OpenShift Container Platform 4.9

설치 후 구성

OpenShift Container Platform의 Day 2 운영

Last Updated: 2023-07-17

OpenShift Container Platform 4.9 설치 후 구성

OpenShift Container Platform의 Day 2 운영

법적 공지

초록

이 문서는 OpenShift Container Platform의 설치 후 수행되는 작업에 대한 지침을 제공합니다.

차례

1장. 설치 후 구성 개요

OpenShift Container Platform을 설치한 후 클러스터 관리자는 다음 구성 요소를 구성하고 사용자 지정할 수 있습니다.

- Machine
- Cluster
- 노드
- 네트워크
- 스토리지
- 사용자
- 알림 및 공지

1.1. 설치 후 수행할 구성 작업

클러스터 관리자는 설치 후 다음 구성 작업을 수행할 수 있습니다.

- 운영 체제 기능 구성: MCO (Machine Config Operator)는 **MachineConfig** 개체를 관리합니다. MCO를 사용하면 OpenShift Container Platform 클러스터에서 다음 작업을 수행할 수 있습니다.
  - **MachineConfig** 개체를 사용하여 노드 구성
  - MCO 관련 사용자 정의 리소스 구성
- 클러스터 기능 구성: 클러스터 관리자는 OpenShift Container Platform 클러스터의 주요 기능의 구성 리소스를 수정할 수 있습니다. 이러한 기능은 다음과 같습니다.
  - 이미지 레지스트리
  - 네트워킹 구성
  - 이미지 빌드 동작
  - ID 공급자
  - etcd 구성
  - 워크로드를 처리할 머신 세트 생성
  - 클라우드 공급자 인증 정보 관리
- 클러스터 구성 요소를 비공개로 구성합니다. 기본적으로 설치 프로그램은 공개적으로 액세스 가능한 DNS 및 끝점을 사용하여 OpenShift Container Platform을 프로비저닝합니다. 내부 네트워크 내에서만 클러스터에 액세스할 수 있도록 하려면 다음 구성 요소를 프라이빗으로 구성합니다.
  - DNS
  - Ingress 컨트롤러
  - API 서버

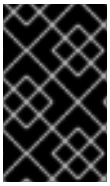
- 노드 작업 수행: 기본적으로 OpenShift Container Platform은 RHCOS (Red Hat Enterprise Linux CoreOS) 컴퓨팅 머신을 사용합니다. 클러스터 관리자는 OpenShift Container Platform 클러스터의 머신을 사용하여 다음 작업을 수행할 수 있습니다.
  - 컴퓨팅 시스템 추가 및 제거
  - 노드에 테인트 및 톨러레이션 추가 및 제거
  - 노드당 최대 Pod 수 구성
  - 장치 관리자 활성화
- 네트워크 구성: OpenShift Container Platform을 설치한 후 다음을 구성할 수 있습니다.
  - Ingress 클러스터 트래픽
  - 노드 포트 서비스 범위
  - 네트워크 정책
  - 클러스터 전체 프록시 사용
- 스토리지 구성: 기본적으로 컨테이너는 임시 스토리지 또는 임시 로컬 스토리지를 사용하여 작동합니다. 임시 스토리지에는 수명 제한이 있습니다. 데이터를 장기간 저장하려면 영구 스토리지를 구성해야 합니다. 다음 방법 중 하나를 사용하여 스토리지를 구성할 수 있습니다.
  - **동적 프로비저닝**: 스토리지 액세스를 포함하여 다양한 수준의 스토리지 클래스를 제어하는 스토리지 클래스를 정의하고 생성하여 필요에 따라 스토리지를 동적으로 프로비저닝할 수 있습니다.
  - **정적 프로비저닝**: Kubernetes 영구 볼륨을 사용하여 기존 스토리지를 클러스터에서 사용할 수 있습니다. 정적 프로비저닝은 다양한 장치 구성 및 마운트 옵션을 지원할 수 있습니다.
- 사용자 구성: OAuth 액세스 토큰을 사용하면 사용자가 API에 자신을 인증할 수 있습니다. 클러스터 관리자는 다음 작업을 수행하도록 OAuth를 구성할 수 있습니다.
- ID 공급자 지정
- 역할 기반 액세스 제어를 사용하여 사용자에게 권한 정의 및 제공
- OperatorHub에서 Operator 설치
- 알림 및 알림 관리: 기본적으로 실행 경고가 웹 콘솔의 알림 UI에 표시됩니다. 경고 알림을 외부 시스템으로 보내도록 OpenShift Container Platform을 구성할 수도 있습니다.

2장. 프라이빗 클러스터 설정

OpenShift Container Platform 버전 4.9 클러스터를 설치한 후 일부 핵심 구성 요소를 프라이빗으로 설정할 수 있습니다.

2.1. 프라이빗 클러스터 정보

기본적으로 OpenShift Container Platform은 공개적으로 액세스 가능한 DNS 및 엔드 포인트를 사용하여 프로비저닝됩니다. 프라이빗 클러스터를 배포한 후 DNS, Ingress 컨트롤러 및 API 서버를 프라이빗으로 설정할 수 있습니다.



중요

클러스터에 퍼블릭 서브넷이 있는 경우 관리자가 생성한 로드 밸런서 서비스에 공개적으로 액세스할 수 있습니다. 클러스터 보안을 보장하기 위해 이러한 서비스에 명시적으로 주석이 지정되었는지 확인합니다.

DNS

설치 프로그램이 프로비저닝한 인프라에 OpenShift Container Platform을 설치하는 경우 설치 프로그램은 기존 퍼블릭 영역에 레코드를 만들고 가능한 경우 클러스터 자체 DNS 확인을 위한 프라이빗 영역을 만듭니다. 퍼블릭 영역과 프라이빗 영역 모두에서 설치 프로그램 또는 클러스터는 API 서버에 대한 *** .apps**, **Ingress** 개체, **api**의 DNS 항목을 만듭니다.

퍼블릭 영역과 프라이빗 영역의 *** .apps** 레코드는 동일하므로 퍼블릭 영역을 삭제하면 프라이빗 영역이 클러스터에 대한 모든 DNS 확인을 완벽하게 제공합니다.

Ingress 컨트롤러

기본 **Ingress** 개체는 퍼블릭으로 생성되기 때문에 로드 밸런서는 인터넷에 연결되어 퍼블릭 서브넷에서 사용됩니다. 기본 Ingress 컨트롤러를 내부 컨트롤러로 교체할 수 있습니다.

API 서버

기본적으로 설치 프로그램은 API 서버가 내부 트래픽 및 외부 트래픽 모두에 사용할 적절한 네트워크로드 밸런서를 만듭니다.

AWS (Amazon Web Services)에서 별도의 퍼블릭 및 프라이빗 로드 밸런서가 생성됩니다. 클러스터에서 사용하기 위해 내부 포트에서 추가 포트를 사용할 수 있다는 점을 제외하고 로드 밸런서는 동일합니다. 설치 프로그램이 API 서버 요구 사항에 따라 로드 밸런서를 자동으로 생성하거나 제거하더라도 클러스터는 이를 관리하거나 유지하지 않습니다. API 서버에 대한 클러스터의 액세스 권한을 유지하는 한 로드 밸런서를 수동으로 변경하거나 이동할 수 있습니다. 퍼블릭 로드 밸런서의 경우 포트 6443이 열려있고 상태 확인은 HTTPS의 / **readyz** 경로에 대해 설정되어 있습니다.

Google Cloud Platform에서 내부 및 외부 API 트래픽을 모두 관리하기 위해 단일 로드 밸런서가 생성되므로 로드 밸런서를 변경할 필요가 없습니다.

Microsoft Azure에서는 퍼블릭 및 프라이빗 로드 밸런서가 모두 생성됩니다. 그러나 현재 구현에 한계가 있기 때문에 프라이빗 클러스터에서 두 로드 밸런서를 유지합니다.

2.2. DNS를 프라이빗으로 설정

클러스터를 배포한 후 프라이빗 영역만 사용하도록 DNS를 변경할 수 있습니다.

프로세스

1. 클러스터의 **DNS** 사용자 정의 리소스를 확인합니다.

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

출력 예

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

spec 섹션에는 프라이빗 영역과 퍼블릭 영역이 모두 포함되어 있습니다.

2. **DNS** 사용자 지정 리소스를 패치하여 퍼블릭 영역을 제거합니다.

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec": {"publicZone": null}}'
dns.config.openshift.io/cluster patched
```

Ingress 컨트롤러는 **Ingress** 개체를 만들 때 **DNS** 정의를 참조하기 때문에 **Ingress** 개체를 만들거나 수정할 때 프라이빗 레코드만 생성됩니다.

중요

퍼블릭 영역을 제거해도 기존 Ingress 개체의 DNS 레코드는 변경되지 않습니다.

3. 선택 사항: 클러스터의 **DNS** 사용자 정의 리소스를 확인하고 퍼블릭 영역이 제거되었는지 확인하십시오.

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

출력 예

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
```

```
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>-wfpg4: owned
status: {}
```

2.3. INGRESS 컨트롤러를 프라이빗으로 설정

클러스터를 배포한 후 프라이빗 영역만 사용하도록 Ingress 컨트롤러를 변경할 수 있습니다.

프로세스

1. 내부 엔드 포인트만 사용하도록 기본 Ingress 컨트롤러를 변경합니다.

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

출력 예

```
ingresscontroller.operator.openshift.io "default" deleted
ingresscontroller.operator.openshift.io/default replaced
```

퍼블릭 DNS 항목이 제거되고 프라이빗 영역 항목이 업데이트됩니다.

2.4. API 서버를 프라이빗으로 제한

AWS (Amazon Web Services) 또는 Microsoft Azure에 클러스터를 배포한 후 프라이빗 영역만 사용하도록 API 서버를 재구성할 수 있습니다.

전제 조건

- OpenShift CLI (**oc**)를 설치합니다.
- **admin** 권한이 있는 사용자로 웹 콘솔에 액세스합니다.

프로세스

1. AWS 또는 Azure의 웹 포털 또는 콘솔에서 다음 작업을 수행합니다.
   a. 적절한 로드 밸런서 구성 요소를 찾아 삭제합니다.

- AWS의 경우 외부 로드 밸런서를 삭제합니다. 프라이빗 영역의 API DNS 항목은 동일한 설정을 사용하는 내부 로드 밸런서를 가리키므로 내부 로드 밸런서를 변경할 필요가 없습니다.
- Azure의 경우 로드 밸런서의 **api-internal** 규칙을 삭제합니다.

b. 퍼블릭 영역의 **api.$clustername.$yourdomain** DNS 항목을 삭제합니다.

2. 외부 로드 밸런서를 제거합니다.

중요

설치 관리자 프로비저닝 인프라(IPI) 클러스터에 대해서만 다음 단계를 실행할 수 있습니다. UPI(사용자 프로비저닝 인프라) 클러스터의 경우 외부 로드 밸런서를 수동으로 제거하거나 비활성화해야 합니다.

a. 터미널에서 클러스터 시스템을 나열합니다.

```
$ oc get machine -n openshift-machine-api
```

출력 예

```
NAME                      STATE    TYPE       REGION     ZONE       AGE
lk4pj-master-0            running  m4.xlarge  us-east-1  us-east-1a  17m
lk4pj-master-1            running  m4.xlarge  us-east-1  us-east-1b  17m
lk4pj-master-2            running  m4.xlarge  us-east-1  us-east-1a  17m
lk4pj-worker-us-east-1a-5fzfj  running  m4.xlarge  us-east-1  us-east-1a  15m
lk4pj-worker-us-east-1a-vbghs  running  m4.xlarge  us-east-1  us-east-1a  15m
lk4pj-worker-us-east-1b-zgpzg  running  m4.xlarge  us-east-1  us-east-1b  15m
```

다음 단계에서 이름에 **master**가 포함된 컨트롤 플레인 시스템을 변경합니다.

b. 각 컨트롤 플레인 시스템에서 외부 로드 밸런서를 제거합니다.

i. 컨트롤 플레인 **Machine** 오브젝트를 편집하여 외부 로드 밸런서에 대한 참조를 제거합니다.

```
$ oc edit machines -n openshift-machine-api <master_name> 1
```

1 변경할 컨트롤 플레인 또는 마스터, **Machine** 개체의 이름을 지정합니다.

ii. 다음 예에 표시된 외부 로드 밸런서를 설명하는 행을 제거하고 개체 사양을 저장한 후 종료합니다.

```
...
spec:
  providerSpec:
    value:
    ...
      loadBalancers:
      - name: lk4pj-ext 1
```

```
    type: network (2)
  - name: lk4pj-int
    type: network
```

(1) (2) 이 행을 삭제합니다.

iii. 이름에 **master**가 포함된 각 시스템에 대해 이 프로세스를 반복합니다.

3장. 설치 후 시스템 구성 작업

OpenShift Container Platform 노드에서 실행되는 운영 체제를 변경해야하는 경우가 있습니다. 여기에는 네트워크 시간 서비스 설정 변경, 커널 인수 추가 또는 특정 방식으로 저널 설정이 포함됩니다.

몇 가지 특수 기능 외에도 OpenShift Container Platform 노드에서 운영 체제 대부분의 변경 사항은 **Machine Config Operator**가 관리하는 MachineConfig 객체를 생성하여 수행할 수 있습니다.

이 섹션의 작업은 Machine Config Operator의 기능을 사용하여 OpenShift Container Platform 노드에서 운영 체제 기능을 구성하는 방법을 설명합니다.

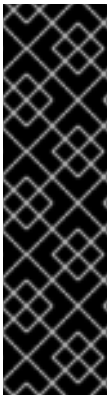
3.1. MACHINE CONFIG OPERATOR 이해

3.1.1. Machine Config Operator

목적
Machine Config Operator는 커널과 kubelet 사이의 모든 것을 포함하여 기본 운영 체제 및 컨테이너 런타임의 구성 및 업데이트를 관리하고 적용합니다.

다음의 네 가지 구성 요소가 있습니다.

- **machine-config-server**: 클러스터에 가입하는 새 머신에 Ignition 설정을 제공합니다.
- **machine-config-controller**: **MachineConfig** 객체에 의해 정의된 설정으로 머신 업그레이드를 조정합니다. 머신 세트의 업그레이드를 개별적으로 제어하는 옵션이 제공됩니다.
- **machine-config-daemon**: 업데이트 중에 새로운 머신 설정을 적용합니다. 머신 상태를 요청한 머신 구성에 대해 검증하고 확인합니다.
- **machine-config**: 처음으로 머신을 설치, 시작 및 업데이트하기위한 완전한 머신 구성 소스를 제공합니다.



중요

현재는 머신 구성 서버 끝점을 차단하거나 제한할 수 있는 방법이 없습니다. 기존 구성 또는 상태가 없는 새로 프로비저닝된 머신이 구성을 가져올 수 있도록 머신 구성 서버를 네트워크에 노출해야 합니다. 이 모델에서 신뢰의 루트는 CSR(인증서 서명 요청) 끝점으로, kubelet이 클러스터에 가입하기 위해 승인하기 위해 인증서 서명 요청을 보내는 위치입니다. 이로 인해 시크릿 및 인증서와 같은 중요한 정보를 배포하는 데 머신 구성을 사용해서는 안 됩니다.

머신 구성 서버 끝점, 포트 22623 및 22624가 베어 메탈 시나리오에서 보호되도록 하려면 고객이 적절한 네트워크 정책을 구성해야 합니다.

추가 리소스

- OpenShift SDN 네트워크 플러그인 정보.

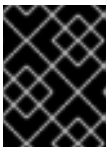
프로젝트
openshift-machine-config-operator

3.1.2. Machine Config 개요

MCO (Machine Config Operator)는 systemd, CRI-O 및 Kubelet, 커널, 네트워크 관리자 및 기타 시스템

기능에 대한 업데이트를 관리합니다. 또한 호스트에 구성 파일을 쓸 수 있는 **MachineConfig** CRD를 제공합니다(machine-config-operator참조). OpenShift Container Platform 클러스터에 대한 고급 시스템 수준을 변경하려면 MCO의 기능과 다른 구성 요소와 상호 작용 방식을 이해하는 것이 중요합니다. MCO, 머신 구성 및 사용 방법에 대해 알아야 할 몇 가지 사항은 다음과 같습니다.

- 머신 구성은 OpenShift Container Platform 노드 풀을 나타내는 각 시스템의 운영 체제에서 파일 또는 서비스를 특정하게 변경할 수 있습니다.
- MCO는 시스템 풀의 운영 체제에 변경 사항을 적용합니다. 모든 OpenShift Container Platform 클러스터는 작업자 및 컨트롤 플레인 노드 풀로 시작합니다. 역할 레이블을 추가하여 사용자 지정 노드 풀을 구성할 수 있습니다. 예를 들어 애플리케이션에 필요한 특정 하드웨어 기능을 포함하는 작업자 노드의 사용자 정의 풀을 설정할 수 있습니다. 그러나 이 섹션의 예에서는 기본 풀 유형의 변경에 중점을 둡니다.



중요

노드는 **master** 또는 **worker**와 같이 유형을 나타내기 위해 여러 레이블을 적용할 수 있지만 **단일** 머신 구성 풀의 멤버일 수 있습니다.

- OpenShift Container Platform을 디스크에 설치하기 전에 일부 머신 구성을 완료해야 합니다. 대부분의 경우 이는 설치 후 머신 구성으로 실행되지 않고 OpenShift Container Platform 설치 프로그램 프로세스에 직접 삽입되는 머신 구성을 생성하여 이를 수행할 수 있습니다. 또는 OpenShift Container Platform 설치 프로그램 시작 시 커널 인수를 전달하는 베어 메탈 설치를 수행해야 노드별 개별 IP 주소 설정 또는 고급 디스크 파티셔닝과 같은 작업을 수행해야 할 수 있습니다.
- MCO는 머신 구성에 설정된 항목을 관리합니다. MCO가 충돌하는 파일을 관리하도록 명시적으로 지시하지 않는 한 MCO는 시스템에 대한 수동 변경 사항을 덮어 쓰지 않습니다. 즉, MCO는 사용자가 요청한 특정 업데이트 만 수행하고 전체 노드에 대한 제어를 요구하지 않습니다.
- 노드를 수동으로 변경하지 않는 것이 좋습니다. 노드를 종료하고 새 노드를 시작해야하는 경우 이러한 직접적인 변경 사항이 손실됩니다.
- MCO는 **/etc** 및 **/var** 디렉토리에있는 파일에 쓰는 경우에만 지원됩니다. 하지만 이러한 영역 중 하나에 심볼릭 링크를 사용하여 쓰기 가능해진 일부 디렉토리에 대한 심볼릭 링크도 있습니다. **/opt** 및 **/usr/local** 디렉토리는 예제입니다.
- Ignition은 MachineConfigs에서 사용되는 구성 형식입니다. 자세한 내용은 Ignition Configuration Specification v3.2.0을 참조하십시오.
- Ignition 구성 설정은 OpenShift Container Platform 설치시 직접 제공될 수 있고 MCO가 Ignition 구성을 제공하는 것과 동일한 방식으로 포맷할 수 있지만 MCO는 원래 Ignition 구성이 무엇인지 확인할 방법이 없습니다. 따라서 Ignition 구성 설정을 배포하기 전에 이를 머신 구성에 래핑해야 합니다.
- MCO에서 관리하는 파일이 MCO 외부에서 변경되면 MCD (Machine Config Daemon)가 노드를 **degraded**로 설정합니다. 이는 문제가 되는 파일을 덮어 쓰지 않으며 성능이 **degraded** 상태에서 계속 작동합니다.
- 머신 구성을 사용하는 주요 이유는 OpenShift Container Platform 클러스터의 풀에 새 노드를 추가할 때 적용되기 때문입니다. **machine-api-operator**는 새 머신을 프로비저닝하고 MCO가 이를 구성합니다.

MCO는 Ignition을 구성 형식으로 사용합니다. OpenShift Container Platform 4.6은 Ignition 구성 사양 버전 2에서 버전 3으로 이동했습니다.

3.1.2.1. 머신 구성에서 변경 가능한 구성

MCO가 변경할 수 있는 구성 요소의 종류는 다음과 같습니다.

- **config**: Ignition 구성 개체 (Ignition 구성 사양 참조)를 생성하여 다음을 포함하여 OpenShift Container Platform 시스템에서 파일, systemd 서비스 및 기타 기능을 변경할 수 있습니다.
  - **Configuration files**: **/var** 또는 **/etc** 디렉토리에 파일을 만들거나 덮어 씁니다.
  - **systemd units**: systemd 서비스의 상태를 생성 및 설정하거나 추가 설정을 기존 systemd 서비스에 추가합니다.
  - **users and groups**: 설치 후 passwd 섹션에서 SSH 키를 변경합니다.

중요

core 사용자만 시스템 구성을 통한 SSH 키 변경을 지원합니다.

- **kernelArguments**: OpenShift Container Platform 노드가 시작될 때 커널 명령 줄에 인수를 추가합니다.
- **kernelType**: 선택 옵션으로 표준 커널 대신 사용할 비표준 커널을 확인합니다. RT 커널 (RAN 용)을 사용하려면 **realtime**을 사용합니다. 이는 일부 플랫폼에서만 지원됩니다.
- **fips**: FIPS 모드를 활성화합니다. FIPS는 설치 후 단계가 아닌 설치시 기본값으로 설정해야합니다.

중요

진행 중인 FIPS 검증 / 모듈 암호화 라이브러리 사용은 **x86_64** 아키텍처의 OpenShift Container Platform 배포에서만 지원됩니다.

- **extensions**: 사전 패키지화된 소프트웨어를 추가하여 RHCOS 기능을 확장합니다. 이 기능의 경우 사용 가능한 확장에는 usbguard 및 커널 모듈이 포함됩니다.
- **사용자 지정 리소스 (ContainerRuntime 및 Kubelet용)**: 머신 구성 외부에서 MCO는 CRI-O 컨테이너 런타임 설정 (**ContainerRuntime** CR) 및 Kubelet 서비스 (**Kubelet** CR)를 변경하기 위해 두 가지 특정 사용자 지정 리소스를 관리합니다.

MCO는 OpenShift Container Platform 노드에서 운영 체제 구성 요소를 변경할 수 있는 유일한 Operator가 아닙니다. 다른 Operator도 운영 체제 수준의 기능을 변경할 수 있습니다. 한 가지 예로 Node Tuning Operator를 사용하여 Tuned 데몬 프로필을 통해 노드 수준 조정을 수행할 수있습니다.

설치 후 수행할 수있는 MCO 구성 작업은 다음에 설명되어 있습니다. OpenShift Container Platform 설치 중 또는 설치 전에 수행해야 하는 시스템 설정 작업은 RHCOS 베어 메탈 설치에 대한 설명을 참조하십시오.

3.1.2.2. 프로젝트

자세한 내용은 openshift-machine-config-operator GitHub 사이트를 참조하십시오.

3.1.3. Machine config pool 상태 확인

MCO(Machine Config Operator), 하위 구성 요소 및 관리하는 리소스의 상태를 보려면 다음 **oc** 명령을 사용합니다.

프로세스

1. 각 MCP(머신 구성 풀)에 대해 클러스터에서 사용 가능한 MCO 관리 노드 수를 보려면 다음 명령을 실행합니다.

```
$ oc get machineconfigpool
```

출력 예

```
NAME     CONFIG                 UPDATED UPDATING   DEGRADED MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master   rendered-master-06c9c4…  True    False      False    3          3                3
0                  4h42m
worker   rendered-worker-f4b64…   False   True       False    3          2                2
0                  4h42m
```

다음과 같습니다.

UPDATED

True 상태는 MCO가 현재 머신 구성을 해당 MCP의 노드에 적용했음을 나타냅니다. 현재 머신 구성은 **oc get mcp** 출력의 **STATUS** 필드에 지정됩니다. **False** 상태는 MCP의 노드가 업데이트 중임을 나타냅니다.

업데이트

True 상태는 **MachineConfigPool** 사용자 정의 리소스에 지정된 대로 MCO가 해당 MCP의 노드 중 하나 이상에 지정된 대로 원하는 머신 구성을 적용함을 나타냅니다. 원하는 머신 구성은 새로 편집된 머신 구성입니다. 업데이트 중인 노드를 예약에 사용할 수 없을 수 있습니다. **False** 상태는 MCP의 모든 노드가 업데이트되었음을 나타냅니다.

DEGRADED

True 상태는 MCO가 현재 또는 원하는 머신 구성을 해당 MCP의 노드 중 하나 이상에 적용하지 못하거나 구성이 실패함을 나타냅니다. 성능이 저하된 노드는 스케줄링에 사용할 수 없을 수 있습니다. **False** 상태는 MCP의 모든 노드가 준비되었음을 나타냅니다.

MACHINECOUNT

해당 MCP의 총 머신 수를 나타냅니다.

READYMACHINECOUNT

예약할 준비가 된 MCP의 총 머신 수를 나타냅니다.

UPDATEDMACHINECOUNT

현재 머신 구성이 있는 MCP의 총 머신 수를 나타냅니다.

DEGRADEDMACHINECOUNT

degraded 또는 Unreconcilable으로 표시된 MCP의 총 머신 수를 나타냅니다.

이전 출력에는 컨트롤 플레인 (마스터) 노드와 3 개의 작업자 노드가 있습니다. 컨트롤 플레인 MCP 및 관련 노드가 현재 머신 구성으로 업데이트됩니다. 작업자 MCP의 노드가 원하는 머신 구성으로 업데이트되고 있습니다. 작업자 MCP의 노드 중 두 개가 업데이트되어 **UPDATEDMACHINECOUNT** 가 **2** 로 표시된 대로 계속 업데이트됩니다. **DEGRADEDMACHINECOUNT** 가 **0** 이고 **DEGRADED** 가 **False** 인 경우 문제가 없습니다.

MCP의 노드가 업데이트되는 동안 **CONFIG** 아래에 나열된 머신 구성은 현재 머신 구성으로, MCP가 업데이트되고 있습니다. 업데이트가 완료되면 나열된 머신 구성이 MCP를 업데이트한 원하는 머신 구성입니다.

참고

노드가 차단되는 경우 해당 노드는 **READYMACHINECOUNT** 에 포함되지 않지만 **MACHINECOUNT** 에 포함됩니다. 또한 MCP 상태는 **UPDATING** 으로 설정됩니다. 노드에 현재 머신 구성이 있으므로 **UPDATEDMACHINECOUNT** 합계에 계산됩니다.

출력 예

```
NAME     CONFIG                 UPDATED  UPDATING   DEGRADED
MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
master   rendered-master-06c9c4…  True     False      False    3          3
3                0                4h42m
worker   rendered-worker-c1b41a…  False    True       False    3          2
3                0                4h42m
```

2. **MachineConfigPool** 사용자 정의 리소스를 검사하여 MCP의 노드 상태를 확인하려면 다음 명령을 실행합니다.

```
$ oc describe mcp worker
```

출력 예

```
...
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        2
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
Events:                       <none>
```

참고

노드가 차단 중이면 노드가 **Ready 머신 수에 포함되지 않습니다**. **Unavailable Machine Count** 에 포함되어 있습니다:

출력 예

```
...
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        2
  Ready Machine Count:        2
  Unavailable Machine Count:  1
  Updated Machine Count:      3
```

3. 기존 **MachineConfig** 오브젝트를 보려면 다음 명령을 실행합니다.

```
$ oc get machineconfigs
```

출력 예

```
NAME                        GENERATEDBYCONTROLLER          IGNITIONVERSION  AGE
00-master                   2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
00-worker                   2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
01-master-container-runtime      2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
01-master-kubelet           2c9371fbb673b97a6fe8b1c52…     3.2.0            5h18m
...
rendered-master-dde...          2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
rendered-worker-fde...          2c9371fbb673b97a6fe8b1c52...   3.2.0            5h18m
```

rendered 로 나열된 **MachineConfig** 오브젝트는 변경하거나 삭제할 수 없습니다.

4. 특정 머신 구성의 내용을 보려면 (이 경우 **01-master-kubelet**) 다음 명령을 실행합니다.

```
$ oc describe machineconfigs 01-master-kubelet
```

명령의 출력에는 이 **MachineConfig** 오브젝트에 구성 파일(**cloud.conf** 및 **kubelet.conf**)과 systemd 서비스(Kubernetes Kubelet)가 모두 포함되어 있음을 보여줍니다.

출력 예

```
Name:         01-master-kubelet
...
Spec:
  Config:
    Ignition:
      Version:  3.2.0
    Storage:
      Files:
        Contents:
          Source:   data:,
        Mode:       420
        Overwrite:  true
        Path:       /etc/kubernetes/cloud.conf
        Contents:
          Source:
data:,kind%3A%20KubeletConfiguration%0AapiVersion%3A%20kubelet.config.k8s.io%2Fv1bet
a1%0Aauthentication%3A%0A%20%20x509%3A%0A%20%20%20%20clientCAFile%3A%20
%2Fetc%2Fkubernetes%2Fkubelet-ca.crt%0A%20%20anonymous...
        Mode:       420
        Overwrite:  true
        Path:       /etc/kubernetes/kubelet.conf
    Systemd:
      Units:
        Contents:  [Unit]
Description=Kubernetes Kubelet
Wants=rpc-statd.service network-online.target crio.service
After=network-online.target crio.service

ExecStart=/usr/bin/hyperkube \
    kubelet \
      --config=/etc/kubernetes/kubelet.conf \ ...
```

적용한 머신 구성에서 문제가 발생하면 언제든지 해당 변경 사항을 취소할 수 있습니다. 예를 들어 **oc create -f ./myconfig.yaml** 을 실행하여 머신 구성을 적용한 경우 다음 명령을 실행하여 해당 머신 구성을 제거할 수 있습니다.

```
$ oc delete -f ./myconfig.yaml
```

이것이 유일한 문제인 경우 영향을 받는 풀 노드는 성능이 저하되지 않은 상태로 돌아갑니다. 이로 인해 실제로 렌더링된 구성이 이전에 렌더링된 상태로 롤백됩니다.

자체 머신 구성을 클러스터에 추가하는 경우 위의 예에 표시된 명령을 사용하여 해당 상태 및 적용되는 풀의 관련 상태를 확인할 수 있습니다.

3.2. MACHINECONFIG 개체를 사용하여 노드 구성

이 섹션의 작업을 통해 **MachineConfig** 객체를 생성하여 OpenShift Container Platform 노드에서 실행되는 파일, systemd 단위 파일 및 기타 운영 체제 기능을 변경할 수 있습니다. 머신 구성 사용에 대한 자세한 내용은 SSH 인증 키 업데이트, 이미지 서명 확인,SCTP 활성화, OpenShift Container Platform 용 iSCSI 개시자 이름 구성과 관련된 내용을 참조하십시오.

OpenShift Container Platform은 Ignition 사양 버전 3.2 을 지원합니다. 앞으로 생성하는 모든 새로운 머신 구성은 Ignition 사양 버전 3.2를 기반으로 해야합니다. OpenShift Container Platform 클러스터를 업그레이드하는 경우 기존 Ignition 사양 버전 2.x 머신 구성은 사양 버전 3.2로 자동 변환됩니다.

작은 정보

OpenShift Container Platform 노드에 다른 구성 파일을 추가하는 방법의 경우 다음 " chrony 타임 서비스 구성" 절차를 모델로 사용하십시오.

3.2.1. chrony 타임 서비스 설정

chrony.conf 파일의 내용을 수정하고 해당 내용을 머신 구성으로 노드에 전달하여 chrony 타임 서비스(**chronyd**)에서 사용하는 시간 서버 및 관련 구성을 설정할 수 있습니다.

프로세스

1. **chrony.conf** 파일의 내용을 포함하여 Butane config를 만듭니다. 예를 들어 작업자 노드에 chrony를 구성하려면 **99-worker-chrony.bu** 파일을 만듭니다.

참고

Butane에 대한 자세한 내용은 “Butane 을 사용하여 머신 구성 생성”을 참조하십시오.

```
variant: openshift
version: 4.9.0
metadata:
  name: 99-worker-chrony 1
  labels:
    machineconfiguration.openshift.io/role: worker 2
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644 3
```

```
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst 4
        driftfile /var/lib/chrony/drift
        makestep 1.0 3
        rtcsync
        logdir /var/log/chrony
```

1 2 컨트롤 플레인 노드에서 두 위치에 있는 **master**를 **worker**로 대체합니다.

3 시스템 구성 파일에서 **mode** 필드의 8진수 값 모드를 지정합니다. 파일을 만들고 변경 사항을 적용하면 **모드가** 10진수 값으로 변환됩니다. **oc get mc <mc-name> -o yaml** 명령을 사용하여 YAML 파일을 확인할 수 있습니다.

4 DHCP 서버에서 제공하는 것과 같은 유효한 시간 소스를 지정합니다. 다른 방법으로 **1.rhel.pool.ntp.org**, **2.rhel.pool.ntp.org** 또는 **3.rhel.pool.ntp.org**의 NTP 서버 중 하나를 지정할 수 있습니다.

2. Butane을 사용하여 노드에 전달할 구성이 포함된 **MachineConfig** 파일 **99-worker-chrony.yaml**을 생성합니다.

```
$ butane 99-worker-chrony.bu -o 99-worker-chrony.yaml
```

3. 다음 두 가지 방법 중 하나로 설정을 적용하십시오.

   - 클러스터가 아직 실행되지 않은 경우 매니페스트 파일을 생성한 후 **<installation_directory>/openshift** 디렉터리에 **MachineConfig** 개체 파일을 추가한 다음 클러스터를 계속 작성합니다.
   - 클러스터가 이미 실행중인 경우 다음과 같은 파일을 적용합니다.

```
$ oc apply -f ./99-worker-chrony.yaml
```

추가 리소스

- Butane을 사용하여 머신 구성 생성

3.2.2. chrony 타임 서비스 비활성화

MachineConfig CR(사용자 정의 리소스)을 사용하여 특정 역할이 있는 노드의 chrony 타임 서비스(**chronyd**)를 비활성화할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

절차

1. 지정된 노드 역할에 대해 **chronyd**를 비활성화하는 **MachineConfig** CR을 만듭니다.

   a. 다음 YAML을 **disable-chronyd.yaml** 파일에 저장합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role> 1
  name: disable-chronyd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpdate.service sntp.service ntpd.service
            Conflicts=ntpd.service systemd-timesyncd.service
            ConditionCapability=CAP_SYS_TIME
            [Service]
            Type=forking
            PIDFile=/run/chrony/chronyd.pid
            EnvironmentFile=-/etc/sysconfig/chronyd
            ExecStart=/usr/sbin/chronyd $OPTIONS
            ExecStartPost=/usr/libexec/chrony-helper update-daemon
            PrivateTmp=yes
            ProtectHome=yes
            ProtectSystem=full
            [Install]
            WantedBy=multi-user.target
          enabled: false
          name: "chronyd.service"
```

1 **chronyd**를 비활성화하려는 노드 역할(예: **master**)입니다.

b. 다음 명령을 실행하여 **MachineConfig** CR을 생성합니다.

```
$ oc create -f disable-chronyd.yaml
```

3.2.3. 노드에 커널 인수 추가

특별한 경우에는 클러스터 노드 세트에 커널 인수를 추가해야 할 수 있습니다. 이 작업을 수행할 때 주의해야 하며 먼저 설정된 인수의 영향을 명확하게 이해하고 있어야합니다.

주의

커널 인수를 잘못 사용하면 시스템이 부팅되지 않을 수 있습니다.

설정할 수 있는 커널 인수의 예는 다음과 같습니다.

- **enforcing=0**: SELinux(Security Enhanced Linux)를 허용 모드에서 실행하도록 구성합니다. 허용 모드에서는 SELinux가 개체에 레이블을 지정하고 로그에 액세스 거부 항목을 내보내는 등 로드된 보안 정책을 적용하는 것처럼 동작하지만 실제로는 어떤 작업도 거부하지 않습니다. 프로덕션 시스템에는 지원되지 않지만 허용 모드는 디버깅에 유용할 수 있습니다.

- **nosmt**: 커널에서 대칭 멀티 스레딩 (SMT)을 비활성화합니다. 멀티 스레딩은 각 CPU마다 여러 개의 논리 스레드를 허용합니다. 멀티 테넌트 환경에서 **nosmt**를 사용하여 잠재적인 크로스 스레드 공격 위험을 줄일 수 있습니다. SMT를 비활성화하는 것은 기본적으로 성능보다는 보안을 중요시하여 선택하는 것과 같습니다.

커널 인수 목록 및 설명은 Kernel.org 커널 매개변수에서 참조하십시오.

다음 프로세스에서는 다음을 식별하는 **MachineConfig**를 만듭니다.

- 커널 인수를 추가하려는 머신 세트입니다. 이 경우 작업자 역할을 갖는 머신입니다.

- 기존 커널 인수 끝에 추가되는 커널 인수입니다.

- 머신 구성 목록에서 변경 사항이 적용되는 위치를 나타내는 라벨입니다.

사전 요구 사항

- OpenShift Container Platform 클러스터에 대한 관리자 권한을 보유하고 있어야 합니다.

프로세스

1. OpenShift Container Platform 클러스터의 기존 **MachineConfig** 오브젝트를 나열하고 머신 구성에 라벨을 지정하는 방법을 결정합니다.

```
$ oc get MachineConfig
```

출력 예

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                 3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                                 3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
```

```
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

2. 커널 인수를 식별하는 **MachineConfig** 파일을 만듭니다 (예: **05-worker-kernelarg-selinuxpermissive.yaml**).

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 05-worker-kernelarg-selinuxpermissive (2)
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
    - enforcing=0 (3)
```

(1) 새 커널 인수를 작업자 노드에만 적용합니다.

(2) 머신 구성(05) 중 적합한 위치와 어떤 기능 (SELinux 허용 모드를 구성하기 위해 커널 매개 변수 추가)을 하는지 식별하기 위해 이름이 지정됩니다.

(3) 정확한 커널 인수를 **enforcing=0**으로 식별합니다.

3. 새 머신 구성을 생성합니다.

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. 머신 구성에서 새 구성이 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

출력 예

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
05-worker-kernelarg-selinuxpermissive                                                3.2.0             105s
99-master-generated-registries                         52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
```

```
3.2.0             33m
99-master-ssh                                                 3.2.0             40m
99-worker-generated-registries                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                 3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

5. 노드를 확인합니다.

```
$ oc get nodes
```

출력 예

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.22.1
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.22.1
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.22.1
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.22.1
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.22.1
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.22.1
```

변경 사항이 적용되어 있기 때문에 각 작업자 노드의 예약이 비활성화되어 있음을 알 수 있습니다.

6. 작업자 노드 중 하나로 이동하여 커널 명령 행 인수 (호스트의 **/proc/cmdline** 에 있음)를 나열하여 커널 인수가 작동하는지 확인합니다.

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

출력 예

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

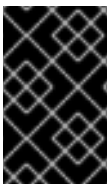
enforcing=0 인수가 다른 커널 인수에 추가된 것을 확인할 수 있습니다.

3.2.4. RHCOS에서 커널 인수로 다중 경로 활성화

RHCOS(Red Hat Enterprise Linux CoreOS)는 기본 디스크에서 다중 경로를 지원하므로 하드웨어 장애에 대한 탄력성이 강화된 호스트 가용성을 높일 수 있습니다. 설치 후 지원은 머신 구성을 통해 다중 경로를 활성화하여 사용할 수 있습니다.

중요

설치 중에 다중 경로를 활성화하는 것은 OpenShift Container Platform 4.8 이상에서 프로비저닝된 노드에 권장됩니다. I/O에서 최적화된 경로로 인해 I/O 시스템 오류가 발생하는 설정에서 설치 시 멀티패스를 활성화해야 합니다. 설치 시 다중 경로를 활성화하는 방법에 대한 자세한 내용은 *베어 메탈에 설치* 문서의 "RHCOS에서 커널 인수를 사용하여 다중 경로 활성화"를 참조하십시오.



중요

IBM Z 및 LinuxONE에서는 설치 중에 클러스터를 구성하는 경우에만 다중 경로를 활성화할 수 있습니다. 자세한 내용은 *IBM Z 및 LinuxONE에 z/VM으로 클러스터 설치* 의 "RHCOS 설치 및 OpenShift Container Platform 부트스트랩 프로세스 시작"을 참조하십시오.

사전 요구 사항

- OpenShift Container Platform 클러스터 (버전 4.7 이상)가 실행되고 있어야 합니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- 멀티패스에 디스크가 활성화되었는지 확인했습니다. 멀티패스는 HBA 어댑터를 통해 SAN에 연결된 호스트에서만 지원됩니다.

절차

1. 컨트롤 플레인 노드에서 다중 경로 설치 후 활성화하려면 다음을 수행합니다.
   - 다음과 같이 클러스터에 **master** 레이블를 추가하도록 지시하고 다중 경로 커널 인수를 식별하는 **99-master-kargs-mpath.yaml**과 같은 머신 구성 파일을 만듭니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "master"
  name: 99-master-kargs-mpath
spec:
  kernelArguments:
    - 'rd.multipath=default'
    - 'root=/dev/disk/by-label/dm-mpath-root'
```

2. 작업자 노드에서 다중 경로 설치 후 활성화하려면 다음을 수행합니다.
   - 다음과 같은 **99-worker-kargs-mpath.yaml** 과 같은 머신 구성 파일을 생성하여 클러스터에 **worker** 레이블을 추가하고 다중 경로 커널 인수를 식별합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
  name: 99-worker-kargs-mpath
spec:
```

```
    kernelArguments:
      - 'rd.multipath=default'
      - 'root=/dev/disk/by-label/dm-mpath-root'
```

3. 이전에 작성한 마스터 또는 작업자 YAML 파일을 사용하여 새 머신 구성을 생성합니다.

```
$ oc create -f ./99-worker-kargs-mpath.yaml
```

4. 머신 구성에서 새 구성이 추가되었는지 확인합니다.

```
$ oc get MachineConfig
```

출력 예

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                       3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-kargs-mpath                                    52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             105s
99-worker-ssh                                                                       3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

5. 노드를 확인합니다.

```
$ oc get nodes
```

출력 예

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.22.1
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.22.1
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.22.1
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.22.1
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.22.1
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.22.1
```

변경 사항이 적용되어 있기 때문에 각 작업자 노드의 예약이 비활성화되어 있음을 알 수 있습니다.

6. 작업자 노드 중 하나로 이동하여 커널 명령 행 인수 (호스트의 **/proc/cmdline** 에 있음)를 나열하여 커널 인수가 작동하는지 확인합니다.

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

출력 예

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
...
rd.multipath=default root=/dev/disk/by-label/dm-mpath-root
...

sh-4.2# exit
```

추가된 커널 인수가 표시되어야 합니다.

추가 리소스

- 설치 시간 동안 다중 경로를 활성화하는 방법에 대한 자세한 내용은 RHCOS에서 커널 인수를 사용하여 다중 경로 활성화를 참조하십시오.

3.2.5. 노드에 실시간 커널 추가

일부 OpenShift Container Platform 워크로드에는 높은 수준의 결정이 필요합니다. Linux는 실시간 운영 체제가 아니지만 Linux 실시간 커널에는 운영 체제에 실시간 기능을 제공하는 선점 형 스케줄러가 포함되어 있습니다.

OpenShift Container Platform 워크로드에 이러한 실시간 기능이 필요한 경우 머신을 Linux 실시간 커널로 전환할 수 있습니다. OpenShift Container Platform, 4.9의 경우 **MachineConfig** 오브젝트를 사용하여 이러한 전환을 수행할 수 있습니다. 머신 구성 **kernelType** 설정을 **realtime**으로 변경하는 것처럼 간단하지만 변경을 수행하기 전에 몇 가지 고려해야 할 사항이 있습니다.

- 현재 실시간 커널은 작업자 노드에서만 지원되며 RAN (Radio Access Network) 사용만 지원됩니다.
- 다음 단계는 Red Hat Enterprise Linux for Real Time 8에서 인증 된 시스템을 사용하는 베어 메탈 설치에 완전히 지원됩니다.
- OpenShift Container Platform에서 실시간 지원은 특정 서브스크립션으로 제한됩니다.
- 다음 단계는 Google Cloud Platform에서의 사용도 지원됩니다.

전제 조건

- 실행중인 OpenShift Container Platform 클러스터 (버전 4.4 이상)가 있어야합니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인합니다.

절차

1. 실시간 커널의 머신 구성을 만듭니다. **realtime** 커널 유형의 **MachineConfig** 개체가 포함된 YAML 파일 (예: **99-worker-realtime.yaml**)을 만듭니다. 다음 예에서는 모든 작업자 노드에 대해 실시간 커널을 사용하도록 클러스터에 지시합니다.

```
$ cat << EOF > 99-worker-realtime.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
  name: 99-worker-realtime
spec:
  kernelType: realtime
EOF
```

2. 머신 구성을 클러스터에 추가합니다. 다음을 입력하여 머신 구성을 클러스터에 추가합니다.

```
$ oc create -f 99-worker-realtime.yaml
```

3. 실시간 커널을 확인합니다 : 영향을 받는 각 노드가 재부팅되면 클러스터에 로그인하고 다음 명령을 실행하여 실시간 커널이 구성된 노드 세트의 일반 커널을 대체하고 있는지 확인합니다.

```
$ oc get nodes
```

출력 예

```
NAME                                        STATUS  ROLES    AGE   VERSION
ip-10-0-143-147.us-east-2.compute.internal  Ready   worker   103m  v1.22.1
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.22.1
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.22.1
```

```
$ oc debug node/ip-10-0-143-147.us-east-2.compute.internal
```

출력 예

```
Starting pod/ip-10-0-143-147us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`

sh-4.4# uname -a
Linux <worker_node> 4.18.0-147.3.1.rt24.96.el8_1.x86_64 #1 SMP PREEMPT RT
        Wed Nov 27 18:29:55 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

커널 이름에는 **rt** 및 "PREEMPT RT"라는 텍스트가 포함되어 이것이 실시간 커널임을 나타냅니다.

4. 일반 커널로 돌아가려면 **MachineConfig** 객체를 삭제합니다.

```
$ oc delete -f 99-worker-realtime.yaml
```

3.2.6. journald 설정 구성

OpenShift Container Platform 노드에서 **journald** 서비스 설정을 구성해야하는 경우 적절한 구성 파일을 수정하고 해당 파일을 머신 구성으로 적절한 노드 풀에 전달하여 이를 수행할 수 있습니다.

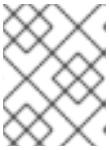
이 프로세스에서는 **/etc/systemd/journald.conf** 파일에서 **journald** 속도 제한 설정을 수정하고 이를 작업자 노드에 적용하는 방법을 설명합니다. 해당 파일을 사용하는 방법에 대한 정보는 **journald.conf** man 페이지를 참조하십시오.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터가 있어야 합니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인합니다.

절차

1. 필요한 설정과 함께 **/etc/systemd/journald.conf** 파일을 포함하는 Butane 구성 파일 **40-worker-custom-journald.bu**를 만듭니다.



참고

Butane에 대한 자세한 내용은 "Butane 을 사용하여 머신 구성 생성"을 참조하십시오.

```
variant: openshift
version: 4.9.0
metadata:
  name: 40-worker-custom-journald
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/systemd/journald.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        # Disable rate limiting
        RateLimitInterval=1s
        RateLimitBurst=10000
        Storage=volatile
        Compress=no
        MaxRetentionSec=30s
```

2. Butane을 사용하여 작업자 노드로 전달할 구성이 포함된 **MachineConfig** 개체 파일 **40-worker-custom-journald.yaml**을 생성합니다.

```
$ butane 40-worker-custom-journald.bu -o 40-worker-custom-journald.yaml
```

3. 머신 구성을 풀에 적용합니다.

```
$ oc apply -f 40-worker-custom-journald.yaml
```

4. 새 머신 구성이 적용되고 노드가 저하된 상태에 있는지 확인합니다. 이 작업을 수행하는 데 몇 분 정도 걸릴 수 있습니다. 각 노드에 새 머신 구성이 성공적으로 적용되어 작업자 풀에 진행중인 업데이트가 표시됩니다.

```
$ oc get machineconfigpool
NAME   CONFIG            UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True    False    False    3         3              3              0
34m
worker rendered-worker-d8 False   True     False    3         1              1              0
34m
```

5. 변경 사항이 적용되었는지 확인하려면 작업자 노드에 로그인합니다.

```
$ oc get node | grep worker
ip-10-0-0-1.us-east-2.compute.internal   Ready    worker   39m   v0.0.0-master+$Format:%h$
$ oc debug node/ip-10-0-0-1.us-east-2.compute.internal
Starting pod/ip-10-0-141-142us-east-2computeinternal-debug ...
...
sh-4.2# chroot /host
sh-4.4# cat /etc/systemd/journald.conf
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
sh-4.4# exit
```

추가 리소스

- Butane을 사용하여 머신 구성 생성

3.2.7. RHCOS에 확장 기능 추가

RHCOS는 모든 플랫폼에서 OpenShift Container Platform 클러스터에 공통적인 기능 세트를 제공하도록 설계된 최소한의 컨테이너 지향 RHEL 운영 체제입니다. RHCOS 시스템에 소프트웨어 패키지를 추가하는 것은 일반적으로 권장되지 않지만 MCO는 RHCOS 노드에 최소한의 기능 세트를 추가하는 데 사용할 수 있는 **extensions** 기능을 제공합니다.

현재 다음 확장 기능을 사용할 수 있습니다.

- usbguard: **usbguard** 확장 기능을 추가하면 간섭적인 USB 장치의 공격으로부터 RHCOS 시스템을 보호합니다. 자세한 내용은 USBGuard를 참조하십시오.

다음 프로세서에서는 머신 구성을 사용하여 RHCOS 노드에 하나 이상의 확장 기능을 추가하는 방법을 설명합니다.

사전 요구 사항

- 실행중인 OpenShift Container Platform 클러스터 (버전 4.6 이상)가 있어야합니다.
- 관리 권한이 있는 사용자로 클러스터에 로그인합니다.

절차

1. 확장 기능을 위한 머신 구성을 만듭니다. **MachineConfig extensions** 개체를 포함하는 YAML 파일 (예 : **80-extensions.yaml**)을 만듭니다. 이 예에서는 클러스터에 **usbguard** 확장 기능을 추가하도록 지시합니다.

```
$ cat << EOF > 80-extensions.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 80-worker-extensions
spec:
  config:
    ignition:
      version: 3.2.0
  extensions:
    - usbguard
EOF
```

2. 머신 구성을 클러스터에 추가합니다. 다음을 입력하여 머신 구성을 클러스터에 추가합니다.

```
$ oc create -f 80-extensions.yaml
```

이렇게하면 모든 작업자 노드에 **usbguard**의 rpm 패키지가 설치됩니다.

3. 확장 기능이 적용되었는지 확인합니다.

```
$ oc get machineconfig 80-worker-extensions
```

출력 예

```
NAME                 GENERATEDBYCONTROLLER IGNITIONVERSION AGE
80-worker-extensions                       3.2.0          57s
```

4. 새 머신 구성이 적용되고 노드가 저하된 상태에 있는지 확인합니다. 이 작업을 수행하는 데 몇 분 정도 걸릴 수 있습니다. 각 머신에 새 머신 구성이 성공적으로 적용되면 작업자 풀에 진행중인 업데이트가 표시됩니다.

```
$ oc get machineconfigpool
```

출력 예

```
NAME   CONFIG              UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True    False    False    3          3              3                0
34m
worker rendered-worker-d8 False   True     False    3          1              1                0
34m
```

5. 확장 기능을 확인합니다. 확장 기능이 적용되었는지 확인하려면 다음을 실행하십시오.

```
$ oc get node | grep worker
```

출력 예

```
NAME                                         STATUS  ROLES    AGE   VERSION
ip-10-0-169-2.us-east-2.compute.internal     Ready   worker   102m  v1.22.1
```

```
$ oc debug node/ip-10-0-169-2.us-east-2.compute.internal
```

출력 예

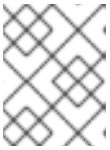
```
...
To use host binaries, run `chroot /host`
sh-4.4# chroot /host
sh-4.4# rpm -q usbguard
usbguard-0.7.4-4.el8.x86_64.rpm
```

3.2.8. 머신 구성 매니페스트에서 사용자 정의 펌웨어 Blob 로드

/usr/lib 에서 펌웨어 Blob의 기본 위치는 읽기 전용이므로 검색 경로를 업데이트하여 사용자 지정 펌웨어 Blob을 찾을 수 있습니다. 이를 통해 RHCOS에서 Blob을 관리하지 않는 경우 머신 구성 매니페스트에 로컬 펌웨어 Blob을 로드할 수 있습니다.

절차

1. 검색 경로를 업데이트하여 로컬 스토리지에 루트 소유 및 쓰기 가능하도록 Butane 구성 파일 **98-worker-firmware-blob.bu** 를 생성합니다. 다음 예제에서는 로컬 워크스테이션의 사용자 지정 Blob 파일을 **/var/lib/firmware** 아래의 노드에 배치합니다.



참고

Butane에 대한 자세한 내용은 "Butane 을 사용하여 머신 구성 생성"을 참조하십시오.

사용자 정의 펌웨어 Blob의 Butane 구성 파일

```
variant: openshift
version: 4.9.0
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 98-worker-firmware-blob
storage:
  files:
  - path: /var/lib/firmware/<package_name> 1
    contents:
      local: <package_name> 2
    mode: 0644 3
openshift:
  kernel_arguments:
    - 'firmware_class.path=/var/lib/firmware' 4
```

1. 펌웨어 패키지가 복사되는 노드에서 경로를 설정합니다.
2. Butane을 실행하는 시스템의 로컬 파일 디렉터리에서 읽은 내용이 포함된 파일을 지정합니다. 로컬 파일의 경로는 다음 단계에서 Butane과 함께 **-- files-dir 옵션을 사용하여 지정해야 하는 files-dir** 디렉터리를 기준으로 합니다.
3. RHCOS 노드에서 파일에 대한 권한을 설정합니다. **0644** 권한을 설정하는 것이 좋습니다.
4. **firmware_class.path** 매개 변수는 로컬 워크스테이션에서 노드의 루트 파일 시스템으로 복사된 사용자 지정 펌웨어 Blob을 찾을 위치에 대한 커널 검색 경로를 사용자 지정합니다. 이 예에서는 **/var/lib/firmware** 를 사용자 지정 경로로 사용합니다.

2. Butane을 실행하여 로컬 워크스테이션 **98-worker-firmware-blob.yaml** 에서 펌웨어 Blob의 사본을 사용하는 **MachineConfig** 오브젝트 파일을 생성합니다. 펌웨어 Blob에는 노드로 전달할 구성이 포함되어 있습니다. 다음 예제에서는 **--files-dir** 옵션을 사용하여 로컬 파일 또는 파일이 있는 워크스테이션의 디렉터리를 지정합니다.

```
$ butane 98-worker-firmware-blob.bu -o 98-worker-firmware-blob.yaml --files-dir
<directory_including_package_name>
```

3. 다음 두 가지 방법 중 하나로 노드에 구성을 적용합니다.
   - 클러스터가 아직 실행되지 않은 경우 매니페스트 파일을 생성한 후 **<installation_directory>/openshift** 디렉터리에 **MachineConfig** 개체 파일을 추가한 다음 클러스터를 계속 작성합니다.
   - 클러스터가 이미 실행중인 경우 다음과 같은 파일을 적용합니다.

     ```
     $ oc apply -f 98-worker-firmware-blob.yaml
     ```

     머신 구성을 완료하기 위해 **MachineConfig** 오브젝트 YAML 파일이 생성됩니다.
4. 향후 **MachineConfig** 오브젝트를 업데이트해야 하는 경우 Butane 구성을 저장합니다.

추가 리소스

- Butane을 사용하여 머신 구성 생성

3.3. MCO 관련 사용자 지정 리소스 구성

MachineConfig 개체를 관리하는 것 외에도 MCO는 **KubeletConfig** 및 **ContainerRuntimeConfig**의 두 가지 사용자 지정 리소스 (CR)를 관리합니다. 이러한 CR을 사용하면 Kubelet 및 CRI-O 컨테이너 런타임 서비스의 작동 방식에 영향을 주는 노드 수준 설정을 변경할 수 있습니다.

3.3.1. KubeletConfig CRD를 생성하여 kubelet 매개변수 편집

kubelet 구성은 현재 Ignition 구성으로 직렬화되어 있으므로 직접 편집할 수 있습니다. 하지만 MCC(Machine Config Controller)에 새 **kubelet-config-controller**도 추가되어 있습니다. 이를 통해 **KubeletConfig** CR(사용자 정의 리소스)을 사용하여 kubelet 매개변수를 편집할 수 있습니다.

참고

kubeletConfig 오브젝트의 필드가 Kubernetes 업스트림에서 kubelet으로 직접 전달되므로 kubelet은 해당 값을 직접 검증합니다. **kubeletConfig** 오브젝트의 값이 유효하지 않으면 클러스터 노드를 사용할 수 없게 될 수 있습니다. 유효한 값은 Kubernetes 설명서를 참조하십시오.

다음 지침 사항을 고려하십시오.

- 해당 풀에 필요한 모든 구성 변경 사항을 사용하여 각 머신 구성 풀에 대해 하나의 **KubeletConfig** CR을 생성합니다. 모든 풀에 동일한 콘텐츠를 적용하는 경우 모든 풀에 대해 하나의 **KubeletConfig** CR만 필요합니다.
- 기존 **KubeletConfig** CR을 편집하여 각 변경 사항에 대한 CR을 생성하는 대신 기존 설정을 수정하거나 새 설정을 추가합니다. 변경 사항을 되돌릴 수 있도록 다른 머신 구성 풀을 수정하거나 임시로 변경하려는 변경 사항만 수정하기 위해 CR을 생성하는 것이 좋습니다.
- 필요에 따라 클러스터당 10개로 제한되는 여러 **KubeletConfig** CR을 생성합니다. 첫 번째 **KubeletConfig** CR의 경우 MCO(Machine Config Operator)는 **kubelet**에 추가된 머신 구성을 생성합니다. 이후 각 CR을 통해 컨트롤러는 숫자 접미사가 있는 다른 **kubelet** 머신 구성을 생성합니다. 예를 들어, **-2** 접미사가 있는 **kubelet** 머신 구성이 있는 경우 다음 **kubelet** 머신 구성에 **-3**이 추가됩니다.

머신 구성을 삭제하려면 제한을 초과하지 않도록 해당 구성을 역순으로 삭제합니다. 예를 들어 **kubelet-2** 머신 구성을 삭제하기 전에 **kubelet-3** 머신 구성을 삭제합니다.

참고

kubelet-9 접미사가 있는 머신 구성이 있고 다른 **KubeletConfig** CR을 생성하는 경우 **kubelet** 머신 구성이 10개 미만인 경우에도 새 머신 구성이 생성되지 않습니다.

KubeletConfig CR 예

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

KubeletConfig 머신 구성 표시 예

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
26m
...
```

다음 프로세스는 작업자 노드의 각 노드에 대한 최대 Pod 수를 구성하는 방법을 보여줍니다.

사전 요구 사항

1. 구성하려는 노드 유형의 정적 **MachineConfigPool** CR와 연관된 라벨을 가져옵니다. 다음 중 하나를 실행합니다.

a. Machine config pool을 표시합니다.

```
$ oc describe machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc describe machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods 1
```

1 라벨이 추가되면 **labels** 아래에 표시됩니다.

b. 라벨이 없으면 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

절차

1. 이 명령은 선택할 수 있는 사용 가능한 머신 구성 오브젝트를 표시합니다.

```
$ oc get machineconfig
```

기본적으로 두 개의 kubelet 관련 구성은 **01-master-kubelet** 및 **01-worker-kubelet**입니다.

2. 노드당 최대 Pod의 현재 값을 확인하려면 다음을 실행합니다.

```
$ oc describe node <node_name>
```

예를 들면 다음과 같습니다.

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

Allocatable 스탠자에서 **value: pods: <value>**를 찾습니다.

출력 예

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

3. 작업자 노드에서 노드당 최대 Pod 수를 설정하려면 kubelet 구성이 포함된 사용자 정의 리소스 파일을 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods 1
  kubeletConfig:
    maxPods: 500 2
```

1 머신 구성 풀에서 레이블을 입력합니다.

2 kubelet 구성을 추가합니다. 이 예에서는 **maxPods**를 사용하여 노드당 최대 Pod를 설정합니다.

참고

kubelet이 API 서버와 통신하는 속도는 QPS(초당 쿼리) 및 버스트 값에 따라 달라집니다. 노드마다 실행되는 Pod 수가 제한된 경우 기본 값인 **50**(**kubeAPIQPS**인 경우) 및 **100**(**kubeAPIBurst**인 경우)이면 충분합니다. 노드에 CPU 및 메모리 리소스가 충분한 경우 kubelet QPS 및 버스트 속도를 업데이트하는 것이 좋습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 라벨을 사용하여 작업자의 머신 구성 풀을 업데이트합니다.

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

b. **KubeletConfig** 오브젝트를 생성합니다.

```
$ oc create -f change-maxPods-cr.yaml
```

c. **KubeletConfig** 오브젝트가 생성되었는지 확인합니다.

```
$ oc get kubeletconfig
```

출력 예

```
NAME            AGE
set-max-pods    15m
```

클러스터의 작업자 노드 수에 따라 작업자 노드가 하나씩 재부팅될 때까지 기다립니다. 작업자 노드가 3개인 클러스터의 경우 약 10~15분이 걸릴 수 있습니다.

4. 변경 사항이 노드에 적용되었는지 확인합니다.

   a. 작업자 노드에서 **maxPods** 값이 변경되었는지 확인합니다.

   ```
   $ oc describe node <node_name>
   ```

   b. **Allocatable** 스탠자를 찾습니다.

   ```
   ...
   Allocatable:
     attachable-volumes-gce-pd:  127
     cpu:                        3500m
     ephemeral-storage:          123201474766
     hugepages-1Gi:              0
     hugepages-2Mi:              0
     memory:                     14225400Ki
     pods:                       500 1
   ...
   ```

   1 이 예에서 **pods** 매개변수는 **KubeletConfig** 오브젝트에 설정한 값을 보고해야 합니다.

5. **KubeletConfig** 오브젝트에서 변경 사항을 확인합니다.

   ```
   $ oc get kubeletconfigs set-max-pods -o yaml
   ```

   다음 예와 같이 **True** 및 **type:Success** 상태가 표시되어야 합니다.

   ```
   spec:
     kubeletConfig:
       maxPods: 500
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: set-max-pods
   status:
     conditions:
     - lastTransitionTime: "2021-06-30T17:04:07Z"
       message: Success
       status: "True"
       type: Success
   ```

3.3.2. CRI-O 매개 변수를 편집하기 위한 ContainerRuntimeConfig CR 작성

특정 MCP(MCP)와 연결된 노드의 OpenShift Container Platform CRI-O 런타임과 관련된 설정을 변경할 수 있습니다. **ContainerRuntimeConfig** 사용자 지정 리소스(CR)를 사용하여 구성 값을 설정하고 MCP와 일치하도록 레이블을 추가합니다. 그런 다음 MCO는 업데이트된 값으로 연결된 노드에서 **crio.conf** 및 **storage.conf** 구성 파일을 다시 빌드합니다.

참고

ContainerRuntimeConfig CR을 사용하여 구현된 변경 사항을 되돌리려면 CR을 삭제해야 합니다. 머신 구성 풀에서 레이블을 제거해도 변경 사항은 복구되지 않습니다.

ContainerRuntimeConfig CR을 사용하여 다음 설정을 수정할 수 있습니다.

- PIDs limit: **pidsLimit** 매개변수는 컨테이너에 허용되는 최대 프로세스 수인 CRI-O **pids_limit** 매개변수를 설정합니다. 기본값은 1024 (**pids_limit = 1024**)입니다.
- Log level: **logLevel** 매개변수는 로그 메시지의 상세 수준인 CRI-O **log_level** 매개변수를 설정합니다. 기본값은 **info** (**log_level = info**)입니다. 기타 다른 옵션에는 **fatal**, **panic**, **error**, **warn**, **debug**, **trace**가 포함됩니다.
- Overlay size: **overlaySize** 매개변수는 컨테이너 이미지의 최대 크기인 CRI-O Overlay 스토리지 드라이버 **size** 매개 변수를 설정합니다.
- Maximum log size: **logSizeMax** 매개변수는 컨테이너 로그 파일에 허용되는 최대 크기인 CRI-O **log_size_max** 매개변수를 설정합니다. 기본값은 무제한입니다 (**log_size_max = -1**). 이 값이 양수로 설정되는 경우 ConMon 읽기 버퍼보다 작지 않게하려면 8192 이상으로 해야합니다. ConMon은 단일 컨테이너의 컨테이너 관리자 (예: Podman 또는 CRI-O)와 OCI 런타임 (예: runc 또는 crun) 간의 통신을 모니터링하는 프로그램입니다.

각 머신 구성 풀에 대해 해당 풀에 필요한 모든 구성 변경 사항이 포함된 하나의 **ContainerRuntimeConfig** CR이 있어야 합니다. 모든 풀에 동일한 콘텐츠를 적용하는 경우 모든 풀에 대해 하나의 **ContainerRuntimeConfig** CR만 있으면 됩니다.

기존 **ContainerRuntimeConfig** CR을 편집하여 새 CR을 생성하는 대신 기존 설정을 편집하거나 새 설정을 추가할 수도 있습니다. 새 **ContainerRuntimeConfig** CR을 생성하여 다른 머신 구성 풀을 수정하거나 임시로 변경하려는 경우에만 변경 사항을 되돌릴 수 있도록 하는 것이 좋습니다.

필요에 따라 여러 **ContainerRuntimeConfig** CR을 생성할 수 있습니다 (클러스터당 10 개 제한). 첫 번째 **ContainerRuntimeConfig** CR의 경우 MCO는 **containerruntime**으로 추가된 머신 구성을 생성합니다. 이후 각 CR을 통해 컨트롤러는 숫자 접미사가 포함된 새 **containerruntime** 머신 구성을 생성합니다. 예를 들어, **-2** 접미사가 있는 **containerruntime** 머신 구성이 있는 경우 다음 **containerruntime** 머신 구성에 **-3**이 추가됩니다.

머신 구성을 삭제하려면 제한을 초과하지 않도록 해당 구성을 역순으로 삭제해야 합니다. 예를 들어 **containerruntime-2** 머신 구성을 삭제하기 전에 **containerruntime-3** 머신 구성을 삭제해야 합니다.

참고

containerruntime-9 접미사가 있는 머신 구성이 있는 경우, 다음 머신 구성에 **ContainerRuntimeConfig** CR이 추가되고, **containerruntime** 머신 구성이 10 개 미만이어도 제한을 초과하여 실패합니다.

여러 ContainerRuntimeConfig CR 표시 예

```
$ oc get ctrcfg
```

출력 예

```
NAME        AGE
ctr-pid     24m
ctr-overlay  15m
```

```
ctr-level    5m45s
```

여러 containerruntime 머신 구성의 예

```
$ oc get mc | grep container
```

출력 예

```
...
01-master-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
57m
...
01-worker-container-runtime                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
57m
...
99-worker-generated-containerruntime            b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0           26m
99-worker-generated-containerruntime-1          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0           17m
99-worker-generated-containerruntime-2          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0           7m26s
...
```

다음 예제에서는 **pids_limit**를 2048로, **log_level**을 **debug**로 설정하고, overlay 크기를 8GB 로 설정하고, **log_size_max**를 무제한으로 설정합니다.

ContainerRuntimeConfig CR 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: '' (1)
 containerRuntimeConfig:
   pidsLimit: 2048 (2)
   logLevel: debug (3)
   overlaySize: 8G (4)
   logSizeMax: "-1" (5)
```

(1) 머신 구성 풀 레이블을 지정합니다.

(2) 선택 사항: 컨테이너에 허용되는 최대 프로세스 수를 설정합니다.

(3) 선택 사항: 로그 메시지의 상세 수준을 설정합니다.

(4) 선택 사항: 컨테이너 이미지의 최대 크기를 지정합니다.

(5) 선택 사항: 컨테이너 로그 파일에 허용되는 최대 크기를 설정합니다. 양수로 설정하는 경우 8192 이상이어야 합니다.

절차

ContainerRuntimeConfig CR을 사용하여 CRI-O 설정을 변경합니다.

1. **ContainerRuntimeConfig** CR의 YAML 파일을 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: " 1
 containerRuntimeConfig: 2
   pidsLimit: 2048
   logLevel: debug
   overlaySize: 8G
   logSizeMax: "-1"
```

1 수정할 머신 구성 풀의 레이블을 지정합니다.

2 필요에 따라 매개변수를 설정합니다.

2. **ContainerRuntimeConfig** CR을 생성합니다.

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

3. CR이 생성되었는지 확인합니다.

```
$ oc get ContainerRuntimeConfig
```

출력 예

```
NAME           AGE
overlay-size   3m19s
```

4. 새 **containerruntime** 머신 구성이 생성되었는지 확인합니다.

```
$ oc get machineconfigs | grep containerrun
```

출력 예

```
99-worker-generated-containerruntime   2c9371fbb673b97a6fe8b1c52691999ed3a1bfc2
3.2.0  31s
```

5. 모두 준비 상태로 표시될 때까지 머신 구성 풀을 모니터링합니다.

```
$ oc get mcp worker
```

출력 예

```
NAME    CONFIG             UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
worker  rendered-worker-169  False    True      False     3          1                  1                  0
9h
```

6. 설정이 CRI-O에 적용되었는지 확인하려면 다음을 실행합니다.

   a. 머신 구성 풀의 노드에 **oc debug** 세션을 열고 **chroot /host**를 실행합니다.

   ```
   $ oc debug node/<node_name>
   ```

   ```
   sh-4.4# chroot /host
   ```

   b. **crio.conf** 파일의 변경 사항을 확인합니다.

   ```
   sh-4.4# crio config | egrep 'log_level|pids_limit|log_size_max'
   ```

   출력 예

   ```
   pids_limit = 2048
   log_size_max = -1
   log_level = "debug"
   ```

   c. 'storage.conf' 파일의 변경 사항을 확인합니다.

   ```
   sh-4.4# head -n 7 /etc/containers/storage.conf
   ```

   출력 예

   ```
   [storage]
     driver = "overlay"
     runroot = "/var/run/containers/storage"
     graphroot = "/var/lib/containers/storage"
     [storage.options]
       additionalimagestores = []
       size = "8G"
   ```

3.3.3. CRI-O를 사용하여 오버레이에 대한 기본 최대 컨테이너 루트 파티션 크기 설정

각 컨테이너의 루트 파티션에는 기본 호스트의 사용 가능한 디스크 공간이 모두 표시됩니다. 다음 지침에 따라 모든 컨테이너의 루트 디스크에 대한 최대 파티션 크기를 설정합니다.

최대 오버레이 크기와 로그 수준 및 PID 제한과 같은 기타 CRI-O 옵션을 구성하려면 다음 **ContainerRuntimeConfig** CRD(사용자 정의 리소스 정의)를 생성할 수 있습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
```

```
    matchLabels:
      custom-crio: overlay-size
  containerRuntimeConfig:
    pidsLimit: 2048
    logLevel: debug
    overlaySize: 8G
```

절차

1. 구성 오브젝트를 생성합니다.

```
$ oc apply -f overlaysize.yml
```

2. 새 CRI-O 구성을 작업자 노드에 적용하려면 작업자 머신 구성 풀을 편집합니다.

```
$ oc edit machineconfigpool worker
```

3. **ContainerRuntimeConfig** CRD에서 설정한 **matchLabels** 이름을 기반으로 **custom-crio** 레이블을 추가합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2020-07-09T15:46:34Z"
  generation: 3
  labels:
    custom-crio: overlay-size
    machineconfiguration.openshift.io/mco-built-in: ""
```

4. 변경 사항을 저장한 다음 머신 구성을 확인합니다.

```
$ oc get machineconfigs
```

새로운 **99-worker-generated-containerruntime** 및 **rendered-worker-xyz** 오브젝트가 생성됩니다.

출력 예

```
99-worker-generated-containerruntime  4173030d89fbf4a7a0976d1665491a4d9a6e54f1
3.2.0             7m42s
rendered-worker-xyz                   4173030d89fbf4a7a0976d1665491a4d9a6e54f1   3.2.0
7m36s
```

5. 해당 오브젝트가 생성된 후 적용할 변경 사항이 있는지 머신 구성 풀을 모니터링합니다.

```
$ oc get mcp worker
```

작업자 노드는 **UPDATING**을 **True**로 표시하고 머신 수, 업데이트된 수 및 기타 세부 정보를 표시합니다.

출력 예

```
NAME   CONFIG            UPDATED   UPDATING   DEGRADED  MACHINECOUNT
```

```
READYMACHINECOUNT  UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz False True False     3           2                2                0
20h
```

완료 후 작업자 노드는 **UPDATING**에서 다시 **False**로 변환되고 **UPDATEDMACHINECOUNT**의 수는 **MACHINECOUNT**의 수와 일치합니다.

출력 예

```
NAME   CONFIG            UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker   rendered-worker-xyz   True     False     False     3       3          3          0
20h
```

작업자 머신을 보면 새로운 8GB 최대 크기 구성이 모든 작업자에 적용되는 것을 확인할 수 있습니다.

출력 예

```
head -n 7 /etc/containers/storage.conf
[storage]
  driver = "overlay"
  runroot = "/var/run/containers/storage"
  graphroot = "/var/lib/containers/storage"
  [storage.options]
    additionalimagestores = []
    size = "8G"
```

컨테이너 내부를 보면 루트 파티션이 이제 8GB가 됩니다.

출력 예

```
~ $ df -h
Filesystem              Size      Used Available Use% Mounted on
overlay                 8.0G      8.0K      8.0G   0% /
```

4장. 설치 후 클러스터 작업

OpenShift Container Platform을 한 후 요구 사항에 맞게 클러스터를 추가로 확장하고 사용자 정의할 수 있습니다.

4.1. 사용 가능한 클러스터 사용자 정의

OpenShift Container Platform 클러스터를 배포한 후 대부분의 클러스터 설정 및 사용자 정의가 완료됩니다. 다양한 *설정 리소스*를 사용할 수 있습니다.

참고

IBM Z에 클러스터를 설치하는 경우 모든 기능을 사용할 수 있는 것은 아닙니다.

설정 리소스를 수정하여 이미지 레지스트리, 네트워킹 설정, 이미지 빌드 동작 및 아이덴티티 제공자와 같은 클러스터의 주요 기능을 설정합니다.

이러한 리소스를 사용하여 기능 제어를 설정하려면 **oc explain** 명령을 사용합니다. (예: **oc explain builds --api-version = config.openshift.io/v1)**

4.1.1. 클러스터 설정 리소스

모든 클러스터 설정 리소스는 전체적으로 범위가 지정되고 (네임 스페이스가 아님) **cluster**라는 이름을 지정할 수 있습니다.

리소스 이름	설명
apiserver.config.openshift.io	인증서 및 인증 기관과 같은 API 서버 설정을 제공합니다.
authentication.config.openshift.io	클러스터의 아이덴티티 공급자 및 인증 구성을 제어합니다.
build.config.openshift.io	클러스터의 모든 빌드에 대한 기본 및 필수 구성되어 있는 설정을 제어합니다.
console.config.openshift.io	로그아웃 동작을 포함하여 웹 콘솔 인터페이스의 동작을 구성합니다.
featuregate.config.openshift.io	기술 프리뷰 기능을 사용할 수 있도록 기능 게이트를 활성화합니다.
image.config.openshift.io	특정 이미지 레지스트리를 처리하는 방법(허용, 허용하지 않음, 비보안, CA 세부 정보)을 구성합니다.
ingress.config.openshift.io	경로의 기본 도메인과 같은 라우팅과 관련된 구성 세부 정보입니다.

리소스 이름	설명
oauth.config.openshift.io	내부 OAuth 서버 흐름과 관련된 아이덴티티 제공자 및 다른 동작을 설정합니다.
project.config.openshift.io	프로젝트 템플릿을 포함하여 프로젝트를 생성하는 방법을 구성합니다.
proxy.config.openshift.io	외부 네트워크 액세스를 필요로 하는 구성 요소에서 사용할 프록시를 정의합니다. 참고: 현재 모든 구성 요소가 이 값을 사용하는 것은 아닙니다.
scheduler.config.openshift.io	정책 및 기본 노드 선택기와 같은 스케줄러 동작을 구성합니다.

4.1.2. Operator 설정 자원

이러한 설정 리소스는 **cluster**라는 클러스터 범위의 인스턴스로 특정 Operator가 소유한 특정 구성 요소의 동작을 제어합니다.

리소스 이름	Description
consoles.operator.openshift.io	브랜딩 사용자 정의와 같은 콘솔 모양을 제어합니다
config.imageregistry.operator.openshift.io	공용 라우팅, 로그 수준, 프록시 설정, 리소스 제약 조건, 복제본 수 및 스토리지 유형과 같은 내부 이미지 레지스트리 설정을 구성합니다.
config.samples.operator.openshift.io	샘플 Operator를 설정하여 클러스터에 설치된 이미지 스트림 및 템플릿 샘플을 제어합니다.

4.1.3. 추가 설정 리소스

이러한 설정 리소스는 특정 구성 요소의 단일 인스턴스를 나타냅니다. 경우에 따라 리소스의 여러 인스턴스를 작성하고 여러 인스턴스를 요청할 수 있습니다. 다른 경우 Operator는 특정 네임 스페이스에서 특정 리소스 인스턴스 이름 만 사용할 수 있습니다. 추가 리소스 인스턴스를 생성하는 방법과 시기에 대한 자세한 내용은 구성 요소 별 설명서를 참조하십시오.

리소스 이름	인스턴스 이름	네임 스페이스	설명
alertmanager.monitoring.coreos.com	**main**	**openshift-monitoring**	Alertmanager 배포 매개 변수를 제어합니다.

리소스 이름	인스턴스 이름	네임 스페이스	설명
ingresscontroller.operator.openshift.io	**default**	**openshift-ingress-operator**	도메인, 복제본 수, 인증서 및 컨트롤러 배치와 같은 Ingress Operator 동작을 구성합니다.

4.1.4. 정보 리소스

이러한 리소스를 사용하여 클러스터에 대한 정보를 검색합니다. 일부 구성에서는 이러한 리소스를 직접 편집해야 할 수 있습니다.

리소스 이름	인스턴스 이름	설명
clusterversion.config.openshift.io	**version**	OpenShift Container Platform 4.9에서는 프로덕션 클러스터에 대한 **ClusterVersion** 리소스를 사용자 정의할 수 없습니다. 대신클러스터 업데이트 프로세스를 수행합니다.
dns.config.openshift.io	**cluster**	클러스터의 DNS 설정을 변경할 수 없습니다. DNS Operator 상태를 표시할 수 있습니다.
infrastructure.config.openshift.io	**cluster**	클러스터가 클라우드 공급자와 상호 작용을 가능하게 하는 설정 세부 정보입니다.
network.config.openshift.io	**cluster**	설치 후 클러스터 네트워크를 변경할 수 없습니다. 네트워크를 사용자 지정하려면 프로세스에 따라 설치 중에 네트워크를 사용자 지정합니다.

4.2. 글로벌 클러스터 풀 시크릿 업데이트

현재 풀 시크릿을 교체하거나 새 풀 시크릿을 추가하여 클러스터의 글로벌 풀 시크릿을 업데이트할 수 있습니다.

사용자가 설치 중에 사용한 레지스트리보다 이미지를 저장하기 위해 별도의 레지스트리를 사용하는 경우 절차가 필요합니다.

사전 요구 사항

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

절차

1. 선택 사항: 기존 풀 시크릿에 새 풀 시크릿을 추가하려면 다음 단계를 완료합니다.
   a. 다음 명령을 입력하여 풀 시크릿을 다운로드합니다.

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> 1
```

1 풀 시크릿 파일에 경로를 제공합니다.

b. 다음 명령을 입력하여 새 풀 시크릿을 추가합니다.

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

1 새 레지스트리를 제공합니다. 동일한 레지스트리에 여러 리포지토리를 포함할 수 있습니다 (예: **--registry="<registry/my-namespace/my-repository>**;).

2 새 레지스트리의 인증 정보를 제공합니다.

3 풀 시크릿 파일에 경로를 제공합니다.

또는 가져오기 시크릿 파일에 대한 수동 업데이트를 수행할 수 있습니다.

2. 다음 명령을 입력하여 클러스터의 글로벌 풀 시크릿을 업데이트합니다.

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

1 새 풀 시크릿 파일의 경로를 제공합니다.

이 업데이트는 모든 노드로 롤아웃되며 클러스터 크기에 따라 작업에 약간의 시간이 걸릴 수 있습니다.

참고

OpenShift Container Platform 4.7.4부터 글로벌 풀 시크릿을 변경해도 더 이상 노드 드레이닝 또는 재부팅이 트리거되지 않습니다.

4.3. 작업자 노드 조정

배포 중에 작업자 노드의 크기를 잘못 조정한 경우 하나 이상의 새 머신 세트를 만들어 확장 한 다음 제거하기 전에 원래 머신 세트를 축소하여 조정할 수 있습니다.

4.3.1. 머신 세트와 머신 구성 풀 간의 차이점

MachineSet 개체는 클라우드 또는 머신 공급자와 관련하여 OpenShift Container Platform 노드를 설명합니다.

MachineConfigPool 개체를 사용하면 **MachineConfigController** 구성 요소가 업그레이드 컨텍스트에서 시스템의 상태를 정의하고 제공할 수 있습니다.

MachineConfigPool 개체를 사용하여 시스템 구성 풀의 OpenShift Container Platform 노드에 대한 업그레이드 방법을 구성할 수 있습니다.

NodeSelector 개체는 **MachineSet**에 대한 참조로 대체할 수 있습니다.

4.3.2. 머신 세트 수동 스케일링

머신 세트에서 머신 인스턴스를 추가하거나 제거하려면 머신 세트를 수동으로 스케일링할 수 있습니다.

이는 완전히 자동화된 설치 프로그램에 의해 프로비저닝된 인프라 설치와 관련이 있습니다. 사용자 지정된 사용자 프로비저닝 인프라 설치에는 머신 세트가 없습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터 및 **oc** 명령행을 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 **oc**에 로그인합니다.

절차

1. 클러스터에 있는 머신 세트를 확인합니다.

```
$ oc get machinesets -n openshift-machine-api
```

   머신 세트는 **<clusterid>-worker-<aws-region-az>** 형식으로 나열됩니다.

2. 클러스터에 있는 머신을 확인합니다.

```
$ oc get machine -n openshift-machine-api
```

3. 삭제하려는 머신에 주석을 설정합니다.

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/cluster-api-delete-machine="true"
```

4. 삭제하려는 노드를 비우고 제외합니다.

```
$ oc adm cordon <node_name>
$ oc adm drain <node_name>
```

5. 머신 세트를 스케일링합니다.

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

   또는 다음을 수행합니다.

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

작은 정보

다음 YAML을 적용하여 머신 세트를 확장할 수도 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

머신 세트를 확장 또는 축소할 수 있습니다. 새 머신을 사용할 수 있을 때 까지 몇 분 정도 소요됩니다.

검증

- 원하는 머신 삭제를 확인합니다.

```
$ oc get machines
```

4.3.3. 머신 세트 삭제 정책

Random, **Newest** 및 **Oldest**의 세 가지 삭제 옵션이 지원됩니다. 기본값은 **Random**입니다. 따라서 머신 세트를 축소할 때 임의의 머신이 선택되어 삭제됩니다. 특정 머신 세트를 변경하고 유스 케이스에 따라 삭제 정책을 설정할 수 있습니다.

```
spec:
  deletePolicy: <delete_policy>
  replicas: <desired_replica_count>
```

삭제 정책에 관계없이 관련 머신에 **machine.openshift.io/cluster-api-delete-machine=true** 주석을 추가하여 특정 머신의 삭제 우선 순위를 지정할 수도 있습니다.

중요

기본적으로 OpenShift Container Platform 라우터 Pod는 작업자에게 배포됩니다. 라우터는 웹 콘솔을 포함한 일부 클러스터 리소스에 액세스해야 하므로 먼저 라우터 Pod를 재배치하지 않는 한 작업자 머신 세트를 **0**으로 스케일링하지 마십시오.

참고

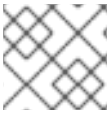
사용자 정의 머신 세트는 특정 노드에서 서비스가 실행되고 작업자 머신 세트가 축소될 때 컨트롤러에서 해당 서비스를 무시해야 하는 유스 케이스에 사용할 수 있습니다. 이로 인해 서비스 중단을 피할 수 있습니다.

4.3.4. 기본 클러스터 수준 노드 선택기 생성

Pod의 기본 클러스터 수준 노드 선택기와 노드의 라벨을 함께 사용하면 클러스터에 생성되는 모든 Pod를 특정 노드로 제한할 수 있습니다.

클러스터 수준 노드 선택기를 사용하여 해당 클러스터에서 Pod를 생성하면 OpenShift Container Platform에서 기본 노드 선택기를 Pod에 추가하고 라벨이 일치하는 노드에 Pod를 예약합니다.

Scheduler Operator CR(사용자 정의 리소스)을 편집하여 클러스터 수준 노드 선택기를 구성합니다. 노드, 머신 세트 또는 머신 구성에 라벨을 추가합니다. 머신 세트에 라벨을 추가하면 노드 또는 머신이 중단되는 경우 새 노드에 라벨이 지정됩니다. 노드 또는 머신이 중단된 경우 노드 또는 머신 구성에 추가된 라벨이 유지되지 않습니다.



참고

Pod에 키/값 쌍을 추가할 수 있습니다. 그러나 기본 키에는 다른 값을 추가할 수 없습니다.

프로세스

기본 클러스터 수준 노드 선택기를 추가하려면 다음을 수행합니다.

1. Scheduler Operator CR을 편집하여 기본 클러스터 수준 노드 선택기를 추가합니다.

```
$ oc edit scheduler cluster
```

노드 선택기를 사용하는 Scheduler Operator CR의 예

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...

spec:
  defaultNodeSelector: type=user-node,region=east (1)
  mastersSchedulable: false
  policy:
    name: ""
```

(1) 적절한 **<key>:<value>** 쌍을 사용하여 노드 선택기를 추가합니다.

변경 후 **openshift-kube-apiserver** 프로젝트의 pod가 재배포될 때까지 기다립니다. 이 작업은 몇 분 정도 걸릴 수 있습니다. 기본 클러스터 수준 노드 선택기는 Pod가 재배포된 후 적용됩니다.

2. 머신 세트를 사용하거나 노드를 직접 편집하여 노드에 라벨을 추가합니다.

- 노드를 생성할 때 머신 세트에서 관리하는 노드에 라벨을 추가하려면 머신 세트를 사용합니다.

  a. 다음 명령을 실행하여 **MachineSet** 오브젝트에 라벨을 추가합니다.

```
$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>"="
<value>","<key>"="<value>"}}]'  -n openshift-machine-api (1)
```

(1) 각 라벨에 **<key>/<value>** 쌍을 추가합니다.

예를 들면 다음과 같습니다.

```
$ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
node","region":"east"}}]'  -n openshift-machine-api
```

작은 정보

다음 YAML을 적용하여 머신 세트에 라벨을 추가할 수도 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b. **oc edit** 명령을 사용하여 라벨이 **MachineSet** 오브젝트에 추가되었는지 확인합니다. 예를 들면 다음과 같습니다.

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

MachineSet 오브젝트의 예

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
  ...
spec:
  ...
  template:
    metadata:
  ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
  ...
```

c. **0** 으로 축소하고 노드를 확장하여 해당 머신 세트와 관련된 노드를 다시 배포합니다. 예를 들면 다음과 같습니다.

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-
machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-
machine-api
```

d. 노드가 준비되고 사용 가능한 경우 **oc get** 명령을 사용하여 라벨이 노드에 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>
```

예를 들면 다음과 같습니다.

```
$ oc get nodes -l type=user-node
```

출력 예

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp   Ready    worker   61s   v1.22.1
```

- 라벨을 노드에 직접 추가합니다.

a. 노드의 **Node** 오브젝트를 편집합니다.

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

예를 들어 노드에 라벨을 지정하려면 다음을 수행합니다.

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node
region=east
```

작은 정보

다음 YAML을 적용하여 노드에 라벨을 추가할 수도 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

b. **oc get** 명령을 사용하여 노드에 라벨이 추가되었는지 확인합니다.

```
$ oc get nodes -l <key>=<value>,<key>=<value>
```

예를 들면 다음과 같습니다.

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

출력 예

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.22.1
```

4.4. 프로덕션 환경의 인프라 머신 세트 생성

머신 세트를 생성하여 기본 라우터, 통합 컨테이너 이미지 레지스트리 및 클러스터 지표 및 모니터링을 위한 구성 요소와 같은 인프라 구성 요소만 호스팅하는 머신을 생성할 수 있습니다. 이러한 인프라 시스템은 환경을 실행하는 데 필요한 총 서브스크립션 수에 포함되지 않습니다.

프로덕션 배포에서는 인프라 구성 요소를 유지하기 위해 3개 이상의 머신 세트를 배포하는 것이 좋습니다. OpenShift Logging 및 Red Hat OpenShift Service Mesh는 모두 Elasticsearch를 배포합니다. 이 경우 서로 다른 노드에 3개의 인스턴스를 설치해야 합니다. 이러한 각 노드는 고가용성을 위해 다양한 가용 영역에 배포할 수 있습니다. 이와 같은 구성에는 가용성 영역마다 하나씩 세 개의 다른 시스템 집합이 필요합니다. 여러 가용성 영역이 없는 글로벌 Azure 리전에서는 가용성 세트를 사용하여 고가용성을 보장할 수 있습니다.

인프라 노드 및 인프라 노드에서 실행할 수 있는 구성 요소에 대한 자세한 내용은 인프라 머신 세트 생성을 참조하십시오.

인프라 노드를 생성하려면 post_installation_configuration/cluster-tasks.adoc#creating-an-infra-node_post-install-cluster-tasks를 사용하여 머신 세트를 사용하거나 머신 구성 풀을 사용할 수 있습니다.

이러한 절차와 함께 사용할 수 있는 샘플 머신 세트의 경우 다른 클라우드의 머신 세트 생성을 참조하십시오.

모든 인프라 구성 요소에 특정 노드 선택기를 적용하면 OpenShift Container Platform에서 해당 라벨을 사용하여 노드에서 해당 워크로드를 예약 합니다.

4.4.1. 머신 세트 만들기

설치 프로그램에서 생성한 컴퓨팅 머신 세트 외에도 고유한 머신 세트를 생성하여 선택한 특정 워크로드의 머신 컴퓨팅 리소스를 동적으로 관리할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터를 배포합니다.
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 **oc**에 로그인합니다.

절차

1. 머신 세트 CR(사용자 지정 리소스) 샘플이 포함된 이름이 **<file_name>.yaml**인 새 YAML 파일을 만듭니다.
   <clusterID> 및 **<role>** 매개 변수 값을 설정해야 합니다.
2. 선택 사항: 특정 필드에 설정할 값이 확실하지 않은 경우 클러스터에서 기존 컴퓨팅 머신 세트를 확인할 수 있습니다.
   a. 클러스터의 컴퓨팅 머신 세트를 나열하려면 다음 명령을 실행합니다.

```
$ oc get machinesets -n openshift-machine-api
```

출력 예

```
NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
```

```
agl030519-vplxk-worker-us-east-1b   1         1         1       1         55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1         55m
agl030519-vplxk-worker-us-east-1d   0         0                           55m
agl030519-vplxk-worker-us-east-1e   0         0                           55m
agl030519-vplxk-worker-us-east-1f   0         0                           55m
```

b. 특정 컴퓨팅 머신 세트 CR(사용자 정의 리소스)의 값을 보려면 다음 명령을 실행합니다.

```
$ oc get machineset <machineset_name> \
  -n openshift-machine-api -o yaml
```

출력 예

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-<role> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      providerSpec: 3
        ...
```

1 클러스터 인프라 ID입니다.

2 기본 노드 레이블입니다.

참고

사용자 프로비저닝 인프라가 있는 클러스터의 경우 컴퓨팅 머신 세트는 **작업자** 및 인프라 유형 머신만 생성할 수 있습니다.

3

컴퓨팅 머신 세트 CR의 <providerSpec> 섹션에 있는 값은 플랫폼에 따라 다릅니다. CR의 <providerSpec> 매개변수에 대한 자세한 내용은 공급자의 샘플 컴퓨팅 머신 세트 CR 구성을 참조하십시오.

3. 다음 명령을 실행하여 **MachineSet CR**을 생성합니다.

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

검증

- 다음 명령을 실행하여 컴퓨팅 머신 세트 목록을 확인합니다.

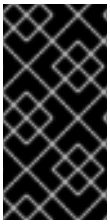
```
$ oc get machineset -n openshift-machine-api
```

출력 예

```
NAME                                 DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-infra-us-east-1a     1         1         1       1           11m
agl030519-vplxk-worker-us-east-1a    1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b    1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c    1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d    0         0                             55m
agl030519-vplxk-worker-us-east-1e    0         0                             55m
agl030519-vplxk-worker-us-east-1f    0         0                             55m
```

새 머신 세트가 사용 가능한 경우**DESIRED** 및 **CURRENT** 값이 일치합니다. 머신 세트를 사용할 수 없는 경우 몇 분 후에 명령을 다시 실행합니다.

4.4.2. 인프라 노드 생성



중요

설치 관리자 프로비저닝 인프라 환경 또는 머신 **API**에서 컨트롤 플레인 노드를 관리하는 클러스터의 인프라 머신 세트 생성을 참조하십시오.

클러스터의 요구 사항은 **infra** 노드라고도 불리는 인프라를 프로비저닝해야 합니다. 설치 프로그램은 컨트롤 플레인 및 작업자 노드에 대한 프로비저닝만 제공합니다. 작업자 노드는 레이블을 통해 인프라 노드 또는 애플리케이션 (**app, app** 이라고도 함)으로 지정할 수 있습니다.

절차

1. 애플리케이션 노드 역할을 수행할 작업자 노드에 레이블을 추가합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/app=""
```

2. 인프라 노드 역할을 수행할 작업자 노드에 레이블을 추가합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

3. 해당 노드에 **infra** 역할 및 **app** 역할이 있는지 확인합니다.

```
$ oc get nodes
```

4. 기본 클러스터 수준 노드 선택기를 생성합니다. 기본 노드 선택기는 모든 네임스페이스에서 생성된 **Pod**에 적용됩니다. 이렇게 하면 **Pod**의 기존 노드 선택기와 교차점이 생성되어 **Pod**의 선택기가 추가로 제한됩니다.

중요

기본 노드 선택기 키가 **Pod** 라벨 키와 충돌하는 경우 기본 노드 선택기가 적용되지 않습니다.

그러나 **Pod**를 예약할 수 없게 만들 수 있는 기본 노드 선택기를 설정하지 마십시오. 예를 들어 **Pod**의 라벨이 **node-role.kubernetes.io/master=""**와 같은 다른 노드 역할로 설정된 경우 기본 노드 선택기를 **node-role.kubernetes.io/infra=""**와 같은 특정 노드 역할로 설정하면 **Pod**를 예약할 수 없게 될 수 있습니다. 따라서 기본 노드 선택기를 특정 노드 역할로 설정할 때 주의해야 합니다.

또는 프로젝트 노드 선택기를 사용하여 클러스터 수준 노드 선택기 키 충돌을 방지할 수 있습니다.

a. **Scheduler** 오브젝트를 편집합니다.

```
$ oc edit scheduler cluster
```

b. 적절한 노드 선택기를 사용하여 **defaultNodeSelector** 필드를 추가합니다.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: topology.kubernetes.io/region=us-east-1 1
...
```

1

이 예제 노드 선택기는 기본적으로 **us-east-1** 리전의 노드에 **Pod**를 배포합니다.

c.

파일을 저장하여 변경 사항을 적용합니다.

인프라 리소스를 새로 레이블이 지정된 인프라 노드로 이동할 수 있습니다.

추가 리소스

- 클러스터 수준 노드 선택기 키 충돌을 방지하기 위해 프로젝트 노드 선택기를 구성하는 방법에 대한 자세한 내용은 프로젝트 노드 선택기를 참조하십시오.

4.4.3. 인프라 머신의 머신 구성 풀 생성

전용 구성을 위한 인프라 머신이 필요한 경우 인프라 풀을 생성해야 합니다.

절차

1. 특정 레이블이 있는 인프라 노드로 할당하려는 노드에 레이블을 추가합니다.

```
$ oc label node <node_name> <label>
```

```
$ oc label node ci-ln-n8mqwr2-f76d1-xscn2-worker-c-6fmtx node-role.kubernetes.io/infra=
```

2. 작업자 역할과 사용자 지정 역할을 모두 포함하는 머신 구성 풀을 머신 구성 선택기로 생성합니다.

```
$ cat infra.mcp.yaml
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
(1)
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: "" (2)
```

(1) 작업자 역할 및 사용자 지정 역할을 추가합니다.

(2) 노드에 추가한 레이블을 **nodeSelector**로 추가합니다.

참고

사용자 지정 머신 구성 풀은 작업자 풀의 머신 구성을 상속합니다. 사용자 지정 풀은 작업자 풀을 대상으로 하는 머신 구성을 사용하지만 사용자 지정 풀을 대상으로 하는 변경 사항만 배포할 수 있는 기능을 추가합니다. 사용자 지정 풀은 작업자 풀에서 리소스를 상속하므로 작업자 풀을 변경하면 사용자 지정 풀에도 영향을 줍니다.

3. **YAML** 파일이 있으면 머신 구성 풀을 생성할 수 있습니다.

```
$ oc create -f infra.mcp.yaml
```

4. 머신 구성을 확인하고 인프라 구성이 성공적으로 렌더링되었는지 확인합니다.

```
$ oc get machineconfig
```

출력 예

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                          365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
00-worker                                          365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0             31d
01-master-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
01-master-kubelet
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
01-worker-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
01-worker-kubelet
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
99-master-ssh                                                 3.2.0             31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             31d
99-worker-ssh                                                 3.2.0             31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             23m
rendered-master-072d4b2da7f88162636902b074e9e28e
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-master-3e88ec72aed3886dec061df60d16d1af
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0             31d
rendered-master-419bee7de96134963a15fdf9dd473b25
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             17d
rendered-master-53f5c91c7661708adce18739cc0f40fb
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             13d
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             7d3h
rendered-master-dc7f874ec77fc4b969674204332da037
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-2640531be11ba43c61d72e82dc634ce6
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0             31d
rendered-worker-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             7d3h
rendered-worker-4f110718fe88e5f349987854a1147755
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             17d
rendered-worker-afc758e194d6188677eb837842d3b379
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0             31d
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0             13d
```

rendered-infra-* 접두사가 있는 새 머신 구성이 표시되어야 합니다.

5. 선택 사항: 사용자 지정 풀에 변경 사항을 배포하려면 **infra**와 같은 사용자 지정 풀 이름을 레이블로 사용하는 머신 구성을 생성합니다. 필수 사항은 아니며 지침 용도로만 표시됩니다. 이렇게 하면 인프라 노드에 고유한 사용자 지정 구성을 적용할 수 있습니다.

참고

새 머신 구성 풀을 생성한 후 **MCO**는 해당 풀에 대해 새로 렌더링된 구성과 해당 풀의 관련 노드를 다시 부팅하여 새 구성을 적용합니다.

a. 머신 구성을 생성합니다.

```
$ cat infra.mc.yaml
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 51-infra
  labels:
    machineconfiguration.openshift.io/role: infra (1)
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - path: /etc/infratest
        mode: 0644
        contents:
          source: data:,infra
```

(1) 노드에 추가한 레이블을 **nodeSelector**로 추가합니다.

b. 머신 구성을 인프라 레이블 노드에 적용합니다.

```
$ oc create -f infra.mc.yaml
```

6. 새 머신 구성 풀을 사용할 수 있는지 확인합니다.

```
$ oc get mcp
```

출력 예

```
NAME    CONFIG                                 UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
infra   rendered-infra-60e35c2e99f42d976e084fa94da4d0fc   True      False      False
1          1                   1                   0                   4m20s
master  rendered-master-9360fdb895d4c131c7c4bebbae099c90  True      False
False      3          3                   3                   0                   91m
worker  rendered-worker-60e35c2e99f42d976e084fa94da4d0fc   True      False      False
2          2                   2                   0                   91m
```

이 예에서는 작업자 노드가 인프라 노드로 변경되었습니다.

추가 리소스

- 사용자 정의 풀에서 인프라 머신 그룹화에 대한 자세한 내용은 머신 구성 풀을 사용한 노드 구성 관리를 참조하십시오.

4.5. 인프라 노드에 머신 세트 리소스 할당

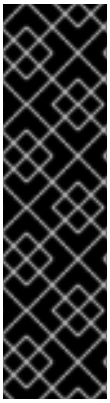
인프라 머신 세트를 생성 한 후 **worker** 및 **infra** 역할이 새 인프라 노드에 적용됩니다. **infra** 역할이 적용된 노드는 **worker** 역할이 적용된 경우에도 환경을 실행하는 데 필요한 총 서브스크립션 수에 포함되지 않습니다.

그러나 인프라 노드에 작업자 역할이 할당되면 사용자 워크로드를 의도치 않게 인프라 노드에 할당할 수 있습니다. 이를 방지하려면 제어하려는 **pod**에 대한 허용 오차를 적용하고 인프라 노드에 테인트를 적

용할 수 있습니다.

4.5.1. 테인트 및 허용 오차를 사용하여 인프라 노드 워크로드 바인딩

infra 및 **worker** 역할이 할당된 인프라 노드가 있는 경우 사용자 워크로드가 할당되지 않도록 노드를 구성해야 합니다.



중요

인프라 노드에 대해 생성된 이중 **infra,worker** 레이블을 유지하고 테인트 및 허용 오차를 사용하여 사용자 워크로드가 예약된 노드를 관리하는 것이 좋습니다. 노드에서 **worker** 레이블을 제거하는 경우 이를 관리할 사용자 지정 풀을 생성해야 합니다. **master** 또는 **worker** 이외의 레이블이 있는 노드는 사용자 지정 풀없이 **MCO**에서 인식되지 않습니다. **worker** 레이블을 유지 관리하면 사용자 정의 레이블을 선택하는 사용자 정의 풀이 없는 경우 기본 작업자 머신 구성 풀에서 노드를 관리할 수 있습니다. **infra** 레이블은 총 서브스크립션 수에 포함되지 않는 클러스터와 통신합니다.

사전 요구 사항

- **OpenShift Container Platform** 클러스터에서 추가 **MachineSet** 개체를 구성합니다.

절차

1. 인프라 노드에 테인트를 추가하여 사용자 워크로드를 예약하지 않도록합니다.

   a. 노드에 테인트가 있는지 확인합니다.

   ```
   $ oc describe nodes <node_name>
   ```

   샘플 출력

   ```
   oc describe node ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Name:               ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Roles:              worker
    ...
   Taints:             node-role.kubernetes.io/infra:NoSchedule
    ...
   ```

이 예에서는 노드에 테인트가 있음을 보여줍니다. 다음 단계에서 **Pod**에 허용 오차를 추가할 수 있습니다.

b.

사용자 워크로드를 예약하지 않도록 테인트를 구성하지 않은 경우 다음을 수행합니다.

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes node1 node-role.kubernetes.io/infra=reserved:NoExecute
```

작은 정보

다음 **YAML**을 적용하여 테인트를 추가할 수도 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      effect: NoExecute
      value: reserved
  ...
```

이 예에서는 키**node-role.kubernetes.io/infra** 및 **taint** 효과 **NoSchedule**이 있는 **node1**에 **taint**를 배치합니다. **NoSchedule** 효과가 있는 노드는 **taint**를 허용하는 **pod**만 예약하지만 기존 **pod**는 노드에서 예약된 상태를 유지할 수 있습니다.

참고

descheduler를 사용하면 노드 **taint**를 위반하는 **pod**가 클러스터에서 제거될 수 있습니다.

2. 라우터, 레지스트리 및 모니터링 워크로드와 같이 인프라 노드에서 예약하려는 **pod** 구성에 대한 허용 오차를 추가합니다. 다음 코드를 **Pod** 개체 사양에 추가합니다.

```
tolerations:
  - effect: NoExecute (1)
    key: node-role.kubernetes.io/infra (2)
    operator: Exists (3)
    value: reserved (4)
```

(1) 노드에 추가한 효과를 지정합니다.

(2) 노드에 추가한 키를 지정합니다.

(3) 노드에 **elasticsearch** 키가 있는 **taint**를 요구하도록 **Exists Operator**를 지정합니다.

(4) 노드에 추가한 키-값 쌍 테인트의 값을 지정합니다.

이 허용 오차는 **oc adm taint** 명령으로 생성된 **taint**와 일치합니다. 이 허용 오차가 있는 **pod**를 인프라 노드에 예약할 수 있습니다.

참고

OLM을 통해 설치된 **Operator**의 **pod**를 인프라 노드로 이동할 수는 없습니다. **Operator pod**를 이동하는 기능은 각 **Operator**의 구성에 따라 다릅니다.

3. 스케줄러를 사용하여 **pod**를 인프라 노드에 예약합니다. 자세한 내용은 노드*에서 pod 배치 제어*에 대한 설명서를 참조하십시오.

추가 리소스

- 노드에 **Pod** 예약에 대한 일반 정보는 스케줄러를 사용하여 **Pod** 배치 제어에서 참조하십시오.

4.6. 인프라 머신 세트로 리소스 이동

일부 인프라 리소스는 기본적으로 클러스터에 배포됩니다. 이를 생성한 인프라 머신 세트로 이동할 수 있습니다.

4.6.1. 라우터 이동

라우터 Pod를 다른 머신 세트에 배포할 수 있습니다. 기본적으로 Pod는 작업자 노드에 배포됩니다.

전제 조건

- OpenShift Container Platform 클러스터에서 추가 머신 세트를 구성합니다.

프로세스

1. 라우터 Operator의 IngressController 사용자 정의 리소스를 표시합니다.

```
$ oc get ingresscontroller default -n openshift-ingress-operator -o yaml
```

명령 출력은 다음 예제와 유사합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: 2019-04-18T12:35:39Z
  finalizers:
  - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  generation: 1
  name: default
  namespace: openshift-ingress-operator
  resourceVersion: "11341"
  selfLink: /apis/operator.openshift.io/v1/namespaces/openshift-ingress-
operator/ingresscontrollers/default
  uid: 79509e05-61d6-11e9-bc55-02ce4781844a
spec: {}
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2019-04-18T12:36:15Z
    status: "True"
    type: Available
  domain: apps.<cluster>.example.com
  endpointPublishingStrategy:
```

```
    type: LoadBalancerService
  selector: ingresscontroller.operator.openshift.io/deployment-
ingresscontroller=default
```

2. **ingresscontroller** 리소스를 편집하고 **infra** 레이블을 사용하도록 **nodeSelector**를 변경합니다.

```
$ oc edit ingresscontroller default -n openshift-ingress-operator
```

```
spec:
  nodePlacement:
    nodeSelector: 1
      matchLabels:
        node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

1 적절한 값이 설정된 **nodeSelector** 매개변수를 이동하려는 구성 요소에 추가합니다. 표시된 형식으로 **nodeSelector**를 사용하거나 노드에 지정된 값에 따라 **<key>: <value>** 쌍을 사용할 수 있습니다. 인프라 노드에 테인트를 추가한 경우 일치하는 허용 오차도 추가합니다.

3. 라우터 **pod**가 **infra** 노드에서 실행되고 있는지 확인합니다.

   a. 라우터 **pod** 목록을 표시하고 실행중인 **pod**의 노드 이름을 기록해 둡니다.

```
$ oc get pod -n openshift-ingress -o wide
```

출력 예

```
NAME                              READY   STATUS        RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
router-default-86798b4b5d-bdlvd   1/1     Running       0          28s     10.130.2.4   ip-
10-0-217-226.ec2.internal   <none>           <none>
router-default-955d875f4-255g8    0/1     Terminating   0          19h     10.129.2.4   ip-
10-0-148-172.ec2.internal   <none>           <none>
```

이 예에서 실행중인 **pod**는 **ip-10-0-217-226.ec2.internal** 노드에 있습니다.

b. 실행중인 **pod**의 노드 상태를 표시합니다.

```
$ oc get node <node_name> 1
```

1 **pod** 목록에서 얻은 **<node_name>**을 지정합니다.

출력 예

```
NAME                           STATUS  ROLES        AGE   VERSION
ip-10-0-217-226.ec2.internal  Ready   infra,worker  17h   v1.22.1
```

역할 목록에 **infra**가 포함되어 있으므로 **pod**가 올바른 노드에서 실행됩니다.

4.6.2. 기본 레지스트리 이동

Pod를 다른 노드에 배포하도록 레지스트리 **Operator**를 구성합니다.

전제 조건

- **OpenShift Container Platform** 클러스터에서 추가 머신 세트를 구성합니다.

프로세스

1. **config/instance** 개체를 표시합니다.

```
$ oc get configs.imageregistry.operator.openshift.io/cluster -o yaml
```

출력 예

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12ffeee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
...
```

2. config/instance 개체를 편집합니다.

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
          topologyKey: kubernetes.io/hostname
        weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector: (1)
    node-role.kubernetes.io/infra: ""
```

```
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/infra
  value: reserved
- effect: NoExecute
  key: node-role.kubernetes.io/infra
  value: reserved
```

1

적절한 값이 설정된 **nodeSelector** 매개변수를 이동하려는 구성 요소에 추가합니다. 표시된 형식으로 **nodeSelector**를 사용하거나 노드에 지정된 값에 따라 **<key>: <value>** 쌍을 사용할 수 있습니다. 인프라 노드에 테인트를 추가한 경우 일치하는 톨러레이션도 추가합니다.

3. 레지스트리 **pod**가 인프라 노드로 이동되었는지 검증합니다.

a. 다음 명령을 실행하여 레지스트리 **pod**가 있는 노드를 식별합니다.

```
$ oc get pods -o wide -n openshift-image-registry
```

b. 노드에 지정된 레이블이 있는지 확인합니다.

```
$ oc describe node <node_name>
```

명령 출력을 확인하고 **node-role.kubernetes.io/infra**가 **LABELS** 목록에 있는지 확인합니다.

4.6.3. 모니터링 솔루션 이동

모니터링 스택에는 **Prometheus, Grafana, Alertmanager**를 포함한 여러 구성 요소가 포함됩니다. **Cluster Monitoring Operator**는 이 스택을 관리합니다. 모니터링 스택을 인프라 노드에 재배포하려면 사용자 정의 구성 맵을 생성하고 적용할 수 있습니다.

프로세스

1. **cluster-monitoring-config** 구성 맵을 편집하고 **infra** 레이블을 사용하도록 **nodeSelector**를 변경합니다.

```
$ oc edit configmap cluster-monitoring-config -n openshift-monitoring
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector: (1)
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    grafana:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
```

```
    effect: NoExecute
 kubeStateMetrics:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
 telemeterClient:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
 openshiftStateMetrics:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
 thanosQuerier:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
```

1

적절한 값이 설정된 **nodeSelector** 매개변수를 이동하려는 구성 요소에 추가합니다. 표시된 형식으로 **nodeSelector**를 사용하거나 노드에 지정된 값에 따라 **<key>: <value>** 쌍을 사용할 수 있습니다. 인프라 노드에 테인트를 추가한 경우 일치하는 톨러레이션도 추가합니다.

2. 모니터링 **pod**가 새 머신으로 이동하는 것을 확인합니다.

```
$ watch 'oc get pod -n openshift-monitoring -o wide'
```

3. 구성 요소가 **infra** 노드로 이동하지 않은 경우 이 구성 요소가 있는 **pod**를 제거합니다.

```
$ oc delete pod -n openshift-monitoring <pod>
```

삭제된 **pod**의 구성 요소가 **infra** 노드에 다시 생성됩니다.

4.6.4. OpenShift Logging 리소스 이동

Elasticsearch 및 **Kibana**와 같은 로깅 하위 시스템 구성 요소를 다른 노드에 배포하도록 **Cluster Logging Operator**를 구성할 수 있습니다. 설치된 위치에서 **Cluster Logging Operator Pod**를 이동할 수 없습니다.

예를 들어 높은 **CPU**, 메모리 및 디스크 요구 사항으로 인해 **Elasticsearch Pod**를 다른 노드로 옮길 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Logging** 및 **Elasticsearch Operator**가 설치되어 있어야 합니다. 이러한 기능은 기본적으로 설치되지 않습니다.

프로세스

1. **openshift-logging** 프로젝트에서 **ClusterLogging** 사용자 정의 리소스(**CR**)를 편집합니다.

```
$ oc edit ClusterLogging instance
```

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:
  collection:
    logs:
      fluentd:
        resources: null
      type: fluentd
  logStore:
    elasticsearch:
      nodeCount: 3
      nodeSelector: 1
        node-role.kubernetes.io/infra: ''
```

```
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
    redundancyPolicy: SingleRedundancy
    resources:
      limits:
        cpu: 500m
        memory: 16Gi
      requests:
        cpu: 500m
        memory: 16Gi
    storage: {}
  type: elasticsearch
managementState: Managed
visualization:
  kibana:
    nodeSelector: 2
      node-role.kubernetes.io/infra: ''
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
    proxy:
      resources: null
    replicas: 1
    resources: null
  type: kibana

...
```

1 2

적절한 값이 설정된 **nodeSelector** 매개변수를 이동하려는 구성 요소에 추가합니다. 표시된 형식으로 **nodeSelector**를 사용하거나 노드에 지정된 값에 따라 **<key>: <value>** 쌍을 사용할 수 있습니다. 인프라 노드에 테인트를 추가한 경우 일치하는 톨러레이션도 추가합니다.

검증

oc get pod -o wide 명령을 사용하여 구성 요소가 이동했는지 확인할 수 있습니다.

예를 들면 다음과 같습니다.

- ip-10-0-147-79.us-east-2.compute.internal 노드에서 Kibana pod를 이동하려고 경우 다음을 실행합니다.

```
$ oc get pod kibana-5b8bdf44f9-ccpq9 -o wide
```

출력 예

```
NAME                     READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
kibana-5b8bdf44f9-ccpq9  2/2     Running   0          27s   10.129.2.18   ip-10-0-147-79.us-
east-2.compute.internal   <none>           <none>
```

- Kibana Pod를 전용 인프라 노드인 ip-10-0-139-48.us-east-2.compute.internal 노드로 이동하려는 경우 다음을 실행합니다.

```
$ oc get nodes
```

출력 예

```
NAME                                         STATUS   ROLES        AGE   VERSION
ip-10-0-133-216.us-east-2.compute.internal   Ready    master       60m   v1.22.1
ip-10-0-139-146.us-east-2.compute.internal   Ready    master       60m   v1.22.1
ip-10-0-139-192.us-east-2.compute.internal   Ready    worker       51m   v1.22.1
ip-10-0-139-241.us-east-2.compute.internal   Ready    worker       51m   v1.22.1
ip-10-0-147-79.us-east-2.compute.internal    Ready    worker       51m   v1.22.1
ip-10-0-152-241.us-east-2.compute.internal   Ready    master       60m   v1.22.1
ip-10-0-139-48.us-east-2.compute.internal    Ready    infra        51m   v1.22.1
```

노드에는 node-role.kubernetes.io/infra : '' 레이블이 있음에 유의합니다.

```
$ oc get node ip-10-0-139-48.us-east-2.compute.internal -o yaml
```

출력 예

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-139-48.us-east-2.compute.internal
  selfLink: /api/v1/nodes/ip-10-0-139-48.us-east-2.compute.internal
  uid: 62038aa9-661f-41d7-ba93-b5f1b6ef8751
  resourceVersion: '39083'
  creationTimestamp: '2020-04-13T19:07:55Z'
  labels:
    node-role.kubernetes.io/infra: ''
...
```

- Kibana pod를 이동하려면 ClusterLogging CR을 편집하여 노드 선택기를 추가합니다.

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:

...

  visualization:
    kibana:
      nodeSelector: 1
        node-role.kubernetes.io/infra: ''
      proxy:
        resources: null
      replicas: 1
      resources: null
    type: kibana
```

1 노드 사양의 레이블과 일치하는 노드 선택기를 추가합니다.

- CR을 저장하면 현재 Kibana pod가 종료되고 새 pod가 배포됩니다.

```
$ oc get pods
```

출력 예

```
NAME                                READY   STATUS      RESTARTS   AGE
cluster-logging-operator-84d98649c4-zb9g7      1/1     Running      0          29m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg   2/2     Running      0          28m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj   2/2     Running      0          28m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78    2/2     Running      0          28m
fluentd-42dzz                       1/1     Running      0          28m
fluentd-d74rq                       1/1     Running      0          28m
fluentd-m5vr9                       1/1     Running      0          28m
fluentd-nkxl7                       1/1     Running      0          28m
fluentd-pdvqb                       1/1     Running      0          28m
fluentd-tflh6                       1/1     Running      0          28m
kibana-5b8bdf44f9-ccpq9             2/2     Terminating  0          4m11s
kibana-7d85dcffc8-bfpfp             2/2     Running      0          33s
```

- 새 **pod**는 **ip-10-0-139-48.us-east-2.compute.internal** 노드에 있습니다.

```
$ oc get pod kibana-7d85dcffc8-bfpfp -o wide
```

출력 예

```
NAME                READY   STATUS      RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
kibana-7d85dcffc8-bfpfp   2/2     Running     0          43s   10.131.0.22   ip-10-0-139-
48.us-east-2.compute.internal   <none>          <none>
```

- 잠시 후 원래 **Kibana pod**가 제거됩니다.

```
$ oc get pods
```

출력 예

```
NAME                                READY   STATUS    RESTARTS   AGE
cluster-logging-operator-84d98649c4-zb9g7      1/1     Running   0          30m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg   2/2     Running   0          29m
```

```
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj   2/2     Running   0          29m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78    2/2     Running   0          29m
fluentd-42dzz                                   1/1     Running   0          29m
fluentd-d74rq                                   1/1     Running   0          29m
fluentd-m5vr9                                   1/1     Running   0          29m
fluentd-nkxl7                                   1/1     Running   0          29m
fluentd-pdvqb                                   1/1     Running   0          29m
fluentd-tflh6                                   1/1     Running   0          29m
kibana-7d85dcffc8-bfpfp                         2/2     Running   0          62s
```

4.7. 클러스터 자동 스케일러 정보

클러스터 자동 스케일러는 현재 배포 요구 사항에 따라 **OpenShift Container Platform** 클러스터의 크기를 조정합니다. 이는 **Kubernetes** 형식의 선언적 인수를 사용하여 특정 클라우드 공급자의 개체에 의존하지 않는 인프라 관리를 제공합니다. 클러스터 자동 스케일러에는 클러스터 범위가 있으며 특정 네임 스페이스와 연결되어 있지 않습니다.

리소스가 부족하여 현재 작업자 노드에서 **Pod**를 예약할 수 없거나 배포 요구 사항을 충족하는 데 다른 노드가 필요한 경우 클러스터 자동 스케일러는 클러스터 크기를 늘립니다. 클러스터 자동 스케일러는 사용자가 지정한 제한을 초과하여 클러스터 리소스를 늘리지 않습니다.

클러스터 자동 스케일러는 컨트롤 플레인 노드를 관리하지 않더라도 클러스터의 모든 노드에서 총 메모리, **CPU** 및 **GPU**를 계산합니다. 이러한 값은 단일 시스템 지향이 아닙니다. 전체 클러스터에 있는 모든 리소스를 집계한 것입니다. 예를 들어 최대 메모리 리소스 제한을 설정하는 경우 클러스터 자동 스케일러에는 현재 메모리 사용량을 계산할 때 클러스터의 모든 노드가 포함됩니다. 그런 다음 해당 계산을 사용하여 클러스터 자동 스케일러에 더 많은 작업자 리소스를 추가할 수 있는 용량이 있는지 확인합니다.

중요

작성한 **ClusterAutoscaler** 리솟스 정의의 **maxNodesTotal** 값이 클러스터에서 예상되는 총 머신 수를 대응하기에 충분한 크기의 값인지 확인합니다. 이 값에는 컨트롤 플레인 머신 수 및 확장 가능한 컴퓨팅 머신 수가 포함되어야 합니다.

10초마다 클러스터 자동 스케일러는 클러스터에서 불필요한 노드를 확인하고 제거합니다. 클러스터 자동 스케일러는 다음 조건이 적용되는 경우 제거할 노드를 고려합니다.

- 노드에서 실행 중인 모든 **Pod**의 **CPU** 및 메모리 요청 합계는 노드에서 할당된 리소스의 **50%** 미만입니다.

- 클러스터 자동 스케일러는 노드에서 실행 중인 모든 포드를 다른 노드로 이동할 수 있습니다.
- 클러스터 자동 확장기에는 축소 비활성화 주석이 없습니다.

노드에 다음 유형의 **pod**가 있는 경우 클러스터 자동 스케일러는 해당 노드를 제거하지 않습니다.

- 제한적인 **PDB (Pod Disruption Budgets)**가 있는 **pod**
- 기본적으로 노드에서 실행되지 않는 **Kube** 시스템 **pod**
- **PDB**가 없거나 제한적인 **PDB**가있는 **Kube** 시스템 **pod**
- **deployment, replica set** 또는 **stateful set**와 같은 컨트롤러 객체가 지원하지 않는 **pod**
- 로컬 스토리지가 있는 **pod**
- 리소스 부족, 호환되지 않는 노드 선택기 또는 어피니티**(affinity)**, 안티**-**어피니티**(anti-affinity)** 일치 등으로 인해 다른 위치로 이동할 수 없는 **pod**
- **"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"** 주석이없는 경우**"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"** 주석이 있는 **pod**

예를 들어 최대 **CPU** 제한을 **64**코어로 설정하고 각각 **8**개의 코어만 있는 머신을 생성하도록 클러스터 자동 스케일러를 구성합니다. 클러스터가 **30**개 코어로 시작하는 경우 클러스터 자동 스케일러는 총 **62**개의 코어가 **32**개의 노드를 더 추가할 수 있습니다.

클러스터 자동 스케일러를 구성하면 추가 사용 제한이 적용됩니다.

- 자동 스케일링된 노드 그룹에 있는 노드를 직접 변경하지 마십시오. 동일한 노드 그룹 내의 모든 노드는 동일한 용량 및 레이블을 가지며 동일한 시스템 **pod**를 실행합니다.

- pod 요청을 지정합니다.
- pod가 너무 빨리 삭제되지 않도록 해야 하는 경우 적절한 PDB를 구성합니다.
- 클라우드 제공자 할당량이 구성하는 최대 노드 풀을 지원할 수 있는 충분한 크기인지를 확인합니다.
- 추가 노드 그룹 Autoscaler, 특히 클라우드 제공자가 제공하는 Autoscaler를 실행하지 마십시오.

HPA (Horizond Pod Autoscaler) 및 클러스터 자동 스케일러는 다른 방식으로 클러스터 리소스를 변경합니다. HPA는 현재 CPU 로드를 기준으로 배포 또는 복제 세트의 복제 수를 변경합니다. 로드가 증가하면 HPA는 클러스터에 사용 가능한 리소스 양에 관계없이 새 복제본을 만듭니다. 리소스가 충분하지 않은 경우 클러스터 자동 스케일러는 리소스를 추가하고 HPA가 생성한 pod를 실행할 수 있도록 합니다. 로드가 감소하면 HPA는 일부 복제를 중지합니다. 이 동작으로 일부 노드가 충분히 활용되지 않거나 완전히 비어 있을 경우 클러스터 자동 스케일러가 불필요한 노드를 삭제합니다.

클러스터 자동 스케일러는 pod 우선 순위를 고려합니다. Pod 우선 순위 및 선점 기능을 사용하면 클러스터에 충분한 리소스가 없는 경우 우선 순위에 따라 pod를 예약할 수 있지만 클러스터 자동 스케일러는 클러스터에 모든 pod를 실행하는 데 필요한 리소스가 있는지 확인합니다. 두 기능을 충족하기 위해 클러스터 자동 스케일러에는 우선 순위 컷오프 기능이 포함되어 있습니다. 이 컷오프 기능을 사용하여 "best-effort" pod를 예약하면 클러스터 자동 스케일러가 리소스를 늘리지 않고 사용 가능한 예비 리소스가 있을 때만 실행됩니다.

컷오프 값보다 우선 순위가 낮은 pod는 클러스터가 확장되지 않거나 클러스터가 축소되지 않도록합니다. pod를 실행하기 위해 추가된 새 노드가 없으며 이러한 pod를 실행하는 노드는 리소스를 확보하기 위해 삭제될 수 있습니다.

4.7.1. ClusterAutoscaler 리소스 정의

이 ClusterAutoscaler 리소스 정의는 클러스터 자동 스케일러의 매개 변수 및 샘플 값을 표시합니다.

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10 (1)
  resourceLimits:
    maxNodesTotal: 24 (2)
```

```
  cores:
    min: 8 3
    max: 128 4
  memory:
    min: 4 5
    max: 256 6
  gpus:
    - type: nvidia.com/gpu 7
      min: 0 8
      max: 16 9
    - type: amd.com/gpu
      min: 0
      max: 4
scaleDown: 10
  enabled: true 11
  delayAfterAdd: 10m 12
  delayAfterDelete: 5m 13
  delayAfterFailure: 30s 14
  unneededTime: 5m 15
```

1

클러스터 자동 스케일러가 추가 노드를 배포하도록 하려면 **pod**가 초과해야하는 우선 순위를 지정합니다. 32 비트 정수 값을 입력합니다. **podPriorityThreshold** 값은 각 **pod**에 할당한 **PriorityClass**의 값과 비교됩니다.

2

배포할 최대 노드 수를 지정합니다. 이 값은 **Autoscaler**가 제어하는 머신뿐 만 아니라 클러스터에 배치 된 총 머신 수입니다. 이 값이 모든 컨트롤 플레인 및 컴퓨팅 머신과 **MachineAutoscaler** 리소스에 지정한 총 복제본 수에 대응할 수 있을 만큼 충분한 크기의 값인지 확인합니다.

3

클러스터에 배포할 최소 코어 수를 지정합니다.

4

클러스터에 배포할 최대 코어 수를 지정합니다.

5

클러스터에서 최소 메모리 크기를 **GiB** 단위로 지정합니다.

6

클러스터에서 최대 메모리 크기를 **GiB**단위로 지정합니다.

7

선택 옵션으로 배포할 GPU 노드 유형을 지정합니다. nvidia.com/gpu 및 amd.com/gpu 만 유효한 유형입니다.

8

클러스터에 배포할 최소 GPU 수를 지정합니다.

9

클러스터에 배포할 최대 GPU 수를 지정합니다.

10

이 섹션에서는 ns, us, ms, s, m 및 h를 포함하여 유효한 ParseDuration 간격을 사용하여 각 작업에 대해 대기하는 기간을 지정할 수 있습니다.

11

클러스터 자동 스케일러가 불필요한 노드를 제거할 수 있는지 여부를 지정합니다.

12

선택 사항으로 노드가 최근에 *추가된* 후 노드를 삭제하기 전까지 대기할 기간을 지정합니다. 값을 지정하지 않으면 기본값으로 10m이 사용됩니다.

13

최근에 노드가 *삭제 된* 후 노드를 삭제하기 전에 대기할 기간을 지정하십시오. 값을 지정하지 않으면 기본값으로 10s가 사용됩니다.

14

스케일 다운 실패 후 노드를 삭제하기 전에 대기할 기간을 지정합니다. 값을 지정하지 않으면 기본값으로 3m가 사용됩니다.

15

불필요한 노드가 삭제되기 전 까지 걸기는 기간을 지정합니다. 값을 지정하지 않으면 기본값으로 10m이 사용됩니다.

참고

스케일링 작업을 수행할 때 클러스터 자동 스케일러는 클러스터에서 배포할 최소 및 최대 코어 수 또는 메모리 양과 같은 **ClusterAutoscaler** 리소스 정의에 설정된 범위 내에 유지됩니다. 그러나 클러스터 자동 스케일러는 해당 범위 내에 있는 클러스터의 현재 값을 수정하지 않습니다.

최소 및 최대 **CPU**, 메모리 및 **GPU** 값은 클러스터의 모든 노드에서 해당 리소스를 계산하여 결정합니다(클러스터 자동 스케일러가 노드를 관리하지 않는 경우에도). 예를 들어 클러스터 자동 스케일러가 컨트롤 플레인 노드를 관리하지 않더라도 컨트롤 플레인 노드는 클러스터의 총 메모리에서 고려됩니다.

4.7.2. 클러스터 자동 스케일러 배포

클러스터 자동 스케일러를 배포하려면 **ClusterAutoscaler** 리소스의 인스턴스를 만듭니다.

절차

1. 사용자 정의된 리소스 정의가 포함된 **ClusterAutoscaler**리소스의 **YAML** 파일을 만듭니다.

2. 클러스터에서 리소스를 생성합니다.

```
$ oc create -f <filename>.yaml 1
```

1 **<filename>**은 사용자 정의 리소스 파일의 이름입니다.

4.8. 머신 자동 스케일러 정보

머신 자동 스케일러는 **OpenShift Container Platform** 클러스터에 배포하는 머신 세트의 **Machine** 수를 조정합니다. 기본 **worker** 머신 세트와 사용자가 만든 다른 머신 세트를 모두 확장할 수 있습니다. 머신 자동 스케일러는 클러스터에 더 많은 배포를 지원하기에 충분한 리소스가 없으면 **Machine**을 추가합니다. 최소 또는 최대 인스턴스 수와 같은 **MachineAutoscaler** 리소스의 값에 대한 모든 변경 사항은 대상이 되는 머신 세트에 즉시 적용됩니다.

중요

머신을 확장하려면 클러스터 자동 스케일러의 머신 자동 스케일러를 배포해야합니다. 클러스터 자동 스케일러는 머신 자동 스케일러가 설정한 머신 세트의 주석을 사용하여 확장할 수있는 리소스를 결정합니다. 머신 자동 스케일러도 정의하지 않고 클러스터 자동 스케일러를 정의하면 클러스터 자동 스케일러는 클러스터를 확장하지 않습니다.

4.8.1. MachineAutoscaler 리소스 정의

이 MachineAutoscaler 리소스 정의는 머신 자동 스케일러의 매개 변수 및 샘플 값을 표시합니다.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "worker-us-east-1a" (1)
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1 (2)
  maxReplicas: 12 (3)
  scaleTargetRef: (4)
    apiVersion: machine.openshift.io/v1beta1
    kind: MachineSet (5)
    name: worker-us-east-1a (6)
```

(1)

머신 자동 스케일러 이름을 지정합니다. 이 머신 자동 스케일러가 스케일링하는 머신 세트 보다 쉽게 식별할 수 있도록 스케일링할 머신 세트의 이름을 지정하거나 포함합니다. 머신 세트 이름의 형식은 <clusterid>-<machineset>-<region>입니다.

(2)

클러스터 자동 스케일러가 클러스터 스케일링을 시작한 후 지정된 영역에 남아 있어야하는 지정된 유형의 최소 머신 수를 지정하십시오. AWS, GCP, Azure, RHOSP, vSphere에서 실행중인 경우 이 값을 0으로 설정할 수 있습니다. 다른 공급 업체의 경우 이 값을 0으로 설정하지 마십시오.

특수 워크로드에 사용되는 비용이 많이 드는 하드웨어 또는 대규모 머신으로 머신 세트를 확장하는 등의 사용 사례에 이 값을 0으로 설정하여 비용을 절감할 수 있습니다. 머신을 사용하지 않는 경우 클러스터 자동 스케일러가 머신 세트를 0으로 축소합니다.

중요

설치 관리자 프로비저닝 인프라의 **OpenShift Container Platform** 설치 프로세스 중에 생성된 세 개의 컴퓨팅 머신 세트의 **spec.minReplicas** 값을 **0** 으로 설정하지 마십시오.

3

클러스터 자동 스케일러가 클러스터 스케일링을 초기화한 후 지정된 영역에 배포할 수 있는 지정된 유형의 최대 머신 수를 지정합니다. **ClusterAutoscaler** 리소스 정의에서 **maxNodesTotal** 값이 머신 자동 스케일러가 머신 수를 배포할 수 있는 충분한 크기의 값임을 확인합니다.

4

이 섹션에서는 스케일링할 기존 머신 세트를 설명하는 값을 지정합니다.

5

kind 매개 변수 값은 항상 **MachineSet**입니다.

6

metadata.name 매개 변수 값에 표시된 것처럼 **name** 값은 기존 머신 세트의 이름과 일치해야 합니다.

4.8.2. 머신 자동 스케일러 배포

머신 자동 스케일러를 배포하려면 **MachineAutoscaler** 리소스의 인스턴스를 만듭니다.

절차

1. 사용자 정의된 리소스 정의가 포함된 **MachineAutoscaler** 리소스의 **YAML** 파일을 생성합니다.

2. 클러스터에서 리소스를 생성합니다.

```
$ oc create -f <filename>.yaml 1
```

1

<filename>은 사용자 정의 리소스 파일의 이름입니다.

4.9. FEATUREGATE를 사용하여 기술 프리뷰 기능 활성화

FeatureGate 사용자 정의 리소스 (CR)를 편집하여 클러스터의 모든 노드에 대해 현재 기술 프리뷰 기능의 일부를 켤 수 있습니다.

4.9.1. FeatureGate 이해

FeatureGate 사용자 정의 리소스 (CR)를 사용하여 클러스터에서 특정 기능 세트를 활성화할 수 있습니다. 기능 세트는 기본적으로 활성화되어 있지 않은 OpenShift Container Platform 기능 컬렉션입니다.

FeatureGate CR을 사용하여 다음 기능 세트를 활성화할 수 있습니다.

- TechPreviewNoUpgrade. 이 기능 세트는 현재 기술 프리뷰 기능의 하위 집합입니다. 이 기능 세트를 사용하면 프로덕션 클러스터에서 기능을 비활성화하는 동안 테스트 클러스터에서 이러한 기술 프리뷰 기능을 활성화할 수 있습니다. 이 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지합니다. 이 기능 세트는 프로덕션 클러스터에서는 권장되지 않습니다.

주의

클러스터에서 TechPreviewNoUpgrade 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

기능 세트를 통해 다음과 같은 기술 프리뷰 기능을 활성화할 수 있습니다.

  - Azure Disk CSI Driver Operator. Microsoft Azure Disk Storage용 CSI(Container Storage Interface) 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.
  - VMware vSphere CSI Driver Operator. VMI(Virtual Machine Disk) 볼륨에 CSI(Container Storage Interface) VMware vSphere 드라이버를 사용하여 PV(영구 볼륨)를 프로비저닝할 수 있습니다.
  - CSI 자동 마이그레이션 지원되는 in-tree 볼륨 플러그인을 동등한 CSI(Container

Storage Interface) 드라이버로 자동 마이그레이션을 활성화합니다. 다음을 위한 기술 프리뷰로 사용할 수 있습니다.

- AWS(Amazon Web Services) EBS(Elastic Block Storage)
- OpenStack Cinder
- Azure Disk
- Azure File
- Google Cloud Platform 영구 디스크(CSI)
- VMware vSphere

- Cluster Cloud Controller Manager Operator. in-tree 클라우드 컨트롤러가 아닌 Cluster Cloud Controller Manager Operator를 활성화합니다. 다음을 위한 기술 프리뷰로 사용할 수 있습니다.
  - AWS(Amazon Web Services)
  - Microsoft Azure
  - Red Hat OpenStack Platform (RHOSP)
- Insights Operator. Red Hat OpenShift Cluster Manager에서 RHEL SCA(Simple Content Access) 인증서를 가져올 수 있습니다.

4.9.2. 웹 콘솔을 사용하여 기능 세트 활성화

OpenShift Container Platform 웹 콘솔을 사용하여 FeatureGate CR(사용자 정의 리소스)을 편집하여 클러스터의 모든 노드에 대해 기능 세트를 활성화할 수 있습니다.

절차

기능 세트를 활성화하려면 다음을 수행합니다.

1. **OpenShift Container Platform** 웹 콘솔에서 관리 → 사용자 지정 리소스 정의 페이지로 전환합니다.

2. 사용자 지정 리소스 정의 페이지에서 **FeatureGate**를 클릭합니다.

3. 사용자 정의 리소스 정의 세부 정보 페이지에서 인스턴스 탭을 클릭합니다.

4. 클러스터 기능 게이트를 클릭한 다음 **YAML** 탭을 클릭합니다.

5. 특정 기능 세트를 추가하려면 클러스터 인스턴스를 편집합니다.

주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

FeatureGate 사용자 지정 리소스 샘플

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster 1
....

spec:
  featureSet: TechPreviewNoUpgrade 2
```

1 **FeatureGate CR의 이름은 cluster이어야 합니다.**

2 **활성화할 기능 세트를 추가합니다.**

- **TechPreviewNoUpgrade를 사용하면 특정 기술 프리뷰 기능을 사용할 수 있습니다.**

변경 사항을 저장하면 새 머신 구성이 생성되고 머신 구성 풀이 업데이트되고 변경 사항이 적용되는 동안 각 노드의 스케줄링이 비활성화됩니다.

검증

노드가 준비 상태로 돌아간 후 노드의 kubelet.conf 파일을 확인하여 기능 게이트가 활성화되어 있는지 확인할 수 있습니다.

1. **웹 콘솔의 관리자 화면에서 컴퓨팅 → 노드로 이동합니다.**
2. **노드를 선택합니다.**
3. **노드 세부 정보 페이지에서 터미널 을 클릭합니다.**
4. **터미널 창에서 root 디렉토리를 /host:로 변경합니다.**

```
sh-4.2# chroot /host
```

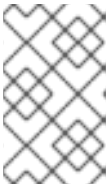
5. **kubelet.conf 파일을 확인합니다.**

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

샘플 출력

```
...
featureGates:
 InsightsOperatorPullingSCA: true,
 LegacyNodeRoleBehavior: false
...
```

true 로 나열된 기능은 클러스터에서 활성화됩니다.



참고

나열된 기능은 **OpenShift Container Platform** 버전에 따라 다릅니다.

4.9.3. CLI를 사용하여 기능 세트 활성화

OpenShift CLI(oc)를 사용하여 **FeatureGate CR**(사용자 정의 리소스)을 편집하여 클러스터의 모든 노드에 대해 기능 세트를 활성화할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**가 설치되어 있습니다.

절차

기능 세트를 활성화하려면 다음을 수행합니다.

1. **cluster**라는 **FeatureGate CR**을 편집합니다.

```
$ oc edit featuregate cluster
```

주의

클러스터에서 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 취소할 수 없으며 마이너 버전 업데이트를 방지할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화해서는 안 됩니다.

FeatureGate 사용자 지정 리소스 샘플

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster 1
spec:
  featureSet: TechPreviewNoUpgrade 2
```

1

FeatureGate CR의 이름은 **cluster**이어야 합니다.

2

활성화할 기능 세트를 추가합니다.

- **TechPreviewNoUpgrade**를 사용하면 특정 기술 프리뷰 기능을 사용할 수 있습니다.

변경 사항을 저장하면 새 머신 구성이 생성되고 머신 구성 풀이 업데이트되고 변경 사항이 적용되는 동안 각 노드의 스케줄링이 비활성화됩니다.

검증

노드가 준비 상태로 돌아간 후 노드의 **kubelet.conf** 파일을 확인하여 기능 게이트가 활성화되어 있는지 확인할 수 있습니다.

1. 웹 콘솔의 관리자 화면에서 컴퓨팅 → 노드로 이동합니다.

2. 노드를 선택합니다.

3. 노드 세부 정보 페이지에서 터미널 을 클릭합니다.

4. 터미널 창에서 **root** 디렉토리를 **/host:**로 변경합니다.

```
sh-4.2# chroot /host
```

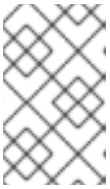
5. **kubelet.conf** 파일을 확인합니다.

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

샘플 출력

```
...
featureGates:
 InsightsOperatorPullingSCA: true,
 LegacyNodeRoleBehavior: false
...
```

true 로 나열된 기능은 클러스터에서 활성화됩니다.



참고

나열된 기능은 **OpenShift Container Platform** 버전에 따라 다릅니다.

4.10. ETCD 작업

etcd를 백업하거나 **etcd** 암호화를 활성화 또는 비활성화하거나 **etcd** 데이터 조각 모음을 실행합니다.

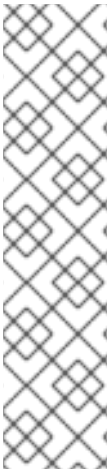
4.10.1. etcd 암호화 정보

기본적으로 **etcd** 데이터는 **OpenShift Container Platform**에서 암호화되지 않습니다. 클러스터에 **etcd** 암호화를 사용하여 추가 데이터 보안 계층을 제공할 수 있습니다. 예를 들어 **etcd** 백업이 잘못된 당사자에게 노출되는 경우 중요한 데이터의 손실을 방지할 수 있습니다.

etcd 암호화를 활성화하면 다음 **OpenShift API** 서버 및 쿠버네티스 **API** 서버 리소스가 암호화됩니다.

- 보안
- 구성 맵
- 라우트
- **OAuth** 액세스 토큰
- **OAuth** 승인 토큰

etcd 암호화를 활성화하면 암호화 키가 생성됩니다. 이 키는 매주 순환됩니다. **etcd** 백업에서 복원하려면 이 키가 있어야 합니다.



참고

etcd 암호화는 키가 아닌 값만 암호화합니다. 리소스 유형, 네임스페이스 및 오브젝트 이름은 암호화되지 않습니다.

백업 중에 **etcd** 암호화가 활성화되면 *static_kuberesources_<datetimestamp>.tar.gz* 파일에 **etcd** 스냅샷의 암호화 키가 포함되어 있습니다. 보안상의 이유로 이 파일을 **etcd** 스냅샷과 별도로 저장합니다. 그러나 이 파일은 해당 **etcd** 스냅샷에서 이전 **etcd** 상태를 복원하는데 필요합니다.

4.10.2. etcd 암호화 활성화

etcd 암호화를 활성화하여 클러스터에서 중요한 리소스를 암호화할 수 있습니다.

주의

초기 암호화 프로세스가 완료될 때까지 **etcd** 리소스를 백업하지 마십시오. 암호화 프로세스가 완료되지 않으면 백업이 부분적으로만 암호화될 수 있습니다.

etcd 암호화를 활성화하면 다음과 같은 몇 가지 변경이 발생할 수 있습니다.

- **etcd** 암호화는 몇 가지 리소스의 메모리 사용에 영향을 줄 수 있습니다.
- 리더가 백업을 제공해야 하기 때문에 백업 성능에 일시적인 영향을 미칠 수 있습니다.
- 디스크 I/O는 백업 상태를 수신하는 노드에 영향을 줄 수 있습니다.

사전 요구 사항

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

프로세스

1. **APIServer** 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. 암호화 필드 유형을 **aescbc**로 설정합니다.

```
spec:
  encryption:
    type: aescbc 1
```

1. **aescbc** 유형은 **PKCS# 7** 패딩 및 **32**바이트 키가 있는 **AES-CBC**가 암호화를 수행하는 데 사용됨을 나타냅니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

암호화 프로세스가 시작됩니다. 클러스터 크기에 따라 이 프로세스를 완료하는 데 20분 이상 걸릴 수 있습니다.

4. etcd 암호화에 성공했는지 확인합니다.

a. OpenShift API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스가 성공적으로 암호화되었는지 확인합니다.

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호화에 성공하면 출력에 EncryptionCompleted가 표시됩니다.

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

출력에 EncryptionInProgress가 표시되는 경우에도 암호화는 계속 진행 중입니다. 몇 분 기다린 후 다시 시도합니다.

b. 쿠버네티스 API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스가 성공적으로 암호화되었는지 확인합니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호화에 성공하면 출력에 EncryptionCompleted가 표시됩니다.

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

출력에 EncryptionInProgress가 표시되는 경우에도 암호화는 계속 진행 중입니다. 몇 분 기다린 후 다시 시도합니다.

c. OpenShift OAuth API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스가 성공적으로 암호화되었는지 확인합니다.

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}
{"\n"}'
```

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

출력에 **EncryptionInProgress**가 표시되는 경우에도 암호화는 계속 진행 중입니다. 몇 분 기다린 후 다시 시도합니다.

4.10.3. etcd 암호화 비활성화

클러스터에서 **etcd** 데이터의 암호화를 비활성화할 수 있습니다.

사전 요구 사항

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

프로세스

1. **APIServer** 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. 암호화 필드 유형을 **identity**로 설정합니다.

```
spec:
 encryption:
  type: identity 1
```

1 **identity** 유형이 기본값이며, 이는 암호화가 수행되지 않음을 의미합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

암호 해독 프로세스가 시작됩니다. 클러스터 크기에 따라 이 프로세스를 완료하는 데 20분 이상 걸릴 수 있습니다.

4. etcd 암호 해독에 성공했는지 확인합니다.

a. OpenShift API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스의 암호가 성공적으로 해독되었는지 확인합니다.

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호 해독에 성공하면 출력에 DecryptionCompleted가 표시됩니다.

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

출력에 DecryptionInProgress가 표시되면 암호 해독이 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

b. 쿠버네티스 API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스의 암호가 성공적으로 해독되었는지 확인합니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

암호 해독에 성공하면 출력에 DecryptionCompleted가 표시됩니다.

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

출력에 DecryptionInProgress가 표시되면 암호 해독이 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

c. OpenShift API 서버의 Encrypted 상태 조건을 검토하여 해당 리소스의 암호가 성공적으로 해독되었는지 확인합니다.

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}
{"\n"}'
```

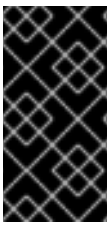
암호 해독에 성공하면 출력에 **DecryptionCompleted**가 표시됩니다.

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

출력에 **DecryptionInProgress**가 표시되면 암호 해독이 여전히 진행 중임을 나타냅니다. 몇 분 기다린 후 다시 시도합니다.

4.10.4. etcd 데이터 백업

다음 단계에 따라 **etcd** 스냅샷을 작성하고 정적 **pod**의 리소스를 백업하여 **etcd** 데이터를 백업합니다. 이 백업을 저장하여 **etcd**를 복원해야하는 경우 나중에 사용할 수 있습니다.



중요

단일 컨트롤 플레인 호스트의 백업만 저장합니다. 클러스터의 각 컨트롤 플레인 호스트에서 백업을 수행하지 마십시오.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 클러스터 전체의 프록시가 활성화되어 있는지 확인해야 합니다.

작은 정보

oc get proxy cluster -o yaml의 출력을 확인하여 프록시가 사용 가능한지 여부를 확인할 수 있습니다. **httpProxy, httpsProxy** 및 **noProxy** 필드에 값이 설정되어 있으면 프록시가 사용됩니다.

절차

1. 컨트롤 플레인 노드의 디버그 세션을 시작합니다.

```
$ oc debug node/<node_name>
```

2. 루트 디렉토리를 /host 로 변경합니다.

```
sh-4.2# chroot /host
```

3. 클러스터 전체의 프록시가 활성화되어 있는 경우 NO_PROXY, HTTP_PROXY 및 https_proxy 환경 변수를 내보내고 있는지 확인합니다.

4. cluster-backup.sh 스크립트를 실행하고 백업을 저장할 위치를 입력합니다.

작은 정보

cluster-backup.sh 스크립트는 etcd Cluster Operator의 구성 요소로 유지 관리되며 etcdctl snapshot save 명령 관련 래퍼입니다.

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

스크립트 출력 예

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-
6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-
controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-
pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg
":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-
25_190035.db.part"}
{"level":"info","ts":"2021-06-
25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot
stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg
":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-
```

```
25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed
snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg
":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114
MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg
":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

이 예제에서는 컨트롤 플레인 호스트의 **/home/core/assets/backup/** 디렉토리에 두 개의 파일이 생성됩니다.

- **snapshot_<datetimestamp>.db**:이 파일은 **etcd** 스냅샷입니다. **cluster-backup.sh** 스크립트는 유효성을 확인합니다.
- **static_kuberesources_<datetimestamp>.tar.gz**: 이 파일에는 정적 **pod** 리소스가 포함되어 있습니다. **etcd** 암호화가 활성화되어 있는 경우 **etcd** 스냅 샷의 암호화 키도 포함됩니다.

참고

etcd 암호화가 활성화되어 있는 경우 보안상의 이유로 이 두 번째 파일을 **etcd** 스냅 샷과 별도로 저장하는 것이 좋습니다. 그러나 이 파일은 **etcd** 스냅 샷에서 복원하는데 필요합니다.

etcd 암호화는 키가 아닌 값만 암호화합니다. 즉, 리소스 유형, 네임 스페이스 및 개체 이름은 암호화되지 않습니다.

4.10.5. etcd 데이터 조각 모음

대규모 및 밀도가 높은 클러스터의 경우 키 공간이 너무 커져서 공간 할당량을 초과하면 **etcd** 성능이 저하될 수 있습니다. 정기적으로 **etcd**를 유지 관리하고 조각 모음하여 데이터 저장소의 공간을 확보합니다. **etcd** 지표에 대한 **Prometheus**를 모니터링하고 필요한 경우 조각 모음을 모니터링하십시오. 그러지 않으면 **etcd**에서 키 읽기 및 삭제만 수락하는 유지 관리 모드로 클러스터를 배치하는 클러스터 전체 알람을 생성할 수 있습니다.

다음 주요 메트릭을 모니터링합니다.

- **etcd_server_quota_backend_bytes,** 현재 할당량 제한
- **etcd_mvcc_db_total_size_in_use_in_bytes.** 이는 기록 압축 후 실제 데이터베이스 사용량을 나타냅니다.
- **etcd_mvcc_db_total_size_in_bytes.gb**는 조각 모음 대기 중인 여유 공간을 포함하여 데이터베이스 크기를 표시합니다.

etcd 기록 압축과 같은 디스크 조각화를 초래하는 이벤트 후 디스크 공간을 회수하기 위해 **etcd** 데이터를 조각 모음합니다.

기록 압축은 **5**분마다 자동으로 수행되며 백엔드 데이터베이스에서 공백이 남습니다. 이 분할된 공간은 **etcd**에서 사용할 수 있지만 호스트 파일 시스템에서 사용할 수 없습니다. 호스트 파일 시스템에서 이 공간을 사용할 수 있도록 **etcd** 조각을 정리해야 합니다.

조각 모음이 자동으로 수행되지만 수동으로 트리거할 수도 있습니다.

참고

etcd Operator는 클러스터 정보를 사용하여 사용자에게 가장 효율적인 작업을 결정하기 때문에 자동 조각 모음은 대부분의 경우에 적합합니다.

4.10.5.1. 자동 조각 모음

etcd Operator는 디스크 조각 모음을 자동으로 수행합니다. 수동 조작이 필요하지 않습니다.

다음 로그 중 하나를 확인하여 조각 모음 프로세스가 성공했는지 확인합니다.

- **etcd** 로그

- cluster-etcd-operator Pod
- Operator 상태 오류 로그

주의

자동 조각 모음으로 인해 Kubernetes 컨트롤러 관리자와 같은 다양한 OpenShift 핵심 구성 요소에서 리더 선택 오류가 발생할 수 있으며 이로 인해 실패한 구성 요소가 다시 시작됩니다. 재시작은 무해하며 다음 실행 중인 인스턴스로 장애 조치를 트리거하거나 다시 시작한 후 구성 요소가 작업을 다시 시작합니다.

로그 출력 예

```
I0907 08:43:12.171919       1
defragcontroller.go:198] etcd member "ip-
10-0-191-150.example.redhat.com"
backend store fragmented: 39.33 %, dbSize:
349138944
```

4.10.5.2. 수동 조각 모음

etcd_db_total_size_in_bytes 메트릭을 모니터링하여 수동 조각 모음이 필요한지 여부를 결정할 수 있습니다.

PromQL 표현식을 사용하여 조각 모음을 사용하여 해제할 etcd 데이터베이스 크기를 MB 단위로 확인하여 조각 모음이 필요한지 여부를 확인할 수도 있습니다. (etcd_mvcc_db_total_size_in_in_bytes - etcd_mvcc_in_in_use_in_bytes)/1024/1024

주의

etcd를 분리하는 것은 차단 작업입니다. 조각화 처리가 완료될 때까지 **etcd** 멤버는 응답하지 않습니다. 따라서 각 **pod**의 조각 모음 작업 간에 클러스터가 정상 작동을 재개할 수 있도록 **1**분 이상 대기해야 합니다.

각 **etcd** 멤버의 **etcd** 데이터 조각 모음을 수행하려면 다음 절차를 따릅니다.

사전 요구 사항

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

절차

1. 리더가 최종 조각화 처리를 수행하므로 어떤 **etcd** 멤버가 리더인지 확인합니다.

   a. **etcd pod** 목록을 가져옵니다.

   ```
   $ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
   ```

   출력 예

   ```
   etcd-ip-10-0-159-225.example.redhat.com            3/3     Running     0          175m
   10.0.159.225   ip-10-0-159-225.example.redhat.com   <none>           <none>
   etcd-ip-10-0-191-37.example.redhat.com             3/3     Running     0          173m
   10.0.191.37    ip-10-0-191-37.example.redhat.com    <none>           <none>
   etcd-ip-10-0-199-170.example.redhat.com            3/3     Running     0          176m
   10.0.199.170   ip-10-0-199-170.example.redhat.com   <none>           <none>
   ```

   b. **Pod**를 선택하고 다음 명령을 실행하여 어떤 **etcd** 멤버가 리더인지 확인합니다.

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl
endpoint status --cluster -w table
```

출력 예

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd'
to see all of the containers in this pod.
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|        ENDPOINT           |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER
| RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.4.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |  104 MB |     false |      false
|         7 |      91624 |              91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  104 MB |      true |      false |
7 |      91624 |              91624 |        |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

이 출력의 **IS LEADER** 열에 따르면 **https://10.0.199.170:2379** 엔드 포인트가 리더입니다. 이전 단계의 출력과 이 앤드 포인트가 일치하면 리더의 **Pod** 이름은 **etcd-ip-10-0199-170.example.redhat.com**입니다.

2. **etcd** 멤버를 분리합니다.

   a. 실행중인 **etcd** 컨테이너에 연결하고 리더가 *아닌* **pod** 이름을 전달합니다.

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
   ```

   b. **ETCDCTL_ENDPOINTS** 환경 변수를 설정 해제합니다.

   ```
   sh-4.4# unset ETCDCTL_ENDPOINTS
   ```

   c. **etcd** 멤버를 분리합니다.

```
sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
```

출력 예

```
Finished defragmenting etcd member[https://localhost:2379]
```

시간 초과 오류가 발생하면 명령이 성공할 때까지 **--command-timeout** 의 값을 늘립니다.

d. 데이터베이스 크기가 감소되었는지 확인합니다.

```
sh-4.4# etcdctl endpoint status -w table --cluster
```

출력 예

```
+---------------------------+-----------------+---------+---------+-----------+-----------+-----------
+------------+-------------------+--------+
|        ENDPOINT           |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER
| RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+-----------------+---------+---------+-----------+-----------+-----------
+------------+-------------------+--------+
| https://10.0.191.37:2379 | 251cd44483d811c3 |   3.4.9 |  104 MB |     false |     false |
7 |     91624 |            91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |   41 MB |     false |     false |
7 |     91624 |            91624 |        | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  104 MB |      true |     false |
7 |     91624 |            91624 |        |
+---------------------------+-----------------+---------+---------+-----------+-----------+-----------
+------------+-------------------+--------+
```

이 예에서는 **etcd** 멤버의 데이터베이스 크기가 시작 크기인 **104MB**와 달리 현재 **41MB**임을 보여줍니다.

e. 다음 단계를 반복하여 다른 **etcd** 멤버에 연결하고 조각 모음을 수행합니다. 항상 리더

의 조각 모음을 마지막으로 수행합니다.

etcd pod가 복구될 수 있도록 조각 모음 작업에서 **1**분 이상 기다립니다. **etcd pod**가 복구될 때까지 **etcd** 멤버는 응답하지 않습니다.

3. 공간 할당량을 초과하여 **NOSPACE** 경고가 발생하는 경우 이를 지우십시오.

a. **NOSPACE** 경고가 있는지 확인합니다.

```
sh-4.4# etcdctl alarm list
```

출력 예

```
memberID:12345678912345678912 alarm:NOSPACE
```

b. 경고를 지웁니다.

```
sh-4.4# etcdctl alarm disarm
```

4.10.6. 이전 클러스터 상태로 복원

저장된 **etcd** 백업을 사용하여 이전 클러스터 상태를 복원하거나 컨트롤 플레인 호스트 대부분을 손실한 클러스터를 복원할 수 있습니다.

중요

클러스터를 복원할 때 동일한 **z-stream** 릴리스에서 가져온 **etcd** 백업을 사용해야 합니다. 예를 들어 **OpenShift Container Platform 4.7.2** 클러스터는 **4.7.2**에서 가져온 **etcd** 백업을 사용해야 합니다.

사전 요구 사항

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

- 복구 호스트로 사용할 정상적인 컨트롤 플레인 호스트가 있어야 합니다.

- 컨트롤 플레인 호스트에 대한 **SSH** 액세스.

- 동일한 백업에서 가져온 **etcd** 스냅샷과 정적 **pod** 리소스가 모두 포함된 백업 디렉토리입니다. 디렉토리의 파일 이름은 **snapshot_<datetimestamp>.db** 및 **static_kuberesources_<datetimestamp>.tar.gz** 형식이어야합니다.

중요

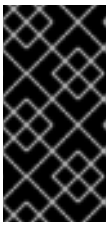
복구되지 않은 컨트롤 플레인 노드의 경우 **SSH** 연결을 설정하거나 정적 **Pod**를 중지할 필요가 없습니다. 복구되지 않은 다른 컨트롤 플레인 시스템을 하나씩 삭제하고 다시 생성할 수 있습니다.

프로세스

1. 복구 호스트로 사용할 컨트롤 플레인 호스트를 선택합니다. 이는 복구 작업을 실행할 호스트입니다.

2. 복구 호스트를 포함하여 각 컨트롤 플레인 노드에 **SSH** 연결을 설정합니다.

복구 프로세스가 시작된 후에는 **Kubernetes API** 서버에 액세스할 수 없으므로 컨트롤 플레인 노드에 액세스할 수 없습니다. 따라서 다른 터미널에서 각 컨트롤 플레인 호스트에 대한 **SSH** 연결을 설정하는 것이 좋습니다.



중요

이 단계를 완료하지 않으면 컨트롤 플레인 호스트에 액세스하여 복구 프로세스를 완료할 수 없으며 이 상태에서 클러스터를 복구할 수 없습니다.

3. **etcd** 백업 디렉토리를 복구 컨트롤 플레인 호스트에 복사합니다.

이 단계에서는 **etcd** 스냅샷 및 정적 **pod**의 리소스가 포함된 **backup** 디렉터리를 복구 컨트롤 플레인 호스트의 **/home/core/** 디렉터리에 복사하는 것을 전제로하고 있습니다.

4.

다른 컨트롤 플레인 노드에서 고정 **Pod**를 중지합니다.

참고

복구 호스트에서 **pod**를 수동으로 중지할 필요는 없습니다. 복구 스크립트는 복구 호스트에서 **pod**를 중지합니다.

a.

복구 호스트가 아닌 컨트롤 플레인 호스트에 액세스합니다.

b.

kubelet 매니페스트 디렉토리에서 기존 **etcd pod** 파일을 이동합니다.

```
$ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

c.

etcd pod가 중지되었는지 확인합니다.

```
$ sudo crictl ps | grep etcd | grep -v operator
```

이 명령의 출력은 비어 있어야합니다. 비어 있지 않은 경우 몇 분 기다렸다가 다시 확인하십시오.

d.

kubelet 매니페스트 디렉토리에서 기존 **Kubernetes API** 서버 **pod** 파일을 이동합니다.

```
$ sudo mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

e.

Kubernetes API 서버 **pod**가 중지되었는지 확인합니다.

```
$ sudo crictl ps | grep kube-apiserver | grep -v operator
```

이 명령의 출력은 비어 있어야합니다. 비어 있지 않은 경우 몇 분 기다렸다가 다시 확인하십시오.

f.

etcd 데이터 디렉토리를 다른 위치로 이동합니다.

```
$ sudo mv /var/lib/etcd/ /tmp
```

g. 복구 호스트가 아닌 다른 컨트롤 플레인 호스트에서 이 단계를 반복합니다.

5. 복구 컨트롤 플레인 호스트에 액세스합니다.

6. 클러스터 전체의 프록시가 활성화되어 있는 경우 **NO_PROXY, HTTP_PROXY** 및 **https_proxy** 환경 변수를 내보내고 있는지 확인합니다.

작은 정보

oc get proxy cluster -o yaml의 출력을 확인하여 프록시가 사용 가능한지 여부를 확인할 수 있습니다. **httpProxy, httpsProxy** 및 **noProxy** 필드에 값이 설정되어 있으면 프록시가 사용됩니다.

7. 복구 컨트롤 플레인 호스트에서 복원 스크립트를 실행하고 **etcd** 백업 디렉터리에 경로를 전달합니다.

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/backup
```

스크립트 출력 예

```
...stopping kube-scheduler-pod.yaml
...stopping kube-controller-manager-pod.yaml
...stopping etcd-pod.yaml
...stopping kube-apiserver-pod.yaml
Waiting for container etcd to stop
.complete
Waiting for container etcdctl to stop
............................complete
Waiting for container etcd-metrics to stop
complete
Waiting for container kube-controller-manager to stop
complete
Waiting for container kube-apiserver to stop
.....................................................................................complete
Waiting for container kube-scheduler to stop
complete
Moving etcd data-dir /var/lib/etcd/member to /var/lib/etcd-backup
starting restore-etcd static pod
starting kube-apiserver-pod.yaml
static-pod-resources/kube-apiserver-pod-7/kube-apiserver-pod.yaml
starting kube-controller-manager-pod.yaml
static-pod-resources/kube-controller-manager-pod-7/kube-controller-manager-
```

```
pod.yaml
starting kube-scheduler-pod.yaml
static-pod-resources/kube-scheduler-pod-8/kube-scheduler-pod.yaml
```

참고

복원 프로세스에서는 마지막 **etcd** 백업 후 노드 인증서가 업데이트된 경우 노드가 **NotReady** 상태가 될 수 있습니다.

8. 노드를 확인하여 **Ready** 상태인지 확인합니다.

a. 다음 명령을 실행합니다.

```
$ oc get nodes -w
```

샘플 출력

```
NAME              STATUS  ROLES        AGE    VERSION
host-172-25-75-28   Ready   master       3d20h   v1.23.3+e419edf
host-172-25-75-38   Ready   infra,worker   3d20h   v1.23.3+e419edf
host-172-25-75-40   Ready   master       3d20h   v1.23.3+e419edf
host-172-25-75-65   Ready   master       3d20h   v1.23.3+e419edf
host-172-25-75-74   Ready   infra,worker   3d20h   v1.23.3+e419edf
host-172-25-75-79   Ready   worker       3d20h   v1.23.3+e419edf
host-172-25-75-86   Ready   worker       3d20h   v1.23.3+e419edf
host-172-25-75-98   Ready   infra,worker   3d20h   v1.23.3+e419edf
```

모든 노드가 상태를 보고하는 데 몇 분이 걸릴 수 있습니다.

b. **NotReady** 상태에 있는 노드가 있는 경우 노드에 로그인하고 각 노드의 **/var/lib/kubelet/pki** 디렉터리에서 모든 **PEM** 파일을 제거합니다. 노드에 **SSH**로 액세스하거나 웹 콘솔의 터미널 창을 사용할 수 있습니다.

```
$  ssh -i <ssh-key-path> core@<master-hostname>
```

샘플 **pki** 디렉터리

```
sh-4.4# pwd
/var/lib/kubelet/pki
sh-4.4# ls
kubelet-client-2022-04-28-11-24-09.pem  kubelet-server-2022-04-28-11-24-15.pem
kubelet-client-current.pem              kubelet-server-current.pem
```

9. 모든 컨트롤 플레인 호스트에서 **kubelet** 서비스를 다시 시작합니다.

   a. 복구 호스트에서 다음 명령을 실행합니다.

   ```
   $ sudo systemctl restart kubelet.service
   ```

   b. 다른 모든 컨트롤 플레인 호스트에서 이 단계를 반복합니다.

10. 보류 중인 **CSR**을 승인합니다.

    a. 현재 **CSR**의 목록을 가져옵니다.

    ```
    $ oc get csr
    ```

    출력 예

    ```
    NAME        AGE    SIGNERNAME                                    REQUESTOR
    CONDITION
    csr-2s94x   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
    Pending 1
    csr-4bd6t   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
    Pending 2
    csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
    3
    csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
    ```

```
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
4
...
```

1 1 2

보류 중인 **kubelet** 서비스 **CSR**(사용자 프로비저닝 설치용)입니다.

3 4

보류 중인 **node-bootstrapper CSR**입니다.

b. **CSR**의 세부 사항을 검토하여 **CSR**이 유효한지 확인합니다.

```
$ oc describe csr <csr_name> 1
```

1

<csr_name>은 현재 **CSR** 목록에 있는 **CSR**의 이름입니다.

c. 각각의 유효한 **node-bootstrapper CSR**을 승인합니다.

```
$ oc adm certificate approve <csr_name>
```

d. 사용자 프로비저닝 설치의 경우 각 유효한 **kubelet** 서비스 **CSR**을 승인합니다.

```
$ oc adm certificate approve <csr_name>
```

11. 단일 멤버 컨트롤 플레인이 제대로 시작되었는지 확인합니다.

a. 복구 호스트에서 **etcd** 컨테이너가 실행 중인지 확인합니다.

```
$ sudo crictl ps | grep etcd | grep -v operator
```

출력 예

```
3ad41b7908e32
36f86e2eeaaffe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running             etcd                                  0
7c05f8af362f0
```

b.

복구 호스트에서 **etcd pod**가 실행 중인지 확인합니다.

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

참고

이 명령을 실행하기 전에 **oc login**을 실행하여 다음 오류가 발생하면 인증 컨트롤러가 시작될 때까지 잠시 기다렸다가 다시 시도하십시오.

```
Unable to connect to the server: EOF
```

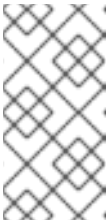
출력 예

```
NAME                                READY   STATUS     RESTARTS   AGE
etcd-ip-10-0-143-125.ec2.internal           1/1     Running    1          2m47s
```

Pending 상태에 있거나 출력에 여러 실행중인 **etcd pod**가 나열되어 있는 경우 몇 분 기다렸다가 다시 확인합니다.

c.

복구 호스트가 아닌 각 손실된 컨트롤 플레인 호스트에 대해 이 단계를 반복합니다.



참고

OVNKubernetes CNI(Container Network Interface) 플러그인을 사용하는 경우에만 다음 단계를 수행합니다.

12. 모든 호스트에서 **OVN(Open Virtual Network) Kubernetes Pod**를 재시작합니다.

a. **northbound** 데이터베이스(**nbdb**) 및 **southbound** 데이터베이스(**sbdb**)를 제거합니다. **SSH(Secure Shell)**를 사용하여 복구 호스트 및 나머지 컨트롤 플레인 노드에 액세스하고 다음 명령을 실행합니다.

```
$ sudo rm -f /var/lib/ovn/etc/*.db
```

b. 다음 명령을 실행하여 모든 **OVN-Kubernetes** 컨트롤 플레인 **Pod**를 삭제합니다.

```
$ oc delete pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

c. 모든 **OVN-Kubernetes** 컨트롤 플레인 **Pod**가 다시 배포되어 다음 명령을 실행하여 **Running** 상태인지 확인합니다.

```
$ oc get pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

출력 예

```
NAME                   READY   STATUS    RESTARTS   AGE
ovnkube-master-nb24h   4/4     Running   0          48s
ovnkube-master-rm8kw   4/4     Running   0          47s
ovnkube-master-zbqnh   4/4     Running   0          56s
```

d. 다음 명령을 실행하여 **ovnkube-node Pod**를 모두 삭제합니다.

```
$ oc get pods -n openshift-ovn-kubernetes -o name | grep ovnkube-node | while
read p ; do oc delete $p -n openshift-ovn-kubernetes ; done
```

e.

모든 **ovnkube-node Pod**가 다시 배포되어 다음 명령을 실행하여 **Running** 상태인지 확인합니다.

```
$ oc get  pods -n openshift-ovn-kubernetes | grep ovnkube-node
```

13. 복구되지 않는 다른 컨트롤 플레인 시스템을 삭제하고 하나씩 다시 생성합니다. 머신을 다시 생성한 후 새 버전이 강제되고 **etcd**가 자동으로 확장됩니다.

- 사용자가 프로비저닝한 베어 메탈 설치를 사용하는 경우 원래 사용했던 것과 동일한 방법을 사용하여 컨트롤 플레인 시스템을 다시 생성할 수 있습니다. 자세한 내용은 " 베어 메탈에 사용자 프로비저닝 클러스터 설치"를 참조하십시오.

> 주의
>
> 복구 호스트의 시스템을 삭제하고 다시 생성하지 마십시오.

- 설치 관리자 프로비저닝 인프라를 실행 중이거나 **Machine API**를 사용하여 머신을 생성한 경우 다음 단계를 따르십시오.

> 주의
>
> 복구 호스트의 시스템을 삭제하고 다시 생성하지 마십시오.
>
> 설치 관리자 프로비저닝 인프라에 베어 메탈 설치의 경우 컨트롤 플레인 머신이 다시 생성되지 않습니다. 자세한 내용은 " 베어 메탈 컨트롤 플레인 노드 교체"를 참조하십시오.

a. 손실된 컨트롤 플레인 호스트 중 하나에 대한 시스템을 가져옵니다.

cluster-admin 사용자로 클러스터에 액세스할 수 있는 터미널에서 다음 명령을 실

행합니다.

```
$ oc get machines -n openshift-machine-api -o wide
```

출력 예:

```
NAME                                 PHASE     TYPE       REGION    ZONE       AGE
NODE                           PROVIDERID                        STATE
clustername-8qw5l-master-0           Running   m4.xlarge  us-east-1  us-
east-1a  3h37m  ip-10-0-131-183.ec2.internal  aws:///us-east-1a/i-
0ec2782f8287dfb7e  stopped (1)
clustername-8qw5l-master-1           Running   m4.xlarge  us-east-1  us-
east-1b  3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-
096c349b700a19631  running
clustername-8qw5l-master-2           Running   m4.xlarge  us-east-1  us-
east-1c  3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-
02626f1dba9ed5bba  running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1
us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1
us-east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1
us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

(1) 이는 손실된 컨트롤 플레인 호스트 **ip-10-0-131-183.ec2.internal**의 컨트롤 플레인 시스템입니다.

b. 시스템 설정을 파일 시스템의 파일에 저장합니다.

```
$ oc get machine clustername-8qw5l-master-0 \ (1)
  -n openshift-machine-api \
  -o yaml \
  > new-master-machine.yaml
```

(1) 손실된 컨트롤 플레인 호스트의 컨트롤 플레인 시스템의 이름을 지정합니다.

c. 이전 단계에서 만든 **new-master-machine.yaml** 파일을 편집하여 새 이름을 할당

하고 불필요한 필드를 제거합니다.

i. 전체 **status** 섹션을 삭제합니다.

```
status:
  addresses:
  - address: 10.0.131.183
    type: InternalIP
  - address: ip-10-0-131-183.ec2.internal
    type: InternalDNS
  - address: ip-10-0-131-183.ec2.internal
    type: Hostname
  lastUpdated: "2020-04-20T17:44:29Z"
  nodeRef:
    kind: Node
    name: ip-10-0-131-183.ec2.internal
    uid: acca4411-af0d-4387-b73e-52b2484295ad
  phase: Running
  providerStatus:
    apiVersion: awsproviderconfig.openshift.io/v1beta1
    conditions:
    - lastProbeTime: "2020-04-20T16:53:50Z"
      lastTransitionTime: "2020-04-20T16:53:50Z"
      message: machine successfully created
      reason: MachineCreationSucceeded
      status: "True"
      type: MachineCreation
    instanceId: i-0fdb85790d76d0c3f
    instanceState: stopped
    kind: AWSMachineProviderStatus
```

ii. **metadata.name** 필드를 새 이름으로 변경합니다.

이전 시스템과 동일한 기본 이름을 유지하고 마지막 번호를 사용 가능한 다음 번호로 변경하는 것이 좋습니다. 이 예에서는 **clustername-8qw5l-master-0** 이 **clustername-8qw5l-master-3** 으로 변경되었습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  name: clustername-8qw5l-master-3
  ...
```

iii. **spec.providerID** 필드를 삭제합니다.

```
providerID: aws:///us-east-1a/i-0fdb85790d76d0c3f
```

iv. metadata.annotations 및 metadata.generation 필드를 제거합니다.

```
annotations:
  machine.openshift.io/instance-state: running
...
generation: 2
```

v. metadata.resourceVersion 및 metadata.uid 필드를 제거합니다.

```
resourceVersion: "13291"
uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

d. 손실된 컨트롤 플레인 호스트의 시스템을 삭제합니다.

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 1
```

1 손실된 컨트롤 플레인 호스트의 컨트롤 플레인 시스템의 이름을 지정합니다.

e. 시스템이 삭제되었는지 확인합니다.

```
$ oc get machines -n openshift-machine-api -o wide
```

출력 예:

```
NAME                                 PHASE     TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1           Running   m4.xlarge   us-east-1   us-
east-1b   3h37m   ip-10-0-143-125.ec2.internal   aws:///us-east-1b/i-
096c349b700a19631   running
clustername-8qw5l-master-2           Running   m4.xlarge   us-east-1   us-
east-1c   3h37m   ip-10-0-154-194.ec2.internal   aws:///us-east-1c/i-
02626f1dba9ed5bba   running
clustername-8qw5l-worker-us-east-1a-wbtgd   Running   m4.large   us-east-1
us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-
010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running   m4.large   us-east-1
us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-
0cb45ac45a166173b   running
```

```
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1
us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

f. new-master-machine.yaml 파일을 사용하여 새 시스템을 만듭니다.

```
$ oc apply -f new-master-machine.yaml
```

g. 새 시스템이 생성되었는지 확인합니다.

```
$ oc get machines -n openshift-machine-api -o wide
```

출력 예:

```
NAME                                 PHASE         TYPE       REGION     ZONE
AGE   NODE                         PROVIDERID                      STATE
clustername-8qw5l-master-1           Running       m4.xlarge  us-east-1  us-
east-1b  3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-
096c349b700a19631  running
clustername-8qw5l-master-2           Running       m4.xlarge  us-east-1  us-
east-1c  3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-
02626f1dba9ed5bba  running
clustername-8qw5l-master-3           Provisioning  m4.xlarge  us-east-1  us-
east-1a  85s    ip-10-0-173-171.ec2.internal  aws:///us-east-1a/i-
015b0888fe17bc2c8  running  (1)
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-
1  us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1
us-east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-
1  us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a  running
```

(1) 새 시스템 clustername-8qw5l-master-3 이 생성되며 단계가 Provisioning(프로비저닝)에서 Running (실행 중)으로 변경된 후 준비됩니다.

새 시스템을 만드는 데 몇 분이 소요될 수 있습니다. etcd 클러스터 Operator는 머신 또는 노드가 정상 상태로 돌아 오면 자동으로 동기화됩니다.

h. 복구 호스트가 아닌 각 손실된 컨트롤 플레인 호스트에 대해 다음 단계를 반복합니

다.

14. 별도의 터미널 창에서 다음 명령을 사용하여 **cluster-admin** 역할의 사용자로 클러스터에 로그인합니다.

```
$ oc login -u <cluster_admin> 1
```

1 **<cluster_admin>**은 **cluster-admin** 역할을 사용하여 사용자 이름을 지정합니다.

15. **etcd**를 강제로 재배포합니다.

클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$( date
--rfc-3339=ns )""}}' --type=merge 1
```

1 **forceRedeploymentReason** 값은 고유해야하므로 타임 스탬프가 추가됩니다.

etcd 클러스터 **Operator**가 재배포를 실행하면 기존 노드가 초기 부트 스트랩 확장과 유사한 새 **pod**를 사용하기 시작합니다.

16. 모든 노드가 최신 버전으로 업데이트되었는지 확인합니다.

클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc get etcd -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

etcd의 **NodeInstallerProgressing** 상태 조건을 확인하고 모든 노드가 최신 버전인지 확인합니다. 업데이트가 성공적으로 실행되면 출력에 **AllNodesAtLatestRevision**이 표시됩니다.

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1

이 예에서 최신 버전 번호는 7입니다.

출력에 2 nodes are at revision 6; 1 nodes are at revision 7와 같은 여러 버전 번호가 표시되면 이는 업데이트가 아직 진행 중임을 의미합니다. 몇 분 기다린 후 다시 시도합니다.

17. etcd를 재배포한 후 컨트롤 플레인에 새 롤아웃을 강제 실행합니다. kubelet이 내부 로드 밸런서를 사용하여 API 서버에 연결되어 있으므로 Kubernetes API 서버는 다른 노드에 다시 설치됩니다.

cluster-admin 사용자로 클러스터에 액세스할 수있는 터미널에서 다음 명령을 실행합니다.

a. Kubernetes API 서버에 대해 새 롤아웃을 강제 적용합니다.

```
$ oc patch kubeapiserver cluster -p='{"spec": {"forceRedeploymentReason":
"recovery-"'$( date --rfc-3339=ns )'"}}' --type=merge
```

모든 노드가 최신 버전으로 업데이트되었는지 확인합니다.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

NodeInstallerProgressing 상태 조건을 확인하고 모든 노드가 최신 버전인지 확인합니다. 업데이트가 성공적으로 실행되면 출력에 AllNodesAtLatestRevision이 표시됩니다.

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1

이 예에서 최신 버전 번호는 7입니다.

출력에 2 nodes are at revision 6; 1 nodes are at revision 7와 같은 여러 버전 번호가 표시되면 이는 업데이트가 아직 진행 중임을 의미합니다. 몇 분 기다린 후 다시 시도합니다.

b. Kubernetes 컨트롤러 관리자에 대해 새 롤아웃을 강제 적용합니다.

```
$ oc patch kubecontrollermanager cluster -p='{"spec":
{"forceRedeploymentReason": "recovery-"'$( date --rfc-3339=ns )"""}}' --
type=merge
```

모든 노드가 최신 버전으로 업데이트되었는지 확인합니다.

```
$ oc get kubecontrollermanager -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

NodeInstallerProgressing 상태 조건을 확인하고 모든 노드가 최신 버전인지 확인합니다. 업데이트가 성공적으로 실행되면 출력에 **AllNodesAtLatestRevision**이 표시됩니다.

```
AllNodesAtLatestRevision
3 nodes are at revision 7 (1)
```

(1) 이 예에서 최신 버전 번호는 **7**입니다.

출력에 **2 nodes are at revision 6; 1 nodes are at revision 7**와 같은 여러 버전 번호가 표시되면 이는 업데이트가 아직 진행 중임을 의미합니다. 몇 분 기다린 후 다시 시도합니다.

c. **Kubernetes** 스케줄러에 대해 새 롤아웃을 강제 적용합니다.

```
$ oc patch kubescheduler cluster -p='{"spec": {"forceRedeploymentReason":
"recovery-"'$( date --rfc-3339=ns )"""}}' --type=merge
```

모든 노드가 최신 버전으로 업데이트되었는지 확인합니다.

```
$ oc get kubescheduler -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

NodeInstallerProgressing 상태 조건을 확인하고 모든 노드가 최신 버전인지 확인합니다. 업데이트가 성공적으로 실행되면 출력에 **AllNodesAtLatestRevision**이 표시됩니다.

```
AllNodesAtLatestRevision
3 nodes are at revision 7 (1)
```

(1)

이 예에서 최신 버전 번호는 7입니다.

출력에 2 nodes are at revision 6; 1 nodes are at revision 7와 같은 여러 버전 번호가 표시되면 이는 업데이트가 아직 진행 중임을 의미합니다. 몇 분 기다린 후 다시 시도합니다.

18. 모든 컨트롤 플레인 호스트가 클러스터를 시작하여 참여하고 있는지 확인합니다.

클러스터에 액세스할 수 있는 터미널에서 cluster-admin 사용자로 다음 명령을 실행합니다.

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

출력 예

```
etcd-ip-10-0-143-125.ec2.internal        2/2     Running     0     9h
etcd-ip-10-0-154-194.ec2.internal        2/2     Running     0     9h
etcd-ip-10-0-173-171.ec2.internal        2/2     Running     0     9h
```

복구 절차 후 모든 워크로드가 정상 작업으로 돌아가도록 하려면 Kubernetes API 정보를 저장하는 각 Pod를 다시 시작합니다. 여기에는 라우터, Operator 및 타사 구성 요소와 같은 OpenShift Container Platform 구성 요소가 포함됩니다.

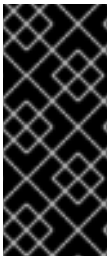
이 프로세스를 완료한 후 모든 서비스를 복구하는데 몇 분 정도 걸릴 수 있습니다. 예를 들어, OAuth 서버 pod가 다시 시작될 때까지 oc login을 사용한 인증이 즉시 작동하지 않을 수 있습니다.

추가 리소스

- 베어 메탈에 사용자 프로비저닝 클러스터 설치
- 베어 메탈 컨트롤 플레인 노드 교체

4.10.7. 영구 스토리지 상태 복원을 위한 문제 및 해결 방법

OpenShift Container Platform 클러스터에서 모든 형식의 영구저장장치를 사용하는 경우 일반적으로 클러스터의 상태가 etcd 외부에 저장됩니다. StatefulSet 오브젝트에서 실행 중인 Pod 또는 데이터베이스에서 실행 중인 Elasticsearch 클러스터일 수 있습니다. etcd 백업에서 복원하면 OpenShift Container Platform의 워크로드 상태도 복원됩니다. 그러나 etcd 스냅샷이 오래된 경우 상태가 유효하지 않거나 오래되었을 수 있습니다.



중요

PV(영구 볼륨)의 내용은 etcd 스냅샷의 일부가 아닙니다. etcd 스냅샷에서 OpenShift Container Platform 클러스터를 복원할 때 중요하지 않은 워크로드가 중요한 데이터에 액세스할 수 있으며 그 반대의 경우로도 할 수 있습니다.

다음은 사용되지 않는 상태를 생성하는 몇 가지 예제 시나리오입니다.

- MySQL 데이터베이스는 PV 오브젝트에서 지원하는 pod에서 실행됩니다. etcd 스냅샷에서 OpenShift Container Platform을 복원해도 스토리지 공급자의 볼륨을 다시 가져오지 않으며 pod를 반복적으로 시작하려고 하지만 실행 중인 MySQL pod는 생성되지 않습니다. 스토리지 공급자에서 볼륨을 복원한 다음 새 볼륨을 가리키도록 PV를 편집하여 이 Pod를 수동으로 복원해야 합니다.

- Pod P1에서는 노드 X에 연결된 볼륨 A를 사용합니다. 다른 pod가 노드 Y에서 동일한 볼륨을 사용하는 동안 etcd 스냅샷을 가져오는 경우 etcd 복원이 수행되면 해당 볼륨이 여전히 Y 노드에 연결되어 있으므로 Pod P1이 제대로 시작되지 않을 수 있습니다. OpenShift Container Platform은 연결을 인식하지 못하고 자동으로 연결을 분리하지 않습니다. 이 경우 볼륨이 노드 X에 연결된 다음 Pod P1이 시작될 수 있도록 노드 Y에서 볼륨을 수동으로 분리해야 합니다.

- etcd 스냅샷을 만든 후 클라우드 공급자 또는 스토리지 공급자 인증 정보가 업데이트되었습니다. 이로 인해 해당 인증 정보를 사용하는 CSI 드라이버 또는 Operator가 작동하지 않습니다. 해당 드라이버 또는 Operator에 필요한 인증 정보를 수동으로 업데이트해야 할 수 있습니다.

- etcd 스냅샷을 만든 후 OpenShift Container Platform 노드에서 장치가 제거되거나 이름이 변경됩니다. Local Storage Operator는 /dev/disk/by-id 또는 /dev 디렉터리에서 관리하는 각 PV에 대한 심볼릭 링크를 생성합니다. 이 경우 로컬 PV가 더 이상 존재하지 않는 장치를 참조할 수 있습니다.

  이 문제를 해결하려면 관리자가 다음을 수행해야 합니다.

  1. 잘못된 장치가 있는 PV를 수동으로 제거합니다.

2. 각 노드에서 심볼릭 링크를 제거합니다.

3. **LocalVolume** 또는 **LocalVolumeSet** 오브젝트를 삭제합니다 (*스토리지 → 영구 스토리지 구성 → 로컬 볼륨을 사용하는 영구 스토리지 → **Local Storage Operator** 리소스 삭제*참조).

4.11. POD 중단 예산

Pod 중단 예산을 이해하고 구성합니다.

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

***Pod** 중단 예산*은 Kubernetes **API**의 일부이며 다른 오브젝트 유형과 같은 **oc** 명령으로 관리할 수 있습니다. 유지 관리를 위해 노드를 드레이닝하는 것과 같이 작업 중에 **pod** 에 대한 보안 제약 조건을 지정할 수 있습니다.

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

PodDisruptionBudget 오브젝트의 구성은 다음과 같은 주요 부분으로 구성되어 있습니다.

- 일련의 **pod**에 대한 라벨 쿼리 기능인 라벨 선택기입니다.
- 동시에 사용할 수 있어야 하는 최소 **pod** 수를 지정하는 가용성 수준입니다.
  - **minAvailable**은 중단 중에도 항상 사용할 수 있어야하는 **pod** 수입니다.
  - **maxUnavailable**은 중단 중에 사용할 수없는 **pod** 수입니다.

참고

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

다음을 사용하여 모든 프로젝트에서 pod 중단 예산을 확인할 수 있습니다.

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

출력 예

```
NAMESPACE         NAME          MIN-AVAILABLE   SELECTOR
another-project   another-pdb   4               bar=foo
test-project      my-pdb        2               foo=bar
```

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.

4.11.2. 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:
   foo: bar
```

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 matchExpressions의 결과는 논리적으로 결합됩니다. 이 매개 변수(예: selector {})를 비워 두면 프로젝트의 모든 Pod를 선택합니다.

또는 다음을 수행합니다.

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

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

2. 다음 명령을 실행하여 오브젝트를 프로젝트에 추가합니다.

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

4.12. 클라우드 공급자 인증 정보 교체 또는 제거

OpenShift Container Platform을 설치한 후 일부 조직에서는 초기 설치 중에 사용된 클라우드 공급자 인증 정보를 교체하거나 제거해야 합니다.

클러스터가 새 인증 정보를 사용할 수 있도록 하려면 CCO(Cloud Credential Operator) 가 클라우드 공급자 인증 정보를 관리하는 데 사용하는 시크릿을 업데이트해야 합니다.

4.12.1. 클라우드 공급자 인증 정보를 수동으로 교체

어떠한 이유로 클라우드 공급자 인증 정보가 변경되면 CCO(Cloud Credential Operator)에서 클라우드 공급자 인증 정보를 관리하기 위해 사용하는 시크릿을 수동으로 업데이트해야 합니다.

클라우드 인증 정보를 교체하는 프로세스는 CCO가 사용하도록 구성된 모드에 따라 달라집니다. Mint 모드를 사용하는 클러스터의 인증 정보를 교체한 후 삭제된 인증 정보를 통해 생성된 구성 요소 인증 정보를 수동으로 제거해야 합니다.

사전 요구 사항

- 클러스터는 다음을 사용하는 CCO 모드로 클라우드 인증 정보 교체를 수동으로 지원하는 플랫폼에 설치됩니다.
  - Mint 모드의 경우 AWS(Amazon Web Services) 및 GCP(Google Cloud Platform)가 지원됩니다.
  - Passthrough 모드의 경우 AWS(Amazon Web Services), Microsoft Azure, GCP(Google Cloud Platform), RHOSP(Red Hat OpenStack Platform), RHV(Red Hat Virtualization) 및 VMware vSphere가 지원됩니다.
- 클라우드 공급자와 인터페이스에 사용되는 인증 정보를 변경했습니다.

- 새 인증 정보에는 클러스터에서 사용할 수 있도록 구성된 모드 **CCO**에 대한 충분한 권한이 있습니다.

절차

1. 웹 콘솔의 **Administrator** 모드에서 **Workloads** → **Secrets**로 이동합니다.

2. **Secrets** 페이지의 표에서 클라우드 공급자의 루트 시크릿을 찾습니다.

플랫폼	시크릿 이름
AWS	**aws-creds**
Azure	**azure-credentials**
GCP	**gcp-credentials**
RHOSP	**openstack-credentials**
RHV	**ovirt-credentials**
VMware vSphere	**vsphere-creds**

3. 시크릿과 동일한 행에서 옵션 메뉴

를 클릭하고 시크릿 편집을 선택합니다.

4. **Value** 필드의 내용을 기록합니다. 이 정보를 사용하여 인증서를 업데이트한 후 값이 다른지 확인할 수 있습니다.

5. 클라우드 공급자에 대한 새로운 인증 정보를 사용하여 **Value** 필드의 텍스트를 업데이트한 다음 저장을 클릭합니다.

6. **vSphere CSI Driver Operator**가 활성화되어 있지 않은 **vSphere** 클러스터의 인증 정보를 업데이트하는 경우 **Kubernetes** 컨트롤러 관리자의 롤아웃을 강제 적용하여 업데이트된 인증 정보를 적용해야 합니다.

참고

vSphere CSI Driver Operator가 활성화된 경우 이 단계가 필요하지 않습니다.

업데이트된 vSphere 인증 정보를 적용하려면 OpenShift Container Platform CLI에 cluster-admin 역할의 사용자로 로그인하고 다음 명령을 실행합니다.

```
$ oc patch kubecontrollermanager cluster \
  -p='{"spec": {"forceRedeploymentReason": "recovery-"'$( date )'""}}' \
  --type=merge
```

인증 정보가 출시되는 동안 Kubernetes Controller Manager Operator의 상태는 Progressing=true 로 보고합니다. 상태를 보려면 다음 명령을 실행합니다.

```
$ oc get co kube-controller-manager
```

7. 클러스터의 CCO가 Mint 모드를 사용하도록 구성된 경우 개별 CredentialsRequest 오브젝트에서 참조하는 각 구성 요소 시크릿을 삭제합니다.

   a. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다.

   b. 참조되는 모든 구성 요소 시크릿의 이름과 네임스페이스를 가져옵니다.

```
$ oc -n openshift-cloud-credential-operator get CredentialsRequest \
  -o json | jq -r '.items[] | select (.spec.providerSpec.kind=="<provider_spec>") |
.spec.secretRef'
```

여기서 <provider_spec >은 클라우드 공급자의 해당 값입니다.

- AWS: AWSProviderSpec
- GCP: GCPProviderSpec

AWS의 부분 예제 출력

```
{
  "name": "ebs-cloud-credentials",
  "namespace": "openshift-cluster-csi-drivers"
}
{
  "name": "cloud-credential-operator-iam-ro-creds",
  "namespace": "openshift-cloud-credential-operator"
}
```

c. 참조된 각 구성 요소 시크릿을 삭제합니다.

```
$ oc delete secret <secret_name> \ 1
  -n <secret_namespace> 2
```

1 보안 이름을 지정합니다.

2 보안이 포함된 네임스페이스를 지정합니다.

AWS 시크릿 삭제 예

```
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

공급자 콘솔에서 인증 정보를 수동으로 삭제할 필요가 없습니다. 참조된 구성 요소 시크릿을 삭제하면 **CCO**가 플랫폼에서 기존 인증 정보를 삭제하고 새 인증서를 생성합니다.

검증

인증 정보가 변경되었는지 확인하려면 다음을 수행하십시오.

1. 웹 콘솔의 **Administrator** 모드에서 **Workloads** → **Secrets**로 이동합니다.

2. **Value** 필드의 내용이 변경되었는지 확인합니다.

추가 리소스

- vSphere CSI Driver Operator

4.12.2. 클라우드 공급자 인증 정보 제거

Mint 모드에서 **CCO(Cloud Credential Operator)**를 사용하여 **OpenShift Container Platform** 클러스터를 설치한 후 클러스터의 **kube-system** 네임스페이스에서 관리자 수준 인증 정보 시크릿을 제거할 수 있습니다. 관리자 수준 인증 정보는 업그레이드와 같은 승격된 권한이 필요한 변경 시에만 필요합니다.

참고

z-stream 외 업그레이드 이전에는 관리자 수준 인증 정보를 사용하여 인증 정보 시크릿을 복원해야 합니다. 인증 정보가 없으면 업그레이드가 차단될 수 있습니다.

사전 요구 사항

- 클러스터는 **CCO**에서 클라우드 인증 정보 제거를 지원하는 플랫폼에 설치되어 있습니다. 지원되는 플랫폼은 **AWS** 및 **GCP**입니다.

절차

1. 웹 콘솔의 **Administrator** 모드에서 **Workloads** → **Secrets**로 이동합니다.

2. **Secrets** 페이지의 표에서 클라우드 공급자의 루트 시크릿을 찾습니다.

플랫폼	시크릿 이름
AWS	**aws-creds**
GCP	**gcp-credentials**

3.

시크릿과 동일한 행에서 옵션 메뉴

를 클릭하고 시크릿 삭제를 선택합니다.

추가 리소스

- Cloud Credential Operator 정보
- Amazon Web Services (AWS) 시크릿 형식
- Microsoft Azure 시크릿 형식
- Google Cloud Platform (GCP) 시크릿 형식

4.13. 연결이 끊긴 클러스터의 이미지 스트림 구성

연결이 끊긴 환경에 OpenShift Container Platform을 설치한 후 Cluster Samples Operator 및 must-gather 이미지 스트림에 대한 이미지 스트림을 구성합니다.

4.13.1. 미러링을 위한 Cluster Samples Operator 지원

설치 프로세스 중에 OpenShift Container Platform은 openshift-cluster-samples-operator 네임스페이스에 imagestreamtag-to-image라는 구성 맵을 생성합니다. imagestreamtag-to-image 구성 맵에는 각 이미지 스트림 태그에 대한 이미지 채우기 항목이 포함되어 있습니다.

구성 맵의 데이터 필드에 있는 각 항목의 키 형식은 <image_stream_name>_<image_stream_tag_name>입니다.

OpenShift Container Platform의 연결이 끊긴 설치 프로세스 중에 Cluster Samples Operator의 상태가 Removed로 설정됩니다. Managed로 변경하려면 샘플이 설치됩니다.

참고

네트워크 제한 또는 중단된 환경에서 샘플을 사용하려면 네트워크 외부의 서비스에 액세스해야 할 수 있습니다. 일부 예제 서비스에는 **GitHub, Maven Central, npm, RubyGems, PyPi** 등이 있습니다. 클러스터 샘플 **Operator**의 오브젝트가 필요한 서비스에 도달할 수 있도록 하는 추가 단계가 있을 수 있습니다.

이 구성 맵을 사용하여 이미지 스트림을 가져오려면 이미지를 미러링해야 하는 이미지 참조로 사용할 수 있습니다.

- **Cluster Samples Operator**가 **Removed**로 설정된 경우 미러링된 레지스트리를 생성하거나 사용할 기존 미러링된 레지스트리를 확인할 수 있습니다.
- 새 구성 맵을 가이드로 사용하여 미러링된 레지스트리에 샘플을 미러링합니다.
- **Cluster Samples Operator** 구성 개체의 **skippedImagestreams** 필드에 미러링되지 않은 이미지 스트림을 추가합니다.
- **Cluster Samples Operator** 구성 개체의 **samplesRegistry** 를 미러링된 레지스트리로 설정합니다.
- 그런 다음 **Cluster Samples Operator**를 **Managed**로 설정하여 미러링된 이미지 스트림을 설치합니다.

4.13.2. 대체 레지스트리 또는 미러링된 레지스트리에서 Cluster Samples Operator 이미지 스트림 사용

Cluster Samples Operator에 의해 관리되는 **openshift** 네임스페이스에 있는 대부분의 이미지 스트림은 **registry.redhat.io**의 **Red Hat** 레지스트리에 있는 이미지를 참조합니다. 미러링은 이러한 이미지 스트림에 적용되지 않습니다.

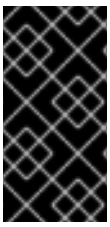
중요

jenkins, jenkins-agent-maven 및 **jenkins-agent-nodejs** 이미지스트림은 설치 페이로드에서 가져오고 **Samples Operator**로 관리되므로 해당 이미지 스트림에 대한 추가 미러링 절차가 필요하지 않습니다.

Sample Operator 설정 파일에서 **samplesRegistry** 필드를 **registry.redhat.io**로 설정하는 것은 **Jenkins** 이미지 및 이미지 스트림을 제외하고 **registry.redhat.io**로 전송되기 때문에 중복될 수 있습니다.

참고

설치 페이로드의 일부인 **cli, installer, must-gather** 및 **test** 이미지 스트림은 **Cluster Samples Operator**가 관리하지 않습니다. 이러한 내용은 이 절차에서 다루지 않습니다.



중요

연결이 끊긴 환경에서 **Cluster Samples Operator**를 **Managed** 로 설정해야 합니다. 이미지 스트림을 설치하려면 미러링된 레지스트리가 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 미러 레지스트리의 풀 시크릿을 생성합니다.

프로세스

1. 미러링할 특정 이미지 스트림의 이미지에 액세스합니다. 예를 들면 다음과 같습니다.

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep
registry.redhat.io
```

2. 제한된 네트워크 환경에서 필요한 모든 이미지 스트림과 관련된 **registry.redhat.io**의 이미지를 지정된 미러 중 하나로 미러링합니다.

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest
${MIRROR_ADDR}/rhscl/ruby-25-rhel7:latest
```

3. 클러스터의 이미지 구성 오브젝트를 생성합니다.

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. 클러스터의 이미지 설정 오브젝트에서 미러에 필요한 신뢰할 수 있는 CA를 추가합니다.

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

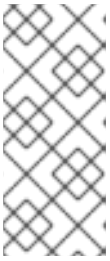
5. 미러 설정에 정의된 미러 위치의 hostname 부분을 포함하도록 Cluster Samples Operator 설정 오브젝트에서 samplesRegistry 필드를 업데이트합니다.

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

참고

현재 이미지 스트림 가져오기 프로세스에서 미러 또는 검색 메커니즘이 사용되지 않기 때문에 이 작업이 필요합니다.

6. Cluster Samples Operator 구성 오브젝트의 skippedImagestreams 필드에 미러링되지 않은 이미지 스트림을 추가합니다. 또는 샘플 이미지 스트림을 모두 지원할 필요가 없는 경우 Cluster Samples Operator 구성 오브젝트에서 Cluster Samples Operator를 Removed 로 설정합니다.



참고

이미지 스트림 가져오기가 실패했으나 Cluster Samples Operator가 주기적으로 재시도하거나 재시도하지 않는 것처럼 보이면 Cluster Samples Operator는 경고를 발행합니다.

openshift 네임스페이스의 여러 템플릿은 이미지 스트림을 참조합니다. 따라서 Removed를 사용하여 이미지 스트림과 템플릿을 모두 제거하면 누락된 이미지 스트림으로 인해 기능이 제대로 작동하지 않을 경우 템플릿을 사용할 가능성이 없어집니다.

4.13.3. 지원 데이터 수집을 위해 클러스터 준비

제한된 네트워크를 사용하는 클러스터는 Red Hat 지원을 위한 디버깅 데이터를 수집하기 위해 기본

must-gather 이미지를 가져와야합니다. must-gather 이미지는 기본적으로 가져 오지 않으며 제한된 네트워크의 클러스터는 원격 저장소에서 최신 이미지를 가져 오기 위해 인터넷에 액세스할 수 없습니다.

절차

1. 미러 레지스트리의 신뢰할 수 있는 CA를 Cluster Samples Operator 설정의 일부로 클러스터의 이미지 구성 오브젝트에 추가하지 않은 경우 다음 단계를 수행합니다.

   a. 클러스터의 이미지 구성 오브젝트를 생성합니다.

   ```
   $ oc create configmap registry-config --from-
   file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
   ```

   b. 클러스터의 이미지 설정 오브젝트에서 미러에 필요한 신뢰할 수 있는 CA를 추가합니다.

   ```
   $ oc patch image.config.openshift.io/cluster --patch '{"spec":
   {"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
   ```

2. 설치 페이로드에서 기본 must-gather 이미지를 가져옵니다.

   ```
   $ oc import-image is/must-gather -n openshift
   ```

oc adm must-gather 명령을 실행하는 경우 다음 예와 같이 --image 플래그를 사용하고 페이로드 이미지를 가리키십시오.

```
$ oc adm must-gather --image=$(oc adm release info --image-for must-gather)
```

4.14. CLUSTER SAMPLE OPERATOR 이미지 스트림 태그의 주기적인 가져오기 구성

새 버전이 사용 가능하게 되면 주기적으로 이미지 스트림 태그를 가져와서 Cluster Sample Operator 이미지의 최신 버전에 항상 액세스할 수 있는지 확인할 수 있습니다.

절차

1. 다음 명령을 실행하여 openshift 네임스페이스의 모든 이미지 스트림을 가져옵니다.

   ```
   oc get imagestreams -nopenshift
   ```

2. 다음 명령을 실행하여 **openshift** 네임스페이스의 모든 이미지 스트림에 대한 태그를 가져옵니다.

```
$ oc get is <image-stream-name> -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}
{.from.name}{'\n'}{end}" -nopenshift
```

예를 들면 다음과 같습니다.

```
$ oc get is ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}
{'\n'}{end}" -nopenshift
```

출력 예

```
1.11 registry.access.redhat.com/ubi8/openjdk-17:1.11
1.12 registry.access.redhat.com/ubi8/openjdk-17:1.12
```

3. 다음 명령을 실행하여 이미지 스트림에 있는 각 태그에 대한 이미지 가져오기를 주기적으로 예약합니다.

```
$ oc tag <repository/image> <image-stream-name:tag> --scheduled -nopenshift
```

예를 들면 다음과 같습니다.

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.11 ubi8-openjdk-17:1.11 --
scheduled -nopenshift
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.12 ubi8-openjdk-17:1.12 --
scheduled -nopenshift
```

이 명령은 **OpenShift Container Platform**이 특정 이미지 스트림 태그를 주기적으로 업데이트하도록 합니다. 이 기간은 기본적으로 **15**분으로 설정되는 클러스터 전체 설정입니다.

4. 다음 명령을 실행하여 주기적인 가져오기의 스케줄링 상태를 확인합니다.

```
oc get imagestream <image-stream-name> -o jsonpath="{range .spec.tags[*]}Tag:
{.name}{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

예를 들면 다음과 같습니다.

```
oc get imagestream ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}Tag: {.name}
{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

출력 예

```
Tag: 1.11 Scheduled: true
Tag: 1.12 Scheduled: true
```

5장. 설치 후 노드 작업

OpenShift Container Platform을 설치한 후 특정 노드 작업을 통해 요구 사항에 맞게 클러스터를 추가로 확장하고 사용자 지정할 수 있습니다.

5.1. OPENSHIFT CONTAINER PLATFORM 클러스터에 RHEL 컴퓨팅 머신 추가

RHEL 컴퓨팅 노드를 이해하고 사용합니다.

5.1.1. 클러스터에 RHEL 컴퓨팅 노드 추가 정보

OpenShift Container Platform 4.9에서는 사용자 프로비저닝 인프라 설치를 사용하는 경우 클러스터에서 RHEL (Red Hat Enterprise Linux) 머신을 컴퓨팅 머신 (작업자 머신이라고도 함)으로 사용하는 옵션이 있습니다. 클러스터의 컨트롤 플레인 또는 마스터 시스템에 RHCOS (Red Hat Enterprise Linux CoreOS) 머신을 사용해야합니다.

사용자 프로비저닝 인프라를 사용하는 모든 설치와 마찬가지로 클러스터에서 RHEL 컴퓨팅 머신을 사용하기로 선택한 경우 시스템 업데이트 수행, 패치 적용 및 기타 필요한 모든 작업 실행을 포함한 모든 운영 체제의 라이프 사이클 관리 및 유지 관리에 대한 책임이 있습니다.

중요

클러스터의 시스템에서 OpenShift Container Platform을 제거하려면 운영 체제를 제거해야하므로 클러스터에 추가한 모든 RHEL 머신에 전용 하드웨어를 사용해야합니다.

중요

OpenShift Container Platform 클러스터에 추가한 모든 RHEL 머신에서 스왑 메모리가 비활성화됩니다. 이 머신에서 스왑 메모리를 활성화할 수 없습니다.

컨트롤 플레인을 초기화한 후 RHEL 컴퓨팅 머신을 클러스터에 추가해야합니다.

5.1.2. RHEL 컴퓨팅 노드의 시스템 요구 사항

OpenShift Container Platform 환경에서 RHEL (Red Hat Enterprise Linux) 컴퓨팅 또는 작업자 머신 호스트는 다음과 같은 최소 하드웨어 사양 및 시스템 수준 요구 사항을 충족해야합니다.

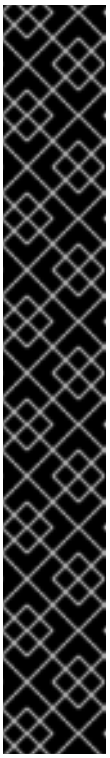
- Red Hat 계정에 유효한 OpenShift Container Platform 서브스크립션이 있어야합니다. 서브스크립션이 없는 경우 영업 담당자에게 자세한 내용을 문의하십시오.

- 프로덕션 환경에서 예상 워크로드를 지원할 수 있는 컴퓨팅 머신을 제공해야합니다. 클러스터 관리자는 예상 워크로드를 계산하고 오버 헤드에 약 10%를 추가해야합니다. 프로덕션 환경의 경우 노드 호스트 장애가 최대 용량에 영향을 미치지 않도록 충분한 리소스를 할당해야 합니다.

- 각 시스템은 다음 하드웨어 요구 사항을 충족해야합니다.

  - 물리적 또는 가상 시스템 또는 퍼블릭 또는 프라이빗 IaaS에서 실행되는 인스턴스.

  - 기본 OS: RHEL 7.9 또는 RHEL 7.9~8.7 및 "최소"설치 옵션.

    중요

    OpenShift Container Platform 클러스터에 RHEL 7 컴퓨팅 머신을 추가하는 것은 더 이상 사용되지 않습니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

    또한 RHEL 7 컴퓨팅 머신을 RHEL 8로 업그레이드할 수 없습니다. 새 RHEL 8 호스트를 배포해야 하며 이전 RHEL 7 호스트를 제거해야 합니다. 자세한 내용은 "노드 삭제" 섹션을 참조하십시오.

    OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

  - FIPS 모드에서 OpenShift Container Platform을 배포하는 경우 부팅하기 전에 RHEL 시스템에서 FIPS를 활성화해야합니다. RHEL 7 설명서에서 FIPS 모드 활성화를 참조하십시오.

중요

진행 중인 **FIPS** 검증 / 모듈 암호화 라이브러리 사용은 **x86_64** 아키텍처의 **OpenShift Container Platform** 배포에서만 지원됩니다.

- **NetworkManager 1.0** 이상
- **vCPU 1**개
- 최소 **8GB RAM**
- **/var/**를 포함하는 파일 시스템의 최소 **15GB** 하드 디스크 공간
- **/usr/local/bin/**을 포함하는 파일 시스템의 최소 **1GB** 하드 디스크 공간
- 임시 디렉토리를 포함하는 파일 시스템의 최소 **1GB**의 하드 디스크 공간 임시 시스템 디렉토리는 **Python** 표준 라이브러리의 **tempfile** 모듈에 정의된 규칙에 따라 결정됩니다.
  - 각 시스템은 시스템 제공 업체의 추가 요구 사항을 충족해야합니다. 예를 들어 **VMware vSphere**에 클러스터를 설치하는 경우 스토리지 지침에 따라 디스크를 구성하고 **disk.enableUUID=true** 속성을 설정해야합니다.
  - 각 시스템은 **DNS** 확인 가능한 호스트 이름을 사용하여 클러스터의 **API** 끝점에 액세스할 수 있어야 합니다. 모든 네트워크 보안 액세스 제어는 클러스터의 **API** 서비스 엔드 포인트에 대한 시스템 액세스를 허용해야합니다.

추가 리소스

- 노드 삭제

5.1.2.1. 인증서 서명 요청 관리

사용자가 프로비저닝하는 인프라를 사용하는 경우 자동 시스템 관리 기능으로 인해 클러스터의 액세스가 제한되므로 설치한 후 클러스터 인증서 서명 요청(**CSR**)을 승인하는 메커니즘을 제공해야 합니다. **kube-controller-manager**는 **kubelet** 클라이언트 **CSR**만 승인합니다. **machine-approver**는 올바른 시스

템에서 발행한 요청인지 확인할 수 없기 때문에 **kubelet** 자격 증명을 사용하여 요청하는 서비스 인증서의 유효성을 보장할 수 없습니다. **kubelet** 서빙 인증서 요청의 유효성을 확인하고 요청을 승인하는 방법을 결정하여 구현해야 합니다.

5.1.3. Playbook 실행을 위한 머신 준비

RHEL(Red Hat Enterprise Linux)을 운영 체제로 사용하는 컴퓨팅 머신을 **OpenShift Container Platform 4.9** 클러스터에 추가하려면 **RHEL 7** 시스템을 준비하여 새 노드를 클러스터에 추가하는 **Ansible** 플레이북을 실행해야 합니다. 이 머신은 클러스터의 일부가 아니지만 클러스터에 액세스할 수 있어야합니다.

전제 조건

- **Playbook**을 실행하는 머신에 **OpenShift CLI (oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

절차

1. 클러스터의 **kubeconfig** 파일과 클러스터를 설치하는 데 사용한 설치 프로그램이 **RHEL 7** 시스템에 있는지 확인합니다. 이를 수행하는 한 가지 방법으로 클러스터 설치에 사용된 머신과 동일한 머신을 사용하는 것입니다.
2. 컴퓨팅 머신으로 사용하려는 모든 **RHEL** 호스트에 액세스하도록 머신을 구성합니다. **SSH** 프록시 또는 **VPN**을 사용하는 **Bastion**를 포함하여 회사에서 허용하는 모든 방법을 사용할 수 있습니다.
3. **Playbook**을 실행하는 머신에서 모든 **RHEL** 호스트에 대한 **SSH** 액세스 권한이있는 사용자를 구성하십시오.

중요

SSH 키 기반 인증을 사용하는 경우 **SSH** 에이전트를 사용하여 키를 관리해야 합니다.

4. 아직 등록하지 않은 경우 **RHSM**으로 머신을 등록하고 **OpenShift** 서브스크립션이 있는 풀을 머신에 연결합니다.

a. RHSM으로 머신를 등록합니다.

```
# subscription-manager register --username=<user_name> --password=
<password>
```

b. RHSM에서 최신 서브스크립션 데이터를 가져옵니다.

```
# subscription-manager refresh
```

c. 사용 가능한 서브스크립션을 나열하십시오.

```
# subscription-manager list --available --matches '*OpenShift*'
```

d. 이전 명령의 출력에서 OpenShift Container Platform 서브스크립션의 풀 ID를 찾아서 이를 연결합니다.

```
# subscription-manager attach --pool=<pool_id>
```

5. OpenShift Container Platform 4.9에 필요한 리포지토리를 활성화합니다.

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ansible-2.9-rpms" \
    --enable="rhel-7-server-ose-4.9-rpms"
```

6. openshift-ansible을 포함한 필수 패키지를 설치합니다.

```
# yum install openshift-ansible openshift-clients jq
```

openshift-ansible 패키지는 설치 프로그램 유틸리티를 제공하고 Ansible, Playbook 및 관련 구성 파일과 같이 RHEL 컴퓨팅 노드를 클러스터에 추가하는데 필요한 다른 패키지를 가져옵니다. openshift-clients는 oc CLI를 제공하고 jq 패키지는 명령 행에서 JSON 출력 표시 방법을 개선할 수 있습니다.

5.1.4. RHEL 컴퓨팅 노드 준비

RHEL (Red Hat Enterprise Linux) 시스템을 OpenShift Container Platform 클러스터에 추가하기

전에 각 호스트를 **RHSM (Red Hat Subscription Manager)**에 등록하고 활성 **OpenShift Container Platform** 서브스크립션을 연결하고 필요한 저장소를 활성화해야합니다.

1. 각 호스트에서 **RHSM**으로 동록합니다.

```
# subscription-manager register --username=<user_name> --password=<password>
```

2. **RHSM**에서 최신 서브스크립션 데이터를 가져옵니다.

```
# subscription-manager refresh
```

3. 사용 가능한 서브스크립션을 나열하십시오.

```
# subscription-manager list --available --matches '*OpenShift*'
```

4. 이전 명령의 출력에서 **OpenShift Container Platform** 서브스크립션의 풀 **ID**를 찾아서 이를 연결합니다.

```
# subscription-manager attach --pool=<pool_id>
```

5. 모든 **yum** 저장소를 비활성화합니다.

   a. 활성화된 모든 **RHSM** 저장소를 비활성화합니다.

   ```
   # subscription-manager repos --disable="*"
   ```

   b. 나머지 **yum** 저장소를 나열하고 **repo id** 아래에 해당 이름을 적어 둡니다.

   ```
   # yum repolist
   ```

   c. **yum-config-manager**를 사용하여 나머지 **yum** 리포지토리를 비활성화합니다.

   ```
   # yum-config-manager --disable <repo_id>
   ```

   또는 모든 리포지토리를 비활성화합니다.

```
# yum-config-manager --disable \*
```

사용 가능한 리포지토리가 많으면 몇 분의 시간이 소요될 수 있습니다.

6. OpenShift Container Platform 4.9에 필요한 리포지토리를 활성화합니다.

a. RHEL 7 노드의 경우 다음 리포지토리를 활성화해야 합니다.

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-fast-datapath-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-optional-rpms" \
    --enable="rhel-7-server-ose-4.9-rpms"
```

참고

RHEL 7 노드는 더 이상 사용되지 않으며 향후 OpenShift Container Platform 4 릴리스에서 제거될 예정입니다.

b. RHEL 8 노드의 경우 다음 리포지토리를 활성화해야 합니다.

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.9-for-rhel-8-x86_64-rpms" \
    --enable="fast-datapath-for-rhel-8-x86_64-rpms"
```

7. 호스트에서 firewalld를 중지하고 비활성화합니다.

```
# systemctl disable --now firewalld.service
```

참고

나중에 firewalld를 활성화할 수 없습니다. 활성화하면 작업자의 OpenShift Container Platform 로그에 액세스할 수 없습니다.

5.1.5. 클러스터에 RHEL 컴퓨팅 머신 추가

Red Hat Enterprise Linux를 운영 체제로 사용하는 컴퓨팅 머신을 OpenShift Container Platform 4.9 클러스터에 추가할 수 있습니다.

사전 요구 사항

- Playbook을 실행하는 머신에 필요한 패키지를 설치하고 필요한 구성이 수행되어 있습니다.
- RHEL 호스트 설치가 준비되어 있습니다.

프로세스

Playbook을 실행할 준비가 되어 있는 머신에서 다음 단계를 수행합니다.

1. 컴퓨팅 머신 호스트 및 필수 변수를 정의하는 /<path>/inventory/hosts라는 Ansible 인벤토리 파일을 만듭니다.

```
[all:vars]
ansible_user=root 1
#ansible_become=True 2

openshift_kubeconfig_path="~/.kube/config" 3

[new_workers] 4
mycluster-rhel8-0.example.com
mycluster-rhel8-1.example.com
```

1. 원격 컴퓨팅 머신에서 Ansible 태스크를 실행하는 사용자 이름을 지정합니다.
2. ansible_user의 root를 지정하지 않으면 ansible_become을 True로 설정하고 사용자 sudo 권한을 지정해야합니다.
3. 클러스터 kubeconfig 파일의 경로와 파일 이름을 지정합니다.
4. 클러스터에 추가할 각 RHEL 머신을 나열합니다. 각 호스트에 대해 정규화된 도메인 이름을 지정해야합니다. 이 이름은 클러스터가 시스템에 액세스하는 데 사용하는 호스트 이

름이므로 올바른 공용 또는 개인 이름을 설정하여 시스템에 액세스합니다.

2. Ansible Playbook 디렉토리로 이동합니다.

```
$ cd /usr/share/ansible/openshift-ansible
```

3. Playbook을 실행합니다.

```
$ ansible-playbook -i /<path>/inventory/hosts playbooks/scaleup.yml 1
```

1 <path>에 대해 생성한 Ansible 인벤토리 파일의 경로를 지정합니다.

5.1.6. Ansible 호스트 파일의 필수 매개 변수

RHEL (Red Hat Enterprise Linux) 컴퓨팅 머신을 클러스터에 추가하기 전에 Ansible 호스트 파일에서 다음 매개 변수를 정의해야합니다.

매개 변수	설명	값
ansible_user	암호없이 SSH 기반 인증을 허용하는 SSH 사용자입니다. SSH 키 기반 인증을 사용하는 경우 SSH 에이전트를 사용하여 키를 관리해야합니다.	시스템의 사용자 이름입니다. 기본값은 **root**입니다.
ansible_become	**ansible_user** 값이 root가 아닌 경우 **ansible_become**을 **True**로 설정하고 **ansible_user**로 지정하는 사용자는 암호없이 sudo 액세스를 구성해야합니다.	**True**. 값이 **True** 가 아닌 경우 이 매개 변수를 지정하거나 정의하지 마십시오.
openshift_kubeconfig_path	클러스터의 **kubeconfig** 파일이 포함된 로컬 디렉토리의 경로와 파일 이름을 지정합니다.	구성 파일의 경로 및 이름

5.1.7. 선택 사항: 클러스터에서 RHCOS 컴퓨팅 머신 제거

RHEL (Red Hat Enterprise Linux) 컴퓨팅 머신을 클러스터에 추가 한 후 선택 옵션으로 RHCOS (Red Hat Enterprise Linux CoreOS) 컴퓨팅 머신을 제거하여 리소스를 확보할 수 있습니다.

전제 조건

- RHEL 컴퓨팅 머신이 클러스터에 추가되어 있습니다.

프로세스

1. 머신 목록을 표시하고 RHCOS 컴퓨팅 머신의 노드 이름을 기록합니다.

```
$ oc get nodes -o wide
```

2. 각 RHCOS 컴퓨팅 머신의 노드를 제거합니다.

   a. oc adm cordon 명령을 실행하여 노드를 스케줄 예약 해제로 표시합니다.

```
$ oc adm cordon <node_name> 1
```

1 RHCOS 컴퓨팅 머신의 노드 이름을 지정합니다.

   b. 노드에서 모든 pod를 드레인합니다.

```
$ oc adm drain <node_name> --force --delete-emptydir-data --ignore-daemonsets
1
```

1 분리 한 RHCOS 컴퓨팅 머신의 노드 이름을 지정합니다.

   c. 노드를 제거합니다.

```
$ oc delete nodes <node_name> 1
```

1 드레인한 RHCOS 컴퓨팅 머신의 노드 이름을 지정합니다.

3. 컴퓨팅 머신 목록을 확인하고 **RHEL** 노드만 남아 있는지 확인합니다.

```
$ oc get nodes -o wide
```

4. 클러스터 컴퓨팅 머신의 로드 밸런서에서 **RHCOS** 머신을 제거합니다. 가상 머신을 삭제하거나 **RHCOS** 컴퓨팅 머신의 실제 하드웨어를 다시 이미지화 할 수 있습니다.

5.2. OPENSHIFT CONTAINER PLATFORM 클러스터에 RHCOS 컴퓨팅 머신 추가

베어 메탈의 **OpenShift Container Platform** 클러스터에 더 많은 **Red Hat Enterprise Linux CoreOS (RHCOS)** 컴퓨팅 머신을 추가할 수 있습니다.

베어메탈 인프라에 설치된 클러스터에 컴퓨팅 머신을 추가하기 전에 사용할 **RHCOS** 머신을 생성해야 합니다. **ISO** 이미지 또는 네트워크 **PXE** 부팅을 사용하여 시스템을 생성합니다.

5.2.1. 사전 요구 사항

- 베어 메탈에 클러스터가 설치되어 있어야 합니다.
- 클러스터를 만드는 데 사용한 설치 미디어 및 **Red Hat Enterprise Linux CoreOS (RHCOS)** 이미지가 있습니다. 이러한 파일이 없는 경우 설치 절차에 따라 파일을 가져와야 합니다.

5.2.2. ISO 이미지를 사용하여 추가 RHCOS 머신 생성

ISO 이미지를 사용하여 머신을 생성함으로써 베어 메탈 클러스터에 대해 더 많은 **Red Hat Enterprise Linux CoreOS (RHCOS)** 컴퓨팅 머신을 생성할 수 있습니다.

사전 요구 사항

- 클러스터의 컴퓨팅 머신에 대한 **Ignition** 구성 파일의 **URL**을 가져옵니다. 설치 중에 이 파일은 **HTTP** 서버에 업로드되어 있어야 합니다.

절차

1. **ISO** 파일을 사용하여 추가 컴퓨팅 머신에 **RHCOS**를 설치합니다. 클러스터를 설치하기 전에 머신을 만들 때 사용한 것과 동일한 방법을 사용합니다.

- ISO 이미지를 디스크에 굽고 직접 부팅합니다.
- LOM 인터페이스에서 ISO 리디렉션을 사용합니다.

2. 옵션을 지정하거나 라이브 부팅 시퀀스를 중단하지 않고 RHCOS ISO 이미지를 부팅합니다. 설치 프로그램이 RHCOS 라이브 환경에서 쉘 프롬프트로 부팅될 때까지 기다립니다.

참고

커널 인수를 추가하기 위해 RHCOS 설치 부팅 프로세스를 중단할 수 있습니다. 그러나 이 ISO 프로세스에서는 커널 인수를 추가하는 대신 다음 단계에 설명된 대로 coreos-installer 명령을 사용해야 합니다.

3. coreos-installer 명령을 실행하고 설치 요구 사항을 충족하는 옵션을 지정합니다. 최소한 노드 유형에 대한 Ignition 구성 파일과 설치할 장치를 가리키는 URL을 지정해야 합니다.

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign
<device> --ignition-hash=sha512-<digest> 1 2
```

1. core 사용자에게 설치를 수행하는 데 필요한 root 권한이 없으므로 sudo를 사용하여 coreos-installer 명령을 실행해야 합니다.
2. 클러스터 노드에서 Ignition 구성 파일을 HTTP URL을 통해 가져오려면 --ignition-hash 옵션이 필요합니다. <digest>는 이전 단계에서 얻은 Ignition 구성 파일 SHA512 다이제스트입니다.

참고

TLS를 사용하는 HTTPS 서버를 통해 Ignition 구성 파일을 제공하려는 경우 coreos-installer를 실행하기 전에 내부 인증 기관(CA)을 시스템 신뢰 저장소에 추가할 수 있습니다.

다음 예제에서는 /dev/sda 장치에 부트스트랩 노드 설치를 초기화합니다. 부트스트랩 노드의 Ignition 구성 파일은 IP 주소 192.168.1.2가 있는 HTTP 웹 서버에서 가져옵니다.

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-
hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78dd
f0116e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

4. 머신 콘솔에서 **RHCOS** 설치 진행률을 모니터링합니다.

중요

OpenShift Container Platform 설치를 시작하기 전에 각 노드에서 성공적으로 설치되었는지 확인합니다. 설치 프로세스를 관찰하면 발생할 수 있는 **RHCOS** 설치 문제의 원인을 파악하는 데 도움이 될 수 있습니다.

5. 계속해서 클러스터에 추가 컴퓨팅 머신을 만듭니다.

5.2.3. PXE 또는 iPXE 부팅을 통해 RHCOS 머신 생성

PXE 또는 **iPXE** 부팅을 사용하여 베어 메탈 클러스터에 대해 추가 **Red Hat Enterprise Linux CoreOS (RHCOS)** 컴퓨팅 머신을 생성할 수 있습니다.

사전 요구 사항

- 클러스터의 컴퓨팅 머신에 대한 **Ignition** 구성 파일의 **URL**을 가져옵니다. 설치 중에 이 파일은 **HTTP** 서버에 업로드되어 있어야 합니다.
- 클러스터 설치 중에 **HTTP** 서버에 업로드 한 **RHCOS ISO** 이미지, 압축된 메탈 **BIOS, kernel** 및 **initramfs** 파일의 **URL**을 가져옵니다.
- 설치 중에 **OpenShift Container Platform** 클러스터에 대한 머신을 생성하는 데 사용한 **PXE** 부팅 인프라에 액세스할 수 있습니다. **RHCOS**가 설치된 후 로컬 디스크에서 머신을 부팅해야합니다.
- **UEFI**를 사용하는 경우 **OpenShift Container Platform** 설치 중에 수정 한 **grub.conf** 파일에 액세스할 수 있습니다.

프로세스

1. **RHCOS** 이미지의 **PXE** 또는 **iPXE**가 올바르게 설치되었는지 확인합니다.

- **PXE**의 경우:

```
DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> (1)
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img (2)
```

(1) **HTTP** 서버에 업로드한 라이브 **kernel** 파일의 위치를 지정합니다.

(2) **HTTP** 서버에 업로드한 **RHCOS** 파일의 위치를 지정합니다. **initrd** 매개변수 값은 **initramfs** 파일의 위치이고 **coreos.inst.ignition_url** 매개변수 값은 작업자 **Ignition** 설정 파일의 위치이며 **coreos.live.rootfs_url** 매개 변수 값은 라이브 **rootfs** 파일의 위치입니다. **coreos.inst.ignition_url** 및 **coreos.live.rootfs_url** 매개변수는 **HTTP** 및 **HTTPS**만 지원합니다.

이 구성은 그래픽 콘솔이 있는 시스템에서 직렬 콘솔 액세스를 활성화하지 않습니다. 다른 콘솔을 구성하려면 **APPEND** 행에 하나 이상의 **console=** 인수를 추가합니다. 예를 들어 **console=tty0 console=ttyS0**을 추가하여 첫 번째 **PC** 직렬 포트를 기본 콘솔로 설정하고 그래픽 콘솔을 보조 콘솔로 설정합니다. 자세한 내용은 **Red Hat Enterprise Linux**에서 직렬 터미널 및/또는 콘솔 설정 방법을 참조하십시오.

- **iPXE**의 경우 :

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.inst.install_dev=/dev/sda coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.<architecture>.img
(1)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.<architecture>.img
(2)
```

(1)

2

HTTP 서버에 업로드한 initramfs 파일의 위치를 지정합니다.

이 구성은 그래픽 콘솔이 있는 시스템에서 직렬 콘솔 액세스를 활성화하지 않습니다. 다른 콘솔을 구성하려면 kernel 행에 하나 이상의 console= 인수를 추가합니다. 예를 들어 console=tty0 console=ttyS0을 추가하여 첫 번째 PC 직렬 포트를 기본 콘솔로 설정하고 그래픽 콘솔을 보조 콘솔로 설정합니다. 자세한 내용은 Red Hat Enterprise Linux에서 직렬 터미널 및/또는 콘솔 설정 방법을 참조하십시오.

1. PXE 또는 iPXE 인프라를 사용하여 클러스터에 필요한 컴퓨팅 머신을 만듭니다.

5.2.4. 시스템의 인증서 서명 요청 승인

클러스터에 시스템을 추가하면 추가한 시스템별로 보류 중인 인증서 서명 요청(CSR)이 두 개씩 생성됩니다. 이러한 CSR이 승인되었는지 확인해야 하며, 필요한 경우 이를 직접 승인해야 합니다. 클라이언트 요청을 먼저 승인한 다음 서버 요청을 승인해야 합니다.

사전 요구 사항

- 클러스터에 시스템을 추가했습니다.

프로세스

1. 클러스터가 시스템을 인식하는지 확인합니다.

```
$ oc get nodes
```

출력 예

```
NAME     STATUS   ROLES   AGE  VERSION
master-0 Ready    master  63m  v1.22.1
master-1 Ready    master  63m  v1.22.1
master-2 Ready    master  64m  v1.22.1
```

출력에 생성된 모든 시스템이 나열됩니다.

참고

이전 출력에는 일부 **CSR**이 승인될 때까지 컴퓨팅 노드(작업자 노드라고도 함)가 포함되지 않을 수 있습니다.

2. 보류 중인 **CSR**을 검토하고 클러스터에 추가한 각 시스템에 대해 **Pending** 또는 **Approved** 상태의 클라이언트 및 서버 요청이 표시되는지 확인합니다.

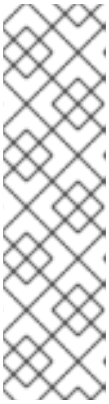
```
$ oc get csr
```

출력 예

```
NAME        AGE    REQUESTOR                                                                   CONDITION
csr-8b2br   15m    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
csr-8vnps   15m    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
...
```

예에서는 두 시스템이 클러스터에 참여하고 있습니다. 목록에는 승인된 **CSR**이 더 많이 나타날 수도 있습니다.

3. **CSR**이 승인되지 않은 경우, 추가된 시스템에 대한 모든 보류 중인 **CSR**이 **Pending** 상태로 전환된 후 클러스터 시스템의 **CSR**을 승인합니다.



참고

CSR은 교체 주기가 자동으로 만료되므로 클러스터에 시스템을 추가한 후 1 시간 이내에 CSR을 승인하십시오. 한 시간 내에 승인하지 않으면 인증서가 교체되고 각 노드에 대해 두 개 이상의 인증서가 표시됩니다. 이러한 인증서를 모두 승인해야 합니다. 클라이언트 CSR이 승인되면 Kubelet은 인증서에 대한 보조 CSR을 생성하므로 수동 승인이 필요합니다. 그러면 Kubelet에서 동일한 매개변수를 사용하여 새 인증서를 요청하는 경우 인증서 갱신 요청은 machine-approver에 의해 자동으로 승인됩니다.

참고

베어 메탈 및 기타 사용자 프로비저닝 인프라와 같이 머신 API를 사용하도록 활성화되지 않는 플랫폼에서 실행되는 클러스터의 경우 CSR(Kubelet service Certificate Request)을 자동으로 승인하는 방법을 구현해야 합니다. 요청이 승인되지 않으면 API 서버가 kubelet에 연결될 때 서비스 인증서가 필요하므로 oc exec, oc rsh, oc logs 명령을 성공적으로 수행할 수 없습니다. Kubelet 엔드 포인트에 연결하는 모든 작업을 수행하려면 이 인증서 승인이 필요합니다. 이 방법은 새 CSR을 감시하고 CSR이 system:node 또는 system:admin 그룹의 node-bootstrapper 서비스 계정에 의해 제출되었는지 확인하고 노드의 ID를 확인합니다.

- 개별적으로 승인하려면 유효한 CSR 각각에 대해 다음 명령을 실행하십시오.

```
$ oc adm certificate approve <csr_name> 1
```

1 <csr_name>은 현재 CSR 목록에 있는 CSR의 이름입니다.

- 보류 중인 CSR을 모두 승인하려면 다음 명령을 실행하십시오.

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}
{{"\n"}}{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

참고

일부 Operator는 일부 CSR이 승인될 때까지 사용할 수 없습니다.

4. 이제 클라이언트 요청이 승인되었으므로 클러스터에 추가한 각 머신의 서버 요청을 검토해야 합니다.

```
$ oc get csr
```

출력 예

```
NAME        AGE     REQUESTOR                                                   CONDITION
csr-bfd72   5m26s   system:node:ip-10-0-50-126.us-east-2.compute.internal        Pending
csr-c57lv   5m26s   system:node:ip-10-0-95-157.us-east-2.compute.internal        Pending
...
```

5. 나머지 **CSR**이 승인되지 않고 **Pending** 상태인 경우 클러스터 머신의 **CSR**을 승인합니다.

- 개별적으로 승인하려면 유효한 **CSR** 각각에 대해 다음 명령을 실행하십시오.

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>**은 현재 **CSR** 목록에 있는 **CSR**의 이름입니다.

- 보류 중인 **CSR**을 모두 승인하려면 다음 명령을 실행하십시오.

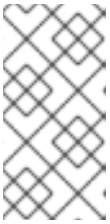
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}
{{"\n"}}{{end}}{{end}}' | xargs oc adm certificate approve
```

6. 모든 클라이언트 및 서버 **CSR**이 승인된 후 머신은 **Ready** 상태가 됩니다. 다음 명령을 실행하여 확인합니다.

```
$ oc get nodes
```

출력 예

```
NAME     STATUS   ROLES   AGE  VERSION
master-0  Ready    master  73m  v1.22.1
master-1  Ready    master  73m  v1.22.1
master-2  Ready    master  74m  v1.22.1
worker-0  Ready    worker  11m  v1.22.1
worker-1  Ready    worker  11m  v1.22.1
```



참고

머신이 **Ready** 상태로 전환하는 데 서버 **CSR**의 승인 후 몇 분이 걸릴 수 있습니다.

추가 정보

- **CSR**에 대한 자세한 내용은 인증서 서명 요청을 참조하십시오.

5.3. 머신 상태 확인

머신 상태 확인을 이해하고 배포합니다.

중요

머신 **API**가 작동하는 클러스터에서만 고급 머신 관리 및 스케일링 기능을 사용할 수 있습니다. 사용자 프로비저닝 인프라가 있는 클러스터에는 **Machine API**를 사용하려면 추가 검증 및 구성이 필요합니다.

인프라 플랫폼 유형의 클러스터가 **Machine API**를 사용할 수 없습니다. 이 제한은 클러스터에 연결된 컴퓨팅 시스템이 기능을 지원하는 플랫폼에 설치된 경우에도 적용됩니다. 이 매개변수는 설치 후 변경할 수 없습니다.

클러스터의 플랫폼 유형을 보려면 다음 명령을 실행합니다.

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

5.3.1. 머신 상태 점검 정보

머신 상태 점검에서는 특정 머신 풀의 비정상적인 머신을 자동으로 복구합니다.

머신 상태를 모니터링하기 위해 컨트롤러 구성을 정의할 리소스를 만듭니다. **NotReady** 상태를 **5** 분 동안 유지하거나 노드 문제 탐지기(**node-problem-detector**)에 영구적인 조건을 표시하는 등 검사할 조건과 모니터링할 머신 세트의 레이블을 설정합니다.

참고

마스터 역할이 있는 머신에는 머신 상태 점검을 적용할 수 없습니다.

MachineHealthCheck 리소스를 관찰하는 컨트롤러에서 정의된 상태를 확인합니다. 머신이 상태 확인에 실패하면 머신이 자동으로 삭제되고 대체할 머신이 만들어집니다. 머신이 삭제되면 **machine deleted** 이벤트가 표시됩니다.

머신 삭제로 인한 영향을 제한하기 위해 컨트롤러는 한 번에 하나의 노드 만 드레인하고 삭제합니다. 대상 머신 풀에서 허용된 **maxUnhealthy** 임계값 보다 많은 비정상적인 머신이 있는 경우 수동 개입이 수행될 수 있도록 복구가 중지됩니다.

참고

워크로드 및 요구 사항을 살펴보고 신중하게 시간 초과를 고려하십시오.

- 시간 제한이 길어지면 비정상 머신의 워크로드에 대한 다운타임이 길어질 수 있습니다.
- 시간 초과가 너무 짧으면 수정 루프가 발생할 수 있습니다. 예를 들어 **NotReady** 상태를 확인하는 시간은 머신이 시작 프로세스를 완료할 수 있을 만큼 충분히 길어야 합니다.

검사를 중지하려면 리소스를 제거합니다.

5.3.1.1. 머신 상태 검사 배포 시 제한 사항

머신 상태 점검을 배포하기 전에 고려해야 할 제한 사항은 다음과 같습니다.

- 머신 세트가 소유한 머신만 머신 상태 검사를 통해 업데이트를 적용합니다.
- 컨트롤 플레인 시스템은 현재 지원되지 않으며 비정상적인 경우 업데이트 적용되지 않습니다.
- 머신의 노드가 클러스터에서 제거되면 머신 상태 점검에서 이 머신을 비정상적으로 간주하고 즉시 업데이트를 적용합니다.
- **nodeStartupTimeout** 후 시스템의 해당 노드가 클러스터에 참여하지 않으면 업데이트가 적용됩니다.
- **Machine** 리소스 단계가 **Failed**하면 즉시 머신에 업데이트를 적용합니다.

5.3.2. MachineHealthCheck 리소스 샘플

베어 메탈 이외의 모든 클라우드 기반 설치 유형에 대한 **MachineHealthCheck** 리소스는 다음 **YAML** 파일과 유사합니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role> 3
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" 5
    status: "False"
  - type:    "Ready"
    timeout: "300s" 6
    status: "Unknown"
  maxUnhealthy: "40%" 7
  nodeStartupTimeout: "10m" 8
```

1

배포할 머신 상태 점검의 이름을 지정합니다.

2 3

확인할 머신 풀의 레이블을 지정합니다.

4

추적할 머신 세트를 **<cluster_name>-<label>-<zone>** 형식으로 지정합니다. 예를 들어 **prod-node-us-east-1a**입니다.

5 6

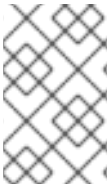
노드 상태에 대한 시간 제한을 지정합니다. 시간 제한 기간 중 상태가 일치되면 머신이 수정됩니다. 시간 제한이 길어지면 비정상 머신의 워크로드에 대한 다운타임이 길어질 수 있습니다.

7

대상 풀에서 동시에 복구할 수 있는 시스템 수를 지정합니다. 이는 백분율 또는 정수로 설정할 수 있습니다. 비정상 머신의 수가 **maxUnhealthy**에서의 설정 제한을 초과하면 복구가 수행되지 않습니다.

8

머신 상태가 비정상으로 확인되기 전에 노드가 클러스터에 참여할 때까지 기다려야 하는 시간 초과 기간을 지정합니다.



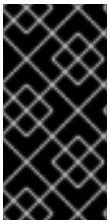
참고

matchLabels는 예제일 뿐입니다. 특정 요구에 따라 머신 그룹을 매핑해야 합니다.

5.3.2.1. 쇼트 서킷 (Short Circuit) 머신 상태 점검 및 수정

쇼트 서킷 (Short Circuit)은 클러스터가 정상일 경우에만 머신 상태 점검을 통해 머신을 조정합니다. 쇼트 서킷은 **MachineHealthCheck** 리소스의 **maxUnhealthy** 필드를 통해 구성됩니다.

사용자가 시스템을 조정하기 전에 **maxUnhealthy** 필드 값을 정의하는 경우 **MachineHealthCheck**는 비정상적으로 결정된 대상 풀 내의 **maxUnhealthy** 값과 비교합니다. 비정상 머신의 수가 **maxUnhealthy** 제한을 초과하면 수정을 위한 업데이트가 수행되지 않습니다.



중요

maxUnhealthy가 설정되지 않은 경우 기본값은 **100%**로 설정되고 클러스터 상태와 관계없이 머신이 수정됩니다.

적절한 **maxUnhealthy** 값은 배포하는 클러스터의 규모와 **MachineHealthCheck**에서 다루는 시스템 수에 따라 달라집니다. 예를 들어 **maxUnhealthy** 값을 사용하여 여러 가용 영역에서 여러 머신 세트를 처리할 수 있으므로 전체 영역을 손실하면 **maxUnhealthy** 설정이 클러스터 내에서 추가 수정을 방지 할 수 있습니다.

maxUnhealthy 필드는 정수 또는 백분율로 설정할 수 있습니다. **maxUnhealthy** 값에 따라 다양한 수정을 적용할 수 있습니다.

5.3.2.1.1. 절대 값을 사용하여 **maxUnhealthy** 설정

maxUnhealthy가 **2**로 설정된 경우

- **2**개 이상의 노드가 비정상인 경우 수정을 위한 업데이트가 수행됩니다.
- **3**개 이상의 노드가 비정상이면 수정을 위한 업데이트가 수행되지 않습니다

이러한 값은 머신 상태 점검에서 확인할 수 있는 머신 수와 관련이 없습니다.

5.3.2.1.2. 백분율을 사용하여 **maxUnhealthy** 설정

maxUnhealthy가 **40%**로 설정되어 있고 **25** 대의 시스템이 확인되고 있는 경우 다음을 수행하십시오.

- **10**개 이상의 노드가 비정상인 경우 수정을 위한 업데이트가 수행됩니다.
- **11**개 이상의 노드가 비정상인 경우 수정을 위한 업데이트가 수행되지 않습니다.

maxUnhealthy가 **40%**로 설정되어 있고 **6** 대의 시스템이 확인되고 있는 경우 다음을 수행하십시오.

- 2개 이상의 노드가 비정상인 경우 수정을 위한 업데이트가 수행됩니다.

- 3개 이상의 노드가 비정상이면 수정을 위한 업데이트가 수행되지 않습니다

참고

maxUnhealthy 머신의 백분율이 정수가 아닌 경우 허용되는 머신 수가 반올림됩니다.

5.3.3. MachineHealthCheck 리소스 만들기

클러스터의 모든 MachineSets에 대해 MachineHealthCheck 리소스를 생성할 수 있습니다. 컨트롤 플레인 시스템을 대상으로 하는 MachineHealthCheck 리소스를 생성해서는 안 됩니다.

사전 요구 사항

- oc 명령행 인터페이스를 설치합니다.

절차

1. 머신 상태 점검 정의가 포함된 healthcheck.yml 파일을 생성합니다.

2. healthcheck.yml 파일을 클러스터에 적용합니다.

```
$ oc apply -f healthcheck.yml
```

5.3.4. 머신 세트 수동 스케일링

머신 세트에서 머신 인스턴스를 추가하거나 제거하려면 머신 세트를 수동으로 스케일링할 수 있습니다.

이는 완전히 자동화된 설치 프로그램에 의해 프로비저닝된 인프라 설치와 관련이 있습니다. 사용자 지정된 사용자 프로비저닝 인프라 설치에는 머신 세트가 없습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터 및 oc 명령행을 설치합니다.
- cluster-admin 권한이 있는 사용자로 oc에 로그인합니다.

절차

1. 클러스터에 있는 머신 세트를 확인합니다.

```
$ oc get machinesets -n openshift-machine-api
```

   머신 세트는 <clusterid>-worker-<aws-region-az> 형식으로 나열됩니다.

2. 클러스터에 있는 머신을 확인합니다.

```
$ oc get machine -n openshift-machine-api
```

3. 삭제하려는 머신에 주석을 설정합니다.

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/cluster-api-delete-machine="true"
```

4. 삭제하려는 노드를 비우고 제외합니다.

```
$ oc adm cordon <node_name>
$ oc adm drain <node_name>
```

5. 머신 세트를 스케일링합니다.

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

   또는 다음을 수행합니다.

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

작은 정보

다음 YAML을 적용하여 머신 세트를 확장할 수도 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

머신 세트를 확장 또는 축소할 수 있습니다. 새 머신을 사용할 수 있을 때 까지 몇 분 정도 소요됩니다.

검증

- 원하는 머신 삭제를 확인합니다.

```
$ oc get machines
```

5.3.5. 머신 세트와 머신 구성 풀 간의 차이점

MachineSet 개체는 클라우드 또는 머신 공급자와 관련하여 OpenShift Container Platform 노드를 설명합니다.

MachineConfigPool 개체를 사용하면 MachineConfigController 구성 요소가 업그레이드 컨텍스트에서 시스템의 상태를 정의하고 제공할 수 있습니다.

MachineConfigPool 개체를 사용하여 시스템 구성 풀의 OpenShift Container Platform 노드에 대한 업그레이드 방법을 구성할 수 있습니다.

NodeSelector 개체는 MachineSet에 대한 참조로 대체할 수 있습니다.

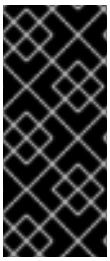
5.4. 노드 호스트 관련 권장 사례

OpenShift Container Platform 노드 구성 파일에는 중요한 옵션이 포함되어 있습니다. 예를 들어 두 개의 매개변수 podsPerCore 및 maxPods는 하나의 노드에 대해 예약할 수 있는 최대 Pod 수를 제어합

니다.

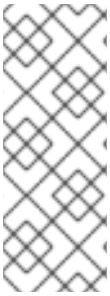
옵션을 둘 다 사용하는 경우 한 노드의 **Pod** 수는 두 값 중 작은 값으로 제한됩니다. 이 값을 초과하면 다음과 같은 결과가 발생할 수 있습니다.

- **CPU** 사용률 증가
- **Pod** 예약 속도 저하
- 노드의 메모리 크기에 따라 메모리 부족 시나리오 발생
- **IP** 주소 모두 소진
- 리소스 초과 커밋으로 인한 사용자 애플리케이션 성능 저하



중요

Kubernetes의 경우 단일 컨테이너를 보유한 하나의 **Pod**에서 실제로 두 개의 컨테이너가 사용됩니다. 두 번째 컨테이너는 실제 컨테이너 시작 전 네트워킹 설정에 사용됩니다. 따라서 **10**개의 **Pod**를 실행하는 시스템에서는 실제로 **20**개의 컨테이너가 실행됩니다.



참고

클라우드 공급자의 디스크 **IOPS** 제한이 **CRI-O** 및 **kubelet**에 영향을 미칠 수 있습니다. 노드에서 다수의 **I/O** 집약적 **Pod**가 실행되고 있는 경우 오버로드될 수 있습니다. 노드에서 디스크 **I/O**를 모니터링하고 워크로드에 대해 처리량이 충분한 볼륨을 사용하는 것이 좋습니다.

podsPerCore는 노드의 프로세서 코어 수에 따라 노드가 실행할 수 있는 **Pod** 수를 설정합니다. 예를 들어 프로세서 코어가 **4**개인 노드에서 **podsPerCore**가 **10**으로 설정된 경우 노드에 허용되는 최대 **Pod** 수는 **40**이 됩니다.

```
kubeletConfig:
  podsPerCore: 10
```

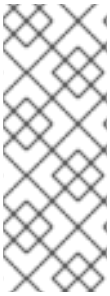
podsPerCore를 0으로 설정하면 이 제한이 비활성화됩니다. 기본값은 0입니다. podsPerCore는 maxPods를 초과할 수 없습니다.

maxPods는 노드의 속성에 관계없이 노드가 실행할 수 있는 Pod 수를 고정된 값으로 설정합니다.

```
kubeletConfig:
  maxPods: 250
```

5.4.1. KubeletConfig CRD를 생성하여 kubelet 매개변수 편집

kubelet 구성은 현재 Ignition 구성으로 직렬화되어 있으므로 직접 편집할 수 있습니다. 하지만 MCC(Machine Config Controller)에 새 kubelet-config-controller도 추가되어 있습니다. 이를 통해 KubeletConfig CR(사용자 정의 리소스)을 사용하여 kubelet 매개변수를 편집할 수 있습니다.



참고

kubeletConfig 오브젝트의 필드가 Kubernetes 업스트림에서 kubelet으로 직접 전달되므로 kubelet은 해당 값을 직접 검증합니다. kubeletConfig 오브젝트의 값이 유효하지 않으면 클러스터 노드를 사용할 수 없게 될 수 있습니다. 유효한 값은 Kubernetes 설명서를 참조하십시오.

다음 지침 사항을 고려하십시오.

- 해당 풀에 필요한 모든 구성 변경 사항을 사용하여 각 머신 구성 풀에 대해 하나의 KubeletConfig CR을 생성합니다. 모든 풀에 동일한 콘텐츠를 적용하는 경우 모든 풀에 대해 하나의 KubeletConfig CR만 필요합니다.
- 기존 KubeletConfig CR을 편집하여 각 변경 사항에 대한 CR을 생성하는 대신 기존 설정을 수정하거나 새 설정을 추가합니다. 변경 사항을 되돌릴 수 있도록 다른 머신 구성 풀을 수정하거나 임시로 변경하려는 변경 사항만 수정하기 위해 CR을 생성하는 것이 좋습니다.
- 필요에 따라 클러스터당 10개로 제한되는 여러 KubeletConfig CR을 생성합니다. 첫 번째 KubeletConfig CR의 경우 MCO(Machine Config Operator)는 kubelet에 추가된 머신 구성을 생성합니다. 이후 각 CR을 통해 컨트롤러는 숫자 접미사가 있는 다른 kubelet 머신 구성을 생성합니다. 예를 들어, -2 접미사가 있는 kubelet 머신 구성이 있는 경우 다음 kubelet 머신 구성에 -3이 추가됩니다.

머신 구성을 삭제하려면 제한을 초과하지 않도록 해당 구성을 역순으로 삭제합니다. 예를 들어

kubelet-2 머신 구성을 삭제하기 전에 kubelet-3 머신 구성을 삭제합니다.

참고

kubelet-9 접미사가 있는 머신 구성이 있고 다른 KubeletConfig CR을 생성하는 경우 kubelet 머신 구성이 10개 미만인 경우에도 새 머신 구성이 생성되지 않습니다.

KubeletConfig CR 예

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

KubeletConfig 머신 구성 표시 예

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                 b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
26m
...
```

다음 프로세스는 작업자 노드의 각 노드에 대한 최대 Pod 수를 구성하는 방법을 보여줍니다.

사전 요구 사항

1. 구성하려는 노드 유형의 정적 MachineConfigPool CR와 연관된 라벨을 가져옵니다. 다음 중 하나를 실행합니다.

   a. Machine config pool을 표시합니다.

```
$ oc describe machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc describe machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods (1)
```

(1)

라벨이 추가되면 **labels** 아래에 표시됩니다.

b.

라벨이 없으면 키/값 쌍을 추가합니다.

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

절차

1.

이 명령은 선택할 수 있는 사용 가능한 머신 구성 오브젝트를 표시합니다.

```
$ oc get machineconfig
```

기본적으로 두 개의 **kubelet** 관련 구성은 **01-master-kubelet** 및 **01-worker-kubelet**입니다.

2.

노드당 최대 **Pod**의 현재 값을 확인하려면 다음을 실행합니다.

```
$ oc describe node <node_name>
```

예를 들면 다음과 같습니다.

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

Allocatable 스탠자에서 value: pods: <value>를 찾습니다.

출력 예

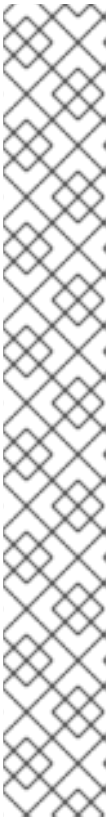
```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

3. 작업자 노드에서 노드당 최대 Pod 수를 설정하려면 kubelet 구성이 포함된 사용자 정의 리소스 파일을 생성합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods (1)
  kubeletConfig:
    maxPods: 500 (2)
```

(1) 머신 구성 풀에서 레이블을 입력합니다.

(2) kubelet 구성을 추가합니다. 이 예에서는 maxPods를 사용하여 노드당 최대 Pod를 설정합니다.



참고

kubelet이 API 서버와 통신하는 속도는 QPS(초당 쿼리) 및 버스트 값에 따라 달라집니다. 노드마다 실행되는 Pod 수가 제한된 경우 기본 값인 50(kubeAPIQPS인 경우) 및 100(kubeAPIBurst인 경우)이면 충분합니다. 노드에 CPU 및 메모리 리소스가 충분한 경우 kubelet QPS 및 버스트 속도를 업데이트하는 것이 좋습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 라벨을 사용하여 작업자의 머신 구성 풀을 업데이트합니다.

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

b. KubeletConfig 오브젝트를 생성합니다.

```
$ oc create -f change-maxPods-cr.yaml
```

c. KubeletConfig 오브젝트가 생성되었는지 확인합니다.

```
$ oc get kubeletconfig
```

출력 예

```
NAME            AGE
set-max-pods    15m
```

클러스터의 작업자 노드 수에 따라 작업자 노드가 하나씩 재부팅될 때까지 기다립니다. 작업자 노드가 3개인 클러스터의 경우 약 10~15분이 걸릴 수 있습니다.

4. 변경 사항이 노드에 적용되었는지 확인합니다.

a. 작업자 노드에서 maxPods 값이 변경되었는지 확인합니다.

```
$ oc describe node <node_name>
```

b. Allocatable 스탠자를 찾습니다.

```
...
Allocatable:
 attachable-volumes-gce-pd:  127
 cpu:                        3500m
 ephemeral-storage:          123201474766
 hugepages-1Gi:              0
 hugepages-2Mi:              0
 memory:                     14225400Ki
 pods:                       500 1
...
```

1 이 예에서 pods 매개변수는 KubeletConfig 오브젝트에 설정한 값을 보고해야 합니다.

5. KubeletConfig 오브젝트에서 변경 사항을 확인합니다.

```
$ oc get kubeletconfigs set-max-pods -o yaml
```

다음 예와 같이 True 및 type:Success 상태가 표시되어야 합니다.

```
spec:
  kubeletConfig:
    maxPods: 500
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
status:
  conditions:
  - lastTransitionTime: "2021-06-30T17:04:07Z"
```

```
    message: Success
    status: "True"
    type: Success
```

5.4.2. 사용할 수 없는 작업자 노드 수 수정

기본적으로 **kubelet** 관련 구성을 사용 가능한 작업자 노드에 적용하는 경우 하나의 머신만 사용할 수 없는 상태로 둘 수 있습니다. 대규모 클러스터의 경우 구성 변경사항을 반영하는 데 시간이 오래 걸릴 수 있습니다. 언제든지 업데이트하는 머신 수를 조정하여 프로세스 속도를 높일 수 있습니다.

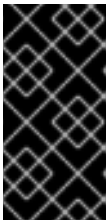
절차

1. **worker** 머신 구성 풀을 편집합니다.

```
$ oc edit machineconfigpool worker
```

2. **maxUnavailable**을 원하는 값으로 설정합니다.

```
spec:
  maxUnavailable: <node_count>
```



중요

값을 설정하는 경우 클러스터에서 실행 중인 애플리케이션에 영향을 미치지 않고 사용 가능한 상태로 둘 수 있는 작업자 노드 수를 고려하십시오.

5.4.3. 컨트롤 플레인 노드 크기 조정

컨트롤 플레인 노드 리소스 요구사항은 클러스터의 노드 수에 따라 달라집니다. 다음 컨트롤 플레인 노드 크기 권장 사항은 컨트롤 플레인 밀도 중심 테스트 결과를 기반으로 합니다. 컨트롤 플레인 테스트에서는 노드 수에 따라 각 네임스페이스의 클러스터에서 다음 오브젝트를 생성합니다.

- 12개의 이미지 스트림
- 3개의 빌드 구성
- 6개의 빌드

- 각각 두 개의 시크릿을 마운트하는 2개의 Pod 복제본이 있는 배포 1개
- 2개의 배포에 1개의 Pod 복제본이 2개의 시크릿을 마운트함
- 이전 배포를 가리키는 서비스 3개
- 이전 배포를 가리키는 경로 3개
- 10개의 시크릿, 이 중 2 개는 이전 배포에 의해 마운트됨
- 10개의 구성 맵, 이 중 두 개는 이전 배포에 의해 마운트됨

작업자 노드 수	클러스터 로드(네임스페이스)	CPU 코어 수	메모리(GB)
25	500	4	16
100	1000	8	32
250	4000	16	96

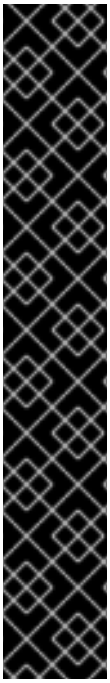
3개의 마스터 또는 컨트롤 플레인 노드가 있는 대규모 및 고밀도 클러스터에서는 노드 중 하나가 중지되거나, 재부팅 또는 실패할 때 CPU 및 메모리 사용량이 증가합니다. 비용 절감을 위해 클러스터를 종료한 후 클러스터를 재시작하는 의도적인 경우 외에도 전원, 네트워크 또는 기본 인프라와 관련된 예기치 않은 문제로 인해 오류가 발생할 수 있습니다. 나머지 두 컨트롤 플레인 노드는 고가용성이 되기 위해 부하를 처리하여 리소스 사용량을 늘려야 합니다. 이는 마스터가 직렬로 연결, 드레이닝, 재부팅되어 운영 체제 업데이트를 적용하고 컨트롤 플레인 Operator 업데이트를 적용하기 때문에 업그레이드 중에도 이 문제가 발생할 수 있습니다. 단계적 오류를 방지하려면 컨트롤 플레인 노드의 전체 CPU 및 메모리 리소스 사용량을 리소스 사용량 급증을 처리하기 위해 사용 가능한 모든 용량의 60%로 유지합니다. 리소스 부족으로 인한 다운타임을 방지하기 위해 컨트롤 플레인 노드에서 CPU 및 메모리를 늘립니다.

중요

노드 크기 조정은 클러스터의 노드 수와 개체 수에 따라 달라집니다. 또한 클러스터에서 개체가 현재 생성되는지에 따라 달라집니다. 개체 생성 중에 컨트롤 플레인은 개체가 running 단계에 있을 때보다 리소스 사용량 측면에서 더 활성화됩니다.

OLM(Operator Lifecycle Manager)은 컨트롤 플레인 노드에서 실행되며, 메모리 공간은 OLM이 클러스터에서 관리해야 하는 네임스페이스 및 사용자 설치된 operator 수에 따라 다릅니다. OOM이 종료되지 않도록 컨트를 플레인 노드의 크기를 적절하게 조정해야 합니다. 다음 데이터 지점은 클러스터 최대값 테스트 결과를 기반으로 합니다.

네임스페이스 수	유휴 상태의 OLM 메모리(GB)	5명의 사용자 operator가 설치된 OLM 메모리(GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6

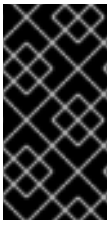


중요

다음 구성에서만 실행 중인 **OpenShift Container Platform 4.9** 클러스터에서 컨트롤 플레인 노드 크기를 수정할 수 있습니다.

- 사용자 프로비저닝 설치 방법으로 설치된 클러스터입니다.
- 설치 관리자 프로비저닝 인프라 설치 방법으로 설치된 **AWS** 클러스터입니다.

다른 모든 구성의 경우 총 노드 수를 추정하고 설치 중에 제안된 컨트롤 플레인 노드 크기를 사용해야 합니다.



중요

권장 사항은 **OpenShift SDN**이 있는 **OpenShift Container Platform** 클러스터에서 네트워크 플러그인으로 캡처된 데이터 포인트를 기반으로 합니다.

참고

OpenShift Container Platform 3.11 및 이전 버전과 비교하면, **OpenShift Container Platform 4.9**에서는 기본적으로 **CPU** 코어의 절반(**500**밀리코어)이 시스템에 의해 예약되어 있습니다. 이러한 점을 고려하여 크기가 결정됩니다.

5.4.4. CPU 관리자 설정

프로세스

1. 선택사항: 노드에 레이블을 지정합니다.

```
# oc label node perf-node.example.com cpumanager=true
```

2. **CPU** 관리자를 활성화해야 하는 노드의 **MachineConfigPool**을 편집합니다. 이 예에서는 모든 작업자의 **CPU** 관리자가 활성화됩니다.

```
# oc edit machineconfigpool worker
```

3. 작업자 머신 구성 풀에 레이블을 추가합니다.

```
metadata:
  creationTimestamp: 2020-xx-xxx
  generation: 3
  labels:
    custom-kubelet: cpumanager-enabled
```

4. KubeletConfig, cpumanager-kubeletconfig.yaml, CR(사용자 정의 리소스)을 생성합니다. 이전 단계에서 생성한 레이블을 참조하여 올바른 노드가 새 kubelet 구성으로 업데이트되도록 합니다. machineConfigPoolSelector 섹션을 참조하십시오.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s 2
```

1 정책을 지정합니다.

- none. 이 정책은 기존 기본 CPU 선호도 체계를 명시적으로 활성화하여 스케줄러가 자동으로 수행하는 것 이상으로 선호도를 제공하지 않도록 합니다. 이는 기본 정책입니다.
- static. 이 정책을 사용하면 정수 CPU 요청이 있는 보장된 Pod의 컨테이너를 사용할 수 있습니다. 또한 노드의 전용 CPU에 대한 액세스 권한을 제한합니다. 정 적인 경우 소문자 s 를 사용해야 합니다.

2 선택사항입니다. CPU 관리자 조정 빈도를 지정합니다. 기본값은 5s입니다.

5. 동적 kubelet 구성을 생성합니다.

```
# oc create -f cpumanager-kubeletconfig.yaml
```

그러면 kubelet 구성에 CPU 관리자 기능이 추가되고 필요한 경우 MCO(Machine Config Operator)가 노드를 재부팅합니다. CPU 관리자를 활성화하는 데는 재부팅이 필요하지 않습니다.

6. 병합된 kubelet 구성을 확인합니다.

```
# oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json |
grep ownerReference -A7
```

출력 예

```
    "ownerReferences": [
        {
            "apiVersion": "machineconfiguration.openshift.io/v1",
            "kind": "KubeletConfig",
            "name": "cpumanager-enabled",
            "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
        }
    ]
```

7. 작업자에서 업데이트된 kubelet.conf를 확인합니다.

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

출력 예

```
cpuManagerPolicy: static      1
cpuManagerReconcilePeriod: 5s 2
```

1. cpuManagerPolicy 는 KubeletConfig CR을 생성할 때 정의됩니다.

2

8. 코어를 하나 이상 요청하는 **Pod**를 생성합니다. 제한 및 요청 둘 다 해당 **CPU** 값이 정수로 설정되어야 합니다. 해당 숫자는 이 **Pod** 전용으로 사용할 코어 수입니다.

```
# cat cpumanager-pod.yaml
```

출력 예

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
  nodeSelector:
    cpumanager: "true"
```

9. **Pod**를 생성합니다.

```
# oc create -f cpumanager-pod.yaml
```

10. 레이블 지정한 노드에 **Pod**가 예약되어 있는지 검증합니다.

```
# oc describe pod cpumanager
```

출력 예

```
Name:              cpumanager-6cqz7
Namespace:         default
Priority:          0
PriorityClassName:  <none>
Node:  perf-node.example.com/xxx.xx.xx.xxx
...
 Limits:
     cpu:     1
     memory:  1G
   Requests:
     cpu:       1
     memory:    1G
...
QoS Class:       Guaranteed
Node-Selectors:  cpumanager=true
```

11. cgroups가 올바르게 설정되었는지 검증합니다. pause 프로세스의 PID(프로세스 ID)를 가져옵니다.

```
# ├─init.scope
  │ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
  └─kubepods.slice
    ├─kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
    │ ├─crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
    │ └─32706 /pause
```

QoS(Quality of Service) 계층 Guaranteed의 Pod는 kubepods.slice에 있습니다. 다른 QoS 계층의 Pod는 kubepods의 하위 cgroups에 있습니다.

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done
```

출력 예

```
cpuset.cpus 1
tasks 32706
```

12. 작업에 허용되는 **CPU** 목록을 확인합니다.

```
# grep ^Cpus_allowed_list /proc/32706/status
```

출력 예

```
Cpus_allowed_list:    1
```

13. **Guaranteed Pod**용으로 할당된 코어에서는 시스템의 다른 **Pod**(이 경우 **burstable QoS** 계층의 **Pod**)를 실행할 수 없는지 검증합니다.

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-besteffort.slice/kubepods-
besteffort-podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.
cpus
0
# oc describe node perf-node.example.com
```

출력 예

```
...
Capacity:
 attachable-volumes-aws-ebs:  39
 cpu:                         2
 ephemeral-storage:           124768236Ki
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      8162900Ki
 pods:                        250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                         1500m
 ephemeral-storage:           124768236Ki
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      7548500Ki
 pods:                        250
-------                       ----                       ------------  ----------  ---------------  -------------  --
-
 default                      cpumanager-6cqz7           1 (66%)       1 (66%)     1G (12%)
1G (12%)      29m
```

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                    Requests         Limits
  --------                    --------         ------
  cpu                         1440m (96%)      1 (66%)
```

이 VM에는 두 개의 CPU 코어가 있습니다. system-reserved 설정은 500밀리코어로 설정되었습니다. 즉, Node Allocatable 양이 되는 노드의 전체 용량에서 한 코어의 절반이 감산되었습니다. Allocatable CPU는 1500 밀리코어임을 확인할 수 있습니다. 즉, Pod마다 하나의 전체 코어를 사용하므로 CPU 관리자 Pod 중 하나를 실행할 수 있습니다. 전체 코어는 1000밀리코어에 해당합니다. 두 번째 Pod를 예약하려고 하면 시스템에서 해당 Pod를 수락하지만 Pod가 예약되지 않습니다.

```
NAME                    READY   STATUS    RESTARTS   AGE
cpumanager-6cqz7        1/1     Running   0          33m
cpumanager-7qc2t        0/1     Pending   0          11s
```

5.5. HUGE PAGE

Huge Page를 이해하고 구성합니다.

5.5.1. Huge Page의 기능

메모리는 페이지라는 블록으로 관리됩니다. 대부분의 시스템에서 한 페이지는 4Ki입니다. 1Mi 메모리는 256페이지와 같고 1Gi 메모리는 256,000페이지에 해당합니다. CPU에는 하드웨어에서 이러한 페이지 목록을 관리하는 내장 메모리 관리 장치가 있습니다. TLB(Translation Lookaside Buffer)는 가상-물리적 페이지 매핑에 대한 소규모 하드웨어 캐시입니다. TLB에 하드웨어 명령어로 전달된 가상 주소가 있으면 매핑을 신속하게 확인할 수 있습니다. 가상 주소가 없으면 TLB 누락이 발생하고 시스템에서 소프트웨어 기반 주소 변환 속도가 느려져 성능 문제가 발생합니다. TLB 크기는 고정되어 있으므로 TLB 누락 가능성을 줄이는 유일한 방법은 페이지 크기를 늘리는 것입니다.

대규모 페이지는 4Ki보다 큰 메모리 페이지입니다. x86_64 아키텍처에서 일반적인 대규모 페이지 크기는 2Mi와 1Gi입니다. 다른 아키텍처에서는 크기가 달라집니다. 대규모 페이지를 사용하려면 애플리케이션이 인식할 수 있도록 코드를 작성해야 합니다. THP(투명한 대규모 페이지)에서는 애플리케이션 지식 없이 대규모 페이지 관리를 자동화하려고 하지만 한계가 있습니다. 특히 페이지 크기 2Mi로 제한됩니다. THP에서는 THP 조각 모음 작업으로 인해 메모리 사용률이 높아지거나 조각화가 발생하여 노드에서 성능이 저하될 수 있으며 이로 인해 메모리 페이지가 잠길 수 있습니다. 이러한 이유로 일부 애플리케이션은 THP 대신 사전 할당된 Huge Page를 사용하도록 설계 (또는 권장)할 수 있습니다.

5.5.2. 애플리케이션이 Huge Page를 소비하는 방법

노드에서 대규모 페이지 용량을 보고하려면 노드가 대규모 페이지를 사전 할당해야 합니다. 노드는 단일 크기의 대규모 페이지만 사전 할당할 수 있습니다.

대규모 페이지는 **hugepages-<size>** 리소스 이름으로 컨테이너 수준 리소스 요구사항에 따라 사용할 수 있습니다. 여기서 크기는 특정 노드에서 지원되는 정수 값이 사용된 가장 간단한 바이너리 표현입니다. 예를 들어 노드에서 **2,048KiB** 페이지 크기를 지원하는 경우 예약 가능한 리소스 **hugepages-2Mi**를 공개합니다. **CPU** 또는 메모리와 달리 대규모 페이지는 초과 커밋을 지원하지 않습니다.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        hugepages-2Mi: 100Mi (1)
        memory: "1Gi"
        cpu: "1"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1)

hugepages의 메모리 양은 할당할 정확한 양으로 지정하십시오. 이 값을 **hugepages**의 메모리 양과 페이지 크기를 곱한 값으로 지정하지 마십시오. 예를 들어 대규모 페이지 크기가 **2MB**이고 애플리케이션에 **100MB**의 대규모 페이지 지원 **RAM**을 사용하려면 **50**개의 대규모 페이지를 할당합니다. **OpenShift Container Platform**에서 해당 계산을 처리합니다. 위의 예에서와 같이 **100MB**를 직접 지정할 수 있습니다.

특정 크기의 대규모 페이지 할당

일부 플랫폼에서는 여러 대규모 페이지 크기를 지원합니다. 특정 크기의 대규모 페이지를 할당하려면 대규모 페이지 부팅 명령 매개변수 앞에 대규모 페이지 크기 선택 매개변수 **hugepagesz=<size>**를 지정합니다. **<size>** 값은 바이트 단위로 지정해야 하며 스케일링 접미사 **[kKmMgG]**를 선택적으로 사용할 수

있습니다. 기본 대규모 페이지 크기는 **default_hugepagesz=<size>** 부팅 매개변수로 정의할 수 있습니다.

대규모 페이지 요구사항

- 대규모 페이지 요청은 제한과 같아야 합니다. 제한은 지정되었으나 요청은 지정되지 않은 경우 제한이 기본값입니다.
- 대규모 페이지는 **Pod** 범위에서 격리됩니다. 컨테이너 격리는 향후 반복에서 계획됩니다.
- 대규모 페이지에서 지원하는 **EmptyDir** 볼륨은 **Pod** 요청보다 더 많은 대규모 페이지 메모리를 사용하면 안 됩니다.
- **SHM_HUGETLB**로 **shmget()**를 통해 대규모 페이지를 사용하는 애플리케이션은 ***proc/sys/vm/hugetlb_shm_group***과 일치하는 보조 그룹을 사용하여 실행되어야 합니다.

5.5.3. 대규모 페이지 구성

노드는 **OpenShift Container Platform** 클러스터에서 사용되는 대규모 페이지를 사전 할당해야 합니다. 대규모 페이지 예약은 부팅 시 예약하는 방법과 런타임 시 예약하는 방법 두 가지가 있습니다. 부팅 시 예약은 메모리가 아직 많이 조각화되어 있지 않으므로 성공할 가능성이 높습니다. **Node Tuning Operator**는 현재 특정 노드에서 대규모 페이지에 대한 부팅 시 할당을 지원합니다.

5.5.3.1. 부팅 시

프로세스

노드 재부팅을 최소화하려면 다음 단계를 순서대로 수행해야 합니다.

1. 동일한 대규모 페이지 설정이 필요한 모든 노드에 하나의 레이블을 지정합니다.

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. 다음 콘텐츠로 파일을 생성하고 이름을 **hugepages-tuned-boottime.yaml**로 지정합니다.

```
apiVersion: tuned.openshift.io/v1
```

```
kind: Tuned
metadata:
  name: hugepages (1)
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: (2)
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 (3)
    name: openshift-node-hugepages

  recommend:
  - machineConfigLabels: (4)
      machineconfiguration.openshift.io/role: "worker-hp"
    priority: 30
    profile: openshift-node-hugepages
```

(1) Tuned 리소스의 name을 hugepages로 설정합니다.

(2) 대규모 페이지를 할당할 profile 섹션을 설정합니다.

(3) 일부 플랫폼에서는 다양한 크기의 대규모 페이지를 지원하므로 매개변수 순서가 중요합니다.

(4) 머신 구성 풀 기반 일치를 활성화합니다.

3. Tuned hugepages 오브젝트를 생성합니다.

```
$ oc create -f hugepages-tuned-boottime.yaml
```

4. 다음 콘텐츠로 파일을 생성하고 이름을 hugepages-mcp.yaml로 지정합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
```

```
  name: worker-hp
  labels:
    worker-hp: ""
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-
hp]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-hp: ""
```

5. 머신 구성 풀을 생성합니다.

```
$ oc create -f hugepages-mcp.yaml
```

조각화되지 않은 메모리가 충분한 경우 **worker-hp** 머신 구성 풀의 모든 노드에 **50**개의 **2Mi** 대규모 페이지가 할당되어 있어야 합니다.

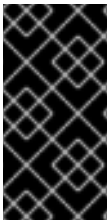
```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```

주의

이 기능은 현재 **RHCOS(Red Hat Enterprise Linux CoreOS) 8.x** 작업자 노드에서만 지원됩니다. **RHEL (Red Hat Enterprise Linux) 7.x** 작업자 노드에서는 현재 **TuneD [bootloader]** 플러그인이 지원되지 않습니다.

5.6. 장치 플러그인 이해

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



중요

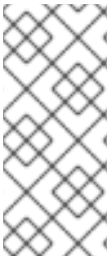
OpenShift Container Platform은 장치 플러그인 **API**를 지원하지만 장치 플러그인 컨테이너는 개별 공급 업체에서 지원합니다.

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

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

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

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

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

장치 플러그인 예

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



참고

간편한 장치 플러그인 참조 구현을 위해 장치 관리자 코드에 *vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go* 의 스텁 장치 플러그인이 있습니다.

5.6.1. 장치 플러그인을 배포하는 방법

- 장치 플러그인 배포에는 데몬 세트 접근 방식을 사용하는 것이 좋습니다.

- 시작 시 장치 플러그인은 노드의 */var/lib/kubelet/device-plugin/* 에 **UNIX** 도메인 소켓을 만들어 장치 관리자의 **RPC**를 제공하려고 합니다.

- 장치 플러그인은 하드웨어 리소스, 호스트 파일 시스템에 대한 액세스 및 소켓 생성을 관리해야 하므로 권한 있는 보안 컨텍스트에서 실행해야 합니다.

- 배포 단계에 대한 보다 구체적인 세부 정보는 각 장치 플러그인 구현에서 확인할 수 있습니다.

5.6.2. 장치 관리자 이해

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

업스트림 코드 변경없이 특수 하드웨어를 공개할 수 있습니다.

중요

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

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

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

장치 관리자는 새 등록 요청을 처리하는 동안 장치 플러그인 서비스에서 **ListAndWatch** 원격 프로시저 호출**(RPC)**을 호출합니다. 이에 대한 응답으로 장치 관리자는 플러그인에서 **gRPC** 스트림을 통해 장치 오브젝트 목록을 가져옵니다. 장치 관리자는 플러그인에서 새로운 업데이트를 위해 스트림을 모니터링합

니다. 플러그인 측에서도 플러그인은 스트림을 열린 상태로 유지하고 장치 상태가 변경될 때마다 동일한 스트리밍 연결을 통해 새 장치 목록이 장치 관리자로 전송됩니다.

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

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

5.6.3. 장치 관리자 활성화

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

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

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

   a. **Machine config**를 표시합니다:

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

   예를 들면 다음과 같습니다.

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

   출력 예

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

1

장치 관리자에 필요한 라벨입니다.

프로세스

1. 구성 변경을 위한 사용자 정의 리소스 (CR)를 만듭니다.

장치 관리자 CR의 설정 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr 2
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true 3
```

1 CR에 이름을 지정합니다.

2 Machine Config Pool에서 라벨을 입력합니다.

3 DevicePlugins를 'true'로 설정합니다.

2. 장치 관리자를 만듭니다.

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

출력 예

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

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

5.7. 테인트(TAINTS) 및 톨러레이션(TOLERATIONS)

테인트(Taints)와 톨러레이션(Tolerations)을 이해하고 사용합니다.

5.7.1. 테인트(Taints) 및 톨러레이션(Tolerations)의 이해

*테인트*를 사용하면 **Pod**에 일치하는 *허용 오차*가 없는 경우 노드에서 **Pod** 예약을 거부할 수 있습니다.

Node 사양(**NodeSpec**)을 통해 노드에 테인트를 적용하고 **Pod** 사양(**PodSpec**)을 통해 **Pod**에 허용 오차를 적용합니다. 노드에 테인트를 적용할 때 **Pod**에서 테인트를 허용할 수 없는 경우 스케줄러에서 해당 노드에 **Pod**를 배치할 수 없습니다.

노드 사양의 테인트 예

```
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
....
```

Pod 사양의 허용 오차 예

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
....
```

테인트 및 톨러레이션은 key, value 및 effect로 구성되어 있습니다.

표 5.1. 테인트 및 톨러레이션 구성 요소

매개변수	설명
key	**key**는 최대 253 자의 문자열입니다. 키는 문자 또는 숫자로 시작해야 하며 문자, 숫자, 하이픈, 점, 밑줄을 포함할 수 있습니다.
value	**value**는 최대 63 자의 문자열입니다. 값은 문자 또는 숫자로 시작해야 하며 문자, 숫자, 하이픈, 점, 밑줄을 포함할 수 있습니다.
effect	다음 명령 중 하나를 실행합니다. **NoSchedule** [1]: • 테인트에 일치하지 않는 새 pod는 해당 노드에 예약되지 않습니다. • 노드의 기존 pod는 그대로 유지됩니다. **PreferNoSchedule**: • 테인트와 일치하지 않는 새 pod는 해당 노드에 예약할 수 있지만 스케줄러는 그렇게하지 않습니다. • 노드의 기존 pod는 그대로 유지됩니다. **NoExecute**: • 테인트에 일치하지 않는 새 pod는 해당 노드에 예약할 수 없습니다. • 일치하는 톨러레이션이 없는 노드의 기존 pod는 제거됩니다.

매개변수	설명	
operator	Equal	**key**/**value**/**effect** 매개변수가 일치해야합니다. 이는 기본값입니다.
	Exists	**key**/**effect** 매개변수가 일치해야합니다. 일치하는 빈 **value** 매개변수를 남겨 두어야합니다.

1. 컨트롤 플레인 노드에 **NoSchedule** 테인트를 추가하는 경우 노드에 기본적으로 추가되는 **node-role.kubernetes.io/master=:NoSchedule** 테인트가 있어야 합니다.

   예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
...
```

톨러레이션은 테인트와 일치합니다.

- **operator** 매개변수가 **Equal**로 설정된 경우:
  - **key** 매개변수는 동일합니다.
  - **value** 매개변수는 동일합니다.

- effect 매개변수는 동일합니다.

- operator 매개변수가 Exists로 설정된 경우:
  - key 매개변수는 동일합니다.
  - effect 매개변수는 동일합니다.

다음 테인트는 OpenShift Container Platform에 빌드됩니다.

- node.kubernetes.io/not-ready: 노드가 준비 상태에 있지 않습니다. 이는 노드 조건 Ready=False에 해당합니다.
- node.kubernetes.io/unreachable: 노드가 노드 컨트롤러에서 연결할 수 없습니다. 이는 노드 조건 Ready=Unknown에 해당합니다.
- node.kubernetes.io/memory-pressure: 노드에 메모리 부족 문제가 있습니다. 이는 노드 조건 MemoryPressure=True에 해당합니다.
- node.kubernetes.io/disk-pressure: 노드에 디스크 부족 문제가 있습니다. 이는 노드 조건 DiskPressure=True에 해당합니다.
- node.kubernetes.io/network-unavailable: 노드 네트워크를 사용할 수 없습니다.
- node.kubernetes.io/unschedulable: 노드를 예약할 수 없습니다.
- node.cloudprovider.kubernetes.io/uninitialized: 노드 컨트롤러가 외부 클라우드 공급자로 시작되면 이 테인트 노드에 사용 불가능으로 표시됩니다. cloud-controller-manager의 컨트롤러가 이 노드를 초기화하면 kubelet이 이 테인트를 제거합니다.
- node.kubernetes.io/pid-pressure: 노드에 pid 부족이 있습니다. 이는 노드 조건 PIDPressure=True 에 해당합니다.

중요

OpenShift Container Platform은 기본 **pid.available evictionHard** 를 설정하지 않습니다.

5.7.1.1. tolerationSeconds를 사용하여 pod 제거를 지연하는 방법

Pod 사양 또는 **MachineSet** 오브젝트에 **tolerationSeconds** 매개변수를 지정하면 **Pod**를 제거하기 전에 노드에 바인딩되는 시간을 지정할 수 있습니다. **NoExecute** 효과가 있는 테인트가 **tolerationSeconds** 매개변수가 있는 테인트를 허용하는 **Pod**인 노드에 추가되면 해당 기간이 만료될 때까지 **Pod**가 제거되지 않습니다.

출력 예

```
spec:
 tolerations:
 - key: "key1"
   operator: "Equal"
   value: "value1"
   effect: "NoExecute"
   tolerationSeconds: 3600
```

여기에서 이 **Pod**가 실행 중이지만 일치하는 허용 오차가 없으면 **Pod**는 **3,600**초 동안 노드에 바인딩된 후 제거됩니다. 이 시간 이전에 테인트가 제거되면 **pod**가 제거되지 않습니다.

5.7.1.2. 여러 테인트를 사용하는 방법

동일한 노드에 여러 테인트를 배치하고 동일한 **pod**에 여러 톨러레이션을 배치할 수 있습니다. **OpenShift Container Platform**은 다음과 같이 여러 테인트 및 톨러레이션을 처리합니다.

1. **Pod**에 일치하는 톨러레이션이 있는 테인트를 처리합니다.
2. 나머지 일치하지 테인트는 **pod**에서 다음 **effect**를 갖습니다.

- effect가 NoSchedule인 일치하지 않는 테인트가 하나 이상있는 경우 OpenShift Container Platform은 해당 노드에 pod를 예약할 수 없습니다.

- effect가 NoSchedule인 일치하지 않는 테인트가 없지만 effect가 PreferNoSchedule인 일치하지 않는 테인트가 하나 이상있는 경우, OpenShift 컨테이너 플랫폼은 노드에 pod를 예약 시도하지 않습니다.

- 효과가 NoExecute인 일치하지 않는 테인트가 하나 이상 있는 경우 OpenShift Container Platform은 Pod가 노드에서 이미 실행되고 있으면 노드에서 Pod를 제거합니다. Pod가 노드에서 아직 실행되고 있지 않으면 Pod가 노드에 예약되지 않습니다.

  - 테인트를 허용하지 Pod는 즉시 제거됩니다.

  - Pod 사양에 tolerationSeconds를 지정하지 않은 테인트를 허용하는 Pod는 영구적으로 바인딩된 상태를 유지합니다.

  - tolerationSeconds가 지정된 테인트를 허용하는 Pod는 지정된 시간 동안 바인딩된 상태로 유지됩니다.

예를 들면 다음과 같습니다.

- 노드에 다음 테인트를 추가합니다.

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- Pod에는 다음과 같은 톨러레이션이 있습니다.

```
spec:
 tolerations:
 - key: "key1"
   operator: "Equal"
   value: "value1"
   effect: "NoSchedule"
```

```
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

이 경우 세 번째 테인트와 일치하는 톨러레이션이 없기 때문에 **pod**를 노드에 예약할 수 없습니다. 세 번째 테인트는 **pod**에서 허용되지 않는 세 번째 테인트 중 하나이기 때문에 테인트가 추가될 때 노드에서 이미 실행되고 있는 경우 **pod**가 계속 실행됩니다.

5.7.1.3. Pod 예약 및 노드 상태 (taint node by condition)

상태별 노드 테인트 기능은 기본적으로 활성화되어 있으며 메모리 부족 및 디스크 부족과 같은 상태를 보고하는 노드를 자동으로 테인트합니다. 노드가 상태를 보고하면 상태가 해제될 때까지 테인트가 추가됩니다. 테인트에는 **NoSchedule effect**가 있습니다. 즉, **pod**에 일치하는 톨러레이션이 없으면 노드에서 **pod**를 예약할 수 없습니다.

스케줄러는 **pod**를 예약하기 전에 노드에서 이러한 테인트를 확인합니다. 테인트가 있는 경우 **pod**는 다른 노드에 예약됩니다. 스케줄러는 실제 노드 상태가 아닌 테인트를 확인하기 때문에 적절한 **pod** 톨러레이션을 추가하여 이러한 노드 상태 중 일부를 무시하도록 스케줄러를 구성합니다.

이전 버전과의 호환성을 보장하기 위해 데몬 세트 컨트롤러는 모든 데몬에 다음과 같은 허용 오차를 자동으로 추가합니다.

- **node.kubernetes.io/memory-pressure**
- **node.kubernetes.io/disk-pressure**
- **node.kubernetes.io/unschedulable (1.10** 이상**)**
- **node.kubernetes.io/network-unavailable (**호스트 네트워크 만**)**

데몬 세트에 임의의 허용 오차를 추가할 수 있습니다.

참고

컨트롤 플레인은 **QoS** 클래스가 있는 **Pod**에 **node.kubernetes.io/memory-pressure** 허용 오차를 추가합니다. 이는 **Kubernetes**가 **Guaranteed** 또는 **Burstable QoS** 클래스에서 **Pod**를 관리하기 때문입니다. 새 **BestEffort Pod**가 영향을 받는 노드에 예약되지 않습니다.

5.7.1.4. 상태 별 pod 제거 (taint-based evictions)

기본적으로 활성화된 **Taint-Based Evictions** 기능은 **not-ready** 및 **unreachable**과 같은 특정 상태에 있는 노드에서 **Pod**를 제거합니다. 노드에 이러한 상태 중 하나가 발생하면 **OpenShift Container Platform**은 자동으로 노드에 테인트를 추가하고 **pod**를 제거하여 다른 도드에서 다시 예약하기 시작합니다.

Taint Based Evictions에는 **NoExecute** 효과가 있으며, 여기서 테인트를 허용하지 않는 **Pod**는 즉시 제거되고 테인트를 허용하는 모든 **Pod**는 **tolerationSeconds** 매개변수를 사용하지 않는 한 제거되지 않습니다.

tolerationSeconds 매개변수를 사용하면 노드 조건이 설정된 노드에 **Pod**가 바인딩되는 기간을 지정할 수 있습니다. **tolerationSeconds** 기간 후에도 이 상태가 계속되면 테인트가 노드에 남아 있고 허용 오차가 일치하는 **Pod**가 제거됩니다. **tolerationSeconds** 기간 전에 상태 조건이 지워지면 허용 오차가 일치하는 **Pod**가 제거되지 않습니다.

값이 없는 **tolerationSeconds** 매개변수를 사용하는 경우 준비되지 않고 연결할 수 없는 노드 상태로 인해 **Pod**가 제거되지 않습니다.

참고

OpenShift Container Platform은 속도가 제한된 방식으로 **pod**를 제거하여 마스터가 노드에서 분할되는 등의 시나리오에서 대규모 **pod** 제거를 방지합니다.

기본적으로 지정된 영역에서 노드의 **55%** 이상이 비정상이면 노드 라이프사이클 컨트롤러는 해당 영역의 상태를 **PartialDisruption** 으로 변경하고 **Pod** 제거 속도가 줄어듭니다. 이 상태에서 소규모 클러스터(기본값 **50**개 이하)의 경우 이 영역의 노드가 테인트되지 않고 제거가 중지됩니다.

자세한 내용은 **Kubernetes** 문서 의 제거 요금 제한을 참조하십시오.

Pod 구성에서 허용 오차를 지정하지 않는 경우 OpenShift Container Platform은 자동으로 node.kubernetes.io/not-ready 및 node.kubernetes.io/unreachable의 허용 오차를 tolerationSeconds=300으로 추가합니다.

```
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300 1
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```

1

이러한 톨러레이션은 이러한 노드 상태 문제 중 하나가 감지된 후 기본 pod 동작을 5 분 동안 바인딩된 상태로 유지할 수 있도록 합니다.

필요에 따라 이러한 톨러레이션을 구성할 수 있습니다. 예를 들어 애플리케이션에 다수의 로컬 상태가 있는 경우 네트워크 파티션 등에 따라 pod를 노드에 더 오래 바인딩하여 파티션을 복구하고 pod 제거를 방지할 수 있습니다.

데몬 세트에 의해 생성된 Pod는 tolerationSeconds가 없는 다음 테인트의 NoExecute 허용 오차를 사용하여 생성됩니다.

- node.kubernetes.io/unreachable
- node.kubernetes.io/not-ready

결과적으로 이러한 노드 상태로 인해 데몬 세트 Pod가 제거되지 않습니다.

5.7.1.5. 모든 테인트 허용

key 및 value 매개변수 없이 operator: "Exists" 허용 오차를 추가하여 모든 테인트를 허용하도록 Pod를 구성할 수 있습니다. 이 허용 오차가 있는 Pod는 테인트가 있는 노드에서 제거되지 않습니다.

모든 테인트를 허용하는 Pod 사양

```
spec:
  tolerations:
  - operator: "Exists"
```

5.7.2. 테인트 및 톨러레이션 추가

Pod에 허용 오차를 추가하고 노드에 테인트를 추가하면 노드에 예약하거나 예약하지 않아야 하는 **Pod**를 노드에서 제어할 수 있습니다. 기존 **Pod** 및 노드의 경우 먼저 **Pod**에 허용 오차를 추가한 다음 노드에 테인트를 추가하여 허용 오차를 추가하기 전에 노드에서 **Pod**가 제거되지 않도록 합니다.

프로세스

1. **tolerations** 스탠자를 포함하도록 **Pod** 사양을 편집하여 **Pod**에 허용 오차를 추가합니다.

Equal 연산자가 있는 **Pod** 구성 파일 샘플

```
spec:
  tolerations:
  - key: "key1" (1)
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 (2)
```

(1) 테인트 및 허용 오차 구성 요소 테이블에 설명된 허용 오차 매개변수입니다.

(2) **tolerationSeconds** 매개변수를 지정하여 **pod**가 제거되기 전까지 노드에 바인딩되는 시간을 설정합니다.

예를 들면 다음과 같습니다.

Exists 연산자가 있는 Pod 구성 파일 샘플

```
spec:
  tolerations:
   - key: "key1"
     operator: "Exists" (1)
     effect: "NoExecute"
     tolerationSeconds: 3600
```

(1) Exists 연산자는 value를 사용하지 않습니다.

이 예에서는 key key1, value value1, 테인트 effect NoExecute를 갖는 node1에 테인트를 배치합니다.

2. 테인트 및 허용 오차 구성 요소 테이블에 설명된 매개변수로 다음 명령을 사용하여 노드에 테인트를 추가합니다.

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

이 명령은 키가 key1, 값이 value1, 효과가 NoExecute인 node1에 테인트를 배치합니다.

참고

컨트롤 플레인 노드에 **NoSchedule** 테인트를 추가하는 경우 노드에 기본적으로 추가되는 **node-role.kubernetes.io/master=:NoSchedule** 테인트가 있어야 합니다.

예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
...
```

Pod의 허용 오차가 노드의 테인트와 일치합니다. 허용 오차 중 하나가 있는 **Pod**를 **node1**에 예약할 수 있습니다.

5.7.3. 머신 세트를 사용하여 테인트 및 허용 오차 추가

머신 세트를 사용하여 노드에 테인트를 추가할 수 있습니다. **MachineSet** 오브젝트와 연결된 모든 노드는 테인트를 사용하여 업데이트됩니다. 허용 오차는 노드에 직접 추가된 테인트와 동일한 방식으로 머신 세트에 의해 추가된 테인트에 응답합니다.

프로세스

1. **tolerations** 스탠자를 포함하도록 **Pod** 사양을 편집하여 **Pod**에 허용 오차를 추가합니다.

Equal 연산자가 있는 **Pod** 구성 파일의 예

```
spec:
  tolerations:
  - key: "key1" 1
```

```
  value: "value1"
  operator: "Equal"
  effect: "NoExecute"
  tolerationSeconds: 3600 2
```

1

테인트 및 허용 오차 구성 요소 테이블에 설명된 허용 오차 매개변수입니다.

2

tolerationSeconds 매개변수는 **Pod**가 제거될 때까지 노드에 바인딩되는 시간을 지정합니다.

예를 들면 다음과 같습니다.

Exists 연산자가 있는 **pod** 구성 파일의 예

```
spec:
 tolerations:
 - key: "key1"
   operator: "Exists"
   effect: "NoExecute"
   tolerationSeconds: 3600
```

2. **MachineSet** 오브젝트에 테인트를 추가합니다.

   a. 테인트할 노드의 **MachineSet YAML**을 편집하거나 새 **MachineSet** 오브젝트를 생성할 수 있습니다.

   ```
   $ oc edit machineset <machineset>
   ```

   b. **spec.template.spec** 섹션에 테인트를 추가합니다.

머신 세트 사양의 테인트 예

```
spec:
....
  template:
....
    spec:
      taints:
      - effect: NoExecute
        key: key1
        value: value1
....
```

이 예제에서는 키가 **key1**, 값이 **value1**, 테인트 효과가 **NoExecute**인 테인트를 노드에 배치합니다.

c. 머신 세트를 **0** 으로 축소합니다.

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

작은 정보

다음 **YAML**을 적용하여 머신 세트를 확장할 수도 있습니다.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

머신이 제거될 때까지 기다립니다.

d. 필요에 따라 머신 세트를 확장합니다.

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

또는 다음을 수행합니다.

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

머신이 시작될 때까지 기다립니다. 테인트는 **MachineSet** 오브젝트와 연결된 노드에 추가됩니다.

5.7.4. 테인트 및 톨러레이션을 사용하여 사용자를 노드에 바인딩

특정 사용자 집합에서 독점적으로 사용하도록 노드 세트를 전용으로 지정하려면 해당 **Pod**에 허용 오차를 추가합니다. 그런 다음 해당 노드에 해당 테인트를 추가합니다. 허용 오차가 있는 **Pod**는 테인트된 노드 또는 클러스터의 다른 노드를 사용할 수 있습니다.

이렇게 테인트된 노드에만 **Pod**를 예약하려면 동일한 노드 세트에도 라벨을 추가하고 해당 라벨이 있는 노드에만 **Pod**를 예약할 수 있도록 **Pod**에 노드 유사성을 추가합니다.

프로세스

사용자가 해당 노드 만 사용할 수 있도록 노드를 구성하려면 다음을 수행합니다.

1. 해당 노드에 해당 테인트를 추가합니다.

   예를 들면 다음과 같습니다.

   ```
   $ oc adm taint nodes node1 dedicated=groupName:NoSchedule
   ```

작은 정보

다음 YAML을 적용하여 테인트를 추가할 수도 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
```

2. 사용자 정의 승인 컨트롤러를 작성하여 Pod에 허용 오차를 추가합니다.

5.7.5. 테인트 및 톨러레이션을 사용하여 특수 하드웨어로 노드 제어

소규모 노드 하위 집합에 특수 하드웨어가 있는 클러스터에서는 테인트 및 허용 오차를 사용하여 특수 하드웨어가 필요하지 않은 Pod를 해당 노드에서 분리하여 특수 하드웨어가 필요한 Pod를 위해 노드를 남겨 둘 수 있습니다. 또한 특정 노드를 사용하기 위해 특수 하드웨어가 필요한 Pod를 요청할 수도 있습니다.

이 작업은 특수 하드웨어가 필요한 Pod에 허용 오차를 추가하고 특수 하드웨어가 있는 노드를 테인트하여 수행할 수 있습니다.

프로세스

특수 하드웨어가 있는 노드를 특정 Pod용으로 예약하려면 다음을 수행합니다.

1. 특수 하드웨어가 필요한 Pod에 허용 오차를 추가합니다.

   예를 들면 다음과 같습니다.

```
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
```

```
    operator: "Equal"
    effect: "NoSchedule"
    tolerationSeconds: 3600
```

2. 다음 명령 중 하나를 사용하여 특수 하드웨어가 있는 노드에 테인트를 설정합니다.

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

또는 다음을 수행합니다.

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

작은 정보

다음 YAML을 적용하여 테인트를 추가할 수도 있습니다.

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
```

5.7.6. 테인트 및 톨러레이션 제거

필요에 따라 노드에서 테인트를 제거하고 Pod에서 톨러레이션을 제거할 수 있습니다. 허용 오차를 추가하려면 먼저 Pod에 허용 오차를 추가한 다음 노드에서 Pod가 제거되지 않도록 노드에 테인트를 추가해야 합니다.

프로세스

테인트 및 톨러레이션을 제거하려면 다음을 수행합니다.

1. 노드에서 테인트를 제거하려면 다음을 수행합니다.

```
$ oc adm taint nodes <node-name> <key>-
```

예를 들면 다음과 같습니다.

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

출력 예

```
node/ip-10-0-132-248.ec2.internal untainted
```

2. Pod에서 Pod 사양을 편집하여 톨러레이션을 제거합니다.

```
spec:
  tolerations:
  - key: "key2"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

5.8. 토폴로지 관리자

토폴로지 관리자를 이해하고 사용합니다.

5.8.1. 토폴로지 관리자 정책

토폴로지 관리자는 CPU 관리자 및 장치 관리자와 같은 힌트 공급자로부터 토폴로지 힌트를 수집하고 수집된 힌트로 Pod 리소스를 정렬하는 방법으로 모든 QoS(Quality of Service) 클래스의 Pod 리소스를 정렬합니다.

토폴로지 관리자는 cpumanager-enabled CR(사용자 정의 리소스)에서 할당하는 데 다음 4가지 할당 정책을 지원합니다.

none 정책

기본 정책으로, 토폴로지 정렬을 수행하지 않습니다.

best-effort 정책

best-effort 토폴로지 관리 정책을 사용하는 **Pod**의 각 컨테이너에서는 **kubelet**이 각 힌트 공급자를 호출하여 해당 리소스 가용성을 검색합니다. 토폴로지 관리자는 이 정보를 사용하여 해당 컨테이너의 기본 **NUMA** 노드 선호도를 저장합니다. 선호도를 기본 설정하지 않으면 토폴로지 관리자가 해당 정보를 저장하고 노드에 대해 **Pod**를 허용합니다.

restricted 정책

restricted 토폴로지 관리 정책을 사용하는 **Pod**의 각 컨테이너에서는 **kubelet**이 각 힌트 공급자를 호출하여 해당 리소스 가용성을 검색합니다. 토폴로지 관리자는 이 정보를 사용하여 해당 컨테이너의 기본 **NUMA** 노드 선호도를 저장합니다. 선호도를 기본 설정하지 않으면 토폴로지 관리자가 노드에서 이 **Pod**를 거부합니다. 그러면 **Pod**는 **Terminated** 상태가 되고 **Pod** 허용 실패가 발생합니다.

single-numa-node 정책

single-numa-node 토폴로지 관리 정책을 사용하는 **Pod**의 각 컨테이너에서는 **kubelet**이 각 힌트 공급자를 호출하여 해당 리소스 가용성을 검색합니다. 토폴로지 관리자는 이 정보를 사용하여 단일 **NUMA** 노드 선호도가 가능한지 여부를 결정합니다. 가능한 경우 노드에 대해 **Pod**가 허용됩니다. 단일 **NUMA** 노드 선호도가 가능하지 않은 경우 토폴로지 관리자가 노드에서 **Pod**를 거부합니다. 그러면 **Pod**는 **Terminated** 상태가 되고 **Pod** 허용 실패가 발생합니다.

5.8.2. 토폴로지 관리자 설정

토폴로지 관리자를 사용하려면 **cpumanager-enabled CR**(사용자 정의 리소스)에서 할당 정책을 구성해야 합니다. **CPU** 관리자를 설정한 경우 해당 파일이 존재할 수 있습니다. 파일이 없으면 파일을 생성할 수 있습니다.

전제 조건

- **CPU** 관리자 정책을 **static**으로 구성하십시오.

절차

토폴로지 관리자를 활성화하려면 다음을 수행합니다.

1. **cpumanager-enabled CR**(사용자 정의 리소스)에서 토폴로지 관리자 할당 정책을 구성합니다.

```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
```

```
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node 2
```

1

이 매개변수는 소문자 **s** 가 있는 **static** 이어야 합니다.

2

선택한 토폴로지 관리자 할당 정책을 지정합니다. 여기서는 정책이 **single-numa-node**입니다. 사용할 수 있는 값은 **default, best-effort, restricted, single-numa-node**입니다.

5.8.3. Pod와 토폴로지 관리자 정책 간의 상호 작용

아래 **Pod** 사양의 예는 **Pod**와 토폴로지 관리자 간 상호 작용을 보여주는 데 도움이 됩니다.

다음 **Pod**는 리소스 요청 또는 제한이 지정되어 있지 않기 때문에 **BestEffort QoS** 클래스에서 실행됩니다.

```
spec:
  containers:
  - name: nginx
    image: nginx
```

다음 **Pod**는 요청이 제한보다 작기 때문에 **Burstable QoS** 클래스에서 실행됩니다.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

선택한 정책이 **none**이 아니면 토폴로지 관리자는 이러한 **Pod** 사양 중 하나를 고려하지 않습니다.

아래 마지막 예의 **Pod**는 요청이 제한과 동일하기 때문에 **Guaranteed QoS** 클래스에서 실행됩니다.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

토폴로지 관리자는 이러한 **Pod**를 고려합니다. 토폴로지 관리자는 포드에 대한 토폴로지 힌트 힌트를 얻으려면 **CPU** 관리자 및 장치 관리자인 힌트 공급자를 참조합니다.

토폴로지 관리자는 이 정보를 사용하여 이 컨테이너에 가장 적합한 토폴로지를 저장합니다. 이 **Pod**의 경우 **CPU** 관리자와 장치 관리자는 리소스 할당 단계에서 이러한 저장된 정보를 사용합니다.

5.9. 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.

5.10. CLUSTER RESOURCE OVERRIDE OPERATOR를 사용한 클러스터 수준 오버 커밋

Cluster Resource Override Operator는 클러스터의 모든 노드에서 오버 커밋 수준을 제어하고 컨테이너 밀도를 관리할 수 있는 승인Webhook입니다. Operator는 특정 프로젝트의 노드가 정의된 메모리 및 CPU 한계를 초과하는 경우에 대해 제어합니다.

다음 섹션에 설명된대로 OpenShift Container Platform 콘솔 또는 CLI를 사용하여 Cluster Resource Override Operator를 설치해야합니다. 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 instance이어야 합니다.

(2)

선택 사항입니다. 컨테이너 메모리 제한이 지정되어 있거나 기본값으로 설정된 경우 메모리 요청이 제한 백분율 (1-100)로 덮어 쓰기됩니다. 기본값은 50입니다.

(3)

선택 사항입니다. 컨테이너 CPU 제한이 지정되어 있거나 기본값으로 설정된 경우 CPU 요청이 1-100 사이의 제한 백분율로 덮어 쓰기됩니다. 기본값은 25입니다.

(4)

선택 사항입니다. 컨테이너 메모리 제한이 지정되어 있거나 기본값으로 설정된 경우, CPU 제한이 지정되어 있는 경우 메모리 제한의 백분율로 덮어 쓰기됩니다. 1Gi의 RAM을 100 %로 스케일링하는 것은 1 개의 CPU 코어와 같습니다. CPU 요청을 재정의하기 전에 처리됩니다 (설정된 경우). 기본값은 200입니다.

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 CR을 감시하고 ClusterResourceOverride 승인 Webhook가 operator와 동일한 네임 스페이스에 설치되어 있는지 확인합니다.

5.10.1. 웹 콘솔을 사용하여 Cluster Resource Override Operator 설치

OpenShift Container Platform 웹 콘솔을 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

OpenShift Container Platform 웹 콘솔을 사용하여 Cluster Resource Override Operator를 설치합니다.

1. OpenShift Container Platform 웹 콘솔에서 Home → Projects로 이동합니다.
   a. Create Project를 클릭합니다.
   b. clusterresourceoverride-operator를 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** 상세 페이지에서 **Create Instance**를 클릭합니다.

b. **Create ClusterResourceOverride** 페이지에서 **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)

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/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1

ClusterResourceOverride 승인 Webhook 참조

5.10.2. CLI를 사용하여 Cluster Resource Override Operator 설치

OpenShift Container Platform CLI를 사용하여 Cluster Resource Override Operator를 설치하면 클러스터의 오버 커밋을 제어할 수 있습니다.

사전 요구 사항

- 컨테이너에 제한이 설정되어 있지 않은 경우 Cluster Resource Override Operator에 영향을 주지 않습니다. 덮어쓰기를 적용하려면 LimitRange 오브젝트를 사용하여 프로젝트의 기본 제한을 지정하거나 Pod 사양에 제한을 구성해야 합니다.

프로세스

CLI를 사용하여 Cluster Resource Override 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: "4.9"
  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

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/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1

ClusterResourceOverride 승인 Webhook 참조

5.10.3. 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.

5.11. 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.

5.11.1. 컴퓨팅 리소스 및 컨테이너 이해

컴퓨팅 리소스에 대한 노드 적용 동작은 리소스 유형에 따라 다릅니다.

5.11.1.1. 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.

5.11.1.2. 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 메모리보다 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.

5.11.2. 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 커밋*됩니다.

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 우선순위가 감소하는 세 가지 QoS 클래스 중 하나로 지정됩니다.

표 5.2. QoS (Quality of Service) 클래스

우선 순위	클래스 이름	설명
1 (가장 높음)	Guaranteed	모든 리소스에 대해 제한 및 요청(선택 사항)이 설정되어 있고(0이 아님) 동일한 경우 Pod는 Guaranteed 로 분류됩니다.
2	Burstable	모든 리소스에 대해 요청 및 제한(선택 사항)이 설정되어 있고(0이 아님) 동일하지 않은 경우 Pod는 Burstable 로 분류됩니다.
3 (가장 낮음)	BestEffort	리소스에 대한 요청 및 제한이 설정되지 않은 경우 Pod는 BestEffort 로 분류됩니다.

메모리는 압축할 수 없는 리소스이므로 메모리가 부족한 경우 우선 순위가 가장 낮은 컨테이너가 먼저 종료됩니다.

- Guaranteed 컨테이너는 우선 순위가 가장 높은 컨테이너로 간주되며 제한을 초과하거나 시스템의 메모리가 부족하고 제거할 수 있는 우선 순위가 낮은 컨테이너가 없는 경우에만 종료됩니다.
- 시스템 메모리 부족 상태에 있는 Burstable 컨테이너는 제한을 초과하고 다른 BestEffort 컨테이너가 없으면 종료될 수 있습니다.
- BestEffort 컨테이너는 우선 순위가 가장 낮은 컨테이너로 처리됩니다. 시스템에 메모리가 부족한 경우 이러한 컨테이너의 프로세스가 먼저 종료됩니다.

5.11.2.1. Quality of Service (QoS) 계층에서 메모리 예약 방법

qos-reserved 매개변수를 사용하여 특정 QoS 수준에서 pod에 예약된 메모리의 백분율을 지정할 수 있습니다. 이 기능은 요청된 리소스를 예약하여 하위 OoS 클래스의 pod가 고급 QoS 클래스의 pod에서 요청한 리소스를 사용하지 못하도록 합니다.

OpenShift Container Platform은 다음과 같이 qos-reserved 매개변수를 사용합니다.

- qos-reserved=memory=100% 값은 Burstable 및 BestEffort QoS 클래스가 더 높은 QoS 클래스에서 요청한 메모리를 소비하지 못하도록 합니다. 이를 통해 BestEffort 및 Burstable 워크로드에서 OOM이 발생할 위험이 증가되어 Guaranteed 및 Burstable 워크로드에 대한 메모리 리소스의 보장 수준을 높이는 것이 우선됩니다.

- qos-reserved=memory=50% 값은 Burstable 및 BestEffort QoS 클래스가 더 높은 QoS 클래스에서 요청한 메모리의 절반을 소비하는 것을 허용합니다.

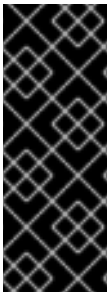
- qos-reserved=memory=0% 값은 Burstable 및 BestEffort QoS 클래스가 사용 가능한 경우 할당 가능한 최대 노드 양까지 소비하는 것을 허용하지만 Guaranteed 워크로드가 요청된 메모리에 액세스하지 못할 위험이 높아집니다. 이로 인해 이 기능은 비활성화되어 있습니다.

5.11.3. 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 pod가 메모리 제한에 도달하면 각 컨테이너가 스왑 메모리를 사용할 수 있습니다. 결국 스왑 공간이 충분하지 않으면 시스템의 초과 구독으로 인해 Pod의 프로세스가 종료될 수 있습니다.

스왑을 비활성화하지 못하면 노드에서 MemoryPressure가 발생하고 있음을 인식하지 못하여 Pod가 스케줄링 요청에서 만든 메모리를 받지 못하게 됩니다. 결과적으로 메모리 Pod를 추가로 늘리기 위해 추가 Pod가 노드에 배치되어 궁극적으로 시스템 메모리 부족 (OOM) 이벤트가 발생할 위험이 높아집니다.



중요

스왑이 활성화되면 사용 가능한 메모리에 대한 리소스 부족 처리 제거 임계 값이 예상대로 작동하지 않을 수 있습니다. 리소스 부족 처리를 활용하여 메모리 부족 상태에서 Pod를 노드에서 제거하고 메모리 부족 상태가 아닌 다른 노드에서 일정을 재조정할 수 있도록 합니다.

5.11.4. 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 확인하기 위해 OpenShift Container Platform은 vm.overcommit_memory 매개변수를 1로 설정하여 기본 운영 체제 설정을 재정의하여 커널이 항상 메모리를 오버 커밋하도록 구성합니다.

OpenShift Container Platform은 vm.panic_on_oom 매개변수를 0으로 설정하여 메모리 부족시 커널이 패닉 상태가되지 않도록 구성합니다. 0으로 설정하면 커널에서 OOM (메모리 부족) 상태일 때 oom_killer를 호출하여 우선 순위에 따라 프로세스를 종료합니다.

노드에서 다음 명령을 실행하여 현재 설정을 볼 수 있습니다.

```
$ sysctl -a |grep commit
```

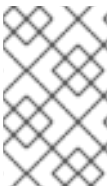
출력 예

```
vm.overcommit_memory = 1
```

```
$ 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

5.11.5. 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

1. 다음 명령을 입력하여 구성할 노드 유형의 정적 MachineConfigPool CRD와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" (1)
  name: worker
```

(1)

레이블이 **Labels** 아래에 표시됩니다.

작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

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

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: (3)
      - "true"
```

1. CR에 이름을 지정합니다.
2. 머신 구성 풀에서 라벨을 지정합니다.
3. cpuCfsQuota 매개변수를 true 로 설정합니다.

2. 다음 명령을 실행하여 CR을 생성합니다.

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

5.11.6. 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.

5.11.7. 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
```

5.12. 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.

5.12.1. 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. 프로젝트의 오브젝트 파일 편집합니다.
2. 다음 주석을 추가합니다.

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

3. 프로젝트 오브젝트를 생성합니다.

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

5.13. 가비지 컬렉션을 사용하여 노드 리소스 해제

가비지 컬렉션을 이해하고 사용합니다.

5.13.1. 가비지 컬렉션을 통해 종료된 컨테이너를 제거하는 방법

컨테이너 가비지 컬렉션은 제거 임계 값을 사용하여 수행할 수 있습니다.

가비지 컬렉션에 제거 임계 값이 설정되어 있으면 노드는 **API**에서 액세스 가능한 모든 **pod**의 컨테이너를 유지하려고합니다. **pod**가 삭제된 경우 컨테이너도 삭제됩니다. **pod**가 삭제되지 않고 제거 임계 값에 도달하지 않는 한 컨테이너는 보존됩니다. 노드가 디스크 부족 **(disk pressure)** 상태가 되면 컨테이너가 삭제되고 **oc logs**를 사용하여 해당 로그에 더 이상 액세스할 수 없습니다.

- **eviction-soft -** 소프트 제거 임계 값은 관리자가 지정한 필수 유예 기간이 있는 제거 임계 값과 일치합니다.
- **eviction-hard -** 하드 제거 임계 값에 대한 유예 기간이 없으며 감지되는 경우 **OpenShift Container Platform**은 즉시 작업을 수행합니다.

다음 표에는 제거 임계값이 나열되어 있습니다.

표 5.3. 컨테이너 가비지 컬렉션을 구성하는 변수

노드 상태	제거 신호	설명
MemoryPressure	**memory.available**	노드에서 사용 가능한 메모리입니다.
DiskPressure	• **nodefs.available** • **nodefs.inodesFree** • **imagefs.available** • **imagefs.inodesFree**	노드 루트 파일 시스템 nodefs 또는 이미지 파일 시스템 image **fs** 에서 사용 가능한 디스크 공간 또는 inode **.**

참고

evictionHard 의 경우 이러한 모든 매개변수를 지정해야 합니다. 모든 매개변수를 지정하지 않으면 지정된 매개변수만 적용되고 가비지 컬렉션이 제대로 작동하지 않습니다.

노드가 소프트 제거 임계 값 상한과 하한 사이에서 변동하고 연관된 유예 기간이 만료되지 않은 경우 해당 노드는 지속적으로 **true** 와 **false** 사이에서 변동합니다. 결과적으로 스케줄러는 잘못된 스케줄링 결정을 내릴 수 있습니다.

이러한 변동을 방지하려면 **eviction-pressure-transition-period** 플래그를 사용하여 **OpenShift Container Platform**이 부족 상태에서 전환하기 전에 대기해야하는 시간을 제어합니다. **OpenShift Container Platform**은 **false** 상태로 전환되기 전에 지정된 기간에 지정된 부족 상태에 대해 제거 임계 값을 충족하도록 설정하지 않습니다.

5.13.2. 가비지 컬렉션을 통해 이미지가 제거되는 방법 이해

이미지 가비지 컬렉션은 노드에서 **cAdvisor**에 의해 보고된 디스크 사용량에 따라 노드에서 제거할 이미지를 결정합니다.

이미지 가비지 컬렉션 정책은 다음 두 가지 조건을 기반으로합니다.

- 이미지 가비지 컬렉션을 트리거하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 **85**입니다.
- 이미지 가비지 컬렉션이 해제하려고 하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 **80**입니다.

이미지 가비지 컬렉션의 경우 사용자 지정 리소스를 사용하여 다음 변수를 수정할 수 있습니다.

표 **5.4.** 이미지 가비지 컬렉션 구성을 위한 변수

설정	설명
imageMinimumGCAge	가비지 컬렉션에 의해 이미지가 제거되기 전에 사용되지 않은 이미지의 최소 보존 기간입니다. 기본값은 **2m**입니다.
imageGCHighThresholdPercent	이미지 가비지 컬렉션을 트리거하는 정수로 표시되는 디스크 사용량의 백분율입니다. 기본값은 **85**입니다.
imageGCLowThresholdPercent	이미지 가비지 컬렉션이 해제하려고하는 디스크 사용량의 백분율 (정수로 표시)입니다. 기본값은 **80**입니다.

각 가비지 컬렉터 실행으로 두 개의 이미지 목록이 검색됩니다.

1. 하나 이상의 **Pod**에서 현재 실행중인 이미지 목록입니다.

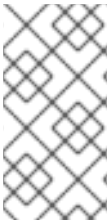
2. 호스트에서 사용 가능한 이미지 목록입니다.

새로운 컨테이너가 실행되면 새로운 이미지가 나타납니다. 모든 이미지에는 타임 스탬프가 표시됩니다. 이미지가 실행 중이거나 (위의 첫 번째 목록) 새로 감지된 경우 (위의 두 번째 목록) 현재 시간으로 표시됩니다. 나머지 이미지는 이미 이전 실행에서 표시됩니다. 모든 이미지는 타임 스탬프별로 정렬됩니다.

컬렉션이 시작되면 중지 기준이 충족될 때까지 가장 오래된 이미지가 먼저 삭제됩니다.

5.13.3. 컨테이너 및 이미지의 가비지 컬렉션 구성

관리자는 각 **machine config pool**마다 **kubeletConfig** 오브젝트를 생성하여 **OpenShift Container Platform**이 가비지 컬렉션을 수행하는 방법을 구성할 수 있습니다.



참고

OpenShift Container Platform은 각 머신 구성 풀에 대해 하나의 **kubeletConfig** 오브젝트만 지원합니다.

다음 중 하나의 조합을 구성할 수 있습니다.

- 소프트 컨테이너 제거
- 하드 컨테이너 제거
- 이미지 제거

사전 요구 사항

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

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```

1

레이블이 **Labels** 아래에 표시됩니다.

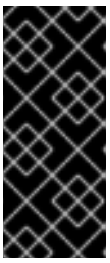
작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

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

프로세스

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



중요

하나의 파일 시스템이 있거나 **/var/lib/kubelet** 및 **/var/lib/containers/** 가 동일한 파일 시스템에 있는 경우 값이 가장 높은 설정이 먼저 충족되므로 제거를 트리거합니다. 파일 시스템이 제거를 트리거합니다.

컨테이너 가비지 컬렉션 **CR**의 설정 예:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: worker-kubeconfig (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    evictionSoft: (3)
      memory.available: "500Mi" (4)
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: (5)
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard: (6)
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s (7)
    imageMinimumGCAge: 5m (8)
    imageGCHighThresholdPercent: 80 (9)
    imageGCLowThresholdPercent: 75 (10)
```

(1) 오브젝트의 이름입니다.

(2) 머신 구성 풀에서 라벨을 지정합니다.

(3) 제거 유형: evictionSoft 또는 evictionHard.

4

특정 제거 트리거 신호에 따른 제거 임계값입니다.

5

소프트 제거의 유예 기간입니다. 이 매개변수는 **eviction-hard**에는 적용되지 않습니다.

6

특정 제거 트리거 신호에 따른 제거 임계값입니다. **evictionHard** 의 경우 이러한 모든 매개변수를 지정해야 합니다. 모든 매개변수를 지정하지 않으면 지정된 매개변수만 적용되고 가비지 컬렉션이 제대로 작동하지 않습니다.

7

제거 부족 상태에서 전환하기 전에 대기하는 시간입니다.

8

가비지 컬렉션에 의해 이미지가 제거되기 전에 사용되지 않은 이미지의 최소 보존 기간입니다.

9

이미지 가비지 컬렉션을 트리거하는 디스크 사용량의 백분율 (정수로 표시)입니다.

10

이미지 가비지 컬렉션이 해제하려고 하는 디스크 사용량의 백분율 (정수로 표시)입니다.

2. 다음 명령을 실행하여 **CR**을 생성합니다.

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

예를 들면 다음과 같습니다.

```
$ oc create -f gc-container.yaml
```

출력 예

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

검증

1. 다음 명령을 입력하여 가비지 컬렉션이 활성화되어 있는지 확인합니다. 사용자 지정 리소스에 지정한 **Machine Config Pool**은 변경 사항이 완전히 구현될 때까지 **UPDATING**과 함께 **'true'**로 표시됩니다.

```
$ oc get machineconfigpool
```

출력 예

```
NAME    CONFIG                                UPDATED   UPDATING
master  rendered-master-546383f80705bd5aeaba93  True      False
worker  rendered-worker-b4c51bb33ccaae6fc4a6a5  False     True
```

5.14. NODE TUNING OPERATOR 사용

Node Tuning Operator를 이해하고 사용합니다.

Node Tuning Operator는 **TuneD** 데몬을 오케스트레이션하여 노드 수준 튜닝을 관리하는 데 도움이 됩니다. 대부분의 고성능 애플리케이션에는 일정 수준의 커널 튜닝이 필요합니다. **Node Tuning Operator**는 노드 수준 **sysctls** 사용자에게 통합 관리 인터페이스를 제공하며 사용자의 필요에 따라 지정되는 사용자 정의 튜닝을 추가할 수 있는 유연성을 제공합니다.

Operator는 **OpenShift Container Platform**의 컨테이너화된 **TuneD** 데몬을 **Kubernetes** 데몬 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.

버전 4.1 이상에서는 Node Tuning Operator가 표준 OpenShift Container Platform 설치에 포함되어 있습니다.

5.14.1. Node Tuning Operator 사양 예에 액세스

이 프로세스를 사용하여 Node Tuning Operator 사양 예에 액세스하십시오.

프로세스

1. 다음을 실행합니다.

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

기본 CR은 OpenShift Container Platform 플랫폼의 표준 노드 수준 튜닝을 제공하기 위한 것이며 Operator 관리 상태를 설정하는 경우에만 수정할 수 있습니다. Operator는 기본 CR에 대한 다른 모든 사용자 정의 변경사항을 덮어씁니다. 사용자 정의 튜닝의 경우 고유한 Tuned CR을 생성합니다. 새로 생성된 CR은 노드 또는 Pod 라벨 및 프로필 우선 순위에 따라 OpenShift Container Platform 노드에 적용된 기본 CR 및 사용자 정의 튜닝과 결합됩니다.

주의

특정 상황에서는 Pod 라벨에 대한 지원이 필요한 튜닝을 자동으로 제공하는 편리한 방법일 수 있지만 이러한 방법은 권장되지 않으며 특히 대규모 클러스터에서는 이러한 방법을 사용하지 않는 것이 좋습니다. 기본 Tuned CR은 Pod 라벨이 일치되지 않은 상태로 제공됩니다. Pod 라벨이 일치된 상태로 사용자 정의 프로필이 생성되면 해당 시점에 이 기능이 활성화됩니다. Pod 라벨 기능은 Node Tuning Operator의 향후 버전에서 더 이상 사용되지 않을 수 있습니다.

5.14.2. 사용자 정의 튜닝 사양

Operator의 CR(사용자 정의 리소스)에는 두 가지 주요 섹션이 있습니다. 첫 번째 섹션인 profile:은 TuneD 프로필 및 해당 이름의 목록입니다. 두 번째인 recommend:은 프로필 선택 논리를 정의합니다.

여러 사용자 정의 튜닝 사양은 Operator의 네임스페이스에 여러 CR로 존재할 수 있습니다. 새로운 CR의 존재 또는 오래된 CR의 삭제는 Operator에서 탐지됩니다. 기존의 모든 사용자 정의 튜닝 사양이 병합

되고 컨테이너화된 TuneD 데몬의 해당 오브젝트가 업데이트됩니다.

관리 상태

Operator 관리 상태는 기본 Tuned CR을 조정하여 설정됩니다. 기본적으로 Operator는 Managed 상태이며 기본 Tuned CR에는 spec.managementState 필드가 없습니다. Operator 관리 상태에 유효한 값은 다음과 같습니다.

- Managed: 구성 리소스가 업데이트되면 Operator가 해당 피연산자를 업데이트합니다.
- Unmanaged: Operator가 구성 리소스에 대한 변경을 무시합니다.
- Removed: Operator가 프로비저닝한 해당 피연산자 및 리소스를 Operator가 제거합니다.

프로필 데이터

profile: 섹션에는 TuneD 프로필 및 해당 이름이 나열됩니다.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

권장 프로필

profile: 선택 논리는 **CR**의 **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)
```

(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_1이 있습니다.

7

선택적 피연산자 구성입니다.

8

TuneD 데몬의 디버깅을 켜거나 끕니다. off의 경우 on 또는 false 의 경우 옵션이 true 입니다. 기본값은 false입니다.

<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 **machineConfigLabels**는 논리 **OR** 연산자로 연결됩니다. **match** 항목은 단락 방식으로 먼저 평가됩니다. 따라서 **true**로 평가되면 **machineConfigLabels** 항목이 고려되지 않습니다.

중요

머신 구성 풀 기반 일치를 사용하는 경우 동일한 하드웨어 구성을 가진 노드를 동일한 머신 구성 풀로 그룹화하는 것이 좋습니다. 이 방법을 따르지 않으면 **TuneD** 피연산자가 동일한 머신 구성 풀을 공유하는 두 개 이상의 노드에 대해 충돌하는 커널 매개변수를 계산할 수 있습니다.

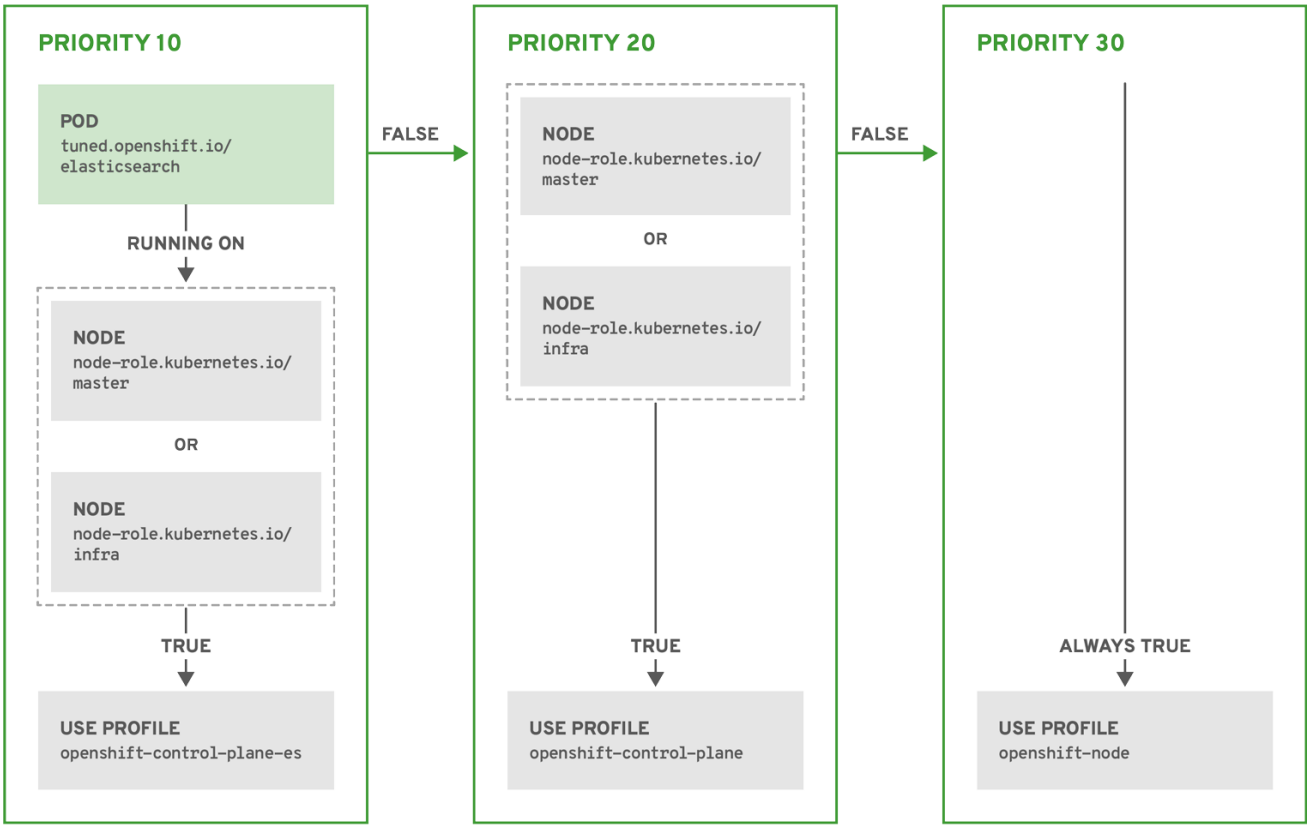
예: 노드 또는 Pod 라벨 기반 일치

```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

위의 **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 10인 프로필의 라벨이 일치하면 **openshift-control-plane-es** 프로필이 적용되고 다른 프로필은 고려되지 않습니다. 노드/Pod 라벨 조합이 일치하지 않으면 두 번째로 높은 우선 순위 프로필 (**openshift-control-plane**)이 고려됩니다. 컨테이너화된 TuneD Pod가 **node-role.kubernetes.io/master** 또는 **node-role.kubernetes.io/infra**. 라벨이 있는 노드에서 실행되는 경우 이 프로필이 적용됩니다.

마지막으로, **openshift-node** 프로필은 우선 순위가 가장 낮은 30입니다. 이 프로필에는 **<match>** 섹션이 없으므로 항상 일치합니다. 지정된 노드에서 우선 순위가 더 높은 다른 프로필이 일치하지 않는 경우 **openshift-node** 프로필을 설정하는 데 **catch-all** 프로필 역할을 합니다.



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.

5.14.3. 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:
  recommend:
  - profile: "openshift-control-plane"
    priority: 30
    match:
    - label: "node-role.kubernetes.io/master"
    - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node"
    priority: 40
```

OpenShift Container Platform 4.9부터 모든 **OpenShift TuneD** 프로필이 **TuneD** 패키지와 함께 제공됩니다. **oc exec** 명령을 사용하여 이러한 프로필의 내용을 볼 수 있습니다.

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find
/usr/lib/tuned/openshift{,-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

5.14.4. 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

일부 플러그인에서 제공하는 동적 튜닝 기능은 지원되지 않습니다. 다음 TuneD 플러그인은 현재 지원되지 않습니다.

- bootloader
- script
- systemd

자세한 내용은 사용 가능한 TuneD 플러그인 및 TuneD 시작하기 를 참조하십시오.

5.15. 노드 당 최대 POD 수 구성

podsPerCore 및 maxPods는 노드에 예약할 수 있는 최대 Pod 수를 제어합니다. 두 옵션을 모두 사용하는 경우 두 옵션 중 더 낮은 값이 노드의 Pod 수를 제한합니다.

예를 들어 4 개의 프로세서 코어가 있는 노드에서 podsPerCore가 10으로 설정된 경우 노드에서 허용되는 최대 Pod 수는 40입니다.

사전 요구 사항

1. 다음 명령을 입력하여 구성할 노드 유형의 정적 MachineConfigPool CRD와 연관된 라벨을 가져옵니다.

```
$ oc edit machineconfigpool <name>
```

예를 들면 다음과 같습니다.

```
$ oc edit machineconfigpool worker
```

출력 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```

1

레이블이 **Labels** 아래에 표시됩니다.

작은 정보

라벨이 없으면 다음과 같은 키/값 쌍을 추가합니다.

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

프로세스

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

max-pods CR의 설정 예

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    podsPerCore: 10 3
    maxPods: 250 4
```

1. CR에 이름을 지정합니다.

2. 머신 구성 풀에서 라벨을 지정합니다.

3. 노드의 프로세서 코어 수에 따라 노드가 실행할 수있는 Pod 수를 지정합니다.

4. 노드의 속성에 관계없이 노드가 고정 값으로 실행할 수 있는 Pod 수를 지정합니다.

참고

podsPerCore를 0으로 설정하면 이 제한이 비활성화됩니다.

위의 예에서 podsPerCore의 기본값은 10이며 maxPods의 기본값은 250입니다. 즉, 노드에 25 개 이상의 코어가 없으면 기본적으로 podsPerCore가 제한 요소가 됩니다.

2. 다음 명령을 실행하여 CR을 생성합니다.

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

검증

1. MachineConfigPool CRD를 나열하여 변경 사항이 적용되는지 확인합니다. Machine Config Controller에서 변경 사항을 선택하면 UPDATING 열에 True가 보고됩니다.

```
$ oc get machineconfigpools
```

출력 예

```
NAME    CONFIG                 UPDATED   UPDATING   DEGRADED
master  master-9cc2c72f205e103bb534  False     False      False
worker  worker-8cecd1236b33ee3f8a5e  False     True       False
```

변경이 완료되면 **UPDATED** 열에 **True**가 보고됩니다.

```
$ oc get machineconfigpools
```

출력 예

```
NAME    CONFIG                 UPDATED   UPDATING   DEGRADED
master  master-9cc2c72f205e103bb534  False     True       False
worker  worker-8cecd1236b33ee3f8a5e  True      False      False
```

6장. 설치 후 네트워크 구성

OpenShift Container Platform을 설치한 후 요구 사항에 맞게 네트워크를 추가로 확장하고 사용자 지정할 수 있습니다.

6.1. CNO(CLUSTER NETWORK OPERATOR) 구성

클러스터 네트워크의 구성은 CNO(Cluster Network Operator) 구성의 일부로 지정되며 cluster라는 이름의 CR(사용자 정의 리소스) 오브젝트에 저장됩니다. CR은 operator.openshift.io API 그룹에서 Network API의 필드를 지정합니다.

CNO 구성은 Network.config.openshift.io API 그룹의 Network API에서 클러스터 설치 중에 다음 필드를 상속하며 이러한 필드는 변경할 수 없습니다.

clusterNetwork

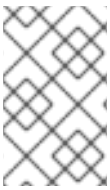
Pod IP 주소가 할당되는 IP 주소 풀입니다.

serviceNetwork

서비스를 위한 IP 주소 풀입니다.

defaultNetwork.type

OpenShift SDN 또는 OVN-Kubernetes와 같은 클러스터 네트워크 공급자입니다.



참고

클러스터를 설치한 후에는 이전 섹션에 나열된 필드를 수정할 수 없습니다.

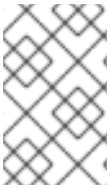
6.2. 클러스터 전체 프록시 사용

프록시 오브젝트는 클러스터 전체 송신 프록시를 관리하는 데 사용됩니다. 프록시를 구성하지 않고 클러스터를 설치하거나 업그레이드해도 프록시 오브젝트는 계속 생성되지만 spec 은 nil이 됩니다. 예를 들면 다음과 같습니다.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
```

```
  trustedCA:
    name: ""
status:
```

클러스터 관리자는 이 **cluster** 프록시 오브젝트를 수정하여 **OpenShift Container Platform**의 프록시를 구성할 수 있습니다.



참고

cluster라는 **Proxy** 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

사전 요구 사항

- 클러스터 관리자 권한
- **OpenShift Container Platform oc CLI** 도구 설치

프로세스

1. **HTTPS** 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 구성 맵을 생성합니다.

참고

프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명한 경우 이 단계를 건너뛸 수 있습니다.

a. 다음 내용으로 **user-ca-bundle.yaml**이라는 파일을 생성하고 **PEM** 인코딩 인증서 값을 제공합니다.

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1

이 데이터 키의 이름은 **ca-bundle.crt**여야 합니다.

2

프록시의 **ID** 인증서 서명에 사용되는 하나 이상의 **PEM** 인코딩 **X.509** 인증서입니다.

3

프록시 오브젝트에서 참조할 구성 맵 이름입니다.

4

구성 맵은 **openshift-config** 네임스페이스에 있어야 합니다.

b. 이 파일에서 구성 맵을 생성합니다.

```
$ oc create -f user-ca-bundle.yaml
```

2. **oc edit** 명령을 사용하여 **Proxy** 오브젝트를 수정합니다.

```
$ oc edit proxy/cluster
```

3. 프록시에 필요한 필드를 구성합니다.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1

클러스터 외부에서 **HTTP** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http**여야 합니다.

2

클러스터 외부에서 **HTTPS** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http** 또는 **https** 여야 합니다. **URL** 스키마를 지원하는 프록시의 **URL**을 지정합니다. 예를 들어 대부분의 프록시는 **https** 를 사용하도록 구성된 경우 오류를 보고하지만 **http** 만 지원합니다. 이 실패 메시지는 로그에 전파되지 않을 수 있으며 대신 네트워크 연결 실패로 표시될 수 있습니다. 클러스터에서 **https** 연결을 수신하는 프록시를 사용하는 경우 프록시에서 사용하는 **CA** 및 인증서를 허용하도록 클러스터를 구성해야 할 수 있습니다.

3

대상 도메인 이름, 도메인, **IP** 주소 또는 프록시를 제외할 기타 네트워크 **CIDR**로 이루어진 쉼표로 구분된 목록입니다.

하위 도메인과 일치하려면 도메인 앞에 **.**을 입력합니다. 예를 들어, **.y.com**은 **x.y.com**과 일치하지만 **y.com**은 일치하지 않습니다. *를 사용하여 모든 대상에 대해 프록시를 바이패스합니다. **networking.machineNetwork[].cidr** 필드에 의해 정의된 네트워크에 포함되어 있지 않은 작업자를 설치 구성에서 확장하려면 연결 문제를 방지하기 위해 이 목록에 해당 작업자를 추가해야 합니다.

httpProxy와 **httpsProxy** 필드가 모두 설정되지 않은 경우 이 필드는 무시됩니다.

4

httpProxy 및 **httpsProxy** 값을 상태에 쓰기 전에 준비 검사를 수행하는 데 사용할 하나 이상의 클러스터 외부 **URL**입니다.

5

HTTPS 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 **openshift-config** 네임스페이스의 구성 맵에 대한 참조입니다. 여기서 구성 맵을 참조하기 전에 구성 맵이 이미 있어야 합니다. 프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명하지 않은 경우 이 필드가 있어야 합니다.

4. 파일을 저장하여 변경 사항을 적용합니다.

6.3. DNS를 프라이빗으로 설정

클러스터를 배포한 후 프라이빗 영역만 사용하도록 **DNS**를 변경할 수 있습니다.

프로세스

1. 클러스터의 DNS 사용자 정의 리소스를 확인합니다.

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

출력 예

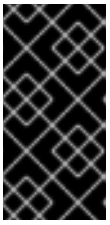
```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

spec 섹션에는 프라이빗 영역과 퍼블릭 영역이 모두 포함되어 있습니다.

2. DNS 사용자 지정 리소스를 패치하여 퍼블릭 영역을 제거합니다.

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec":
{"publicZone": null}}'
dns.config.openshift.io/cluster patched
```

Ingress 컨트롤러는 Ingress 개체를 만들 때 DNS 정의를 참조하기 때문에 Ingress 개체를 만들거나 수정할 때 프라이빗 레코드만 생성됩니다.



중요

퍼블릭 영역을 제거해도 기존 Ingress 개체의 DNS 레코드는 변경되지 않습니다.

3. 선택 사항: 클러스터의 DNS 사용자 정의 리소스를 확인하고 퍼블릭 영역이 제거되었는지 확인하십시오.

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

출력 예

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>-wfpg4: owned
status: {}
```

6.4. 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 다음 방법을 통해 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다.

- HTTP/HTTPS가 있는 경우 Ingress 컨트롤러를 사용합니다.
- HTTPS 이외의 TLS 암호화 프로토콜 (예: TLS 및 SNI 헤더의 사용 등)이 있는 경우 Ingress 컨트롤러를 사용합니다.

- 그 외에는 로드 밸런서, 외부 IP 또는 노드 포트를 사용합니다.

방법	목적
Ingress 컨트롤러 사용	HTTPS 이외의 HTTP/HTTPS 트래픽 및 TLS 암호화 프로토콜(예: TLS 및 SNI 헤더의 사용 등)에 액세스할 수 있습니다.
로드 밸런서 서비스를 사용하여 외부 IP 자동 할당	풀에서 할당된 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다.
서비스에 외부 IP를 수동으로 할당	특정 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다.
NodePort구성	클러스터의 모든 노드에 서비스를 공개합니다.

6.5. 노드 포트 서비스 범위 구성

클러스터 관리자는 사용 가능한 노드 포트 범위를 확장할 수 있습니다. 클러스터에서 많은 수의 노드 포트를 사용하는 경우 사용 가능한 포트 수를 늘려야 할 수 있습니다.

기본 포트 범위는 30000~32767입니다. 기본 범위 이상으로 확장한 경우에도 포트 범위는 축소할 수 없습니다.

6.5.1. 사전 요구 사항

- 클러스터 인프라는 확장된 범위 내에서 지정한 포트에 대한 액세스를 허용해야 합니다. 예를 들어, 노드 포트 범위를 30000~32900으로 확장하는 경우 방화벽 또는 패킷 필터링 구성에서 32768~32900의 포함 포트 범위를 허용해야 합니다.

6.5.1.1. 노드 포트 범위 확장

클러스터의 노드 포트 범위를 확장할 수 있습니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

- cluster-admin 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

1. 노드 포트 범위를 확장하려면 다음 명령을 입력합니다. <port>를 새 범위에서 가장 큰 포트 번호로 변경합니다.

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
  '{
    "spec":
      { "serviceNodePortRange": "30000-<port>" }
  }'
```

작은 정보

또는 다음 YAML을 적용하여 노드 포트 범위를 업데이트할 수 있습니다.

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

출력 예

```
network.config.openshift.io/cluster patched
```

2. 구성이 활성 상태인지 확인하려면 다음 명령을 입력합니다. 업데이트가 적용되려면 몇 분 정도 걸릴 수 있습니다.

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo "'service-node-port-range':["[[:digit:]]+-[[:digit:]]+"]'
```

출력 예

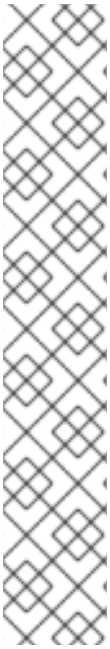
```
"service-node-port-range":["30000-33000"]
```

6.6. 네트워크 정책 구성

클러스터 관리자 또는 프로젝트 관리자는 프로젝트에 대한 네트워크 정책을 구성할 수 있습니다.

6.6.1. 네트워크 정책 정의

Kubernetes 네트워크 정책을 지원하는 **CNI(Kubernetes Container Network Interface)** 플러그인을 사용하는 클러스터에서 네트워크 격리는 **NetworkPolicy** 개체에 의해서만 제어됩니다. **OpenShift Container Platform 4.9**에서 **OpenShift SDN**은 기본 네트워크 격리 모드에서 네트워크 정책 사용을 지원합니다.



참고

OpenShift SDN 클러스터 네트워크 공급자를 사용할 경우 네트워크 정책과 관련하여 다음과 같은 제한 사항이 적용됩니다.

- 송신 필드에서 지정한 네트워크 정책 송신 은 지원되지 않습니다. 송신 방화벽은 **OpenShift SDN**에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.
- **IPBlock**은 네트워크 정책에서 지원되지만 **except** 절에는 지원되지 않습니다. **except** 절이 포함된 **IPBlock** 섹션이 포함된 정책을 생성하면 **SDN Pod** 로그가 경고를 생성하고 해당 정책의 전체 **IPBlock** 섹션이 무시됩니다.

주의

네트워크 정책은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워킹이 활성화된 **Pod**는 네트워크 정책 규칙의 영향을 받지 않습니다.

기본적으로 네트워크 정책 모드에서는 다른 **Pod** 및 네트워크 끝점에서 프로젝트의 모든 **Pod**에 액세

스할 수 있습니다. 프로젝트에서 하나 이상의 Pod를 분리하기 위해 해당 프로젝트에서 NetworkPolicy 오브젝트를 생성하여 수신되는 연결을 표시할 수 있습니다. 프로젝트 관리자는 자신의 프로젝트 내에서 NetworkPolicy 오브젝트를 만들고 삭제할 수 있습니다.

하나 이상의 NetworkPolicy 오브젝트에서 선택기와 Pod가 일치하면 Pod는 해당 NetworkPolicy 오브젝트 중 하나 이상에서 허용되는 연결만 허용합니다. NetworkPolicy 오브젝트가 선택하지 않은 Pod에 완전히 액세스할 수 있습니다.

다음 예제 NetworkPolicy 오브젝트는 다양한 시나리오 지원을 보여줍니다.

- 모든 트래픽 거부:

  기본적으로 프로젝트를 거부하려면 모든 Pod와 일치하지만 트래픽을 허용하지 않는 NetworkPolicy 오브젝트를 추가합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector: {}
  ingress: []
```

- OpenShift Container Platform Ingress 컨트롤러의 연결만 허용합니다.

  프로젝트에서 OpenShift Container Platform Ingress 컨트롤러의 연결만 허용하도록 하려면 다음 NetworkPolicy 개체를 추가합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

- 프로젝트 내 Pod 연결만 허용:

  Pod가 동일한 프로젝트 내 다른 Pod의 연결은 수락하지만 다른 프로젝트에 속하는 Pod의 기타 모든 연결을 거부하도록 하려면 다음 NetworkPolicy 오브젝트를 추가합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
```

- Pod 레이블을 기반으로 하는 HTTP 및 HTTPS 트래픽만 허용:

  특정 레이블(다음 예에서 role=frontend)을 사용하여 Pod에 대한 HTTP 및 HTTPS 액세스만 활성화하려면 다음과 유사한 NetworkPolicy 오브젝트를 추가합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- 네임스페이스와 Pod 선택기를 모두 사용하여 연결 수락:

  네임스페이스와 Pod 선택기를 결합하여 네트워크 트래픽을 일치시키려면 다음과 유사한 NetworkPolicy 오브젝트를 사용하면 됩니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
```

```
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
        podSelector:
          matchLabels:
            name: test-pods
```

NetworkPolicy 오브젝트는 추가 기능이므로 여러 NetworkPolicy 오브젝트를 결합하여 복잡한 네트워크 요구 사항을 충족할 수 있습니다.

예를 들어, 이전 샘플에서 정의된 NetworkPolicy 오브젝트의 경우 동일한 프로젝트 내에서 allow-same-namespace 정책과 allow-http-and-https 정책을 모두 정의할 수 있습니다. 따라서 레이블이 role=frontend로 지정된 Pod는 각 정책에서 허용하는 모든 연결을 허용할 수 있습니다. 즉 동일한 네임스페이스에 있는 Pod의 모든 포트 연결과 모든 네임스페이스에 있는 Pod에서 포트 80 및 443에 대한 연결이 허용됩니다.

6.6.2. NetworkPolicy 오브젝트 예

다음은 예제 NetworkPolicy 오브젝트에 대한 주석입니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

(1) NetworkPolicy 오브젝트의 이름입니다.

(2)

3

정책 오브젝트에서 수신 트래픽을 허용하는 **Pod**와 일치하는 선택기입니다. 선택기는 **NetworkPolicy**와 동일한 네임스페이스의 **Pod**와 일치합니다.

4

트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

6.6.3. 네트워크 정책 생성

클러스터의 네임스페이스에서 허용된 수신 또는 송신 네트워크 트래픽을 설명하는 세분화된 규칙을 정의하기 위해 네트워크 정책을 생성할 수 있습니다.

참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

사전 요구 사항

- 클러스터에서 **mode: NetworkPolicy**로 설정된 **OVF-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 개체를 지원하는 클러스터 네트워크 공급자를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

프로세스

1. 다음과 같이 정책 규칙을 생성합니다.
   a. **<policy_name>.yaml** 파일을 생성합니다.

```
$ touch <policy_name>.yaml
```

다음과 같습니다.

<policy_name>

네트워크 정책 파일 이름을 지정합니다.

b.

방금 만든 파일에서 다음 예와 같이 네트워크 정책을 정의합니다.

모든 네임스페이스의 모든 **Pod**에서 수신 거부

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

.동일한 네임 스페이스에 있는 모든 **Pod**의 수신 허용

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

2.

다음 명령을 실행하여 네트워크 정책 오브젝트를 생성합니다.

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

다음과 같습니다.

<policy_name>

네트워크 정책 파일 이름을 지정합니다.

<namespace>

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

출력 예

```
networkpolicy.networking.k8s.io/default-deny created
```

참고

콘솔에서 **cluster-admin** 역할을 사용하여 사용자로 로그인하는 경우 **YAML** 보기 또는 웹 콘솔의 양식에서 직접 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

6.6.4. 네트워크 정책을 사용하여 다중 테넌트 격리 구성

다른 프로젝트 네임스페이스의 **Pod** 및 서비스에서 격리하도록 프로젝트를 구성할 수 있습니다.

사전 요구 사항

- 클러스터에서 **mode: NetworkPolicy**로 설정된 **OVF-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 개체를 지원하는 클러스터 네트워크 공급자를 사용하고 있습니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

1. 다음 NetworkPolicy 오브젝트를 생성합니다.

   a. 이름이 allow-from-openshift-ingress인 정책입니다.

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

참고

policy-group.network.openshift.io/ingress: "" 는 OpenShift SDN의 기본 네임스페이스 선택기 레이블입니다. network.openshift.io/policy-group: ingress 네임스페이스 선택기 레이블을 사용할 수 있지만 이는 레거시 레이블입니다.

   b. 이름이 allow-from-openshift-monitoring인 정책:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c.

이름이 **allow-same-namespace**인 정책:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

2.

선택 사항: 현재 프로젝트에 네트워크 정책이 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc describe networkpolicy
```

출력 예

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

6.6.5. 새 프로젝트에 대한 기본 네트워크 정책 만들기

클러스터 관리자는 새 프로젝트를 만들 때 **NetworkPolicy** 오브젝트를 자동으로 포함하도록 새 프로젝트 템플릿을 수정할 수 있습니다.

6.6.6. 새 프로젝트의 템플릿 수정

클러스터 관리자는 사용자 정의 요구 사항을 사용하여 새 프로젝트를 생성하도록 기본 프로젝트 템플릿을 수정할 수 있습니다.

사용자 정의 프로젝트 템플릿을 만들려면:

프로세스

1. **cluster-admin** 권한이 있는 사용자로 로그인합니다.

2. 기본 프로젝트 템플릿을 생성합니다.

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 텍스트 편집기를 사용하여 오브젝트를 추가하거나 기존 오브젝트를 수정하여 생성된 **template.yaml** 파일을 수정합니다.

4. 프로젝트 템플릿은 **openshift-config** 네임스페이스에서 생성해야 합니다. 수정된 템플릿을 불러옵니다.

```
$ oc create -f template.yaml -n openshift-config
```

5. 웹 콘솔 또는 **CLI**를 사용하여 프로젝트 구성 리소스를 편집합니다.

- 웹 콘솔에 액세스:

i. 관리 → 클러스터 설정으로 이동합니다.

ii. Configuration(구성) 을 클릭하여 모든 구성 리소스를 확인합니다.

iii. 프로젝트 항목을 찾아 YAML 편집을 클릭합니다.

- CLI 사용:

i. 다음과 같이 project.config.openshift.io/cluster 리소스를 편집합니다.

```
$ oc edit project.config.openshift.io/cluster
```

6. projectRequestTemplate 및 name 매개변수를 포함하도록 spec 섹션을 업데이트하고 업로드된 프로젝트 템플릿의 이름을 설정합니다. 기본 이름은 project-request입니다.

사용자 정의 프로젝트 템플릿이 포함된 프로젝트 구성 리소스

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. 변경 사항을 저장한 후 새 프로젝트를 생성하여 변경 사항이 성공적으로 적용되었는지 확인합니다.

6.6.6.1. 새 프로젝트 템플릿에 네트워크 정책 추가

클러스터 관리자는 네트워크 정책을 새 프로젝트의 기본 템플릿에 추가할 수 있습니다. OpenShift Container Platform은 프로젝트의 템플릿에 지정된 모든 NetworkPolicy 개체를 자동으로 생성합니다.

사전 요구 사항

- 클러스터는 NetworkPolicy 오브젝트를 지원하는 기본 CNI 네트워크 공급자(예: mode: NetworkPolicy로 설정된 OpenShift SDN 네트워크 공급자)를 사용합니다. 이 모드는 OpenShift SDN의 기본값입니다.
- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 권한이 있는 사용자로 클러스터에 로그인해야 합니다.
- 새 프로젝트에 대한 사용자 정의 기본 프로젝트 템플릿을 생성해야 합니다.

프로세스

1. 다음 명령을 실행하여 새 프로젝트의 기본 템플릿을 편집합니다.

```
$ oc edit template <project_template> -n openshift-config
```

<project_template>을 클러스터에 대해 구성한 기본 템플릿의 이름으로 변경합니다. 기본 템플릿 이름은 project-request입니다.

2. 템플릿에서 각 NetworkPolicy 오브젝트를 objects 매개변수의 요소로 추가합니다. objects 매개변수는 하나 이상의 오브젝트 컬렉션을 허용합니다.

   다음 예제에서 objects 매개변수 컬렉션에는 여러 NetworkPolicy 오브젝트가 포함됩니다.

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
```

```
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
    - Ingress
...
```

3. 선택 사항: 다음 명령을 실행하여 새 프로젝트를 생성하고 네트워크 정책 오브젝트가 생성되었는지 확인합니다.

a. 새 프로젝트를 생성합니다.

```
$ oc new-project <project> 1
```

1 <project> 를 생성중인 프로젝트의 이름으로 변경합니다.

b. 새 프로젝트 템플릿의 네트워크 정책 오브젝트가 새 프로젝트에 있는지 확인합니다.

```
$ oc get networkpolicy
NAME                           POD-SELECTOR   AGE
allow-from-openshift-ingress   <none>         7s
allow-from-same-namespace      <none>         7s
```

6.7. 지원되는 구성

Red Hat OpenShift Service Mesh의 현재 릴리스에서는 다음 구성이 지원됩니다.

6.7.1. 지원되는 플랫폼

Red Hat OpenShift Service Mesh Operator는 ServiceMeshControlPlane 리소스의 여러 버전을 지원합니다. 버전 2.3 서비스 메시 컨트롤 플레인은 다음 플랫폼 버전에서 지원됩니다.

- Red Hat OpenShift Container Platform 버전 4.9 이상

- Red Hat OpenShift Dedicated 버전 4.
- Azure Red Hat OpenShift (ARO) 버전 4.
- Red Hat OpenShift Service on AWS(ROSA).

6.7.2. 지원되지 않는 로깅 구성

명시적으로 지원되지 않는 경우는 다음과 같습니다.

- OpenShift Online은 Red Hat OpenShift Service Mesh에서 지원되지 않습니다.
- Red Hat OpenShift Service Mesh는 Service Mesh가 실행 중인 클러스터 외부에서 마이크로 서비스 관리를 지원하지 않습니다.

6.7.3. 지원되는 네트워크 구성

Red Hat OpenShift Service Mesh는 다음과 같은 네트워크 구성을 지원합니다.

- OpenShift-SDN
- OVN-Kubernetes는 OpenShift Container Platform 4.7.32 이상, OpenShift Container Platform 4.8.12 이상 및 OpenShift Container Platform 4.9 이상에서 지원됩니다.
- OpenShift Container Platform에서 인증되었으며 Service Mesh 적합성 테스트를 통과한 타사 CNI(Container Network Interface) 플러그인입니다. 자세한 내용은 Certified OpenShift CNI 플러그인을 참조하십시오.

6.7.4. Service Mesh에 지원되는 구성

- 이번 Red Hat OpenShift Service Mesh 릴리스는 OpenShift Container Platform x86_64, IBM Z 및 IBM Power Systems에서만 사용 가능합니다.

- IBM Z는 Openshift Container Platform 4.6 이상에서만 지원됩니다.
- IBM Power Systems은 OpenShift Container Platform 4.6 이상에서만 지원됩니다.

- 모든 Service Mesh 구성 요소가 단일 OpenShift Container Platform 클러스터에 포함된 구성입니다.
- 가상 머신과 같은 외부 서비스를 통합하지 않는 구성입니다.
- Red Hat OpenShift Service Mesh는 명시적으로 문서화된 경우를 제외하고 EnvoyFilter 구성을 지원하지 않습니다.

6.7.5. Kiali에 대해 지원되는 구성

- Kiali 콘솔은 Chrome, Edge, Firefox 또는 Safari 브라우저의 두 가지 최신 릴리스에서만 지원됩니다.

6.7.6. 분산 추적에 지원되는 구성

- 사이드카로서의 Jaeger 에이전트는 Jaeger에 대해 지원되는 유일한 구성입니다. 다중 테넌트 설치 또는 OpenShift Dedicated에서는 데몬 세트로 Jaeger가 지원되지 않습니다.

6.7.7. 지원되는 WebAssembly 모듈

- 3scale WebAssembly는 제공된 유일한 WebAssembly 모듈입니다. 사용자 정의 WebAssembly 모듈을 생성할 수 있습니다.

6.7.8. Operator 개요

Red Hat OpenShift Service Mesh에는 다음과 같은 네 가지 Operator가 필요합니다.

- OpenShift Elasticsearch - (선택 사항) 분산 추적 플랫폼과의 추적 및 로깅을 위한 데이터베이스 스토리지를 제공합니다. 오픈 소스 Elasticsearch 프로젝트를 기반으로 합니다.
-

Red Hat OpenShift distributed tracing 플랫폼 - 복잡한 분산 시스템의 트랜잭션을 모니터링하고 해결하기 위해 분산 추적을 제공합니다. 오픈 소스 Jaeger 프로젝트를 기반으로 합니다.

- **Kiali** - 서비스 메시에 대한 가시성을 제공합니다. 단일 콘솔에서 구성을 보고, 트래픽을 모니터링하며 추적을 분석할 수 있습니다. 오픈 소스 Kiali 프로젝트를 기반으로 합니다.

- **Red Hat OpenShift Service Mesh** - 애플리케이션을 구성하는 마이크로 서비스를 연결, 보안, 제어 및 관찰할 수 있습니다. **Service Mesh Operator**는 **Service Mesh** 구성 요소의 배포, 업데이트 및 삭제를 관리하는 **ServiceMeshControlPlane** 리소스를 정의하고 모니터링합니다. 오픈소스 Istio 프로젝트를 기반으로 합니다.

다음 단계

- **OpenShift Container Platform** 환경에 Red Hat OpenShift Service Mesh를 설치합니다.

6.8. 라우팅 최적화

OpenShift Container Platform HAProxy 라우터는 성능을 최적화하도록 확장 또는 구성할 수 있습니다.

6.8.1. 기본 Ingress 컨트롤러(라우터) 성능

OpenShift Container Platform Ingress 컨트롤러 또는 라우터는 경로 및 인그레스를 사용하여 구성된 애플리케이션 및 서비스의 수신 트래픽의 수신 지점입니다.

초당 처리된 **HTTP** 요청 측면에서 단일 **HAProxy** 라우터 성능을 평가할 때 성능은 여러 요인에 따라 달라집니다. 특히 중요한 요인은 다음과 같습니다.

- **HTTP** 연결 유지/닫기 모드

- 경로 유형

- **TLS** 세션 재개 클라이언트 지원

- 대상 경로당 동시 연결 수
- 대상 경로 수
- 백엔드 서버 페이지 크기
- 기본 인프라(네트워크/SDN 솔루션, CPU 등)

특정 환경의 성능은 달라질 수 있으나 Red Hat 랩은 크기가 4 vCPU/16GB RAM인 퍼블릭 클라우드 인스턴스에서 테스트합니다. 1kB 정적 페이지를 제공하는 백엔드에서 종료한 100개의 경로를 처리하는 단일 HAProxy 라우터가 처리할 수 있는 초당 트랜잭션 수는 다음과 같습니다.

HTTP 연결 유지 모드 시나리오에서는 다음과 같습니다.

Encryption	LoadBalancerService	HostNetwork
none	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

HTTP 닫기(연결 유지 제외) 시나리오에서는 다음과 같습니다.

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

기본 Ingress 컨트롤러 구성은 spec.tuningOptions.threadCount 필드에서 4 로 설정된 상태에서 사용되었습니다. 두 가지 엔드 포인트 게시 전략이 테스트되었습니다: Load Balancer Service 및 Host Network. 암호화된 경로에는 TLS 세션 재개가 사용되었습니다. HTTP keep-alive를 사용하면 단일 HAProxy 라우터가 8kB의 작은 페이지 크기에서 1Gbit NIC를 포화시킬 수 있습니다.

최신 프로세서가 있는 베어 메탈에서 실행하는 경우 성능이 위 퍼블릭 클라우드 인스턴스의 약 2배가 될 것을 예상할 수 있습니다. 이 오버헤드는 퍼블릭 클라우드에서 가상화 계층에 의해 도입되며 프라이빗 클라우드 기반 가상화에도 적용됩니다. 다음 표는 라우터 뒤에서 사용할 애플리케이션 수에 대한 가이드입니다.

애플리케이션 수	애플리케이션 유형
5-10	정적 파일/웹 서버 또는 캐싱 프록시
100-1000	동적 콘텐츠를 생성하는 애플리케이션

일반적으로 HAProxy는 사용 중인 기술에 따라 최대 1000개의 애플리케이션 경로를 지원할 수 있습니다. Ingress 컨트롤러 성능은 언어 또는 정적 콘텐츠 대비 동적 콘텐츠 등 지원하는 애플리케이션의 기능과 성능에 따라 제한될 수 있습니다.

Ingress 또는 라우터 샤딩을 사용하여 애플리케이션에 대한 더 많은 경로를 제공하고 라우팅 계층을 수평으로 확장할 수 있도록 합니다.

6.9. 설치 후 RHOSP 네트워크 구성

설치 후 RHOSP(Red Hat OpenStack Platform) 클러스터에서 OpenShift Container Platform의 일부 측면을 구성할 수 있습니다.

6.9.1. 부동 IP 주소로 애플리케이션 액세스 구성

OpenShift Container Platform을 설치한 후 애플리케이션 네트워크 트래픽을 허용하도록 RHOSP(Red Hat OpenStack Platform)를 구성합니다.

참고

install-config.yaml 파일에서 platform.openstack.apiFloatingIP 및 platform.openstack.ingressFloatingIP 에 대한 값을 제공하거나 설치 중에 inventory.yaml 플레이 북의 os_api_fip 및 os_ingress_fip에 대한 값을 제공한 경우 이 절차를 수행 할 필요가 없습니다. 부동 IP 주소가 이미 설정되어 있습니다.

전제 조건

- OpenShift Container Platform 클러스터가 설치되어 있어야 합니다.
- RHOSP의 OpenShift Container Platform 설치에 관한 문서에 설명 된대로 부동 IP 주소가 활성화됩니다.

프로세스

OpenShift Container Platform 클러스터를 설치한 후 부동 IP 주소를 인그레스 포트에 연결합니다.

1. 포트 표시:

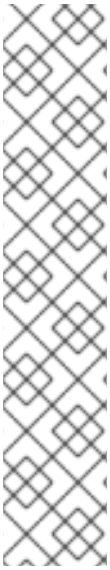
```
$ openstack port show <cluster_name>-<cluster_ID>-ingress-port
```

2. IP 주소에 포트 연결:

```
$ openstack floating ip set --port <ingress_port_ID> <apps_FIP>
```

3. *apps의 와일드카드 A 레코드를 DNS 파일에 추가:

```
*.apps.<cluster_name>.<base_domain>  IN  A  <apps_FIP>
```



참고

DNS 서버를 제어하지 않지만 프로덕션 이외의 목적으로 애플리케이션 액세스를 활성화하려는 경우 다음과 같은 호스트 이름을 /etc/hosts에 추가할 수 있습니다.

```
<apps_FIP> console-openshift-console.apps.<cluster name>.<base domain>
<apps_FIP> integrated-oauth-server-openshift-authentication.apps.<cluster
name>.<base domain>
<apps_FIP> oauth-openshift.apps.<cluster name>.<base domain>
<apps_FIP> prometheus-k8s-openshift-monitoring.apps.<cluster name>.<base
domain>
<apps_FIP> grafana-openshift-monitoring.apps.<cluster name>.<base domain>
<apps_FIP> <app name>.apps.<cluster name>.<base domain>
```

6.9.2. Kuryr 포트 풀

Kuryr 포트 풀은 Pod 생성을 위해 대기 중인 다수의 포트를 유지 관리합니다.

포트를 대기 상태로 유지하면 Pod 생성 시간이 최소화됩니다. 포트 풀이 없으면 Kuryr는 Pod를 생성하거나 삭제할 때마다 포트 생성 또는 삭제를 명시적으로 요청해야 합니다.

Kuryr가 사용하는 Neutron 포트는 네임스페이스에 연결된 서브넷에 생성됩니다. 이러한 Pod 포트도 OpenShift Container Platform 클러스터 노드의 기본 포트에 하위 포트로 추가됩니다.

Kuryr는 각 네임스페이스를 별도의 서브넷에 유지하므로 각 네임스페이스-작업자 쌍에 대해 별도의 포트 풀이 유지됩니다.

클러스터를 설치하기 전에 cluster-network-03-config.yml 매니페스트 파일에서 다음 매개변수를 설정하여 포트 풀 동작을 구성할 수 있습니다.

- enablePortPoolsPrepopulation 매개변수는 풀 사전 채우기를 제어하므로 새 호스트가 추가되거나 새 네임스페이스가 생성되는 경우와 같이 풀 생성 시 Kuryr가 풀에 포트를 추가하도록 합니다. 기본값은 false입니다.
- poolMinPorts 매개변수는 풀에 보관되는 사용 가능한 최소 포트 수입니다. 기본값은 1 입니다.
- poolMaxPorts 매개변수는 풀에 보관되는 사용 가능한 최대 포트 수입니다. 값이 0이면 해당 상한이 비활성화됩니다. 이 설정은 기본 설정입니다.

OpenStack 포트 할당량이 낮거나 pod 네트워크에 IP 주소가 제한된 경우 이 옵션을 설정하여 불필요한 포트가 삭제되었는지 확인합니다.

- poolBatchPorts 매개 변수는 한 번에 생성할 수 있는 최대 Neutron 포트 수를 정의합니다. 기본값은 3입니다.

6.9.3. RHOSP의 활성 배포에서 Kuryr 포트 풀 설정 조정

CR(사용자 정의 리소스)을 사용하여 Kuryr가 RHOSP(Red Hat OpenStack Platform) Neutron 포트를 관리하는 방법을 구성하여 배포된 클러스터에서 Pod 생성 속도와 효율성을 제어할 수 있습니다.

절차

1. 명령줄에서 편집을 위해 CNO(Cluster Network Operator) CR을 엽니다.

```
$ oc edit networks.operator.openshift.io cluster
```

2. 요구 사항에 맞게 설정을 편집합니다. 다음 파일은 예제로 제공됩니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  defaultNetwork:
    type: Kuryr
    kuryrConfig:
      enablePortPoolsPrepopulation: false (1)
      poolMinPorts: 1 (2)
      poolBatchPorts: 3 (3)
      poolMaxPorts: 5 (4)
```

(1) 네임스페이스가 생성되거나 클러스터에 새 노드를 추가한 후 Kuryr가 새 Neutron 포트를 생성하도록 하려면 enablePortPoolsPrepopulation을 true로 설정합니다. 이 설정은 Neutron 포트 할당량을 높이지만 pod를 생성하는 데 필요한 시간을 줄일 수 있습니다. 기본값은 false입니다.

2

Kuryr는 풀의 사용 가능한 포트 수가 **poolMinPorts** 값보다 낮은 경우 풀에 대한 새 포트를 만듭니다. 기본값은 **1** 입니다.

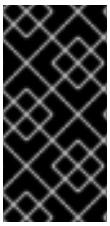
3

poolBatchPorts는 사용 가능한 포트 수가 **poolMinPorts** 값보다 낮은 경우 생성되는 새 포트 수를 제어합니다. 기본값은 **3**입니다.

4

풀에서 사용 가능한 포트 수가 **poolMaxPorts** 값보다 크면 **Kuryr**는 숫자가 해당 값과 일치할 때까지 해당 포트를 삭제합니다. 값을 **0**으로 설정하면 이 상한이 비활성화되므로 풀이 축소되지 않습니다. 기본값은 **0**입니다.

3. 변경 사항을 저장하고 텍스트 편집기를 종료하여 변경 사항을 커밋합니다.



중요

실행 중인 클러스터에서 이러한 옵션을 수정하면 **kuryr-controller** 및 **kuryr-cni Pod**가 다시 시작됩니다. 결과적으로 새 **Pod** 및 서비스 생성이 지연됩니다.

7장. 설치 후 스토리지 구성

OpenShift Container Platform을 설치한 후 스토리지 구성을 포함하여 요구 사항에 맞게 클러스터를 추가로 확장하고 사용자 정의할 수 있습니다.

7.1. 동적 프로비저닝

7.1.1. 동적 프로비저닝 소개

StorageClass 리소스 객체는 요청 가능한 스토리지를 설명하고 분류할 뿐 만 아니라 필요에 따라 동적으로 프로비저닝된 스토리지에 대한 매개 변수를 전달하는 수단을 제공합니다. **StorageClass** 객체는 다른 수준의 스토리지 및 스토리지에 대한 액세스를 제어하기 위한 관리 메커니즘으로 사용될 수 있습니다. 클러스터 관리자 **(cluster-admin)** 또는 스토리지 관리자 **(storage-admin)**는 사용자가 기본 스토리지 볼륨 소스에 대한 자세한 지식이 없어도 요청할 수 있는 **StorageClass** 오브젝트를 정의하고 생성할 수 있습니다.

OpenShift Container Platform 영구 볼륨 프레임 워크를 사용하면이 기능을 사용할 수 있으며 관리자는 영구 스토리지로 클러스터를 프로비저닝할 수 있습니다. 또한 이 프레임 워크를 통해 사용자는 기본 인프라에 대한 지식이 없어도 해당 리소스를 요청할 수 있습니다.

OpenShift Container Platform에서 많은 스토리지 유형을 영구 볼륨으로 사용할 수 있습니다. 관리자가 모두 정적으로 프로비저닝할 수 있지만 일부 스토리지 유형은 기본 제공 공급자 및 플러그인 **API**를 사용하여 동적으로 생성됩니다.

7.1.2. 사용 가능한 동적 프로비저닝 플러그인

OpenShift Container Platform에서는 다음 프로비저너 플러그인을 제공합니다. 이 플러그인에는 클러스터의 구성된 공급자 **API**를 사용하여 새 스토리지 리소스를 생성하는 동적 프로비저닝을 위한 일반 구현이 있습니다.

스토리지 유형	프로비저너 플러그인 이름	참고
Red Hat OpenStack Platform (RHOSP) Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface (CSI)	**manila.csi.openstack.org**	설치 완료되면 OpenStack Manila CSI Driver Operator 및 ManilaDriver가 동적 프로비저닝에 필요한 모든 사용 가능한 Manila 공유 유형에 필요한 스토리지 클래스를 자동으로 생성합니다.

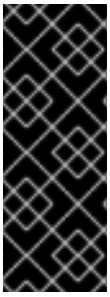
스토리지 유형	프로비저너 플러그인 이름	참고
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	다른 영역에서 여러 클러스터를 사용할 때 동적 프로비저닝의 경우 각 노드에 **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>**로 태그를 지정합니다. 여기서 **<cluster_name>** 및 **<cluster_id>**는 클러스터마다 고유합니다.
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** 서비스 계정에는 Azure 스토리지 계정 및 키를 저장할 시크릿을 생성하고 검색할 수 있는 권한이 필요합니다.
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	멀티 존 설정에서는 현재 클러스터에 노드가 없는 영역에서 PV가 생성되지 않도록 GCE 프로젝트 당 하나의 OpenShift Container Platform 클러스터를 실행하는 것이 좋습니다.
VMware vSphere	**kubernetes.io/vsphere-volume**	

중요

선택한 프로비저너 플러그인에는 관련 문서에 따라 클라우드, 호스트 또는 타사 공급자를 구성해야 합니다.

7.2. 스토리지 클래스 정의

StorageClass 객체는 현재 전역 범위 객체이며 cluster-admin 또는 storage-admin 사용자가 만들어야 합니다.



중요

클러스터 스토리지 작업자는 사용 중인 플랫폼에 따라 기본 스토리지 클래스를 설치할 수 있습니다. 이 스토리지 클래스는 **Operator**가 소유하고 제어합니다. 주석 및 레이블 정의 외에는 삭제하거나 변경할 수 없습니다. 다른 동작이 필요한 경우 사용자 정의 스토리지 클래스를 정의해야 합니다.

다음 섹션에서는 **StorageClass** 오브젝트의 기본 정의와 지원되는 각 플러그인 유형에 대한 구체적인 예를 설명합니다.

7.2.1. 기본 StorageClass 개체 정의

다음 리소스는 스토리지 클래스를 구성하는 데 사용되는 매개변수 및 기본값을 보여줍니다. 이 예에서는 **AWS ElasticBlockStore (EBS)** 객체 정의를 사용합니다.

StorageClass 정의 예

```
kind: StorageClass 1
apiVersion: storage.k8s.io/v1 2
metadata:
  name: <storage-class-name> 3
  annotations: 4
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs 5
parameters: 6
  type: gp2
...
```

1

(필수) **API** 객체 유형입니다.

2

(필수) 현재 **apiVersion**입니다.

3

(필수) 스토리지 클래스의 이름입니다.

4

(선택 사항) 스토리지 클래스의 주석입니다.

5

(필수) 이 스토리지 클래스에 연결된 프로비저너의 유형입니다.

6

(선택 사항) 특정 프로비저너에 필요한 매개 변수로, 플러그인으로 변경됩니다.

7.2.2. 스토리지 클래스 주석

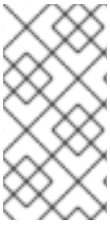
스토리지 클래스를 클러스터 전체 기본값으로 설정하려면 스토리지 클래스의 메타데이터에 다음 주석을 추가합니다.

```
storageclass.kubernetes.io/is-default-class: "true"
```

예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

이렇게 하면 특정 스토리지 클래스를 지정하지 않은 모든 PVC(영구 볼륨 클레임)가 기본 스토리지 클래스를 통해 자동으로 프로비저닝됩니다. 그러나 클러스터에는 두 개 이상의 스토리지 클래스가 있을 수 있지만, 그 중 하나만 기본 스토리지 클래스일 수 있습니다.



참고

베타 주석 storageclass.beta.kubernetes.io/is-default-class는 여전히 작동하지만 향후 릴리스에서는 제거될 예정입니다.

스토리지 클래스 설명을 설정하려면 스토리지 클래스의 메타데이터에 다음 주석을 추가합니다.

```
kubernetes.io/description: My Storage Class Description
```

예를 들면 다음과 같습니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

7.2.3. RHOSP Cinder 오브젝트 정의

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/cinder
parameters:
  type: fast 2
  availability: nova 3
  fsType: ext4 4
```

1

StorageClass의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

Cinder에서 생성 된 볼륨 유형입니다. 기본값은 비어 있습니다.

3

가용성 영역입니다. 지정하지 않으면 일반적으로 **OpenShift Container Platform** 클러스터에 노드가 있는 모든 활성 영역에서 볼륨이 라운드 로빈됩니다.

4

7.2.4. AWS Elastic Block Store (EBS) 객체 정의

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 2
  iopsPerGB: "10" 3
  encrypted: "true" 4
  kmsKeyId: keyvalue 5
  fsType: ext4 6
```

1

(필수) 스토리지 클래스의 이름입니다. 영구 볼륨 클래임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

(필수 사항)**io1, gp2, sc1, st1** 중에서 선택합니다. 기본값은 **gp2**입니다. 유효한 **Amazon Resource Name (ARN)** 값은 **AWS** 설명서를 참조하십시오.

3

선택 사항: **io1** 볼륨만 해당합니다. **GiB**마다 초당 **I/O** 작업 수입니다. **AWS** 볼륨 플러그인은 이 값과 요청된 볼륨 크기를 곱하여 볼륨의 **IOPS**를 계산합니다. 값의 상한은 **AWS**가 지원하는 최대치인 **20,000 IOPS**입니다. 자세한 내용은 **AWS** 설명서를 참조하십시오.

4

선택 사항: **EBS** 볼륨을 암호화할지 여부를 나타냅니다. 유효한 값은 **true** 또는 **false**입니다.

5

선택 사항: 볼륨을 암호화할 때 사용할 키의 전체 **ARN**입니다. 값을 지정하지 않는 경우에도 **encypted** 가 **true**로 설정되어 있는 경우 **AWS**가 키를 생성합니다. 유효한 **ARN** 값은 **AWS** 설명서를

참조하십시오.

6

선택 사항: 동적으로 프로비저닝된 볼륨에서 생성된 파일 시스템입니다. 이 값은 동적으로 프로비저닝된 영구 볼륨의 **fsType** 필드에 복사되며 볼륨이 처음 마운트될 때 파일 시스템이 작성됩니다. 기본값은 **ext4**입니다.

7.2.5. Azure Disk 오브젝트 정의

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer 2
allowVolumeExpansion: true
parameters:
  kind: Managed 3
  storageaccounttype: Premium_LRS 4
reclaimPolicy: Delete
```

1

StorageClass의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

WaitForFirstConsumer를 사용하는 것이 좋습니다. 이렇게 하면 볼륨을 프로비저닝하고 사용 가능한 영역에서 사용 가능한 작업자 노드의 **Pod**를 예약할 수 있는 충분한 스토리지를 허용합니다.

3

가능한 값은 **Shared**(기본값), **Managed**, **Dedicated**입니다.

중요

Red Hat은 스토리지 클래스에서 **kind: Managed**의 사용만 지원합니다.

Shared 및 **Dedicated**를 사용하여 **Azure**는 관리되지 않은 디스크를 생성합니다. 반면 **OpenShift Container Platform**은 머신 **OS(root)** 디스크의 관리 디스크를 생성합니다. **Azure Disk**는 노드에서 관리 및 관리되지 않은 디스크를 모두 사용하도록 허용하지 않으므로 **Shared** 또는 **Dedicated**로 생성된 관리되지 않은 디스크를 **OpenShift Container Platform** 노드에 연결할 수 없습니다.

4

Azure 스토리지 계정 **SKU**층입니다. 기본값은 비어 있습니다. 프리미엄 **VM**은 **Standard_LRS** 및 **Premium_LRS** 디스크를 모두 연결할 수 있으며 표준 **VM**은 **Standard_LRS** 디스크만 연결할 수 있습니다. 관리형 **VM**은 관리 디스크만 연결할 수 있으며 관리되지 않는 **VM**은 관리되지 않는 디스크만 연결할 수 있습니다.

a. **kind**가 **Shared**로 설정된 경우 **Azure**는 클러스터와 동일한 리소스 그룹에 있는 일부 공유 스토리지 계정에서 관리되지 않는 디스크를 만듭니다.

b. **kind**가 **Managed**로 설정된 경우 **Azure**는 새 관리 디스크를 만듭니다.

c. **kind**가 **Dedicated**로 설정되고 **storageAccount**가 지정된 경우 **Azure**는 클러스터와 동일한 리소스 그룹에서 새로운 관리되지 않는 디스크에 대해 지정된 스토리지 계정을 사용합니다. 이 기능이 작동하려면 다음 사항이 전제가 되어야 합니다.

- 지정된 스토리지 계정이 같은 지역에 있어야 합니다.
- **Azure Cloud Provider**는 스토리지 계정에 대한 쓰기 권한이 있어야 합니다.

d. **kind**가 **Dedicated**로 설정되어 있고 **storageAccount**가 지정되지 않은 경우 **Azure**는 클러스터와 동일한 리소스 그룹에 새로운 관리되지 않는 디스크에 대한 새로운 전용 스토리지 계정을 만듭니다.

7.2.6. Azure File 객체 정의

Azure File 스토리지 클래스는 시크릿을 사용하여 **Azure** 파일 공유를 만드는 데 필요한 **Azure** 스토리

지 계정 이름과 스토리지 계정 키를 저장합니다. 이러한 권한은 다음 절차의 일부로 생성됩니다.

절차

1. 시크릿을 작성하고 볼 수 있는 액세스를 허용하는 **ClusterRole** 오브젝트를 정의합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
#  name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: ['']
  resources: ['secrets']
  verbs:     ['get','create']
```

1 시크릿을 표시하고 작성하는 클러스터 역할의 이름입니다.

2. 서비스 계정에 클러스터 역할을 추가합니다.

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
system:serviceaccount:kube-system:persistent-volume-binder
```

3. **Azure File StorageClass** 오브젝트를 만듭니다.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> 1
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus 2
  skuName: Standard_LRS 3
  storageAccount: <storage-account> 4
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 **StorageClass**의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

eastus와 같은 Azure 스토리지 계정의 위치입니다. 기본값은 비어 있습니다. 즉, OpenShift Container Platform 클러스터 위치에 새 Azure 스토리지 계정이 만들어집니다.

3

Azure 스토리지 계정의 SKU 계층입니다 (예: Standard_LRS). 기본값은 비어 있습니다. 즉, Standard_LRS SKU를 사용하여 새 Azure 스토리지 계정이 만들어집니다.

4

Azure 스토리지 계정 이름입니다. 스토리지 계정이 제공되면 skuName 및 location이 무시됩니다. 스토리지 계정이 제공되지 않으면 스토리지 클래스는 정의된 skuName 및 location과 일치하는 계정의 리소스 그룹과 연관된 스토리지 계정을 검색합니다.

7.2.6.1. Azure File 사용시 고려 사항

기본 Azure File 스토리지 클래스는 다음 파일 시스템 기능을 지원하지 않습니다.

- 심볼릭 링크
- 하드 링크
- 확장 속성
- 스파스 파일
- 명명된 파이프

또한 Azure File이 마운트되는 디렉터리의 소유자 ID (UID)는 컨테이너의 프로세스 UID와 다릅니다. 마운트된 디렉터리에 사용할 특정 사용자 ID를 정의하기 위해 StorageClass 오브젝트에서 uid 마운트 옵션을 지정할 수 있습니다.

다음 StorageClass 오브젝트는 마운트된 디렉터리의 심볼릭 링크를 활성화한 상태에서 사용자 및 그룹 ID를 변경하는 방법을 보여줍니다.

-

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 마운트된 디렉터리에 사용할 사용자 ID를 지정합니다.

2 마운트된 디렉터리에 사용할 그룹 ID를 지정합니다.

3 심볼릭 링크를 활성화합니다.

7.2.7. GCE PersistentDisk (gcePD) 오브젝트 정의

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 2
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1

StorageClass의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

pd-standard 또는 pd-ssd 중 하나를 선택합니다. 기본값은 pd-standard입니다.

7.2.8. VMware vSphere 오브젝트 정의

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/vsphere-volume 2
parameters:
  diskformat: thin 3
```

1

StorageClass의 이름입니다. 영구 볼륨 클레임은 이 스토리지 클래스를 사용하여 관련 영구 볼륨을 프로비저닝합니다.

2

OpenShift Container Platform과 함께 VMware vSphere를 사용하는 방법에 대한 자세한 내용은 VMware vSphere 설명서를 참조하십시오.

3

diskformat: thin, zeroedthick 및 eagerzeroedthick은 모두 유효한 디스크 형식입니다. 디스크 형식 유형에 대한 자세한 내용은 vSphere 설명서를 참조하십시오. 기본값은 thin입니다.

7.2.9. RHV(Red Hat Virtualization) 개체 정의

OpenShift Container Platform은 동적으로 프로비저닝된 영구 볼륨을 생성하는 데 사용되는 ovirt-csi-sc라는 이름의 StorageClass 유형의 기본 오브젝트를 생성합니다.

다양한 구성을 위한 추가 스토리지 클래스를 생성하려면 다음 샘플 YAML에서 설명되는 StorageClass 오브젝트로 파일을 생성하고 저장합니다.

ovirt-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage_class_name> 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "<boolean>" 2
provisioner: csi.ovirt.org
allowVolumeExpansion: <boolean> 3
reclaimPolicy: Delete 4
volumeBindingMode: Immediate 5
parameters:
  storageDomainName: <rhv-storage-domain-name> 6
  thinProvisioning: "<boolean>" 7
  csi.storage.k8s.io/fstype: <file_system_type> 8
```

1 StorageClass의 이름입니다.

2 스토리지 클래스가 클러스터의 기본 스토리지 클래스인 경우 false로 설정합니다. true로 설정하면 기존 기본 스토리지 클래스를 편집한 후 false로 설정해야 합니다.

3 True는 동적 볼륨 확장을 활성화하고 false는 금지합니다. true가 권장됩니다.

4 이 스토리지 클래스의 동적으로 프로비저닝된 영구 볼륨은 이 회수 정책을 사용하여 생성됩니다. 기본 정책은 Delete 입니다.

5 PersistentVolumeClaims를 프로비저닝하고 바인딩하는 방법을 나타냅니다. 설정하지 않으면 VolumeBindingImmediate가 사용됩니다. 이 필드는 VolumeScheduling 기능을 활성화하는 서버

에만 적용됩니다.

6

사용할 RHV 스토리지 도메인 이름입니다.

7

true인 경우 디스크는 씬 프로비저닝됩니다. false인 경우 디스크가 사전 할당됩니다. 씬 프로비저닝이 권장됩니다.

8

선택 사항: 생성할 파일 시스템 유형입니다. 가능한 값: ext4 (기본값) 또는 xfs.

7.3. 기본 스토리지 클래스 변경

다음 프로세스를 사용하여 기본 스토리지 클래스를 변경합니다. 예를 들어 두 개의 스토리지 클래스(gp2 및 standard)가 있으며 기본 스토리지 클래스를 gp2 에서 standard로 변경하려고 합니다.

1. 스토리지 클래스를 나열합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp2 (default)        kubernetes.io/aws-ebs (1)
standard             kubernetes.io/aws-ebs
```

1

(default)는 기본 스토리지 클래스를 나타냅니다.

2. 기본 스토리지 클래스에 대해 storageclass.kubernetes.io/is-default-class 주석의 값을 false로 변경합니다.

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. storageclass.kubernetes.io/is-default-class 주석을 true로 설정하여 다른 스토리지 클래스를 기본값으로 설정합니다.

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 변경 사항을 확인합니다.

```
$ oc get storageclass
```

출력 예

```
NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```

7.4. 스토리지 최적화

스토리지를 최적화하면 모든 리소스에서 스토리지 사용을 최소화할 수 있습니다. 관리자는 스토리지를 최적화하여 기존 스토리지 리소스가 효율적으로 작동하도록 합니다.

7.5. 사용 가능한 영구 스토리지 옵션

OpenShift Container Platform 환경을 최적화할 수 있도록 영구 스토리지 옵션에 대해 알아보십시오.

표 7.1. 사용 가능한 스토리지 옵션

스토리지 유형	설명	예

스토리지 유형	설명	예
블록	• 운영 체제(OS)에 블록 장치로 제공됩니다. • 스토리지에 대한 모든 권한이 필요하며 파일 시스템을 우회하여 파일의 낮은 수준에서 작동하는 애플리케이션에 적합합니다. • SAN(Storage Area Network)이라고도 합니다. • 공유가 불가능합니다. 즉, 한 번에 하나의 클라이언트만 이 유형의 끝점을 마운트할 수 있습니다.	AWS EBS 및 VMware vSphere는 OpenShift Container Platform에서 기본적으로 동적 PV(영구 볼륨) 프로비저닝을 지원합니다.
파일	• OS에 마운트할 파일 시스템 내보내기로 제공됩니다. • NAS(Network Attached Storage)라고도 합니다. • 동시성, 대기 시간, 파일 잠금 메커니즘 및 기타 기능은 프로토콜, 구현, 벤더 및 스케일링에 따라 크게 다릅니다.	RHEL NFS, NetApp NFS [1] 및 Vendor NFS
개체	• REST API 끝점을 통해 액세스할 수 있습니다. • OpenShift Container Platform 레지스트리에서 사용하도록 구성할 수 있습니다. • 애플리케이션에서 해당 드라이버를 애플리케이션 및/또는 컨테이너에 빌드해야 합니다.	AWS S3

1. **NetApp NFS는 Trident 플러그인을 사용할 때 동적 PV 프로비저닝을 지원합니다.**

중요

OpenShift Container Platform 4.9에서는 현재 CNS가 지원되지 않습니다.

7.6. 권장되는 구성 가능한 스토리지 기술

다음 표에는 지정된 OpenShift Container Platform 클러스터 애플리케이션에 권장되는 구성 가능한 스토리지 기술이 요약되어 있습니다.

표 7.2. 권장되는 구성 가능한 스토리지 기술

스토리지 유형	ROX[1]	RWX[2]	Registry	확장 레지스트리	Metrics[3]	로깅	앱
블록	제공됨[4]	없음	구성 가능	구성 불가능	권장	권장	권장
파일	제공됨[4]	예	구성 가능	구성 가능	구성 가능[5]	구성 가능[6]	권장
개체	예	예	권장	권장	구성 불가능	구성 불가능	구성 불가능[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

[3] Prometheus는 메트릭에 사용되는 기본 기술입니다.

[4] 물리적 디스크, VM 물리적 디스크, VMDK, NFS를 통한 루프백, AWS EBS 및 Azure Disk에는 적용되지 않습니다.

[5] 메트릭의 경우 RWX(**ReadWriteMany**) 액세스 모드로 파일 스토리지를 사용하는 것은 안정적이지 않습니다. 파일 스토리지를 사용하는 경우 지표와 함께 사용하도록 구성된 PVC(영구 볼륨 클레임)에서 RWX 액세스 모드를 구성하지 마십시오.

[6] 로깅의 경우 공유 스토리지를 사용하는 것이 패턴 차단입니다. Elasticsearch당 하나의 볼륨이 필요합니다.

[7] OpenShift Container Platform의 PV 또는 PVC를 통해서는 오브젝트 스토리지가 사용되지 않습니다. 앱은 오브젝트 스토리지 REST API와 통합해야 합니다.

참고

확장 레지스트리는 두 개 이상의 **Pod** 복제본이 실행되는 **OpenShift Container Platform** 레지스트리입니다.

7.6.1. 특정 애플리케이션 스토리지 권장 사항

중요

테스트에서는 **RHEL(Red Hat Enterprise Linux)**의 **NFS** 서버를 핵심 서비스용 스토리지 백엔드로 사용하는 데 문제가 있는 것을 보여줍니다. 여기에는 **OpenShift Container Registry and Quay,** 스토리지 모니터링을 위한 **Prometheus,** 로깅 스토리지를 위한 **Elasticsearch**가 포함됩니다. 따라서 **RHEL NFS**를 사용하여 핵심 서비스에서 사용하는 **PV**를 백업하는 것은 권장되지 않습니다.

마켓플레이스의 다른 **NFS** 구현에는 이러한 문제가 나타나지 않을 수 있습니다. 이러한 **OpenShift Container Platform** 핵심 구성 요소에 대해 완료된 테스트에 대한 자세한 내용은 개별 **NFS** 구현 공급업체에 문의하십시오.

7.6.1.1. 레지스트리

비확장/**HA(**고가용성**) OpenShift Container Platform** 레지스트리 클러스터 배포에서는 다음 사항에 유의합니다.

- 스토리지 기술에서 **RWX** 액세스 모드를 지원할 필요가 없습니다.
- 스토리지 기술에서 쓰기 후 읽기 일관성을 보장해야 합니다.
- 기본 스토리지 기술은 오브젝트 스토리지, 블록 스토리지 순입니다.
- 프로덕션 워크로드가 있는 **OpenShift Container Platform** 레지스트리 클러스터 배포에는 파일 스토리지를 사용하지 않는 것이 좋습니다.

7.6.1.2. 확장 레지스트리

확장/**HA OpenShift Container Platform** 레지스트리 클러스터 배포에서는 다음 사항에 유의합니다.

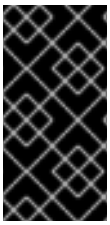
- 스토리지 기술은 **RWX** 액세스 모드를 지원해야 합니다.
- 스토리지 기술에서 쓰기 후 읽기 일관성을 보장해야 합니다.

- 기본 스토리지 기술은 오브젝트 스토리지입니다.
- Red Hat OpenShift Data Foundation(ODF), Amazon Simple Storage Service(Amazon S3), GCS(Google Cloud Storage), Microsoft Azure Blob Storage 및 OpenStack Swift가 지원됩니다.
- 오브젝트 스토리지는 S3 또는 Swift와 호환되어야 합니다.
- vSphere, 베어 메탈 설치 등 클라우드 이외의 플랫폼에서는 구성 가능한 유일한 기술이 파일 스토리지입니다.
- 블록 스토리지는 구성 불가능합니다.

7.6.1.3. 지표

OpenShift Container Platform 호스트 지표 클러스터 배포에서는 다음 사항에 유의합니다.

- 기본 스토리지 기술은 블록 스토리지입니다.
- 오브젝트 스토리지는 구성 불가능합니다.



중요

프로덕션 워크로드가 있는 호스트 지표 클러스터 배포에는 파일 스토리지를 사용하지 않는 것이 좋습니다.

7.6.1.4. 로깅

OpenShift Container Platform 호스트 로깅 클러스터 배포에서는 다음 사항에 유의합니다.

- 기본 스토리지 기술은 블록 스토리지입니다.

- 오브젝트 스토리지는 구성 불가능합니다.

7.6.1.5. 애플리케이션

애플리케이션 사용 사례는 다음 예에 설명된 대로 애플리케이션마다 다릅니다.

- 동적 PV 프로비저닝을 지원하는 스토리지 기술은 마운트 대기 시간이 짧고 정상 클러스터를 지원하는 노드와 관련이 없습니다.

- 애플리케이션 개발자는 애플리케이션의 스토리지 요구사항을 잘 알고 있으며 제공된 스토리지로 애플리케이션을 작동시켜 애플리케이션이 스토리지 계층을 스케일링하거나 스토리지 계층과 상호 작용할 때 문제가 발생하지 않도록 하는 방법을 이해하고 있어야 합니다.

7.6.2. 다른 특정 애플리케이션 스토리지 권장 사항

중요

etcd 와 같은 쓰기 집약적 워크로드에서는 RAID 구성을 사용하지 않는 것이 좋습니다. RAID 구성으로 etcd 를 실행하는 경우 워크로드의 성능 문제가 발생할 위험이 있을 수 있습니다.

- RHOSP(Red Hat OpenStack Platform) Cinder: RHOSP Cinder는 ROX 액세스 모드 사용 사례에 적합합니다.

- 데이터베이스: 데이터베이스 (RDBMS, NoSQL DB 등)는 전용 블록 스토리지에서 최적으로 작동하는 경향이 있습니다.

- etcd 데이터베이스에는 대규모 클러스터를 활성화하려면 충분한 스토리지와 충분한 성능 용량이 있어야 합니다. 충분한 스토리지 및 고성능 환경을 구축하기 위한 모니터링 및 벤치마킹 툴에 대한 정보는 *권장 etcd 관행에* 설명되어 있습니다.

추가 리소스

- etcd 관련 권장 사례

7.7. RED HAT OPENSHIFT CONTAINER STORAGE 배포

Red Hat OpenShift Container Storage는 내부 또는 하이브리드 클라우드 환경에서 파일, 블록 및 객체 스토리지를 지원하는 OpenShift Container Platform의 영구 스토리지 제공 업체입니다. Red Hat 스토리지 솔루션인 Red Hat OpenShift Container Storage는 배포, 관리 및 모니터링을 위해 OpenShift Container Platform과 완전히 통합되어 있습니다.

Red Hat OpenShift Container Storage 관련 정보를 찾고 있는 경우	다음 Red Hat OpenShift Container Storage 문서를 참조
새로운 알려진 문제, 중요한 버그 수정 및 기술 미리보기	OpenShift Container Storage 4.7 릴리스 노트
지원되는 워크로드, 레이아웃, 하드웨어 및 소프트웨어 요구 사항, 크기 조정 및 확장 권장 사항	Planning your OpenShift Container Storage 4.5 deployment
환경이 인터넷에 직접 연결되지 않은 경우 배포 준비를 위한 지침	Preparing to deploy OpenShift Container Storage 4.5 in a disconnected environment
외부 Red Hat Ceph Storage 클러스터를 사용하기 위해 OpenShift Container Storage를 배포하는 방법	Deploying OpenShift Container Storage 4.5 in external mode
베어 메탈 인프라의 로컬 스토리지에 OpenShift Container Storage를 배포하는 방법	Deploying OpenShift Container Storage 4.5 using bare metal infrastructure
Red Hat OpenShift Container Platform VMware vSphere 클러스터에 OpenShift Container Storage를 배포하는 방법	Deploying OpenShift Container Storage 4.5 on VMware vSphere
로컬 또는 클라우드 스토리지 용 Amazon Web Services를 사용하여 OpenShift Container Storage를 배포하는 방법	Deploying OpenShift Container Storage 4.5 using Amazon Web Services
기존 Red Hat OpenShift Container Platform Google Cloud 클러스터에서 OpenShift Container Storage를 배포 및 관리하는 방법	Deploying and managing OpenShift Container Storage 4.5 using Google Cloud
기존 Red Hat OpenShift Container Platform Azure 클러스터에서 OpenShift Container Storage를 배포 및 관리하는 방법	Deploying and managing OpenShift Container Storage 4.5 using Microsoft Azure
Red Hat OpenShift Container Storage 4.5 클러스터 관리	Managing OpenShift Container Storage 4.5
Red Hat OpenShift Container Storage 4.5 클러스터 모니터링	Monitoring Red Hat OpenShift Container Storage 4.5
작업 중 발생한 문제 해결	Troubleshooting OpenShift Container Storage 4.5

Red Hat OpenShift Container Storage 관련 정보를 찾고 있는 경우	다음 Red Hat OpenShift Container Storage 문서를 참조
OpenShift Container Platform 클러스터를 버전 3에서 버전 4로 마이그레이션	Migration

8장. 사용자를 위한 준비

OpenShift Container Platform을 설치한 후에는 사용자 준비 단계를 포함하여 요구 사항에 맞게 클러스터를 추가로 확장하고 사용자 정의할 수 있습니다.

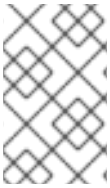
8.1. ID 공급자 구성 이해

OpenShift Container Platform 컨트롤 플레인에는 내장 **OAuth** 서버가 포함되어 있습니다. 개발자와 관리자는 **OAuth** 액세스 토큰을 가져와 **API** 인증을 수행합니다.

관리자는 클러스터를 설치한 후 **ID** 공급자를 지정하도록 **OAuth**를 구성할 수 있습니다.

8.1.1. OpenShift Container Platform의 ID 공급자 정보

기본적으로는 **kubeadmin** 사용자만 클러스터에 있습니다. **ID** 공급자를 지정하려면 해당 **ID** 공급자를 설명하는 **CR**(사용자 정의 리소스)을 생성하여 클러스터에 추가해야 합니다.



참고

/, :, %를 포함하는 **OpenShift Container Platform** 사용자 이름은 지원되지 않습니다.

8.1.2. 지원되는 ID 공급자

다음 유형의 **ID** 공급자를 구성할 수 있습니다.

ID 공급자	설명
htpasswd	**htpasswd**를 사용하여 생성된 플랫 파일에 대해 사용자 이름 및 암호의 유효성을 확인하도록 **htpasswd** ID 공급자를 구성합니다.
Keystone	내부 데이터베이스에 사용자를 저장하는 OpenStack Keystone v3 서버와의 공유 인증을 지원하기 위해 OpenShift Container Platform 클러스터를 Keystone과 통합하도록 **keystone** ID 공급자를 구성합니다.
LDAP	단순 바인드 인증을 사용하여 LDAPv3 서버에 대해 사용자 이름 및 암호의 유효성을 확인하도록 **ldap** ID 공급자를 구성합니다.

ID 공급자	설명
기본 인증	사용자가 원격 ID 공급자에 대해 검증된 자격 증명을 사용하여 OpenShift Container Platform에 로그인할 수 있도록 **기본 인증** ID 공급자를 구성합니다. 기본 인증은 일반적인 백엔드 통합 메커니즘입니다.
요청 헤더	**X-Remote-User**와 같은 요청 헤더 값에서 사용자를 확인하도록 **요청 헤더** ID 공급자를 구성합니다. 일반적으로 요청 헤더 값을 설정하는 인증 프록시와 함께 사용됩니다.
GitHub 또는 GitHub Enterprise	GitHub 또는 GitHub Enterprise의 OAuth 인증 서버에 대해 사용자 이름 및 암호의 유효성을 확인하도록 **github** ID 공급자를 구성합니다.
GitLab	GitLab.com 또는 기타 GitLab 인스턴스를 ID 공급자로 사용하도록 **gitlab** ID 공급자를 구성합니다.
Google	Google의 OpenID Connect 통합을 사용하여 **google** ID 공급자를 구성합니다.
OpenID Connect	인증 코드 Flow를 사용하여 OpenID Connect ID 공급자와 통합하도록 **oidc** ID 공급자를 구성합니다.

ID 공급자를 정의한 후 RBAC를 사용하여 권한을 정의 및 적용할 수 있습니다.

8.1.3. ID 공급자 매개변수

다음 매개변수는 모든 ID 공급자에 공통입니다.

매개변수	설명
name	공급자 사용자 이름에 접두어로 공급자 이름을 지정하여 ID 이름을 만듭니다.

매개변수	설명
mappingMethod	사용자가 로그인할 때 새 ID를 사용자에게 매핑하는 방법을 정의합니다. 다음 값 중 하나를 입력하십시오. **claim** 기본값입니다. 사용자에게 ID의 기본 사용자 이름을 프로비저닝합니다. 해당 사용자 이름의 사용자가 이미 다른 ID에 매핑되어 있는 경우 실패합니다. **lookup** 기존 ID, 사용자 ID 매핑 및 사용자를 조회하지만 사용자 또는 ID를 자동으로 프로비저닝하지는 않습니다. 클러스터 관리자는 이를 통해 수동으로 또는 외부 프로세스를 사용하여 ID 및 사용자를 설정할 수 있습니다. 이 방법을 사용하려면 사용자를 수동으로 프로비저닝해야 합니다. **generate** 사용자에게 ID의 기본 사용자 이름을 프로비저닝합니다. 기본 사용자 이름을 가진 사용자가 이미 기존 ID에 매핑되어 있는 경우 고유한 사용자 이름이 생성됩니다. 예를 들면 **myuser2**입니다. OpenShift Container Platform 사용자 이름과 ID 공급자 사용자 이름(예: LDAP 그룹 동기화)이 정확히 일치해야 하는 외부 프로세스와 함께 이 방법을 사용해서는 안 됩니다. **add** 사용자에게 ID의 기본 사용자 이름을 프로비저닝합니다. 해당 사용자 이름을 가진 사용자가 이미 존재하는 경우 ID가 기존 사용자에게 매핑되고 그 사용자의 기존 ID 매핑에 추가됩니다. 동일한 사용자 집합을 식별하고 동일한 사용자 이름에 매핑되는 ID 공급자를 여럿 구성한 경우 필요합니다.

참고

ID 공급자를 추가하거나 변경할 때 mappingMethod 매개변수를 add로 설정하면 새 공급자의 ID를 기존 사용자에게 매핑할 수 있습니다.

8.1.4. ID 공급자 CR 샘플

다음 CR(사용자 정의 리소스)에서는 ID 공급자를 구성하는 데 사용되는 매개변수 및 기본값을 보여줍니다. 이 예에서는 htpasswd ID 공급자를 사용합니다.

ID 공급자 CR 샘플

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_identity_provider 1
    mappingMethod: claim 2
    type: HTPasswd
```

```
htpasswd:
  fileData:
    name: htpass-secret 3
```

1

이 공급자 이름은 공급자 사용자 이름에 접두어로 지정되어 **ID** 이름을 형성합니다.

2

이 공급자의 **ID**와 **User** 오브젝트 간 매핑 설정 방법을 제어합니다.

3

htpasswd를 사용하여 생성한 파일이 포함된 기존 시크릿입니다.

8.2. RBAC를 사용하여 권한 정의 및 적용

역할 기반 액세스 제어를 이해하고 적용합니다.

8.2.1. RBAC 개요

RBAC(역할 기반 액세스 제어) 오브젝트에 따라 사용자가 프로젝트 내에서 지정된 작업을 수행할 수 있는지가 결정됩니다.

클러스터 관리자는 클러스터 역할 및 바인딩을 사용하여 **OpenShift Container Platform** 플랫폼 자체 및 모든 프로젝트에 대해 다양한 액세스 수준을 보유한 사용자를 제어할 수 있습니다.

개발자는 로컬 역할 및 바인딩을 사용하여 프로젝트에 액세스할 수 있는 사용자를 제어할 수 있습니다. 권한 부여는 인증과 별도의 단계이며, 여기서는 조치를 수행할 사용자의 신원을 파악하는 것이 더 중요합니다.

권한 부여는 다음을 사용하여 관리합니다.

권한 부여 오브젝트	설명
규칙	오브젝트 집합에 허용되는 동사 집합입니다. 예를 들면 사용자 또는 서비스 계정의 Pod **생성** 가능 여부입니다.
역할	규칙 모음입니다. 사용자와 그룹을 여러 역할에 연결하거나 바인딩할 수 있습니다.
바인딩	역할이 있는 사용자 및/또는 그룹 간 연결입니다.

권한 부여를 제어하는 두 가지 수준의 RBAC 역할 및 바인딩이 있습니다.

RBAC 수준	설명
클러스터 RBAC	모든 프로젝트에 적용할 수 있는 역할 및 바인딩입니다. *클러스터 역할*은 클러스터 전체에 존재하며 *클러스터 역할 바인딩*은 클러스터 역할만 참조할 수 있습니다.
지역 RBAC	지정된 프로젝트에 적용되는 역할 및 바인딩입니다. *로컬 역할*은 단일 프로젝트에만 존재하지만 로컬 역할 바인딩은 클러스터 및 로컬 역할을 *모두* 참조할 수 있습니다.

클러스터 역할 바인딩은 클러스터 수준에 존재하는 바인딩입니다. 역할 바인딩은 프로젝트 수준에 있습니다. 해당 사용자가 프로젝트를 보려면 로컬 역할 바인딩을 사용하여 클러스터 역할 *보기*를 사용자에게 바인딩해야 합니다. 클러스터 역할이 특정 상황에 필요한 권한 집합을 제공하지 않는 경우에만 로컬 역할을 생성하십시오.

이러한 2단계 계층 구조로 인해 클러스터 역할로는 여러 프로젝트에서 재사용하고, 로컬 역할로는 개별 프로젝트 내에서 사용자 정의할 수 있습니다.

평가 중에는 클러스터 역할 바인딩과 로컬 역할 바인딩이 모두 사용됩니다. 예를 들면 다음과 같습니다.

1. **클러스터 전체의 "허용" 규칙을 확인합니다.**
2. **로컬 바인딩된 "허용" 규칙을 확인합니다.**
3. **기본적으로 거부합니다.**

8.2.1.1. 기본 클러스터 역할

OpenShift Container Platform에는 클러스터 전체 또는 로컬로 사용자 및 그룹에 바인딩할 수 있는 기본 클러스터 역할 집합이 포함되어 있습니다.

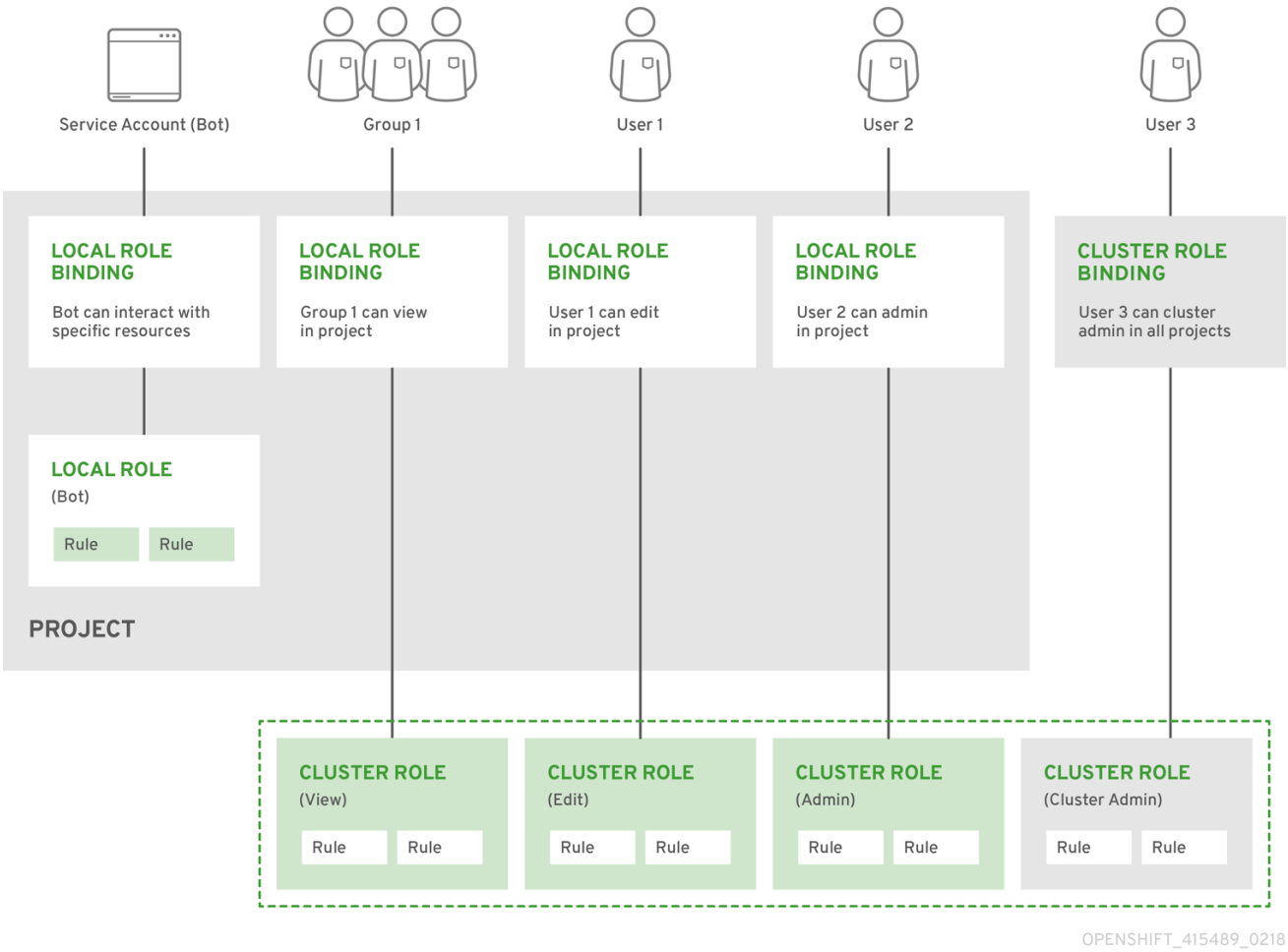
중요

기본 클러스터 역할을 수동으로 수정하지 않는 것이 좋습니다. 이러한 시스템 역할에 대한 수정으로 인해 클러스터가 제대로 작동하지 않을 수 있습니다.

기본 클러스터 역할	설명
admin	프로젝트 관리자입니다. 로컬 바인딩에 사용되는 경우 **admin**은 프로젝트의 모든 리소스를 보고 할당량을 제외한 프로젝트의 모든 리소스를 수정할 수 있는 권한이 있습니다.
basic-user	프로젝트 및 사용자에 대한 기본 정보를 가져올 수 있는 사용자입니다.
cluster-admin	모든 프로젝트에서 모든 작업을 수행할 수 있는 슈퍼 유저입니다. 로컬 바인딩을 통해 사용자에게 바인딩하면 할당량은 물론 프로젝트의 모든 리소스에 대한 모든 조치를 완전히 제어할 수 있습니다.
cluster-status	기본 클러스터 상태 정보를 가져올 수 있는 사용자입니다.
cluster-reader	대부분의 개체를 가져오거나 볼 수 있지만 수정할 수는 없는 사용자입니다.
edit	프로젝트에서 대부분의 오브젝트를 수정할 수 있지만 역할이나 바인딩을 보거나 수정할 권한은 없는 사용자입니다.
self-provisioner	자체 프로젝트를 만들 수 있는 사용자입니다.
view	수정할 수는 없지만 프로젝트의 오브젝트를 대부분 볼 수 있는 사용자입니다. 역할 또는 바인딩을 보거나 수정할 수 없습니다.

로컬 바인딩과 클러스터 바인딩의 차이점에 유의하십시오. 예를 들어 로컬 역할 바인딩을 사용하여 cluster-admin 역할을 사용자에게 바인딩하는 경우, 이 사용자에게 클러스터 관리자 권한이 있는 것처럼 보일 수 있습니다. 사실은 그렇지 않습니다. 프로젝트의 사용자에게 cluster-admin을 바인딩하면 해당 프로젝트에 대해서만 슈퍼 관리자 권한이 사용자에게 부여됩니다. 해당 사용자에게는 클러스터 역할 admin의 권한을 비롯하여 해당 프로젝트에 대한 속도 제한 편집 기능과 같은 몇 가지 추가 권한이 있습니다. 이 바인딩은 실제 클러스터 관리자에게 바인딩된 클러스터 역할 바인딩이 나열되지 않는 웹 콘솔 UI로 인해 혼동될 수 있습니다. 그러나 cluster-admin을 로컬로 바인딩하는 데 사용할 수 있는 로컬 역할 바인딩은 나열됩니다.

아래에는 클러스터 역할, 로컬 역할, 클러스터 역할 바인딩, 로컬 역할 바인딩, 사용자, 그룹, 서비스 계정 간의 관계가 설명되어 있습니다.



주의

get pods/exec,pods/*, get * 규칙은 역할에 적용될 때 실행 권한을 부여합니다. 최소 권한 원칙을 적용하고 사용자 및 에이전트에 필요한 최소 **RBAC** 권한만 할당합니다. 자세한 내용은 **RBAC** 규칙 실행 권한을 참조하십시오.

8.2.1.2. 권한 부여 평가

OpenShift Container Platform에서는 다음을 사용하여 권한 부여를 평가합니다.

ID

사용자 이름 및 사용자가 속한 그룹 목록입니다.

작업

수행하는 작업입니다. 대부분의 경우 다음으로 구성됩니다.

- 프로젝트: 액세스하는 프로젝트입니다. 프로젝트는 추가 주석이 있는 쿠버네티스 네임스페이스로, 사용자 커뮤니티가 다른 커뮤니티와 별도로 컨텐츠를 구성하고 관리할 수 있습니다.

- 동사: 작업 자체를 나타내며 **get, list, create, update, delete, deletecollection** 또는 **watch**에 해당합니다.

- 리소스 이름: 액세스하는 **API** 끝점입니다.

바인딩

전체 바인딩 목록으로, 역할이 있는 사용자 또는 그룹 간 연결을 나타냅니다.

OpenShift Container Platform에서는 다음 단계를 사용하여 권한 부여를 평가합니다.

1. **ID** 및 프로젝트 범위 작업은 사용자 또는 해당 그룹에 적용되는 모든 바인딩을 찾는 데 사용됩니다.

2. 바인딩은 적용되는 모든 역할을 찾는 데 사용됩니다.

3. 역할은 적용되는 모든 규칙을 찾는 데 사용됩니다.

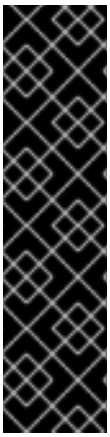
4. 일치하는 규칙을 찾기 위해 작업을 각 규칙에 대해 확인합니다.

5. 일치하는 규칙이 없으면 기본적으로 작업이 거부됩니다.

작은 정보

사용자 및 그룹을 동시에 여러 역할과 연결하거나 바인딩할 수 있습니다.

프로젝트 관리자는 **CLI**를 사용하여 각각 연결된 동사 및 리소스 목록을 포함하여 로컬 역할 및 바인딩을 볼 수 있습니다.



중요

프로젝트 관리자에게 바인딩된 클러스터 역할은 로컬 바인딩을 통해 프로젝트에서 제한됩니다. **cluster-admin** 또는 **system:admin**에 부여되는 클러스터 역할과 같이 클러스터 전체에 바인딩되지 않습니다.

클러스터 역할은 클러스터 수준에서 정의된 역할이지만 클러스터 수준 또는 프로젝트 수준에서 바인딩할 수 있습니다.

8.2.1.2.1. 클러스터 역할 집계

기본 관리, 편집, 보기 및 클러스터 독자 클러스터 역할에서는 새 역할이 생성될 때 각 역할에 대한 클러스터 규칙이 동적으로 업데이트되는 클러스터 역할 집계를 지원합니다. 이 기능은 사용자 정의 리소스를 생성하여 쿠버네티스 **API**를 확장한 경우에만 관련이 있습니다.

8.2.2. 프로젝트 및 네임스페이스

쿠버네티스 *네임스페이스*는 클러스터의 리소스 범위를 지정하는 메커니즘을 제공합니다. 쿠버네티스 설명서에 네임스페이스에 대한 자세한 정보가 있습니다.

네임스페이스는 다음에 대한 고유 범위를 제공합니다.

- 기본 이름 지정 충돌을 피하기 위해 이름이 지정된 리소스
- 신뢰할 수 있는 사용자에게 위임된 관리 권한
- 커뮤니티 리소스 사용을 제한하는 기능

시스템에 있는 대부분의 오브젝트는 네임스페이스에 따라 범위가 지정되지만, 노드 및 사용자를 비롯한 일부는 여기에 해당하지 않으며 네임스페이스가 없습니다.

*프로젝트*는 추가 주석이 있는 쿠버네티스 네임스페이스이며, 일반 사용자용 리소스에 대한 액세스를

관리하는 가장 중요한 수단입니다. 사용자 커뮤니티는 프로젝트를 통해 다른 커뮤니티와 별도로 콘텐츠를 구성하고 관리할 수 있습니다. 사용자는 관리자로부터 프로젝트에 대한 액세스 권한을 부여받아야 합니다. 프로젝트를 생성하도록 허용된 경우 자신의 프로젝트에 액세스할 수 있는 권한이 자동으로 제공됩니다.

프로젝트에는 별도의 **name, displayName, description**이 있을 수 있습니다.

- 필수 항목인 **name**은 프로젝트의 고유 식별자이며 **CLI** 도구 또는 **API**를 사용할 때 가장 잘 보입니다. 최대 이름 길이는 **63**자입니다.
- 선택적 **displayName**은 프로젝트가 웹 콘솔에 표시되는 방법입니다(기본값: **name**).
- 선택적 **description**은 프로젝트에 대한 보다 자세한 설명으로, 웹 콘솔에서도 볼 수 있습니다.

각 프로젝트의 범위는 다음과 같습니다.

오브젝트	설명
Objects	Pod, 서비스, 복제 컨트롤러 등입니다.
Policies	사용자는 오브젝트에서 이 규칙에 대해 작업을 수행할 수 있거나 수행할 수 없습니다.
Constraints	제한할 수 있는 각 종류의 오브젝트에 대한 할당량입니다.
Service accounts	서비스 계정은 프로젝트의 오브젝트에 지정된 액세스 권한으로 자동으로 작동합니다.

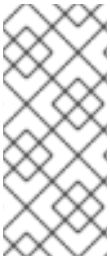
클러스터 관리자는 프로젝트를 생성하고 프로젝트에 대한 관리 권한을 사용자 커뮤니티의 모든 멤버에게 위임할 수 있습니다. 클러스터 관리자는 개발자가 자신의 프로젝트를 만들 수 있도록 허용할 수도 있습니다.

개발자와 관리자는 **CLI** 또는 웹 콘솔을 사용하여 프로젝트와 상호 작용할 수 있습니다.

8.2.3. 기본 프로젝트

OpenShift Container Platform에는 다양한 기본 프로젝트가 제공되며, **openshift-**로 시작하는 프로

젝트가 사용자에게 가장 중요합니다. 이러한 프로젝트는 **Pod** 및 기타 인프라 구성 요소로 실행되는 마스터 구성 요소를 호스팅합니다. 중요 **Pod** 주석이 있는 네임스페이스에 생성된 **Pod**는 중요한 **Pod**로 간주되며, **kubelet**의 승인이 보장됩니다. 이러한 네임스페이스에서 마스터 구성 요소용으로 생성된 **Pod**는 이미 중요로 표시되어 있습니다.



참고

기본 네임스페이스(**default, kube-system, kube-public, openshift-node, openshift-infra** 및 **openshift**) 중 하나에서 생성된 **Pod**에는 **SCC**를 할당할 수 없습니다. 이러한 네임스페이스는 **Pod** 또는 서비스를 실행하는 데 사용할 수 없습니다.

8.2.4. 클러스터 역할 및 바인딩 보기

oc CLI에서 **oc describe** 명령을 사용하여 클러스터 역할 및 바인딩을 볼 수 있습니다.

사전 요구 사항

- **oc CLI**를 설치합니다.
- 클러스터 역할 및 바인딩을 볼 수 있는 권한을 얻습니다.

cluster-admin 기본 클러스터 역할이 클러스터 전체에서 바인딩된 사용자는 클러스터 역할 및 바인딩 보기를 포함하여 모든 리소스에 대해 모든 작업을 수행할 수 있습니다.

절차

1. 클러스터 역할 및 관련 규칙 집합을 보려면 다음을 수행합니다.

```
$ oc describe clusterrole.rbac
```

출력 예

```
Name:         admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names  Verbs
  ---------                                  -----------------  --------------  -----
```

```
 .packages.apps.redhat.com                                []              []              [* create update
patch delete get list watch]
 imagestreams                                             []              []              [create delete
deletecollection get list patch update watch create get list watch]
 imagestreams.image.openshift.io                          []              []              [create delete
deletecollection get list patch update watch create get list watch]
 secrets                                                  []              []              [create delete
deletecollection get list patch update watch get list watch create delete
deletecollection patch update]
 buildconfigs/webhooks                                    []              []              [create delete
deletecollection get list patch update watch get list watch]
 buildconfigs                                             []              []              [create delete
deletecollection get list patch update watch get list watch]
 buildlogs                                                []              []              [create delete
deletecollection get list patch update watch get list watch]
 deploymentconfigs/scale                                  []              []              [create delete
deletecollection get list patch update watch get list watch]
 deploymentconfigs                                        []              []              [create delete
deletecollection get list patch update watch get list watch]
 imagestreamimages                                        []              []              [create delete
deletecollection get list patch update watch get list watch]
 imagestreammappings                                      []              []              [create delete
deletecollection get list patch update watch get list watch]
 imagestreamtags                                          []              []              [create delete
deletecollection get list patch update watch get list watch]
 processedtemplates                                       []              []              [create delete
deletecollection get list patch update watch get list watch]
 routes                                                   []              []              [create delete
deletecollection get list patch update watch get list watch]
 templateconfigs                                          []              []              [create delete
deletecollection get list patch update watch get list watch]
 templateinstances                                        []              []              [create delete
deletecollection get list patch update watch get list watch]
 templates                                                []              []              [create delete
deletecollection get list patch update watch get list watch]
 deploymentconfigs.apps.openshift.io/scale                []              []              [create
delete deletecollection get list patch update watch get list watch]
 deploymentconfigs.apps.openshift.io                      []              []              [create delete
deletecollection get list patch update watch get list watch]
 buildconfigs.build.openshift.io/webhooks                 []              []              [create
delete deletecollection get list patch update watch get list watch]
 buildconfigs.build.openshift.io                          []              []              [create delete
deletecollection get list patch update watch get list watch]
 buildlogs.build.openshift.io                             []              []              [create delete
deletecollection get list patch update watch get list watch]
 imagestreamimages.image.openshift.io                     []              []              [create
delete deletecollection get list patch update watch get list watch]
 imagestreammappings.image.openshift.io                   []              []              [create
delete deletecollection get list patch update watch get list watch]
 imagestreamtags.image.openshift.io                       []              []              [create delete
deletecollection get list patch update watch get list watch]
 routes.route.openshift.io                                []              []              [create delete
deletecollection get list patch update watch get list watch]
 processedtemplates.template.openshift.io                 []              []              [create
delete deletecollection get list patch update watch get list watch]
 templateconfigs.template.openshift.io                    []              []              [create delete
```

```
deletecollection get list patch update watch get list watch]
  templateinstances.template.openshift.io                  []              []              [create delete
deletecollection get list patch update watch get list watch]
  templates.template.openshift.io                          []              []              [create delete
deletecollection get list patch update watch get list watch]
  serviceaccounts                                          []              []              [create delete
deletecollection get list patch update watch impersonate create delete deletecollection
patch update get list watch]
  imagestreams/secrets                                     []              []              [create delete
deletecollection get list patch update watch]
  rolebindings                                             []              []              [create delete
deletecollection get list patch update watch]
  roles                                                    []              []              [create delete
deletecollection get list patch update watch]
  rolebindings.authorization.openshift.io                  []              []              [create delete
deletecollection get list patch update watch]
  roles.authorization.openshift.io                         []              []              [create delete
deletecollection get list patch update watch]
  imagestreams.image.openshift.io/secrets                  []              []              [create
delete deletecollection get list patch update watch]
  rolebindings.rbac.authorization.k8s.io                   []              []              [create delete
deletecollection get list patch update watch]
  roles.rbac.authorization.k8s.io                          []              []              [create delete
deletecollection get list patch update watch]
  networkpolicies.extensions                               []              []              [create delete
deletecollection patch update create delete deletecollection get list patch update
watch get list watch]
  networkpolicies.networking.k8s.io                        []              []              [create delete
deletecollection patch update create delete deletecollection get list patch update
watch get list watch]
  configmaps                                               []              []              [create delete
deletecollection patch update get list watch]
  endpoints                                                []              []              [create delete
deletecollection patch update get list watch]
  persistentvolumeclaims                                   []              []              [create delete
deletecollection patch update get list watch]
  pods                                                     []              []              [create delete
deletecollection patch update get list watch]
  replicationcontrollers/scale                             []              []              [create delete
deletecollection patch update get list watch]
  replicationcontrollers                                   []              []              [create delete
deletecollection patch update get list watch]
  services                                                 []              []              [create delete
deletecollection patch update get list watch]
  daemonsets.apps                                          []              []              [create delete
deletecollection patch update get list watch]
  deployments.apps/scale                                   []              []              [create delete
deletecollection patch update get list watch]
  deployments.apps                                         []              []              [create delete
deletecollection patch update get list watch]
  replicasets.apps/scale                                   []              []              [create delete
deletecollection patch update get list watch]
  replicasets.apps                                         []              []              [create delete
deletecollection patch update get list watch]
  statefulsets.apps/scale                                  []              []              [create delete
deletecollection patch update get list watch]
```

```
  statefulsets.apps                                  []                  []                  [create delete
deletecollection patch update get list watch]
  horizontalpodautoscalers.autoscaling               []                  []                  [create delete
deletecollection patch update get list watch]
  cronjobs.batch                                     []                  []                  [create delete
deletecollection patch update get list watch]
  jobs.batch                                         []                  []                  [create delete
deletecollection patch update get list watch]
  daemonsets.extensions                              []                  []                  [create delete
deletecollection patch update get list watch]
  deployments.extensions/scale                       []                  []                  [create delete
deletecollection patch update get list watch]
  deployments.extensions                             []                  []                  [create delete
deletecollection patch update get list watch]
  ingresses.extensions                               []                  []                  [create delete
deletecollection patch update get list watch]
  replicasets.extensions/scale                       []                  []                  [create delete
deletecollection patch update get list watch]
  replicasets.extensions                             []                  []                  [create delete
deletecollection patch update get list watch]
  replicationcontrollers.extensions/scale            []                  []                  [create delete
deletecollection patch update get list watch]
  poddisruptionbudgets.policy                        []                  []                  [create delete
deletecollection patch update get list watch]
  deployments.apps/rollback                          []                  []                  [create delete
deletecollection patch update]
  deployments.extensions/rollback                    []                  []                  [create delete
deletecollection patch update]
  catalogsources.operators.coreos.com                []                  []                  [create
update patch delete get list watch]
  clusterserviceversions.operators.coreos.com        []                  []                  [create
update patch delete get list watch]
  installplans.operators.coreos.com                  []                  []                  [create update
patch delete get list watch]
  packagemanifests.operators.coreos.com              []                  []                  [create
update patch delete get list watch]
  subscriptions.operators.coreos.com                 []                  []                  [create
update patch delete get list watch]
  buildconfigs/instantiate                           []                  []                  [create]
  buildconfigs/instantiatebinary                     []                  []                  [create]
  builds/clone                                       []                  []                  [create]
  deploymentconfigrollbacks                          []                  []                  [create]
  deploymentconfigs/instantiate                      []                  []                  [create]
  deploymentconfigs/rollback                         []                  []                  [create]
  imagestreamimports                                 []                  []                  [create]
  localresourceaccessreviews                         []                  []                  [create]
  localsubjectaccessreviews                          []                  []                  [create]
  podsecuritypolicyreviews                           []                  []                  [create]
  podsecuritypolicyselfsubjectreviews                []                  []                  [create]
  podsecuritypolicysubjectreviews                    []                  []                  [create]
  resourceaccessreviews                              []                  []                  [create]
  routes/custom-host                                 []                  []                  [create]
  subjectaccessreviews                               []                  []                  [create]
  subjectrulesreviews                                []                  []                  [create]
  deploymentconfigrollbacks.apps.openshift.io        []                  []                  [create]
  deploymentconfigs.apps.openshift.io/instantiate    []                  []                  [create]
```

```
  deploymentconfigs.apps.openshift.io/rollback          []                []                [create]
  localsubjectaccessreviews.authorization.k8s.io        []                []                [create]
  localresourceaccessreviews.authorization.openshift.io []                []
[create]
  localsubjectaccessreviews.authorization.openshift.io  []                []                [create]
  resourceaccessreviews.authorization.openshift.io      []                []                [create]
  subjectaccessreviews.authorization.openshift.io       []                []                [create]
  subjectrulesreviews.authorization.openshift.io        []                []                [create]
  buildconfigs.build.openshift.io/instantiate           []                []                [create]
  buildconfigs.build.openshift.io/instantiatebinary     []                []                [create]
  builds.build.openshift.io/clone                       []                []                [create]
  imagestreamimports.image.openshift.io                 []                []                [create]
  routes.route.openshift.io/custom-host                 []                []                [create]
  podsecuritypolicyreviews.security.openshift.io        []                []                [create]
  podsecuritypolicyselfsubjectreviews.security.openshift.io []            []
[create]
  podsecuritypolicysubjectreviews.security.openshift.io []                []
[create]
  jenkins.build.openshift.io                            []                []                [edit view view
admin edit view]
  builds                                                []                []                [get create delete
deletecollection get list patch update watch get list watch]
  builds.build.openshift.io                             []                []                [get create delete
deletecollection get list patch update watch get list watch]
  projects                                              []                []                [get delete get delete get
patch update]
  projects.project.openshift.io                         []                []                [get delete get
delete get patch update]
  namespaces                                            []                []                [get get list watch]
  pods/attach                                           []                []                [get list watch create
delete deletecollection patch update]
  pods/exec                                             []                []                [get list watch create
delete deletecollection patch update]
  pods/portforward                                      []                []                [get list watch create
delete deletecollection patch update]
  pods/proxy                                            []                []                [get list watch create
delete deletecollection patch update]
  services/proxy                                        []                []                [get list watch create
delete deletecollection patch update]
  routes/status                                         []                []                [get list watch update]
  routes.route.openshift.io/status                      []                []                [get list watch
update]
  appliedclusterresourcequotas                          []                []                [get list watch]
  bindings                                              []                []                [get list watch]
  builds/log                                            []                []                [get list watch]
  deploymentconfigs/log                                 []                []                [get list watch]
  deploymentconfigs/status                              []                []                [get list watch]
  events                                                []                []                [get list watch]
  imagestreams/status                                   []                []                [get list watch]
  limitranges                                           []                []                [get list watch]
  namespaces/status                                     []                []                [get list watch]
  pods/log                                              []                []                [get list watch]
  pods/status                                           []                []                [get list watch]
  replicationcontrollers/status                         []                []                [get list watch]
  resourcequotas/status                                 []                []                [get list watch]
  resourcequotas                                        []                []                [get list watch]
```

```
  resourcequotausages                                []              []              [get list watch]
  rolebindingrestrictions                            []              []              [get list watch]
  deploymentconfigs.apps.openshift.io/log            []              []              [get list
watch]
  deploymentconfigs.apps.openshift.io/status         []              []              [get list
watch]
  controllerrevisions.apps                           []              []              [get list watch]
  rolebindingrestrictions.authorization.openshift.io []              []              [get list
watch]
  builds.build.openshift.io/log                      []              []              [get list watch]
  imagestreams.image.openshift.io/status             []              []              [get list
watch]
  appliedclusterresourcequotas.quota.openshift.io    []              []              [get list
watch]
  imagestreams/layers                                []              []              [get update get]
  imagestreams.image.openshift.io/layers             []              []              [get update
get]
  builds/details                                     []              []              [update]
  builds.build.openshift.io/details                  []              []              [update]


Name:         basic-user
Labels:       <none>
Annotations:  openshift.io/description: A user that can get basic information about
projects.
              rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
 Resources                                  Non-Resource URLs  Resource Names  Verbs
  ---------                                 -----------------  --------------  -----
  selfsubjectrulesreviews                   []                 []              [create]
  selfsubjectaccessreviews.authorization.k8s.io  []            []              [create]
  selfsubjectrulesreviews.authorization.openshift.io  []        []              [create]
  clusterroles.rbac.authorization.k8s.io    []                 []              [get list watch]
  clusterroles                              []                 []              [get list]
  clusterroles.authorization.openshift.io   []                 []              [get list]
  storageclasses.storage.k8s.io             []                 []              [get list]
  users                                     []                 [~]             [get]
  users.user.openshift.io                   []                 [~]             [get]
  projects                                  []                 []              [list watch]
  projects.project.openshift.io             []                 []              [list watch]
  projectrequests                           []                 []              [list]
  projectrequests.project.openshift.io      []                 []              [list]

Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
Resources  Non-Resource URLs  Resource Names  Verbs
---------  -----------------  --------------  -----
*.*        []                 []              [*]
           [*]                []              [*]

...
```

2. 다양한 역할에 바인딩된 사용자 및 그룹을 표시하는 현재 클러스터 역할 바인딩 집합을 보려면 다음을 수행하십시오.

```
$ oc describe clusterrolebinding.rbac
```

출력 예

```
Name:         alertmanager-main
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  alertmanager-main
Subjects:
  Kind          Name             Namespace
  ----          ----             ---------
  ServiceAccount  alertmanager-main  openshift-monitoring


Name:         basic-users
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  basic-user
Subjects:
  Kind   Name              Namespace
  ----   ----              ---------
  Group  system:authenticated


Name:         cloud-credential-operator-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cloud-credential-operator-role
Subjects:
  Kind          Name     Namespace
  ----          ----     ---------
  ServiceAccount  default  openshift-cloud-credential-operator


Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
```

```
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name          Namespace
  ----   ----          ---------
  Group  system:masters


Name:         cluster-admins
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name               Namespace
  ----   ----               ---------
  Group  system:cluster-admins
  User   system:admin


Name:         cluster-api-manager-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cluster-api-manager-role
Subjects:
  Kind          Name     Namespace
  ----          ----     ---------
  ServiceAccount  default  openshift-machine-api

...
```

8.2.5. 로컬 역할 및 바인딩 보기

oc CLI에서 **oc describe** 명령을 사용하여 로컬 역할 및 바인딩을 볼 수 있습니다.

사전 요구 사항

- **oc CLI**를 설치합니다.
- 로컬 역할 및 바인딩을 볼 수 있는 권한을 얻습니다.
  - **cluster-admin** 기본 클러스터 역할이 클러스터 전체에서 바인딩된 사용자는 로컬 역할

및 바인딩 보기를 포함하여 모든 리소스에 대해 모든 작업을 수행할 수 있습니다.

- **admin** 기본 클러스터 역할이 로컬로 바인딩된 사용자는 해당 프로젝트의 역할 및 바인딩을 보고 관리할 수 있습니다.

절차

1. 현재 프로젝트의 다양한 역할에 바인딩된 사용자 및 그룹을 표시하는 현재의 로컬 역할 바인딩 집합을 보려면 다음을 실행합니다.

```
$ oc describe rolebinding.rbac
```

2. 다른 프로젝트에 대한 로컬 역할 바인딩을 보려면 명령에 **-n** 플래그를 추가합니다.

```
$ oc describe rolebinding.rbac -n joe-project
```

출력 예

```
Name:         admin
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name       Namespace
  ----  ----       ---------
  User  kube:admin


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind          Name      Namespace
  ----          ----      ---------
  ServiceAccount  deployer  joe-project
```

```
Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind          Name     Namespace
  ----          ----     ---------
  ServiceAccount  builder  joe-project


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:serviceaccounts:joe-project
```

8.2.6. 사용자 역할 추가

oc adm 관리자 **CLI**를 사용하여 역할 및 바인딩을 관리할 수 있습니다.

사용자 또는 그룹에 역할을 바인딩하거나 추가하면 역할에 따라 사용자 또는 그룹에 부여되는 액세스 권한이 부여됩니다. **oc adm policy** 명령을 사용하여 사용자 및 그룹에 역할을 추가하거나 사용자 및 그룹으로부터 역할을 제거할 수 있습니다.

기본 클러스터 역할을 프로젝트의 로컬 사용자 또는 그룹에 바인딩할 수 있습니다.

절차

1. 특정 프로젝트의 사용자에게 역할을 추가합니다.

```
$ oc adm policy add-role-to-user <role> <user> -n <project>
```

예를 들면 다음을 실행하여 **joe** 프로젝트의 **alice** 사용자에게 **admin** 역할을 추가할 수 있습니다.

```
$ oc adm policy add-role-to-user admin alice -n joe
```

작은 정보

다음 **YAML**을 적용하여 사용자에게 역할을 추가할 수도 있습니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: admin-0
  namespace: joe
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

2. 로컬 역할 바인딩을 보고 출력에 추가되었는지 확인합니다.

```
$ oc describe rolebinding.rbac -n <project>
```

예를 들어, **joe** 프로젝트의 로컬 역할 바인딩을 보려면 다음을 수행합니다.

```
$ oc describe rolebinding.rbac -n joe
```

출력 예

```
Name:         admin
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
```

```
  Kind  Name       Namespace
  ----  ----       ---------
  User  kube:admin


Name:         admin-0
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice  (1)


Name:         system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind          Name      Namespace
  ----          ----      ---------
  ServiceAccount  deployer  joe


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind          Name     Namespace
  ----          ----     ---------
  ServiceAccount  builder  joe


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
```

```
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:serviceaccounts:joe
```

1

alice 사용자가 admins RoleBinding에 추가되었습니다.

8.2.7. 로컬 역할 생성

프로젝트의 로컬 역할을 생성하여 이 역할을 사용자에게 바인딩할 수 있습니다.

절차

1. 프로젝트의 로컬 역할을 생성하려면 다음 명령을 실행합니다.

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
```

이 명령에서는 다음을 지정합니다.

- <name>: 로컬 역할 이름
- <verb>: 역할에 적용할 동사를 쉼표로 구분한 목록
- <resource>: 역할이 적용되는 리소스
- <project>: 프로젝트 이름

예를 들어, 사용자가 blue 프로젝트의 Pod를 볼 수 있는 로컬 역할을 생성하려면 다음 명령을 실행합니다.

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. 새 역할을 사용자에게 바인딩하려면 다음 명령을 실행합니다.

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
```

8.2.8. 클러스터 역할 생성

클러스터 역할을 만들 수 있습니다.

절차

1. 클러스터 역할을 만들려면 다음 명령을 실행합니다.

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

이 명령에서는 다음을 지정합니다.

- **<name>:** 로컬 역할 이름
- **<verb>:** 역할에 적용할 동사를 쉼표로 구분한 목록
- **<resource>:** 역할이 적용되는 리소스

예를 들어 사용자가 **pod**를 볼 수 있는 클러스터 역할을 만들려면 다음 명령을 실행합니다.

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

8.2.9. 로컬 역할 바인딩 명령

다음 작업을 사용하여 로컬 역할 바인딩에 대한 사용자 또는 그룹의 연결된 역할을 관리하는 경우, **-n** 플래그를 사용하여 프로젝트를 지정할 수 있습니다. 지정하지 않으면 현재 프로젝트가 사용됩니다.

다음 명령을 사용하여 로컬 **RBAC**를 관리할 수 있습니다.

표 **8.1.** 로컬 역할 바인딩 작업

명령	설명
$ oc adm policy who-can *<verb> <resource>*	리소스에 작업을 수행할 수 있는 사용자를 나타냅니다.
$ oc adm policy add-role-to-user *<role> <username>*	현재 프로젝트에서 지정된 사용자에게 지정된 역할을 바인딩합니다.
$ oc adm policy remove-role-from-user *<role> <username>*	현재 프로젝트에서 지정된 사용자로부터 지정된 역할을 제거합니다.
$ oc adm policy remove-user *<username>*	현재 프로젝트에서 지정된 사용자 및 해당 사용자의 역할을 모두 제거합니다.
$ oc adm policy add-role-to-group *<role> <groupname>*	현재 프로젝트에서 지정된 그룹에 지정된 역할을 바인딩합니다.
$ oc adm policy remove-role-from-group *<role> <groupname>*	현재 프로젝트에서 지정된 그룹의 지정된 역할을 제거합니다.
$ oc adm policy remove-group *<groupname>*	현재 프로젝트에서 지정된 그룹과 해당 그룹의 역할을 모두 제거합니다.

8.2.10. 클러스터 역할 바인딩 명령

다음 작업을 사용하여 클러스터 역할 바인딩을 관리할 수도 있습니다. 클러스터 역할 바인딩에 네임스페이스가 아닌 리소스가 사용되므로 **-n** 플래그가 해당 작업에 사용되지 않습니다.

표 8.2. 클러스터 역할 바인딩 작업

명령	설명
$ oc adm policy add-cluster-role-to-user *<role> <username>*	클러스터의 모든 프로젝트에 대해 지정된 사용자에게 지정된 역할을 바인딩합니다.
$ oc adm policy remove-cluster-role-from-user *<role> <username>*	클러스터의 모든 프로젝트에 대해 지정된 사용자로부터 지정된 역할을 제거합니다.
$ oc adm policy add-cluster-role-to-group *<role> <groupname>*	클러스터의 모든 프로젝트에 대해 지정된 역할을 지정된 그룹에 바인딩합니다.
$ oc adm policy remove-cluster-role-from-group *<role> <groupname>*	클러스터의 모든 프로젝트에 대해 지정된 그룹에서 지정된 역할을 제거합니다.

8.2.11. 클러스터 관리자 생성

클러스터 리소스 수정과 같은 **OpenShift Container Platform** 클러스터에서 관리자 수준 작업을 수행

하려면 **cluster-admin** 역할이 필요합니다.

사전 요구 사항

- 클러스터 관리자로 정의할 사용자를 생성해야 합니다.

절차

- 사용자를 클러스터 관리자로 정의합니다.

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user>
```

8.3. KUBEADMIN 사용자

OpenShift Container Platform에서는 설치 프로세스가 완료되면 클러스터 관리자 **kubeadmin**을 생성합니다.

이 사용자는 **cluster-admin** 역할이 자동으로 적용되며 클러스터의 루트 사용자로 취급됩니다. 암호는 동적으로 생성되며 **OpenShift Container Platform** 환경에서 고유합니다. 설치가 완료되면 설치 프로그램의 출력에 암호가 제공됩니다. 예를 들면 다음과 같습니다.

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

8.3.1. kubeadmin 사용자 제거

ID 공급자를 정의하고 새 **cluster-admin** 사용자를 만든 다음 **kubeadmin**을 제거하여 클러스터 보안을 강화할 수 있습니다.

주의

다른 사용자가 **cluster-admin**이 되기 전에 이 절차를 수행하는 경우 **OpenShift Container Platform**을 다시 설치해야 합니다. 이 명령은 취소할 수 없습니다.

사전 요구 사항

- 하나 이상의 **ID** 공급자를 구성해야 합니다.
- 사용자에게 **cluster-admin** 역할을 추가해야 합니다.
- 관리자로 로그인해야 합니다.

절차

- **kubeadmin** 시크릿을 제거합니다.

```
$ oc delete secrets kubeadmin -n kube-system
```

8.4. 이미지 구성

이미지 레지스트리 설정을 이해하고 구성합니다.

8.4.1. 이미지 컨트롤러 구성 매개변수

image.config.openshift.io/cluster 리소스에는 이미지를 처리하는 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 **cluster**입니다. **spec**에서는 다음 구성 매개변수를 제공합니다.

참고

DisableScheduledImport,MaxImagesBulkImportedPerRepository,MaxScheduledImportsPerMinute,ScheduledImageImportMinimumIntervalSeconds 와 같은 매개변수는 구성할 수 없습니다.

매개변수	설명
allowedRegistriesForImport	일반 사용자가 이미지를 가져올 수 있는 컨테이너 이미지 레지스트리를 제한합니다. 이 목록은 유효한 이미지를 포함한다고 신뢰할 수 있으며 애플리케이션을 가져올 수 있도록 하려는 레지스트리로 설정합니다. 이미지를 생성할 권한이 있는 사용자 또는 API의 **ImageStreamMappings**는 이 정책의 영향을 받지 않습니다. 일반적으로 클러스터 관리자에게만 적절한 권한이 있습니다. 이 목록의 모든 요소에는 레지스트리 도메인 이름으로 지정된 레지스트리 위치가 포함되어 있습니다. **domainName**: 레지스트리의 도메인 이름을 지정합니다. 레지스트리에서 비표준 **80** 또는 **443** 포트를 사용하는 경우 도메인 이름에도 포트가 포함되어야 합니다. **insecure**: 비보안은 레지스트리가 안전한지 안전하지 않은지를 나타냅니다. 달리 지정하지 않으면 기본적으로 레지스트리는 안전한 것으로 간주됩니다.
additionalTrustedCA	**ImageStream import**, **pod image pull**, **openshift-image-registry pullthrough** 및 빌드 중에 신뢰해야 하는 추가 CA가 포함된 구성 맵에 대한 참조입니다. 이 구성 맵의 네임스페이스는 **openshift-config**입니다. 구성 맵 형식에서는 신뢰할 추가 레지스트리 CA마다 레지스트리 호스트 이름을 키로 사용하고 PEM으로 인코딩된 인증서를 값으로 사용합니다.
externalRegistryHostnames	기본 외부 이미지 레지스트리의 호스트 이름을 제공합니다. 외부 호스트 이름은 이미지 레지스트리가 외부에 노출되는 경우에만 설정되어야 합니다. 첫 번째 값은 이미지 스트림의 **publicDockerImageRepository** 필드에서 사용됩니다. 값은 **hostname[:port]** 형식이어야 합니다.

매개변수	설명
registrySources	빌드 및 pod 이미지에 액세스하는 경우 컨테이너 런타임에서 개별 레지스트리를 처리하는 방법을 결정할 구성이 포함되어 있습니다. 비보안 액세스 허용 여부를 예로 들 수 있습니다. 내부 클러스터 레지스트리에 대한 구성은 포함되어 있지 않습니다. **insecureRegistries**: 유효한 TLS 인증서가 없거나 HTTP 연결만 지원하는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. **blockedRegistries**: 이미지 풀 및 푸시 작업이 거부되는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. 다른 모든 레지스트리는 허용됩니다. **allowedRegistries**: 이미지 풀 및 푸시 작업을 허용하는 레지스트리입니다. 모든 하위 도메인을 지정하려면 별표(*) 와일드카드 문자를 도메인 이름에 접두사로 추가합니다. 예: ***.example.com** 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest**. 다른 모든 레지스트리는 차단됩니다. **containerRuntimeSearchRegistries**: 이미지의 짧은 이름을 사용하여 이미지 풀 및 푸시 작업을 허용하는 레지스트리입니다. 다른 모든 레지스트리는 차단됩니다. **blockedRegistries** 또는 **allowedRegistries**를 설정할 수 있으나 둘 다 설정할 수는 없습니다.

주의

allowedRegistries 매개변수가 정의되면 명시적으로 나열되지 않은 경우 registry.redhat.io, quay.io 및 기본 내부 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 Pod 실패를 방지하기 위해 환경의 페이로드 이미지에서 필요한 registry.redhat.io 및 quay.io 레지스트리 및 internalRegistryHostname을 포함한 모든 레지스트리를 allowedRegistries 목록에 추가합니다. 연결 해제된 클러스터의 경우 미러 레지스트리도 추가해야 합니다.

image.config.openshift.io/cluster 리소스의 상태 필드에는 클러스터에서 관찰된 값이 들어 있습니다.

매개변수	설명
internalRegistryHostname	**internalRegistryHostname**을 제어하는 Image Registry Operator가 설정합니다. 기본 내부 이미지 레지스트리의 호스트 이름을 설정합니다. 값은 **hostname[:port]** 형식이어야 합니다. 이전 버전과의 호환성을 위해 **OPENSHIFT_DEFAULT_REGISTRY** 환경 변수를 계속 사용할 수 있지만 이 설정을 통해 환경 변수가 재정의됩니다.
externalRegistryHostnames	Image Registry Operator가 설정하며, 이미지 레지스트리가 외부에 노출되는 경우 이미지 레지스트리의 외부 호스트 이름을 제공합니다. 첫 번째 값은 이미지 스트림의 **publicDockerImageRepository** 필드에서 사용됩니다. 값은 **hostname[:port]** 형식이어야 합니다.

8.4.2. 이미지 레지스트리 설정 구성

image.config.openshift.io/cluster 사용자 지정 리소스 (CR)를 편집하여 이미지 레지스트리 설정을 구성할 수 있습니다. MCO(Machine Config Operator)는 레지스트리에 대한 변경 사항이 있는지 image.config.openshift.io/cluster CR을 감시하고 변경 사항이 탐지되면 노드를 재부팅합니다.

절차

1. 다음과 같이 **project.config.openshift.io/cluster 사용자 정의 리소스를 편집합니다.**

```
$ oc edit image.config.openshift.io/cluster
```

다음은 **image.config.openshift.io/cluster CR의 예입니다.**

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: (2)
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: (3)
    name: myconfigmap
  registrySources: (4)
```

```
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1

Image: 이미지 처리 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 **cluster**입니다.

2

allowedRegistriesForImport: 일반 사용자가 이미지를 가져올 수 있는 컨테이너 이미지 레지스트리를 제한합니다. 이 목록은 유효한 이미지를 포함한다고 신뢰할 수 있으며 애플리케이션을 가져올 수 있도록 하려는 레지스트리로 설정합니다. 이미지를 생성할 권한이 있는 사용자 또는 **API**의 **ImageStreamMappings**는 이 정책의 영향을 받지 않습니다. 일반적으로 클러스터 관리자에게만 적절한 권한이 있습니다.

3

additionalTrustedCA: 이미지 스트림 가져오기, **pod** 이미지 가져오기, **openshift-image-registry** 풀스루 및 빌드 중에 신뢰해야 하는 추가 **CA** (인증 기관)가 포함된 구성 맵에 대한 참조입니다. 이 구성 맵의 네임스페이스는 **openshift-config**입니다. 구성 맵 형식에서는 신뢰할 추가 레지스트리 **CA**마다 레지스트리 호스트 이름을 키로 사용하고 **PEM** 인증서를 값으로 사용합니다.

4

registrySources: 빌드 및 **pod** 이미지에 액세스할 때 컨테이너 런타임에서 개별 레지스트리를 허용하는지 여부를 결정하는 구성이 포함되어 있습니다. **allowedRegistries** 매개변수 또는 **blockedRegistries** 매개변수 중 하나를 설정할 수 있지만 둘 다 설정할 수는 없습니다. 이미지 단축 이름을 사용하는 레지스트리를 허용하는 비보안 레지스트리 또는 레지스트리에 대한 액세스를 허용할지 여부를 정의할 수도 있습니다. 이 예에서는 사용할 수 있는 레지스트리를 정의하는 **allowedRegistries** 매개변수를 사용합니다. 비보안 레지스트리 **insecure.com** 도 허용됩니다. **registrySources paramter**에는 내부 클러스터 레지스트리에 대한 구성이 포함되어 있지 않습니다.

참고

allowedRegistries 매개변수가 정의되면 명시적으로 나열되지 않은 경우 **registry.redhat.io, quay.io** 레지스트리 및 기본 내부 이미지 레지스트리를 포함한 모든 레지스트리가 차단됩니다. 이 매개변수를 사용하는 경우 **Pod** 실패를 방지하기 위해 환경의 페이로드 이미지에서 필요한 **registry.redhat.io** 및 **quay.io** 레지스트리와 **internalRegistryHostname**을 **allowedRegistries** 목록에 추가해야 합니다. **registry.redhat.io** 및 **quay.io** 레지스트리를 **blockedRegistries** 목록에 추가하지 마십시오.

allowedRegistries, blockedRegistries 또는 **insecureRegistries** 매개변수를 사용하는 경우 레지스트리 내에서 개별 리포지토리를 지정할 수 있습니다. 예: **reg1.io/myrepo/myapp:latest.**

가능한 보안 위험을 줄이려면 안전하지 않은 외부 레지스트리의 사용을 피해야합니다.

2. 변경 사항이 적용되었는지 확인하려면 노드를 나열합니다.

```
$ oc get nodes
```

출력 예

```
NAME                           STATUS                     ROLES    AGE   VERSION
ci-ln-j5cd0qt-f76d1-vfj5x-master-0       Ready                      master   98m   v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-master-1       Ready,SchedulingDisabled   master   99m
v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-master-2       Ready                      master   98m   v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-worker-b-nsnd4   Ready                      worker   90m   v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-worker-c-5z2gz   NotReady,SchedulingDisabled   worker   90m
v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-worker-d-stsjv   Ready                      worker   90m   v1.22.1
```

허용, 차단 및 비보안 레지스트리 매개변수에 대한 자세한 내용은 이미지 레지스트리 설정 구성을 참조하십시오.

8.4.2.1. 이미지 레지스트리 액세스를 위한 추가 신뢰 저장소 구성

image.config.openshift.io/cluster 사용자 지정 리소스에는 이미지 레지스트리 액세스 중에 신뢰할 수 있는 추가 인증 기관이 포함된 구성 맵에 대한 참조가 포함될 수 있습니다.

사전 요구 사항

- 인증 기관(CA)은 PEM으로 인코딩되어야 합니다.

절차

openshift-config 네임 스페이스에 구성 맵을 만들고 **image.config.openshift.io** 사용자 지정 리소스에서 **AdditionalTrustedCA**의 해당 이름을 사용하여 외부 레지스트리에 연결할 때 신뢰할 수있는 추가 CA를 제공할 수 있습니다.

구성 맵 키는 이 CA가 신뢰할 수있는 포트가 있는 레지스트리의 호스트 이름이며 PEM 인증서 콘텐츠는 신뢰할 수 있는 각 추가 레지스트리 CA의 값입니다.

이미지 레지스트리 CA 구성 맵의 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1

레지스트리에 **registry-with-port.example.com:5000** 같은 포트가 있는 경우 **:**이 **..**로 교체되어야 합니다.

다음 절차에 따라 추가 CA를 구성할 수 있습니다.

1. 추가 CA를 구성하려면 다음을 실행합니다.

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -
n openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

8.4.2.2. 이미지 레지스트리 저장소 미러링 설정

컨테이너 레지스트리 저장소 미러링을 설정하면 다음을 수행할 수 있습니다.

- 소스 이미지 레지스트리의 저장소에서 이미지를 가져오기 위해 요청을 리디렉션하고 미러링된 이미지 레지스트리의 저장소에서 이를 해석하도록 OpenShift Container Platform 클러스터를 설정합니다.
- 하나의 미러가 다운된 경우 다른 미러를 사용할 수 있도록 각 대상 저장소에 대해 여러 미러링된 저장소를 확인합니다.

다음은 OpenShift Container Platform의 저장소 미러링의 몇 가지 속성입니다.

- 이미지 풀은 레지스트리 다운타임에 탄력적으로 대처할 수 있습니다.
- 연결이 끊긴 환경의 클러스터는 중요한 위치(예: quay.io)에서 이미지를 가져오고 회사 방화벽 뒤의 레지스트리에서 요청된 이미지를 제공하도록 할 수 있습니다.
- 이미지 가져오기 요청이 있으면 특정한 레지스트리 순서로 가져오기를 시도하며 일반적으로 영구 레지스트리는 마지막으로 시도합니다.
- 입력한 미러링 정보는 OpenShift Container Platform 클러스터의 모든 노드에서 /etc/containers/registries.conf 파일에 추가됩니다.
- 노드가 소스 저장소에서 이미지를 요청하면 요청된 컨텐츠를 찾을 때 까지 미러링된 각 저장

소를 차례로 시도합니다. 모든 미러가 실패하면 클러스터는 소스 저장소를 시도합니다. 성공하면 이미지를 노드로 가져올 수 있습니다.

저장소 미러링은 다음과 같은 방법으로 설정할 수 있습니다.

- **OpenShift Container Platform** 설치 시

  OpenShift Container Platform에 필요한 컨테이너 이미지를 가져온 다음 해당 이미지를 회사 방화벽 뒤에 배치하면 연결이 끊긴 환경에 있는 데이터 센터에 **OpenShift Container Platform**을 설치할 수 있습니다.

- **OpenShift Container Platform** 설치 후

  OpenShift Container Platform 설치 시 미러링을 설정하지 않고 **ImageContentSourcePolicy** 개체를 사용하여 나중에 설정할 수 있습니다.

다음 절차에서는 설치 후 미러 구성을 제공합니다. 이때 다음을 식별하는 **ImageContentSourcePolicy** 오브젝트를 생성할 수 있습니다.

- 미러링하려는 컨테이너 이미지 저장소의 소스
- 소스 저장소에서 요청된 컨텐츠를 제공하는 각 미러 저장소에 대한 개별 항목

참고

ImageContentSourcePolicy 개체가 있는 클러스터에 대한 글로벌 풀 시크릿만 구성할 수 있습니다. 프로젝트에 풀 시크릿을 추가할 수 없습니다.

사전 요구 사항

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

프로세스

1

1. 미러링된 저장소를 설정합니다.

   - Red Hat Quay Repository Mirroring에 설명된대로 Red Hat Quay를 사용하여 미러링된 저장소를 설정합니다. Red Hat Quay를 사용하면 한 저장소에서 다른 저장소로 이미지를 복사하고 시간이 지남에 따라 해당 저장소를 반복해서 자동으로 동기화할 수 있습니다.

   - skopeo와 같은 툴을 사용하여 소스 디렉토리에서 미러링된 저장소로 이미지를 수동으로 복사합니다.

     예를 들어, Red Hat Enterprise Linux(RHEL) 7 또는 RHEL 8 시스템에 skopeo RPM 패키지를 설치한 후 다음 예와 같이 skopeo 명령을 사용합니다.

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi8/ubi-minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455ebaa6 \
     docker://example.io/example/ubi-minimal
     ```

     이 예제에는 example.io라는 컨테이너 이미지 레지스트리가 있으며, registry.access.redhat.com에서 ubi8/ubi-minimal 이미지를 복사할 example이라는 이미지 저장소가 있습니다. 레지스트리를 생성한 후 OpenShift Container Platform 클러스터를 설정하여 소스 저장소의 요청을 미러링된 저장소로 리디렉션할 수 있습니다.

2. OpenShift Container Platform 클러스터에 로그인합니다.

3. ImageContentSourcePolicy 파일(예: registryrepomirror.yaml)을 생성하고 소스 및 미러를 특정 레지스트리 및 저장소 쌍과 이미지로 교체합니다.

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: ImageContentSourcePolicy
   metadata:
     name: ubi8repo
   spec:
     repositoryDigestMirrors:
     - mirrors:
       - example.io/example/ubi-minimal (1)
       - example.com/example/ubi-minimal (2)
       source: registry.access.redhat.com/ubi8/ubi-minimal (3)
     - mirrors:
       - mirror.example.com/redhat
       source: registry.redhat.io/openshift4 (4)
     - mirrors:
       - mirror.example.com
   ```

```
  source: registry.redhat.io 5
- mirrors:
  - mirror.example.net/image
  source: registry.example.com/example/myimage 6
- mirrors:
  - mirror.example.net
  source: registry.example.com/example 7
- mirrors:
  - mirror.example.net/registry-example-com
  source: registry.example.com 8
```

1

이미지 레지스트리 및 저장소의 이름을 가리킵니다.

2

각 대상 저장소에 대해 여러 미러 리포지토리를 나타냅니다. 하나의 미러가 다운된 경우 대상 저장소에서 다른 미러를 사용할 수 있습니다.

3

미러링된 컨텐츠를 포함하는 레지스트리 및 저장소를 가리킵니다.

4

해당 네임스페이스의 이미지를 사용하도록 레지스트리 내에서 네임스페이스를 구성할 수 있습니다. 레지스트리 도메인을 소스로 사용하는 경우 **ImageContentSourcePolicy** 리소스가 레지스트리의 모든 리포지토리에 적용됩니다.

5

레지스트리 이름을 구성하면 **ImageContentSourcePolicy** 리소스가 소스 레지스트리에서 미러 레지스트리로 모든 리포지토리에 적용됩니다.

6

mirror.example.net/image@sha256:... 이미지를 가져옵니다.

7

미러 **mirror.example.net/ myimage @sha256:...** 에서 소스 레지스트리 네임스페이스의 **myimage** 이미지를 가져옵니다.

8

미러 레지스트리 **mirror.example.net/registry-example-com/example/myimage@sha256:...** 에서 이미지 **registry.example.com/example/myimage /myimage/myimage/myimage@sha256**을 가

져옵니다. ImageContentSourcePolicy 리소스는 소스 레지스트리에서 미러 레지스트리 mirror.example.net/registry-example-com 으로 모든 리포지토리에 적용됩니다.

4. 새 ImageContentSourcePolicy 개체를 생성합니다.

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

ImageContentSourcePolicy 개체가 생성된 후 새 설정이 각 노드에 배포된 클러스터는 소스 저장소에 대한 요청에 미러링된 저장소를 사용하기 시작합니다.

5. 미러링된 설정이 적용되었는지 확인하려면 노드 중 하나에서 다음을 수행하십시오.

a. 노드를 나열합니다.

```
$ oc get node
```

출력 예

```
NAME                          STATUS                     ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                         worker   7m   v1.24.0
ip-10-0-138-148.ec2.internal   Ready                          master   11m  v1.24.0
ip-10-0-139-122.ec2.internal   Ready                          master   11m  v1.24.0
ip-10-0-147-35.ec2.internal    Ready                         worker   7m   v1.24.0
ip-10-0-153-12.ec2.internal    Ready                         worker   7m   v1.24.0
ip-10-0-154-10.ec2.internal    Ready                         master   11m  v1.24.0
```

Imagecontentsourcepolicy 리소스는 노드를 재시작하지 않습니다.

b. 디버깅 프로세스를 시작하고 노드에 액세스합니다.

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

출력 예

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

c. 루트 디렉토리를 **/host** 로 변경합니다.

```
sh-4.2# chroot /host
```

d. **/etc/containers/registries.conf** 파일을 체크하여 변경 사항이 적용되었는지 확인합니다.

```
sh-4.2# cat /etc/containers/registries.conf
```

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi8/ubi-minimal"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal"

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"

[[registry]]
  prefix = ""
  location = "registry.example.com"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"
  mirror-by-digest-only = true
```

```
    [[registry.mirror]]
      location = "mirror.example.net"

  [[registry]]
    prefix = ""
    location = "registry.example.com/example/myimage"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.net/image"

  [[registry]]
    prefix = ""
    location = "registry.redhat.io"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.com"

  [[registry]]
    prefix = ""
    location = "registry.redhat.io/openshift4"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.com/redhat"
```

e. 소스의 이미지 다이제스트를 노드로 가져와 실제로 미러링에 의해 해결되는지 확인합니다. **ImageContentSourcePolicy** 개체는 이미지 태그가 아닌 이미지 다이제스트만 지원합니다.

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455ebaa6
```

저장소 미러링 문제 해결

저장소 미러링 절차가 설명대로 작동하지 않는 경우 저장소 미러링 작동 방법에 대한 다음 정보를 사용하여 문제를 해결하십시오.

- 가져온 이미지는 첫 번째 작동 미러를 사용하여 공급합니다.
- 주요 레지스트리는 다른 미러가 작동하지 않는 경우에만 사용됩니다.

- 시스템 컨텍스트에서 **Insecure** 플래그가 폴백으로 사용됩니다.

- **/etc/containers/registries.conf** 파일 형식이 최근에 변경되었습니다. 현재 버전은 **TOML** 형식의 버전 **2**입니다.

8.5. 미러링된 OPERATOR 카탈로그에서 OPERATORHUB 채우기

연결이 끊긴 클러스터에 사용하기 위해 **Operator** 카탈로그를 미러링한 경우 미러링된 카탈로그에서 **Operator**로 **OperatorHub**를 채울 수 있습니다. 미러링 프로세스에서 생성된 매니페스트를 사용하여 필요한 **ImageContentSourcePolicy** 및 **CatalogSource** 오브젝트를 생성할 수 있습니다.

8.5.1. 사전 요구 사항

- 연결이 끊긴 클러스터와 함께 사용할 **Operator** 카탈로그 미러링

8.5.2. ImageContentSourcePolicy 오브젝트 생성

Operator 카탈로그 콘텐츠를 미러 레지스트리에 미러링한 후 필요한 **ImageContentSourcePolicy (ICSP)** 오브젝트를 생성합니다. **ICSP** 오브젝트는 **Operator** 매니페스트에 저장된 이미지 참조와 미러링된 레지스트리 간에 변환하도록 노드를 구성합니다.

절차

- 연결이 끊긴 클러스터에 액세스할 수 있는 호스트에서 매니페스트 디렉터리에 **imageContentSourcePolicy.yaml** 파일을 지정하도록 다음 명령을 실행하여 **ICSP**를 생성합니다.

```
$ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
```

여기서 **<path/to/manifests/dir>**은 미러링된 콘텐츠의 매니페스트 디렉터리 경로입니다.

이제 미러링된 인덱스 이미지 및 **Operator** 콘텐츠를 참조하도록 **CatalogSource** 오브젝트를 생성할 수 있습니다.

8.5.3. 클러스터에 카탈로그 소스 추가

OpenShift Container Platform 클러스터에 카탈로그 소스를 추가하면 사용자를 위한 **Operator**를 검

색하고 설치할 수 있습니다. 클러스터 관리자는 인덱스 이미지를 참조하는 **CatalogSource** 오브젝트를 생성할 수 있습니다. **OperatorHub**는 카탈로그 소스를 사용하여 사용자 인터페이스를 채웁니다.

사전 요구 사항

- 인덱스 이미지를 빌드하여 레지스트리로 내보냈습니다.

절차

1. 인덱스 이미지를 참조하는 **CatalogSource** 오브젝트를 생성합니다. **oc adm catalog mirror** 명령을 사용하여 카탈로그를 대상 레지스트리에 미러링한 경우 매니페스트 디렉터리에서 생성된 **catalogSource.yaml** 파일을 시작점으로 사용할 수 있습니다.

   a. 다음을 사양에 맞게 수정하고 **catalogsource.yaml** 파일로 저장합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog (1)
  namespace: openshift-marketplace (2)
spec:
  sourceType: grpc
  image: <registry>/<namespace>/redhat-operator-index:v4.9 (3)
  displayName: My Operator Catalog
  publisher: <publisher_name> (4)
  updateStrategy:
    registryPoll: (5)
      interval: 30m
```

(1) 레지스트리에 업로드하기 전에 콘텐츠를 로컬 파일에 미러링한 경우 오브젝트를 생성할 때 "잘못된 리소스 이름" 오류가 발생하지 않도록 **metadata.name** 필드에서 백슬래시(/) 문자를 제거합니다.

(2) 카탈로그 소스를 모든 네임스페이스의 사용자가 전역적으로 사용할 수 있도록 하려면 **openshift-marketplace** 네임스페이스를 지정합니다. 그러지 않으면 카탈로그의 범위가 지정되고 해당 네임스페이스에 대해서만 사용할 수 있도록 다른 네임스페이스를 지정할 수 있습니다.

(3) 인덱스 이미지를 지정합니다.

4

카탈로그를 게시하는 이름 또는 조직 이름을 지정합니다.

5

카탈로그 소스는 새 버전을 자동으로 확인하여 최신 상태를 유지할 수 있습니다.

b. 파일을 사용하여 **CatalogSource** 오브젝트를 생성합니다.

```
$ oc apply -f catalogSource.yaml
```

2. 다음 리소스가 성공적으로 생성되었는지 확인합니다.

a. **Pod**를 확인합니다.

```
$ oc get pods -n openshift-marketplace
```

출력 예

```
NAME                                  READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6             1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. 카탈로그 소스를 확인합니다.

```
$ oc get catalogsource -n openshift-marketplace
```

출력 예

```
NAME              DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc          5s
```

c. 패키지 매니페스트 확인합니다.

```
$ oc get packagemanifest -n openshift-marketplace
```

출력 예

```
NAME                  CATALOG               AGE
jaeger-product        My Operator Catalog   93s
```

이제 OpenShift Container Platform 웹 콘솔의 OperatorHub 페이지에서 Operator를 설치할 수 있습니다.

- 인덱스 이미지가 프라이빗 레지스트리에서 호스팅되고 인증이 필요한 경우 프라이빗 레지스트리에서 Operator용 이미지에 액세스를 참조하십시오.
- Kubernetes 버전 기반 이미지 태그를 사용하여 클러스터 업그레이드 후 인덱스 이미지 버전을 자동으로 업데이트하려면 사용자 정의 카탈로그 소스에 대한 이미지 템플릿을 참조하십시오.

8.6. OPERATORHUB를 통한 OPERATOR 설치 정보

OperatorHub는 Operator를 검색하는 사용자 인터페이스입니다. 이는 클러스터에 Operator를 설치하고 관리하는 OLM(Operator Lifecycle Manager)과 함께 작동합니다.

클러스터 관리자는 OpenShift Container Platform CLI 또는 웹 콘솔을 사용하여 OperatorHub에서 Operator를 설치할 수 있습니다. 그런 다음 Operator를 하나 이상의 네임 스페이스에 가입시켜 Operator를 클러스터의 개발자가 사용할 수 있도록 합니다.

설치하는 동안 Operator의 다음 초기 설정을 결정해야합니다.

설치 모드

All namespaces on the cluster (default)를 선택하여 **Operator**를 모든 네임 스페이스에 설치하거나 사용 가능한 경우 개별 네임 스페이스를 선택하여 선택한 네임 스페이스에만 **Operator**를 설치합니다. 이 예에서는 모든 사용자와 프로젝트 **Operator**를 사용할 수 있도록 **All namespaces...** 선택합니다.

업데이트 채널

여러 채널을 통해 **Operator**를 사용할 수있는 경우 구독할 채널을 선택할 수 있습니다. 예를 들어, **stable** 채널에서 배치하려면 (사용 가능한 경우) 목록에서 해당 채널을 선택합니다.

승인 전략

자동 또는 수동 업데이트를 선택할 수 있습니다.

설치된 **Operator**에 대해 자동 업데이트를 선택하는 경우 선택한 채널에 해당 **Operator**의 새 버전이 제공되면 **OLM(Operator Lifecycle Manager)**에서 **Operator**의 실행 중인 인스턴스를 개입 없이 자동으로 업그레이드합니다.

수동 업데이트를 선택하면 최신 버전의 **Operator**가 사용 가능할 때 **OLM**이 업데이트 요청을 작성합니다. 클러스터 관리자는 **Operator**를 새 버전으로 업데이트하려면 **OLM** 업데이트 요청을 수동으로 승인해야 합니다.

8.6.1. 웹 콘솔을 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하여 **OperatorHub**에서 **Operator**를 설치하고 구독할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.

프로세스

1. 웹 콘솔에서 **Operators → OperatorHub** 페이지로 이동합니다.

2. 원하는 **Operator**를 찾으려면 키워드를 **Filter by keyword** 상자에 입력하거나 스크롤합니다. 예를 들어, **Jaeger Operator**를 찾으려면 **jaeger** 를 입력합니다.

   인프라 기능에서 옵션을 필터링할 수 있습니다. 예를 들어, 연결이 끊긴 환경 (제한된 네트워

크 환경이라고도 함)에서 작업하는 **Operator**를 표시하려면 **Disconnected**를 선택합니다.

3. **Operator**를 선택하여 추가 정보를 표시합니다.

참고

커뮤니티 **Operator**를 선택하면 **Red Hat**이 커뮤니티 **Operator**를 인증하지 않는다고 경고합니다. 계속하기 전에 경고를 확인해야합니다.

4. **Operator**에 대한 정보를 확인하고 **Install**을 클릭합니다.

5. **Operator** 설치 페이지에서 다음을 수행합니다.

   a. 다음 명령 중 하나를 선택합니다.

   - **All namespaces on the cluster (default)**에서는 기본 **openshift-operators** 네임스페이스에 **Operator**가 설치되므로 **Operator**가 클러스터의 모든 네임스페이스를 모니터링하고 사용할 수 있습니다. 이 옵션을 항상 사용할 수있는 것은 아닙니다.
   - **A specific namespace on the cluster**를 사용하면 **Operator**를 설치할 특정 단일 네임 스페이스를 선택할 수 있습니다. **Operator**는 이 단일 네임 스페이스에서만 모니터링 및 사용할 수 있게 됩니다.

   b. **Update Channe**을 선택합니다 (하나 이상이 사용 가능한 경우).

   c. 앞에서 설명한 대로 자동 또는 수동 승인 전략을 선택합니다.

6. 이 **OpenShift Container Platform** 클러스터에서 선택한 네임스페이스에서 **Operator**를 사용할 수 있도록 하려면 설치를 클릭합니다.

   a. 수동 승인 전략을 선택한 경우 설치 계획을 검토하고 승인할 때까지 서브스크립션의 업그레이드 상태가 업그레이드 중으로 유지됩니다.

Install Plan 페이지에서 승인 한 후 **subscription** 업그레이드 상태가 **Up to date**로 이동합니다.

b. 자동 승인 전략을 선택한 경우 업그레이드 상태가 개입 없이 최신 상태로 확인되어야 합니다.

7. 서브스크립션의 업그레이드 상태가 최신이면 **Operator → 설치된 Operator**를 선택하여 설치된 **Operator**의 **CSV**(클러스터 서비스 버전)가 최종적으로 표시되는지 확인합니다. 상태는 최종적으로 관련 네임스페이스에서 **InstallSucceeded**로 확인되어야 합니다.

참고

모든 네임스페이스... 설치 모드의 경우, **openshift-operators** 네임스페이스에서 상태가 **InstallSucceeded**로 확인되지만 다른 네임스페이스에서 확인하면 상태가 복사됩니다.

그렇지 않은 경우 다음을 수행합니다.

a. 워크로드 → **Pod** 페이지의 **openshift-operators** 프로젝트(또는 특정 네임스페이스...설치 모드가 선택된 경우 기타 관련 네임스페이스)에서 문제를 보고하는 모든 **Pod**의 로그를 확인하여 문제를 추가로 해결합니다.

8.6.2. CLI를 사용하여 OperatorHub에서 설치

OpenShift Container Platform 웹 콘솔을 사용하는 대신 **CLI**를 사용하여 **OperatorHub**에서 **Operator**를 설치할 수 있습니다. **oc** 명령을 사용하여 **Subscription** 개체를 만들거나 업데이트합니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- 로컬 시스템에 **oc** 명령을 설치합니다.

프로세스

1. OperatorHub에서 클러스터에 사용 가능한 Operator의 목록을 표시합니다.

```
$ oc get packagemanifests -n openshift-marketplace
```

출력 예

```
NAME                               CATALOG               AGE
3scale-operator                    Red Hat Operators     91m
advanced-cluster-management        Red Hat Operators     91m
amq7-cert-manager                  Red Hat Operators     91m
...
couchbase-enterprise-certified     Certified Operators   91m
crunchy-postgres-operator          Certified Operators   91m
mongodb-enterprise                 Certified Operators   91m
...
etcd                               Community Operators   91m
jaeger                             Community Operators   91m
kubefed                            Community Operators   91m
...
```

필요한 Operator의 카탈로그를 기록해 둡니다.

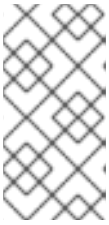
2. 필요한 Operator를 검사하여 지원되는 설치 모드 및 사용 가능한 채널을 확인합니다.

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

3. OperatorGroup 오브젝트로 정의되는 Operator group에서 Operator group과 동일한 네임스페이스에 있는 모든 Operator에 대해 필요한 RBAC 액세스 권한을 생성할 대상 네임스페이스를 선택합니다.

Operator를 서브스크립션하는 네임스페이스에는 Operator의 설치 모드, 즉 AllNamespaces 또는 SingleNamespace 모드와 일치하는 Operator group이 있어야 합니다. 설치하려는 Operator에서 AllNamespaces를 사용하는 경우 openshift-operators 네임스페이스에 적절한 Operator group이 이미 있습니다.

그러나 Operator에서 SingleNamespace 모드를 사용하고 적절한 Operator group이 없는 경우 이를 생성해야 합니다.



참고

이 프로세스의 웹 콘솔 버전에서는 **SingleNamespace** 모드를 선택할 때 자동으로 **OperatorGroup** 및 **Subscription** 개체 생성을 자동으로 처리합니다.

a. **OperatorGroup** 개체 **YAML** 파일을 만듭니다 (예: **operatorgroup.yaml**).

OperatorGroup 오브젝트의 예

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace>
spec:
  targetNamespaces:
  - <namespace>
```

b. **OperatorGroup** 개체를 생성합니다.

```
$ oc apply -f operatorgroup.yaml
```

4. **Subscription** 개체 **YAML** 파일을 생성하여 **OpenShift Pipelines Operator**에 네임스페이스를 등록합니다(예: **sub.yaml**).

Subscription 개체 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace 5
```

```
config:
  env: 6
  - name: ARGS
    value: "-v=10"
  envFrom: 7
  - secretRef:
      name: license-secret
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1

AllNamespaces 설치 모드를 사용하려면 **openshift-operators** 네임스페이스를 지정합니다. 그 외에는 **SingleNamespace** 설치 모드를 사용하도록 관련 단일 네임스페이스를 지정합니다.

2

등록할 채널의 이름입니다.

3

등록할 **Operator**의 이름입니다.

4

Operator를 제공하는 카탈로그 소스의 이름입니다.

5

6

env 매개변수는 **OLM**에서 생성한 **Pod**의 모든 컨테이너에 존재해야 하는 환경 변수 목록을 정의합니다.

7

envFrom 매개 변수는 컨테이너에 환경 변수를 채울 소스 목록을 정의합니다.

8

volumes 매개변수는 **OLM**에서 생성한 **Pod**에 있어야 하는 볼륨 목록을 정의합니다.

9

volumeMounts 매개변수는 **OLM**에서 생성한 **Pod**의 모든 컨테이너에 존재해야 하는 **VolumeMounts** 목록을 정의합니다. **volumeMount** 가 존재하지 않는 볼륨을 참조하는 경우 **OLM**에서 **Operator**를 배포하지 못합니다.

10

tolerations 매개변수는 **OLM**에서 생성한 **Pod**의 허용 오차 목록을 정의합니다.

11

resources 매개변수는 **OLM**에서 생성한 **Pod**의 모든 컨테이너에 대한 리소스 제약 조건을 정의합니다.

12

nodeSelector 매개변수는 **OLM**에서 생성한 **Pod**에 대한 **NodeSelector** 를 정의합니다.

5. **Subscription** 오브젝트를 생성합니다.

```
$ oc apply -f sub.yaml
```

이 시점에서 **OLM**은 이제 선택한 **Operator**를 인식합니다. **Operator**의 **CSV**(클러스터 서비스 버전)가 대상 네임스페이스에 표시되고 **Operator**에서 제공하는 **API**를 생성에 사용할 수 있어야 합니다.

추가 리소스

- About OperatorGroups

9장. 경고 알림 구성

OpenShift Container Platform에서는 경고 규칙에 정의된 조건이 true이면 경고가 실행됩니다. 경고는 클러스터 내에서 일련의 상황이 발생한다는 통지를 제공합니다. 기본적으로 OpenShift Container Platform 웹 콘솔의 알림 UI에서 실행 경고가 표시됩니다. 설치 후 OpenShift Container Platform을 구성하여 외부 시스템에 경고 알림을 보낼 수 있습니다.

9.1. 외부 시스템에 알림 전송

OpenShift Container Platform 4.9에서는 알림 UI에서 실행 경고를 볼 수 있습니다. 알림은 기본적으로 모든 알림 시스템으로 전송되지 않습니다. 다음 수신자 유형으로 알림을 전송하도록 OpenShift Container Platform을 구성할 수 있습니다.

- PagerDuty
- Webhook
- 이메일
- Slack

알림을 수신기로 라우팅하면 오류가 발생할 때 적절한 팀에게 적절한 알림을 보낼 수 있습니다. 예를 들어, 심각한 경고는 즉각적인 주의가 필요하며 일반적으로 개인 또는 문제 대응팀으로 호출됩니다. 심각하지 않은 경고 알림을 제공하는 경고는 즉각적이지 않은 검토를 위해 티켓팅 시스템으로 라우팅할 수 있습니다.

워치독 경고를 사용하여 해당 경고가 제대로 작동하는지 확인

OpenShift Container Platform 모니터링에는 지속적으로 트리거되는 워치독 경고가 포함되어 있습니다. Alertmanager는 구성된 알림 공급자에게 워치독 경고 알림을 반복적으로 보냅니다. 일반적으로 공급자는 워치독 경고를 수신하지 않을 때 관리자에게 알리도록 구성됩니다. 이 메커니즘을 사용하면 Alertmanager와 알림 공급자 간의 모든 통신 문제를 빠르게 식별할 수 있습니다.

9.1.1. 경고 수신자 구성

클러스터의 중요한 문제를 파악할 수 있도록 경고 수신자를 설정할 수 있습니다.

사전 요구 사항

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

절차

1. 관리자 관점에서 관리 → 클러스터 설정 → 구성 → **Alertmanager** 로 이동합니다.

참고

또는 알림 창을 통해 동일한 페이지로 이동할 수 있습니다. **OpenShift Container Platform** 웹 콘솔의 오른쪽 상단에서 호출 아이콘을 선택하고 **AlertmanagerReceiverNotConfigured** 경고에서 구성을 선택합니다.

2. 이 페이지의 수신자 섹션에서 수신자 만들기를 선택합니다.

3. 수신자 만들기에서 수신자 이름을 추가하고 목록에서 수신자 유형을 선택합니다.

4. 수신자 구성을 편집합니다.

- **PagerDuty** 수신자의 경우:

a. 통합 유형을 선택하고 **PagerDuty** 통합 키를 추가합니다.

b. **PagerDuty** 설치의 **URL**을 추가합니다.

c. 클라이언트와 인스턴스 세부 정보 또는 심각도 사양을 편집하려면 고급 설정 표시를 선택합니다.

- **Webhook** 수신자의 경우:

a. **HTTP POST** 요청이 전송되는 끝점을 추가합니다.

b. 해결된 경보를 수신자에게 보내는 기본 옵션을 편집하려면 고급 설정 표시를 선택합니다.

- 이메일 수신자의 경우:

a. 알림을 받을 이메일 주소를 추가합니다.

b. 알림을 전송할 주소, 이메일 전송에 사용되는 스마트 호스트 및 포트 번호, SMTP 서버의 호스트 이름, 인증 세부 정보를 포함하여 SMTP 구성 세부 정보를 추가합니다.

c. TLS가 필요한지 여부를 선택합니다.

d. 해결된 경고를 수신자에게 보내지 않도록 기본 옵션을 편집하거나 이메일 알림 구성을 편집하려면 고급 설정 표시를 선택합니다.

- Slack 수신자의 경우:

a. Slack Webhook의 URL을 추가합니다.

b. 알림을 보낼 Slack 채널 또는 사용자 이름을 추가합니다.

c. 해결된 경고를 수신자에게 보내지 않도록 기본 옵션을 편집하거나 아이콘 및 사용자 이름 구성을 편집하려면 고급 설정 표시를 선택합니다. 채널 이름과 사용자 이름을 찾고 연결할지 여부를 선택할 수도 있습니다.

5. 기본적으로 모든 선택 항목과 일치하는 라벨을 사용하여 경고를 수신자에게 보냅니다. 수신자로 전송되기 전에 실행 경고에 대한 라벨 값을 정확히 일치시키려면 다음을 수행하십시오.

a. 양식의 라우팅 라벨 섹션에 라우팅 라벨 이름과 값을 추가합니다.

b. 정규식을 사용하려면 정규식을 선택합니다.

c.

라벨 추가를 선택하여 추가 라우팅 라벨을 추가합니다.

6.

Create을 선택하여 수신자를 생성합니다.

9.2. 추가 리소스

- 모니터링 개요
- 알림 관리

10장. 연결된 클러스터를 연결이 끊긴 클러스터로 변환

OpenShift Container Platform 클러스터를 연결된 클러스터에서 연결이 끊긴 클러스터로 변환해야 하는 몇 가지 시나리오가 있을 수 있습니다.

제한된 클러스터라고도 하는 연결이 끊긴 클러스터는 인터넷에 연결되어 있지 않습니다. 따라서 레지스트리 및 설치 미디어의 내용을 미러링해야 합니다. 인터넷과 폐쇄 네트워크에 모두 액세스할 수 있는 호스트에 미러 레지스트리를 생성하거나 네트워크 경계를 이동할 수 있는 장치에 이미지를 복사할 수 있습니다.

이 주제에서는 연결된 기존 클러스터를 연결이 끊긴 클러스터로 변환하는 일반 프로세스에 대해 설명합니다.

10.1. 미러 레지스트리 정보

OpenShift Container Platform 설치 및 후속 제품 업데이트에 Red Hat Quay, JFrog Artifactory, Sonatype Nexus Repository 또는 Harbor와 같은 컨테이너 미러 레지스트리에 필요한 이미지를 미러링할 수 있습니다. 대규모 컨테이너 레지스트리에 액세스할 수 없는 경우 OpenShift Container Platform 서브스크립션에 포함된 소규모 컨테이너 *레지스트리인 Red Hat OpenShift에 미러* 레지스트리를 사용할 수 있습니다.

Docker v2-2, *Red Hat OpenShift의 미러 레지스트리*, Artifactory, Sonatype Nexus Repository 또는 Harbor와 같은 Docker v2-2를 지원하는 모든 컨테이너 레지스트리를 사용할 수 있습니다. 선택한 레지스트리에 관계없이 인터넷상의 Red Hat 호스팅 사이트의 콘텐츠를 격리된 이미지 레지스트리로 미러링하는 절차는 동일합니다. 콘텐츠를 미러링한 후 미러 레지스트리에서 이 콘텐츠를 검색하도록 각 클러스터를 설정합니다.

중요

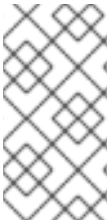
OpenShift Container Platform 클러스터의 내부 레지스트리는 미러링 프로세스 중에 필요한 태그 없이 푸시를 지원하지 않으므로 대상 레지스트리로 사용할 수 없습니다.

Red Hat OpenShift의 미러 레지스트리가 아닌 컨테이너 레지스트리를 선택하는 경우 프로비저닝하는 클러스터의 모든 시스템에서 액세스할 수 있어야 합니다. 레지스트리에 연결할 수 없는 경우 설치, 업데이트 또는 워크로드 재배치와 같은 일반 작업이 실패할 수 있습니다. 따라서 고가용성 방식으로 미러 레지스트리를 실행해야하며 미러 레지스트리는 최소한 OpenShift Container Platform 클러스터의 프로덕션 환경의 가용성조건에 일치해야 합니다.

미러 레지스트리를 OpenShift Container Platform 이미지로 채우면 다음 두 가지 시나리오를 수행할

수 있습니다. 호스트가 인터넷과 미러 레지스트리에 모두 액세스할 수 있지만 클러스터 노드에 액세스 할 수 없는 경우 해당 머신의 콘텐츠를 직접 미러링할 수 있습니다. 이 프로세스를 ***connected mirroring***(미러링 연결)이라고 합니다. 그러한 호스트가 없는 경우 이미지를 파일 시스템에 미러링한 다음 해당 호스트 또는 이동식 미디어를 제한된 환경에 배치해야 합니다. 이 프로세스를 *미러링 연결 해제*라고 합니다.

미러링된 레지스트리의 경우 가져온 이미지의 소스를 보려면 **CRI-O** 로그의 **Trying to access** 로그 항목을 검토해야 합니다. 노드에서 **crictl images** 명령을 사용하는 등의 이미지 가져오기 소스를 보는 다른 방법은 미러링되지 않은 이미지 이름을 표시합니다.



참고

Red Hat은 **OpenShift Container Platform**에서 타사 레지스트리를 테스트하지 않습니다.

10.2. 사전 요구 사항

- **oc** 클라이언트가 설치되어 있어야 합니다.
- 실행 중인 클러스터가 있어야 합니다.
- 설치된 미러 레지스트리: 다음 레지스트리 중 하나와 같이 **OpenShift Container Platform** 클러스터를 호스팅할 위치에서 **Docker v2-2** 를 지원하는 컨테이너 이미지 레지스트리입니다.
  - **Red Hat Quay**
  - **JFrog Artifactory**
  - **Sonatype Nexus Repository**
  - **Harbor**

Red Hat Quay에 대한 서브스크립션이 있는 경우 개념 증명 목적으로 또는 **Quay Operator**를 사용하여 **Red Hat Quay** 배포에 대한 설명서를 참조하십시오.

이미지를 공유하도록 미러 저장소를 구성해야 합니다. 예를 들어 이미지를 공유하려면 Red Hat Quay 리포지토리에 조직이 필요합니다.

- 필요한 컨테이너 이미지를 얻으려면 인터넷에 액세스합니다.

10.3. 미러링을 위해 클러스터 준비

클러스터의 연결을 해제하기 전에 연결이 끊긴 클러스터의 모든 노드에서 액세스할 수 있는 미러 레지스트리에 이미지를 미러링하거나 복사해야 합니다. 이미지를 미러링하려면 다음을 통해 클러스터를 준비해야 합니다.

- 미러 레지스트리 인증서를 호스트의 신뢰할 수 있는 CA 목록에 추가합니다.
- cloud . openshift.com 토큰에서 가져온 이미지 가져오기 보안이 포함된. dockerconfigjson 파일을 생성합니다.

절차

1. 이미지 미러링을 허용하는 인증 정보 설정:

   a. 단순한 PEM 또는 DER 파일 형식에 미러 레지스트리의 CA 인증서를 신뢰할 수 있는 CA 목록에 추가합니다. 예를 들면 다음과 같습니다.

   ```
   $ cp </path/to/cert.crt> /usr/share/pki/ca-trust-source/anchors/
   ```

   다음과 같습니다., </path/to/cert.crt>

   로컬 파일 시스템의 인증서 경로를 지정합니다.

   b. CA 트러스트를 업데이트합니다. 예를 들어 Linux에서 다음을 수행합니다.

   ```
   $ update-ca-trust
   ```

   c. 글로벌 가져오기 보안에서 .dockerconfigjson 파일을 추출합니다.

   ```
   $ oc extract secret/pull-secret -n openshift-config --confirm --to=.
   ```

출력 예

```
.dockerconfigjson
```

d. **.dockerconfigjson** 파일을 편집하여 미러 레지스트리 및 인증 자격 증명을 추가하고 새 파일로 저장합니다.

```
{"auths":{"<local_registry>": {"auth": "<credentials>","email":
"you@example.com"}}},"<registry>:<port>/<namespace>/":{"auth":"<token>"}}}
```

다음과 같습니다.

<local_registry>

미러 레지스트리가 콘텐츠를 제공하는 데 사용하는 레지스트리 도메인 이름 및 선택적으로 포트를 지정합니다.

auth

미러 레지스트리의 **base64**로 인코딩된 사용자 이름 및 암호를 지정합니다.

<registry>:<port>/<namespace>

미러 레지스트리 세부 정보를 지정합니다.

<token>

미러 레지스트리에 대한 **base64** 인코딩 **username:password** 를 지정합니다.

예를 들면 다음과 같습니다.

```
$ {"auths":{"cloud.openshift.com":
{"auth":"b3BlbnNoaWZ0Y3UjhGOVZPT0lOMEFaUjdPUzRGTA==","email":"user
@example.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UtZGOVZPT0lOMEFaUGSTd4VGVGVUjdP
UzRGTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VHkxOSTd4VGVGVU1MdTpleUpoYkdj
aUailA==","email":"user@example.com"},
```

```
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VH3BGSTd4VGVGVU1MdTpleUpoYkdj
aU9fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwWVFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwa
jJhN1ZXeTRV"},"my-registry:5000/my-namespace/":
{"auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}
```

10.4. 이미지 미러링

클러스터가 올바르게 구성되면 외부 저장소에서 미러 저장소로 이미지를 미러링할 수 있습니다.

절차

1. OLM(Operator Lifecycle Manager) 이미지를 미러링합니다.

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v{product-
version} <mirror_registry>:<port>/olm -a <reg_creds>
```

다음과 같습니다.

product-version

설치할 OpenShift Container Platform 버전에 해당하는 태그를 지정합니다(예: 4.8).

mirror_registry

Operator 콘텐츠를 미러링할 대상 레지스트리 및 네임스페이스의 정규화된 도메인 이름(FQDN)을 지정합니다. 여기서 <namespace> 는 레지스트리의 기존 네임스페이스입니다.

reg_creds

modified .dockerconfigjson 파일의 위치를 지정합니다.

예를 들면 다음과 같습니다.

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

2. 다른 Red Hat 제공 Operator의 콘텐츠를 미러링합니다.

```
$ oc adm catalog mirror <index_image> <mirror_registry>:<port>/<namespace> -a
<reg_creds>
```

다음과 같습니다.

index_image

미러링할 카탈로그의 인덱스 이미지를 지정합니다. 예를 들어 이전에 생성한 정리된 인덱스 이미지 또는 {index-image-pullspec} 과 같은 기본 카탈로그의 소스 인덱스 이미지 중 하나일 수 있습니다.

mirror_registry

Operator 콘텐츠를 미러링할 대상 레지스트리 및 네임스페이스의 FQDN을 지정합니다. 여기서 <namespace> 는 레지스트리의 기존 네임스페이스입니다.

reg_creds

선택 사항: 필요한 경우 레지스트리 자격 증명 파일의 위치를 지정합니다.

예를 들면 다음과 같습니다.

```
$ oc adm catalog mirror registry.redhat.io/redhat/community-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

3. OpenShift Container Platform 이미지 저장소를 미러링합니다.

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-
dev/ocp-release:v<product-version>-<architecture> --to=
<local_registry>/<local_repository> --to-release-image=
<local_registry>/<local_repository>:v<product-version>-<architecture>
```

다음과 같습니다.

product-version

설치할 OpenShift Container Platform 버전에 해당하는 태그를 지정합니다(예: 4.8.15-x86_64).

아키텍처

서버의 아키텍처 유형(예: x86_64)을 지정합니다.

local_registry

미러 저장소의 레지스트리 도메인 이름을 지정합니다.

local_repository

레지스트리에 작성할 리포지토리의 이름을 지정합니다(예: **ocp4/openshift 4**).

예를 들면 다음과 같습니다.

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-
dev/ocp-release:4.8.15-x86_64 --to=mirror.registry.com:443/ocp/release --to-release-
image=mirror.registry.com:443/ocp/release:4.8.15-x86_64
```

출력 예

```
info: Mirroring 109 images to mirror.registry.com/ocp/release ...
mirror.registry.com:443/
  ocp/release
 manifests:
   sha256:086224cadce475029065a0efc5244923f43fb9bb3bb47637e0aaf1f32b9cad47 ->
4.8.15-x86_64-thanos
   sha256:0a214f12737cb1cfbec473cc301aa2c289d4837224c9603e99d1e90fc00328db ->
4.8.15-x86_64-kuryr-controller
   sha256:0cf5fd36ac4b95f9de506623b902118a90ff17a07b663aad5d57c425ca44038c ->
4.8.15-x86_64-pod
   sha256:0d1c356c26d6e5945a488ab2b050b75a8b838fc948a75c0fa13a9084974680cb ->
4.8.15-x86_64-kube-client-agent

…..
sha256:66e37d2532607e6c91eedf23b9600b4db904ce68e92b43c43d5b417ca6c8e63c
mirror.registry.com:443/ocp/release:4.5.41-multus-admission-controller
sha256:d36efdbf8d5b2cbc4dcdbd64297107d88a31ef6b0ec4a39695915c10db4973f1
mirror.registry.com:443/ocp/release:4.5.41-cluster-kube-scheduler-operator
sha256:bd1baa5c8239b23ecdf76819ddb63cd1cd6091119fecdbf1a0db1fb3760321a2
mirror.registry.com:443/ocp/release:4.5.41-aws-machine-controllers
info: Mirroring completed in 2.02s (0B/s)

Success
Update image:  mirror.registry.com:443/ocp/release:4.5.41-x86_64
Mirror prefix: mirror.registry.com:443/ocp/release
```

4. 필요에 따라 다른 레지스트리를 미러링합니다.

```
$ oc image mirror <online_registry>/my/image:latest <mirror_registry>
```

추가 정보

- Operator 카탈로그 미러링에 대한 자세한 내용은 Operator 카탈로그 미러링을 참조하십시오.
- oc adm catalog mirror 명령에 대한 자세한 내용은 OpenShift CLI 관리자 명령 참조를 참조하십시오.

10.5. 미러 레지스트리의 클러스터 구성

미러 레지스트리에 이미지를 생성하고 미러링한 후 Pod가 미러 레지스트리에서 이미지를 가져올 수 있도록 클러스터를 수정해야 합니다.

다음을 수행해야 합니다.

- 글로벌 풀 시크릿에 미러 레지스트리 인증 정보를 추가합니다.
- 미러 레지스트리 서버 인증서를 클러스터에 추가합니다.
- 미러 레지스트리를 소스 레지스트리와 연결하는 ImageContentSourcePolicy 사용자 지정 리소스(MCP)를 생성합니다.

1. 클러스터 글로벌 pull-secret에 미러 레지스트리 인증 정보를 추가합니다.

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

1. 새 풀 시크릿 파일의 경로를 제공합니다.

예를 들면 다음과 같습니다.

```
$ oc set data secret/pull-secret -n openshift-config --from-
file=.dockerconfigjson=.mirrorsecretconfigjson
```

2. CA 서명된 미러 레지스트리 서버 인증서를 클러스터의 노드에 추가합니다.

   a. 미러 레지스트리의 서버 인증서가 포함된 구성 맵 생성

   ```
   $ oc create configmap <config_map_name> --from-file=
   <mirror_address_host>..<port>=$path/ca.crt -n openshift-config
   ```

   예를 들면 다음과 같습니다.

   ```
   S oc create configmap registry-config --from-
   file=mirror.registry.com..443=/root/certs/ca-chain.cert.pem -n openshift-config
   ```

   b. 구성 맵을 사용하여 image.config.openshift.io/cluster 사용자 정의 리소스(CR)를 업데이트합니다. OpenShift Container Platform은 이 CR의 변경 사항을 클러스터의 모든 노드에 적용합니다.

   ```
   $ oc patch image.config.openshift.io/cluster --patch '{"spec":
   {"additionalTrustedCA":{"name":"<config_map_name>"}}}' --type=merge
   ```

   예를 들면 다음과 같습니다.

   ```
   $ oc patch image.config.openshift.io/cluster --patch '{"spec":
   {"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
   ```

3. 온라인 레지스트리에서 미러 레지스트리로 컨테이너 가져오기 요청을 리디렉션하려면 ICSP를 생성합니다.

   a. ImageContentSourcePolicy 사용자 정의 리소스를 생성합니다.

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: ImageContentSourcePolicy
   metadata:
     name: mirror-ocp
   ```

```
spec:
  repositoryDigestMirrors:
  - mirrors:
    - mirror.registry.com:443/ocp/release 1
    source: quay.io/openshift-release-dev/ocp-release 2
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1 미러 이미지 레지스트리 및 저장소의 이름을 지정합니다.

2 미러링된 콘텐츠를 포함하는 온라인 레지스트리 및 저장소를 지정합니다.

b. ICSP 오브젝트를 생성합니다.

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

출력 예

```
imagecontentsourcepolicy.operator.openshift.io/mirror-ocp created
```

OpenShift Container Platform은 이 CR에 대한 변경 사항을 클러스터의 모든 노드에 적용합니다.

4. 미러 레지스트리에 대한 인증 정보, CA 및 ICSP가 추가되었는지 확인합니다.

a. 노드에 로그인합니다.

```
$ oc debug node/<node_name>
```

b. 디버그 쉘 내에서 /host를 root 디렉터리로 설정합니다.

-

```
sh-4.4# chroot /host
```

c. 인증 정보가 있는지 **config.json** 파일을 확인합니다.

```
sh-4.4# cat /var/lib/kubelet/config.json
```

출력 예

```
{"auths":{"brew.registry.redhat.io":{"xx=="},"brewregistry.stage.redhat.io":
{"auth":"xxx=="},"mirror.registry.com:443":{"auth":"xx="}}} 1
```

1 미러 레지스트리 및 인증 정보가 있는지 확인합니다.

d. **certs.d** 디렉터리로 변경합니다.

```
sh-4.4# cd /etc/docker/certs.d/
```

e. **certs.d** 디렉터리에 인증서를 나열합니다.

```
sh-4.4# ls
```

출력 예

```
image-registry.openshift-image-registry.svc.cluster.local:5000
image-registry.openshift-image-registry.svc:5000
mirror.registry.com:443 1
```

1

미러 레지스트리가 목록에 있는지 확인합니다.

f. ICSP에서 미러 레지스트리를 registries.conf 파일에 추가했는지 확인합니다.

```
sh-4.4# cat /etc/containers/registries.conf
```

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-release"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"

[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-v4.0-art-dev"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"
```

registry.mirror 매개변수는 미러 레지스트리가 원래 레지스트리보다 먼저 검색되었음을 나타냅니다.

g. 노드를 종료합니다.

```
sh-4.4# exit
```

10.6. 애플리케이션이 계속 작동하도록 보장

네트워크에서 클러스터 연결을 해제하기 전에 클러스터가 예상대로 작동하고 모든 애플리케이션이 예상대로 작동하는지 확인합니다.

절차

다음 명령을 사용하여 클러스터 상태를 확인합니다.

- Pod가 실행 중인지 확인합니다.

```
$ oc get pods --all-namespaces
```

출력 예

```
NAMESPACE                                  NAME                                                READY
STATUS     RESTARTS   AGE
kube-system                                apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-
master-0        1/1     Running    0          39m
kube-system                                apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-
master-1        1/1     Running    0          39m
kube-system                                apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-
master-2        1/1     Running    0          39m
openshift-apiserver-operator                   openshift-apiserver-operator-79c7c646fd-
5rvr5           1/1     Running    3          45m
openshift-apiserver                        apiserver-b944c4645-q694g
2/2     Running    0          29m
openshift-apiserver                        apiserver-b944c4645-shdxb
2/2     Running    0          31m
openshift-apiserver                        apiserver-b944c4645-x7rf2
2/2     Running    0          33m
 ...
```

- 노드가 READY 상태인지 확인합니다.

```
$ oc get nodes
```

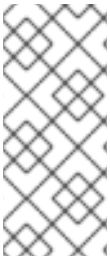
출력 예

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-47ltxtb-f76d1-mrffg-master-0      Ready    master   42m   v1.21.1+a620f50
ci-ln-47ltxtb-f76d1-mrffg-master-1      Ready    master   42m   v1.21.1+a620f50
ci-ln-47ltxtb-f76d1-mrffg-master-2      Ready    master   42m   v1.21.1+a620f50
```

```
ci-ln-47ltxtb-f76d1-mrffg-worker-a-gsxbz   Ready    worker   35m   v1.21.1+a620f50
ci-ln-47ltxtb-f76d1-mrffg-worker-b-5qqdx   Ready    worker   35m   v1.21.1+a620f50
ci-ln-47ltxtb-f76d1-mrffg-worker-c-rjkpq   Ready    worker   34m   v1.21.1+a620f50
```

10.7. 네트워크에서 클러스터 연결 끊기

필요한 모든 리포지토리를 미러링하고 연결이 끊긴 클러스터로 작동하도록 클러스터를 구성한 후 네트워크에서 클러스터의 연결을 끊을 수 있습니다.



참고

클러스터가 인터넷 연결이 끊어지면 Insights Operator의 성능이 저하됩니다. 이를 복원할 때까지 Insights Operator를 일시적으로 비활성화하여 이 문제를 방지할 수 있습니다.

10.8. 성능이 저하된 INSIGHTS OPERATOR 복원

네트워크에서 클러스터의 연결을 끊으면 클러스터가 인터넷 연결이 끊어야 합니다. Insights Operator는 Red Hat Insights 에 액세스할 수 있어야 하므로 성능이 저하됩니다.

이 주제에서는 성능이 저하된 Insights Operator에서 복구하는 방법에 대해 설명합니다.

절차

1. 다음과 같이 cloud. openshift.com 항목을 제거하려면.dockerconfigjson 파일을 편집합니다.

```
"cloud.openshift.com":{"auth":"<hash>","email":"user@example.com"}
```

2. 파일을 저장합니다.

3. edit .dockerconfigjson 파일로 클러스터 보안을 업데이트합니다.

```
$ oc set data secret/pull-secret -n openshift-config --from-
file=.dockerconfigjson=./.dockerconfigjson
```

4. Insights Operator가 더 이상 성능이 저하되지 않는지 확인합니다.

```
$ oc get co insights
```

출력 예

```
NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
insights  4.5.41    True        False         False      3d
```

10.9. 네트워크 복원

연결이 끊긴 클러스터를 다시 연결하고 온라인 레지스트리에서 이미지를 가져오려면 클러스터의 ICP(ImageContentSourcePolicy) 오브젝트를 삭제합니다. ICSP가 없으면 외부 레지스트리에 대한 가져오기 요청이 더 이상 미러 레지스트리로 리디렉션되지 않습니다.

절차

1. 클러스터에서 ICSP 오브젝트를 확인합니다.

```
$ oc get imagecontentsourcepolicy
```

출력 예

```
NAME              AGE
mirror-ocp        6d20h
ocp4-index-0      6d18h
qe45-index-0      6d15h
```

2. 클러스터 연결을 해제할 때 생성한 모든 ICSP 오브젝트를 삭제합니다.

```
$ oc delete imagecontentsourcepolicy <icsp_name> <icsp_name> <icsp_name>
```

예를 들면 다음과 같습니다.

```
$ oc delete imagecontentsourcepolicy mirror-ocp ocp4-index-0 qe45-index-0
```

출력 예

```
imagecontentsourcepolicy.operator.openshift.io "mirror-ocp" deleted
imagecontentsourcepolicy.operator.openshift.io "ocp4-index-0" deleted
imagecontentsourcepolicy.operator.openshift.io "qe45-index-0" deleted
```

3. 모든 노드가 재시작되고 **READY** 상태로 돌아갈 때까지 기다린 후 **registries.conf** 파일이 미러 레지스트리가 아닌 원래 레지스트리를 가리키는지 확인합니다.

a. 노드에 로그인합니다.

```
$ oc debug node/<node_name>
```

b. 디버그 쉘 내에서 **/host**를 **root** 디렉터리로 설정합니다.

```
sh-4.4# chroot /host
```

c. **registries.conf** 파일을 검사합니다.

```
sh-4.4# cat /etc/containers/registries.conf
```

출력 예

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"] 1
```

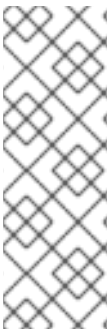
1

11장. IBM Z 또는 LINUXONE 환경에서 추가 장치 구성

OpenShift Container Platform을 설치한 후 z/VM과 함께 설치된 IBM Z 또는 LinuxONE 환경에서 클러스터에 대한 추가 장치를 구성할 수 있습니다. 다음 장치를 구성할 수 있습니다.

- 파이버 채널 프로토콜(FCP) 호스트
- FCP LUN
- DASD
- qeth

MCO(Machine Config Operator)를 사용하여 udev 규칙을 추가하거나 장치를 수동으로 구성할 수 있습니다.



참고

여기에 설명된 절차는 z/VM 설치에만 적용됩니다. IBM Z 또는 LinuxONE 인프라에 RHEL KVM으로 클러스터를 설치한 경우 KVM 게스트에 장치를 추가한 후 KVM 게스트 내에 추가 구성이 필요하지 않습니다. 그러나 z/VM 및 RHEL KVM 환경에서 모두 Local Storage Operator 및 Kubernetes NMState Operator를 구성하려면 다음 단계를 적용해야 합니다.

추가 리소스

- 설치 후 시스템 구성 작업

11.1. MCO(MACHINE CONFIG OPERATOR)를 사용하여 추가 장치 구성

이 섹션의 작업에서는 MCO(Machine Config Operator) 기능을 사용하여 IBM Z 또는 LinuxONE 환경에서 추가 장치를 구성하는 방법을 설명합니다. MCO를 사용하여 장치를 구성하는 것은 영구적이지만 컴퓨팅 노드에 대한 특정 구성만 허용합니다. MCO는 컨트롤 플레인 노드가 다른 구성을 갖는 것을 허용하지 않습니다.

사전 요구 사항

- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- z/VM 게스트에서 장치를 사용할 수 있어야 합니다.
- 장치가 이미 연결되어 있습니다.
- 장치는 커널 매개변수에서 설정할 수 있는 cio_ignore 목록에 포함되어 있지 않습니다.
- 다음 YAML을 사용하여 MachineConfig 오브젝트 파일을 생성했습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker0
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values:
[worker,worker0]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker0: ""
```

11.1.1. FCP(Fibre Channel Protocol) 호스트 구성

다음은 udev 규칙을 추가하여 NPIV(N_Port Identifier Virtualization)를 사용하여 FCP 호스트 어댑터를 구성하는 방법의 예입니다.

절차

1. 다음 샘플 udev 규칙 441-zfcp-host-0.0.8000.rules 를 사용합니다.

```
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.8000", DRIVER=="zfcp",
GOTO="cfg_zfcp_host_0.0.8000"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="zfcp", TEST=="[ccw/0.0.8000]",
GOTO="cfg_zfcp_host_0.0.8000"
GOTO="end_zfcp_host_0.0.8000"

LABEL="cfg_zfcp_host_0.0.8000"
```

```
ATTR{[ccw/0.0.8000]online}="1"

LABEL="end_zfcp_host_0.0.8000"
```

2. 다음 명령을 실행하여 규칙을 **Base64**로 인코딩합니다.

```
$ base64 /path/to/file/
```

3. 다음 **MCO** 샘플 프로필을 **YAML** 파일에 복사합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-zfcp-host-0.0.8000.rules (3)
```

(1) 머신 구성 파일에 정의된 역할입니다.

(2) 이전 단계에서 생성한 **Base64** 인코딩 문자열입니다.

(3) **udev** 규칙이 있는 경로입니다.

11.1.2. FCP LUN 구성

다음은 **udev** 규칙을 추가하여 **FCP LUN**을 구성하는 방법의 예입니다. 새 **FCP LUN**을 추가하거나 다중 경로로 이미 구성된 **LUN**에 경로를 추가할 수 있습니다.

절차

1. 다음 샘플 udev 규칙 41-zfcp-lun-0.0.8000:0x500507680d760026:0x00bc000000000000.rules:

```
ACTION=="add", SUBSYSTEMS=="ccw", KERNELS=="0.0.8000",
GOTO="start_zfcp_lun_0.0.8207"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="start_zfcp_lun_0.0.8000"
SUBSYSTEM=="fc_remote_ports", ATTR{port_name}=="0x500507680d760026",
GOTO="cfg_fc_0.0.8000_0x500507680d760026"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="cfg_fc_0.0.8000_0x500507680d760026"
ATTR{[ccw/0.0.8000]0x500507680d760026/unit_add}="0x00bc000000000000"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="end_zfcp_lun_0.0.8000"
```

2. 다음 명령을 실행하여 규칙을 Base64로 인코딩합니다.

```
$ base64 /path/to/file/
```

3. 다음 MCO 샘플 프로필을 YAML 파일에 복사합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-zfcp-lun-0.0.8000:0x500507680d760026:0x00bc000000000000.rules (3)
```

(1) 머신 구성 파일에 정의된 역할입니다.

2

이전 단계에서 생성한 **Base64** 인코딩 문자열입니다.

3

udev 규칙이 있는 경로입니다.

11.1.3. DASD 구성

다음은 **udev** 규칙을 추가하여 **DASD** 장치를 구성하는 방법의 예입니다.

절차

1. 다음 샘플 **udev** 규칙 **41-dasd-0.0.4444.rules**를 사용합니다.

```
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.4444", DRIVER=="dasd-eckd",
GOTO="cfg_dasd_eckd_0.0.4444"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="dasd-eckd", TEST=="
[ccw/0.0.4444]", GOTO="cfg_dasd_eckd_0.0.4444"
GOTO="end_dasd_eckd_0.0.4444"

LABEL="cfg_dasd_eckd_0.0.4444"
ATTR{[ccw/0.0.4444]online}="1"

LABEL="end_dasd_eckd_0.0.4444"
```

2. 다음 명령을 실행하여 규칙을 **Base64**로 인코딩합니다.

```
$ base64 /path/to/file/
```

3. 다음 **MCO** 샘플 프로필을 **YAML** 파일에 복사합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 1
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
```

```
storage:
  files:
  - contents:
      source: data:text/plain;base64,<encoded_base64_string> 2
    filesystem: root
    mode: 420
    path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules 3
```

1 머신 구성 파일에 정의된 역할입니다.

2 이전 단계에서 생성한 Base64 인코딩 문자열입니다.

3 udev 규칙이 있는 경로입니다.

11.1.4. qeth 구성

다음은 udev 규칙을 추가하여 qeth 장치를 구성하는 방법의 예입니다.

절차

1. 다음 샘플 udev 규칙 41-qeth-0.0.1000.rules 를 사용합니다.

```
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1001", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1002", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="cfg_qeth_0.0.1000"
GOTO="end_qeth_0.0.1000"

LABEL="group_qeth_0.0.1000"
TEST=="[ccwgroup/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1001]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1002]", GOTO="end_qeth_0.0.1000"
ATTR{[drivers/ccwgroup:qeth]group}="0.0.1000,0.0.1001,0.0.1002"
GOTO="end_qeth_0.0.1000"
```

```
LABEL="cfg_qeth_0.0.1000"
ATTR{[ccwgroup/0.0.1000]online}="1"

LABEL="end_qeth_0.0.1000"
```

2. 다음 명령을 실행하여 규칙을 **Base64**로 인코딩합니다.

```
$ base64 /path/to/file/
```

3. 다음 **MCO** 샘플 프로필을 **YAML** 파일에 복사합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 (1)
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> (2)
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules (3)
```

(1) 머신 구성 파일에 정의된 역할입니다.

(2) 이전 단계에서 생성한 **Base64** 인코딩 문자열입니다.

(3) **udev** 규칙이 있는 경로입니다.

다음 단계

- Local Storage Operator 설치 및 구성

- 노드 네트워크 구성 업데이트

11.2. 추가 장치 수동 구성

이 섹션의 작업은 IBM Z 또는 LinuxONE 환경에서 추가 장치를 수동으로 구성하는 방법을 설명합니다. 이 설정 방법은 노드를 재시작해도 유지되지만 OpenShift Container Platform 네이티브는 아니며 노드를 교체하는 경우 단계를 다시 수행해야 합니다.

사전 요구 사항

- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- 노드에서 장치를 사용할 수 있어야 합니다.
- z/VM 환경에서 장치를 z/VM 게스트에 연결해야 합니다.

절차

1. 다음 명령을 실행하여 SSH를 통해 노드에 연결합니다.

```
$ ssh <user>@<node_ip_address>
```

다음 명령을 실행하여 노드에 대한 디버그 세션을 시작할 수도 있습니다.

```
$ oc debug node/<node_name>
```

2. chzdev 명령으로 장치를 활성화하려면 다음 명령을 입력합니다.

```
$ sudo chzdev -e 0.0.8000
  sudo chzdev -e 1000-1002
  sude chzdev -e 4444
  sudo chzdev -e 0.0.8000:0x500507680d760026:0x00bc000000000000
```

추가 리소스

- IBM 문서의 영구 장치 구성 을 참조하십시오.

11.3. ROCE 네트워크 카드

RoCE(RDMA over Converged Ethernet) 네트워크 카드를 활성화할 필요가 없으며 노드에서 사용 가능할 때마다 Kubernetes NMState Operator로 해당 인터페이스를 구성할 수 있습니다. 예를 들어, RoCE 네트워크 카드는 z/VM 환경에 연결되거나 RHEL KVM 환경에서 전달되는 경우 사용할 수 있습니다.

11.4. FCP LUN 멀티패스 활성화

이 섹션의 작업은 IBM Z 또는 LinuxONE 환경에서 추가 장치를 수동으로 구성하는 방법을 설명합니다. 이 설정 방법은 노드를 재시작해도 유지되지만 OpenShift Container Platform 네이티브는 아니며 노드를 교체하는 경우 단계를 다시 수행해야 합니다.

중요

IBM Z 및 LinuxONE에서는 설치 중에 클러스터를 구성하는 경우에만 다중 경로를 활성화할 수 있습니다. 자세한 내용은 *IBM Z 및 LinuxONE에 z/VM으로 클러스터 설치*의 "RHCOS 설치 및 OpenShift Container Platform 부트스트랩 프로세스 시작"을 참조하십시오.

사전 요구 사항

- 관리 권한이 있는 사용자로 클러스터에 로그인했습니다.
- 위에서 설명한 방법 중 하나를 사용하여 LUN에 대한 여러 경로를 구성했습니다.

절차

1. 다음 명령을 실행하여 SSH를 통해 노드에 연결합니다.

```
$ ssh <user>@<node_ip_address>
```

다음 명령을 실행하여 노드에 대한 디버그 세션을 시작할 수도 있습니다.

```
$ oc debug node/<node_name>
```

2. 다중 경로를 활성화하려면 다음 명령을 실행합니다.

```
$ sudo /sbin/mpathconf --enable
```

3. multipathd 데몬을 시작하려면 다음 명령을 실행합니다.

```
$ sudo multipath
```

4. 선택 사항:ECDHE로 다중 경로 장치를 포맷하려면 다음 명령을 실행합니다.

```
$ sudo fdisk /dev/mapper/mpatha
```

검증

- 장치가 그룹화되었는지 확인하려면 다음 명령을 실행합니다.

```
$ sudo multipath -ll
```

출력 예

```
mpatha (20017380030290197) dm-1 IBM,2810XIV
  size=512G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
-+- policy='service-time 0' prio=50 status=enabled
 |- 1:0:0:6  sde 68:16  active ready running
 |- 1:0:1:6  sdf 69:24  active ready running
 |- 0:0:0:6  sdg  8:80  active ready running
 `- 0:0:1:6  sdh 66:48  active ready running
```

다음 단계

- Local Storage Operator 설치 및 구성
- 노드 네트워크 구성 업데이트