다립니다.

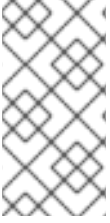
Kubernetes 컨트롤러 관리자는 노드의 비정상적인 것으로 간주하기 위해 5분 정도 기다립니다. 또 다른 순간에 제거 프로세스가 시작됩니다.

Profile	구성 요소	매개변수	값
LowUpdateSlowReaction	kubelet	node-status-update-frequency	1m
	kubelet 컨트롤러 관리자	node-monitor-grace-period	5m
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	60s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	60s

8.8.2. 작업자 대기 시간 프로파일 사용 및 변경

네트워크 대기 시간을 처리하기 위해 작업자 대기 시간 프로필을 변경하려면 **node.config** 오브젝트를 편집하여 프로파일 이름을 추가합니다. 대기 시간이 증가하거나 감소하면 언제든지 프로필을 변경할 수 있습니다.

한 번에 하나의 작업자 대기 시간 프로필을 이동해야 합니다. 예를 들어 기본 프로파일에서 **LowUpdateSlowReaction** 작업자 대기 시간 프로필로 직접 이동할 수 없습니다. 기본 작업자 대기 시간 프로파일에서 먼저 **MediumUpdateAverageReaction** 프로필로 이동한 다음 **LowUpdateSlowReaction** 으로 이동해야 합니다. 마찬가지로 **Default** 프로필로 돌아갈 때 먼저 **low** 프로파일에서 **medium** 프로파일로 이동한 다음 **Default** 로 이동해야 합니다.



참고

OpenShift Container Platform 클러스터를 설치할 때 작업자 대기 시간 프로필을 구성할 수도 있습니다.

절차

기본 작업자 대기 시간 프로필에서 이동하려면 다음을 수행합니다.

1. 중간 작업자 대기 시간 프로파일로 이동합니다.

- a. `node.config` 오브젝트를 편집합니다.

```
$ oc edit nodes.config/cluster
```

- b. `spec.workerLatencyProfile` 추가: `MediumUpdateAverageReaction`:

`node.config` 오브젝트의 예

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction 1
# ...
```

1

중간 작업자 대기 시간 정책을 지정합니다.

변경 사항이 적용되므로 각 작업자 노드에서 예약이 비활성화됩니다.

2.

선택 사항: 작업자 대기 시간이 짧은 프로필로 이동합니다.

a.

`node.config` 오브젝트를 편집합니다.

```
$ oc edit nodes.config/cluster
```

b.

`spec.workerLatencyProfile` 값을 `LowUpdateSlowReaction` 으로 변경합니다.

`node.config` 오브젝트의 예

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: LowUpdateSlowReaction 1
# ...
```

1

낮은 작업자 대기 시간 정책 사용을 지정합니다.

변경 사항이 적용되므로 각 작업자 노드에서 예약이 비활성화됩니다.

검증

•

모든 노드가 **Ready** 상태로 돌아 오면 다음 명령을 사용하여 **Kubernetes** 컨트롤러 관리자를 확인하여 적용되었는지 확인할 수 있습니다.

```
$ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
```

출력 예

```
# ...
- lastTransitionTime: "2022-07-11T19:47:10Z"
  reason: ProfileUpdated
  status: "False"
  type: WorkerLatencyProfileProgressing
- lastTransitionTime: "2022-07-11T19:47:10Z" 1
  message: all static pod revision(s) have updated latency profile
  reason: ProfileUpdated
  status: "True"
  type: WorkerLatencyProfileComplete
- lastTransitionTime: "2022-07-11T19:20:11Z"
  reason: AsExpected
  status: "False"
  type: WorkerLatencyProfileDegraded
- lastTransitionTime: "2022-07-11T19:20:36Z"
  status: "False"
# ...
```

1

프로필이 적용되고 활성 상태가 되도록 지정합니다.

미디어 프로필을 기본값으로 변경하거나 기본값을 **medium**로 변경하려면 **node.config** 오브젝트를 편집하고 **spec.workerLatencyProfile** 매개변수를 적절한 값으로 설정합니다.

9장. 네트워크 엣지의 원격 작업자 노드

9.1. 네트워크 엣지에서 원격 작업자 노드 사용

네트워크 엣지에 있는 노드를 사용하여 **OpenShift Container Platform** 클러스터를 구성할 수 있습니다. 이 주제에서는 해당 노드를 원격 작업자 노드라고 합니다. 원격 작업자 노드가 있는 일반 클러스터에서는 온프레미스 마스터 및 작업자 노드를 클러스터에 연결된 다른 위치에 있는 작업자 노드와 결합합니다. 이 주제는 원격 작업자 노드 사용 모범 사례에 대한 지침을 제공하기 위한 것으로, 특정 구성 세부 정보는 포함하지 않습니다.

통신, 소매, 제조, 정부와 같이 다양한 업계에서 원격 작업자 노드에서 배포 패턴을 사용하는 다양한 사용 사례가 있습니다. 예를 들어 원격 작업자 노드를 **Kubernetes 영역에 결합하여 프로젝트와 워크로드를 분리하고 격리할 수 있습니다.**

그러나 원격 작업자 노드가 있으면 대기 시간이 길어지고 네트워크 연결이 간헐적으로 끊어지며 기타 문제가 발생할 수 있습니다. 원격 작업자 노드가 있는 클러스터의 문제점은 다음과 같습니다.

- **네트워크 분리:** **OpenShift Container Platform** 컨트롤 플레인과 원격 작업자 노드가 서로 통신할 수 있어야 합니다. 컨트롤 플레인과 원격 작업자 노드 간의 거리로 인해 네트워크 문제가 발생하여 이러한 통신을 방해할 수 있었습니다. **OpenShift Container Platform**이 **네트워크 분리**에 응답하는 방법과 클러스터에 미치는 영향을 완화하는 방법에 대한 자세한 내용은 원격 작업자 노드를 사용하여 네트워크 분리를 참조하십시오.
- **정전:** 컨트롤 플레인 및 원격 작업자 노드가 별도의 위치에 있기 때문에 원격 위치 또는 이 둘 사이의 어느 지점에서 정전이 발생하면 클러스터에 부정적인 영향을 미칠 수 있습니다. **OpenShift Container Platform**이 **노드 손실에 응답하는 방법과 클러스터에 미치는 영향을 줄이는 방법**에 대한 자세한 내용은 원격 작업자 노드의 정전을 참조하십시오.
- **대기 시간 급증 또는 일시적인 처리량 감소:** 모든 네트워크와 마찬가지로 클러스터와 원격 작업자 노드 간의 네트워크 조건이 변경되면 클러스터에 부정적인 영향을 미칠 수 있습니다. **OpenShift Container Platform**은 대기 시간 문제에 대한 클러스터 반응을 제어할 수 있는 여러 작업자 대기 시간 프로필을 제공합니다.

원격 작업자 노드가 있는 클러스터를 계획할 때 다음과 같은 제한 사항에 유의하십시오.

- **OpenShift Container Platform**은 온프레미스 클러스터에서 사용하는 것과 다른 클라우드 공급자를 사용하는 원격 작업자 노드를 지원하지 않습니다.
-

특정 **Kubernetes** 영역에서 다른 **Kubernetes** 영역으로 워크로드를 이동하는 경우 다른 영역에서 사용할 수 없는 특정 유형의 메모리 등 시스템 및 환경 문제로 인해 문제가 발생할 수 있습니다.

- 프록시 및 방화벽은 이 문서의 범위를 벗어나는 추가 제한 사항을 제공할 수 있습니다. 방화벽 구성과 같은 제한 사항을 해결하는 방법은 관련 [OpenShift Container Platform 설명서](#)를 참조하십시오.
- 컨트롤 플레인과 네트워크 엣지 노드 간에 **L2/L3** 수준 네트워크 연결을 구성하고 유지 관리해야 합니다.

9.1.1. 원격 작업자 노드 추가

클러스터에 원격 작업자 노드를 추가하려면 몇 가지 추가 고려 사항이 필요합니다.

- 컨트롤 플레인과 모든 원격 작업자 노드 간에 트래픽을 라우팅하려면 경로 또는 기본 게이트웨이가 제 위치에 있는지 확인해야 합니다.
- 컨트롤 플레인에 **Ingress VIP**를 배치해야 합니다.
- 사용자 프로비저닝 인프라를 사용하여 원격 작업자 노드를 추가하는 것은 다른 작업자 노드를 추가하는 것과 동일합니다.
- 설치 시 설치 시 설치 관리자 프로비저닝 클러스터에 원격 작업자 노드를 추가하려면 설치 전에 **install-config.yaml** 파일의 각 작업자 노드의 서브넷을 지정합니다. **DHCP** 서버에는 추가 설정이 필요하지 않습니다. 원격 작업자 노드는 로컬 프로비저닝 네트워크에 액세스할 수 없으므로 가상 미디어를 사용해야 합니다.
- **provisioning** 네트워크와 함께 배포된 설치 관리자 프로비저닝 클러스터에 원격 작업자 노드를 추가하려면 **install-config.yaml** 파일에서 **virtualMediaViaExternalNetwork** 플래그가 **true**로 설정되어 가상 미디어를 사용하여 노드를 추가해야 합니다. 원격 작업자 노드는 로컬 프로비저닝 네트워크에 액세스할 수 없습니다. **PXE**가 아닌 가상 미디어와 함께 배포해야 합니다. 또한 **DHCP** 서버의 각 원격 작업자 노드 및 컨트롤 플레인 노드의 각 서브넷을 지정합니다.

추가 리소스

- [서브넷 간 통신 설정](#)

- [서브넷에 대한 호스트 네트워크 인터페이스 구성](#)
- [컨트롤 플레인에서 실행되도록 네트워크 구성 요소 구성](#)

9.1.2. 원격 작업자 노드를 사용하여 네트워크 분리

모든 노드는 10초마다 OpenShift Container Platform 클러스터의 **Kubernetes Controller Manager Operator(kube 컨트롤러)**로 하트비트를 보냅니다. 클러스터에서 노드의 하트비트를 수신하지 않는 경우 OpenShift Container Platform은 여러 기본 메커니즘을 사용하여 응답합니다.

OpenShift Container Platform은 네트워크 파티션 및 기타 중단에 대해 탄력적으로 설계되었습니다. 소프트웨어 업그레이드로 인한 중단, 네트워크 분할, 라우팅 문제와 같이 비교적 일반적인 일부 중단을 완화할 수 있습니다. 완화 전략에는 원격 작업자 노드의 Pod가 올바른 양의 CPU 및 메모리 리소스를 요청하는지 확인, 적절한 복제 정책 구성, 영역 전체에서 중복성 사용, 워크로드에 Pod 중단 예산 사용이 포함됩니다.

구성된 기간이 지난 후 kube 컨트롤러와 노드의 연결이 끊어지면 컨트롤 플레인의 노드 컨트롤러에서 노드 상태를 **Unhealthy**로 업데이트하고 노드 **Ready** 조건을 **Unknown**으로 표시합니다. 스케줄러는 이에 대한 응답으로 해당 노드에 대한 Pod 예약을 중지합니다. 온프레미스 노드 컨트롤러는 **NoExecute** 효과가 있는 **node.kubernetes.io/unreachable** 테인트를 노드에 추가하고 노드의 Pod를 기본적으로 5분 후에 제거하도록 예약합니다.

Deployment 오브젝트 또는 **StatefulSet** 오브젝트와 같은 워크로드 컨트롤러에서 비정상 노드의 Pod로 트래픽을 전송하고 기타 노드에서 클러스터에 연결할 수 있는 경우 OpenShift Container Platform은 노드의 Pod 외부로 트래픽을 라우팅합니다. 클러스터에 연결할 수 없는 노드는 새 트래픽 라우팅을 통해 업데이트되지 않습니다. 결과적으로 해당 노드의 워크로드가 비정상 노드에 도달하기 위해 계속 시도할 수 있습니다.

다음과 같은 방법으로 연결 손실의 영향을 완화할 수 있습니다.

- [테몬 세트를 사용하여 테인트를 허용하는 pod 생성](#)
- [노드가 중단되는 경우 자동으로 재시작하는 정적 Pod 사용](#)
- [Kubernetes 영역을 사용하여 Pod 제거 제어](#)

- **Pod** 제거를 지연하거나 방지하도록 **Pod** 허용 오차 구성
- 노드를 비정상적으로 표시하는 시기를 제어하도록 **kubelet** 구성.

원격 작업자 노드가 있는 클러스터에서 이러한 오브젝트를 사용하는 방법에 대한 자세한 내용은 [원격 작업자 노드 전략](#) 정보를 참조하십시오.

9.1.3. 원격 작업자 노드의 정전

원격 작업자 노드가 정전되거나 강제 다시 시작되면 **OpenShift Container Platform**은 여러 기본 메커니즘을 사용하여 응답합니다.

구성된 기간이 지난 후 **Kubernetes Controller Manager Operator(kube 컨트롤러)**와 노드의 연결이 끊어지면 컨트롤 플레인에서 노드 상태를 **Unhealthy**로 업데이트하고 노드 **Ready** 조건을 **Unknown**으로 표시합니다. 스케줄러는 이에 대한 응답으로 해당 노드에 대한 **Pod** 예약을 중지합니다. 온프레미스 노드 컨트롤러는 **NoExecute** 효과가 있는 **node.kubernetes.io/unreachable** 테인트를 노드에 추가하고 노드의 **Pod**를 기본적으로 5분 후에 제거하도록 예약합니다.

노드가 전원을 복구하고 컨트롤 플레인에 다시 연결되면 노드에서 **Pod**를 재시작해야 합니다.



참고

재시작 즉시 **Pod**를 재시작하려면 정적 **Pod**를 사용하십시오.

노드를 재시작하면 **kubelet**도 재시작되고 노드에 예약된 **Pod**를 재시작하려고 합니다. 컨트롤 플레인에 대한 연결이 기본값인 5분보다 오래 걸리는 경우 컨트롤 플레인에서 노드 상태를 업데이트하고 **node.kubernetes.io/unreachable** 테인트를 제거할 수 없습니다. 노드에서 **kubelet**은 실행 중인 **Pod**를 모두 종료합니다. 이러한 조건이 해제되면 스케줄러는 해당 노드에 **Pod**를 예약할 수 있습니다.

다음을 통해 정전의 영향을 완화할 수 있습니다.

- 데몬 세트를 사용하여 테인트를 허용하는 **pod** 생성

- 노드와 함께 자동으로 재시작하는 정적 Pod 사용
- Pod 제거를 지연하거나 방지하도록 Pod 허용 오차 구성
- 노드 컨트롤러에서 노드를 비정상적으로 표시하는 시기를 제어하도록 kubelet 구성.

원격 작업자 노드가 있는 클러스터에서 이러한 오브젝트를 사용하는 방법에 대한 자세한 내용은 원격 작업자 노드 전략 정보를 참조하십시오.

9.1.4. 대기 시간 급증 또는 원격 작업자로의 처리량 감소

클러스터 관리자가 플랫폼 확인을 위해 대기 시간 테스트를 수행한 경우 대기 시간이 긴 경우 안정성을 보장하기 위해 클러스터의 작동을 조정해야 할 수 있습니다. 클러스터 관리자는 파일에 기록된 하나의 매개 변수만 변경해야 합니다. 이 매개 변수는 감독자 프로세스가 상태를 읽고 클러스터의 상태를 해석하는 방법에 영향을 미치는 매개 변수 4개를 제어합니다. 하나의 매개 변수만 변경하면 지원 가능한 방식으로 클러스터 튜닝이 제공됩니다.

Kubelet 프로세스는 클러스터 상태를 모니터링하기 위한 시작점을 제공합니다. Kubelet 은 OpenShift Container Platform 클러스터의 모든 노드에 대한 상태 값을 설정합니다. Kubernetes Controller Manager(kube 컨트롤러)는 기본적으로 10초마다 상태 값을 읽습니다. kube 컨트롤러에서 노드 상태 값을 읽을 수 없는 경우 구성된 기간이 지난 후 해당 노드와의 연결이 끊어집니다. 기본 동작은 다음과 같습니다.

1. 컨트롤 플레인의 노드 컨트롤러는 노드 상태를 Unhealthy 로 업데이트하고 노드 Ready 조건 'Unknown'을 표시합니다.
2. 스케줄러는 이에 대한 응답으로 해당 노드에 대한 Pod 예약을 중지합니다.
3. Node Lifecycle Controller는 NoExecute 효과가 있는 node.kubernetes.io/unreachable 테인트를 노드에 추가하고 기본적으로 5분 후에 제거하도록 노드에 Pod를 예약합니다.

이 동작은 특히 네트워크 엣지에 노드가 있는 경우 네트워크가 대기 시간 문제가 발생하는 경우 문제가 발생할 수 있습니다. 경우에 따라 네트워크 대기 시간으로 인해 Kubernetes 컨트롤러 관리자에서 정상적인 노드에서 업데이트를 수신하지 못할 수 있습니다. Kubelet 은 노드가 정상이지만 노드에서 Pod를 제거합니다.

이 문제를 방지하려면 작업자 대기 시간 프로필을 사용하여 **Kubelet** 및 **Kubernetes** 컨트롤러 관리자가 작업을 수행하기 전에 상태 업데이트를 기다리는 빈도를 조정할 수 있습니다. 이러한 조정은 컨트롤 플레인과 작업자 노드 간의 네트워크 대기 시간이 최적이지 아닌 경우 클러스터가 올바르게 실행되도록 하는데 도움이 됩니다.

이러한 작업자 대기 시간 프로필에는 대기 시간을 높이기 위해 클러스터의 응답을 제어하기 위해 신중하게 조정된 값으로 미리 정의된 세 가지 매개변수 세트가 포함되어 있습니다. 실험적으로 최상의 값을 수동으로 찾을 필요가 없습니다.

클러스터를 설치할 때 또는 언제든지 클러스터 네트워크에서 대기 시간을 늘릴 때 작업자 대기 시간 프로필을 구성할 수 있습니다.

추가 리소스

- [작업자 대기 시간 프로필을 사용하여 대기 시간이 많은 환경에서 클러스터 안정성 개선](#)

9.1.5. 원격 작업자 노드 전략

원격 작업자 노드를 사용하는 경우 애플리케이션을 실행하는 데 사용할 오브젝트를 고려하십시오.

네트워크 문제 또는 정전 시 원하는 동작을 기반으로 데몬 세트 또는 정적 **Pod**를 사용하는 것이 좋습니다. 또한 컨트롤 플레인에서 원격 작업자 노드에 연결할 수 없는 경우 **Kubernetes** 영역 및 허용 오차를 사용하여 **Pod** 제거를 제어하거나 방지할 수 있습니다.

데몬 세트

데몬 세트는 다음과 같은 이유로 원격 작업자 노드에서 **Pod**를 관리하는 가장 좋은 방법입니다.

- 데몬 세트에는 일반적으로 일정 조정 동작이 필요하지 않습니다. 노드와 클러스터의 연결이 끊어지면 노드의 **Pod**가 계속 실행될 수 있습니다. **OpenShift Container Platform**에서는 데몬 세트 **Pod**의 상태를 변경하지 않고 **Pod**를 마지막으로 보고된 상태로 유지합니다. 예를 들어 데몬 세트 **Pod**가 **Running** 상태에 있는 경우 노드에서 통신을 중지하면 **Pod**가 계속 실행되고 **OpenShift Container Platform**에서 **Pod**가 실행 중인 것으로 간주합니다.
- 기본적으로 데몬 세트 **Pod**는 **tolerationSeconds** 값이 없는 **node.kubernetes.io/unreachable** 및 **node.kubernetes.io/not-ready** 테인트에 대한 **NoExecute** 허용 오차로 생성됩니다. 이러한 기본값을 사용하면 컨트롤 플레인에서 노드에 연결할 수 없는 경우 데몬 세트 **Pod**가 제거되지 않습니다. 예를 들면 다음과 같습니다.

데몬 세트 Pod에 기본적으로 허용 오차 추가

tolerations:

- key: `node.kubernetes.io/not-ready`
operator: `Exists`
effect: `NoExecute`
- key: `node.kubernetes.io/unreachable`
operator: `Exists`
effect: `NoExecute`
- key: `node.kubernetes.io/disk-pressure`
operator: `Exists`
effect: `NoSchedule`
- key: `node.kubernetes.io/memory-pressure`
operator: `Exists`
effect: `NoSchedule`
- key: `node.kubernetes.io/pid-pressure`
operator: `Exists`
effect: `NoSchedule`
- key: `node.kubernetes.io/unschedulable`
operator: `Exists`
effect: `NoSchedule`

- 데몬 세트에서는 라벨을 사용하여 일치하는 작업자 노드에서 워크로드를 실행할 수 있습니다.
- **OpenShift Container Platform** 서비스 끝점을 사용하여 데몬 세트 Pod의 부하를 분산할 수 있습니다.



참고

OpenShift Container Platform에서 노드에 연결할 수 없는 경우 데몬 세트에서는 노드 재부팅 후 Pod를 예약하지 않습니다.

고정 Pod

노드가 재부팅될 때(예: 정전 후) Pod를 재시작하려면 **정적 Pod**를 사용하는 것이 좋습니다. 노드의 **Kubelet**은 노드가 재시작되면 정적 Pod를 자동으로 재시작합니다.



참고

정적 Pod에서는 보안 및 구성 맵을 사용할 수 없습니다.

Kubernetes 영역

Kubernetes 영역은 속도를 늦추거나 경우에 따라 Pod 제거를 완전히 중지할 수 있습니다.

컨트롤 플레인에서 노드에 연결할 수 없는 경우 노드 컨트롤러는 기본적으로 `node.kubernetes.io/unreachable` 테인트를 적용하고 초당 0.1 노드의 속도로 Pod를 제거합니다. 그러나 Kubernetes 영역을 사용하는 클러스터에서는 Pod 제거 동작이 변경됩니다.

영역이 완전히 중단되고 해당 영역에 있는 모든 노드의 **Ready** 조건이 **False** 또는 **Unknown**인 경우 컨트롤 플레인에서 해당 영역의 노드에 `node.kubernetes.io/unreachable` 테인트를 적용하지 않습니다.

부분적으로 중단된 영역의 경우 노드의 55% 이상에 **False** 또는 **Unknown** 조건이 있으면 Pod 제거 비율이 초당 0.01 노드로 줄어듭니다. 노드가 50개 미만인 소규모 클러스터의 노드는 테인트되지 않습니다. 이러한 동작을 적용하려면 클러스터에 영역이 네 개 이상 있어야 합니다.

노드 사양에 `topology.kubernetes.io/region` 라벨을 적용하여 특정 영역에 노드를 할당합니다.

Kubernetes 영역의 노드 라벨 샘플

```
kind: Node
apiVersion: v1
metadata:
  labels:
    topology.kubernetes.io/region=east
```

KubeletConfig 오브젝트

Kubelet에서 각 노드의 상태를 확인하는 시간을 조정할 수 있습니다.

온프레미스 노드 컨트롤러에서 노드를 **Unhealthy** 또는 **Unreachable** 조건으로 표시하는 타이밍에 영

항을 미치는 간격을 설정하려면 `node-status-update-frequency` 및 `node-status-report-frequency` 매개변수가 포함된 `KubeletConfig` 오브젝트를 생성합니다.

각 노드의 `kubelet`은 `node-status-update-frequency` 설정에서 정의하는 노드 상태를 확인하고 `node-status-report-frequency` 설정에 따라 클러스터에 상태를 보고합니다. 기본적으로 `kubelet`은 Pod 상태를 10초 간격으로 확인하고 상태를 1분 간격으로 보고합니다. 그러나 노드 상태가 변경되면 `Kubelet`에서 이러한 변경을 클러스터에 즉시 보고합니다. `OpenShift Container Platform`에서는 노드 리스 기능 게이트가 활성화된 경우(`OpenShift Container Platform` 클러스터의 기본 상태)에만 `node-status-report-frequency` 설정을 사용합니다. 노드 리스 기능 게이트가 비활성화된 경우 노드는 `node-status-update-frequency` 설정에 따라 해당 상태를 보고합니다.

kubelet 구성의 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: worker 1
  kubeletConfig:
    node-status-update-frequency: 2
      - "10s"
    node-status-report-frequency: 3
      - "1m"
```

1

`MachineConfig` 오브젝트의 라벨을 사용하여 이 `KubeletConfig` 오브젝트가 적용되는 노드 유형을 지정합니다.

2

`Kubelet`에서 이 `MachineConfig` 오브젝트와 연결된 노드의 상태를 점검하는 빈도를 지정합니다. 기본값은 10s입니다. 이 기본값을 변경하면 `node-status-report-frequency` 값이 동일한 값으로 변경됩니다.

3

`Kubelet`에서 이 `MachineConfig` 오브젝트와 연결된 노드의 상태를 보고하는 빈도를 지정합니다. 기본값은 1m입니다.

node-status-update-frequency 매개변수는 **node-monitor-grace-period** 및 **pod-eviction-timeout** 매개변수와 함께 작동합니다.

- node-monitor-grace-period** 매개변수는 컨트롤러 관리자가 노드 하트비트를 수신하지 못하는 경우 **MachineConfig** 오브젝트와 연결된 노드가 **Unhealthy**로 표시된 후 **OpenShift Container Platform**에서 대기하는 시간을 지정합니다. 노드의 워크로드를 이 시간 이후 계속 실행됩니다. **node-monitor-grace-period**가 완료된 후 원격 작업자 노드가 클러스터에 다시 참여하면 **Pod**가 계속 실행됩니다. 해당 노드에 새 **Pod**를 예약할 수 있습니다. **node-monitor-grace-period** 간격은 40s입니다. **node-status-update-frequency** 값은 **node-monitor-grace-period** 값보다 작아야 합니다.
- pod-eviction-timeout** 매개변수는 **Pod**를 제거하도록 표시하기 위해 **MachineConfig** 오브젝트와 연결된 노드를 **Unreachable**로 표시한 후 **OpenShift Container Platform**에서 대기하는 시간을 지정합니다. 제거된 **Pod**는 다른 노드에 다시 예약됩니다. 노드 컨트롤러가 온프레미스에서 **Pod**를 제거했기 때문에 **pod-eviction-timeout**이 완료된 후 원격 작업자 노드가 클러스터에 다시 참여하면 원격 작업자 노드에서 실행 중인 **Pod**가 종료됩니다. 그런 다음 **Pod**를 해당 노드에 다시 예약할 수 있습니다. **pod-eviction-timeout** 기간은 5m0s입니다.



참고

node-monitor-grace-period 및 **pod-eviction-timeout** 매개변수 수정은 지원되지 않습니다.

허용 오차

온프레미스 노드 컨트롤러에서 **NoExecute** 효과가 있는 **node.kubernetes.io/unreachable** 테인트를 연결할 수 없는 노드에 추가하는 경우 **Pod** 허용 오차를 사용하여 영향을 완화할 수 있습니다.

NoExecute 효과가 있는 테인트는 다음과 같은 방식으로 노드에서 이미 실행되고 있는 **Pod**에 영향을 미칩니다.

- 해당 테인트를 허용하지 않는 **Pod**가 제거를 위해 큐에 추가됩니다.
- 허용 오차 사양에 **tolerationSeconds** 값을 지정하지 않은 상태로 테인트를 허용하는 **Pod**는 영구적으로 바인딩된 상태로 유지됩니다.
- tolerationSeconds** 값이 지정된 테인트를 허용하는 **Pod**는 지정된 시간 동안 바인딩된 상태로 유지됩니다. 시간이 지나면 **Pod**가 제거를 위해 큐에 추가됩니다.

`node.kubernetes.io/unreachable` 및 `node.kubernetes.io/not-ready` 테인트에 대해 `NoExecute` 효과를 사용하여 Pod 허용 오차를 구성하면 Pod 제거를 지연하거나 방지할 수 있습니다.

Pod 사양의 허용 오차 예

```
...
tolerations:
- key: "node.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute" 1
- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute" 2
  tolerationSeconds: 600
...
```

1

`tolerationSeconds` 없이 `NoExecute` 효과를 사용하면 컨트롤 플레인에서 노드에 연결할 수 없는 경우 Pod를 영구적으로 유지할 수 있습니다.

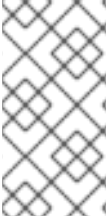
2

`tolerationSeconds: 600`을 사용하는 `NoExecute` 효과를 사용하면 컨트롤 플레인에서 노드를 `Unhealthy`로 표시하는 경우 Pod가 10분 동안 유지됩니다.

OpenShift Container Platform에서는 `pod-eviction-timeout` 값이 경과하면 `tolerationSeconds` 값을 사용합니다.

기타 유형의 OpenShift Container Platform 오브젝트

복제본 세트, 배포, 복제 컨트롤러를 사용할 수 있습니다. 노드가 5분 동안 연결이 끊기면 스케줄러에서 해당 Pod를 다른 노드에 다시 예약할 수 있습니다. 관리자가 특정 수의 Pod가 실행되고 해당 Pod에 액세스하도록 보장할 수 있는 일부 워크로드의 경우(예: REST API) 다른 노드에 다시 예약하면 유용할 수 있습니다.



참고

원격 작업자 노드로 작업할 때 원격 작업자 노드가 특정 기능을 위해 예약되도록 의도된 경우 다른 노드에 **Pod**를 다시 예약하지 못할 수 있습니다.

상태 저장 세트는 중단 발생 시 재시작되지 않습니다. 컨트롤 플레인에서 **Pod**가 종료되었음을 인식할 때까지 **Pod**는 **terminating** 상태로 유지됩니다.

네트워크 분리의 경우 **OpenShift Container Platform**에서는 동일한 유형의 영구 스토리지에 액세스할 수 없는 노드에 예약하지 않도록 영구 볼륨이 필요한 **Pod**를 다른 영역에 마이그레이션할 수 없습니다.

추가 리소스

- **Daemonsets**에 대한 자세한 내용은 **DaemonSets** 를 참조하십시오.
- 테인트 및 허용 오차에 대한 자세한 내용은 **노드 테인트를 사용하여 Pod 배치 제어**를 참조하십시오.
- **KubeletConfig** 오브젝트 구성에 대한 자세한 내용은 **KubeletConfig CRD 생성**을 참조하십시오.
- 복제본 세트에 대한 자세한 내용은 **ReplicaSets** 를 참조하십시오.
- 배포에 대한 자세한 내용은 배포를 **참조하십시오**.
- 복제 컨트롤러에 대한 자세한 내용은 복제 컨트롤러 **를** 참조하십시오.
- 컨트롤러 관리자에 대한 자세한 내용은 **Kubernetes Controller Manager Operator** 를 참조하십시오.

10장. 단일 노드 OPENS SHIFT 클러스터의 작업자 노드

10.1. 단일 노드 OPENS SHIFT 클러스터에 작업자 노드 추가

단일 노드 OpenShift 클러스터는 단일 호스트에 배포하기 위한 호스트 사전 요구 사항을 줄입니다. 이는 제한된 환경 또는 네트워크 엣지에 있는 배포에 유용합니다. 그러나 통신 및 네트워크 엣지 시나리오와 같이 클러스터에 용량을 더 추가해야 하는 경우가 있습니다. 이러한 시나리오에서는 단일 노드 클러스터에 작업자 노드를 추가할 수 있습니다.

단일 노드 클러스터에 작업자 노드를 추가할 수 있는 방법은 여러 가지가 있습니다. [Red Hat OpenShift Cluster Manager](#) 를 사용하거나 [Assisted Installer REST API](#)를 직접 사용하여 클러스터에 작업자 노드를 수동으로 추가할 수 있습니다.



중요

작업자 노드를 추가하면 클러스터 컨트롤 플레인 이 확장되지 않으며 클러스터에 고가용성을 제공하지 않습니다. 단일 노드 OpenShift 클러스터의 경우 다른 사이트로 장애 조치하여 고가용성을 처리합니다. 여러 개의 작업자 노드를 단일 노드 클러스터에 추가하지 않는 것이 좋습니다.



참고

멀티 노드 클러스터와 달리 기본적으로 모든 수신 트래픽은 작업자 노드를 추가한 후에도 단일 컨트롤 플레인 노드로 라우팅됩니다.

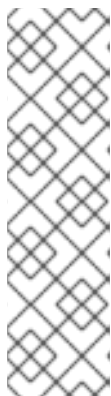
10.1.1. 단일 노드 OpenShift 작업자 노드를 설치하기 위한 요구사항

단일 노드 OpenShift 작업자 노드를 설치하려면 다음 요구 사항을 충족해야 합니다.

- **관리 호스트:** ISO를 준비하고 설치를 모니터링하려면 컴퓨터가 있어야 합니다.
- **production-grade 서버:** 단일 노드 OpenShift 작업자 노드를 설치하려면 OpenShift Container Platform 서비스 및 프로덕션 워크로드를 실행하기에 충분한 리소스가 있는 서버가 필요합니다.

표 10.1. 최소 리소스 요구사항

Profile	vCPU	메모리	스토리지
최소	vCPU 코어 2개	8GB RAM	100GB



참고

SMT(동시 멀티스레딩) 또는 하이퍼 스레딩이 활성화되지 않은 경우 하나의 vCPU는 하나의 물리적 코어와 동일합니다. 사용 가능한 경우 다음 공식을 사용하여 해당 비율을 계산합니다.

$$(threads\ per\ core \times cores) \times sockets = vCPUs$$

가상 미디어를 사용하여 부팅할 때 서버에 **BMC(Baseboard Management Controller)**가 있어야 합니다.

• **네트워킹:** 작업자 노드 서버는 인터넷에 액세스하거나 라우팅 가능한 네트워크에 연결되지 않은 경우 로컬 레지스트리에 액세스해야 합니다. 작업자 노드 서버에는 **DHCP** 예약 또는 고정 **IP** 주소가 있어야 하며 단일 노드 **OpenShift** 클러스터 **Kubernetes API**, 수신 경로 및 클러스터 노드 도메인 이름에 액세스할 수 있어야 합니다. 단일 노드 **OpenShift** 클러스터에 대해 **IP** 주소를 다음 정규화된 도메인 이름(**FQDN**)으로 확인하도록 **DNS**를 구성해야 합니다.

표 10.2. 필수 **DNS** 레코드

사용법	FQDN	설명
Kubernetes API	api.<cluster_name>.<base_domain>	DNS A/AAAA 또는 CNAME 레코드를 추가합니다. 이 레코드는 클러스터 외부의 클라이언트가 확인할 수 있어야 합니다.
내부 API	api-int.<cluster_name>.<base_domain>	ISO를 수동으로 생성할 때 DNS A/AAAA 또는 CNAME 레코드를 추가합니다. 이 레코드는 클러스터 내의 노드에서 확인할 수 있어야 합니다.
Ingress 경로	*.apps.<cluster_name>.<base_domain>	노드를 대상으로 하는 와일드카드 DNS A/AAAA 또는 CNAME 레코드를 추가합니다. 이 레코드는 클러스터 외부의 클라이언트가 확인할 수 있어야 합니다.

영구 IP 주소가 없으면 **apiserver** 와 **etcd** 간의 통신이 실패할 수 있습니다.

추가 리소스

- [클러스터 설치를 위한 최소 리소스 요구 사항](#)
- [클러스터 스케일링에 대한 권장 사례](#)
- [사용자 프로비저닝 DNS 요구사항](#)
- [USB 드라이브에 부팅 가능한 ISO 이미지 생성](#)
- [Redfish API를 사용하여 HTTP를 통해 제공되는 ISO 이미지에서 부팅](#)
- [클러스터에서 노드 삭제](#)

10.1.2. 지원 설치 프로그램 및 **OpenShift Cluster Manager**를 사용하여 작업자 노드 추가

[지원 설치 프로그램](#)을 사용하여 **Red Hat OpenShift Cluster Manager** 에서 생성된 단일 노드 **OpenShift** 클러스터에 작업자 노드를 추가할 수 있습니다.



중요

단일 노드 **OpenShift** 클러스터에 작업자 노드를 추가하는 것은 **OpenShift Container Platform** 버전 4.11 이상을 실행하는 클러스터에만 지원됩니다.

사전 요구 사항

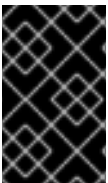
- [지원 설치 프로그램](#)을 사용하여 설치된 단일 노드 **OpenShift** 클러스터에 액세스할 수 있습니다.
- **OpenShift CLI(oc)**를 설치합니다.

- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- 작업자 노드를 추가하는 클러스터에 필요한 모든 **DNS** 레코드가 있는지 확인합니다.

절차

1. **OpenShift Cluster Manager** 에 로그인하고 작업자 노드를 추가할 단일 노드 클러스터를 클릭합니다.
2. 호스트 추가를 클릭하고 새 작업자 노드의 검색 **ISO**를 다운로드하여 **SSH** 공개 키를 추가하고 필요에 따라 클러스터 전체 프록시 설정을 구성합니다.
3. 검색 **ISO**를 사용하여 대상 호스트를 부팅하고 콘솔에서 호스트가 검색될 때까지 기다립니다. 호스트가 검색되면 설치를 시작합니다.
4. 설치가 진행되어 설치 과정에서 작업자 노드에 대한 보류 중인 인증서 서명 요청(**CSR**)이 생성됩니다. 메시지가 표시되면 보류 중인 **CSR**을 승인하여 설치를 완료합니다.

작업자 노드가 정상적으로 설치되면 클러스터 웹 콘솔에 작업자 노드로 나열됩니다.



중요

새 작업자 노드는 원래 클러스터와 동일한 방법으로 암호화됩니다.

추가 리소스

- [사용자 프로비저닝 DNS 요구사항](#)
- [머신의 인증서 서명 요청 승인](#)

10.1.3. 지원 설치 프로그램 API를 사용하여 작업자 노드 추가

Assisted Installer REST API를 사용하여 단일 노드 OpenShift 클러스터에 작업자 노드를 추가할 수 있습니다. 작업자 노드를 추가하기 전에 **OpenShift Cluster Manager** 에 로그인하고 API에 대해 인증해

야 합니다.

10.1.3.1. 보조 설치 관리자 REST API에 대해 인증

Assisted Installer REST API를 사용하려면 생성한 **JSON 웹 토큰(JWT)**을 사용하여 **API**에 대해 인증해야 합니다.

사전 요구 사항

- 클러스터 생성 권한이 있는 사용자로 **OpenShift Cluster Manager**에 로그인합니다.
- **jq**를 설치합니다.

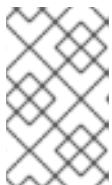
절차

1. **OpenShift Cluster Manager**에 로그인하고 **API** 토큰을 복사합니다.
2. 다음 명령을 실행하여 복사된 **API** 토큰을 사용하여 **\$OFFLINE_TOKEN** 변수를 설정합니다.

```
$ export OFFLINE_TOKEN=<copied_api_token>
```

3. 이전에 설정한 **\$OFFLINE_TOKEN** 변수를 사용하여 **\$JWT_TOKEN** 변수를 설정합니다.

```
$ export JWT_TOKEN=$(
  curl \
  --silent \
  --header "Accept: application/json" \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "grant_type=refresh_token" \
  --data-urlencode "client_id=cloud-services" \
  --data-urlencode "refresh_token=${OFFLINE_TOKEN}" \
  "https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token" \
  | jq --raw-output ".access_token"
)
```



참고

JWT 토큰은 15분 동안만 유효합니다.

검증

- 선택 사항: 다음 명령을 실행하여 API에 액세스할 수 있는지 확인합니다.

```
$ curl -s https://api.openshift.com/api/assisted-install/v2/component-versions -H
"Authorization: Bearer ${JWT_TOKEN}" | jq
```

출력 예

```
{
  "release_tag": "v2.5.1",
  "versions":
  {
    "assisted-installer": "registry.redhat.io/rhai-tech-preview/assisted-installer-
rhel8:v1.0.0-175",
    "assisted-installer-controller": "registry.redhat.io/rhai-tech-preview/assisted-
installer-reporter-rhel8:v1.0.0-223",
    "assisted-installer-service": "quay.io/app-sre/assisted-service:ac87f93",
    "discovery-agent": "registry.redhat.io/rhai-tech-preview/assisted-installer-agent-
rhel8:v1.0.0-156"
  }
}
```

10.1.3.2. 도움말 설치 프로그램 REST API를 사용하여 작업자 노드 추가

보조 설치 관리자 REST API를 사용하여 클러스터에 작업자 노드를 추가할 수 있습니다.

사전 요구 사항

- **OpenShift Cluster Manager CLI(ocm)**를 설치합니다.
- 클러스터 생성 권한이 있는 사용자로 **OpenShift Cluster Manager** 에 로그인합니다.
- **jq** 를 설치합니다.
- 작업자 노드를 추가하는 클러스터에 필요한 모든 **DNS** 레코드가 있는지 확인합니다.

절차

1. 지원되는 설치 관리자 REST API에 대해 인증하고 세션에 대한 JSON 웹 토큰(JWT)을 생성합니다. 생성된 JWT 토큰은 15분 동안만 유효합니다.
2. 다음 명령을 실행하여 \$API_URL 변수를 설정합니다.

```
$ export API_URL=<api_url> 1
```

1

<api_url>을 Assisted Installer API URL로 교체합니다(예: <https://api.openshift.com>)

3. 다음 명령을 실행하여 단일 노드 OpenShift 클러스터를 가져옵니다.
 - a. \$OPENSIFT_CLUSTER_ID 변수를 설정합니다. 클러스터에 로그인하고 다음 명령을 실행합니다.

```
$ export OPENSIFT_CLUSTER_ID=$(oc get clusterversion -o jsonpath='{.items[].spec.clusterID}')
```

- b. 클러스터를 가져오는 데 사용되는 \$CLUSTER_REQUEST 변수를 설정합니다.

```
$ export CLUSTER_REQUEST=$(jq --null-input --arg openshift_cluster_id "$OPENSIFT_CLUSTER_ID" '{
  "api_vip_dnsname": "<api_vip>", 1
  "openshift_cluster_id": $openshift_cluster_id,
  "name": "<openshift_cluster_name>" 2
}')
```

1

<api_vip>를 클러스터의 API 서버의 호스트 이름으로 바꿉니다. 이는 API 서버의 DNS 도메인 또는 작업자 노드가 연결할 수 있는 단일 노드의 IP 주소일 수 있습니다. 예: api.compute-1.example.com.

2

<openshift_cluster_name>을 클러스터의 일반 텍스트 이름으로 바꿉니다. 클러스터 이름은 1일 차 클러스터 설치 중에 설정된 클러스터 이름과 일치해야 합니다.

c.

클러스터를 가져오고 `$CLUSTER_ID` 변수를 설정합니다. 다음 명령을 실행합니다.

```
$ CLUSTER_ID=$(curl "$API_URL/api/assisted-install/v2/clusters/import" -H
"Authorization: Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-
Type: application/json' \
-d "$CLUSTER_REQUEST" | tee /dev/stderr | jq -r '.id')
```

4.

클러스터에 대한 `InfraEnv` 리소스를 생성하고 다음 명령을 실행하여 `$INFRA_ENV_ID` 변수를 설정합니다.

a.

console.redhat.com 의 Red Hat OpenShift Cluster Manager에서 풀 시크릿 파일을 다운로드합니다.

b.

`$INFRA_ENV_REQUEST` 변수를 설정합니다.

```
export INFRA_ENV_REQUEST=$(jq --null-input \
--slurpfile pull_secret <path_to_pull_secret_file> | 1
--arg ssh_pub_key "$(cat <path_to_ssh_pub_key>)" | 2
--arg cluster_id "$CLUSTER_ID" '{
"name": "<infraenv_name>", 3
"pull_secret": $pull_secret[0] | tojson,
"cluster_id": $cluster_id,
"ssh_authorized_key": $ssh_pub_key,
"image_type": "<iso_image_type>" 4
}')
```

1

< path_to_pull_secret_file >을 console.redhat.com 의 Red Hat OpenShift Cluster Manager에서 다운로드한 풀 시크릿이 포함된 로컬 파일의 경로로 바꿉니다.

2

& lt;path_to_ssh_pub_key >를 호스트에 액세스하는 데 필요한 공용 SSH 키의 경로로 바꿉니다. 이 값을 설정하지 않으면 검색 모드에서는 호스트에 액세스할 수 없습니다.

3

& lt;infraenv_name> 을 `InfraEnv` 리소스의 일반 텍스트 이름으로 바꿉니다.

4

& It;iso_image_type >을 full-iso 또는 minimal-iso 로 바꿉니다.

c.

`$INFRA_ENV_REQUEST` 를 `/v2/infra-envs` API에 게시하고 `$INFRA_ENV_ID` 변수를 설정합니다.

```
$ INFRA_ENV_ID=$(curl "$API_URL/api/assisted-install/v2/infra-envs" -H
"Authorization: Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-
Type: application/json' -d "$INFRA_ENV_REQUEST" | tee /dev/stderr | jq -r '.id')
```

5.

다음 명령을 실행하여 클러스터 작업자 노드의 검색 ISO URL을 가져옵니다.

```
$ curl -s "$API_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID" -H
"Authorization: Bearer ${JWT_TOKEN}" | jq -r '.download_url'
```

출력 예

```
https://api.openshift.com/api/assisted-images/images/41b91e72-c33e-42ee-b80f-
b5c5bbf6431a?
arch=x86_64&image_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiJlE2NTY
wMjYzNzEsInN1Yil6IjQxYjYkxZTcyLWZmM2UtNDJlZS1iODBmLWI1YzViYmY2NDMxYSJ9.
1EX_VGaMNejMhrAvVRBS7PDPIQtbOOc8LtG8OukE1a4&type=minimal-
iso&version=4.12
```

6.

ISO를 다운로드합니다.

```
$ curl -L -s '<iso_url>' --output rhcos-live-minimal.iso 1
```

1

& It;iso_url >을 이전 단계의 ISO URL로 바꿉니다.

7.

다운로드한 `rhcos-live-minimal.iso` 에서 새 작업자 호스트를 부팅합니다.

8.

설치 되지 않은 클러스터의 호스트 목록을 가져옵니다. 새 호스트가 나타날 때까지 다음 명령을 계속 실행하십시오.

```
$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer ${JWT_TOKEN}" | jq -r '.hosts[] | select(.status != "installed").id'
```

출력 예

```
2294ba03-c264-4f11-ac08-2f1bb2f8c296
```

9.

새 작업자 노드의 `$HOST_ID` 변수를 설정합니다. 예를 들면 다음과 같습니다.

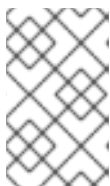
```
$ HOST_ID=<host_id> 1
```

1

<host_id>를 이전 단계의 호스트 ID로 바꿉니다.

10.

다음 명령을 실행하여 호스트를 설치할 준비가 되었는지 확인합니다.



참고

전체 jq 표현식을 포함하여 전체 명령을 복사해야 합니다.

```
$ curl -s $API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID -H "Authorization: Bearer ${JWT_TOKEN}" | jq '
def host_name($host):
  if (.suggested_hostname // "") == "" then
    if (.inventory // "") == "" then
      "Unknown hostname, please wait"
    else
      .inventory | fromjson | .hostname
    end
  else
    .suggested_hostname
  end;

def is_notable($validation):
```

```

["failure", "pending", "error"] | any( == $validation.status);

def notable_validations($validations_info):
[
  $validations_info // "{}"
  | fromjson
  | to_entries[].value[]
  | select(is_notable(.))
];

{
  "Hosts validations": {
    "Hosts": [
      .hosts[]
      | select(.status != "installed")
      | {
        "id": .id,
        "name": host_name(.),
        "status": .status,
        "notable_validations": notable_validations(.validations_info)
      }
    ]
  },
  "Cluster validations info": {
    "notable_validations": notable_validations(.validations_info)
  }
}
'-r

```

출력 예

```

{
  "Hosts validations": {
    "Hosts": [
      {
        "id": "97ec378c-3568-460c-bc22-df54534ff08f",
        "name": "localhost.localdomain",
        "status": "insufficient",
        "notable_validations": [
          {
            "id": "ntp-synced",
            "status": "failure",
            "message": "Host couldn't synchronize with any NTP server"
          },
          {
            "id": "api-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          },
          {
            "id": "api-int-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          }
        ]
      }
    ]
  }
}

```

```

    },
    {
      "id": "apps-domain-name-resolved-correctly",
      "status": "error",
      "message": "Parse error for domain name resolutions result"
    }
  ]
}
]
},
"Cluster validations info": {
  "notable_validations": []
}
}
}

```

11.

이전 명령이 호스트가 준비되었다고 표시되면 다음 명령을 실행하여 `/v2/infra-envs/{infra_env_id}/hosts/{host_id}/install` API를 사용하여 설치를 시작합니다.

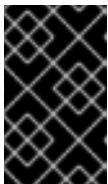
```

$ curl -X POST -s "$API_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts/$HOST_ID/actions/install" -H "Authorization: Bearer ${JWT_TOKEN}"

```

12.

설치가 진행되어 설치 과정에서 작업자 노드에 대한 보류 중인 인증서 서명 요청(CSR)이 생성됩니다.



중요

설치를 완료하려면 CSR을 승인해야 합니다.

클러스터 설치를 모니터링하려면 다음 API 호출을 계속 실행합니다.

```

$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer ${JWT_TOKEN}" | jq '{
  "Cluster day-2 hosts":
    [
      .hosts[]
      | select(.status != "installed")
      | {id, requested_hostname, status, status_info, progress, status_updated_at,
        updated_at, infra_env_id, cluster_id, created_at}
    ]
}'

```

출력 예

```
{
  "Cluster day-2 hosts": [
    {
      "id": "a1c52dde-3432-4f59-b2ae-0a530c851480",
      "requested_hostname": "control-plane-1",
      "status": "added-to-existing-cluster",
      "status_info": "Host has rebooted and no further updates will be posted. Please
check console for progress and to possibly approve pending CSRs",
      "progress": {
        "current_stage": "Done",
        "installation_percentage": 100,
        "stage_started_at": "2022-07-08T10:56:20.476Z",
        "stage_updated_at": "2022-07-08T10:56:20.476Z"
      },
      "status_updated_at": "2022-07-08T10:56:20.476Z",
      "updated_at": "2022-07-08T10:57:15.306369Z",
      "infra_env_id": "b74ec0c3-d5b5-4717-a866-5b6854791bd3",
      "cluster_id": "8f721322-419d-4eed-aa5b-61b50ea586ae",
      "created_at": "2022-07-06T22:54:57.161614Z"
    }
  ]
}
```

13.

선택 사항: 다음 명령을 실행하여 클러스터의 모든 이벤트를 확인합니다.

```
$ curl -s "$API_URL/api/assisted-install/v2/events?cluster_id=$CLUSTER_ID" -H
"Authorization: Bearer ${JWT_TOKEN}" | jq -c '.[] | {severity, message, event_time,
host_id}'
```

출력 예

```
{"severity":"info","message":"Host compute-0: updated status from insufficient to
known (Host is ready to be installed)","event_time":"2022-07-
08T11:21:46.346Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from known to
installing (Installation is in progress)","event_time":"2022-07-
08T11:28:28.647Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing to
installing-in-progress (Starting installation)","event_time":"2022-07-
08T11:28:52.068Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Uploaded logs for host compute-0 cluster 8f721322-
419d-4eed-aa5b-61b50ea586ae","event_time":"2022-07-
```



```
08T11:29:47.802Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing-in-
progress to added-to-existing-cluster (Host has rebooted and no further updates will
be posted. Please check console for progress and to possibly approve pending
CSRs)","event_time":"2022-07-08T11:29:48.259Z","host_id":"9d7b3b44-1125-4ad0-
9b14-76550087b445"}
{"severity":"info","message":"Host: compute-0, reached installation stage
Rebooting","event_time":"2022-07-08T11:29:48.261Z","host_id":"9d7b3b44-1125-4ad0-
9b14-76550087b445"}
```

14.

클러스터에 로그인하고 보류 중인 **CSR**을 승인하여 설치를 완료합니다.

검증

- 상태가 **Ready** 인 클러스터에 새 작업자 노드가 성공적으로 추가되었는지 확인합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
control-plane-1.example.com	Ready	master,worker	56m	v1.25.0
compute-1.example.com	Ready	worker	11m	v1.25.0

추가 리소스

- [사용자 프로비저닝 DNS 요구사항](#)
- [머신의 인증서 서명 요청 승인](#)

10.1.4. 수동으로 단일 노드 OpenShift 클러스터에 작업자 노드 추가

RHCOS(Red Hat Enterprise Linux CoreOS) ISO에서 작업자 노드를 부팅하고 클러스터 **worker.ign** 파일을 사용하여 새 작업자 노드를 클러스터에 조인하여 수동으로 단일 노드 **OpenShift** 클러스터에 작업

자 노드를 추가할 수 있습니다.

사전 요구 사항

- 베어 메탈에 단일 노드 **OpenShift** 클러스터를 설치합니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- 작업자 노드를 추가하는 클러스터에 필요한 모든 **DNS** 레코드가 있는지 확인합니다.

절차

1. **OpenShift Container Platform** 버전을 설정합니다.

```
$ OCP_VERSION=<ocp_version> 1
```

1

& It;ocp_version& gt;을 현재 버전으로 바꿉니다(예: latest-4.12)

2. 호스트 아키텍처를 설정합니다.

```
$ ARCH=<architecture> 1
```

1

& It;architecture& gt;를 대상 호스트 아키텍처(예: aarch64 또는 x86_64)로 바꿉니다.

3. 다음 명령을 실행하여 실행 중인 단일 노드 클러스터에서 **worker.ign** 데이터를 가져옵니다.

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to- > worker.ign
```

4. 네트워크에서 액세스할 수 있는 웹 서버에서 **worker.ign** 파일을 호스팅합니다.

5. **OpenShift Container Platform** 설치 프로그램을 다운로드하여 다음 명령을 실행하여 사용할 수 있도록 합니다.

```
$ curl -k https://mirror.openshift.com/pub/openshift-
v4/clients/ocp/$OCP_VERSION/openshift-install-
linux.tar.gz > openshift-install-
linux.tar.gz
```

```
$ tar zxvf openshift-install-linux.tar.gz
```

```
$ chmod +x openshift-install
```

6. **RHCOS ISO URL**을 검색합니다.

```
$ ISO_URL=$(./openshift-install coreos print-stream-json | grep location | grep $ARCH
| grep iso | cut -d\" -f4)
```

7. **RHCOS ISO**를 다운로드합니다.

```
$ curl -L $ISO_URL -o rhcos-live.iso
```

8. **RHCOS ISO**와 호스팅된 **worker.ign** 파일을 사용하여 작업자 노드를 설치합니다.

a. **RHCOS ISO** 및 기본 설치 방법으로 대상 호스트를 부팅합니다.

b. 대상 호스트가 **RHCOS ISO**에서 부팅되면 대상 호스트에서 콘솔을 엽니다.

c. 로컬 네트워크에 **DHCP**가 활성화되어 있지 않은 경우 **RHCOS** 설치를 실행하기 전에 새 호스트 이름으로 **Ignition** 파일을 생성하고 작업자 노드의 고정 IP 주소를 구성해야 합니다. 다음 단계를 수행합니다.

i. 고정 IP를 사용하여 작업자 호스트 네트워크 연결을 구성합니다. 대상 호스트 콘솔에서 다음 명령을 실행합니다.

```
$ nmcli con mod <network_interface> ipv4.method manual /
```

```

ipv4.addresses <static_ip> ipv4.gateway <network_gateway> ipv4.dns
<dns_server> /
802-3-ethernet.mtu 9000
    
```

다음과 같습니다.

<static_ip>

는 호스트 고정 IP 주소와 CIDR입니다(예: 10.1.101.50/24).

<network_gateway>

네트워크 게이트웨이(예: 10.1.101.1)

ii.

수정된 네트워크 인터페이스를 활성화합니다.

```

$ nmcli con up <network_interface>
    
```

iii.

원래 **worker.ign** 에 대한 참조와 **coreos-installer** 프로그램이 새 작업자 호스트의 **/etc/hostname** 파일을 채우는 데 사용하는 추가 명령을 포함하는 새로운 Ignition 파일 **new-worker.ign** 을 만듭니다. 예를 들면 다음과 같습니다.

```

{
  "ignition":{
    "version":"3.2.0",
    "config":{
      "merge":[
        {
          "source":"<hosted_worker_ign_file>" 1
        }
      ]
    }
  },
  "storage":{
    "files":[
      {
        "path":"/etc/hostname",
        "contents":{
          "source":"data:,<new_fqdn>" 2
        },
        "mode":420,
        "overwrite":true,
        "path":"/etc/hostname"
      }
    ]
  }
}
    
```

1

<hosted_worker_ign_file >은 원래 `worker.ign` 파일의 로컬에 액세스할 수 있는 URL입니다. 예: <http://webserver.example.com/worker.ign>

2

<new_fqdn >은 작업자 노드에 설정한 새 FQDN입니다. 예를 들면 `new-worker.example.com` 입니다.

iv.

네트워크에서 액세스할 수 있는 웹 서버에서 `new-worker.ign` 파일을 호스팅합니다.

v.

다음 `coreos-installer` 명령을 실행하여 `ignition-url` 및 하드 디스크 세부 정보를 전달합니다.

```
$ sudo coreos-installer install --copy-network /
--ignition-url=<new_worker_ign_file> <hard_disk> --insecure-ignition
```

다음과 같습니다.

<new_worker_ign_file>

은 호스팅된 `new-worker.ign` 파일의 로컬에 액세스할 수 있는 URL입니다 (예: <http://webserver.example.com/new-worker.ign>).

<hard_disk>

RHCOS를 설치하는 하드 디스크입니다(예: `/dev/sda`)

d.

DHCP가 활성화된 네트워크의 경우 고정 IP를 설정할 필요가 없습니다. 대상 호스트 콘솔에서 다음 `coreos-installer` 명령을 실행하여 시스템을 설치합니다.

```
$ coreos-installer install --ignition-url=<hosted_worker_ign_file> <hard_disk>
```

e.

DHCP를 수동으로 활성화하려면 다음 `NMStateConfig CR`을 단일 노드 OpenShift 클러스터에 적용합니다.

```
apiVersion: agent-install.openshift.io/v1
kind: NMStateConfig
metadata:
```

```

name: nmstateconfig-dhcp
namespace: example-sno
labels:
  nmstate_config_cluster_name: <nmstate_config_cluster_label>
spec:
  config:
    interfaces:
      - name: eth0
        type: ethernet
        state: up
        ipv4:
          enabled: true
          dhcp: true
        ipv6:
          enabled: false
    interfaces:
      - name: "eth0"
        macAddress: "AA:BB:CC:DD:EE:11"

```



중요

NMStateConfig CR은 고정 IP 주소를 사용하여 작업자 노드를 성공적으로 배포하고 단일 노드 **OpenShift**가 고정 IP 주소로 배포된 경우 동적 IP 주소가 있는 작업자 노드를 추가하는 데 필요합니다. 클러스터 네트워크 **DHCP**가 새 작업자 노드에 대해 이러한 네트워크 설정을 자동으로 설정하지 않습니다.

9.

설치가 진행되어 설치 과정에서 작업자 노드에 대한 보류 중인 인증서 서명 요청(**CSR**)이 생성됩니다. 메시지가 표시되면 보류 중인 **CSR**을 승인하여 설치를 완료합니다.

10.

설치가 완료되면 호스트를 재부팅합니다. 호스트는 클러스터에 새 작업자 노드로 참여합니다.

검증

-

상태가 **Ready** 인 클러스터에 새 작업자 노드가 성공적으로 추가되었는지 확인합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
control-plane-1.example.com	Ready	master,worker	56m	v1.25.0
compute-1.example.com	Ready	worker	11m	v1.25.0

추가 리소스

- [사용자 프로비저닝 DNS 요구사항](#)
- [머신의 인증서 서명 요청 승인](#)

10.1.5. 머신의 인증서 서명 요청 승인

클러스터에 시스템을 추가하면 추가한 시스템별로 보류 중인 인증서 서명 요청(CSR)이 두 개씩 생성됩니다. 이러한 CSR이 승인되었는지 확인해야 하며, 필요한 경우 이를 직접 승인해야 합니다. 클라이언트 요청을 먼저 승인한 다음 서버 요청을 승인해야 합니다.

사전 요구 사항

- 클러스터에 시스템을 추가했습니다.

프로세스

1. 클러스터가 시스템을 인식하는지 확인합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
master-0	Ready	master	63m	v1.25.0
master-1	Ready	master	63m	v1.25.0
master-2	Ready	master	64m	v1.25.0

출력에 생성된 모든 시스템이 나열됩니다.



참고

이전 출력에는 일부 **CSR**이 승인될 때까지 컴퓨팅 노드(작업자 노드라고도 함)가 포함되지 않을 수 있습니다.

2.

보류 중인 **CSR**을 검토하고 클러스터에 추가한 각 시스템에 대해 **Pending** 또는 **Approved** 상태의 클라이언트 및 서버 요청이 표시되는지 확인합니다.

```
$ oc get csr
```

출력 예

NAME	AGE	REQUESTOR	CONDITION
csr-8b2br	15m	system:serviceaccount:openshift-machine-config-operator:node-bootstrapper	Pending
csr-8vnps	15m	system:serviceaccount:openshift-machine-config-operator:node-bootstrapper	Pending
...			

예에서는 두 시스템이 클러스터에 참여하고 있습니다. 목록에는 승인된 **CSR**이 더 많이 나타날 수도 있습니다.

3.

CSR이 승인되지 않은 경우, 추가된 시스템에 대한 모든 보류 중인 **CSR**이 **Pending** 상태로 전환된 후 클러스터 시스템의 **CSR**을 승인합니다.



참고

CSR은 교체 주기가 자동으로 만료되므로 클러스터에 시스템을 추가한 후 1 시간 이내에 **CSR**을 승인하십시오. 한 시간 내에 승인하지 않으면 인증서가 교체되고 각 노드에 대해 두 개 이상의 인증서가 표시됩니다. 이러한 인증서를 모두 승인해야 합니다. 클라이언트 **CSR**이 승인되면 **Kubelet**은 인증서에 대한 보조 **CSR**을 생성하므로 수동 승인이 필요합니다. 그러면 **Kubelet**에서 동일한 매개변수를 사용하여 새 인증서를 요청하는 경우 인증서 갱신 요청은 **machine-approver**에 의해 자동으로 승인됩니다.



참고

베어 메탈 및 기타 사용자 프로비저닝 인프라와 같이 머신 **API**를 사용하도록 활성화되지 않는 플랫폼에서 실행되는 클러스터의 경우 **CSR(Kubelet service Certificate Request)**을 자동으로 승인하는 방법을 구현해야 합니다. 요청이 승인되지 않으면 **API** 서버가 **kubelet**에 연결될 때 서비스 인증서가 필요하므로 **oc exec, oc rsh, oc logs** 명령을 성공적으로 수행할 수 없습니다. **Kubelet** 엔드 포인트에 연결하는 모든 작업을 수행하려면 이 인증서 승인이 필요합니다. 이 방법은 새 **CSR**을 감시하고 **CSR**이 **system:node** 또는 **system:admin** 그룹의 **node-bootstrapper** 서비스 계정에 의해 제출되었는지 확인하고 노드의 **ID**를 확인합니다.

•

개별적으로 승인하려면 유효한 **CSR** 각각에 대해 다음 명령을 실행하십시오.

```
$ oc adm certificate approve <csr_name> 1
```

1

<csr_name>은 현재 **CSR** 목록에 있는 **CSR**의 이름입니다.

•

보류 중인 **CSR**을 모두 승인하려면 다음 명령을 실행하십시오.

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}\n\n{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```



참고

일부 **Operator**는 일부 **CSR**이 승인될 때까지 사용할 수 없습니다.

4.

이제 클라이언트 요청이 승인되었으므로 클러스터에 추가한 각 머신의 서버 요청을 검토해야 합니다.

```
$ oc get csr
```

출력 예

```
NAME      AGE  REQUESTOR                                     CONDITION
csr-bfd72 5m26s system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv 5m26s system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5.

나머지 **CSR**이 승인되지 않고 **Pending** 상태인 경우 클러스터 머신의 **CSR**을 승인합니다.

-

개별적으로 승인하려면 유효한 **CSR** 각각에 대해 다음 명령을 실행하십시오.

```
$ oc adm certificate approve <csr_name> 1
```

1

<csr_name>은 현재 **CSR** 목록에 있는 **CSR**의 이름입니다.

-

보류 중인 **CSR**을 모두 승인하려면 다음 명령을 실행하십시오.

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}
{{"\n"}}{{end}}{{end}}' | xargs oc adm certificate approve
```

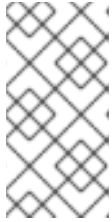
6.

모든 클라이언트 및 서버 **CSR**이 승인된 후 머신은 **Ready** 상태가 됩니다. 다음 명령을 실행하여 확인합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
master-0	Ready	master	73m	v1.25.0
master-1	Ready	master	73m	v1.25.0
master-2	Ready	master	74m	v1.25.0
worker-0	Ready	worker	11m	v1.25.0
worker-1	Ready	worker	11m	v1.25.0



참고

머신이 **Ready** 상태로 전환하는 데 서버 **CSR**의 승인 후 몇 분이 걸릴 수 있습니다.

추가 정보

- **CSR**에 대한 자세한 내용은 [인증서 서명 요청](#)을 참조하십시오.