OpenShift Dedicated 4

노드

OpenShift Dedicated 노드

Last Updated: 2024-09-15

OpenShift Dedicated 4 노드

OpenShift Dedicated 노드

법적 공지

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

1.1. 노드 정보

노드는 Kubernetes 클러스터의 가상 또는 베어 메탈 머신입니다. 작업자 노드는 포드로 그룹화된 애플리케이션 컨테이너를 호스팅합니다. 컨트롤 플레인 노드는 Kubernetes 클러스터를 제어하는 데 필요한 서비스를 실행합니다. OpenShift Dedicated에서 컨트롤 플레인 노드에는 OpenShift Dedicated 클러스터를 관리하기 위한 Kubernetes 서비스 이상의 것이 포함되어 있습니다.

클러스터에서 안정적이고 정상적인 노드를 보유하는 것은 호스팅된 애플리케이션의 원활한 작동을 위한 필수 요소입니다. OpenShift Dedicated에서는 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 Pod의 실행을 유지 관리하고 Kubernetes 런타임 환경을 제공합니다.

컨테이너 런타임

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

kubelet

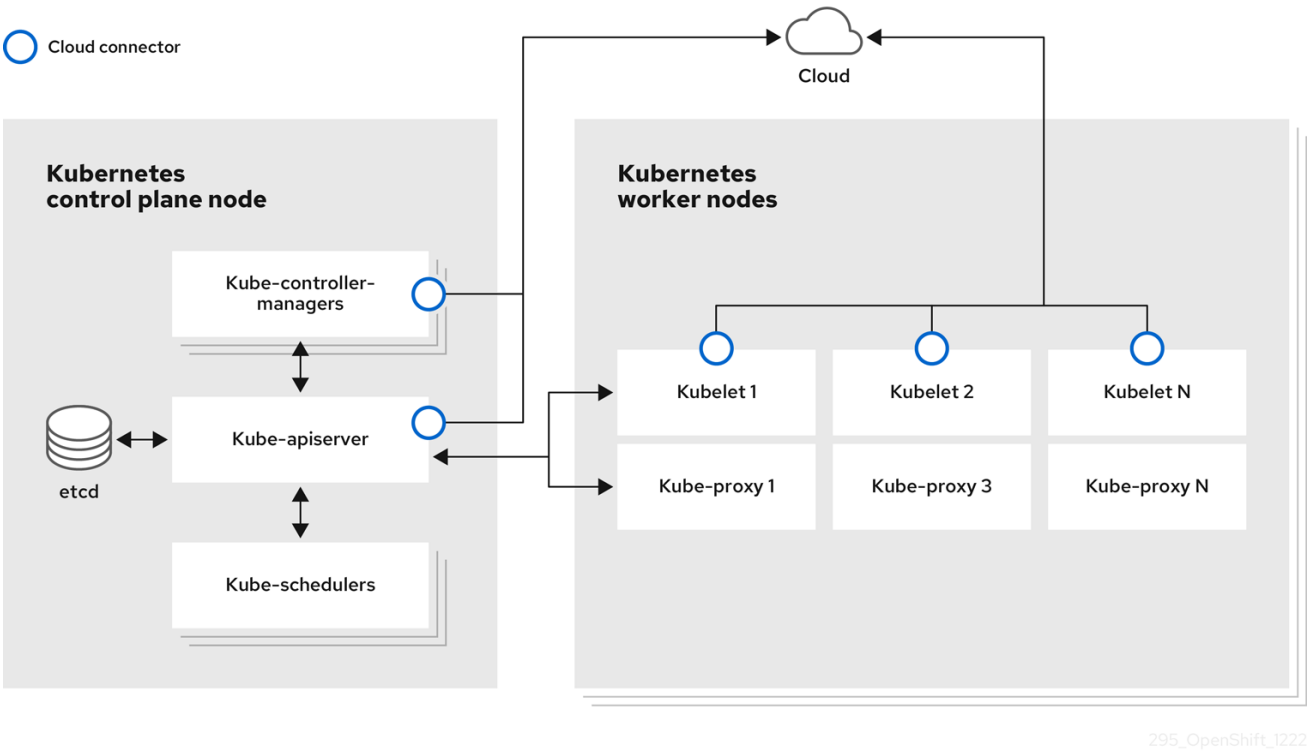
kubelet은 노드에서 실행되며 컨테이너 매니페스트를 읽습니다. 이렇게 하면 정의된 컨테이너가 시작되어 실행 중인지 확인합니다. kubelet 프로세스는 작업 상태와 노드 서버를 유지 관리합니다. kubelet은 네트워크 규칙 및 포트 전달을 관리합니다. kubelet은 Kubernetes에서만 생성된 컨테이너를 관리합니다.

kube-proxy

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

DNS

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



읽기 작업

읽기 작업을 사용하면 관리자 또는 개발자가 OpenShift Dedicated 클러스터의 노드에 대한 정보를 가져올 수 있습니다.

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

기능 개선 작업

OpenShift Dedicated를 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 Operator를 사용하여 일정 수준의 커널 튜닝이 필요한 고성능 애플리케이션에 대한 노드 수준 튜닝을 관리합니다.
- 데몬 세트를 사용하여 노드에서 백그라운드 작업을 자동으로 실행합니다 . 데몬 세트를 생성하고 사용하여 공유 스토리지를 생성하거나, 모든 노드에서 로깅 Pod를 실행하거나, 모든 노드에 모니터링 에이전트를 배포할 수 있습니다.

1.2. POD 정보

Pod는 노드에 함께 배포되는 하나 이상의 컨테이너입니다. 클러스터 관리자는 Pod를 정의하고 예약할 준비가 된 정상 노드에서 실행하도록 할당할 수 있습니다. Pod는 컨테이너가 실행되는 동안 실행됩니다. Pod가 정의되고 실행되면 변경할 수 없습니다. Pod로 작업할 때 수행할 수 있는 일부 작업은 다음과 같습니다.

읽기 작업

관리자는 다음 작업을 통해 프로젝트의 Pod에 대한 정보를 가져올 수 있습니다.

- 복제본 수 및 재시작, 현재 상태 및 경과와 같은 정보를 포함하여 프로젝트와 연결된 Pod를 나열합니다.
- CPU, 메모리, 스토리지 소비와 같은 Pod 사용량 통계를 확인합니다.

관리 작업
다음 작업 목록은 관리자가 OpenShift Dedicated 클러스터에서 Pod를 관리하는 방법에 대한 개요를 제공합니다.

- OpenShift Dedicated에서 사용할 수 있는 고급 스케줄링 기능을 사용하여 Pod 예약을 제어합니다.
  - Pod 유사성, 노드 유사성 및 유사성 방지와 같은 node -to- pod 바인딩 규칙입니다.
  - 노드 레이블 및 선택기.
  - Pod 토폴로지 분배 제약 조건.
- 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 Dedicated에서 사용할 수 있는 다양한 툴 및 기능의 도움을 통해 Pod를 보다 쉽고 효율적으로 사용할 수 있습니다. 다음 작업에는 Pod를 더 잘 관리하기 위해 해당 툴과 기능을 사용하는 작업이 포함됩니다.

- 보안: 일부 애플리케이션에는 암호 및 사용자 이름과 같은 민감한 정보가 필요합니다. 관리자는 **Secret** 오브젝트를 사용하여 **Secret** 오브젝트 를 사용하여 Pod에 중요한 데이터를 제공할 수 있습니다.

1.3. 컨테이너 정보

컨테이너는 OpenShift Dedicated 애플리케이션의 기본 단위이며, 종속 항목, 라이브러리 및 바이너리와 함께 패키지된 애플리케이션 코드로 구성됩니다. 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 격리하고 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 Dedicated는 Init 컨테이너라는 특수 컨테이너를 제공합니다. Init 컨테이너는 애플리케이션 컨테이너보다 먼저 실행되며 애플리케이션 이미지에 없는 유틸리티 또는 설정 스크립트를 포함할 수 있습니다. Init 컨테이너를 사용하여 나머지 Pod를 배포하기 전에 작업을 수행할 수 있습니다.

노드, Pod 및 컨테이너에서 특정 작업을 수행하는 것 외에도 전체 OpenShift Dedicated 클러스터에서 작업하여 클러스터 효율성과 애플리케이션 pod의 가용성을 유지할 수 있습니다.

1.4. OPENSHIFT DEDICATED 노드의 일반 용어집

이 용어집은 노드 콘텐츠에 사용되는 일반적인 용어를 정의합니다.

컨테이너

이는 소프트웨어와 모든 종속 항목을 포함하는 가볍고 실행 가능한 이미지입니다. 따라서 컨테이너는 운영 체제를 가상화하므로 데이터 센터에서 퍼블릭 또는 프라이빗 클라우드, 개발자의 노트북까지 컨테이너를 실행할 수 있습니다.

데몬 세트

Pod 복제본이 OpenShift Dedicated 클러스터의 적격 노드에서 실행되는지 확인합니다.

egress

Pod의 네트워크 아웃 바운드 트래픽을 통해 외부적으로 데이터 공유 프로세스.

가비지 컬렉션

실행 중인 Pod에서 참조하지 않는 종료된 컨테이너 및 이미지와 같은 클러스터 리소스를 정리하는 프로세스입니다.

Ingress

Pod로 들어오는 트래픽입니다.

작업

완료까지 실행되는 프로세스입니다. 작업은 하나 이상의 Pod 오브젝트를 생성하고 지정된 Pod가 성공적으로 완료되었는지 확인합니다.

라벨

키-값 쌍인 레이블을 사용하여 Pod와 같은 오브젝트 하위 집합을 구성하고 선택할 수 있습니다.

노드

OpenShift Dedicated 클러스터의 작업자 시스템입니다. 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. 이렇게 하면 데몬이 이해할 수 있는 형식으로 클러스터에서 실행되는 모든 컨테이너화된 TuneD 데몬에 사용자 정의 튜닝 사양이 전달됩니다. 데몬은 클러스터의 모든 노드에서 노드당 하나씩 실행됩니다.

Self Node Remediation Operator

Operator는 클러스터 노드에서 실행되며 비정상인 노드를 식별하고 재부팅합니다.

Pod

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

톨러레이션

일치하는 테인트가 있는 노드 또는 노드 그룹에 Pod를 예약할 수 있지만 필요하지는 않음을 나타냅니다. 허용 오차를 사용하여 스케줄러에서 일치하는 테인트로 Pod를 예약할 수 있습니다.

taint

key,value 및 effect로 구성된 코어 오브젝트입니다. 테인트 및 허용 오차가 함께 작동하여 관련 노드에서 Pod를 예약하지 않도록 합니다.

2장. 노드 작업

2.1. POD 사용

*Pod*는 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 Pod에는 자체 내부 IP 주소가 할당되므로 해당 Pod가 전체 포트 공간을 소유하고 Pod 내의 컨테이너는 로컬 스토리지와 네트워킹을 공유할 수 있습니다.

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

OpenShift Dedicated는 포드를 대체로 변경할 수 없는 것으로 취급합니다. 실행 중에 포드 정의를 변경할 수 없습니다. OpenShift Dedicated는 기존 pod를 종료하고 수정된 구성, 기본 이미지 또는 둘 다로 다시 생성하여 변경 사항을 구현합니다. Pod를 다시 생성하면 확장 가능한 것으로 취급되고 상태가 유지되지 않습니다. 따라서 일반적으로 Pod는 사용자가 직접 관리하는 대신 상위 수준의 컨트롤러에서 관리해야 합니다.

주의

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

2.1.2. Pod 구성의 예

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

다음은 Pod 정의의 예입니다. 이 예제에서는 Pod의 다양한 기능을 보여줍니다. 대부분 다른 주제에서 설명하므로 여기에서는 간단히 언급합니다.

Pod 오브젝트 정의(YAML)

```
kind: Pod
apiVersion: v1
metadata:
  name: example
  labels:
    environment: production
    app: abc 1
spec:
  restartPolicy: Always 2
  securityContext: 3
    runAsNonRoot: true
    seccompProfile:
```

```
    type: RuntimeDefault
containers: 4
  - name: abc
    args:
    - sleep
    - "1000000"
    volumeMounts: 5
     - name: cache-volume
       mountPath: /cache 6
    image: registry.access.redhat.com/ubi7/ubi-init:latest 7
    securityContext:
      allowPrivilegeEscalation: false
      runAsNonRoot: true
      capabilities:
        drop: ["ALL"]
    resources:
      limits:
        memory: "100Mi"
        cpu: "1"
      requests:
        memory: "100Mi"
        cpu: "1"
volumes: 8
- name: cache-volume
  emptyDir:
    sizeLimit: 500Mi
```

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 Dedicated는 컨테이너에 대한 보안 컨텍스트를 정의하여 권한이 있는 컨테이너로 실행할 수 있는지, 선택한 사용자로 실행할 수 있는지 등을 지정합니다. 기본 컨텍스트는 매우 제한적이지만 필요에 따라 관리자가 수정할 수 있습니다.

4. **containers**는 하나 이상의 컨테이너 정의로 이루어진 배열을 지정합니다.

5. 이 컨테이너는 컨테이너 내에서 외부 스토리지 볼륨이 마운트되는 위치를 지정합니다.

6. Pod에 제공할 볼륨을 지정합니다. 지정된 경로에 볼륨을 마운트합니다. 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. Pod는 사용할 컨테이너에서 사용할 수 있는 스토리지 볼륨을 정의합니다.

   파일 수가 많은 영구 볼륨을 Pod에 연결하면 해당 Pod가 실패하거나 시작하는 데 시간이 오래 걸릴 수 있습니다. 자세한 내용은 OpenShift에서 파일 수가 많은 영구 볼륨을 사용하는 경우 Pod를 시작하지 못하거나 "Ready" 상태를 달성하는 데 과도한 시간이 걸리는 이유를 참조하십시오.

참고

이 pod 정의에는 Pod가 생성되고 라이프사이클이 시작된 후 OpenShift Dedicated에 의해 자동으로 채워지는 특성이 포함되지 않습니다. Kubernetes Pod 설명서 에는 Pod의 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 Pod를 보고 해당 Pod 및 클러스터의 상태를 전체적으로 확인할 수 있습니다.

2.2.1. Pod 정보

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

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 Pod를 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-594fcccf94-bcxk8    3m           18Mi
downloads-594fcccf94-kv4p6    2m           15Mi
```

2. 다음 명령을 실행하여 라벨이 있는 Pod의 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

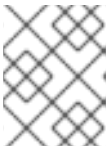
OpenShift CLI(**oc**) 및 웹 콘솔에서 다양한 리소스의 로그를 볼 수 있습니다. 로그는 로그의 말미 또는 끝에서 읽습니다.

사전 요구 사항

- OpenShift CLI(**oc**)에 액세스합니다.

프로세스(UI)

1. OpenShift Dedicated 콘솔에서 **워크로드 → Pod** 로 이동하거나 조사할 리소스를 통해 Pod로 이동합니다.



참고

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

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

프로세스(CLI)

- 특정 Pod의 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 OPENSHIFT DEDICATED 클러스터 구성

관리자는 Pod에 효율적인 클러스터를 생성하고 유지 관리할 수 있습니다.

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

2.3.1. 재시작 후 Pod 작동 방식 구성

Pod 재시작 정책에 따라 해당 Pod의 컨테이너가 종료될 때 OpenShift Dedicated가 응답하는 방식이 결정됩니다. 정책은 해당 Pod의 모든 컨테이너에 적용됩니다.

가능한 값은 다음과 같습니다.

- **Always** - 기하급수적 백오프 지연(10s, 20s, 40s)이 5분으로 제한되어 Pod에서 성공적으로 종료된 컨테이너를 지속적으로 재시작합니다. 기본값은 **Always**입니다.
- **OnFailure** - 급격한 백오프 지연(10초, 20초, 40초)을 5분으로 제한하여 Pod에서 실패한 컨테이너를 재시작합니다.
- **Never** - Pod에서 종료되거나 실패한 컨테이너를 재시작하지 않습니다. Pod가 즉시 실패하고 종료됩니다.

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

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

Pod의 컨테이너가 실패하고 재시작 정책이 **OnFailure**로 설정된 경우 Pod가 노드에 남아 있고 컨테이너가 재시작됩니다. 컨테이너를 재시작하지 않으려면 재시작 정책 **Never**를 사용하십시오.

전체 Pod가 실패하면 OpenShift Dedicated에서 새 Pod를 시작합니다. 개발자는 애플리케이션이 새 Pod에서 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 다운되면 kubelet에서 OpenShift Dedicated가 다시 시작되지 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 Dedicated에서 실패한 컨테이너와 함께 재시작 정책을 사용하는 방법에 대한 자세한 내용은 Kubernetes 문서의 예제 상태를 참조하십시오.

2.3.2. Pod에서 사용할 수 있는 대역폭 제한

Pod에 서비스 품질 트래픽 조절 기능을 적용하고 사용 가능한 대역폭을 효과적으로 제한할 수 있습니다. Pod에서 송신하는 트래픽은 구성된 속도를 초과하는 패킷을 간단히 삭제하는 정책에 따라 처리합니다. Pod에 수신되는 트래픽은 데이터를 효과적으로 처리하기 위해 대기 중인 패킷을 구성하여 처리합니다. 특정 Pod에 대한 제한 사항은 다른 Pod의 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/s로 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 Pod를 생성합니다.

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

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

*Pod 중단 예산*을 사용하면 유지보수를 위해 노드를 드레이닝하는 등 작업 중에 Pod에 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 pod에 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 조건이 **Ready=True** 인 Pod 수를 나타냅니다. **ready=True** 는 요청을 제공할 수 있는 Pod를 나타내며 일치하는 모든 서비스의 로드 밸런싱 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.

주의

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

다음을 사용하여 모든 프로젝트에서 pod 중단 예산을 확인할 수 있습니다.

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

참고

다음 예제에는 AWS의 OpenShift Dedicated와 관련된 몇 가지 값이 포함되어 있습니다.

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               1                     125m
openshift-cloud-credential-operator    pod-identity-webhook                1               N/A               1                     117m
```

```
openshift-cluster-csi-drivers          aws-ebs-csi-driver-controller-pdb     N/A         1             1
121m
openshift-cluster-storage-operator     csi-snapshot-controller-pdb           N/A         1             1
122m
openshift-cluster-storage-operator     csi-snapshot-webhook-pdb              N/A         1             1
122m
openshift-console                      console                               N/A         1             1
116m
#...
```

PodDisruptionBudget은 시스템에서 최소 **minAvailable** pod가 실행중인 경우 정상으로 간주됩니다. 이 제한을 초과하는 모든 pod는 제거할 수 있습니다.

참고

Pod 우선 순위 및 선점 설정에 따라 우선 순위가 낮은 pod는 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 수 입니다. 정수 또는 백분율 (예: **20 %**)을 지정하는 문자열을 사용할 수 있습니다.
3. 리소스 집합에 대한 라벨 쿼리입니다. **matchLabels** 및 **matchExpressions**의 결과는 논리적으로 결합됩니다. 프로젝트의 모든 포드를 선택하려면 이 매개변수(예: **selector {}**)를 비워 둡니다.

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 수없는 최대 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. 프로젝트의 모든 포드를 선택하려면 이 매개변수(예: **selector {}**)를 비워 둡니다.

2. 다음 명령을 실행하여 오브젝트를 프로젝트에 추가합니다.

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

2.4. 보안을 사용하여 POD에 민감한 데이터 제공

추가 리소스

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.

2.4.1. 보안 이해

Secret 오브젝트 유형은 암호, OpenShift Dedicated 클라이언트 구성 파일, 개인 소스 리포지토리 자격 증명 등과 같은 중요한 정보를 보유하는 메커니즘을 제공합니다. 보안은 Pod에서 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 용어집 의 **DNS_SUBDOMAIN** 값에 있는 지침을 충족해야 합니다.
3. **data** 맵의 키와 관련된 값은 base64로 인코딩되어야 합니다.
4. **stringData** 맵의 항목이 base64로 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 Pod를 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 Pod를 생성합니다.

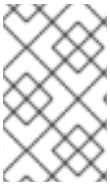
2.4.1.1. 보안 유형

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.

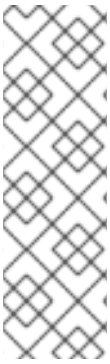
다양한 유형의 시크릿 생성 예를 보려면 시크릿을 *생성하는 방법 이해* 를 참조하십시오.

2.4.1.2. 보안 데이터 키

보안키는 DNS 하위 도메인에 있어야 합니다.

2.4.1.3. 자동으로 생성된 이미지 풀 시크릿

기본적으로 OpenShift Dedicated는 각 서비스 계정에 대한 이미지 풀 시크릿을 생성합니다.



참고

OpenShift Dedicated 4.16 이전에는 생성된 각 서비스 계정에 대해 장기 서비스 계정 API 토큰 시크릿도 생성되었습니다. OpenShift Dedicated 4.16부터 이 서비스 계정 API 토큰 시크릿은 더 이상 생성되지 않습니다.

4로 업그레이드한 후 장기 서비스 계정 API 토큰 시크릿은 삭제되지 않으며 계속 작동합니다. 클러스터에서 사용 중인 장기 API 토큰을 감지하거나 필요하지 않은 경우 삭제하는 방법에 대한 자세한 내용은 OpenShift Container Platform의 장기 서비스 계정 API 토큰을 참조하십시오.

이 이미지 풀 시크릿은 OpenShift 이미지 레지스트리를 클러스터의 사용자 인증 및 권한 부여 시스템에 통합하기 위해 필요합니다.

그러나 **ImageRegistry** 기능을 활성화하지 않거나 Cluster Image Registry Operator 구성에서 통합 OpenShift 이미지 레지스트리를 비활성화하면 각 서비스 계정에 대해 이미지 풀 시크릿이 생성되지 않습니다.

이전에 활성화된 클러스터에서 통합 OpenShift 이미지 레지스트리가 비활성화되면 이전에 생성된 이미지 가져오기 보안이 자동으로 삭제됩니다.

2.4.2. 보안 생성 방법 이해

관리자는 개발자가 해당 보안을 사용하는 Pod를 생성하기 전에 보안을 생성해야 합니다.

보안 생성 시 다음을 수행합니다.

1. 보안을 유지하려는 데이터가 포함된 보안 오브젝트를 생성합니다. 각 보안 유형에 필요한 특정 데이터는 다음 섹션에서 축소됩니다.

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. Pod의 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 Pod를 생성합니다.

보안 데이터로 볼륨의 파일을 채우는 Pod의 YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: secret-volume
```

```
    secret:
      secretName: test-secret 4
  restartPolicy: Never
```

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

2 시크릿을 표시할 사용하지 않는 디렉터리 이름을 지정합니다. 시크릿 데이터 맵의 각 키는 **mountPath** 아래의 파일 이름이 됩니다.

3 **true** 로 설정합니다. true인 경우 드라이버에 읽기 전용 볼륨을 제공하도록 지시합니다.

4 보안 이름을 지정합니다.

보안 데이터로 환경 변수를 채우는 Pod의 YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  restartPolicy: Never
```

1 시크릿 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: (1)
          name: test-secret
          key: username
    from:
      kind: ImageStreamTag
      namespace: openshift
      name: 'cli:latest'
```

(1) 시크릿 키를 사용하는 환경 변수를 지정합니다.

2.4.2.1. 보안 생성 제한 사항

보안을 사용하려면 Pod에서 보안을 참조해야 합니다. 보안은 다음 세 가지 방법으로 Pod에서 사용할 수 있습니다.

- 컨테이너에 환경 변수를 채우기 위해 사용.
- 하나 이상의 컨테이너에 마운트된 볼륨에서 파일로 사용.
- Pod에 대한 이미지를 가져올 때 kubelet으로 사용.

볼륨 유형 보안은 볼륨 메커니즘을 사용하여 데이터를 컨테이너에 파일로 작성합니다. 이미지 가져오기 보안은 서비스 계정을 사용하여 네임스페이스의 모든 Pod에 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 Pod를 생성해야 합니다. 가장 효과적인 방법은 서비스 계정을 사용하여 자동으로 삽입되도록 하는 것입니다.

Secret API 오브젝트는 네임스페이스에 있습니다. 동일한 네임스페이스에 있는 Pod만 참조할 수 있습니다.

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

2.4.2.2. 불투명 보안 생성

관리자는 임의의 값을 포함할 수 있는 구조화되지 않은 **키:값** 쌍을 저장할 수 있는 opaque 보안을 생성할 수 있습니다.

프로세스

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. "보안을 생성하는 방법 이해" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

   b. "보안 생성 방법 이해" 섹션에 표시된 대로 **보안** 을 환경 변수 또는 파일로 사용하는 Pod를 생성합니다.

추가 리소스

- 보안 생성 방법 이해

2.4.2.3. 레거시 서비스 계정 토큰 시크릿 생성

관리자는 레거시 서비스 계정 토큰 시크릿을 생성할 수 있으므로 API에 인증해야 하는 애플리케이션에 서비스 계정 토큰을 배포할 수 있습니다.

주의

기존 서비스 계정 토큰 시크릿을 사용하는 대신 TokenRequest API를 사용하여 바인딩된 서비스 계정 토큰을 가져오는 것이 좋습니다. TokenRequest API를 사용할 수 없고 읽을 수 있는 API 오브젝트에서 만료되지 않은 토큰의 보안 노출이 허용되는 경우에만 서비스 계정 토큰 시크릿을 생성해야 합니다.

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.

바인딩된 서비스 계정 토큰을 얻기 위해 예상 볼륨과 함께 워크로드가 자동으로 삽입됩니다. 워크로드에 추가 서비스 계정 토큰이 필요한 경우 워크로드 매니페스트에 예상 볼륨을 추가합니다.

자세한 내용은 "볼륨 프로젝션을 사용하여 바인딩된 서비스 계정 토큰 구성"을 참조하십시오.

프로세스

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

Secret 오브젝트의 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 시크릿을 지정합니다.

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

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

3. Pod에서 보안을 사용하려면 다음을 수행합니다.

   a. "보안을 생성하는 방법 이해" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

   b. "보안 생성 방법 이해" 섹션에 표시된 대로 **보안** 을 환경 변수 또는 파일로 사용하는 Pod를 생성합니다.

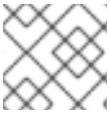
추가 리소스

- 보안 생성 방법 이해

2.4.2.4. 기본 인증 보안 생성

관리자는 기본 인증에 필요한 인증 정보를 저장할 수 있는 기본 인증 보안을 생성할 수 있습니다. 이 보안 유형을 사용하는 경우 **Secret** 오브젝트의 **data** 매개변수에는 base64 형식으로 인코딩된 다음 키가 포함되어야 합니다.

- **username**: 인증을 위한 사용자 이름
- **암호**: 인증을 위한 암호 또는 토큰



참고

stringData 매개변수를 사용하여 일반 텍스트 콘텐츠를 사용할 수 있습니다.

프로세스

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

보안 오브젝트의 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. "보안을 생성하는 방법 이해" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

   b. "보안 생성 방법 이해" 섹션에 표시된 대로 **보안** 을 환경 변수 또는 파일로 사용하는 Pod를 생성합니다.

추가 리소스

- 보안 생성 방법 이해

2.4.2.5. SSH 인증 보안 생성

관리자는 SSH 인증에 사용되는 데이터를 저장할 수 있는 SSH 인증 보안을 생성할 수 있습니다. 이 보안 유형을 사용하는 경우 **Secret** 오브젝트의 **data** 매개변수에 사용할 SSH 인증 정보가 포함되어야 합니다.

프로세스

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

   보안 오브젝트의 예

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

(1) SSH 인증 보안을 지정합니다.

(2) 사용할 SSH 자격 증명으로 SSH 키/값 쌍을 지정합니다.

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

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

3. Pod에서 보안을 사용하려면 다음을 수행합니다.

   a. "보안을 생성하는 방법 이해" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

   b. "보안 생성 방법 이해" 섹션에 표시된 대로 **보안** 을 환경 변수 또는 파일로 사용하는 Pod를 생성합니다.

추가 리소스

- 보안 생성 방법 이해

2.4.2.6. Docker 구성 보안 생성

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

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

프로세스

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

Docker 구성 시크릿 오브젝트의 예

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

.dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cgYXV
0aCBrZXIzCg== 2
```

1 보안이 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:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXIzCg== (2)
```

(1) 보안이 Docker 구성 JSON 파일을 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. "보안을 생성하는 방법 이해" 섹션에 표시된 대로 Pod의 서비스 계정을 업데이트하여 보안을 참조합니다.

   b. "보안 생성 방법 이해" 섹션에 표시된 대로 **보안** 을 환경 변수 또는 파일로 사용하는 Pod를 생성합니다.

추가 리소스

- 보안 생성 방법 이해

2.4.2.7. 웹 콘솔을 사용하여 보안 생성

웹 콘솔을 사용하여 보안을 생성할 수 있습니다.

프로세스

1. **워크로드 → 시크릿**으로 이동합니다.
2. **생성 → YAML** 을 클릭합니다.
   a. 사양에 맞게 YAML을 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 base64로 변환된 후 해당 항목이 자동으로 **data** 맵으로 이동합니다. 이 필드는 쓰기 전용이며 값은 **data** 필드를 통해서만 반환됩니다.

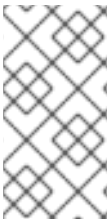
3. **생성**을 클릭합니다.
4. **워크로드에 시크릿 추가**를 클릭합니다.
   a. 드롭다운 메뉴에서 추가할 워크로드를 선택합니다.
   b. **저장**을 클릭합니다.

2.4.3. 보안 업데이트 방법 이해

보안 값을 수정해도 이미 실행 중인 Pod에서 사용하는 값은 동적으로 변경되지 않습니다. 보안을 변경하려면 원래 Pod를 삭제하고 새 Pod를 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** 값은 참조 시 지정되지 않습니다. 따라서 Pod가 시작되는 동시에 보안이 업데이트되는 경우 Pod에 사용되는 보안의 버전이 정의되지 않습니다.



참고

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

2.4.4. 보안 생성 및 사용

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

프로세스

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> 1
  annotations:
    kubernetes.io/service-account.name: "sa-name" 2
type: kubernetes.io/service-account-token 3
```

-

1. & **lt;secret_name&** gt;을 서비스 토큰 시크릿의 이름으로 바꿉니다.
2. 기존 서비스 계정 이름을 지정합니다. **ServiceAccount** 및 **Secret** 오브젝트를 모두 생성하는 경우 **ServiceAccount** 오브젝트를 먼저 생성합니다.
3. 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 1
```

출력 예

```
ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpoNnh5YnZhSE9QMkk3YnRZMVZoclFf
QTZfRFp1YlUifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5
pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZX
J2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImJ1aWxkZXItdG9rZW4tdHZrbnIiLCJrdWJlcm5l
dGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiYnVpbGRlciIsImt1
YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjNmZGU
2MGZmLTA1NGYtNDkyZi04YzhjLTNlZjE0NDk3MmFmNyIsInN1YiI6InN5c3RlbTpzZXJ2aWNl
YWNjb3VudDpkZWZhdWx0OmJ1aWxkZXIifQ.OmqFTDuMHC_lYvvEUrjr1x453hlEEHYcxS9VK
SzmRkP1SiVZWPNPkTWlfNRp6bIUZD3U6aN3N7dMSN0eI5hu36xPgpKTdvuckKLTCnelMx6c
xOdAbrcw1mCmOClNscwjS1KO1kzMtYnnq8rXHiMJELsNlhnRyyIXRTtNBsy4t64T3283s3SLsa
ncyx0gy0ujx-Ch3uKAKdZi5iT-I8jnnQ-ds5THDs2h65RJhgglQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnvi3DGn2XI-
hZxI1yD2yGH1RBpYUHA
```

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. &lt **;openshift_cluster_api&gt**;를 OpenShift 클러스터 API로 바꿉니다.
2. & **lt;token** >을 이전 명령에서 출력하는 서비스 계정 토큰으로 바꿉니다.

2.4.5. 보안이 포함된 서명된 인증서 사용 정보

서비스에 대한 통신을 보호하려면 프로젝트의 보안에 추가할 수 있는 서명된 제공 인증서/키 쌍을 생성하도록 OpenShift Dedicated를 구성할 수 있습니다.

*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.

2.4.5.1. 보안과 함께 사용할 서명된 인증서 생성

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

프로세스

*서비스 제공 인증서 보안*을 생성하려면 다음을 수행합니다.

1. 서비스에 대한 **Pod** 사양을 편집합니다.
2. 보안에 사용할 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
```

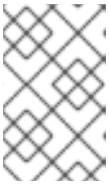
5. 해당 보안을 사용하여 **Pod** 사양을 편집합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: my-container
      mountPath: "/etc/my-path"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  volumes:
  - name: my-volume
    secret:
      secretName: my-cert
```

```
items:
- key: username
  path: my-group/my-username
  mode: 511
```

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

인증서/키 쌍은 만료 시기가 다가오면 자동으로 교체됩니다. 시크릿의 **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.

2.4.6. 보안 문제 해결

서비스 인증서 생성에 실패하는 경우(서비스의 **service.beta.openshift.io/serving-cert-generation-error** 주석에는 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 서비스 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.

2.5. 구성 맵 생성 및 사용

다음 섹션에서는 구성 맵과 이를 생성하고 사용하는 방법을 정의합니다.

2.5.1. 구성 맵 이해

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

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

ConfigMap 오브젝트에는 Pod에서 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 64에 파일 데이터를 입력합니다.

참고

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

다양한 방법으로 Pod에서 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 Pod에서만 참조할 수 있습니다.

Kubelet은 API 서버에서 가져오는 Pod에 대한 구성 맵만 지원합니다.

여기에는 CLI를 사용하거나 복제 컨트롤러에서 간접적으로 생성되는 모든 Pod가 포함됩니다. OpenShift Dedicated 노드의 **--manifest-url** 플래그, **--config** 플래그 또는 해당 REST API를 사용하여 생성된 Pod는 Pod를 생성하는 일반적인 방법이 아니므로 포함하지 않습니다.

2.5.2. OpenShift Dedicated 웹 콘솔에서 구성 맵 생성

OpenShift Dedicated 웹 콘솔에서 구성 맵을 생성할 수 있습니다.

프로세스

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

2.5.3. 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]
```

2.5.3.1. 디렉토리에서 구성 맵 생성

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

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

예를 들어 다음 명령은 **example-files** 디렉터리의 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=UUDDLRLRBABAS
  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
  ```

프로세스

- 다음 명령을 입력하여 이 디렉터리에 있는 각 파일의 콘텐츠를 포함하는 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

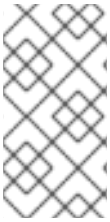
```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    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
```

2.5.3.2. 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 UTF8이 아닌 데이터를 손상시키지 않고 이 필드에 배치된 UTF8이 아닌 데이터가 포함된 파일을 포함할 수 있습니다. OpenShift Dedicated는 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=UUDDLRLRBABAS
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
```

프로세스

- 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=UUDDLRLRBABAS
    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.

2.5.3.3. 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.

프로세스

- 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
```

2.5.4. 사용 사례: Pod에서 구성 맵 사용

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

2.5.4.1. 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 Pod에서만 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  restartPolicy: Never
```

1 **ConfigMap**에서 지정된 환경 변수를 가져오는 스탠자입니다.

2 키 값을 삽입하는 pod 환경 변수의 이름입니다.

3 5 특정 환경 변수를 끌어올 **ConfigMap**의 이름입니다.

4 6 **ConfigMap**에서 가져올 환경 변수입니다.

7 환경 변수를 선택적으로 만듭니다. 선택 사항으로 지정된 **ConfigMap** 및 키가 없는 경우에도 Pod가 시작됩니다.

8 **ConfigMap**에서 모든 환경 변수를 가져오는 스탠자입니다.

9 모든 환경 변수를 가져올 **ConfigMap**의 이름입니다.

이 Pod가 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.

2.5.4.2. 구성 맵을 사용하여 컨테이너 명령의 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
```

프로세스

- 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  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
```

2.5.4.3. 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
```

프로세스

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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: config-volume
      configMap:
        name: special-config 1
  restartPolicy: Never
```

1 키가 포함된 파일입니다.

이 Pod가 실행되면 cat 명령의 출력은 다음과 같습니다.

```
very
```

- 구성 맵 키가 프로젝션된 볼륨 내의 경로를 제어할 수도 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
```

```
    type: RuntimeDefault
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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
volumes:
  - name: config-volume
    configMap:
      name: special-config
      items:
      - key: special.how
        path: path/to/special-key (1)
restartPolicy: Never
```

(1) 구성 맵 키 경로입니다.

이 Pod가 실행되면 cat 명령의 출력은 다음과 같습니다.

```
very
```

2.6. POD 예약 결정에 POD 우선순위 포함

클러스터에서 Pod 우선순위 및 선점을 활성화할 수 있습니다. Pod 우선순위는 다른 Pod와 관련된 Pod의 중요도를 나타내며 해당 우선 순위에 따라 Pod를 대기열에 넣습니다. Pod 선점을 사용하면 적절한 노드 Pod 우선순위에 사용 가능한 공간이 없는 경우 우선 순위가 낮은 Pod를 제거하거나 우선순위가 낮은 Pod를 예약할 수 있습니다. 노드의 예약 순서 및 리소스 부족 제거 순서에도 영향을 미칩니다.

우선순위 및 선점 기능을 사용하려면 Pod 사양의 우선순위 클래스를 참조하여 스케줄링에 해당 가중치를 적용합니다.

2.6.1. Pod 우선순위 이해

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

2.6.1.1. Pod 우선순위 클래스

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

우선순위 클래스 오브젝트에는 1000000000(10억)보다 작거나 같은 32비트 정수 값을 사용할 수 있습니다. 선점 또는 제거해서는 안 되는 중요한 Pod의 경우 10억보다 크거나 같은 수를 예약합니다. 기본적으로 OpenShift Dedicated에는 중요한 시스템 Pod의 스케줄링을 보장하기 위해 두 가지 우선순위 클래스가 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 Pod에 사용합니다. 이 우선순위 클래스가 있는 Pod의 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 구성한 Pod가 우선할 수 있습니다. 그러나 이 우선순위 클래스는 예약을 보장합니다. 이 우선순위 클래스를 사용할 수 있는 Pod의 예로는 fluentd, Descheduler와 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 - **priorityClassName** 필드를 리소스 사용을 바인딩할 수 없고 예측 가능한 리소스 사용량 동작이 없는 중요한 Pod와 함께 사용할 수 있습니다. **openshift-monitoring** 및 **openshift-user-workload-monitoring** 네임스페이스 아래의 Prometheus Pod는 **openshift-user-critical priorityClassName** 을 사용합니다. 모니터링 워크로드는 첫 번째 **priorityClass** 로 **시스템 중요** 를 사용하지만 모니터링에서 과도한 메모리를 사용하고 노드를 제거할 수 없는 경우 문제가 발생합니다. 결과적으로 모니터링은 중요한 노드 작동을 유지하기 위해 스케줄러의 유연성을 제공하기 위해 우선 순위가 떨어지고 많은 워크로드를 이동합니다.
- **cluster-logging** - 이 우선순위는 Fluentd에서 Fluentd Pod가 다른 앱보다 먼저 노드에 예약되도록 하는 데 사용합니다.

2.6.1.2. Pod 우선순위 이름

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

2.6.2. Pod 선점 이해

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

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

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

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

선점을 수행해도 우선순위가 낮은 Pod가 노드에서 모두 제거되는 것은 아닙니다. 스케줄러는 우선순위가 낮은 Pod의 일부를 제거하여 보류 중인 Pod를 예약할 수 있습니다.

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

2.6.2.1. 선점되지 않은 우선 순위 클래스

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

비 선점 Pod는 우선순위가 높은 다른 Pod에서 계속 선점할 수 있습니다.

2.6.2.2. 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 Pod를 최상의 노력 수준에서 선점할 때 OpenShift Dedicated에서 해당 예산을 준수합니다. 스케줄러는 Pod 중단 예산을 위반하지 않고 Pod를 선점하려고 합니다. 이러한 Pod를 찾을 수 없는 경우 Pod 중단 예산 요구 사항과 관계없이 우선순위가 낮은 Pod를 선점할 수 있습니다.

Pod 우선순위 및 Pod 유사성

Pod 유사성을 위해서는 동일한 라벨이 있는 다른 Pod와 같은 노드에서 새 Pod를 예약해야 합니다.

노드에서 우선순위가 낮은 하나 이상의 Pod와 보류 중인 Pod에 Pod 간 유사성이 있는 경우 스케줄러는 선호도 요구 사항을 위반하지 않고 우선순위가 낮은 Pod를 선점할 수 없습니다. 이 경우 스케줄러는 보류

중인 Pod를 예약할 다른 노드를 찾습니다. 그러나 스케줄러에서 적절한 노드를 찾을 수 있다는 보장이 없고 보류 중인 Pod가 예약되지 않을 수 있습니다.

이러한 상황을 방지하려면 우선순위가 같은 Pod를 사용하여 Pod 유사성을 신중하게 구성합니다.

2.6.2.3. 선점된 Pod의 정상 종료

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

이 간격을 최소화하려면 우선순위가 낮은 Pod의 정상 종료 기간을 짧게 구성하십시오.

2.6.3. 우선순위 및 선점 구성

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

참고

예약된 기존 Pod에 우선순위 클래스를 직접 추가할 수 없습니다.

프로세스

우선순위 및 선점을 사용하도록 클러스터를 구성하려면 다음을 수행합니다.

1. 다음과 유사한 YAML 파일을 생성하여 우선순위 클래스의 이름을 포함하도록 Pod 사양을 정의합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: system-cluster-critical (1)
```

(1) 이 Pod에 사용할 우선순위 클래스를 지정합니다.

2. Pod를 생성합니다.

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

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

2.7. 노드 선택기를 사용하여 특정 노드에 POD 배치

*노드 선택기*는 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.

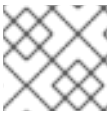
동일한 Pod 구성에서 노드 유사성 및 노드 선택기를 사용하는 경우 아래의 중요 고려 사항을 참조하십시오.

2.7.1. 노드 선택기를 사용하여 Pod 배치 제어

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

노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 시스템 세트에 레이블을 추가하면 노드 또는 머신이 중단되면 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
# ...
```

프로세스

- Pod에 일치하는 노드 선택기를 추가합니다.
  - 기존 및 향후 Pod에 노드 선택기를 추가하려면 Pod의 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 Pod에 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.

3장. CUSTOM METRICS AUTOSCALER OPERATOR를 사용하여 POD 자동 스케일링

3.1. 릴리스 노트

3.1.1. 사용자 정의 Metrics Autoscaler Operator 릴리스 노트

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 이벤트 기반 자동 스케일러(KEDA)를 사용하며 OpenShift Dedicated HPA(Horizontal Pod Autoscaler) 상단에 빌드됩니다.

참고

Custom Metrics Autoscaler Operator for Red Hat OpenShift는 코어 OpenShift Dedicated와 별도의 릴리스 사이클과 함께 설치 가능한 구성 요소로 제공됩니다. Red Hat OpenShift Container Platform 라이프 사이클 정책은 릴리스 호환성에 대해 간단히 설명합니다.

3.1.1.1. 지원되는 버전

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

버전	OpenShift Dedicated 버전	정식 출시일 (GA)
2.14.1	4.16	정식 출시일 (GA)
2.14.1	4.15	정식 출시일 (GA)
2.14.1	4.14	정식 출시일 (GA)
2.14.1	4.13	정식 출시일 (GA)
2.14.1	4.12	정식 출시일 (GA)

3.1.1.2. 사용자 정의 지표 Autoscaler Operator 2.14.1 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.14.1-454 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 CVE, 새로운 기능 및 버그 수정을 제공합니다. RHBA-2024:5865 에 대해 다음 권고를 사용할 수 있습니다.

중요

이 버전의 Custom Metrics Autoscaler Operator를 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 Cron 트리거 지원

Custom Metrics Autoscaler Operator는 Cron 트리거를 사용하여 시간별 스케줄에 따라 Pod를 스케일링할 수 있습니다. 지정된 시간 프레임이 시작되면 Custom Metrics Autoscaler Operator는 Pod를 원하는 양으로 스케일링합니다. 시간 프레임이 종료되면 Operator는 이전 수준으로 다시 축소됩니다.

자세한 내용은 Cron 트리거 이해를 참조하십시오.

3.1.1.2.2. 버그 수정

- 이전에는 **KedaController** 사용자 정의 리소스에서 구성 매개변수를 감사하는 경우 **keda-metrics-server-audit-policy** 구성 맵이 업데이트되지 않았습니다. 결과적으로 Custom Metrics Autoscaler의 초기 배포 후 감사 구성 매개변수를 변경할 수 없었습니다. 이번 수정을 통해 구성 맵에서 감사 구성을 올바르게 렌더링하여 설치 후 언제든지 감사 구성을 변경할 수 있습니다. (OCPBUGS-32521)

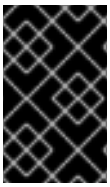
3.1.2. Custom Metrics Autoscaler Operator의 이전 릴리스 릴리스 노트

다음 릴리스 노트는 이전 버전의 Custom Metrics Autoscaler Operator에 대한 것입니다.

현재 버전의 경우 Custom Metrics Autoscaler Operator 릴리스 노트를 참조하십시오.

3.1.2.1. 사용자 정의 지표 Autoscaler Operator 2.13.1 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.13.1-421 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. RHBA-2024:4837 에 대해 다음 권고를 사용할 수 있습니다.



중요

이 버전의 Custom Metrics Autoscaler Operator를 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. 새로운 기능 및 개선 사항

3.1.2.1.1.1. 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 Operator는 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 .

3.1.2.1.2. 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)

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

이번 Custom Metrics Autoscaler Operator 2.12.1-394 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 버그 수정을 제공합니다. RHSA-2024:2901 에 대해 다음 권고를 사용할 수 있습니다.

중요

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

3.1.2.2.1. 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. (OCPBUGS-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. (OCPBUGS-30360)

- 이전 버전에서는 HTTP 리디렉션을 따르는 경우 일치하는 하위 도메인에 없거나 초기 도메인의 정확한 일치에 따라 HTTP 클라이언트가 **Authorization** 또는 Cryostat와 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. (OCPBUGS-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. (OCPBUGS-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. (OCPBUGS-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)

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

이번 Custom Metrics Autoscaler Operator 2.12.1-384 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 버그 수정을 제공합니다. RHBA-2024:2043 에 대해 다음 권고를 사용할 수 있습니다.

중요

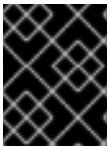
이 버전의 Custom Metrics Autoscaler Operator를 설치하기 전에 이전에 설치한 기술 프리뷰 버전 또는 커뮤니티 지원 KEDA 버전을 제거합니다.

3.1.2.3.1. 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)

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

이번 Custom Metrics Autoscaler Operator 2.12.1-376 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 보안 업데이트 및 버그 수정을 제공합니다. RHSA-2024:1812 에 대해 다음 권고를 사용할 수 있습니다.



중요

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

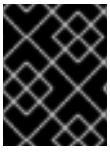
3.1.2.4.1. 버그 수정

- 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 CR에서 **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 CR이 수정되었습니다**. 결과적으로 서비스 모니터가 제대로 작동합니다. (OCPBUGS-25806)

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

이번 Custom Metrics Autoscaler Operator 2.11.2-322 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 보안 업데이트 및 버그 수정을 제공합니다. RHSA-2023:6144 에 대해 다음 권고를 사용할 수 있습니다.



중요

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

3.1.2.5.1. 버그 수정

- Custom Metrics Autoscaler Operator 버전 3.11.2-311이 Operator 배포에 필요한 볼륨 마운트 없이 릴리스되었으므로 Custom Metrics Autoscaler Operator Pod가 15분마다 다시 시작됩니다. 이번 수정에서는 Operator 배포에 필요한 볼륨 마운트가 추가되었습니다. 결과적으로 Operator가 15분마다 더 이상 재시작되지 않습니다. (OCPBUGS-22361)

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

이번 Custom Metrics Autoscaler Operator 2.11.2-311 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. Custom Metrics Autoscaler Operator 2.11.2-311의 구성 요소는 RHBA-2023:5981 에서 릴리스되었습니다.

중요

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

3.1.2.6.1. 새로운 기능 및 개선 사항

3.1.2.6.1.1. 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 Operator는 **openshift-keda** 네임스페이스에만 설치할 수 있었습니다. 이로 인해 OpenShift ROSA 및 OpenShift Dedicated 클러스터에 Operator가 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.

3.1.2.6.2. 버그 수정

- 이전 버전에서는 Custom Metrics Autoscaler Operator를 설치 및 구성했지만 사용하지 않은 경우 OpenShift CLI에서 **oc** 명령을 입력한 후 **external.metrics.k8s.io/v1beta1에 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. (OCPBUGS-15779)

- 이전에는 Custom Metrics Autoscaler에서 관리하는 오브젝트에 대한 주석 또는 레이블 변경이 Keda Controller를 수정할 때마다 Custom Metrics Autoscaler Operator에 의해 취소되었습니다 (예: 구성 변경 후). 이로 인해 오브젝트에서 라벨이 지속적으로 변경되었습니다. Custom Metrics Autoscaler는 이제 자체 주석을 사용하여 레이블 및 주석을 관리하고 주석 또는 레이블을 더 이상 잘못 되돌리지 않습니다. (OCPBUGS-15590)

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

이번 Custom Metrics Autoscaler Operator 2.10.1-267 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. Custom Metrics Autoscaler Operator 2.10.1-267의 구성 요소는 RHBA-2023:4089 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. 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. (OCPBUGS-15264)

- 이전에는 Custom Metrics Autoscaler Operator에서 다른 네임스페이스 및 클러스터 범위 오브젝트의 오브젝트를 포함하여 모든 관리 오브젝트의 소유권을 시도했습니다. 이로 인해 Custom Metrics Autoscaler Operator에서 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 Operator는 **ownerReference** 필드를 다른 네임스페이스 또는 클러스터 범위 오브젝트의 오브젝트에 추가하는 것을 건너뜁니다. 결과적으로 이제 오류 없이 역할 바인딩이 생성됩니다. (OCPBUGS-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. (OCPBUGS-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. (OCPBUGS-15274)

3.1.2.8. 사용자 정의 Metrics Autoscaler Operator 2.10.1 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.10.1 릴리스는 OpenShift Dedicated 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.

3.1.2.8.1. 새로운 기능 및 개선 사항

3.1.2.8.1.1. 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 범위를 참조하십시오.

3.1.2.8.1.2. 성능 지표

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

3.1.2.8.1.3. 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.

3.1.2.8.1.4. 확장 오브젝트의 복제본 대체

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

3.1.2.8.1.5. 확장된 오브젝트에 대해 사용자 정의 가능한 HPA 이름 지정

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

3.1.2.8.1.6. 활성화 및 스케일링 임계값

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

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

이번 Custom Metrics Autoscaler Operator 2.8.2-174 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. Custom Metrics Autoscaler Operator 2.8.2-174의 구성 요소는 RHEA-2023:1683 에서 릴리스되었습니다.

중요

Custom Metrics Autoscaler Operator 버전 2.8.2-174는 기술 프리뷰 기능입니다.

3.1.2.9.1. 새로운 기능 및 개선 사항

3.1.2.9.1.1. Operator 업그레이드 지원

이제 이전 버전의 Custom Metrics Autoscaler Operator에서 업그레이드할 수 있습니다. Operator 업그레이드에 대한 자세한 내용은 "Operator의 업데이트 채널 변경"에서 "추가 리소스"를 참조하십시오.

3.1.2.9.1.2. must-gather 지원

OpenShift Dedicated **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 Operator의 경우와 다릅니다. 자세한 내용은 "추가 리소스"에서 "추가 리소스"의 디버깅 데이터를 참조하십시오.

3.1.2.10. 사용자 정의 지표 Autoscaler Operator 2.8.2 릴리스 노트

이번 Custom Metrics Autoscaler Operator 2.8.2 릴리스는 OpenShift Dedicated 클러스터에서 Operator를 실행하기 위한 새로운 기능 및 버그 수정을 제공합니다. Custom Metrics Autoscaler Operator 2.8.2의 구성 요소는 RHSA-2023:1042 에서 릴리스되었습니다.

중요

Custom Metrics Autoscaler Operator 버전 2.8.2는 기술 프리뷰 기능입니다.

3.1.2.10.1. 새로운 기능 및 개선 사항

3.1.2.10.1.1. 감사 로깅

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

3.1.2.10.1.2. Apache Kafka 메트릭을 기반으로 애플리케이션 스케일링

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

3.1.2.10.1.3. CPU 메트릭을 기반으로 애플리케이션 스케일링

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

3.1.2.10.1.4. 메모리 메트릭을 기반으로 애플리케이션 확장

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

3.2. 사용자 정의 METRICS AUTOSCALER OPERATOR 개요

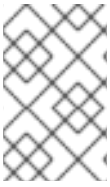
개발자는 Red Hat OpenShift용 Custom Metrics Autoscaler Operator를 사용하여 CPU 또는 메모리를 기반으로 하지 않는 사용자 정의 메트릭을 기반으로 OpenShift Dedicated에서 배포, 상태 저장 세트, 사용자 정의 리소스 또는 작업의 Pod 수를 자동으로 늘리거나 줄이는 방법을 지정할 수 있습니다.

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

사용자 정의 지표 자동 스케일러는 현재 Prometheus, CPU, 메모리 및 Apache Kafka 지표만 지원합니다.

Custom Metrics Autoscaler Operator는 특정 애플리케이션의 사용자 지정 외부 메트릭을 기반으로 Pod를 확장 및 축소합니다. 다른 애플리케이션에서는 다른 확장 방법을 계속 사용합니다. 사용자 정의 메트릭 자동 스케일러가 확장 방법을 결정하는 데 사용하는 이벤트 및 메트릭의 소스인 스케일러라고도 하는 *트리거*를 구성합니다. 사용자 정의 메트릭 자동 스케일러는 메트릭 API를 사용하여 OpenShift Dedicated에서 사용할 수 있는 양식으로 외부 메트릭을 변환합니다. 사용자 정의 지표 자동 스케일러는 실제 스케일링을 수행하는 HPA(수평 Pod 자동 스케일러)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), 허용된 최소 및 최대 복제본 수와 같은 기타 매개변수를 지정합니다.



참고

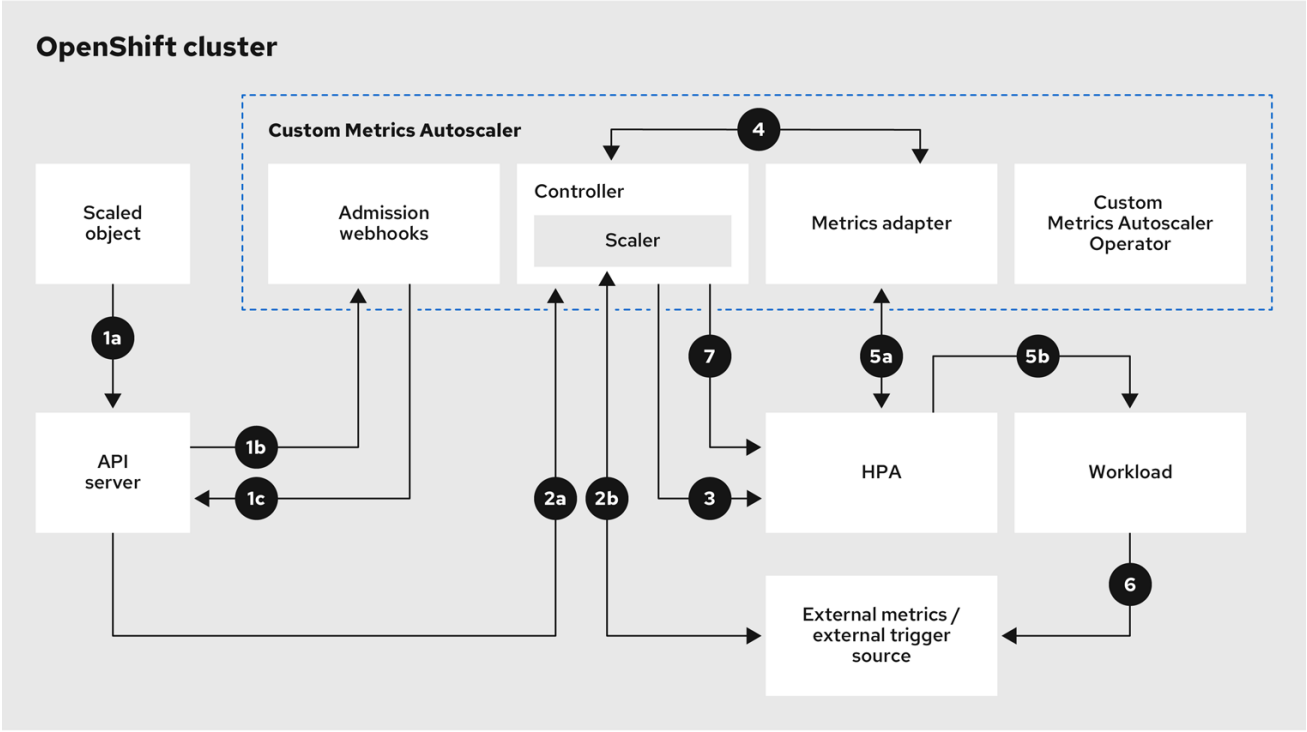
스케일링할 각 워크로드에 대해 하나의 확장 오브젝트 또는 스케일링된 작업만 생성할 수 있습니다. 또한 확장 오브젝트 또는 확장 작업 및 동일한 워크로드에서 HPA(수평 Pod 자동 스케일러)를 사용할 수 없습니다.

HPA와 달리 사용자 정의 지표 자동 스케일러는 0으로 확장할 수 있습니다. 사용자 정의 지표 자동 스케일러 CR에서 **minReplicaCount** 값을 **0** 으로 설정하면 사용자 정의 메트릭 자동 스케일러가 1에서 복제본 0 또는 0 복제본으로 또는 최대 0개의 복제본으로 워크로드를 축소합니다. 이를 *활성화 단계* 라고 합니다. 최대 1개의 복제본을 확장한 후 HPA는 스케일링을 제어합니다. 이를 *스케일링 단계* 라고 합니다.

일부 트리거를 사용하면 클러스터 지표 자동 스케일러에서 스케일링하는 복제본 수를 변경할 수 있습니다. 모든 경우에 활성화 단계를 구성하는 매개 변수는 항상 동일한 문구를 사용하며 *활성화* 접두사가 붙습니다. 예를 들어 **threshold** 매개변수가 스케일링을 구성하는 경우 **activationThreshold** 는 활성화를 구성합니다. 활성화 및 스케일링 단계를 구성하면 확장 정책에 대한 유연성을 높일 수 있습니다. 예를 들어 메트릭이 특히 낮은 경우 확장 또는 축소되지 않도록 더 높은 활성화 단계를 구성할 수 있습니다.

활성화 값은 각각에 대해 서로 다른 의사 결정의 경우 스케일링 값보다 우선 순위가 높습니다. 예를 들어 **임계값** 이 **10** 으로 설정되고 **activationThreshold** 가 **50** 개인 경우 지표에서 **40** 을 보고하는 경우 scaler는 활성 상태가 아니며 HPA에 4개의 인스턴스가 필요한 경우에도 Pod는 0으로 확장됩니다.

그림 3.1. 사용자 정의 메트릭 자동 스케일러 워크플로



1. 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) Operator가 트리거와 연결된 스케일링 데이터 모니터링을 시작합니다. HPA는 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 scaler에서 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 HPA에서 워크로드를 확장할 수 있습니다. 스케일링은 비활성화되어 있지만 HPA는 작업을 수행하지 않습니다.

3.2.1. 사용자 정의 지표 자동 스케일러의 사용자 정의 CA 인증서

기본적으로 Custom Metrics Autoscaler Operator는 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. Operator는 시작 시 해당 인증서를 로드하고 Operator에서 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-*** Pod를 다시 시작해야 합니다.

3.3. 사용자 정의 메트릭 자동 스케일러 설치

OpenShift Dedicated 웹 콘솔을 사용하여 Custom Metrics Autoscaler Operator를 설치할 수 있습니다.

설치 시 다음 5개의 CRD가 생성됩니다.

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

3.3.1. 사용자 정의 메트릭 자동 스케일러 설치

다음 절차를 사용하여 Custom Metrics Autoscaler Operator를 설치할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

OpenShift Dedicated 클러스터가 Red Hat(CCS 외)이 소유한 클라우드 계정에 있는 경우 **cluster-admin** 권한을 요청해야 합니다.

- Cluster Metrics Autoscaler Operator의 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
```

- **keda** 네임스페이스가 있는지 확인합니다. 그렇지 않은 경우 **keda** 네임스페이스를 생성해야 합니다.
- 선택 사항: 외부 Kafka 클러스터 또는 외부 Prometheus 서비스와 같은 클러스터 외부 서비스에 연결하는 데 Custom Metrics Autoscaler Operator가 필요한 경우 필요한 서비스 CA 인증서를 구성 맵에 배치합니다. 구성 맵은 Operator가 설치된 동일한 네임스페이스에 있어야 합니다. 예를 들면 다음과 같습니다.

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

프로세스

1. OpenShift Dedicated 웹 콘솔에서 **Operator → OperatorHub** 를 클릭합니다.
2. 사용 가능한 Operator 목록에서 **사용자 정의 지표 자동** 스케일러를 선택하고 **설치를** 클릭합니다.
3. **Operator 설치 페이지에서 설치** 모드에 **대해 클러스터 옵션의 특정 네임스페이스**가 선택되어 있는지 **확인합니다**.
4. **설치된 네임스페이스** 의 경우 **네임스페이스 선택을** 클릭합니다.
5. **프로젝트 선택을** 클릭합니다.
   - **keda** 네임스페이스가 있는 경우 목록에서 **keda** 를 선택합니다.
   - **keda** 네임스페이스가 없는 경우:
     a. **Create Project** 를 선택하여 **Create Project** 창을 엽니다.
     b. **이름** 필드에 **keda** 를 입력합니다.
     c. **표시 이름** 필드에 **keda** 와 같은 설명이 포함된 이름을 입력합니다.
     d. 선택 사항: **표시 이름** 필드에 네임스페이스에 대한 설명을 추가합니다.
     e. **생성**을 클릭합니다.
6. **설치**를 클릭합니다.
7. Custom Metrics Autoscaler Operator 구성 요소를 나열하여 설치를 확인합니다.
   a. **워크로드 → Pod**로 이동합니다.

b. 드롭다운 메뉴에서 **keda** 프로젝트를 선택하고 **custom-metrics-autoscaler-operator-*** Pod가 실행 중인지 확인합니다.

c. **워크로드 → 배포로** 이동하여 **custom-metrics-autoscaler-operator** 배포가 실행 중인지 확인합니다.

8. **선택 사항: 다음 명령을 사용하여 OpenShift CLI에서 설치를 확인합니다.**

```
$ oc get all -n 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
```

9. **필요한 CRD를 생성하는 KedaController 사용자 정의 리소스를 설치합니다.**

a. OpenShift Dedicated 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 만들기 를 클릭하고 파일을 편집합니다.

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: 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 정보 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 Operator에서 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** 의 경우 **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

스케일러라고도 하는 트리거는 Custom Metrics Autoscaler Operator에서 Pod를 확장하는 데 사용하는 메트릭을 제공합니다.

사용자 정의 지표 자동 스케일러는 현재 Prometheus, CPU, 메모리 및 Apache Kafka 트리거만 지원합니다.

scaled **Object** 또는 **scaled Job** 사용자 정의 리소스를 사용하여 다음 섹션에 설명된 대로 특정 오브젝트에 대한 트리거를 구성합니다.

확장된 오브젝트 또는 클러스터의 모든 확장기와 함께 사용할 인증 기관을 구성할 수 있습니다.

3.4.1. Prometheus 트리거 이해

설치된 OpenShift Dedicated 모니터링 또는 외부 Prometheus 서버를 메트릭 소스로 사용할 수 있는 Prometheus 메트릭을 기반으로 Pod를 확장할 수 있습니다. OpenShift Dedicated 모니터링을 메트릭 소스로 사용하는 데 필요한 구성에 대한 정보는 "사용자 정의 메트릭 자동 스케일러가 OpenShift Dedicated 모니터링을 사용하도록 구성"을 참조하십시오.

참고

Prometheus가 사용자 정의 지표 자동 스케일러가 스케일링하는 애플리케이션에서 지표를 수집하는 경우 사용자 정의 리소스에서 최소 복제본을 **0** 으로 설정하지 마십시오. 애플리케이션 Pod가 없는 경우 사용자 정의 메트릭 자동 스케일러에는 확장할 메트릭이 없습니다.

Prometheus 대상이 있는 확장 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 Dedicated 모니터링을 사용합니다.

(3) 선택 사항: 스케일링할 오브젝트의 네임스페이스를 지정합니다. 이 매개변수는 OpenShift Dedicated 모니터링을 메트릭 소스로 사용하는 경우 필수입니다.

(4) **external.metrics.k8s.io** API에서 메트릭을 식별하는 이름을 지정합니다. 둘 이상의 트리거를 사용하는 경우 모든 메트릭 이름은 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), 기본 인증(기본) 또는 **TLS** 인증(**TLS**)을 지원합니다. 다음 섹션에 설명된 대로 트리거 인증에 특정 인증 매개 변수를 구성합니다. 필요에 따라 시크릿을 사용할 수도 있습니다.

(8) 선택 사항: **X-Scope-OrgID** 헤더를 **Prometheus**의 다중 테넌트 **Cortex** 또는 **Mimir** 스토리지에 전달합니다. 이 매개변수는 **Prometheus**가 반환해야 하는 데이터를 표시하기 위해 다중 테넌트 **Prometheus** 스토리지에만 필요합니다.

9

선택 사항: **Prometheus** 대상이 손실된 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 건너뛸지 여부를 지정합니다. 예를 들어 테스트 환경에서 실행 중이고 **Prometheus** 끝점에서 자체 서명된 인증서를 사용하는 경우 검사를 건너뛸 수 있습니다.

- **false** 인 경우 인증서 검사가 수행됩니다. 이는 기본 동작입니다.
- **true** 인 경우 인증서 검사가 수행되지 않습니다.

중요

검사를 건너뛰는 것은 권장되지 않습니다.

3.4.1.1. OpenShift Dedicated 모니터링을 사용하도록 사용자 정의 메트릭 자동 스케일러 구성

설치된 **OpenShift Dedicated Prometheus** 모니터링을 사용자 정의 메트릭 자동 스케일러에서 사용하는 지표의 소스로 사용할 수 있습니다. 그러나 수행해야 하는 몇 가지 추가 구성이 있습니다.

참고

다음 단계는 외부 **Prometheus** 소스에 필요하지 않습니다.

이 섹션에 설명된 대로 다음 작업을 수행해야 합니다.

- 서비스 계정을 생성합니다.

- 서비스 계정에 대한 토큰을 생성하는 시크릿을 생성합니다.
- 트리거 인증을 생성합니다.
- 역할을 생성합니다.
- 서비스 계정에 해당 역할을 추가합니다.
- Prometheus에서 사용하는 트리거 인증 오브젝트에서 토큰을 참조합니다.

사전 요구 사항

- OpenShift Dedicated 모니터링이 설치되어 있어야 합니다.
- 사용자 정의 워크로드 모니터링 섹션에 설명된 대로 사용자 정의 워크로드 모니터링은 OpenShift Dedicated 모니터링 에서 활성화되어야 합니다.
- Custom Metrics Autoscaler Operator가 설치되어 있어야 합니다.

프로세스

1. 스케일링할 오브젝트를 사용하여 프로젝트로 변경합니다.

```
$ oc project my-project
```

2. 클러스터에 없는 경우 서비스 계정 및 토큰을 생성합니다.

   a. 다음 명령을 사용하여 서비스 계정 오브젝트를 생성합니다.

```
$ oc create serviceaccount thanos 1
```

1

서비스 계정의 이름을 지정합니다.

b. 시크릿 YAML을 생성하여 서비스 계정 토큰을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-token
  annotations:
    kubernetes.io/service-account.name: thanos 1
  type: kubernetes.io/service-account-token
```

1 서비스 계정의 이름을 지정합니다.

c. 다음 명령을 사용하여 보안 오브젝트를 생성합니다.

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

d. 다음 명령을 사용하여 서비스 계정에 할당된 토큰을 찾습니다.

```
$ oc describe serviceaccount thanos 1
```

1 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 1
Events:             <none>
```

1

트리거 인증에 이 토큰을 사용합니다.

3. 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 3
    key: token 4
  - parameter: ca
    name: thanos-token
    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
```

4. Thanos 지표를 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
```

5. 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 스케일링할 오브젝트의 네임스페이스를 지정합니다.

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 Dedicated 모니터링을 소스로 사용하려면 트리거 또는 scaler에서 다음 매개변수를 포함해야 합니다.

- triggers.type 은 prometheus여야 합니다
- triggers.metadata.serverAddress 는 https://thanos-querier.openshift-monitoring.svc.cluster.local:9092이어야 합니다
- triggers.metadata.authModes 는 베어러여야 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 메트릭을 지표 소스로 사용합니다.

사용자 정의 메트릭 자동 스케일러는 오브젝트와 연결된 Pod를 스케일링하여 사용자가 지정하는 CPU 사용량을 유지합니다. 자동 스케일러는 최소 및 최대 수 간에 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 scaled Job 사용자 정의 리소스와 함께 사용할 수 없습니다.
- 메모리 트리거를 사용하여 오브젝트를 확장할 때 여러 트리거를 사용하는 경우에도 오브젝트는 0 으로 스케일링되지 않습니다.

CPU 대상이 있는 확장 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

2

사용할 메트릭 유형(사용 또는 **AverageValue**) 을 지정합니다.

3

스케일링을 트리거하는 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

- **Utilization** 을 사용하는 경우 대상 값은 **Pod**에 대해 요청된 리소스의 백분율로 표시되는 모든 관련 **Pod**의 리소스 지표의 평균 값입니다.
- **AverageValue** 를 사용하는 경우 대상 값은 모든 관련 **Pod**의 지표의 평균입니다.

4

축소 시 최소 복제본 수를 지정합니다. **CPU** 트리거의 경우 **CPU** 지표만 사용하는 경우 **HPA**를 **0**으로 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 메트릭을 지표 소스로 사용합니다.

사용자 정의 메트릭 자동 스케일러는 오브젝트와 연결된 **Pod**를 스케일링하여 사용자가 지정하는 평균 메모리 사용량을 유지합니다. 자동 스케일러는 최소 및 최대 수 간에 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 **scaled Job** 사용자 정의 리소스와 함께 사용할 수 없습니다.
- 메모리 트리거를 사용하여 오브젝트를 확장할 때 여러 트리거를 사용하는 경우에도 오브젝트는 **0** 으로 스케일링되지 않습니다.

메모리 대상이 있는 확장 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 메트릭 유형(사용 또는 AverageValue) 을 지정합니다.

(3) 스케일링을 트리거하는 값을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다.

- Utilization 을 사용하는 경우 대상 값은 Pod에 대해 요청된 리소스의 백분율로 표시되는 모든 관련 Pod의 리소스 지표의 평균 값입니다.
- AverageValue 를 사용하는 경우 대상 값은 모든 관련 Pod의 지표의 평균입니다.

(4) 선택 사항: 전체 Pod가 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. 확장된 오브젝트 또는 확장 작업에서 **allowIdleConsumers** 매개변수를 **true** 로 설정하지 않는 한 사용자 정의 지표 자동 스케일러는 **Kafka** 파티션 수보다 확장되지 않습니다.

참고

소비자 그룹의 수가 주제의 파티션 수를 초과하면 추가 소비자 그룹은 유휴 상태로 유지됩니다. 이를 방지하려면 기본적으로 복제본 수를 초과하지 않습니다.

- 주제가 지정된 경우 주제의 파티션 수
- 주제가 지정되지 않은 경우 소비자 그룹에 있는 모든 항목의 파티션 수
- 확장 오브젝트 또는 확장 작업 **CR**에 지정된 **maxReplicaCount**

allowIdleConsumers 매개변수를 사용하여 이러한 기본 동작을 비활성화할 수 있습니다.

Kafka 대상이 있는 스케일링 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
tls: enable 13
```

1

Kafka를 트리거 유형으로 지정합니다.

2

Kafka가 오프셋 지연을 처리하는 **Kafka** 주제의 이름을 지정합니다.

3

연결할 **Kafka** 브로커의 쉼표로 구분된 목록을 지정합니다.

4

주제의 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 가장 빠른 값입니다. 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 **Kafka** 소비자를 유휴 상태로 설정할 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 방식을 지정합니다.

- true 인 경우 소비자는 해당 파티션에 대해 0으로 확장됩니다.

- false 인 경우 scaler는 해당 파티션에 대해 단일 소비자를 유지합니다. 이는 기본값입니다.

10

선택 사항: 트리거에서 현재 오프셋이 이전 폴링 주기의 현재 오프셋과 동일한 파티션의 파티션 지연을 포함하거나 제외하는지 여부를 지정합니다.

- true 인 경우 scaler는 이러한 파티션의 파티션 지연을 제외합니다.

- false 인 경우 트리거에는 모든 파티션에 모든 소비자 지연이 포함됩니다. 이는 기본값입니다.

11

선택 사항: Kafka 브로커의 버전을 지정합니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다. 기본값은 1.0.0 입니다.

12

선택 사항: 쉼표로 구분된 파티션 ID 목록을 지정하여 스케일링 범위를 지정합니다. 설정된 경우 지연을 계산할 때 나열된 ID만 고려됩니다. 따옴표로 묶은 문자열 값으로 지정해야 합니다. 기본값은 모든 파티션을 고려하는 것입니다.

13

선택 사항: Kafka에 TSL 클라이언트 인증을 사용할지 여부를 지정합니다. 기본값은 disable 입니다. TLS 구성에 대한 자세한 내용은 "사용자 정의 메트릭 자동 스케일러 트리거 인증 이해"를 참조하십시오.

3.4.5. Cron 트리거 이해

시간 범위를 기반으로 Pod를 확장할 수 있습니다.

시간 범위가 시작되면 사용자 정의 지표 자동 스케일러는 구성된 최소 Pod 수에서 지정된 Pod 수로 오브젝트와 연결된 Pod를 스케일링합니다. 시간 범위가 끝나면 Pod가 구성된 최소로 다시 확장됩니다. 시간 기간은 cron 형식으로 구성해야 합니다.

참고

Cron 트리거를 사용하는 사용자 정의 메트릭 자동 스케일러는 지정된 횟수에 따라 Pod를 스케일링하고 매일, 매주, 월간 또는 연간 스케줄에 Pod를 스케일링할 수 없습니다.

다음 예제에서는 이 확장 오브젝트와 연결된 Pod를 오전 6시부터 오후 6 시 30분에서 오후 6시 30분으로 스케일링합니다.

Cron 트리거가 있는 확장 오브젝트의 예

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cron-scaledobject
  namespace: default
spec:
  scaleTargetRef:
    name: my-deployment
  minReplicaCount: 0 (1)
  maxReplicaCount: 100 (2)
  cooldownPeriod: 300
  triggers:
  - type: cron (3)
    metadata:
      timezone: Asia/Kolkata (4)
      start: "0 6 * * *" (5)
      end: "30 18 * * *" (6)
      desiredReplicas: "100" (7)
```

(1) 시간 프레임이 끝날 때 축소할 최소 Pod 수를 지정합니다.

2

확장 시 최대 복제본 수를 지정합니다. 이 값은 **desiredReplicas** 와 동일해야 합니다. 기본값은 **100**입니다.

3

Cron 트리거를 지정합니다.

4

시간 프레임의 시간대를 지정합니다. 이 값은 **IANA** 시간대 데이터베이스 여야 합니다.

5

시간 프레임의 시작을 지정합니다.

6

시간 프레임의 끝을 지정합니다.

7

시간 프레임의 시작과 종료 사이에 스케일링할 **Pod** 수를 지정합니다. 이 값은 **maxReplicaCount** 와 동일해야 합니다.

3.5. 사용자 정의 메트릭 자동 스케일러 트리거 인증 이해

트리거 인증을 사용하면 확장 오브젝트 또는 관련 컨테이너에서 사용할 수 있는 확장 작업에 인증 정보를 포함할 수 있습니다. 트리거 인증을 사용하여 **OpenShift Dedicated** 보안, 플랫폼 네이티브 **Pod** 인증 메커니즘, 환경 변수를 전달할 수 있습니다.

스케일링할 오브젝트와 동일한 네임스페이스에 **TriggerAuthentication** 오브젝트를 정의합니다. 해당 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.

트리거 인증 및 클러스터 트리거 인증은 동일한 구성을 사용합니다. 그러나 클러스터 트리거 인증에는 확장된 오브젝트의 인증 참조에 추가 **kind** 매개변수가 필요합니다.

기본 인증을 위한 시크릿 예

```
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-secret
  namespace: default
data:
  username: "dXNlcm5hbWU=" (1)
  password: "cGFzc3dvcmQ="
```

(1)

트리거 인증에 제공할 사용자 이름 및 암호입니다. 데이터 스탠자의 값은 **base-64**로 인코딩되어야 합니다.

기본 인증의 보안을 사용하여 트리거 인증의 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace (1)
spec:
  secretTargetRef: (2)
  - parameter: username (3)
    name: my-basic-secret (4)
    key: username (5)
  - parameter: password
    name: my-basic-secret
    key: password
```

(1)

스케일링할 오브젝트의 네임스페이스를 지정합니다.

(2)

3

보안을 사용하여 제공할 인증 매개변수를 지정합니다.

4

사용할 시크릿 이름을 지정합니다.

5

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

기본 인증을 위한 시크릿을 사용한 클러스터 트리거 인증 예

```
kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata: 1
  name: secret-cluster-triggerauthentication
spec:
  secretTargetRef: 2
  - parameter: username 3
    name: my-basic-secret 4
    key: username 5
  - parameter: password
    name: my-basic-secret
    key: password
```

1

클러스터 트리거 인증에 네임스페이스는 사용되지 않습니다.

2

이 트리거 인증이 메트릭 끝점에 연결할 때 권한 부여에 시크릿을 사용하도록 지정합니다.

3

보안을 사용하여 제공할 인증 매개변수를 지정합니다.

4

5

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

CA(인증 기관) 세부 정보가 있는 보안 예

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: my-namespace
data:
  ca-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0... 1
  client-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0... 2
  client-key.pem: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0t...
```

1

지표 끝점 인증을 위한 **TLS CA** 인증서를 지정합니다. 값은 **base-64**로 인코딩되어야 합니다.

2

TLS 클라이언트 인증을 위한 **TLS** 인증서 및 키를 지정합니다. 값은 **base-64**로 인코딩되어야 합니다.

CA 세부 정보용 보안을 사용하여 트리거 인증의 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
    - parameter: key 3
      name: my-secret 4
      key: client-key.pem 5
```

```
- parameter: ca 6
  name: my-secret 7
  key: ca-cert.pem 8
```

1

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

이 트리거 인증이 메트릭 끝점에 연결할 때 권한 부여에 시크릿을 사용하도록 지정합니다.

3

사용할 인증 유형을 지정합니다.

4

사용할 시크릿 이름을 지정합니다.

5

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

6

메트릭 끝점에 연결할 때 사용자 정의 **CA**의 인증 매개 변수를 지정합니다.

7

사용할 시크릿 이름을 지정합니다.

8

지정된 매개변수와 함께 사용할 시크릿의 키를 지정합니다.

전달자 토큰이 있는 시크릿 예

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: my-secret
  namespace: my-namespace
data:
  bearerToken: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV" 1
```

1

전달자 인증에 사용할 전달자 토큰을 지정합니다. 데이터 스탠자의 값은 **base-64**로 인코딩되어야 합니다.

전달자 토큰을 사용한 트리거 인증 예

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: token-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: bearerToken 3
    name: my-secret 4
    key: bearerToken 5
```

1

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

이 트리거 인증이 메트릭 끝점에 연결할 때 권한 부여에 시크릿을 사용하도록 지정합니다.

3

사용할 인증 유형을 지정합니다.

4

사용할 시크릿 이름을 지정합니다.

5

지정된 매개변수와 함께 사용할 토큰의 키를 지정합니다.

환경 변수를 사용한 트리거 인증 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 메트릭 끝점에 연결할 때 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 확장된 오브젝트에서 **scaleTargetRef** 에서 참조하는 것과 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

스케일링할 오브젝트의 네임스페이스를 지정합니다.

2

메트릭 끝점에 연결할 때 이 트리거 인증이 플랫폼 네이티브 **Pod** 인증을 사용하도록 지정합니다.

3

Pod ID를 지정합니다. 지원되는 값은 **none,azure,gcp,aws-eks** 또는 **aws-kiam** 입니다. 기본값은 **none** 입니다.

3.5.1. 트리거 인증 사용

사용자 지정 리소스를 사용하여 인증을 생성한 다음 확장된 오브젝트 또는 확장 작업에 대한 참조를 추가하여 트리거 인증 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.

시크릿 예

-

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  user-name: <base64_USER_NAME>
  password: <base64_USER_PASSWORD>
```

프로세스

1. TriggerAuthentication 또는 ClusterTriggerAuthentication 오브젝트를 생성합니다.

   a. 오브젝트를 정의하는 YAML 파일을 생성합니다.

   보안을 사용한 트리거 인증의 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 일시 중지하거나 해제 중요하지 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 지표 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"
# ...
```

프로세스

1. 다음 명령을 사용하여 워크로드에 대한 **scaledObject CR**을 편집합니다.

```
$ oc edit ScaledObject scaledobject
```

2. 값이 있는 **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 Operator**가 복제본을 지정된 값으로 확장하고 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"
# ...
```

프로세스

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

개별 사용자, 관리자 또는 시스템의 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 들어오는 위치를 이해하는 데 도움이 될 수 있습니다. 이는 사용자 애플리케이션에서 만든 자동 확장 요청에 의해 백엔드가 과부하되는 경우 주요 정보입니다. 그러면 문제가 있는 애플리케이션인지 확인해야 합니다.

3.7.1. 감사 로깅 구성

KedaController 사용자 정의 리소스를 편집하여 **Custom Metrics Autoscaler Operator**에 대한 감사를 구성할 수 있습니다. 로그는 **KedaController CR**에서 영구 볼륨 클레임을 사용하여 보안되는 볼륨의 감사 로그 파일로 전송됩니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator**가 설치되어 있어야 합니다.

프로세스

1. **KedaController** 사용자 정의 리소스를 편집하여 **auditConfig** 스탠자를 추가합니다.

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: 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. 감사 로그의 출력 형식을 **legacy** 또는 **json** 으로 지정합니다.

2. 로그 데이터를 저장하기 위한 기존 영구 볼륨 클레임을 지정합니다. **API** 서버로 들어오는 모든 요청은 이 영구 볼륨 클레임에 기록됩니다. 이 필드를 비워 두면 로그 데이터가 **stdout**으로 전송됩니다.

3.

- **None:** 이벤트를 기록하지 마십시오.
- **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 텍스트입니다. 이 옵션은 리소스가 아닌 요청에는 적용되지 않습니다.

4 이벤트가 생성되지 않는 단계를 지정합니다.

5 요청 및 응답 본문의 관리 필드가 **API** 감사 로그에 기록되지 않을지 여부를 생략할지 여부를 **true** 로 지정하여 필드를 포함하는 필드 또는 **false** 를 생략합니다.

6 감사 로그의 크기와 수명을 지정합니다.

- **maxAge:** 파일 이름에 인코딩된 타임스탬프에 따라 감사 로그 파일을 유지하는 최대 일 수입니다.
- **maxBackup:** 유지할 최대 감사 로그 파일 수입니다. 모든 감사 로그 파일을 유지하려면 **0** 으로 설정합니다.
- **maxSize:** 교체되기 전에 감사 로그 파일의 최대 크기(**MB**)입니다.

검증

1. 감사 로그 파일을 직접 확인합니다.

a.

keda-metrics-apiserver-* Pod의 이름을 가져옵니다.

```
oc get pod -n 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. Metadata, Request ,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":"4c81d
41b-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/1.28","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. 다음과 유사한 명령을 사용하여 **keda-metrics-apiserver-* Pod**에 로그인합니다.

```
$ oc rsh pod/keda-metrics-apiserver-<hash> -n keda
```

   예를 들면 다음과 같습니다.

```
$ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n 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. 필요에 따라 로그를 확인합니다.

```
sh-4.4$ cat <log_name>/<pvc_name>|grep -i <log_level> 1
```

   1

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 **ID**입니다.

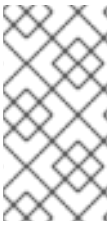
must-gather 툴을 사용하여 다음 항목을 포함하여 **Custom Metrics Autoscaler Operator** 및 해당 구성 요소에 대한 데이터를 수집할 수 있습니다.

- keda 네임스페이스 및 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 Operator의 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 Dedicated must-gather 명령 oc adm must-gather 에서는 Custom Metrics Autoscaler Operator 데이터를 수집하지 않습니다.

사전 요구 사항

- dedicated-admin 역할의 사용자로 OpenShift Dedicated에 로그인했습니다.
- OpenShift Dedicated CLI(oc)가 설치되어 있어야 합니다.

프로세스

1. must-gather 데이터를 저장하려는 디렉터리로 이동합니다.
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 가져와 Custom Metric Autoscaler Operator를 사용할 수 있는 모든 클러스터에서 작동합니다.

- Custom Metric Autoscaler Operator 정보 외에도 기본 must-gather 데이터를 수집하려면 다음을 수행합니다.

  a. 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}')"
  ```

  b. Custom 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

```
└── keda
    ├── apps
    │   ├── daemonsets.yaml
    │   ├── deployments.yaml
    │   ├── replicasets.yaml
    │   └── statefulsets.yaml
    ├── apps.openshift.io
    │   └── deploymentconfigs.yaml
    ├── autoscaling
    │   └── horizontalpodautoscalers.yaml
    ├── batch
    │   ├── cronjobs.yaml
    │   └── jobs.yaml
    ├── build.openshift.io
    │   ├── buildconfigs.yaml
    │   └── builds.yaml
    ├── core
```

```
    ├── configmaps.yaml
    ├── endpoints.yaml
    ├── events.yaml
    ├── persistentvolumeclaims.yaml
    ├── pods.yaml
    ├── replicationcontrollers.yaml
    ├── secrets.yaml
    └── services.yaml
├── discovery.k8s.io
│   └── endpointslices.yaml
├── image.openshift.io
│   └── imagestreams.yaml
├── k8s.ovn.org
│   ├── egressfirewalls.yaml
│   └── egressqoses.yaml
├── keda.sh
│   ├── kedacontrollers
│   │   └── keda.yaml
│   ├── scaledobjects
│   │   └── example-scaledobject.yaml
│   └── triggerauthentications
│       └── example-triggerauthentication.yaml
├── monitoring.coreos.com
│   └── servicemonitors.yaml
├── networking.k8s.io
│   └── networkpolicies.yaml
├── keda.yaml
├── pods
│   ├── custom-metrics-autoscaler-operator-58bd9f458-ptgwx
│   │   ├── custom-metrics-autoscaler-operator
│   │   │   └── custom-metrics-autoscaler-operator
│   │   │       └── logs
│   │   │           ├── current.log
│   │   │           ├── previous.insecure.log
│   │   │           └── previous.log
│   │   └── custom-metrics-autoscaler-operator-58bd9f458-ptgwx.yaml
│   ├── custom-metrics-autoscaler-operator-58bd9f458-thbsh
│   │   └── custom-metrics-autoscaler-operator
│   │       └── custom-metrics-autoscaler-operator
│   │           └── logs
│   ├── keda-metrics-apiserver-65c7cc44fd-6wq4g
│   │   ├── keda-metrics-apiserver
│   │   │   └── keda-metrics-apiserver
│   │   │       └── logs
│   │   │           ├── current.log
│   │   │           ├── previous.insecure.log
│   │   │           └── previous.log
│   │   └── keda-metrics-apiserver-65c7cc44fd-6wq4g.yaml
│   └── keda-operator-776cbb6768-fb6m5
│       ├── keda-operator
│       │   └── keda-operator
│       │       └── logs
│       │           ├── current.log
│       │           ├── previous.insecure.log
│       │           └── previous.log
│       └── keda-operator-776cbb6768-fb6m5.yaml
```

```
├── policy
│   └── poddisruptionbudgets.yaml
└── route.openshift.io
    └── routes.yaml
```

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 클러스터의 온-클러스터 모니터링 구성 요소에서 가져오는 즉시 사용 가능한 메트릭을 표시합니다. **PromQL(Prometheus Query Language)**을 사용하여 문제를 분석하고 진단하여 메트릭을 쿼리할 수 있습니다. 컨트롤러 **Pod**가 다시 시작되면 모든 메트릭이 재설정됩니다.

3.9.1. 성능 지표 액세스

OpenShift Dedicated 웹 콘솔을 사용하여 메트릭에 액세스하고 쿼리를 실행할 수 있습니다.

프로세스

1. **OpenShift Dedicated** 웹 콘솔에서 관리자 화면을 선택합니다.
2. 모니터링 → 메트릭 을 선택합니다.
3. 사용자 지정 쿼리를 만들려면 표현식 필드에 **PromQL** 쿼리를 추가합니다.

4. 여러 쿼리를 추가하려면 쿼리 추가를 선택합니다.

3.9.1.1. 제공된 Operator 지표

Custom Metrics Autoscaler Operator는 OpenShift Dedicated 웹 콘솔을 사용하여 볼 수 있는 다음 메트릭을 노출합니다.

표 3.1. 사용자 정의 메트릭 자동 스케일러 Operator 지표

메트릭 이름	설명
keda_scaler_activity	특정 스케일러가 활성 상태인지 아니면 비활성 상태인지 여부입니다. 값 1 은 스케일러가 활성임을 나타냅니다. 값이 0 이면 스케일러가 비활성 상태임을 나타냅니다.
keda_scaler_metrics_value	대상 평균을 계산할 때 Horizontal Pod Autoscaler(HPA)에서 사용하는 각 scaler 메트릭의 현재 값입니다.
keda_scaler_metrics_latency	각 scaler에서 현재 메트릭을 검색하는 대기 시간입니다.
keda_scaler_errors	각 scaler에 대해 발생한 오류 수입니다.
keda_scaler_errors_total	모든 확장성에 대해 발생한 총 오류 수입니다.
keda_scaled_object_errors	스케일링된 각 관찰에 대해 발생한 오류 수입니다.
keda_resource_totals	각 사용자 정의 리소스 유형에 대한 각 네임 스페이스의 총 사용자 정의 지표 자동 스케일러 사용자 정의 리소스 수입니다.
keda_trigger_totals	트리거 유형별 총 트리거 수입니다.

사용자 정의 메트릭 자동 스케일러 Admission webhook 메트릭

Custom Metrics Autoscaler Admission webhook에서는 다음 Prometheus 메트릭도 노출합니다.

메트릭 이름	설명
keda_scaled_object_validation_total	확장 가능한 오브젝트 검증 수입니다.
keda_scaled_object_validation_errors	검증 오류 수입니다.

3.10. 사용자 정의 메트릭 자동 스케일러를 추가하는 방법

사용자 지정 지표 자동 스케일러를 추가하려면 배포, 상태 저장 세트 또는 사용자 정의 리소스에 대한 **scaled Object** 사용자 정의 리소스를 만듭니다. 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 오브젝트에서 생성한 워크로드에 대한 사용자 정의 메트릭 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 출력은 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>
```

- 스케일링할 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"
# ...
```

프로세스

1. 다음과 유사한 YAML 파일을 생성합니다. 이름 &lt ;2> , 오브젝트 이름 &lt ;4& gt; 및 오브젝트 종류 &lt ;5& gt;만 필요합니다.

확장된 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
```

```
authentication: 17
  name: prom-triggerauthentication
  kind: TriggerAuthentication
```

1

선택 사항: **Custom Metrics Autoscaler Operator**가 "워크로드에 사용자 정의 메트릭 자동 스케일러 사용" 섹션에 설명된 대로 복제본을 지정된 값으로 확장하고 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 오브젝트의 이름을 지정합니다.

5

type을 **Deployment,StatefulSet** 또는 **CustomResource** 로 지정합니다.

6

선택 사항: 사용자 지정 지표 자동 스케일러가 시크릿을 보유하는 환경 변수를 가져오는 대상 리소스의 컨테이너 이름을 지정합니다. 기본값은 **.spec.template.spec.containers[0]** 입니다.

7

선택 사항: **minReplicaCount** 가 **0** 으로 설정된 경우 배포를 다시 **0** 으로 스케일링하기 전에 마지막 트리거가 보고된 후 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: **scaler**가 **failureThreshold** 매개변수로 정의된 횟수에 대한 소스에서 메트릭을 가져오지 못하는 경우 다시 대체할 복제본 수를 지정합니다. 대체 동작에 대한 자세한 내용은 **KEDA** 설명서 를 참조하십시오.

12

선택 사항: 각 트리거를 확인하는 간격을 초 단위로 지정합니다. 기본값은 **30** 입니다.

13

선택 사항: 확장된 개체를 삭제한 후 대상 리소스를 원래 복제본 수로 확장할지 여부를 지정합니다. 기본값은 **false** 이며, 확장 오브젝트를 삭제할 때 복제본 수를 그대로 유지합니다.

14

선택 사항: 수평 **Pod** 자동 스케일러의 이름을 지정합니다. 기본값은 **keda-hpa-{scaled-object-name}** 입니다.

15

선택 사항: "확장 정책" 섹션에 설명된 대로 **Pod**를 확장 또는 축소하는 데 사용할 스케일링 정책을 지정합니다.

16

"사용자 정의 메트릭 자동 스케일러 트리거 이해" 섹션에 설명된 대로 스케일링의 기준으로 사용할 트리거를 지정합니다. 이 예에서는 **OpenShift Dedicated** 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 메트릭 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 확인합니다.

- **TRIGGERS:** 사용 중인 트리거 또는 스케일러를 나타냅니다.
- **AUTHENTICATION:** 사용 중인 트리거 인증의 이름을 나타냅니다.
- **READY:** 스케일링된 오브젝트가 스케일링을 시작할 준비가 되었는지 여부를 나타냅니다.
  - **True** 인 경우 확장 오브젝트가 준비됩니다.
  - **False** 인 경우 생성한 오브젝트 중 하나 이상의 오브젝트의 문제로 인해 확장 오브젝트가 준비되지 않은 것입니다.

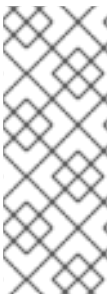
- **ACTIVE:** 스케일링이 수행되는지 여부를 나타냅니다.
  - **True** 인 경우 스케일링이 수행됩니다.
  - **False** 인 경우 메트릭이 없거나 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 스케일링이 수행되지 않습니다.
- **FALLBACK:** 사용자 정의 메트릭 자동 스케일러가 소스에서 메트릭을 가져올 수 있는지 여부를 나타냅니다.
  - **False** 인 경우 사용자 정의 메트릭 자동 스케일러에 메트릭이 표시됩니다.
  - **True** 인 경우 메트릭이 없거나 생성한 오브젝트 중 하나 이상에 문제가 있기 때문에 사용자 정의 메트릭 자동 스케일러가 메트릭을 가져오고 있습니다.

3.10.2. 추가 리소스

- 사용자 정의 메트릭 자동 스케일러 트리거 인증 이해

3.11. CUSTOM METRICS AUTOSCALER OPERATOR 제거

OpenShift Dedicated 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 **keda** 프로젝트를 삭제할 때 **OpenShift Dedicated**가 중단될 수 있습니다. **CR**을 삭제하기 전에 **Custom Metrics Autoscaler Operator**를 삭제하면 **CR**을 삭제할 수 없습니다.

3.11.1. Custom Metrics Autoscaler Operator 설치 제거

다음 절차에 따라 **OpenShift Dedicated** 클러스터에서 사용자 정의 메트릭 자동 스케일러를 제거합니다.

사전 요구 사항

- **Custom Metrics Autoscaler Operator**가 설치되어 있어야 합니다.

프로세스

1. **OpenShift Dedicated** 웹 콘솔에서 **Operator** → 설치된 **Operator** 를 클릭합니다.
2. **keda** 프로젝트로 전환합니다.
3. **KedaController** 사용자 지정 리소스를 제거합니다.
   a. **CustomMetricsAutoscaler Operator**를 찾아 **KedaController** 탭을 클릭합니다.
   b. 사용자 지정 리소스를 찾은 다음 **KedaController** 삭제 를 클릭합니다.
   c. 제거를 클릭합니다.
4. **Custom Metrics Autoscaler Operator**를 제거합니다.
   a. **Operators** → 설치된 **Operators**를 클릭합니다.
   b. **CustomMetricsAutoscaler Operator**를 찾아 옵션 메뉴  를 클릭하고 **Operator** 설치 제거를 선택합니다.
   c. 제거를 클릭합니다.
5. 선택 사항: **OpenShift CLI**를 사용하여 사용자 정의 메트릭 자동 스케일러 구성 요소를 제거합니다.

a. 사용자 정의 메트릭 자동 스케일러 CRD를 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 메트릭 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 메트릭 자동 스케일러 프로젝트를 삭제합니다.

```
$ oc delete project keda
```

7. Cluster Metric Autoscaler Operator를 삭제합니다.

```
$ oc delete operator/openshift-custom-metrics-autoscaler-operator.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 Dedicated에는 대부분의 사용자의 요구를 충족하는 기본 스케줄러가 제공됩니다. 기본 스케줄러는 고유 툴과 사용자 정의 툴을 모두 사용하여 Pod에 가장 적합한 항목을 결정합니다.

고급 Pod 예약

새 Pod가 배치되는 위치를 더 많이 제어해야 하는 경우 OpenShift Dedicated 고급 스케줄링 기능을 사용하여 Pod가 필요하거나 특정 노드에서 또는 특정 Pod와 함께 실행하는 기본 설정이 있도록 Pod를 구성할 수 있습니다.

다음 스케줄링 기능을 사용하여 Pod 배치를 제어할 수 있습니다.

- Pod 유사성 및 유사성 방지 규칙
- 노드 유사성
- 노드 선택기
- 노드 과다 할당

4.1.1. 기본 스케줄러 정보

기본 OpenShift Dedicated Pod 스케줄러는 클러스터 내의 노드에 새 Pod 배치를 결정합니다. Pod에서 데이터를 읽고 구성된 프로필에 따라 적합한 노드를 찾습니다. 이는 완전히 독립적이며 독립 실행형 솔

루션으로 존재합니다. Pod를 수정하지 않습니다. Pod를 특정 노드에 연결하는 Pod에 대한 바인딩을 생성합니다.

4.1.1.1. 기본 예약 이해

기존 일반 스케줄러는 3단계 작업에서 Pod를 호스팅할 노드를 선택하는 기본 플랫폼 제공 스케줄러 *엔진*에 해당합니다.

노드 필터링

사용 가능한 노드를 지정된 제약 조건 또는 요구 사항에 따라 필터링합니다. 이 작업은 *서술자* 또는 필터라는 필터 함수 목록을 통해 각 노드를 실행하여 수행됩니다.

필터링된 노드 목록 우선 순위

이는 일련의 우선 순위 또는 점수 지정을 통해 각 노드를 통과하며 0에서 10 사이의 점수를 할당하는 함수로, 0은 Pod를 호스팅하는 데 적합하지 않음을 나타내는 10을 나타냅니다. 스케줄러 구성은 각 점수 함수에 대해 간단한 가중치 (수정 숫자 값)를 사용할 수도 있습니다. 각 점수 함수에서 제공하는 노드 점수는 가중치(대부분 점수의 기본 가중치는 1)를 곱한 다음 모든 점수에서 제공하는 각 노드의 점수를 더하여 결합합니다. 관리자가 이 가중치 속성을 사용하여 일부 점수에 더 높은 중요성을 부여할 수 있습니다.

최적의 노드 선택

노드는 해당 점수에 따라 정렬되며 점수가 가장 높은 노드가 Pod를 호스팅하도록 선택됩니다. 여러 노드의 점수가 동일한 경우 해당 노드 중 하나가 무작위로 선택됩니다.

4.1.2. 스케줄러 사용 사례

OpenShift Dedicated 내에서 예약하는 중요한 사용 사례 중 하나는 유연한 유사성 및 유사성 방지 정책을 지원하는 것입니다.

4.1.2.1. 유사성

관리자는 임의의 토폴로지 수준 또는 여러 수준에도 유사성을 지정하도록 스케줄러를 구성할 수 있어야 합니다. 특정 수준의 유사성은 동일한 서비스에 속하는 모든 Pod가 동일한 수준에 속하는 노드에 예약됨을 나타냅니다. 이렇게 하면 관리자가 피어 Pod가 지리적으로 너무 멀리 떨어져 있지 않도록 할 수 있어 애플리케이션의 대기 시간 요구 사항이 처리됩니다. 동일한 유사성 그룹 내에서 Pod를 호스팅할 수 있는 노드가 없는 경우 Pod를 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 Pod와 관련된 Pod 배치를 참조하십시오.

관리자는 이러한 고급 예약 기능을 사용하여 Pod를 예약할 수 있는 노드를 지정하고 기타 Pod와 관련된 예약을 강제 적용하거나 거부할 수 있습니다.

4.1.2.2. anti-affinity

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 Pod가 해당 수준에 속하는 노드에 분배되어 있음을 나타냅니다. 이 경우 고가용성을 위해 애플리케이션이 잘 분배됩니다. 스케줄러는 적용 가능한 모든 노드에서 가능한 한 균등하게 서비스 Pod의 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 Pod와 관련된 Pod 배치를 참조하십시오.

관리자는 이러한 고급 예약 기능을 사용하여 Pod를 예약할 수 있는 노드를 지정하고 기타 Pod와 관련된 예약을 강제 적용하거나 거부할 수 있습니다.

4.2. 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 Dedicated에서 Pod 유사성 및 Pod 유사성 방지를 사용하면 다른 Pod의 키 값 라벨에 따라 Pod를 예약할 수 있는 노드를 제한할 수 있습니다.

4.2.1. 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.

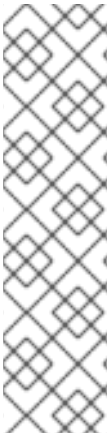
*예를 들어 유사성 규칙을 사용하여 서비스 내에서 또는 다른 서비스의 **Pod**와 관련하여 **Pod**를 분배하거나 패키징할 수 있습니다. 유사성 방지 규칙을 사용하면 특정 서비스의 **Pod**가 첫 번째 서비스의 **Pod** 성능을 방해하는 것으로 알려진 다른 서비스의 **Pod**와 동일한 노드에 예약되지 않도록 할 수 있습니다. 또는 서비스의 **Pod**를 노드, 가용성 영역 또는 가용성 세트에 분배하여 관련 오류를 줄일 수 있습니다.*

참고

*라벨 선택기는 여러 **Pod** 배포가 있는 **Pod**와 일치할 수 있습니다. 일치하는 **Pod**를 방지하려면 유사성 방지 규칙을 구성할 때 레이블의 고유한 조합을 사용합니다.*

***Pod** 유사성 규칙에는 필수 및 기본 두 가지의 유형이 있습니다.*

*노드에 **Pod**를 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 **Pod**에 적절한 노드를 찾지 못하는 경우가 있습니다. 이 경우 **Pod**를 예약하지 못할 수 있습니다.*

*이러한 상황을 방지하려면 우선순위가 같은 **Pod**를 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 **Pod**가 한 개 이상 있고 키가 **security**이고 값이 **S1**인 라벨이 있는 경우에만 노드에 **Pod**를 예약할 수 있음을 나타냅니다. **Pod** 유사성 방지 규칙은 노드에서 이미 **Pod**를 실행 중이고 키가 **security**이고 값이 **S2**인 라벨이 있는 경우 **Pod**를 노드에 예약하지 않는 것을 선호함을 나타냅니다.*

***Pod** 유사성이 포함된 샘플 **Pod** 구성 파일*

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAffinity: (1)
      requiredDuringSchedulingIgnoredDuringExecution: (2)
      - labelSelector:
          matchExpressions:
          - key: security (3)
            operator: In (4)
            values:
            - S1 (5)
        topologyKey: topology.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

(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 DoesNotExist일 수 있습니다.*

Pod 유사성 방지가 포함된 샘플 Pod 구성 파일

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1

Pod 유사성 방지를 구성하는 스탠자입니다.

2

기본 규칙을 정의합니다.

3

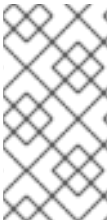
기본 규칙의 가중치를 지정합니다. 가중치가 가장 높은 노드가 우선합니다.

4

유사성 방지 규칙이 적용되는 시기를 결정하는 Pod 라벨에 대한 설명입니다. 라벨의 키와 값을 지정합니다.

5

*이 연산자는 기존 **Pod**의 라벨과 새 **Pod** 사양에 있는 **matchExpression** 매개변수의 값 집합 간의 관계를 나타냅니다. **In, NotIn, Exists** 또는 **DoesNotExist**일 수 있습니다.*

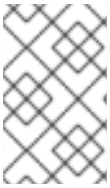


참고

*런타임 시 노드의 라벨이 변경되어 **Pod**의 유사성 규칙이 더 이상 충족되지 않는 경우 **Pod**가 노드에서 계속 실행됩니다.*

4.2.2. Pod 유사성 규칙 구성

*다음 단계에서는 라벨이 있는 **Pod** 및 유사성을 사용하여 해당 **Pod**에 예약할 수 있는 **Pod**를 생성하는 간단한 **2-Pod** 구성을 보여줍니다.*



참고

*예약된 **Pod**에 선호도를 직접 추가할 수 없습니다.*

프로세스

1. ***Pod** 사양에서 특정 라벨을 사용하여 **Pod**를 생성합니다.*

   a. *다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
```

```
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

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)

5

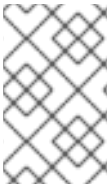
*이러한 토폴로지 도메인을 나타내기 위해 사용하며 미리 채워져 있는 Kubernetes 라벨인 **topologyKey**를 지정합니다.*

b. *Pod를 생성합니다.*

```
$ oc create -f <pod-spec>.yaml
```

4.2.3. Pod 유사성 방지 규칙 구성

*다음 단계에서는 라벨이 있는 **Pod** 및 유사성 방지 기본 규칙을 사용하여 해당 **Pod**에 예약하지 않는 **Pod**를 생성하는 간단한 **2-Pod** 구성을 보여줍니다.*



참고

*예약된 **Pod**에 선호도를 직접 추가할 수 없습니다.*

프로세스

1. *Pod 사양에서 특정 라벨을 사용하여 **Pod**를 생성합니다.*

a. *다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
```

```
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

b. Pod를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

2. 다른 Pod를 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 1-100을 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

연산자 를 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. **Pod**를 생성합니다.

```
$ oc create -f <pod-spec>.yaml
```

4.2.4. 샘플 Pod 유사성 및 유사성 방지 규칙

다음 예제에서는 **Pod** 유사성 및 **Pod** 유사성 방지를 보여줍니다.

4.2.4.1. Pod 유사성

다음 예제에서는 일치하는 라벨 및 라벨 선택기가 있는 **Pod**의 **Pod** 유사성을 보여줍니다.

- **Pod team4**에는 라벨 **team:4**가 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: ocp
```

```
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- Pod team4a에는 podAffinity 아래에 라벨 선택기 team:4가 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: team4a
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- team4a Pod는 team4 Pod와 동일한 노드에 예약됩니다.

4.2.4.2. 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- *Pod pod-s2에는 podAntiAffinity 아래에 라벨 선택기 security:s1이 있습니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- *Pod pod-s2는 pod-s1과 동일한 노드에 예약할 수 없습니다.*

4.2.4.3. 일치하는 라벨이 없는 Pod 유사성

다음 예제에서는 일치하는 라벨 및 라벨 선택기가 없는 Pod의 Pod 유사성을 보여줍니다.

- *Pod pod-s1에는 라벨 security:s1이 있습니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- *Pod pod-s2에는 라벨 선택기 security:s2가 있습니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- *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. 노드 유사성 규칙을 사용하여 노드에 대한 POD 배치 제어

유사성은 예약할 노드를 제어하는 Pod의 속성입니다.

OpenShift Dedicated 노드 유사성은 스케줄러에서 Pod를 배치할 수 있는 위치를 결정하는 데 사용하는 규칙 집합입니다. 규칙은 노드의 사용자 정의 라벨과 Pod에 지정된 라벨 선택기를 사용하여 정의합니다.

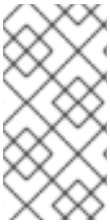
4.3.1. 노드 유사성 이해

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 **Pod**를 예약하려면 먼저 필수 규칙을 충족해야 합니다. 기본 규칙은 규칙이 충족되는 경우 스케줄러가 규칙을 적용하려고 하지만 반드시 적용되는 것은 아닙니다.



참고

노드의 라벨이 런타임에 변경되어 **Pod**에 대한 노드 유사성 규칙이 더 이상 충족되지 않으면 **Pod**가 해당 노드에서 계속 실행됩니다.

노드 유사성은 **Pod** 사양 파일을 통해 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 예제는 키가 **e2e-az-NorthSouth**이고 값이 **e2e-az-North** 또는 **e2e-az-South**인 라벨이 있는 노드에 **Pod**를 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
```

```
    capabilities:
      drop: [ALL]
# ...
```

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 Gt일 수 있습니다.

다음 예제는 Pod에 대해 키가 e2e-az-EastWest이고 값이 e2e-az-East 또는 e2e-az-West인 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

1

노드 유사성을 구성하는 스탠자입니다.

2

기본 규칙을 정의합니다.

3

기본 규칙의 가중치를 지정합니다. 가중치가 높은 노드가 우선합니다.

4 6 7

규칙을 적용하려면 일치해야 하는 키/값(라벨)입니다.

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.*

*명시적인 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 __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.*

4.3.2. 필수 노드 유사성 규칙 구성

노드에 __Pod__를 예약하려면 먼저 필수 규칙을 충족해야 합니다.

프로세스

다음 단계에서는 하나의 노드 및 스케줄러에서 해당 노드에 배치해야 하는 하나의 __Pod__를 생성하는 간단한 구성을 보여줍니다.

1. *__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 Pod를 예약하려면 노드의 라벨과 동일한 key 및 values 매개변수를 사용합니다.*

(4) *연산자 를 지정합니다. 연산자는 In, NotIn, Exists 또는 DoesNotExist일 수 있습니다. 예를 들어 노드에 라벨이 있어야 하는 경우 연산자 In을 사용합니다.*

b. *Pod를 생성합니다.*

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

4.3.3. 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. *특정 라벨을 사용하여 **Pod**를 생성합니다.*

   a. *다음 콘텐츠를 사용하여 **YAML** 파일을 생성합니다.*

참고

*예약된 **Pod**에 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 **1~100**으로 지정합니다. 가중치가 높은 노드가 우선합니다.*

(4)

5

*연산자 를 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
```

4.3.4. 노드 유사성 규칙 샘플

다음 예제에서는 노드 유사성을 보여줍니다.

4.3.4.1. 일치하는 라벨이 있는 노드 유사성

*다음 예제에서는 일치하는 라벨이 있는 노드 및 **Pod**의 노드 유사성을 보여줍니다.*

- *Node1 노드에는 라벨 zone:us가 있습니다.*

```
$ oc label node node1 zone=us
```

작은 정보

다음 YAML을 적용하여 레이블을 추가할 수도 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  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
```

4.3.4.2. 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 YAML을 적용하여 레이블을 추가할 수도 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

```
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
#...
```

- *pod-s1 Pod는 Node1에 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. 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.

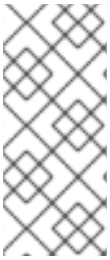
관리자는 요청 및 제한을 통해 노드의 리소스 과다 할당을 허용하고 관리할 수 있습니다. 스케줄러는 요청을 사용하여 컨테이너를 예약하고 최소 서비스 보장 기능을 제공합니다. 제한은 노드에서 사용할 수 있는 컴퓨팅 리소스의 양을 제한합니다.

4.4.1. 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 Dedicated 관리자는 개발자 컨테이너에 설정된 요청과 제한 사이의 비율을 재정의하도록 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.

4.4.2. 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 안됩니다.

이 동작을 보장하기 위해 **OpenShift Dedicated**는 **vm.overcommit_memory** 매개변수를 **1** 로 설정하여 기본 운영 체제 설정을 재정의하여 커널이 항상 메모리를 오버 커밋하도록 구성합니다.

OpenShift Dedicated는 **vm.panic_on_oom** 매개변수를 **0** 으로 설정하여 메모리 부족 시 커널이 패닉 상태가 되지 않도록 구성합니다. **0**으로 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 있어야하며 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

4.5. 노드 선택기를 사용하여 특정 노드에 POD 배치

노드 선택기는 노드의 사용자 정의 라벨 및 Pod에 지정된 선택기를 사용하여 정의한 키/값 쌍으로 구성

된 맵을 지정합니다.

*노드에서 **Pod**를 실행하려면 노드의 라벨과 동일한 키/값 노드 선택기가 **Pod**에 있어야 합니다.*

4.5.1. 노드 선택기 정보

***Pod**의 노드 선택기와 노드의 라벨을 사용하여 **Pod**가 예약되는 위치를 제어할 수 있습니다. 노드 선택기를 사용하면 **OpenShift Dedicated**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 **5**개로 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 Pod가 예약된 노드를 제어할 수 있습니다.

노드 선택기와 라벨을 사용하려면 먼저 Pod의 일정이 조정되지 않도록 노드에 라벨을 지정한 다음 노드 선택기를 Pod에 추가합니다.

참고

예약된 기존 Pod에 노드 선택기를 직접 추가할 수 없습니다. 배포 구성과 같이 Pod를 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
    topology.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: ''
    topology.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    node.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    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 생성하면 OpenShift Dedicated에서 기본 노드 선택기를 Pod에 추가하고 라벨이 일치하는 노드에 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
#...
```

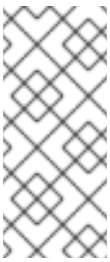
노드 선택기가 있는 **Pod** 오브젝트의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector:
    region: east
#...
```

예제 클러스터에서 예제 **Pod** 사양을 사용하여 **Pod**를 생성하면 **Pod**가 클러스터 수준 노드 선택기와 함께 생성되어 라벨이 지정된 노드에 예약됩니다.

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>
```



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 Dedicated에서 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 **Pod**를 생성하면 **Pod**가 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 Pod가 생성되거나 예약되지 않습니다. 예를 들어 다음 Pod를 예제 프로젝트에 배포하면 Pod가 생성되지 않습니다.

노드 선택기가 유효하지 않은 Pod 오브젝트의 예

```
apiVersion: v1
kind: Pod
metadata:
  name: west-region
#...
spec:
  nodeSelector:
    region: west
#...
```

4.5.2. 노드 선택기를 사용하여 Pod 배치 제어

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

노드, 컴퓨팅 머신 세트 또는 머신 구성에 라벨을 추가합니다. 컴퓨팅 시스템 세트에 레이블을 추가하면 노드 또는 머신이 중단되면 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 **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 YAML**의 **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
# ...
```

프로세스

- *__Pod__에 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
# ...
```

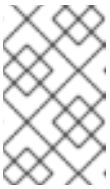
1

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.

4.6. 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.

4.6.1. 사용 사례 예

- 관리자는 내 워크로드가 자동으로 **2~15**개의 **Pod**로 확장되기를 바랍니다. 단일 장애 지점을 방지하기 위해 두 개의 **Pod**만 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.

4.6.2. 중요한 고려 사항

- **OpenShift Dedicated** 클러스터의 **Pod**는 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 **Pod**에 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 Dedicated** 동작이 혼동되고 문제 해결이 더 어려워질 수 있습니다. 토폴로지 도메인의 모든 노드에 일관되게 레이블이 지정되도록 하여 이 문제를 방지할 수 있습니다. **OpenShift Dedicated**는 **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 **Pod**만 일치하고 함께 그룹화됩니다.

- 여러 **Pod** 토폴로지 분배 제약 조건을 지정할 수 있지만 서로 충돌하지 않도록 해야 합니다. **Pod**를 배치하려면 모든 **Pod** 토폴로지 분배 제약 조건을 충족해야 합니다.

4.6.3. 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.

4.6.3.1. skew 계산 예

세 개의 영역(**A, B, C**)이 있으며 이러한 영역에 **Pod**를 균등하게 배포하려고 합니다. 영역 **A**에 **Pod**가 **5**개, 영역 **B**에 **3**개의 **Pod**가 있고 영역 **C**에 포드가 **2**개 있는 경우 각 영역에 현재 **Pod** 수에서 예약된 **Pod** 수가 가장 낮은 도메인의 **Pod** 수를 차감할 수 있습니다. 즉, **A** 영역의 스큐는 **3**이고, 영역 **B**의 스큐는 **1**이고, **C** 영역의 스큐는 **0**입니다.

4.6.3.2. maxSkew 매개변수

maxSkew 매개변수는 두 토폴로지 도메인 간의 **Pod** 수에서 허용되는 최대 차이점 또는 **skew**를 정의합니다. **maxSkew** 가 **1** 로 설정된 경우 토폴로지 도메인의 **Pod** 수는 다른 도메인과 **1** 이상 달라서는 안 됩니다. **skew**가 **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 **C**에 배치하여 스큐를 줄이고 더 균형 있는 배포를 수행하여 토폴로지 도메인이 **1**의 스큐를 초과하지 않도록 합니다.

4.6.4. 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 **Pod**를 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 **1**이고 값 **0**은 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 **Pod**를 처리하는 방법입니다. 기본값은 스케줄러에 **Pod**를 예약하지 않도록 지시하는 **DoNotSchedule**입니다. **Pod**를 계속 예약하기 위해 **ScheduleAnyway**로 설정하지만 스케줄러는 클러스터의 불균형이 더 심해지지 않도록 불일치에 따라 우선순위를 부여합니다.

(4) 이 라벨 선택기와 일치하는 **Pod**는 제약 조건을 충족하기 위해 분배될 때 계산되고 그룹으로 인식됩니다. 라벨 선택기를 지정해야 합니다. 그러지 않으면 일치하는 **Pod**가 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

이전 예제에서는 하나의 Pod 토폴로지 분배 제약 조건을 사용하여 Pod 사양을 정의합니다. region 레이블이 지정된 Pod와 일치합니다. us-east 는 영역에 배포되고, 1 의 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

*이전 예제에서는 두 개의 **Pod** 토폴로지 분배 제약 조건이 있는 **Pod** 사양을 정의합니다. 둘 다 **region: us-east** 레이블이 지정된 **Pod**와 일치하고, 불일치를 **1** 로 지정하고, 이러한 요구 사항을 충족하지 않는 경우 **Pod**를 예약하지 않습니다.*

*첫 번째 제약 조건에서는 사용자 정의 라벨 **node**를 기반으로 **Pod**를 배포하고, 두 번째는 제약 조건에서는 사용자 정의 라벨 **rack**을 기반으로 **Pod**를 배포합니다. **Pod**를 예약하려면 두 제약 조건을 모두 충족해야 합니다.*

5장. 작업 및 DAEMONSET 사용

5.1. 데몬 세트를 사용하여 노드에서 자동으로 백그라운드 작업 실행

관리자는 데몬 세트를 생성하고 사용하여 OpenShift Dedicated 클러스터의 특정 또는 모든 노드에서 Pod의 복제본을 실행할 수 있습니다.

데몬 세트를 사용하면 모든(또는 일부) 노드에서 하나의 Pod 복사본을 실행할 수 있습니다. 노드가 클러스터에 추가되면 Pod도 해당 클러스터에 추가됩니다. 노드가 클러스터에서 제거되면 해당 Pod도 가비지 컬렉션을 통해 제거됩니다. 데몬 세트를 삭제하면 데몬 세트에서 생성한 Pod가 정리됩니다.

데몬 세트를 사용하여 공유 스토리지를 생성하거나 클러스터의 모든 노드에서 로깅 Pod를 실행하거나 모든 노드에 모니터링 에이전트를 배포할 수 있습니다.

보안상의 이유로 클러스터 관리자와 프로젝트 관리자는 데몬 세트를 생성할 수 있습니다.

데몬 세트에 대한 자세한 내용은 Kubernetes 설명서를 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 Pod가 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 Pod가 실행되는 노드는 Kubernetes 스케줄러에서 선택합니다. 그러나 데몬 세트 Pod는 데몬 세트 컨트롤러에서 생성하고 예약합니다. 이 경우 다음과 같은 문제가 발생할 수 있습니다.

- 일관되지 않는 Pod 동작: 예약 대기 중인 정상 Pod가 생성되고 보류 중 상태이지만 데몬 세트 Pod는 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 Dedicated에서 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.template는 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-selector를 빈 문자열로 설정하여 네임스페이스에서 기본 프로젝트 수준 노드 선택기를 비활성화하십시오.*

```
$ oc patch namespace myproject -p \
    '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

작은 정보

또는 다음 YAML을 적용하여 네임스페이스의 기본 프로젝트 수준 노드 선택기를 비활성화할 수 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/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. ***Pod**가 생성되었고 각 노드에 **Pod** 복제본이 있는지 확인하려면 다음을 수행합니다.*

   a. ***daemonset Pod**를 찾습니다.*

   ```
   $ oc get pods
   ```

   출력 예

   ```
   hello-daemonset-cx6md   1/1     Running   0       2m
   hello-daemonset-e3md9   1/1     Running   0       2m
   ```

   b. ***Pod**를 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 **Pod**를 추가하고 새 라벨과 일치하지 않는 노드에서 **Pod**를 삭제합니다.

데몬 세트를 업데이트하려면 이전 복제본 또는 노드를 삭제하여 새 **Pod** 복제본을 강제로 생성합니다.

5.2. 작업을 사용하여 POD에서 작업 실행

작업은 **OpenShift Dedicated** 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 Pod가 완료되어야 합니다.*

(3) *작업을 실행할 수 있는 최대 기간입니다.*

(4) *작업 재시도 횟수입니다.*

(5) *컨트롤러에서 생성하는 Pod에 사용할 템플릿입니다.*

(6)

추가 리소스

- Kubernetes 문서의 작업

5.2.1. 작업 및 cron 작업 이해

작업에서는 작업의 전반적인 진행률을 추적하고 활성 상태에 있거나 성공 또는 실패한 Pod에 대한 정보를 사용하여 해당 상태를 업데이트합니다. 작업을 삭제하면 작업에서 생성한 Pod가 모두 정리됩니다. 작업은 Kubernetes API의 일부이며 다른 오브젝트 유형과 같이 oc 명령으로 관리할 수 있습니다.

OpenShift Dedicated에서 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:
  - Pod가 실패하지 않는 한 하나의 Pod만 시작하는 작업입니다.
  - Pod가 성공적으로 종료되면 작업이 완료됩니다.
- 완료 횟수가 고정된 병렬 작업:
  - 여러 Pod를 시작하는 작업입니다.
  - 이 작업은 전체 작업을 나타내며 1에서 completions 값 사이의 각 값에 대해 하나의 성공적인 Pod가 있을 때 완료됩니다.

- *작업 큐가 있는 병렬 작업:*
  - *지정된 **Pod**에 여러 병렬 작업자 프로세스가 있는 작업입니다.*
  - ***OpenShift Dedicated**는 **Pod**를 조정하여 각 작업을 확인하거나 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 패턴을 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 시간에 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 Dedicated**에서 생성하는 **Pod**를 설명합니다.
- **parallelism** 매개변수: 작업을 실행해야 하는 임의의 시점에 병렬로 실행되는 **Pod** 수를 지정합니다.
  - 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 **1**입니다.
- **completions** 매개변수: 작업을 완료하는 데 필요한 성공적인 **Pod** 완료 횟수를 지정합니다.
  - 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 **1**입니다.
  - 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 Dedicated**에서 작업이 종료됩니다.

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 **6**분으로 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**).

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 늘어나는 것은 아닙니다. 또한 **kubelet**은 동일한 노드에서 **Pod**를 시작하여 실패한 작업을 재시도합니다.

5.2.2. 작업 생성

작업 오브젝트를 생성하여 **OpenShift Dedicated**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: 작업이 병렬로 실행해야 하는 **Pod** 복제본 수를 지정합니다. 기본값은 **1** 입니다.

- 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 **1**입니다.

2

선택 사항: 작업이 완료된 것으로 표시하는 데 필요한 성공적인 **Pod** 완료 횟수를 지정합니다.

- 비병렬 작업의 경우 설정되지 않은 상태로 둡니다. 설정되지 않은 경우 기본값은 **1**입니다.
- 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 **6**입니다.

5

컨트롤러에서 생성하는 **Pod**에 사용할 템플릿을 지정합니다.

6

Pod 재시작 정책을 지정합니다.

- **Never**. 작업을 재시작하지 않습니다.
- **OnFailure**. 실패하는 경우에만 작업을 재시작합니다.
- **Always** 작업을 항상 재시작합니다.

OpenShift Dedicated에서 실패한 컨테이너에 재시작 정책을 사용하는 방법에 대한 자세한 내용은 **Kubernetes** 문서 의 예제 상태를 참조하십시오.

2. 작업을 생성합니다.

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

참고

oc create job을 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 Dedicated에서 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
  concurrencyPolicy: "Replace"     2
  startingDeadlineSeconds: 200     3
  suspend: true                    4
  successfulJobsHistoryLimit: 3    5
  failedJobsHistoryLimit: 1        6
  jobTemplate:                     7
    spec:
      template:
        metadata:
          labels:                  8
            parent: "cronjobpi"
        spec:
          containers:
          - name: pi
```

```
    image: perl
    command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
  restartPolicy: OnFailure 9
```

1

cron 형식에 지정된 작업의 스케줄입니다. 이 예제에서는 작업은 분마다 실행됩니다.

2

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.

- Allow는 cron 작업이 동시에 실행되도록 허용합니다.
- Forbid는 동시 실행을 금지하고 이전 실행이 아직 완료되지 않은 경우 다음 실행을 건너뜁니다.
- Replace는 현재 실행 중인 작업을 취소하고 새 작업으로 교체합니다.

3

어떠한 이유로 예약된 시간을 놓치는 경우 작업을 시작하는 선택적 데드라인(초)입니다. 누락된 작업 실행은 실패한 작업으로 간주됩니다. 지정하지 않으면 데드라인이 없습니다.

4

cron 작업의 중지를 허용하는 선택적 플래그입니다. true로 설정하면 이후의 모든 실행이 일시 중지됩니다.

5

유지해야 하는 성공적으로 완료한 작업의 수입니다(기본값: 3).

6

유지해야 하는 실패한 작업의 수입니다(기본값: 1).

7

작업 템플릿입니다. 이 템플릿은 작업 예제와 유사합니다.

8

이 cron 작업에서 생성한 작업에 라벨을 설정합니다.

9

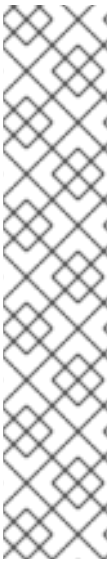
Pod의 재시작 정책입니다. 이 정책은 작업 컨트롤러에 적용되지 않습니다.

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. OPENSHIFT DEDICATED 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      v1.29.4
node1.example.com    Ready    worker   7h      v1.29.4
node2.example.com    Ready    worker   7h      v1.29.4
```

다음 예제는 하나의 비정상 노드가 있는 클러스터입니다.

```
$ oc get nodes
```

출력 예

```
NAME                 STATUS                     ROLES    AGE     VERSION
master.example.com   Ready                      master   7h      v1.29.4
node1.example.com    NotReady,SchedulingDisabled worker  7h      v1.29.4
node2.example.com    Ready                      worker   7h      v1.29.4
```

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  v1.29.4   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://1.29.4-30.rhaos4.10.gitf2f339d.el8-dev
node1.example.com    Ready    worker   72m   v1.29.4   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://1.29.4-30.rhaos4.10.gitf2f339d.el8-dev
node2.example.com    Ready    worker   164m  v1.29.4   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://1.29.4-30.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

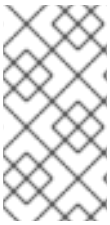
```
NAME                STATUS   ROLES    AGE     VERSION
node1.example.com   Ready    worker   7h      v1.29.4
```

- *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
```



참고

다음 예제에는 AWS의 OpenShift Dedicated와 관련된 몇 가지 값이 포함되어 있습니다.

출력 예

```
Name:               node1.example.com 1
Roles:              worker 2
Labels:             kubernetes.io/os=linux
                    kubernetes.io/hostname=ip-10-0-131-14
                    kubernetes.io/arch=amd64 3
                    node-role.kubernetes.io/worker=
                    node.kubernetes.io/instance-type=m4.large
                    node.openshift.io/os_id=rhcos
                    node.openshift.io/os_version=4.5
                    region=east
                    topology.kubernetes.io/region=us-east-1
                    topology.kubernetes.io/zone=us-east-1a
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://1.29.4-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
 Kubelet Version:            v1.29.4
 Kube-Proxy Version:         v1.29.4
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-ebs  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

노드에 적용되는 라벨입니다.

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.

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.

6.1.2. 클러스터의 노드에 있는 Pod 나열

*특정 노드의 모든 **Pod**를 나열할 수 있습니다.*

프로세스

- *하나 이상의 노드에 있는 모든 **Pod** 또는 선택한 **Pod**를 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 **Pod**를 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 **Pod**를 포함하여 특정 노드의 모든 **Pod**를 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. NODE TUNING OPERATOR 사용

Node Tuning Operator에 대해 알아보고, **Node Tuning Operator**를 사용하여 **Tuned** 데몬을 오케스트레이션하고 노드 수준 튜닝을 관리하는 방법도 알아봅니다.

목적

Node Tuning Operator는 **TuneD** 데몬을 오케스트레이션하여 노드 수준 튜닝을 관리하고 **Performance Profile** 컨트롤러를 사용하여 대기 시간이 짧은 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 Dedicated**의 컨테이너화된 **TuneD** 데몬을 **Kubernetes** 데몬 세트로 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.

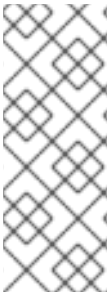
컨테이너화된 **TuneD** 데몬을 통해 적용되는 노드 수준 설정은 프로필 변경을 트리거하는 이벤트 시 또는 컨테이너화된 **TuneD** 데몬이 종료 신호를 수신하고 처리하여 정상적으로 종료될 때 롤백됩니다.

Node Tuning Operator는 **Performance Profile** 컨트롤러를 사용하여 **OpenShift Dedicated** 애플리케이션에 대한 짧은 대기 시간 성능을 달성하기 위해 자동 튜닝을 구현합니다.

클러스터 관리자는 다음과 같은 노드 수준 설정을 정의하도록 성능 프로필을 구성합니다.

- 커널을 **kernel-rt**로 업데이트합니다.
- 하우스키핑을 위한 **CPU** 선택.
- 실행 중인 워크로드를 위한 **CPU** 선택.

Node Tuning Operator는 버전 **4.1** 이상에서 표준 **OpenShift Dedicated** 설치의 일부입니다.



참고

이전 버전의 **OpenShift Dedicated**에서는 **Performance Addon Operator**를 사용하여 **OpenShift** 애플리케이션에 대한 대기 시간이 짧은 성능을 달성하기 위해 자동 튜닝을 구현했습니다. **OpenShift Dedicated 4.11** 이상에서 이 기능은 **Node Tuning Operator**의 일부입니다.

6.2.1. Node Tuning Operator 사양 예에 액세스

이 프로세스를 사용하여 **Node Tuning Operator** 사양 예에 액세스하십시오.

프로세스

- 다음 명령을 실행하여 **Node Tuning Operator** 사양 예제에 액세스합니다.

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

기본 **CR**은 **OpenShift Dedicated** 플랫폼에 대한 표준 노드 수준 튜닝을 제공하기 위한 것이며 **Operator** 관리 상태를 설정하는 경우에만 수정할 수 있습니다. **Operator**는 기본 **CR**에 대한 다른 모든 사용자 정의 변경사항을 덮어씁니다. 사용자 정의 튜닝의 경우 고유한 **Tuned CR**을 생성합니다. 새로 생성된 **CR**은 노드 또는 **Pod** 라벨 및 프로필 우선 순위에 따라 **OpenShift Dedicated** 노드에 적용되는 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 CR**은 **Pod** 라벨이 일치되지 않은 상태로 제공됩니다. **Pod** 라벨이 일치된 상태로 사용자 정의 프로필이 생성되면 해당 시점에 이 기능이 활성화됩니다. **Pod** 레이블 기능은 **Node Tuning Operator**의 향후 버전에서 더 이상 사용되지 않습니다.*

6.2.2. 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 **Operator**의 네임스페이스에 여러 **CR**로 존재할 수 있습니다. 새로운 **CR**의 존재 또는 오래된 **CR**의 삭제는 **Operator**에서 탐지됩니다. 기존의 모든 사용자 정의 튜닝 사양이 병합되고 컨테이너화된 **TuneD** 데몬의 해당 오브젝트가 업데이트됩니다.*

관리 상태

*Operator 관리 상태는 기본 **Tuned CR**을 조정하여 설정됩니다. 기본적으로 **Operator**는 **Managed** 상태이며 기본 **Tuned CR**에는 **spec.managementState** 필드가 없습니다. **Operator** 관리 상태에 유효한 값은 다음과 같습니다.*

- *__Managed:__ 구성 리소스가 업데이트되면 **Operator**가 해당 피연산자를 업데이트합니다.*
- *__Unmanaged:__ **Operator**가 구성 리소스에 대한 변경을 무시합니다.*
- *__Removed:__ **Operator**가 프로비저닝한 해당 피연산자 및 리소스를 **Operator**가 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(**0**이 가장 높은 우선 순위임).*

6

*일치에 적용할 **TuneD** 프로필입니다. 예를 들어 **tuned_profile_1**이 있습니다.*

7

선택적 피연산자 구성입니다.

8

***TuneD** 데몬에 대해 디버깅을 켜거나 끕니다. **on** 또는 **false** 의 경우 옵션은 **true** 입니다. 기본값은 **false**입니다.*

9

***TuneD** 데몬의 경우 **reapply_sysctl** 기능을 켭니다. **on** 및 **false** 의 경우 옵션은 **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 node라고 가정합니다.

4

선택사항 <match> 목록입니다.

<match>를 생략하지 않으면 모든 중첩 <match> 섹션도 true로 평가되어야 합니다. 생략하면 false로 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 true로 평가됩니다. 따라서 이 목록이 논리 OR 연산자 역할을 합니다.

machineConfigLabels가 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 machineConfigLabels는 논리 OR 연산자로 연결됩니다. match 항목은 단락 방식으로 먼저 평가됩니다. 따라서 true로 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.

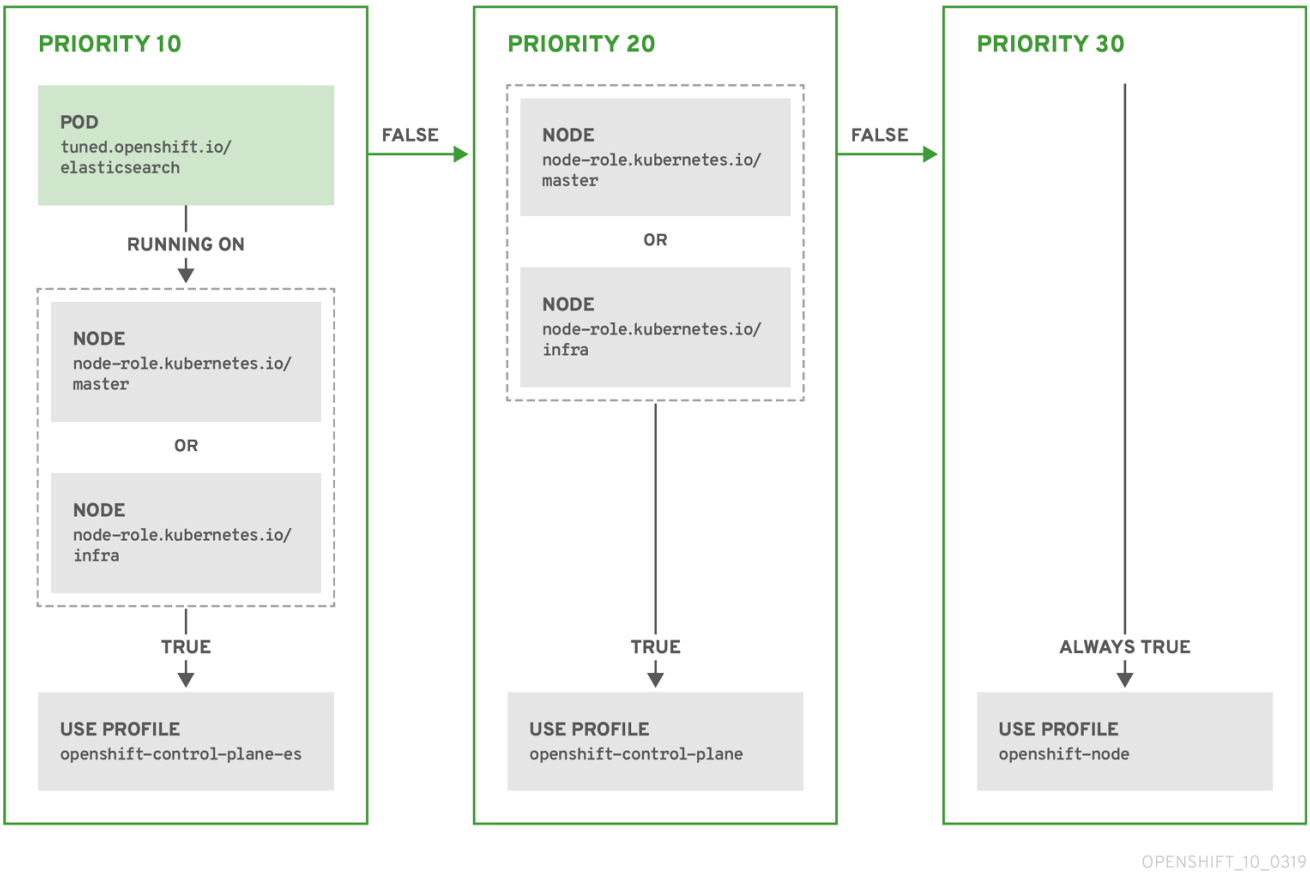
예: 노드 또는 Pod 라벨 기반 일치

```
- 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 **CR**은 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 **Pod**가 있는지 확인합니다. 없는 경우 전체 **<match>** 섹션이 **false**로 평가됩니다. 라벨이 있는 **Pod**가 있는 경우 **<match>** 섹션을 **true**로 평가하려면 노드 라벨도 **node-role.kubernetes.io/master** 또는 **node-role.kubernetes.io/infra**여야 합니다.*

*우선 순위가 **10**인 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 Pod**가 **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 **30**입니다. 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.*



예: 머신 구성 풀 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 Dedicated** 클러스터의 지정된 클라우드 공급자에 특별히 조정된 **TuneD** 프로필을 편리하게 할당할 수 있습니다. 이 작업은 노드를 머신 구성 풀에 추가하거나 노드를 그룹화하지 않고 수행할 수 있습니다.

이 기능은 **<cloud-provider> ://<cloud-provider-specific-id>** 형식의 **spec.provider ID** 노드 오브젝트 값을 활용하고 **NTO** 피연산자 컨테이너의 **< cloud-provider>** 값으로 **/var/lib/ocp-tuned/provider** 파일을 씁니다. 그런 다음 이 파일의 내용은 해당 프로필이 존재하는 경우 **TuneD**에서 **provider-<cloud-provider >** 프로필을 로드하는 데 사용됩니다.

이제 **openshift -control-plane** 및 **openshift-node** 프로필에서 설정을 상속하는 **openshift** 프로파일이 조건부 프로필 로드를 사용하여 이 기능을 사용하도록 업데이트되었습니다. **NTO** 및 **TuneD**에는 현재 클라우드 공급자별 프로필이 포함되어 있지 않습니다. 그러나 모든 **Cloud** 공급자별 클러스터 노드에 적용할 사용자 지정 프로필 **provider-<cloud- provider>**를 생성할 수 있습니다.

GCE 클라우드 공급자 프로파일의 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 프로필이 덮어씁니다.

6.2.3. 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/ocp-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 Dedicated 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 ^ {} \;
```

6.2.4. 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***

이러한 플러그인 중 일부에서 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

- ***Available TuneD Plugins***
- ***TuneD 시작하기***

6.3. 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 실패로 인해 데이터 손실, 손상 또는 둘 다 위험이 있습니다. 워크로드 복구(수정 이라고 함) 및 노드 복구를 시작하기 전에 펜싱 이라는 노드를 격리하는 것이 중요합니다.*

노드 수정, 펜싱 및 유지 관리에 대한 자세한 내용은 ***Workload Availability for Red Hat OpenShift*** *설명서를 참조하십시오.*

7장. 컨테이너 작업

7.1. 컨테이너 이해

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

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

Linux 커널은 수년간 컨테이너 기술을 위한 기능을 통합해 왔습니다. **OpenShift Dedicated** 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 캐시에서 많은 양의 커널 메모리가 발생하면 **OpenShift Dedicated** 컨테이너가 구성된 메모리 제한을 초과하여 컨테이너가 종료될 수 있습니다.

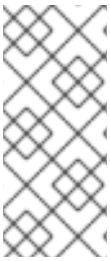
커널 메모리 문제로 인한 컨테이너 손실을 방지하려면 컨테이너에서 메모리를 충분히 요청해야 합니다. 다음 공식을 사용하여 **kmem_cache** 에서 사용하는 메모리 양을 추정할 수 있습니다. 여기서 **nproc**은 **nproc** 명령에서 보고하는 처리 단위의 수입니다. 컨테이너 요청의 하한은 이 값에 컨테이너 메모리 요구 사항을 더한 값이어야 합니다.

> **$(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 엔진에서는 하위 수준 컨테이너 런타임 이라고도 하는 컨테이너 런타임 을 사용하여 컨테

이너 배포 및 운영에 필요한 구성 요소를 실행하고 관리합니다. 컨테이너 엔진 또는 컨테이너 런타임과 상호 작용할 필요가 없습니다.



참고

OpenShift Dedicated 설명서에서는 컨테이너 런타임 이라는 용어를 사용하여 하위 수준 컨테이너 런타임을 참조합니다. 기타 설명서에서는 컨테이너 엔진을 컨테이너 런타임으로 참조할 수 있습니다.

OpenShift Dedicated는 **CRI-O**를 컨테이너 엔진으로 사용하고 **C** 또는 **crun**을 컨테이너 런타임으로 사용합니다. 기본 컨테이너 런타임은 **runC**입니다.

7.2. POD를 배포하기 전에 INIT CONTAINER를 사용하여 작업 수행

OpenShift Dedicated는 애플리케이션 컨테이너보다 먼저 실행되고 앱 이미지에 없는 유틸리티 또는 설정 스크립트를 포함할 수 있는 특수 컨테이너인 **init** 컨테이너를 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 Container는 다음 컨테이너를 시작하기 전에 성공적으로 완료해야 합니다. 이를 위해 Init Container에서는 몇 가지 사전 조건 집합이 충족될 때까지 앱 컨테이너의 시작을 차단하거나 지연할 수 있는 쉬운 방법을 제공합니다.

예를 들어 다음은 Init Container를 사용할 수 있는 몇 가지 방법입니다.

- 다음과 같은 쉘 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 하위 API에서 원격 서버에 이 Pod를 등록합니다.

```
$ curl -X POST
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d
'instance=$()&ip=$()'
```

- sleep 60과 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 Jinja를 사용하여 기본 앱 구성 파일을 생성합니다.

자세한 내용은 Kubernetes 설명서를 참조하십시오.

7.2.2. Init Container 생성

다음 예제에서는 두 개의 Init Container가 있는 간단한 Pod를 간략하게 설명합니다. 첫 번째 컨테이너는 myservice를 기다리고 두 번째 컨테이너는 mydb를 기다립니다. 두 컨테이너가 모두 완료되면 Pod가 시작됩니다.

프로세스

1. Init Container의 Pod를 생성합니다.

a.

다음과 유사한 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: myapp-container
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  initContainers:
  - name: init-myservice
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for
myservice; sleep 2; done;']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  - name: init-mydb
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep
2; done;']
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

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. ***Pod**를 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

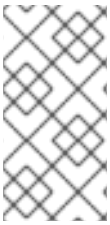
사용하여 Pod의 컨테이너에서 사용하는 데이터를 유지할 수 있습니다. 볼륨은 Pod 수명 동안 데이터가 저장되는 Pod의 컨테이너에 액세스할 수 있는 디렉터리입니다.

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 확인하기 위해 오류가 있는 경우 이를 복구하기 위해 OpenShift Dedicated는 mount 유틸리티 전에 fsck 유틸리티를 호출합니다. 이러한 작업은 볼륨을 추가하거나 기존 볼륨을 업데이트할 때 수행됩니다.

가장 간단한 볼륨 유형은 단일 머신의 임시 디렉터리인 emptyDir입니다. 관리자는 Pod에 자동으로 연결된 영구 볼륨을 요청할 수도 있습니다.



참고

클러스터 관리자가 FSGroup 매개변수를 활성화하면 Pod의 FSGroup을 기반으로 한 할당량에 따라 emptyDir 볼륨 스토리지가 제한될 수 있습니다.

7.3.2. OpenShift Dedicated CLI를 사용하여 볼륨 작업

CLI 명령 oc set volume을 사용하여 복제 컨트롤러 또는 배포 구성과 같은 Pod 템플릿이 있는 오브젝트의 볼륨 및 볼륨 마운트를 추가하고 제거할 수 있습니다. Pod의 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
<object_type> <name>	유형 ***<object_type>***의 ***<name>***을 선택합니다.	**deploymentConfig registry**
<object_type>/<name>	유형 ***<object_type>***의 ***<name>***을 선택합니다.	**deploymentConfig/registry**
<object_type>*--selector=*<object_label_selector>	지정된 라벨 선택기와 일치하는 유형 ***<object_type>***의 리소스를 선택합니다.	**deploymentConfig--selector="name=registry"**
<object_type> **--all**	유형 ***<object_type>***의 모든 리소스를 선택합니다.	**deploymentConfig --all**
-f 또는 **--filename=*<file_name>***	리소스를 편집하는 데 사용할 파일 이름, 디렉터리 또는 URL입니다.	**-f registry-deployment-config.json**

작업

oc set volume 명령의 operation 매개변수에 --add 또는 --remove를 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. **/host**를 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. **/host**를 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 수도 있습니다.*

*예 **7.1.** 추가된 볼륨이 있는 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 YAML을 적용하여 볼륨을 추가할 수도 있습니다.

예 7.2. 볼륨 및 시크릿이 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 **pvc1**인 기존 영구 볼륨 **v1**을 디스크의 배포 구성 **dc.json**에 추가하려면 **/data**의 컨테이너 **c1**에 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 YAML을 적용하여 볼륨을 추가할 수도 있습니다.

예 7.3. 영구 볼륨이 추가된 샘플 배포 구성

```
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) *'pvc1이라는 영구 볼륨 클레임을 추가합니다.*

(2) *컨테이너 마운트 경로를 추가합니다.*

- *모든 복제 컨트롤러에서 수정 버전이 5125c45f9f563인 Git 리포지토리 https://github.com/namespace1/project1을 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 YAML을 적용하여 볼륨을 교체할 수 있습니다.

예 7.4. pvc1이라는 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 v1의 /opt로 변경하려면 다음을 실행합니다.

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

작은 정보

다음 YAML을 적용하여 마운트 지점을 변경할 수도 있습니다.

예 7.5. 마운트 지점을 선택하도록 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.

프로세스

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 d1에서 볼륨 v1을 제거하려면 다음을 실행합니다.*

```
$ oc set volume dc/d1 --remove --name=v1
```

- *DeploymentConfig 오브젝트 d1의 컨테이너에서 참조하지 않는 경우 d1의 컨테이너 c1에서 볼륨 v1을 마운트 해제하고 볼륨 v1을 제거하려면 다음을 실행합니다.*

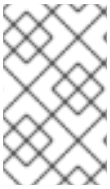
```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- *복제 컨트롤러 r1의 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 **Pod**에서 다양한 용도로 하나의 볼륨을 공유하도록 볼륨을 구성할 수 있습니다.



참고

예약된 기존 **Pod**에 **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:
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
```

```
    subPath: mysql (1)
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
- name: php
  image: php
  volumeMounts:
  - mountPath: /var/www/html
    name: site-data
    subPath: html (2)
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
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 있습니다.

- 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 **Pod**의 보안 컨텍스트에 **RunAsUser** 권한이 설정되면 컨테이너 사용자 소유권을 포함하여 예상 파일에 올바른 권한이 설정되어 있습니다. 그러나 **Windows** 동등한 **RunAsUsername** 권한이 **Windows Pod**에 설정된 경우 **kubelet**은 예상 볼륨의 파일에 대한 소유권을 올바르게 설정할 수 없습니다.

따라서 **Windows Pod**의 보안 컨텍스트에서 설정된 **RunAsUsername** 권한은 **OpenShift Dedicated**에서 실행되는 **Windows** 예상 볼륨에 대해 적용되지 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. **Pod**에 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts: 1
    - name: all-in-one
      mountPath: "/projected-volume" 2
      readOnly: true 3
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  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

특정 예상에 대한 모드 설정

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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
```

```
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    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
          items:
            - key: username
              path: my-group/data
      - configMap:
          name: myconfigmap
          items:
            - key: config
              path: my-group/data
```

볼륨 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 볼륨 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 **Pod**를 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

```
MWYyZDFlMmU2N2Rm
```

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: MWYyZDFlMmU2N2Rm
  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: MWYyZDFlMmU2N2Rm
  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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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

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 쉘을 엽니다.

```
$ oc exec -it <pod> <command>
```

예를 들면 다음과 같습니다.

```
$ oc exec -it test-projected-volume -- /bin/sh
```

5. 쉘에서 **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 API 는 **OpenShift Dedicated**에 결합하지 않고 컨테이너에서 **API** 오브젝트에 대한 정보를 사용할 수 있는 메커니즘입니다. 이러한 정보에는 **Pod** 이름, 네임스페이스, 리소스 값이 포함됩니다. 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 메서드에 따라 컨테이너에서 다음을 사용할 수 있습니다.*

- ***Pod** 이름*
- ***Pod** 프로젝트/네임스페이스*
- ***Pod** 주석*

- *Pod 라벨*

볼륨 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 Pod의 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  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 Pod를 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 API를 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 API에서 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 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: 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.

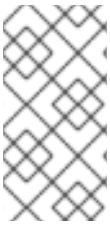
pod.yaml 파일에서 Pod를 생성합니다.

```
$ 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 Pod를 생성합니다.*

a. *다음과 유사한 pod.yaml 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  restartPolicy: Never
# ...
```

b. *pod.yaml 파일에서 Pod를 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 API를 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 Pod를 생성합니다.

   a. 다음과 유사한 pod.yaml 파일을 생성합니다.

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  restartPolicy: Always
# ...
```

b. *pod.yaml 파일에서 Pod를 생성합니다.*

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

검증

- *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 Pod를 생성합니다.*

a. *다음과 유사한 pod.yaml 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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)
    securityContext:
     allowPrivilegeEscalation: false
     capabilities:
       drop: [ALL]
  restartPolicy: Never
# ...
```

b. *pod.yaml 파일에서 Pod를 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 Pod를 생성합니다.*

a. *다음과 유사한 pod.yaml 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

```
  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)
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  restartPolicy: Never
# ...
```

b. *pod.yaml 파일에서 Pod를 생성합니다.*

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

검증

- *컨테이너의 로그에 MY_NEW_ENV 값이 있는지 확인합니다.*

```
$ oc logs -p dapi-env-test-pod
```

7.6. OPENSHIFT DEDICATED 컨테이너에 또는 OPENSHIFT DEDICATED 컨테이너에 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 Pod에서 및 Pod로 데이터베이스 아카이브를 복사하는 유용한 툴입니다. 실행 중인 Pod에서 소스 파일의 핫 리로드를 지원하는 경우 개발 디버깅을 위해 oc rsync를 사용하여 소스 코드 변경 사항을 실행 중인 Pod에 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 디렉터리를 가리켜야 합니다. 개별 파일은 지원되지 않습니다.

Pod 디렉터리를 지정할 때는 디렉터리 이름 앞에 Pod 이름을 붙여야 합니다.

```
<pod name>:<dir>
```

디렉터리 이름이 경로 구분자(/)로 끝나는 경우 디렉터리의 콘텐츠만 대상에 복사됩니다. 그러지 않으면 디렉터리 및 해당 콘텐츠가 대상에 복사됩니다.

복사 대상 지정

oc rsync 명령의 대상 인수는 디렉터리를 가리켜야 합니다. 해당 디렉터리가 존재하지 않지만 rsync가 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 rsync에 전달되는 인수를 포함하여 oc rsync를 수동으로 반복해서 호출하는 것과 사실상 동일합니다. 따라서 -delete와 같이 oc rsync를 수동으로 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.

- rsync가 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 tar를 사용할 수 없는 경우 복사가 실패합니다.

tar 복사 방법에서는 oc rsync와 동일한 기능을 제공하지 않습니다. 예를 들어 oc rsync는 대상 디렉터리가 존재하지 않는 경우 대상 디렉터리를 생성하고 소스와 대상 간에 다른 파일만 보냅니다.

참고

Windows에서는 oc rsync 명령과 함께 사용할 수 있도록 cwRsync 클라이언트를 설치하고 PATH에 추가해야 합니다.

프로세스

- 로컬 디렉터리를 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 디렉터리를 로컬 디렉터리에 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 rsync보다 적은 수의 명령줄 옵션을 표시합니다. oc rsync에서 사용할 수 없는

표준 rsync 명령줄 옵션(예: --exclude-from=FILE 옵션)을 사용하려는 경우 표준 rsync의 --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 rsh를 원격 쉘 프로그램으로 사용하여 원격 Pod에 연결할 수 있도록 표준 rsync를 구성하고 oc rsync를 실행하는 대신 사용할 수 있습니다.

7.7. OPENSHIFT DEDICATED 컨테이너에서 원격 명령 실행

CLI를 사용하여 OpenShift Dedicated 컨테이너에서 원격 명령을 실행할 수 있습니다.

7.7.1. 컨테이너에서 원격 명령 실행

원격 컨테이너 명령 실행을 위한 지원은 CLI에 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 **URL**에서

- **<node_name>**은 노드의 **FQDN**입니다.
- **<namespace>**는 대상 **Pod**의 프로젝트입니다.
- **<pod>**는 대상 **Pod**의 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 Dedicated에서는 **Pod**로의 포트 전달을 지원합니다.

7.8.1. 포트 전달 이해

CLI를 사용하여 하나 이상의 로컬 포트를 **Pod**로 전달할 수 있습니다. 이 경우 지정된 포트 또는 임의의 포트에서 로컬로 수신 대기하고 **Pod**의 지정된 포트와 데이터를 주고받을 수 있습니다.

포트 전달 기능을 위한 지원은 **CLI**에 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 Dedicated는 클라이언트의 포트 전달 요청을 처리합니다. 요청을 수신하면 OpenShift Dedicated에서 응답을 업그레이드하고 클라이언트가 포트 전달 스트림을 생성할 때까지 기다립니다. OpenShift Dedicated에서 새 스트림을 수신하면 스트림과 Pod 포트 간의 데이터를 복사합니다.

구조적으로 Pod의 포트로 전달할 수 있는 옵션이 있습니다. 지원되는 OpenShift Dedicated 구현은 노드 호스트에서 직접 nsenter 를 호출하여 Pod의 네트워크 네임스페이스에 입력한 다음 socat 을 호출하여 스트림과 Pod 포트 간의 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 Pod로 포트 전달할 수 있습니다.

프로세스

다음 명령을 사용하여 Pod의 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 6000에서 로컬로 수신 대기하고 Pod의 포트 5000 및 6000에서 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 **Pod**의 **5000**으로 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 Version 2(HTTP/2) 를 사용합니다.

클라이언트는 Pod에 대상 포트가 포함된 port 헤더를 사용하여 스트림을 생성합니다. 스트림에 기록된 모든 데이터는 kubelet을 통해 대상 Pod 및 포트로 전달됩니다. 마찬가지로 이렇게 전달된 연결에 대해 Pod에서 전송되는 모든 데이터는 클라이언트의 동일한 스트림으로 다시 전달됩니다.

클라이언트는 포트 전달 요청을 완료하면 모든 스트림, 업그레이드된 연결, 기본 연결을 종료합니다.

8장. 클러스터 작업

8.1. OPENSHIFT DEDICATED 클러스터에서 시스템 이벤트 정보 보기

OpenShift Dedicated의 이벤트는 OpenShift Dedicated 클러스터의 API 오브젝트에 발생하는 이벤트를 기반으로 모델링됩니다.

8.1.1. 이벤트 이해

OpenShift Dedicated에서는 이벤트를 통해 실제 이벤트에 대한 정보를 리소스와 무관한 방식으로 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 Dedicated** 콘솔에서 프로젝트의 이벤트를 보려면 다음을 수행합니다.

  1. **OpenShift Dedicated** 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 Dedicated**의 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.
ExceededGracePeriod	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	Ctr 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 쉐이퍼입니다.
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 디렉터리를 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	Starting OpenShift SDN.
NetworkFailed	Pod의 네트워크 인터페이스가 손실되어 Pod가 중지됩니다.

표 8.12. 네트워크 이벤트(kube-proxy)

이름	설명
NeedPods	서비스 포트 **<serviceName>:<port>**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. OPENSHIFT DEDICATED 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 Tool**은 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 **Pod**의 여러 인스턴스 측면에서 여전히 사용할 수 있는 가용 용량을 추정합니다.*

*또한 **Pod**는 선택 및 유사성 기준에 따라 특정 노드 집합에서만 예약 기능이 지원될 수 있습니다. 이로 인해 클러스터에서 예약할 수 있는 나머지 **Pod**를 추정하기 어려울 수 있습니다.*

***OpenShift Cluster Capacity Tool**을 명령줄에서 독립형 유틸리티로 실행하거나 **OpenShift Dedicated** 클러스터 내부의 **Pod**에서 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 Tool을 실행하여 클러스터에 예약할 수 있는 포드 수를 추정할 수 있습니다.

툴에서 리소스 사용량을 추정하는 데 사용하는 샘플 Pod 사양 파일을 생성합니다. Pod 사양은 리소스 요구 사항을 제한 또는 요청으로 지정합니다. 클러스터 용량 툴에서는 추정 분석에 Pod의 리소스 요구 사항을 고려합니다.

사전 요구 사항

1. Red Hat Ecosystem Catalog의 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

b.

클러스터 역할을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f pod-spec.yaml
```

프로세스

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)
```

위의 예에서 클러스터에 예약할 수 있는 예상 **Pod** 수는 **88**입니다.

8.2.3. Pod 내에서 OpenShift Cluster Capacity Tool을 작업으로 실행

포드 내에서 **OpenShift Cluster Capacity Tool**을 작업으로 실행하면 사용자 개입 없이도 툴을 여러 번 실행할 수 있습니다. **ConfigMap** 오브젝트를 사용하여 **OpenShift Cluster Capacity Tool**을 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

b. 다음 명령을 실행하여 Pod를 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 툴에 클러스터 내에서 **Pod**로 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 **Pod**에서 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 Dedicated** 클러스터에서 바인딩되지 않은 컴퓨팅 리소스로 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** 값은 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 Dedicated**는 제한을 요청으로 나눠 제한 대 요청 비율을 계산합니다. 이 값은 음수가 아닌 **1**보다 큰 정수여야 합니다.

  예를 들어 컨테이너의 **limit** 값이 **cpu: 500**이고 **request** 값이 **cpu: 100**인 경우 **cpu**의 제한 대 요청 비율은 **5**입니다. 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. 그러지 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" (1)
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" (2)
        memory: "1Gi" (3)
      min:
        cpu: "200m" (4)
        memory: "6Mi" (5)
      maxLimitRequestRatio:
        cpu: "10" (6)
```

(1) 제한 범위 오브젝트의 이름입니다.

(2) **Pod**에서 모든 컨테이너에 요청할 수 있는 최대 **CPU** 양입니다.

(3) **Pod**에서 모든 컨테이너에 요청할 수 있는 최대 메모리 양입니다.

(4) **Pod**에서 모든 컨테이너에 요청할 수 있는 최소 **CPU** 양입니다.

(5) **Pod**에서 모든 컨테이너에 요청할 수 있는 최소 메모리 양입니다.

6

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.

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, DockerImage입니다. 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. 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 PVC의 min 제약 조건보다 크거나 같아야 합니다.*
- *PVC(영구 볼륨 클레임)의 리소스 요청이 LimitRange 오브젝트에 지정된 PVC의 max 제약 조건보다 작거나 같아야 합니다.*

PVC LimitRange 오브젝트 정의

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi" 2
      max:
        storage: "50Gi" 3
```

1 *LimitRange 오브젝트의 이름입니다.*

2 *영구 볼륨 클레임에서 요청할 수 있는 최소 스토리지 양입니다.*

3 *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" 1
spec:
  limits:
    - type: "Pod" 2
      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.

***CLI**를 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 Dedicated**에서 컴퓨팅 리소스를 관리하는 방법에 대한 개요를 완전히 확인하는 것이 좋습니다.

각 종류의 리소스(메모리, **CPU**, 스토리지)에 대해 **OpenShift Dedicated**에서는 선택적 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 Dedicated** 스케줄러에 영향을 미칩니다. 스케줄러는 노드에 컨테이너를 예약할 때 메모리 요청을 고려한 다음 컨테이너 사용을 위해 선택한 노드에서 요청된 메모리를 차단합니다.
  - 노드의 메모리가 소모되면 **OpenShift Dedicated**에서 메모리 사용량이 메모리 요청을 가장 많이 초과하는 컨테이너를 제거하는 데 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 할당한 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 **12MB**입니다. 메모리를 할당할 수 없음 **Pod** 이벤트로 인해 컨테이너가 시작되지 않으면 메모리 제한이 너무 낮은 것입니다. 메모리 제한을 늘리거나 제거합니다. 제한을 제거하면 **Pod**에서 바인딩되지 않은 노드 리소스를 사용할 수 있습니다.

8.4.1.1. 애플리케이션 메모리 전략 관리

OpenShift Dedicated에서 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 Dedicated** 클러스터에는 제한 값을 설정해야 할 수 있습니다. 일부는 제한에 따라 요청을 덮어쓸 수 있습니다. 일부 애플리케이션 이미지는 요청 값보다 탐지하기 때문에 설정되는 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 **JVM**과 같이 메모리를 풀링하는 애플리케이션과 관련이 있습니다. 이 페이지의 나머지 부분에서는 이 작업에 대해 설명합니다.

8.4.2. OpenShift Dedicated의 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:+UseCGroupMemoryLimitForHeap을 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 덮어씀)보다 적지 않습니다. 자세한 내용은 OpenShift에서 Java 풋프린트 튜닝(1부), OpenShift에서 Java 풋프린트 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.

많은 Java 툴은 다양한 환경 변수(JAVA_OPTS,GRADLE_OPTS 등)를 사용하여 JVM을 구성하며 올바른 설정이 올바른 JVM으로 전달되도록 하는 것이 어려울 수 있습니다.

OpenJDK는 항상 JAVA_TOOL_OPTIONS 환경 변수를 준수하고 JAVA_TOOL_OPTIONS에 지정된 값은 JVM 명령줄에 지정된 다른 옵션에서 덮어씁니다. 기본적으로 이러한 옵션이 Java 기반 에이전트 이미지에서 실행되는 모든 JVM 워크로드에 기본적으로 사용되도록 OpenShift Dedicated Jenkins Maven 에이전트 이미지는 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 API를 사용해야 합니다.

프로세스

1. *MEMORY_REQUEST 및 MEMORY_LIMIT 스탠자를 추가하도록 Pod를 구성합니다.*

   a. *다음과 유사한 YAML 파일을 생성합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

(1) *이 스탠자를 추가하여 애플리케이션 메모리 요청 값을 검색합니다.*

(2) *이 스탠자를 추가하여 애플리케이션 메모리 제한 값을 검색합니다.*

b. 다음 명령을 실행하여 **Pod**를 생성합니다.

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

검증

1. 원격 셸을 사용하여 **Pod**에 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 노드 메모리 소모가 심각한 경우 **OpenShift Dedicated**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 **137**로 종료되면 **SIGKILL** 신호가 수신되었음을 나타냅니다.*

*컨테이너가 즉시 종료되지 않으면 다음과 같이 **OOM** 종료를 탐지할 수 있습니다.*

1. *원격 셸을 사용하여 **Pod**에 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_control에서 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 Pod가 종료되면(즉시 여부와 관계없이) 단계는 실패, 이유는 OOM 종료가 됩니다. restartPolicy 값에 따라 OOM 종료된 Pod가 다시 시작될 수 있습니다. 재시작되지 않는 경우 복제 컨트롤러와 같은 컨트롤러는 Pod의 실패 상태를 확인하고 새 Pod를 생성하여 이전 Pod를 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 소진되면 OpenShift Dedicated에서 노드의 Pod를 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 Pod의 단계는 실패, 이유는 제거됨입니다. restartPolicy 값과 관계없이 재시작되지 않습니다. 그러나 복제 컨트롤러와 같은 컨트롤러는 Pod의 실패 상태를 확인하고 새 Pod를 생성하여 이전 Pod를 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 POD를 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. **Pod**의 컴퓨팅 리소스 요청 및 노드의 사용 가능한 용량을 고려하여 특정 노드에 **Pod**를 배치합니다.

OpenShift Dedicated 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 과 함께 리소스 제한을 조정하고 요청하여 원하는 수준의 오버 커밋을 수행할 수 있습니다.

참고

OpenShift Dedicated에서는 클러스터 수준 과다 할당을 활성화해야 합니다. 노드 과다 할당은 기본적으로 활성화되어 있습니다. 노드의 과다 할당 비활성화를 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 기반으로하는 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 **1Gi**의 메모리 요청과 **2Gi**의 메모리 제한이 지정되면 노드에서 사용 가능한 **1Gi** 요청에 따라 컨테이너가 예약되지만 최대 **2Gi**를 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 Operator*는 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 표시된 대로 *OpenShift Dedicated* 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)
    limitCPUToMemoryPercent: 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** 덮어 쓰기가 적용되지 않습니다. 프로젝트별 기본 제한이 있는 **LimitRange** 오브젝트를 생성하거나 **Pod** 사양에 제한을 구성하여 덮어쓰기를 적용하십시오.

각 프로젝트의 네임 스페이스 오브젝트에 다음 라벨을 적용하여 프로젝트별로 덮어 쓰기를 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 Dedicated 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 Dedicated 웹 콘솔을 사용하여 **Cluster Resource Override Operator**를 설치하려면 다음을 수행합니다.

1. **OpenShift Dedicated** 웹 콘솔에서 홈 → 프로젝트로 이동합니다 .

a. *프로젝트 만들기를 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)
      limitCPUToMemoryPercent: 200 (4)
# ...
```

(1) 이름은 instance이어야 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 Webhook의 현재 상태를 확인합니다.

a. ClusterResourceOverride Operator 페이지에서 cluster를 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":"ClusterResourceOver
ride","metadata":{"annotations":{},"name":"cluster"},"spec":
{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequ
estToLimitPercent":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 Dedicated CLI를 사용하여 Cluster Resource Override Operator를 설치하여 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 Operator를 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"
  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)
   limitCPUToMemoryPercent: 200 (4)
```

(1) 이름은 instance이어야 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. 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 Webhook의 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,"memoryRequestT
oLimitPercent":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

8.5.2.3. 클러스터 수준 오버 커밋 설정

Cluster Resource Override Operator에는 Operator가 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 Operator에 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 CR을 편집합니다.*

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
spec:
 podResourceOverride:
  spec:
    memoryRequestToLimitPercent: 50 (1)
    cpuRequestToLimitPercent: 25 (2)
    limitCPUToMemoryPercent: 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 CPU의 양이 보장되며 컨테이너에서 지정한 한도까지 노드에서 사용 가능한 초과 CPU를 추가로 소비할 수 있습니다. 여러 컨테이너가 초과 CPU를 사용하려고하면 각 컨테이너에서 요청된 CPU 양에 따라 CPU 시간이 분배됩니다.

예를 들어, 한 컨테이너가 500m의 CPU 시간을 요청하고 다른 컨테이너가 250m의 CPU 시간을 요청한 경우 노드에서 사용 가능한 추가 CPU 시간이 2:1 비율로 컨테이너간에 분배됩니다. 컨테이너가 제한을 지정한 경우 지정된 한도를 초과하는 많은 CPU를 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 메모

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 지정해야합니다. 이러한 결정을 내리는 데 사용되는 기능을 QoS (Quality of Service) 클래스라고 합니다.

Pod는 우선순위 순서가 감소된 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 설정되고 (0과 같지 않음) Pod는 **Guaranteed** 로 분류됩니다.
2	Burstable	모든 리소스에 대해 요청 및 제한(선택 사항)이 설정되고 (0과 같지 않음) Pod는 **Burstable** 로 분류됩니다.
3 (가장 낮음)	BestEffort	리소스에 대해 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 **pod**에 예약된 메모리의 백분율을 지정할 수 있습니다. 이 기능은 요청된 리소스를 예약하여 하위 **OoS** 클래스의 **pod**가 고급 **QoS** 클래스의 **pod**에서 요청한 리소스를 사용하지 못하도록 합니다.

OpenShift Dedicated는 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 **OOM**이 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.

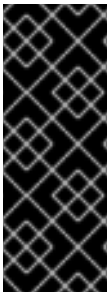
- **qos-reserved=memory=0%** 값은 **Burstable** 및 **BestEffort QoS** 클래스가 사용 가능한 경우 할당 가능한 최대 노드 양까지 소비하는 것을 허용하지만 **Guaranteed** 워크로드가 요청된 메모리에 액세스하지 못할 위험이 높아집니다. 이로 인해 이 기능은 비활성화되어 있습니다.

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 **Pod**의 프로세스가 종료될 수 있습니다.

스왑을 비활성화하지 못하면 노드에서 **MemoryPressure**가 발생하고 있음을 인식하지 못하여 **Pod**가 스케줄링 요청에서 만든 메모리를 받지 못하게 됩니다. 결과적으로 메모리 **Pod**를 추가로 늘리기 위해 추가 **Pod**가 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 **Pod**를 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 안됩니다.

이 동작을 보장하기 위해 **OpenShift Dedicated**는 **vm.overcommit_memory** 매개변수를 **1**로 설정하여 기본 운영 체제 설정을 재정의하여 커널이 항상 메모리를 오버 커밋하도록 구성합니다.

OpenShift Dedicated는 **vm.panic_on_oom** 매개변수를 **0**으로 설정하여 메모리 부족 시 커널이 패닉 상태가 되지 않도록 구성합니다. **0**으로 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
```

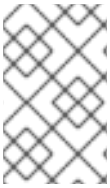
출력 예

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

출력 예

```
#...
vm.panic_on_oom = 0
#...
```



참고

위의 플래그는 이미 노드에 설정되어 있어야하며 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 연결된 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

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

프로세스

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 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    cpuCfsQuota: false 3
```

1. CR에 이름을 지정합니다.
2. 머신 구성 풀에서 라벨을 지정합니다.
3. 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 수없는 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 오브젝트 파일을 편집합니다.
2. 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.