OpenShift sandboxed containers 1.6

사용자 가이드

OpenShift Container Platform에서 샌드박스 컨테이너 배포

Last Updated: 2024-10-05

OpenShift sandboxed containers 1.6 사용자 가이드

OpenShift Container Platform에서 샌드박스 컨테이너 배포

법적 공지

초록

베어 메탈, 퍼블릭 클라우드 및 IBM 플랫폼의 OpenShift Container Platform에 OpenShift 샌드박스 컨테이너 배포.

차례

머리말 3
RED HAT 문서에 관한 피드백 제공 3

1장. OPENSHIFT 샌드박스 컨테이너 정보 4
1.1. 기능 4
1.2. OPENSHIFT CONTAINER PLATFORM과의 호환성 5
1.3. 노드 자격 확인 6
1.4. 일반 용어 7
1.5. OPENSHIFT 샌드박스 컨테이너 OPERATOR 7
1.6. OPENSHIFT VIRTUALIZATION 8
1.7. 스토리지 고려 사항 8
1.8. FIPS 컴플라이언스 9

2장. 베어 메탈에 워크로드 배포 10
2.1. 환경 준비 10
2.2. 웹 콘솔을 사용하여 워크로드 배포 16
2.3. 명령줄을 사용하여 워크로드 배포 19

3장. 퍼블릭 클라우드에 워크로드 배포 26
3.1. AWS에 워크로드 배포 26
3.2. AZURE에 워크로드 배포 45

4장. IBM에 워크로드 배포 66
4.1. 환경 준비 66
4.2. 명령줄을 사용하여 워크로드 배포 70

5장. 모니터링 81
5.1. 메트릭 정보 81
5.2. 메트릭 보기 81

6장. 설치 제거 83
6.1. 웹 콘솔을 사용하여 설치 제거 83
6.2. CLI를 사용하여 설치 제거 85

7장. 업그레이드 89
7.1. 리소스 업그레이드 89
7.2. OPERATOR 업그레이드 89

8장. 문제 해결 90
8.1. RED HAT 지원을 위한 데이터 수집 90
8.2. 로그 데이터 수집 91

부록 A. KATACONFIG 상태 메시지 94

머리말

RED HAT 문서에 관한 피드백 제공

Jira에서 **문제 생성** 양식을 제출하여 피드백을 제공하거나 오류를 보고할 수 있습니다. Jira 문제는 Red Hat Hybrid Cloud Infrastructure Jira 프로젝트에서 생성되어 피드백의 진행 상황을 추적할 수 있습니다.

1. Jira에 로그인했는지 확인합니다. Jira 계정이 없는 경우 Red Hat Jira 계정을 생성해야 합니다.
2. **Create Issue** 양식을 시작합니다.
3. **요약**,**설명** 및 **보고자** 필드를 완료합니다.
   설명 필드에 문서 URL, 장 또는 섹션 번호, 문제에 대한 자세한 설명을 포함합니다.
4. **생성**을 클릭합니다.

1장. OPENSHIFT 샌드박스 컨테이너 정보

OpenShift Container Platform용 OpenShift 샌드박스 컨테이너는 Kata Containers를 선택적 런타임으로 통합하여 경량 가상 머신에서 컨테이너화된 애플리케이션을 실행하여 향상된 보안 및 격리 기능을 제공합니다. 이러한 통합은 기존 OpenShift 워크플로우를 크게 변경하지 않고 민감한 워크로드를 위한 보다 안전한 런타임 환경을 제공합니다. 이 런타임은 전용 VM(가상 머신)의 컨테이너를 지원하므로 워크로드 분리가 향상됩니다.

1.1. 기능

OpenShift 샌드박스 컨테이너는 다음과 같은 기능을 제공합니다.

권한이 있거나 신뢰할 수 없는 워크로드 실행

권한 있는 컨테이너를 실행하여 클러스터 노드를 손상시킬 위험이 없으면 특정 권한이 필요한 워크로드를 안전하게 실행할 수 있습니다. 특수 권한이 필요한 워크로드에는 다음이 포함됩니다.

- 커널의 특수 기능이 필요한 워크로드는 CRI-O와 같은 표준 컨테이너 런타임에서 부여하는 기본 기능 이상으로, 예를 들어 낮은 수준의 네트워킹 기능에 액세스합니다.
- 예를 들어 특정 물리적 장치에 액세스하기 위해 높은 루트 권한이 필요한 워크로드입니다. OpenShift 샌드박스 컨테이너를 사용하면 특정 장치만 VM(가상 머신)에 전달하여 워크로드가 나머지 시스템에 액세스하거나 잘못 구성할 수 없도록 할 수 있습니다.
- **set-uid** 루트 바이너리를 설치하거나 사용하는 워크로드입니다. 이러한 바이너리는 특수 권한을 부여하므로 보안 위험이 발생할 수 있습니다. OpenShift 샌드박스 컨테이너를 사용하면 추가 권한이 가상 머신으로 제한되며 클러스터 노드에 대한 특별한 액세스 권한이 부여되지 않습니다.
  일부 워크로드에서는 특히 클러스터 노드를 구성하려면 권한이 필요합니다. 가상 머신에서 실행하면 이러한 워크로드가 작동하지 않기 때문에 권한 있는 컨테이너를 계속 사용해야 합니다.

민감한 워크로드에 대한 격리 확인

Red Hat OpenShift Container Platform의 OpenShift 샌드박스 컨테이너는 Kata Containers를 선택적 런타임으로 통합하여 경량 가상 머신에서 컨테이너화된 애플리케이션을 실행하여 향상된 보안 및 격리 기능을 제공합니다. 이러한 통합은 기존 OpenShift 워크플로우를 크게 변경하지 않고 민감한 워크로드를 위한 보다 안전한 런타임 환경을 제공합니다. 이 런타임은 전용 VM(가상 머신)의 컨테이너를 지원하므로 워크로드 분리가 향상됩니다.

각 워크로드에 대한 커널 격리 확인

사용자 지정 커널 튜닝(예: **sysctl**, 스케줄러 변경 또는 캐시 튜닝)이 필요한 워크로드와 사용자 지정 커널 모듈 생성(예: **트리 부족** 또는 특수 인수)을 실행할 수 있습니다.

테넌트 간에 동일한 워크로드를 공유

동일한 OpenShift Container Platform 클러스터를 공유하는 다양한 조직의 여러 사용자(테넌트)를 지원하는 워크로드를 실행할 수 있습니다. 또한 이 시스템은 컨테이너 네트워크 기능(CNF) 및 엔터프라이즈 애플리케이션과 같은 여러 벤더의 타사 워크로드 실행을 지원합니다. 예를 들어 타사 CNF는 사용자 지정 설정이 패킷 튜닝 또는 다른 애플리케이션에서 설정한 **sysctl** 변수를 방해하는 것을 원하지 않을 수 있습니다. 완전히 격리된 커널 내에서 실행하면 "noisy neighbor" 구성 문제를 방지하는 데 도움이 됩니다.

소프트웨어 테스트를 위한 적절한 격리 및 샌드박스 확인

알려진 취약점으로 컨테이너화된 워크로드를 실행하거나 기존 애플리케이션의 문제를 처리할 수 있습니다. 관리자는 이러한 격리를 통해 개발자에게 pod에 대한 관리 권한을 부여할 수 있습니다. 이 제어는 개발자가 일반적으로 부여한 것 이외의 구성을 테스트하거나 검증하려고 할 때 유용합니다. 예를 들어 관리자는 커널 패킷 필터링(eBPF)을 개발자에게 안전하게 위임할 수 있습니다. eBPF에는 **CAP_ADMIN** 또는 **CAP_BPF** 권한이 필요하므로 컨테이너 호스트 작업자 노드의 모든 프로세스에 대

한 액세스 권한이 부여되므로 표준 CRI-O 설정에서 허용되지 않습니다. 마찬가지로 관리자는 **SystemTap** 과 같은 침입 툴에 대한 액세스 권한을 부여하거나 개발 중에 사용자 지정 커널 모듈 로드를 지원할 수 있습니다.

VM 경계를 통한 기본 리소스 포함 확인

기본적으로 OpenShift 샌드박스 컨테이너는 CPU, 메모리, 스토리지 및 네트워킹과 같은 리소스를 강력하고 안전한 방식으로 관리합니다. OpenShift 샌드박스 컨테이너는 VM에 배포되므로 추가 격리 및 보안 계층을 통해 리소스에 대한 보다 세분화된 액세스 제어가 가능합니다. 예를 들어, 잘못된 컨테이너는 VM에서 사용할 수 있는 것보다 더 많은 메모리를 할당할 수 없습니다. 반대로 네트워크 카드 또는 디스크에 대한 전용 액세스 권한이 필요한 컨테이너는 다른 장치에 액세스하지 않고도 해당 장치를 완전히 제어할 수 있습니다.

1.2. OPENSHIFT CONTAINER PLATFORM과의 호환성

OpenShift Container Platform 플랫폼에 필요한 기능은 다음 두 가지 주요 구성 요소에서 지원됩니다.

- Kata Runtime: 모든 OpenShift Container Platform 릴리스와 함께 RHCOS(Red Hat Enterprise Linux CoreOS) 및 업데이트가 포함됩니다.
- OpenShift 샌드박스 컨테이너 Operator: 웹 콘솔 또는 OpenShift CLI(**oc**)를 사용하여 Operator를 설치합니다.

OpenShift 샌드박스 컨테이너 Operator는 Rolling Stream Operator 로, 최신 버전이 지원되는 유일한 버전입니다. 현재 지원되는 모든 OpenShift Container Platform 버전에서 작동합니다. 자세한 내용은 OpenShift Container Platform 라이프 사이클 정책을 참조하십시오.

Operator는 RHCOS 호스트와 함께 제공되는 기능과 실행되는 환경에 따라 다릅니다.

참고

작업자 노드에 RHCOS(Red Hat Enterprise Linux CoreOS)를 설치해야 합니다. RHEL 노드는 지원되지 않습니다.

OpenShift 샌드박스 컨테이너와 OpenShift Container Platform 릴리스 간의 호환성 매트릭스는 호환되는 기능과 환경을 식별합니다.

표 1.1. 지원되는 아키텍처

아키텍처	OpenShift Container Platform 버전
x86_64	4.8 이상
s390x	4.14 이상

Kata 컨테이너 런타임을 배포하는 방법은 다음 두 가지가 있습니다.

- 베어 메탈
- 피어 Pod

피어 Pod 기술은 OpenShift 샌드박스 컨테이너 1.5/OpenShift Container Platform 4.14에서 시작되었습니다. 공용 클라우드에서 OpenShift 샌드박스 컨테이너를 배포할 수 있습니다.

표 1.2. OpenShift 버전의 기능 가용성

기능	배포 방법	OpenShift Container Platform 4.15	OpenShift Container Platform 4.16
기밀 컨테이너[1]	베어 메탈	없음	없음
	피어 Pod	개발자 프리뷰	개발자 프리뷰
GPU 지원[2]	베어 메탈	없음	없음
	피어 Pod	개발자 프리뷰	개발자 프리뷰

1. 기밀 컨테이너는 AMD SEV-SNP에서만 지원됩니다.
2. GPU 기능은 s390x에서 사용할 수 없습니다.

표 1.3. OpenShift 샌드박스 컨테이너에서 지원되는 플랫폼

플랫폼	피어 Pod	GPU	기밀 컨테이너
AWS 클라우드 컴퓨팅 서비스	제공됨	개발자 프리뷰	없음
Microsoft Azure 클라우드 컴퓨팅 서비스	제공됨	개발자 프리뷰	개발자 프리뷰

추가 리소스

- 개발자 프리뷰 지원 범위
- AWS에 워크로드 배포
- Azure에 워크로드 배포
- 베어 메탈에 워크로드 배포

1.3. 노드 자격 확인

OpenShift 샌드박스 컨테이너를 배포하기 전에 베어 메탈 클러스터의 노드가 OpenShift 샌드박스 컨테이너를 실행할 수 있는지 확인할 수 있습니다. 노드 자격에 대한 가장 일반적인 이유는 가상화 지원 부족입니다. 자격 없는 노드에서 샌드박스 워크로드를 실행하는 경우 오류가 발생합니다.

고급 워크플로

1. Node Feature Discovery Operator를 설치합니다.
2. **NodeFeatureDiscovery** CR(사용자 정의 리소스)을 생성합니다.
3. **Kataconfig** CR을 생성할 때 노드 자격 검사를 활성화합니다. 모든 작업자 노드 또는 선택한 노드에서 노드 자격 검사를 실행할 수 있습니다.

추가 리소스

- Node Feature Discovery Operator 설치

1.4. 일반 용어

설명서 전반에 걸쳐 다음 용어가 사용됩니다.

샌드 박스

샌드박스는 프로그램을 실행할 수 있는 격리된 환경입니다. 샌드박스에서는 호스트 시스템이나 운영 체제에 손상을 주지 않고 테스트되지 않았거나 신뢰할 수 없는 프로그램을 실행할 수 있습니다. OpenShift 샌드박스 컨테이너의 경우 가상화를 사용하여 다른 커널에서 워크로드를 실행하여 동일한 호스트에서 실행되는 여러 워크로드 간의 상호 작용을 보다 효과적으로 제어할 수 있습니다.

Pod

Pod는 Kubernetes 및 OpenShift Container Platform에서 상속된 구성 요소입니다. 컨테이너를 배포할 수 있는 리소스를 나타냅니다. 컨테이너는 Pod 내부에서 실행되며 Pod는 여러 컨테이너 간에 공유할 수 있는 리소스를 지정하는 데 사용됩니다.
OpenShift 샌드박스 컨테이너의 컨텍스트에서 Pod가 가상 시스템으로 구현됩니다. 동일한 가상 시스템의 동일한 Pod에서 여러 컨테이너를 실행할 수 있습니다.

OpenShift 샌드박스 컨테이너 Operator

Operator는 시스템에서 수동으로 수행할 수 있는 작업을 자동화하는 소프트웨어 구성 요소입니다. OpenShift 샌드박스 컨테이너 Operator는 클러스터에서 샌드박스 컨테이너의 라이프사이클을 관리하는 작업을 수행합니다. OpenShift 샌드박스 컨테이너 Operator를 사용하여 샌드박스 컨테이너 설치 및 제거, 소프트웨어 업데이트 및 상태 모니터링과 같은 작업을 수행할 수 있습니다.

Kata 컨테이너

Kata 컨테이너는 OpenShift 샌드박스 컨테이너를 빌드하는 데 사용되는 핵심 업스트림 프로젝트입니다. OpenShift 샌드 박스 컨테이너는 Kata Container와 OpenShift Container Platform을 통합합니다.

KataConfig

KataConfig 오브젝트는 샌드박스 컨테이너의 구성을 나타냅니다. 소프트웨어가 배포된 노드와 같이 클러스터 상태에 대한 정보를 저장합니다.

런타임 클래스

RuntimeClass 오브젝트는 지정된 워크로드를 실행하는 데 사용할 수 있는 런타임에 대해 설명합니다. **kata**라는 런타임 클래스가 OpenShift 샌드박스 컨테이너 Operator에 의해 설치 및 배포됩니다. 런타임 클래스에는 Pod 오버헤드와 같이 런타임에서 작동해야 하는 리소스를 설명하는 런타임에 대한 정보가 포함되어 있습니다.

피어 Pod

OpenShift 샌드박스 컨테이너의 피어 Pod는 표준 Pod의 개념을 확장합니다. 가상 머신이 작업자 노드 자체에서 생성되는 표준 샌드박스 컨테이너와 달리 피어 Pod에서 지원되는 하이퍼바이저 또는 클라우드 공급자 API를 사용하여 원격 하이퍼바이저를 통해 가상 머신이 생성됩니다. 피어 포드는 작업자 노드에서 일반 pod 역할을 하며 해당 VM이 다른 위치에서 실행됩니다. VM의 원격 위치는 사용자에게 투명하며 Pod 사양의 런타임 클래스에 의해 지정됩니다. 피어 Pod 설계는 중첩된 가상화의 필요성을 우회합니다.

1.5. OPENSHIFT 샌드박스 컨테이너 OPERATOR

OpenShift 샌드박스 컨테이너 Operator는 Kata 컨테이너의 모든 구성 요소를 캡슐화합니다. 설치, 라이프사이클 및 구성 작업을 관리합니다.

OpenShift 샌드박스 컨테이너 Operator는 Operator 번들 형식으로 두 개의 컨테이너 이미지로 패키지됩니다.

- 번들 이미지에는 Operator OLM을 준비하는데 필요한 메타데이터가 포함되어 있습니다.
- 두 번째 컨테이너 이미지에는 **KataConfig** 리소스를 모니터링하고 관리하는 실제 컨트롤러가 포함되어 있습니다.

OpenShift 샌드박스된 컨테이너 Operator는 RHCOS(Red Hat Enterprise Linux CoreOS) 확장 개념을 기반으로 합니다. RHCOS 확장은 선택적 OpenShift Container Platform 소프트웨어를 설치하는 메커니즘입니다. OpenShift 샌드박스된 컨테이너 Operator는 이 메커니즘을 사용하여 클러스터에 샌드박스 컨테이너를 배포합니다.

샌드박스 컨테이너 RHCOS 확장에는 Kata, QEMU 및 종속 항목에 대한 RPM이 포함되어 있습니다. Machine Config Operator에서 제공하는 **MachineConfig** 리소스를 사용하여 활성화할 수 있습니다.

추가 리소스

- RHCOS에 확장 기능 추가

1.6. OPENSHIFT VIRTUALIZATION

OpenShift Virtualization을 사용하여 클러스터에 OpenShift 샌드박스 컨테이너를 배포할 수 있습니다.

OpenShift Virtualization 및 OpenShift 샌드박스 컨테이너를 동시에 실행하려면 가상 머신이 실시간 업그레이드 가능해야 노드 재부팅을 차단하지 않도록 합니다. 자세한 내용은 OpenShift Virtualization 설명서의 실시간 마이그레이션 정보를 참조하십시오.

1.7. 스토리지 고려 사항

1.7.1. 블록 볼륨 지원

OpenShift Container Platform은 원시 블록 볼륨을 정적으로 프로비저닝할 수 있습니다. 이러한 볼륨에는 파일 시스템이 없으며 디스크에 직접 쓰거나 자체 스토리지 서비스를 구현하는 애플리케이션에 성능 이점을 제공할 수 있습니다.

로컬 블록 장치를 OpenShift 샌드박스 컨테이너의 PV(영구 볼륨) 스토리지로 사용할 수 있습니다. 이 블록 장치는 LSO(Local Storage Operator)를 사용하여 프로비저닝할 수 있습니다.

Local Storage Operator는 기본적으로 OpenShift Container Platform에 설치되지 않습니다. 설치 지침은 Local Storage Operator 설치를 참조하십시오.

OpenShift 샌드박스 컨테이너의 원시 블록 볼륨은 PV 사양에 **volumeMode:Block** 을 지정하여 프로비저닝됩니다.

블록 볼륨 예

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage"
spec:
  nodeSelector:
    nodeSelectorTerms:
```

```
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - worker-0
  storageClassDevices:
    - storageClassName: "local-sc"
      forceWipeDevicesAndDestroyAllData: false
      volumeMode: Block (1)
      devicePaths:
        - /path/to/device (2)
```

(1) 이 PV가 원시 블록 볼륨임을 나타내려면 **volumeMode**를 **Block**으로 설정해야 합니다.

(2) 이 값을 **LocalVolume** 리소스의 실제 로컬 디스크 파일 경로로 **바꿉니다**. 프로비저너가 배포되면 이러한 로컬 디스크에 PV가 생성됩니다. 또한 이 경로를 사용하여 OpenShift 샌드박스 컨테이너를 배포할 때 블록 장치를 사용하는 노드에 레이블을 지정해야 합니다.

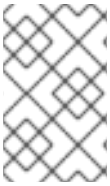
1.8. FIPS 컴플라이언스

OpenShift Container Platform은 FIPS(Federal Information Processing Standards) 140-2 및 140-3를 위해 설계되었습니다. FIPS 모드에서 부팅된 RHEL(Red Hat Enterprise Linux CoreOS) 또는 RHCOS(Red Hat Enterprise Linux CoreOS)를 실행하는 경우 OpenShift Container Platform 코어 구성 요소는 **x86_64**,**ppc64le**, **s390x** 아키텍처에서만 FIPS 140-2/140-3 Validation에 대해 NIST에 제출된 RHEL 암호화 라이브러리를 사용합니다.

NIST 검증 프로그램에 대한 자세한 내용은 암호화 모듈 유효성 검사 프로그램을 참조하십시오. 검증을 위해 제출된 개별 RHEL 암호화 라이브러리의 최신 NIST 상태는 규정 준수 활동 및 정부 표준을 참조하십시오.

OpenShift 샌드박스 컨테이너는 FIPS가 활성화된 클러스터에서 사용할 수 있습니다.

FIPS 모드에서 실행하면 OpenShift 샌드박스 컨테이너 구성 요소, VM 및 VM 이미지가 FIPS를 준수하도록 조정됩니다.



참고

OpenShift 샌드박스 컨테이너에 대한 FIPS 컴플라이언스는 **kata** 런타임 클래스에만 적용됩니다. 피어 Pod 런타임 클래스 **kata-remote** 는 아직 완전히 지원되지 않으며 FIPS 규정 준수를 위해 테스트되지 않았습니다.

FIPS 컴플라이언스는 보안 수준이 높은 환경에서 요구되는 가장 중요한 구성요소 중 하나로, 지원되는 암호화 기술만 노드에서 허용합니다.

중요

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

OpenShift Container Platform 컴플라이언스 프레임워크에 대한 Red Hat의 관점을 이해하려면 OpenShift 보안 가이드의 위험 관리 및 규제 준비 장을 참조하십시오.

2장. 베어 메탈에 워크로드 배포

작업자 노드에 RHCOS(Red Hat Enterprise Linux CoreOS)가 설치된 온프레미스 베어 메탈 서버에 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

참고

- RHEL 노드는 지원되지 않습니다.
- 중첩된 가상화는 지원되지 않습니다.

사용자 프로비저닝 , 설치 관리자 프로비저닝 또는 지원설치 프로그램을 포함하여 설치 방법을 사용하여 클러스터를 배포할 수 있습니다.

AWS(Amazon Web Services) 베어 메탈 인스턴스에 OpenShift 샌드박스 컨테이너를 설치할 수도 있습니다. 다른 클라우드 공급자가 제공하는 베어 메탈 인스턴스는 지원되지 않습니다.

배포 워크플로

다음 단계를 수행하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

1. 환경을 준비합니다.
2. **KataConfig** 사용자 지정 리소스를 생성합니다.
3. **kata** 런타임 클래스를 사용하도록 워크로드 오브젝트를 구성합니다.

2.1. 환경 준비

환경을 준비하려면 다음 단계를 수행합니다.

1. 클러스터에 충분한 리소스가 있는지 확인합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 설치합니다.
3. 선택 사항: 작업자 노드가 OpenShift 샌드박스 컨테이너를 지원하는지 확인하도록 노드 자격 검사를 구성합니다.
    a. NFD(Node Feature Discovery) Operator를 설치합니다. 자세한 내용은 NFD Operator 설명서 를 참조하십시오.
    b. **NodeFeatureDiscovery** CR(사용자 정의 리소스)을 생성하여 NFD Operator에서 확인하는 노드 구성 매개변수를 정의합니다.

2.1.1. 리소스 요구 사항

OpenShift 샌드박스 컨테이너를 사용하면 사용자가 샌드박스 런타임(Kata) 내의 OpenShift Container Platform 클러스터에서 워크로드를 실행할 수 있습니다. 각 Pod는 VM(가상 머신)으로 표시됩니다. 각 VM은 QEMU 프로세스에서 실행되며 컨테이너 워크로드를 관리하기 위한 감독자 역할을 하는 **kata-agent** 프로세스 및 해당 컨테이너에서 실행되는 프로세스를 호스팅합니다. 두 개의 추가 프로세스는 오버헤드를 더 추가합니다.

- **containerd-shim-kata-v2**는 pod와 통신하는 데 사용됩니다.
- **virtiofsd**는 게스트 대신 호스트 파일 시스템 액세스를 처리합니다.

각 VM은 기본 메모리 양으로 구성됩니다. 메모리를 명시적으로 요청하는 컨테이너의 경우 VM에 추가 메모리가 핫플러그됩니다.

메모리 리소스 없이 실행되는 컨테이너는 VM에서 사용하는 총 메모리가 기본 할당에 도달할 때까지 사용 가능한 메모리를 사용합니다. 게스트와 I/O 버퍼도 메모리를 소비합니다.

컨테이너에 특정 양의 메모리가 제공되면 컨테이너를 시작하기 전에 해당 메모리는 VM에 핫플러그됩니다.

메모리 제한을 지정하면 제한보다 많은 메모리를 사용하는 경우 워크로드가 종료됩니다. 메모리 제한을 지정하지 않으면 VM에서 실행 중인 커널이 메모리 부족을 실행할 수 있습니다. 커널이 메모리 부족하면 VM의 다른 프로세스를 종료할 수 있습니다.

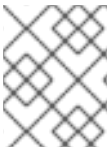
기본 메모리 크기

다음 표에는 리소스 할당에 대한 기본값이 나열되어 있습니다.

리소스	현재의
기본적으로 가상 머신에 할당된 메모리	2Gi
부팅 시 게스트 Linux 커널 메모리 사용량	~110Mi
QEMU 프로세스에서 사용하는 메모리 (VM 메모리 제외)	~30Mi
virtiofsd 프로세스에서 사용하는 메모리(VM I/O 버퍼 제외)	~10Mi
containerd-shim-kata-v2 프로세스에서 사용하는 메모리	~20Mi
Fedora에서 **dnf install**을 실행한 후 파일 버퍼 캐시 데이터	~300Mi* [1]

파일 버퍼가 나타나고 다음과 같은 여러 위치에서 고려됩니다.

- 게스트에서 파일 버퍼 캐시로 표시
- 허용된 사용자 공간 파일 I/O 작업을 매핑된 **virtiofsd** 데몬
- 게스트 메모리로 사용되는 QEMU 프로세스



참고

총 메모리 사용량은 메모리 사용량 통계에 따라 적절히 계산되며, 이 메트릭은 해당 메모리를 한 번만 계산합니다.

Pod 오버헤드는 노드의 Pod에서 사용하는 시스템 리소스의 양을 설명합니다. 다음과 같이 **oc describe runtimeclass kata** 를 사용하여 Kata 런타임에 대한 현재 Pod 오버헤드를 가져올 수 있습니다.

예

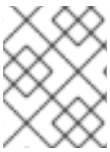
```
-
```

```
$ oc describe runtimeclass kata
```

출력 예

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata
overhead:
  podFixed:
    memory: "500Mi"
    cpu: "500m"
```

RuntimeClass의 **spec.overhead** 필드를 변경하여 Pod 오버헤드를 변경할 수 있습니다. 예를 들어 컨테이너에 대해 실행하는 구성이 QEMU 프로세스 및 게스트 커널 데이터에 350Mi 이상의 메모리를 사용하는 경우 필요에 맞게 **RuntimeClass** 오버헤드를 변경할 수 있습니다.



참고

Red Hat은 지정된 기본 오버헤드 값을 지원합니다. 기본 오버헤드 값 변경은 지원되지 않으며 값을 변경하면 기술적인 문제가 발생할 수 있습니다.

게스트에서 모든 유형의 파일 시스템 I/O를 수행할 때 게스트 커널에 파일 버퍼가 할당됩니다. 파일 버퍼는 **virtiofsd** 프로세스뿐만 아니라 호스트의 QEMU 프로세스에도 매핑됩니다.

예를 들어 게스트에서 300Mi 파일 버퍼 캐시를 사용하는 경우 QEMU와 **virtiofsd**는 모두 300Mi 추가 메모리를 사용하는 것처럼 나타납니다. 그러나 세 가지 경우 모두 동일한 메모리가 사용됩니다. 따라서 총 메모리 사용량은 300Mi에 불과하며 3개의 다른 위치에 매핑됩니다. 이는 메모리 사용량 메트릭을 보고할 때 올바르게 계산됩니다.

2.1.2. OpenShift 샌드박스 컨테이너 Operator 설치

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

2.1.2.1. 웹 콘솔을 사용하여 Operator 설치

Red Hat OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

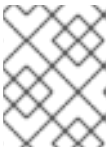
사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**로 이동합니다.
2. **키워드로 필터링** 필드에 **OpenShift sandboxed containers**를 입력합니다.
3. **OpenShift 샌드박스 컨테이너 Operator** 타일을 선택하고 **설치**를 클릭합니다.
4. **Operator 설치** 페이지의 사용 가능한 **업데이트 채널** 옵션 목록에서 **stable** 을 선택합니다.

5. 설치된 네임스페이스 용으로 **Operator 권장 네임스페이스** 가 선택되어 **있는지 확인합니다**. 이렇게 하면 필수 **openshift-sandboxed-containers-operator** 네임스페이스에 Operator가 설치됩니다. 이 네임스페이스가 아직 존재하지 않으면 자동으로 생성됩니다.



참고

openshift-sandboxed-containers-operator 이외의 네임스페이스에 OpenShift 샌드박스 컨테이너 Operator를 설치하려고 하면 설치가 실패합니다.

6. **승인 전략에 대해 자동 이 선택되어 있는지 확인합니다. Automatic** 은 기본값이며 새 z-stream 릴리스를 사용할 수 있을 때 OpenShift 샌드박스 컨테이너에 대한 자동 업데이트를 활성화합니다.

7. **설치**를 클릭합니다.

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

1. **Operators → 설치된 Operator**로 이동합니다.

2. OpenShift 샌드박스 컨테이너 Operator가 표시되는지 확인합니다.

추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경에 대한 Operator Lifecycle Manager에서 프록시 지원 구성 .

2.1.2.2. CLI를 사용하여 Operator 설치

CLI를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. **Namespace.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. 다음 명령을 실행하여 네임스페이스를 생성합니다.

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

3. **OperatorGroup.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

4. 다음 명령을 실행하여 operator 그룹을 생성합니다.

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

5. **Subscription.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0
```

6. 다음 명령을 실행하여 서브스크립션을 생성합니다.

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

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

- 다음 명령을 실행하여 Operator가 올바르게 설치되었는지 확인합니다.

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

출력 예

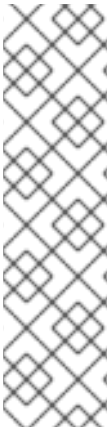
```
NAME                                DISPLAY                                VERSION   REPLACES   PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.6.0    1.5.3      Succeeded
```

2.1.2.3. 추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경을 위한 Operator Lifecycle Manager에서 프록시 지원 구성

2.1.3. NodeFeatureDiscovery CR 생성

NodeFeatureDiscovery CR(사용자 정의 리소스)을 생성하여 NFD(Node Feature Discovery) Operator에서 작업자 노드에서 OpenShift 샌드박스 컨테이너를 지원할 수 있는지 확인하는 구성 매개변수를 정의합니다.



참고

사용자가 알고 있는 선택된 작업자 노드에만 **kata** 런타임을 설치하려면 **feature.node.kubernetes.io/runtime.kata=true** 레이블을 선택한 노드에 적용하고 **KataConfig** CR에서 **checkNodeEligibility: true** 를 설정합니다.

모든 작업자 노드에 **kata** 런타임을 설치하려면 **KataConfig** CR에 **checkNodeEligibility: false** 를 설정합니다.

이러한 두 시나리오에서는 **NodeFeatureDiscovery** CR을 생성할 필요가 없습니다. 노드가 OpenShift 샌드박스 컨테이너를 실행할 수 있는지 확인하는 경우 **feature.node.kubernetes.io/runtime.kata=true** 레이블만 수동으로 적용해야 합니다.

다음 절차에서는 **feature.node.kubernetes.io/runtime.kata=true** 레이블을 모든 적격 노드에 적용하고 노드 자격을 확인하도록 **KataConfig** 리소스를 구성합니다.

사전 요구 사항

- NFD Operator가 설치되어 있습니다.

프로세스

1. 다음 예에 따라 **nfd.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: nfd-kata
  namespace: openshift-nfd
spec:
  workerConfig:
    configData: |
      sources:
        custom:
          - name: "feature.node.kubernetes.io/runtime.kata"
            matchOn:
              - cpuId: ["SSE4", "VMX"]
                loadedKMod: ["kvm", "kvm_intel"]
              - cpuId: ["SSE4", "SVM"]
                loadedKMod: ["kvm", "kvm_amd"]
# ...
```

2. **NodeFeatureDiscovery** CR을 생성합니다.

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

NodeFeatureDiscovery CR은 **feature.node.kubernetes.io/runtime.kata=true** 레이블을 모든 적격 작업자 노드에 적용합니다.

1. 다음 예에 따라 **kata-config.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: example-kataconfig
spec:
  checkNodeEligibility: true
```

2. **KataConfig** CR을 생성합니다.

```
$ oc create -f kata-config.yaml
```

검증

- 클러스터의 적격한 노드에 올바른 레이블이 적용되었는지 확인합니다.

```
$ oc get nodes --selector='feature.node.kubernetes.io/runtime.kata=true'
```

출력 예

```
NAME                        STATUS                     ROLES    AGE     VERSION
compute-3.example.com       Ready                      worker   4h38m   v1.25.0
compute-2.example.com       Ready                      worker   4h35m   v1.25.0
```

2.2. 웹 콘솔을 사용하여 워크로드 배포

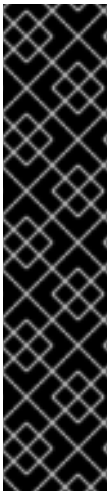
웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

2.2.1. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata** 를 **RuntimeClass** 로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

kata 런타임 클래스는 기본적으로 모든 작업자 노드에 설치됩니다. 특정 노드에만 **kata** 를 설치하려면 해당 노드에 레이블을 추가한 다음 **KataConfig** CR에 라벨을 정의할 수 있습니다.

OpenShift 샌드박스 컨테이너는 **kata** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.



중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 다음 요인은 재부팅 시간을 늘릴 수 있습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 선택 사항: 노드 자격 검사를 활성화하려면 Node Feature Discovery Operator를 설치했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 선택합니다.
3. KataConfig 탭에서 **KataConfig 만들기** 를 클릭합니다.
4. 다음 세부 정보를 입력합니다.
   - **이름**: 선택 사항: 기본 이름은 **example-kataconfig** 입니다.
   - labels: 선택 사항: **KataConfig** 리소스에 대한 특성을 식별하는 모든 관련 정보를 입력합니다. 각 레이블은 키-값 쌍을 나타냅니다.
   - checkNodeEligibility: 선택 사항: NFD(Node Feature Discovery Operator)를 사용하여 노드 자격을 감지하도록 선택합니다.
   - kataConfigPoolSelector. 선택 사항: 선택한 노드에 **kata** 를 설치하려면 선택한 노드의 라벨에 일치하는 표현식을 추가합니다.
     a. kataConfigPoolSelector 영역을 확장합니다.
     b. kataConfigPoolSelector 영역에서 matchExpressions 를 확장합니다. 이는 라벨 선택기 요구 사항 목록입니다.
     c. matchExpressions **추가를** 클릭합니다.
     d. **키** 필드에 선택기가 적용되는 라벨 키를 입력합니다.
     e. Operator 필드에 레이블 값과의 키 관계를 입력합니다. 유효한 연산자는 **In**,**NotIn**,**Exists** 및 **DoesNotExist** 입니다.
     f. **값** 영역을 확장한 다음 **값 추가** 를 클릭합니다.
     g. **값** 필드에 **키** 레이블 값에 **true** 또는 **false** 를 입력합니다.
   - loglevel: **kata** 런타임 클래스가 있는 노드에 대해 검색된 로그 데이터의 수준을 정의합니다.
5. **생성**을 클릭합니다. **KataConfig** CR이 생성되고 작업자 노드에 **kata** 런타임 클래스를 설치합니다.
   kata 설치가 완료되고 설치를 확인하기 전에 작업자 노드가 재부팅될 때까지 기다립니다.

검증

1. KataConfig 탭에서 **KataConfig** CR을 클릭하여 세부 정보를 확인합니다.
2. YAML 탭을 클릭하여 **상태 스탠자를 확인합니다**.
   상태 스탠자에는 **조건** 및 **kataNodes** 키가 포함되어 있습니다. **status.kataNodes** 의 값은 노드 배열이며 각각 특정 상태의 **kata** 설치 노드를 나열합니다. 업데이트가 있을 때마다 메시지가 표시됩니다.

3. **Reload** (다시 로드)를 클릭하여 YAML을 새로 고칩니다.
   status.kataNodes 어레이의 모든 작업자가 **설치** 및 **조건.InProgress: False** 를 지정하는 이유 없이 False를 표시하면 클러스터에 **kata** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

2.2.2. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트

중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **워크로드 → 워크로드** 유형(예: **Pod**)으로 이동합니다.
2. 워크로드 유형 페이지에서 오브젝트를 클릭하여 세부 정보를 확인합니다.
3. **YAML** 탭을 클릭합니다.
4. 다음 예와 같이 **spec.runtimeClassName: kata** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata
# ...
```

OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

2.3. 명령줄을 사용하여 워크로드 배포

명령줄을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

2.3.1. 선택 사항: Local Storage Operator를 사용하여 로컬 블록 볼륨 프로비저닝

OpenShift 샌드박스 컨테이너의 로컬 블록 볼륨은 LSO(Local Storage Operator)를 사용하여 프로비저닝할 수 있습니다. 로컬 볼륨 프로비저너는 정의된 리소스에 지정된 경로에서 블록 볼륨 장치를 찾습니다.

사전 요구 사항

- Local Storage Operator가 설치되어 있습니다.
- 다음 조건을 충족하는 로컬 디스크가 있습니다.
  - 노드에 연결되어 있습니다.
  - 마운트되지 않았습니다.
  - 파티션이 포함되어 있지 않습니다.

프로세스

1. 로컬 볼륨 리소스를 생성합니다. 이 리소스는 로컬 볼륨에 대한 노드 및 경로를 정의해야 합니다.

참고

동일한 장치에 다른 스토리지 클래스 이름을 사용하지 마십시오. 이렇게 하면 여러 영구 볼륨(PV)이 생성됩니다.

예: 블록

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" (1)
spec:
  nodeSelector: (2)
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-136-143
          - ip-10-0-140-255
          - ip-10-0-144-180
```

```
storageClassDevices:
  - storageClassName: "local-sc" 3
    forceWipeDevicesAndDestroyAllData: false 4
    volumeMode: Block
    devicePaths: 5
      - /path/to/device 6
```

1. Local Storage Operator가 설치된 네임스페이스입니다.
2. 선택 사항: 로컬 스토리지 볼륨이 연결된 노드 목록이 포함된 노드 선택기입니다. 이 예에서는 **oc get node**에서 가져온 노드 호스트 이름을 사용합니다. 값을 정의하지 않으면 Local Storage Operator에서 사용 가능한 모든 노드에서 일치하는 디스크를 찾습니다.
3. 영구 볼륨 오브젝트를 생성할 때 사용할 스토리지 클래스의 이름입니다.
4. 이 설정은 **wipefs** 를 호출할지 여부를 정의합니다. 즉, 파티션 테이블 서명(마이크 문자열)을 제거하여 Local Storage Operator 프로비저닝에 디스크를 사용할 준비가 되었습니다. 서명 이외의 다른 데이터는 삭제되지 않습니다. 기본값은 "false"입니다(**wipefs** 가 호출되지 않음). **forceWipeDevicesAndDestroyAllData** 를 "true"로 설정하면 이전 데이터를 다시 사용해야 하는 디스크에 남아 있을 수 있는 시나리오에서 유용할 수 있습니다. 이러한 시나리오에서는 이 필드를 true로 설정하면 관리자가 디스크를 수동으로 지울 필요가 없습니다.
5. 선택할 로컬 스토리지 장치 목록이 포함된 경로입니다. 블록 장치에 샌드박스 컨테이너 노드를 배포할 때 이 경로를 사용해야 합니다.
6. 이 값을 **LocalVolume** 리소스의 실제 로컬 디스크 파일 경로(예: **/dev/disk/ by-id /wwn**)로 바꿉니다. 프로비저너가 배포되면 이러한 로컬 디스크에 PV가 생성됩니다.

2. OpenShift Container Platform 클러스터에 로컬 볼륨 리소스를 생성합니다. 방금 생성한 파일을 지정합니다.

```
$ oc create -f <local-volume>.yaml
```

3. 프로비저너가 생성되었고 해당 데몬 세트가 생성되었는지 확인합니다.

```
$ oc get all -n openshift-local-storage
```

출력 예

```
NAME                                      READY   STATUS    RESTARTS   AGE
pod/diskmaker-manager-9wzms               1/1     Running   0          5m43s
pod/diskmaker-manager-jgvjp               1/1     Running   0          5m43s
pod/diskmaker-manager-tbdsj               1/1     Running   0          5m43s
pod/local-storage-operator-7db4bd9f79-t6k87   1/1     Running   0          14m

NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/local-storage-operator-metrics   ClusterIP   172.30.135.36   <none>
8383/TCP,8686/TCP   14m

NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR   AGE
daemonset.apps/diskmaker-manager   3         3         3       3            3           <none>
5m43s
```

```
NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/local-storage-operator   1/1     1            1           14m

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/local-storage-operator-7db4bd9f79   1         1         1       14m
```

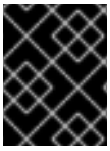
원하는 데몬 세트 프로세스 및 **현재** 개수를 기록해 둡니다. **원하는** 개수가 **0** 이면 라벨 선택기가 유효하지 않음을 나타냅니다.

4. 영구 볼륨이 생성되었는지 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available           local-sc                88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available           local-sc
82m
local-pv-3fa1c73    100Gi      RWO            Delete           Available           local-sc                48m
```



중요

LocalVolume 오브젝트를 편집해도 안전하지 않은 작업이 발생할 수 있으므로 기존 영구 볼륨이 변경되지 않습니다.

2.3.2. 선택 사항: 블록 장치에 노드 배포

OpenShift 샌드박스 컨테이너의 로컬 블록 볼륨을 프로비저닝하는 경우 정의된 볼륨 리소스에 지정된 경로의 블록 장치에 노드를 배포하도록 선택할 수 있습니다.

사전 요구 사항

- Local Storage Operator를 사용하여 블록 장치를 프로비저닝

프로세스

- 블록 장치를 사용하여 배포할 각 노드에 대해 다음 명령을 실행합니다.

```
$ oc debug node/worker-0 -- chcon -vt container_file_t /host/path/to/device
```

+ **/path/to/device** 는 로컬 스토리지 리소스를 생성할 때 정의한 경로와 동일해야 합니다.

+ .출력 예

```
system_u:object_r:container_file_t:s0 /host/path/to/device
```

2.3.3. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata** 를 런타임 클래스로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

KataConfig CR을 생성하면 OpenShift 샌드박스 컨테이너 Operator가 다음을 수행합니다.

- RHCOS 노드에 QEMU 및 **kata-containers** 와 같은 필요한 RHCOS 확장을 설치합니다.
- CRI-O 런타임이 올바른 런타임 처리기로 구성되었는지 확인합니다.
- 기본 구성으로 **kata** 라는 **RuntimeClass** CR을 생성합니다. 이를 통해 사용자는 **RuntimeClassName** 필드에서 CR을 참조하여 **kata** 를 런타임으로 사용하도록 워크로드를 구성할 수 있습니다. 이 CR은 런타임의 리소스 오버헤드도 지정합니다.

OpenShift 샌드박스 컨테이너는 **kata** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.

중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 선택 사항: 노드 자격 검사를 활성화하려면 Node Feature Discovery Operator를 설치했습니다.

프로세스

1. 다음 예에 따라 **cluster-kataconfig.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  checkNodeEligibility: false (1)
  logLevel: info
```

(1) 선택 사항: 노드 자격 검사를 실행하려면'checkNodeEligibility'를 **true** 로 설정합니다.

2. 선택 사항: 선택한 노드에 **kata** 를 설치하려면 다음 예에 따라 노드 라벨을 지정합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
```

```
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 선택한 노드의 라벨을 지정합니다.

3. **KataConfig** CR을 생성합니다.

```
$ oc create -f cluster-kataconfig.yaml
```

새로운 **KataConfig** CR이 생성되고 작업자 노드에 **kata** 를 런타임 클래스로 설치합니다.

kata 설치가 완료되고 설치를 확인하기 전에 작업자 노드가 재부팅될 때까지 기다립니다.

검증

- 다음 명령을 실행하여 설치 진행 상황을 모니터링합니다.

```
$ watch "oc describe kataconfig | sed -n /^Status:/,/^Events/p"
```

kataNodes 아래의 모든 작업자의 상태가 **설치되고** 이유를 지정하지 않고 **InProgress** 조건이 **False** 이면 클러스터에 **kata** 가 설치됩니다.

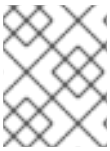
자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

2.3.4. 선택 사항: Pod 오버헤드 수정

Pod 오버헤드는 노드의 Pod에서 사용하는 시스템 리소스의 양을 설명합니다. **RuntimeClass** 사용자 정의 리소스의 **spec.overhead** 필드를 변경하여 Pod 오버헤드를 수정할 수 있습니다. 예를 들어 컨테이너에 대해 실행하는 구성이 QEMU 프로세스 및 게스트 커널 데이터에 350Mi 이상의 메모리를 사용하는 경우 필요에 맞게 **RuntimeClass** 오버헤드를 변경할 수 있습니다.

게스트에서 모든 유형의 파일 시스템 I/O를 수행할 때 게스트 커널에 파일 버퍼가 할당됩니다. 파일 버퍼는 **virtiofsd** 프로세스뿐만 아니라 호스트의 QEMU 프로세스에도 매핑됩니다.

예를 들어 게스트에서 300Mi 파일 버퍼 캐시를 사용하는 경우 QEMU와 **virtiofsd**는 모두 300Mi 추가 메모리를 사용하는 것처럼 나타납니다. 그러나 세 가지 경우 모두 동일한 메모리가 사용됩니다. 따라서 총 메모리 사용량은 300Mi에 불과하며 3개의 다른 위치에 매핑됩니다. 이는 메모리 사용량 메트릭을 보고할 때 올바르게 계산됩니다.



참고

기본값은 Red Hat에서 지원합니다. 기본 오버헤드 값 변경은 지원되지 않으며 값을 변경하면 기술적인 문제가 발생할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 **RuntimeClass** 오브젝트를 가져옵니다.

```
$ oc describe runtimeclass kata
```

2. **overhead.podFixed.memory** 및 **cpu** 값을 업데이트하고 **RuntimeClass.yaml** 로 저장합니다.

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata
overhead:
  podFixed:
    memory: "500Mi"
    cpu: "500m"
```

2.3.5. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트

중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. 다음 예와 같이 **spec.runtimeClassName: kata** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata
# ...
```

OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

3장. 퍼블릭 클라우드에 워크로드 배포

AWS 클라우드 컴퓨팅 서비스 및 Microsoft Azure 클라우드 컴퓨팅 서비스에 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

클러스터 요구 사항

- Red Hat OpenShift Container Platform 4.13 이상을 설치했습니다.
- 클러스터에 작업자 노드가 하나 이상 있습니다.

3.1. AWS에 워크로드 배포

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 AWS 클라우드 컴퓨팅 서비스에 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

배포 워크플로

1. 포트를 활성화합니다.
2. AWS의 시크릿을 생성합니다.
3. AWS의 구성 맵을 생성합니다.
4. **KataConfig** 사용자 지정 리소스를 생성합니다.
5. 선택 사항: 노드당 피어 Pod VM 제한을 수정합니다.
6. **kata-remote** 런타임 클래스를 사용하도록 워크로드 오브젝트를 구성합니다.

3.1.1. 환경 준비

환경을 준비하려면 다음 단계를 수행합니다.

1. 클러스터에 충분한 리소스가 있는지 확인합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 설치합니다.
3. 포트 15150 및 9000을 활성화하여 피어 Pod와의 내부 통신을 허용합니다.

3.1.1.1. 리소스 요구 사항

피어 Pod 가상 머신(VM)에는 다음 두 위치에 있는 리소스가 필요합니다.

- 작업자 노드입니다. 작업자 노드는 메타데이터, Kata shim 리소스(**containerd-shim-kata-v2**), remote-hypervisor 리소스(**cloud-api-adaptor**) 및 작업자 노드와 피어 Pod VM 간의 터널 설정을 저장합니다.
- 클라우드 인스턴스입니다. 클라우드에서 실행되는 실제 피어 Pod VM입니다.

Kubernetes 작업자 노드에 사용되는 CPU 및 메모리 리소스는 피어 Pod를 생성하는 데 사용되는 RuntimeClass(**kata-remote**) 정의에 포함된 Pod 오버헤드 에 의해 처리됩니다.

클라우드에서 실행되는 총 피어 Pod VM 수는 Kubernetes 노드 확장 리소스로 정의됩니다. 이 제한은 노드당이며 **peerpodConfig** CR(사용자 정의 리소스)의 **limit** 속성으로 설정됩니다.

peerpodconfig-openshift 라는 **peerpodConfig** CR은 **kataConfig** CR을 생성하고 피어 Pod를 활성화할 때 생성되며 **openshift-sandboxed-containers-operator** 네임스페이스에 있습니다.

다음 **peerpodConfig** CR 예제에서는 기본 **사양** 값을 표시합니다.

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" (1)
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

1 기본 제한은 노드당 VM 10개입니다.

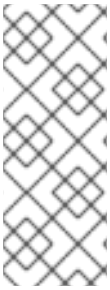
확장된 리소스의 이름은 **kata.peerpods.io/vm** 이며 Kubernetes 스케줄러에서 용량 추적 및 계정을 처리할 수 있습니다.

환경의 요구 사항에 따라 노드당 제한을 편집할 수 있습니다. 자세한 내용은 "Peer pods에서 노드당 VM 제한 수정"을 참조하십시오.

변경 웹 후크 는 확장된 리소스 **kata.peerpods.io/vm** 을 Pod 사양에 추가합니다. 또한 Pod 사양에서 리소스별 항목도 제거합니다(있는 경우). 이를 통해 Kubernetes 스케줄러에서 이러한 확장 리소스를 고려하여 리소스를 사용할 수 있는 경우에만 피어 Pod를 예약할 수 있습니다.

변경 웹 후크는 다음과 같이 Kubernetes Pod를 수정합니다.

- 변경 웹 후크는 **TARGET_RUNTIME_CLASS** 환경 변수에 지정된 예상 **RuntimeClassName** 값을 Pod에 확인합니다. Pod 사양의 값이 **TARGET_RUNTIME_CLASS** 의 값과 일치하지 않으면 Pod를 수정하지 않고 웹 후크가 종료됩니다.
- **RuntimeClassName** 값이 일치하는 경우 Webhook에서 Pod 사양을 다음과 같이 변경합니다.
  1. Webhook는 Pod에 있는 모든 컨테이너 및 init 컨테이너의 **resources** 필드에서 모든 리소스 사양을 제거합니다.
  2. Webhook는 Pod의 첫 번째 컨테이너의 resources 필드를 수정하여 확장 리소스 (**kata.peerpods.io/vm**)를 사양에 추가합니다. 확장된 리소스 **kata.peerpods.io/vm** 은 회계 목적으로 Kubernetes 스케줄러에서 사용합니다.



참고

변경 웹 후크는 OpenShift Container Platform의 특정 시스템 네임스페이스가 변경되지 않습니다. 해당 시스템 네임스페이스에 피어 Pod가 생성되면 Pod 사양에 확장 리소스가 포함되지 않는 한 Kubernetes 확장 리소스를 사용하는 리소스 계정이 작동하지 않습니다.

특정 네임스페이스에서 피어 Pod 생성만 허용하도록 클러스터 전체 정책을 정의하는 것이 좋습니다.

3.1.1.2. AWS의 포트 활성화

AWS에서 실행되는 피어 Pod와의 내부 통신을 허용하려면 포트 15150 및 9000을 활성화해야 합니다.

사전 요구 사항

- OpenShift 샌드박스 컨테이너 Operator가 설치되어 있습니다.
- AWS 명령줄 툴을 설치했습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. OpenShift Container Platform 클러스터에 로그인하고 인스턴스 ID를 검색합니다.

```
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o
jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
```

2. AWS 리전을 검색합니다.

```
$ AWS_REGION=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.aws.region}')
```

3. 보안 그룹 ID를 검색하여 배열에 저장합니다.

```
$ AWS_SG_IDS=($(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --output text --region
$AWS_REGION))
```

4. 각 보안 그룹 ID에 대해 피어 pod shim에 kata-agent 통신에 액세스하고 피어 Pod 터널을 설정하도록 권한을 부여합니다.

```
$ for AWS_SG_ID in "${AWS_SG_IDS[@]}"; do

    aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
15150 --source-group $AWS_SG_ID --region $AWS_REGION

    aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
9000 --source-group $AWS_SG_ID --region $AWS_REGION

done
```

이제 포트가 활성화됩니다.

3.1.1.3. OpenShift 샌드박스 컨테이너 Operator 설치

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

3.1.1.3.1. 웹 콘솔을 사용하여 Operator 설치

Red Hat OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**로 이동합니다.
2. **키워드로 필터링** 필드에 **OpenShift sandboxed containers**를 입력합니다.
3. **OpenShift 샌드박스 컨테이너 Operator** 타일을 선택하고 **설치를** 클릭합니다.
4. **Operator 설치** 페이지의 사용 가능한 **업데이트 채널** 옵션 목록에서 **stable** 을 선택합니다.
5. 설치된 네임스페이스 용으로 **Operator 권장 네임스페이스** 가 선택되어 **있는지 확인합니다**. 이렇게 하면 필수 **openshift-sandboxed-containers-operator** 네임스페이스에 Operator가 설치됩니다. 이 네임스페이스가 아직 존재하지 않으면 자동으로 생성됩니다.

참고

openshift-sandboxed-containers-operator 이외의 네임스페이스에 OpenShift 샌드박스 컨테이너 Operator를 설치하려고 하면 설치가 실패합니다.

6. **승인 전략에 대해 자동 이 선택되어 있는지 확인합니다. Automatic** 은 기본값이며 새 z-stream 릴리스를 사용할 수 있을 때 OpenShift 샌드박스 컨테이너에 대한 자동 업데이트를 활성화합니다.
7. **설치**를 클릭합니다.

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

1. **Operators → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator가 표시되는지 확인합니다.

추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경에 대한 Operator Lifecycle Manager에서 프록시 지원 구성 .

3.1.1.3.2. CLI를 사용하여 Operator 설치

CLI를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. **Namespace.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. 다음 명령을 실행하여 네임스페이스를 생성합니다.

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

3. **OperatorGroup.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

4. 다음 명령을 실행하여 operator 그룹을 생성합니다.

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

5. **Subscription.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0
```

6. 다음 명령을 실행하여 서브스크립션을 생성합니다.

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

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

- 다음 명령을 실행하여 Operator가 올바르게 설치되었는지 확인합니다.

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

출력 예

```
```

```
NAME                    DISPLAY                                 VERSION   REPLACES
PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded
```

3.1.1.3.3. 추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경을 위한 Operator Lifecycle Manager에서 프록시 지원 구성

3.1.2. 웹 콘솔을 사용하여 워크로드 배포

웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

3.1.2.1. 시크릿 생성

OpenShift Container Platform 클러스터에 **Secret** 오브젝트를 생성해야 합니다. 시크릿은 Pod VM(가상 머신) 이미지 및 피어 Pod 인스턴스를 생성하기 위한 클라우드 공급자 자격 증명을 저장합니다. 기본적으로 OpenShift 샌드박스 컨테이너 Operator는 클러스터를 생성하는 데 사용되는 인증 정보를 기반으로 보안을 생성합니다. 그러나 다른 인증 정보를 사용하는 보안을 수동으로 생성할 수 있습니다.

사전 요구 사항

- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**
  이러한 값은 AWS 콘솔에서 생성할 수 있습니다.

프로세스

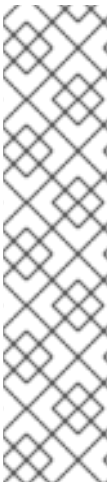
1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator 타일을 클릭합니다.
3. 오른쪽 상단에 있는 가져오기 아이콘(**+**)을 클릭합니다.
4. **YAML 가져오기 창에서 다음 YAML** 매니페스트를 붙여넣습니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: "<aws_access_key>" (1)
  AWS_SECRET_ACCESS_KEY: "<aws_secret_access_key>" (2)
```

(1) **AWS_ACCESS_KEY_ID** 값을 지정합니다.

(2) **AWS_SECRET_ACCESS_KEY** 값을 지정합니다.

5. **저장을** 클릭하여 변경 사항을 적용합니다.



참고

피어 Pod 시크릿을 업데이트하는 경우 **peerpodconfig-ctrl-caa-daemon** DaemonSet을 다시 시작하여 변경 사항을 적용해야 합니다.

시크릿을 업데이트한 후 **저장을** 클릭하여 변경 사항을 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

데몬 세트를 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod는 업데이트하지 않습니다.

검증

- **워크로드 → 시크릿 으로** 이동하여 시크릿을 확인합니다.

3.1.2.2. 구성 맵 생성

클라우드 공급자의 OpenShift Container Platform 클러스터에 구성 맵을 생성해야 합니다.

AMI(Amazon Machine Image) ID를 설정해야 합니다. 구성 맵을 생성하기 전에 이 값을 검색할 수 있습니다.

프로세스

1. AWS 인스턴스에서 다음 값을 가져옵니다.
   a. 인스턴스 ID를 검색하고 기록합니다.

   ```
   $ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
   ```

   이는 secret 오브젝트의 다른 값을 검색하는 데 사용됩니다.

   b. AWS 리전을 검색하고 기록합니다.

   ```
   $ AWS_REGION=$(oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.aws.region}') && echo "AWS_REGION: \"$AWS_REGION\""
   ```

   c. AWS 서브넷 ID를 검색하고 기록합니다.

   ```
   $ AWS_SUBNET_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query 'Reservations[*].Instances[*].SubnetId' --region ${AWS_REGION} --output text) && echo "AWS_SUBNET_ID: \"$AWS_SUBNET_ID\""
   ```

   d. AWS VPC ID를 검색하고 기록합니다.

```
$ AWS_VPC_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].VpcId' --region ${AWS_REGION} --output text) && echo
"AWS_VPC_ID: \"$AWS_VPC_ID\""
```

e. AWS 보안 그룹 ID를 검색하고 기록합니다.

```
$ AWS_SG_IDS=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --region ${AWS_REGION} --
output text)
&& echo "AWS_SG_IDS: \"$AWS_SG_IDS\""
```

2. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
3. Operator 목록에서 OpenShift 샌드박스 컨테이너 Operator를 선택합니다.
4. 오른쪽 상단에 있는 가져오기 아이콘(**+**)을 클릭합니다.
5. **YAML 가져오기 창에서 다음 YAML** 매니페스트를 붙여넣습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "aws"
  VXLAN_PORT: "9000"
  PODVM_INSTANCE_TYPE: "t3.medium" (1)
  PODVM_INSTANCE_TYPES: "t2.small,t2.medium,t3.large" (2)
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  PODVM_AMI_ID: "<podvm_ami_id>" (3)
  AWS_REGION: "<aws_region>" (4)
  AWS_SUBNET_ID: "<aws_subnet_id>" (5)
  AWS_VPC_ID: "<aws_vpc_id>" (6)
  AWS_SG_IDS: "<aws_sg_ids>" (7)
```

(1) 유형이 워크로드에 정의되지 않은 경우 사용되는 기본 인스턴스 유형을 정의합니다.

(2) Pod를 생성할 때 지정할 수 있는 모든 인스턴스 유형을 나열합니다. 이를 통해 더 적은 메모리와 더 적은 CPU 또는 대규모 워크로드에 대해 더 큰 인스턴스 유형이 필요한 워크로드에 대해 더 작은 인스턴스 유형을 정의할 수 있습니다.

(3) 선택 사항: 기본적으로 클러스터 인증 정보를 기반으로 AMI ID를 사용하여 **KataConfig** CR을 실행할 때 이 값이 채워집니다. 고유한 AMI를 생성하는 경우 올바른 AMI ID를 지정합니다.

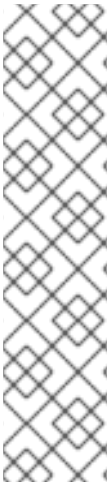
(4) 검색한 **AWS_REGION** 값을 지정합니다.

(5) 검색한 **AWS_SUBNET_ID** 값을 지정합니다.

(6) 검색한 **AWS_VPC_ID** 값을 지정합니다.

(7) 검색한 **AWS_SG_IDS** 값을 지정합니다.

6. **저장을** 클릭하여 변경 사항을 적용합니다.
   클라우드 공급자에 대한 구성 맵이 생성됩니다.



참고

피어 Pod 구성 맵을 업데이트하는 경우 변경 사항을 적용하려면 **peerpodconfig-ctrl-caa-daemon** 데몬 세트를 다시 시작해야 합니다.

구성 맵을 업데이트한 후 **저장을** 클릭하여 변경 사항을 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

daemonset을 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod를 업데이트하지 않습니다.

검증

- **워크로드 → ConfigMap** 으로 이동하여 새 구성 맵을 확인합니다.

3.1.2.3. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata-remote** 를 **RuntimeClass** 로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

kata-remote 런타임 클래스는 기본적으로 모든 작업자 노드에 설치됩니다. 특정 노드에만 **kata-remote** 를 설치하려면 해당 노드에 레이블을 추가한 다음 **KataConfig** CR에 레이블을 정의할 수 있습니다.

OpenShift 샌드박스 컨테이너는 **kata-remote** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.

중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 다음 요인은 재부팅 시간을 늘릴 수 있습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.

2. OpenShift 샌드박스 컨테이너 Operator를 선택합니다.
3. **KataConfig** 탭에서 **KataConfig 만들기** 를 클릭합니다.
4. 다음 세부 정보를 입력합니다.
   - **이름**: 선택 사항: 기본 이름은 **example-kataconfig** 입니다.
   - **labels**: 선택 사항: **KataConfig** 리소스에 대한 특성을 식별하는 모든 관련 정보를 입력합니다. 각 레이블은 키-값 쌍을 나타냅니다.
   - **enablePeerPods**: 퍼블릭 클라우드, IBM Z® 및 IBM® LinuxONE 배포에는 선택합니다.
   - **kataConfigPoolSelector**. 선택 사항: 선택한 노드에 **kata-remote** 를 설치하려면 선택한 노드의 라벨에 일치하는 표현식을 추가합니다.
     a. **kataConfigPoolSelector** 영역을 확장합니다.
     b. **kataConfigPoolSelector** 영역에서 **matchExpressions** 를 확장합니다. 이는 라벨 선택기 요구 사항 목록입니다.
     c. **matchExpressions 추가를** 클릭합니다.
     d. **키** 필드에 선택기가 적용되는 라벨 키를 입력합니다.
     e. **Operator** 필드에 레이블 값과의 키 관계를 입력합니다. 유효한 연산자는 **In**,**NotIn**,**Exists** 및 **DoesNotExist** 입니다.
     f. **값** 영역을 확장한 다음 **값 추가** 를 클릭합니다.
     g. **값** 필드에 **키** 레이블 값에 **true** 또는 **false** 를 입력합니다.
   - **loglevel**: **kata-remote** 런타임 클래스를 사용하여 노드에 대해 검색된 로그 데이터의 수준을 정의합니다.
5. **생성**을 클릭합니다. **KataConfig** CR이 생성되고 작업자 노드에 **kata-remote** 런타임 클래스를 설치합니다.
   설치를 확인하기 전에 **kata-remote** 설치가 완료되고 작업자 노드가 재부팅될 때까지 기다립니다.

검증

1. **KataConfig** 탭에서 **KataConfig** CR을 클릭하여 세부 정보를 확인합니다.
2. **YAML** 탭을 클릭하여 **상태 스탠자를 확인합니다**.
   상태 스탠자에는 **조건** 및 **kataNodes** 키가 포함되어 있습니다. **status.kataNodes** 의 값은 노드 배열이며 각 노드는 특정 **kata-remote** 설치의 노드를 나열합니다. 업데이트가 있을 때마다 메시지가 표시됩니다.
3. **Reload** (다시 로드)를 클릭하여 YAML을 새로 고칩니다.
   status.kataNodes 어레이의 모든 작업자가 **설치** 및 **조건.InProgress: False** 를 지정하는 이유 없이 False를 표시하면 클러스터에 **kata-remote** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

3.1.2.3.1. 선택 사항: Pod VM 이미지 확인

클러스터에 **kata-remote** 가 설치되면 OpenShift 샌드박스 컨테이너 Operator에서 피어 Pod를 생성하는

데 사용되는 Pod VM 이미지를 생성합니다. 이 프로세스는 클라우드 인스턴스에서 이미지가 생성되므로 시간이 오래 걸릴 수 있습니다. 클라우드 공급자에 대해 생성한 구성 맵을 확인하여 Pod VM 이미지가 성공적으로 생성되었는지 확인할 수 있습니다.

프로세스

1. **워크로드 → ConfigMap** 으로 이동합니다.
2. 공급자 구성 맵을 클릭하여 세부 정보를 확인합니다.
3. **YAML** 탭을 클릭합니다.
4. YAML 파일의 **상태** 스탠자를 확인합니다.
   PODVM_AMI_ID 매개변수가 채워지면 Pod VM 이미지가 성공적으로 생성됩니다.

문제 해결

1. 다음 명령을 실행하여 이벤트 로그를 검색합니다.

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. 다음 명령을 실행하여 작업 로그를 검색합니다.

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

문제를 해결할 수 없는 경우 Red Hat 지원 케이스를 제출하고 두 로그의 출력을 첨부합니다.

3.1.2.4. 선택 사항: 노드당 피어 Pod VM 수 수정

peerpodConfig CR(사용자 정의 리소스)을 편집하여 노드당 피어 Pod 가상 머신(VM) 제한을 변경할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 현재 제한을 확인합니다.

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator
\
-o jsonpath='{.spec.limit}{"\n"}'
```

2. 다음 명령을 실행하여 **peerpodConfig** CR의 **limit** 속성을 수정합니다.

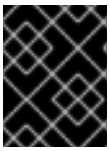
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' (1)
```

(1) <value>를 정의할 제한으로 바꿉니다.

3.1.2.5. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata-remote** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트



중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

YAML 파일에 주석을 추가하여 구성 맵에 정의한 기본 인스턴스 유형을 사용하여 워크로드를 배포해야 하는지 여부를 정의할 수 있습니다.

인스턴스 유형을 수동으로 정의하지 않으려면 사용 가능한 메모리에 따라 자동 인스턴스 유형을 사용하도록 주석을 추가할 수 있습니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **워크로드 → 워크로드** 유형(예: **Pod**)으로 이동합니다.
2. 워크로드 유형 페이지에서 오브젝트를 클릭하여 세부 정보를 확인합니다.
3. **YAML** 탭을 클릭합니다.
4. 다음 예와 같이 **spec.runtimeClassName: kata-remote** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

5. pod-templated 오브젝트에 주석을 추가하여 수동으로 정의된 인스턴스 유형 또는 자동 인스턴스 유형을 사용합니다.

- 수동으로 정의한 인스턴스 유형을 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: t3.medium (1)
# ...
```

(1) 구성 맵에서 정의한 인스턴스 유형을 지정합니다.

- 자동 인스턴스 유형을 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

워크로드에서 사용할 수 있는 메모리 양을 정의합니다. 워크로드는 사용 가능한 메모리 양에 따라 자동 인스턴스 유형에서 실행됩니다.

6. **저장을** 클릭하여 변경 사항을 적용합니다.
OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata-remote** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

3.1.3. 명령줄을 사용하여 워크로드 배포

명령줄을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

3.1.3.1. 시크릿 생성

OpenShift Container Platform 클러스터에 **Secret** 오브젝트를 생성해야 합니다. 시크릿은 Pod VM(가상 머신) 이미지 및 피어 Pod 인스턴스를 생성하기 위한 클라우드 공급자 자격 증명을 저장합니다. 기본적으로 OpenShift 샌드박스 컨테이너 Operator는 클러스터를 생성하는 데 사용되는 인증 정보를 기반으로 보안을 생성합니다. 그러나 다른 인증 정보를 사용하는 보안을 수동으로 생성할 수 있습니다.

사전 요구 사항

- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**
이러한 값은 AWS 콘솔에서 생성할 수 있습니다.

프로세스

1. 다음 예에 따라 **peer-pods-secret.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: "<aws_access_key>" 1
  AWS_SECRET_ACCESS_KEY: "<aws_secret_access_key>" 2
```

1. **AWS_ACCESS_KEY_ID** 값을 지정합니다.
2. **AWS_SECRET_ACCESS_KEY** 값을 지정합니다.

2. 매니페스트를 **적용하여 보안** 오브젝트를 생성합니다.

```
$ oc apply -f peer-pods-secret.yaml
```

참고

피어 Pod 시크릿을 업데이트하는 경우 **peerpodconfig-ctrl-caa-daemon** DaemonSet을 다시 시작하여 변경 사항을 적용해야 합니다.

시크릿을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

데몬 세트를 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod는 업데이트하지 않습니다.

3.1.3.2. 구성 맵 생성

클라우드 공급자의 OpenShift Container Platform 클러스터에 구성 맵을 생성해야 합니다.

AMI(Amazon Machine Image) ID를 설정해야 합니다. 구성 맵을 생성하기 전에 이 값을 검색할 수 있습니다.

프로세스

1. AWS 인스턴스에서 다음 값을 가져옵니다.
   a. 인스턴스 ID를 검색하고 기록합니다.

```
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
```

   이는 secret 오브젝트의 다른 값을 검색하는 데 사용됩니다.

   b. AWS 리전을 검색하고 기록합니다.

```
$ AWS_REGION=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.aws.region}') && echo "AWS_REGION:
\"$AWS_REGION\""
```

c. AWS 서브넷 ID를 검색하고 기록합니다.

```
$ AWS_SUBNET_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --
query 'Reservations[*].Instances[*].SubnetId' --region ${AWS_REGION} --output text) &&
echo "AWS_SUBNET_ID: \"$AWS_SUBNET_ID\""
```

d. AWS VPC ID를 검색하고 기록합니다.

```
$ AWS_VPC_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].VpcId' --region ${AWS_REGION} --output text) && echo
"AWS_VPC_ID: \"$AWS_VPC_ID\""
```

e. AWS 보안 그룹 ID를 검색하고 기록합니다.

```
$ AWS_SG_IDS=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --region ${AWS_REGION} --
output text)
&& echo "AWS_SG_IDS: \"$AWS_SG_IDS\""
```

2. 다음 예에 따라 **peer-pods-cm.yaml** 매니페스트를 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "aws"
  VXLAN_PORT: "9000"
  PODVM_INSTANCE_TYPE: "t3.medium" (1)
  PODVM_INSTANCE_TYPES: "t2.small,t2.medium,t3.large" (2)
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  PODVM_AMI_ID: "<podvm_ami_id>" (3)
  AWS_REGION: "<aws_region>" (4)
  AWS_SUBNET_ID: "<aws_subnet_id>" (5)
  AWS_VPC_ID: "<aws_vpc_id>" (6)
  AWS_SG_IDS: "<aws_sg_ids>" (7)
```

(1) 유형이 워크로드에 정의되지 않은 경우 사용되는 기본 인스턴스 유형을 정의합니다.

(2) Pod를 생성할 때 지정할 수 있는 모든 인스턴스 유형을 나열합니다. 이를 통해 더 적은 메모리와 더 적은 CPU 또는 대규모 워크로드에 대해 더 큰 인스턴스 유형이 필요한 워크로드에 대해 더 작은 인스턴스 유형을 정의할 수 있습니다.

(3) 선택 사항: 기본적으로 클러스터 인증 정보를 기반으로 AMI ID를 사용하여 **KataConfig** CR을 실행할 때 이 값이 채워집니다. 고유한 AMI를 생성하는 경우 올바른 AMI ID를 지정합니다.

(4) 검색한 **AWS_REGION** 값을 지정합니다.

5 검색한 **AWS_SUBNET_ID** 값을 지정합니다.

6 검색한 **AWS_VPC_ID** 값을 지정합니다.

7 검색한 **AWS_SG_IDS** 값을 지정합니다.

3. 매니페스트를 적용하여 구성 맵을 생성합니다.

```
$ oc apply -f peer-pods-cm.yaml
```

클라우드 공급자에 대한 구성 맵이 생성됩니다.

참고

피어 Pod 구성 맵을 업데이트하는 경우 변경 사항을 적용하려면 **peerpodconfig-ctrl-caa-daemon** 데몬 세트를 다시 시작해야 합니다.

구성 맵을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

daemonset을 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod를 업데이트하지 않습니다.

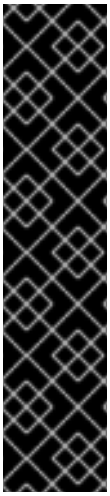
3.1.3.3. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata-remote** 를 런타임 클래스로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

KataConfig CR을 생성하면 OpenShift 샌드박스 컨테이너 Operator가 다음을 수행합니다.

- 기본 구성을 사용하여 **kata-remote** 라는 **RuntimeClass** CR을 생성합니다. 이를 통해 사용자는 **RuntimeClassName** 필드에서 CR을 참조하여 **kata-remote** 를 런타임으로 사용하도록 워크로드를 구성할 수 있습니다. 이 CR은 런타임의 리소스 오버헤드도 지정합니다.

OpenShift 샌드박스 컨테이너는 **kata-remote** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.



중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

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

프로세스

1. 다음 예에 따라 **cluster-kataconfig.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
```

2. 선택 사항: 선택한 노드에 **kata-remote** 를 설치하려면 다음 예에 따라 노드 라벨을 지정합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 선택한 노드의 라벨을 지정합니다.

3. **KataConfig** CR을 생성합니다.

```
$ oc create -f cluster-kataconfig.yaml
```

새로운 **KataConfig** CR이 생성되고 작업자 노드에 **kata-remote** 가 런타임 클래스로 설치됩니다.

설치를 확인하기 전에 **kata-remote** 설치가 완료되고 작업자 노드가 재부팅될 때까지 기다립니다.

검증

- 다음 명령을 실행하여 설치 진행 상황을 모니터링합니다.

```
$ watch "oc describe kataconfig | sed -n /^Status:/,/^Events/p"
```

kataNodes 아래의 모든 작업자의 상태가 **설치되고** 이유를 지정하지 않고 **InProgress** 조건이 **False** 이면 클러스터에 **kata-remote** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

3.1.3.3.1. 선택 사항: Pod VM 이미지 확인

클러스터에 **kata-remote** 가 설치되면 OpenShift 샌드박스 컨테이너 Operator에서 피어 Pod를 생성하는 데 사용되는 Pod VM 이미지를 생성합니다. 이 프로세스는 클라우드 인스턴스에서 이미지가 생성되므로 시간이 오래 걸릴 수 있습니다. 클라우드 공급자에 대해 생성한 구성 맵을 확인하여 Pod VM 이미지가 성

공적으로 생성되었는지 확인할 수 있습니다.

프로세스

1. 피어 Pod에 대해 생성한 구성 맵을 가져옵니다.

```
$ oc get configmap peer-pods-cm -n openshift-sandboxed-containers-operator -o yaml
```

2. YAML 파일의 **상태** 스탠자를 확인합니다.
PODVM_AMI_ID 매개변수가 채워지면 Pod VM 이미지가 성공적으로 생성됩니다.

문제 해결

1. 다음 명령을 실행하여 이벤트 로그를 검색합니다.

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. 다음 명령을 실행하여 작업 로그를 검색합니다.

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

문제를 해결할 수 없는 경우 Red Hat 지원 케이스를 제출하고 두 로그의 출력을 첨부합니다.

3.1.3.4. 선택 사항: 노드당 피어 Pod VM 수 수정

peerpodConfig CR(사용자 정의 리소스)을 편집하여 노드당 피어 Pod 가상 머신(VM) 제한을 변경할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 현재 제한을 확인합니다.

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator
\
-o jsonpath='{.spec.limit}{"\n"}'
```

2. 다음 명령을 실행하여 **peerpodConfig** CR의 **limit** 속성을 수정합니다.

```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

1 <value>를 정의할 제한으로 바꿉니다.

3.1.3.5. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata-remote** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트

- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트

중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

YAML 파일에 주석을 추가하여 구성 맵에 정의한 기본 인스턴스 유형을 사용하여 워크로드를 배포해야 하는지 여부를 정의할 수 있습니다.

인스턴스 유형을 수동으로 정의하지 않으려면 사용 가능한 메모리에 따라 자동 인스턴스 유형을 사용하도록 주석을 추가할 수 있습니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. 다음 예와 같이 **spec.runtimeClassName: kata-remote** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

2. pod-templated 오브젝트에 주석을 추가하여 수동으로 정의된 인스턴스 유형 또는 자동 인스턴스 유형을 사용합니다.
   - 수동으로 정의한 인스턴스 유형을 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: t3.medium (1)
# ...
```

(1) 구성 맵에서 정의한 인스턴스 유형을 지정합니다.

- 자동 인스턴스 유형을 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

  워크로드에서 사용할 수 있는 메모리 양을 정의합니다. 워크로드는 사용 가능한 메모리 양에 따라 자동 인스턴스 유형에서 실행됩니다.

3. 다음 명령을 실행하여 워크로드 오브젝트에 변경 사항을 적용합니다.

```
$ oc apply -f <object.yaml>
```

   OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata-remote** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

3.2. AZURE에 워크로드 배포

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 Microsoft Azure 클라우드 컴퓨팅 서비스에 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

배포 워크플로

1. Azure 액세스 키에 대한 시크릿을 생성합니다.
2. 구성 맵을 생성하여 Azure 인스턴스 크기 및 기타 매개변수를 정의합니다.
3. SSH 키 시크릿을 생성합니다.
4. **KataConfig** 사용자 지정 리소스를 생성합니다.
5. 선택 사항: 노드당 피어 Pod VM 제한을 수정합니다.
6. **kata-remote** 런타임 클래스를 사용하도록 워크로드 오브젝트를 구성합니다.

3.2.1. 환경 준비

환경을 준비하려면 다음 단계를 수행합니다.

1. 클러스터에 충분한 리소스가 있는지 확인합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 설치합니다.

3.2.1.1. 리소스 요구 사항

피어 Pod 가상 머신(VM)에는 다음 두 위치에 있는 리소스가 필요합니다.

- 작업자 노드입니다. 작업자 노드는 메타데이터, Kata shim 리소스(**containerd-shim-kata-v2**), remote-hypervisor 리소스(**cloud-api-adaptor**) 및 작업자 노드와 피어 Pod VM 간의 터널 설정을 저장합니다.

- 클라우드 인스턴스입니다. 클라우드에서 실행되는 실제 피어 Pod VM입니다.

Kubernetes 작업자 노드에 사용되는 CPU 및 메모리 리소스는 피어 Pod를 생성하는 데 사용되는 RuntimeClass(**kata-remote**) 정의에 포함된 Pod 오버헤드 에 의해 처리됩니다.

클라우드에서 실행되는 총 피어 Pod VM 수는 Kubernetes 노드 확장 리소스로 정의됩니다. 이 제한은 노드당이며 **peerpodConfig** CR(사용자 정의 리소스)의 **limit** 속성으로 설정됩니다.

peerpodconfig-openshift 라는 **peerpodConfig** CR은 **kataConfig** CR을 생성하고 피어 Pod를 활성화할 때 생성되며 **openshift-sandboxed-containers-operator** 네임스페이스에 있습니다.

다음 **peerpodConfig** CR 예제에서는 기본 **사양** 값을 표시합니다.

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" (1)
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

1 기본 제한은 노드당 VM 10개입니다.

확장된 리소스의 이름은 **kata.peerpods.io/vm** 이며 Kubernetes 스케줄러에서 용량 추적 및 계정을 처리할 수 있습니다.

환경의 요구 사항에 따라 노드당 제한을 편집할 수 있습니다. 자세한 내용은 "Peer pods에서 노드당 VM 제한 수정"을 참조하십시오.

변경 웹 후크 는 확장된 리소스 **kata.peerpods.io/vm** 을 Pod 사양에 추가합니다. 또한 Pod 사양에서 리소스별 항목도 제거합니다(있는 경우). 이를 통해 Kubernetes 스케줄러에서 이러한 확장 리소스를 고려하여 리소스를 사용할 수 있는 경우에만 피어 Pod를 예약할 수 있습니다.

변경 웹 후크는 다음과 같이 Kubernetes Pod를 수정합니다.

- 변경 웹 후크는 **TARGET_RUNTIME_CLASS** 환경 변수에 지정된 예상 **RuntimeClassName** 값을 Pod에 확인합니다. Pod 사양의 값이 **TARGET_RUNTIME_CLASS** 의 값과 일치하지 않으면 Pod를 수정하지 않고 웹 후크가 종료됩니다.

- **RuntimeClassName** 값이 일치하는 경우 Webhook에서 Pod 사양을 다음과 같이 변경합니다.

  1. Webhook는 Pod에 있는 모든 컨테이너 및 init 컨테이너의 **resources** 필드에서 모든 리소스 사양을 제거합니다.

  2. Webhook는 Pod의 첫 번째 컨테이너의 resources 필드를 수정하여 확장 리소스(**kata.peerpods.io/vm**)를 사양에 추가합니다. 확장된 리소스 **kata.peerpods.io/vm** 은 회계 목적으로 Kubernetes 스케줄러에서 사용합니다.

참고

변경 웹 후크는 OpenShift Container Platform의 특정 시스템 네임스페이스가 변경되지 않습니다. 해당 시스템 네임스페이스에 피어 Pod가 생성되면 Pod 사양에 확장 리소스가 포함되지 않는 한 Kubernetes 확장 리소스를 사용하는 리소스 계정이 작동하지 않습니다.

특정 네임스페이스에서 피어 Pod 생성만 허용하도록 클러스터 전체 정책을 정의하는 것이 좋습니다.

3.2.1.2. OpenShift 샌드박스 컨테이너 Operator 설치

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

3.2.1.2.1. 웹 콘솔을 사용하여 Operator 설치

Red Hat OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**로 이동합니다.
2. **키워드로 필터링** 필드에 **OpenShift sandboxed containers**를 입력합니다.
3. **OpenShift 샌드박스 컨테이너 Operator** 타일을 선택하고 **설치를** 클릭합니다.
4. **Operator 설치** 페이지의 사용 가능한 **업데이트 채널** 옵션 목록에서 **stable** 을 선택합니다.
5. 설치된 네임스페이스 용으로 **Operator 권장 네임스페이스** 가 선택되어 **있는지 확인합니다**. 이렇게 하면 필수 **openshift-sandboxed-containers-operator** 네임스페이스에 Operator가 설치됩니다. 이 네임스페이스가 아직 존재하지 않으면 자동으로 생성됩니다.

   참고

   openshift-sandboxed-containers-operator 이외의 네임스페이스에 OpenShift 샌드박스 컨테이너 Operator를 설치하려고 하면 설치가 실패합니다.

6. **승인 전략에 대해 자동 이 선택되어 있는지 확인합니다. Automatic** 은 기본값이며 새 z-stream 릴리스를 사용할 수 있을 때 OpenShift 샌드박스 컨테이너에 대한 자동 업데이트를 활성화합니다.
7. **설치**를 클릭합니다.

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

1. **Operators → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator가 표시되는지 확인합니다.

추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경에 대한 Operator Lifecycle Manager에서 프록시 지원 구성 .

3.2.1.2.2. CLI를 사용하여 Operator 설치

CLI를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. **Namespace.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. 다음 명령을 실행하여 네임스페이스를 생성합니다.

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

3. **OperatorGroup.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

4. 다음 명령을 실행하여 operator 그룹을 생성합니다.

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

5. **Subscription.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
```

```
name: sandboxed-containers-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
startingCSV: sandboxed-containers-operator.v1.6.0
```

6. 다음 명령을 실행하여 서브스크립션을 생성합니다.

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

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

- 다음 명령을 실행하여 Operator가 올바르게 설치되었는지 확인합니다.

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

출력 예

```
NAME                                DISPLAY                                  VERSION   REPLACES
PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded
```

3.2.1.2.3. 추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경을 위한 Operator Lifecycle Manager에서 프록시 지원 구성

3.2.2. 웹 콘솔을 사용하여 워크로드 배포

웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

3.2.2.1. 시크릿 생성

OpenShift Container Platform 클러스터에 **Secret** 오브젝트를 생성해야 합니다. 시크릿은 Pod VM(가상 머신) 이미지 및 피어 Pod 인스턴스를 생성하기 위한 클라우드 공급자 자격 증명을 저장합니다. 기본적으로 OpenShift 샌드박스 컨테이너 Operator는 클러스터를 생성하는 데 사용되는 인증 정보를 기반으로 보안을 생성합니다. 그러나 다른 인증 정보를 사용하는 보안을 수동으로 생성할 수 있습니다.

사전 요구 사항

- Azure CLI 도구를 설치하고 구성했습니다.

프로세스

1. Azure 서브스크립션 ID를 검색합니다.

```
$ AZURE_SUBSCRIPTION_ID=$(az account list --query "[?isDefault].id" -o tsv) && echo
"AZURE_SUBSCRIPTION_ID: \"$AZURE_SUBSCRIPTION_ID\""
```

2. RBAC 콘텐츠를 생성합니다. 이렇게 하면 클라이언트 ID, 클라이언트 시크릿, 테넌트 ID가 생성됩니다.

```
$ az ad sp create-for-rbac --role Contributor --scopes
/subscriptions/$AZURE_SUBSCRIPTION_ID --query "{ client_id: appId, client_secret:
password, tenant_id: tenant }
```

출력 예:

```
{
  "client_id": `AZURE_CLIENT_ID`,
  "client_secret": `AZURE_CLIENT_SECRET`,
  "tenant_id": `AZURE_TENANT_ID`
}
```

3. **보안** 오브젝트에 사용할 RBAC 출력을 기록합니다.

4. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.

5. OpenShift 샌드박스 컨테이너 Operator 타일을 클릭합니다.

6. 오른쪽 상단에 있는 가져오기 아이콘(**+**)을 클릭합니다.

7. **YAML 가져오기 창에서 다음 YAML** 매니페스트를 붙여넣습니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AZURE_CLIENT_ID: "<azure_client_id>" (1)
  AZURE_CLIENT_SECRET: "<azure_client_secret>" (2)
  AZURE_TENANT_ID: "<azure_tenant_id>" (3)
  AZURE_SUBSCRIPTION_ID: "<azure_subscription_id>" (4)
```

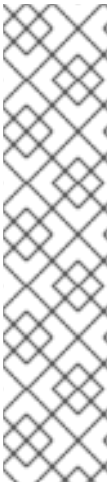
(1) **AZURE_CLIENT_ID** 값을 지정합니다.

(2) **AZURE_CLIENT_SECRET** 값을 지정합니다.

(3) **AZURE_TENANT_ID** 값을 지정합니다.

(4) **AZURE_SUBSCRIPTION_ID** 값을 지정합니다.

8. **저장을** 클릭하여 변경 사항을 적용합니다.



참고

피어 Pod 시크릿을 업데이트하는 경우 **peerpodconfig-ctrl-caa-daemon** DaemonSet을 다시 시작하여 변경 사항을 적용해야 합니다.

시크릿을 업데이트한 후 **저장을** 클릭하여 변경 사항을 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

데몬 세트를 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod는 업데이트하지 않습니다.

검증

- **워크로드 → 시크릿 으로** 이동하여 시크릿을 확인합니다.

3.2.2.2. 구성 맵 생성

클라우드 공급자의 OpenShift Container Platform 클러스터에 구성 맵을 생성해야 합니다.

프로세스

1. Azure 인스턴스에서 다음 값을 가져옵니다.
   a. Azure VNet 이름을 검색하고 기록합니다.

   ```
   $ AZURE_VNET_NAME=$(az network vnet list --resource-group ${AZURE_RESOURCE_GROUP} --query "[].{Name:name}" --output tsv)
   ```

   이 값은 Azure 서브넷 ID를 검색하는 데 사용됩니다.

   b. Azure 서브넷 ID를 검색하고 기록합니다.

   ```
   $ AZURE_SUBNET_ID=$(az network vnet subnet list --resource-group ${AZURE_RESOURCE_GROUP} --vnet-name $AZURE_VNET_NAME --query "[].{Id:id} | [? contains(Id, 'worker')]" --output tsv) && echo "AZURE_SUBNET_ID: \"$AZURE_SUBNET_ID\""
   ```

   c. Azure NSS(Network Security Group) ID를 검색하고 기록합니다.

   ```
   $ AZURE_NSG_ID=$(az network nsg list --resource-group ${AZURE_RESOURCE_GROUP} --query "[].{Id:id}" --output tsv) && echo "AZURE_NSG_ID: \"$AZURE_NSG_ID\""
   ```

   d. Azure 리소스 그룹을 검색하고 기록합니다.

   ```
   $ AZURE_RESOURCE_GROUP=$(oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.azure.resourceGroupName}') && echo "AZURE_RESOURCE_GROUP: \"$AZURE_RESOURCE_GROUP\""
   ```

   e. Azure 리전을 검색하고 기록합니다.

```
$ AZURE_REGION=$(az group show --resource-group
${AZURE_RESOURCE_GROUP} --query "{Location:location}" --output tsv) && echo
"AZURE_REGION: \"$AZURE_REGION\""
```

2. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
3. Operator 목록에서 OpenShift 샌드박스 컨테이너 Operator를 선택합니다.
4. 오른쪽 상단에 있는 가져오기 아이콘(+)을 클릭합니다.
5. **YAML 가져오기 창에서 다음 YAML** 매니페스트를 붙여넣습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_B2als_v2" (1)
  AZURE_INSTANCE_SIZES:
"Standard_B2als_v2,Standard_D2as_v5,Standard_D4as_v5,Standard_D2ads_v5" (2)
  AZURE_SUBNET_ID: "<azure_subnet_id>" (3)
  AZURE_NSG_ID: "<azure_nsg_id>" (4)
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  AZURE_IMAGE_ID: "<azure_image_id>" (5)
  AZURE_REGION: "<azure_region>" (6)
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" (7)
```

(1) 유형이 워크로드에 정의되지 않은 경우 사용되는 기본 인스턴스 크기를 정의합니다.

(2) Pod를 생성할 때 지정할 수 있는 모든 인스턴스 크기를 나열합니다. 이를 통해 더 적은 메모리와 더 적은 CPU 또는 대규모 워크로드의 인스턴스 크기가 필요한 워크로드에 대해 더 작은 인스턴스 크기를 정의할 수 있습니다.

(3) 검색한 **AZURE_SUBNET_ID** 값을 지정합니다.

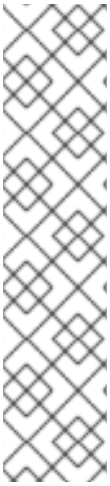
(4) 검색한 **AZURE_NSG_ID** 값을 지정합니다.

(5) 선택 사항: 기본적으로 클러스터 인증 정보를 기반으로 Azure 이미지 ID를 사용하여 **KataConfig** CR을 실행할 때 이 값이 채워집니다. 자체 Azure 이미지를 생성하는 경우 올바른 이미지 ID를 지정합니다.

(6) 검색한 **AZURE_REGION** 값을 지정합니다.

(7) 검색한 **AZURE_RESOURCE_GROUP** 값을 지정합니다.

6. **저장을** 클릭하여 변경 사항을 적용합니다.
   클라우드 공급자에 대한 구성 맵이 생성됩니다.



참고

피어 Pod 구성 맵을 업데이트하는 경우 변경 사항을 적용하려면 **peerpodconfig-ctrl-caa-daemon** 데몬 세트를 다시 시작해야 합니다.

구성 맵을 업데이트한 후 **저장을** 클릭하여 변경 사항을 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

daemonset을 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod를 업데이트하지 않습니다.

검증

- **워크로드 → ConfigMap** 으로 이동하여 새 구성 맵을 확인합니다.

3.2.2.3. SSH 키 보안 생성

Azure에 대한 SSH 키 **시크릿** 오브젝트를 생성해야 합니다.

프로세스

1. OpenShift Container Platform 클러스터에 로그인합니다.
2. 다음 명령을 실행하여 SSH 키 쌍을 생성합니다.

```
$ ssh-keygen -f ./id_rsa -N ""
```

3. OpenShift Container Platform 웹 콘솔에서 **워크로드 → 시크릿** 으로 이동합니다.
4. **시크릿** 페이지에서 **openshift-sandboxed-containers-operator** 프로젝트에 있는지 확인합니다.
5. **생성을** 클릭하고 **키/값 시크릿을** 선택합니다.
6. **시크릿 이름** 필드에 **ssh-key-secret** 을 입력합니다.
7. **키** 필드에 **id_rsa.pub** 를 입력합니다.
8. **값** 필드에 공개 SSH 키를 붙여넣습니다.
9. **생성**을 클릭합니다.
   SSH 키 시크릿이 생성됩니다.
10. 생성한 SSH 키를 삭제합니다.

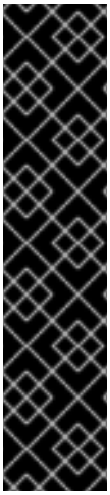
```
$ shred -remove id_rsa.pub id_rsa
```

3.2.2.4. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata-remote** 를 **RuntimeClass** 로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

kata-remote 런타임 클래스는 기본적으로 모든 작업자 노드에 설치됩니다. 특정 노드에만 **kata-remote** 를 설치하려면 해당 노드에 레이블을 추가한 다음 **KataConfig** CR에 레이블을 정의할 수 있습니다.

OpenShift 샌드박스 컨테이너는 **kata-remote** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.



중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 다음 요인은 재부팅 시간을 늘릴 수 있습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 선택합니다.
3. **KataConfig** 탭에서 **KataConfig 만들기** 를 클릭합니다.
4. 다음 세부 정보를 입력합니다.
   - **이름**: 선택 사항: 기본 이름은 **example-kataconfig** 입니다.
   - **labels**: 선택 사항: **KataConfig** 리소스에 대한 특성을 식별하는 모든 관련 정보를 입력합니다. 각 레이블은 키-값 쌍을 나타냅니다.
   - **enablePeerPods**: 퍼블릭 클라우드, IBM Z® 및 IBM® LinuxONE 배포에는 선택합니다.
   - **kataConfigPoolSelector**. 선택 사항: 선택한 노드에 **kata-remote** 를 설치하려면 선택한 노드의 라벨에 일치하는 표현식을 추가합니다.
     a. **kataConfigPoolSelector** 영역을 확장합니다.
     b. **kataConfigPoolSelector** 영역에서 **matchExpressions** 를 확장합니다. 이는 라벨 선택기 요구 사항 목록입니다.
     c. **matchExpressions 추가를** 클릭합니다.
     d. **키** 필드에 선택기가 적용되는 라벨 키를 입력합니다.
     e. **Operator** 필드에 레이블 값과의 키 관계를 입력합니다. 유효한 연산자는 **In**,**NotIn**,**Exists** 및 **DoesNotExist** 입니다.

f. **값** 영역을 확장한 다음 **값 추가** 를 클릭합니다.

g. **값** 필드에 **키** 레이블 값에 **true** 또는 **false** 를 입력합니다.

- loglevel: **kata-remote** 런타임 클래스를 사용하여 노드에 대해 검색된 로그 데이터의 수준을 정의합니다.

5. **생성**을 클릭합니다. **KataConfig** CR이 생성되고 작업자 노드에 **kata-remote** 런타임 클래스를 설치합니다.
설치를 확인하기 전에 **kata-remote** 설치가 완료되고 작업자 노드가 재부팅될 때까지 기다립니다.

검증

1. KataConfig 탭에서 **KataConfig** CR을 클릭하여 세부 정보를 확인합니다.

2. YAML 탭을 클릭하여 **상태 스탠자를 확인합니다**.
상태 스탠자에는 **조건** 및 **kataNodes** 키가 포함되어 있습니다. **status.kataNodes** 의 값은 노드 배열이며 각 노드는 특정 **kata-remote** 설치의 노드를 나열합니다. 업데이트가 있을 때마다 메시지가 표시됩니다.

3. Reload (다시 로드)를 클릭하여 YAML을 새로 고칩니다.
status.kataNodes 어레이의 모든 작업자가 **설치** 및 **조건.InProgress: False** 를 지정하는 이유 없이 False를 표시하면 클러스터에 **kata-remote** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

3.2.2.4.1. 선택 사항: Pod VM 이미지 확인

클러스터에 **kata-remote** 가 설치되면 OpenShift 샌드박스 컨테이너 Operator에서 피어 Pod를 생성하는 데 사용되는 Pod VM 이미지를 생성합니다. 이 프로세스는 클라우드 인스턴스에서 이미지가 생성되므로 시간이 오래 걸릴 수 있습니다. 클라우드 공급자에 대해 생성한 구성 맵을 확인하여 Pod VM 이미지가 성공적으로 생성되었는지 확인할 수 있습니다.

프로세스

1. **워크로드 → ConfigMap** 으로 이동합니다.

2. 공급자 구성 맵을 클릭하여 세부 정보를 확인합니다.

3. **YAML** 탭을 클릭합니다.

4. YAML 파일의 **상태** 스탠자를 확인합니다.
AZURE_IMAGE_ID 매개변수가 채워지면 Pod VM 이미지가 성공적으로 생성되었습니다.

문제 해결

1. 다음 명령을 실행하여 이벤트 로그를 검색합니다.

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. 다음 명령을 실행하여 작업 로그를 검색합니다.

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

문제를 해결할 수 없는 경우 Red Hat 지원 케이스를 제출하고 두 로그의 출력을 첨부합니다.

3.2.2.5. 선택 사항: 노드당 피어 Pod VM 수 수정

peerpodConfig CR(사용자 정의 리소스)을 편집하여 노드당 피어 Pod 가상 머신(VM) 제한을 변경할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 현재 제한을 확인합니다.

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator
\
-o jsonpath='{.spec.limit}{"\n"}'
```

2. 다음 명령을 실행하여 **peerpodConfig** CR의 **limit** 속성을 수정합니다.

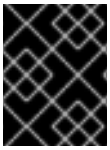
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

1 <value>를 정의할 제한으로 바꿉니다.

3.2.2.6. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata-remote** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트



중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

YAML 파일에 주석을 추가하여 구성 맵에 정의된 기본 인스턴스 크기를 사용하여 워크로드를 배포해야 하는지 여부를 정의할 수 있습니다.

인스턴스 크기를 수동으로 정의하지 않으려면 사용 가능한 메모리에 따라 자동 인스턴스 크기를 사용하도록 주석을 추가할 수 있습니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **워크로드 → 워크로드** 유형(예: **Pod**)으로 이동합니다.
2. 워크로드 유형 페이지에서 오브젝트를 클릭하여 세부 정보를 확인합니다.
3. **YAML** 탭을 클릭합니다.
4. 다음 예와 같이 **spec.runtimeClassName: kata-remote** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

5. 수동으로 정의된 인스턴스 크기 또는 자동 인스턴스 크기를 사용하도록 pod-templated 오브젝트에 주석을 추가합니다.
   - 수동으로 정의한 인스턴스 크기를 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: Standard_B2als_v2 (1)
# ...
```

(1) 구성 맵에서 정의한 인스턴스 크기를 지정합니다.

   - 자동 인스턴스 크기를 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

     워크로드에서 사용할 수 있는 메모리 양을 정의합니다. 워크로드는 사용 가능한 메모리 크기에 따라 자동 인스턴스 크기에서 실행됩니다.

6. **저장을** 클릭하여 변경 사항을 적용합니다.
   OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata-remote** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

3.2.3. 명령줄을 사용하여 워크로드 배포

명령줄을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

3.2.3.1. 시크릿 생성

OpenShift Container Platform 클러스터에 **Secret** 오브젝트를 생성해야 합니다. 시크릿은 Pod VM(가상 머신) 이미지 및 피어 Pod 인스턴스를 생성하기 위한 클라우드 공급자 자격 증명을 저장합니다. 기본적으로 OpenShift 샌드박스 컨테이너 Operator는 클러스터를 생성하는 데 사용되는 인증 정보를 기반으로 보안을 생성합니다. 그러나 다른 인증 정보를 사용하는 보안을 수동으로 생성할 수 있습니다.

사전 요구 사항

- Azure CLI 도구를 설치하고 구성했습니다.

프로세스

1. Azure 서브스크립션 ID를 검색합니다.

```
$ AZURE_SUBSCRIPTION_ID=$(az account list --query "[?isDefault].id" -o tsv) && echo
"AZURE_SUBSCRIPTION_ID: \"$AZURE_SUBSCRIPTION_ID\""
```

2. RBAC 콘텐츠를 생성합니다. 이렇게 하면 클라이언트 ID, 클라이언트 시크릿, 테넌트 ID가 생성됩니다.

```
$ az ad sp create-for-rbac --role Contributor --scopes
/subscriptions/$AZURE_SUBSCRIPTION_ID --query "{ client_id: appId, client_secret:
password, tenant_id: tenant }
```

출력 예:

```
{
  "client_id": `AZURE_CLIENT_ID`,
  "client_secret": `AZURE_CLIENT_SECRET`,
  "tenant_id": `AZURE_TENANT_ID`
}
```

3. **보안** 오브젝트에 사용할 RBAC 출력을 기록합니다.

4. 다음 예에 따라 **peer-pods-secret.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AZURE_CLIENT_ID: "<azure_client_id>" (1)
```

```
AZURE_CLIENT_SECRET: "<azure_client_secret>" 2
AZURE_TENANT_ID: "<azure_tenant_id>" 3
AZURE_SUBSCRIPTION_ID: "<azure_subscription_id>" 4
```

1. **AZURE_CLIENT_ID** 값을 지정합니다.
2. **AZURE_CLIENT_SECRET** 값을 지정합니다.
3. **AZURE_TENANT_ID** 값을 지정합니다.
4. **AZURE_SUBSCRIPTION_ID** 값을 지정합니다.

5. 매니페스트를 **적용하여 보안** 오브젝트를 생성합니다.

```
$ oc apply -f peer-pods-secret.yaml
```

참고

피어 Pod 시크릿을 업데이트하는 경우 **peerpodconfig-ctrl-caa-daemon** DaemonSet을 다시 시작하여 변경 사항을 적용해야 합니다.

시크릿을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-
operator REBOOT="$(date)"
```

데몬 세트를 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod는 업데이트하지 않습니다.

3.2.3.2. 구성 맵 생성

클라우드 공급자의 OpenShift Container Platform 클러스터에 구성 맵을 생성해야 합니다.

프로세스

1. Azure 인스턴스에서 다음 값을 가져옵니다.
   a. Azure VNet 이름을 검색하고 기록합니다.

```
$ AZURE_VNET_NAME=$(az network vnet list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Name:name}" --output tsv)
```

   이 값은 Azure 서브넷 ID를 검색하는 데 사용됩니다.

   b. Azure 서브넷 ID를 검색하고 기록합니다.

```
$ AZURE_SUBNET_ID=$(az network vnet subnet list --resource-group
${AZURE_RESOURCE_GROUP} --vnet-name $AZURE_VNET_NAME --query "[].{Id:id}
| [? contains(Id, 'worker')]" --output tsv) && echo "AZURE_SUBNET_ID:
\"$AZURE_SUBNET_ID\""
```

   c. Azure NSS(Network Security Group) ID를 검색하고 기록합니다.

```
$ AZURE_NSG_ID=$(az network nsg list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Id:id}" --output tsv) && echo
"AZURE_NSG_ID: \"$AZURE_NSG_ID\""
```

d. Azure 리소스 그룹을 검색하고 기록합니다.

```
$ AZURE_RESOURCE_GROUP=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.azure.resourceGroupName}') && echo
"AZURE_RESOURCE_GROUP: \"$AZURE_RESOURCE_GROUP\""
```

e. Azure 리전을 검색하고 기록합니다.

```
$ AZURE_REGION=$(az group show --resource-group
${AZURE_RESOURCE_GROUP} --query "{Location:location}" --output tsv) && echo
"AZURE_REGION: \"$AZURE_REGION\""
```

2. 다음 예에 따라 **peer-pods-cm.yaml** 매니페스트를 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_B2als_v2" (1)
  AZURE_INSTANCE_SIZES:
"Standard_B2als_v2,Standard_D2as_v5,Standard_D4as_v5,Standard_D2ads_v5" (2)
  AZURE_SUBNET_ID: "<azure_subnet_id>" (3)
  AZURE_NSG_ID: "<azure_nsg_id>" (4)
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  AZURE_IMAGE_ID: "<azure_image_id>" (5)
  AZURE_REGION: "<azure_region>" (6)
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" (7)
```

(1) 유형이 워크로드에 정의되지 않은 경우 사용되는 기본 인스턴스 크기를 정의합니다.

(2) Pod를 생성할 때 지정할 수 있는 모든 인스턴스 크기를 나열합니다. 이를 통해 더 적은 메모리와 더 적은 CPU 또는 대규모 워크로드의 인스턴스 크기가 필요한 워크로드에 대해 더 작은 인스턴스 크기를 정의할 수 있습니다.

(3) 검색한 **AZURE_SUBNET_ID** 값을 지정합니다.

(4) 검색한 **AZURE_NSG_ID** 값을 지정합니다.

(5) 선택 사항: 기본적으로 클러스터 인증 정보를 기반으로 Azure 이미지 ID를 사용하여 **KataConfig** CR을 실행할 때 이 값이 채워집니다. 자체 Azure 이미지를 생성하는 경우 올바른 이미지 ID를 지정합니다.

(6) 검색한 **AZURE_REGION** 값을 지정합니다.

(7) 검색한 **AZURE_RESOURCE_GROUP** 값을 지정합니다.

3. 매니페스트를 적용하여 구성 맵을 생성합니다.

```
$ oc apply -f peer-pods-cm.yaml
```

클라우드 공급자에 대한 구성 맵이 생성됩니다.

참고

피어 Pod 구성 맵을 업데이트하는 경우 변경 사항을 적용하려면 **peerpodconfig-ctrl-caa-daemon** 데몬 세트를 다시 시작해야 합니다.

구성 맵을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

daemonset을 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod를 업데이트하지 않습니다.

3.2.3.3. SSH 키 보안 생성

Azure에 대한 SSH 키 **시크릿** 오브젝트를 생성해야 합니다.

프로세스

1. OpenShift Container Platform 클러스터에 로그인합니다.
2. 다음 명령을 실행하여 SSH 키 쌍을 생성합니다.

```
$ ssh-keygen -f ./id_rsa -N ""
```

3. 다음 명령을 실행하여 **Secret** 오브젝트를 생성합니다.

```
$ oc create secret generic ssh-key-secret \
    -n openshift-sandboxed-containers-operator \
    --from-file=id_rsa.pub=./id_rsa.pub \
    --from-file=id_rsa=./id_rsa
```

SSH 키 시크릿이 생성됩니다.

4. 생성한 SSH 키를 삭제합니다.

```
$ shred -remove id_rsa.pub id_rsa
```

3.2.3.4. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata-remote** 를 런타임 클래스로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

KataConfig CR을 생성하면 OpenShift 샌드박스 컨테이너 Operator가 다음을 수행합니다.

- 기본 구성을 사용하여 **kata-remote** 라는 **RuntimeClass** CR을 생성합니다. 이를 통해 사용자는 **RuntimeClassName** 필드에서 CR을 참조하여 **kata-remote** 를 런타임으로 사용하도록 워크로드를 구성할 수 있습니다. 이 CR은 런타임의 리소스 오버헤드도 지정합니다.

OpenShift 샌드박스 컨테이너는 **kata-remote** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.

중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

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

프로세스

1. 다음 예에 따라 **cluster-kataconfig.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
```

2. 선택 사항: 선택한 노드에 **kata-remote** 를 설치하려면 다음 예에 따라 노드 라벨을 지정합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 선택한 노드의 라벨을 지정합니다.

3. **KataConfig** CR을 생성합니다.

```
$ oc create -f cluster-kataconfig.yaml
```

새로운 **KataConfig** CR이 생성되고 작업자 노드에 **kata-remote** 가 런타임 클래스로 설치됩니다.

설치를 확인하기 전에 **kata-remote** 설치가 완료되고 작업자 노드가 재부팅될 때까지 기다립니다.

검증

- 다음 명령을 실행하여 설치 진행 상황을 모니터링합니다.

```
$ watch "oc describe kataconfig | sed -n /^Status:/,/^Events/p"
```

kataNodes 아래의 모든 작업자의 상태가 **설치되고** 이유를 지정하지 않고 **InProgress** 조건이 **False** 이면 클러스터에 **kata-remote** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

3.2.3.4.1. 선택 사항: Pod VM 이미지 확인

클러스터에 **kata-remote** 가 설치되면 OpenShift 샌드박스 컨테이너 Operator에서 피어 Pod를 생성하는 데 사용되는 Pod VM 이미지를 생성합니다. 이 프로세스는 클라우드 인스턴스에서 이미지가 생성되므로 시간이 오래 걸릴 수 있습니다. 클라우드 공급자에 대해 생성한 구성 맵을 확인하여 Pod VM 이미지가 성공적으로 생성되었는지 확인할 수 있습니다.

프로세스

1. 피어 Pod에 대해 생성한 구성 맵을 가져옵니다.

```
$ oc get configmap peer-pods-cm -n openshift-sandboxed-containers-operator -o yaml
```

2. YAML 파일의 **상태** 스탠자를 확인합니다.
AZURE_IMAGE_ID 매개변수가 채워지면 Pod VM 이미지가 성공적으로 생성되었습니다.

문제 해결

1. 다음 명령을 실행하여 이벤트 로그를 검색합니다.

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. 다음 명령을 실행하여 작업 로그를 검색합니다.

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

문제를 해결할 수 없는 경우 Red Hat 지원 케이스를 제출하고 두 로그의 출력을 첨부합니다.

3.2.3.5. 선택 사항: 노드당 피어 Pod VM 수 수정

peerpodConfig CR(사용자 정의 리소스)을 편집하여 노드당 피어 Pod 가상 머신(VM) 제한을 변경할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 현재 제한을 확인합니다.

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator
\
-o jsonpath='{.spec.limit}{"\n"}'
```

2. 다음 명령을 실행하여 **peerpodConfig** CR의 **limit** 속성을 수정합니다.

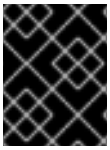
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

1 <value>를 정의할 제한으로 바꿉니다.

3.2.3.6. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata-remote** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트



중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

YAML 파일에 주석을 추가하여 구성 맵에 정의된 기본 인스턴스 크기를 사용하여 워크로드를 배포해야 하는지 여부를 정의할 수 있습니다.

인스턴스 크기를 수동으로 정의하지 않으려면 사용 가능한 메모리에 따라 자동 인스턴스 크기를 사용하도록 주석을 추가할 수 있습니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. 다음 예와 같이 **spec.runtimeClassName: kata-remote** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

2. 수동으로 정의된 인스턴스 크기 또는 자동 인스턴스 크기를 사용하도록 pod-templated 오브젝트에 주석을 추가합니다.

   - 수동으로 정의한 인스턴스 크기를 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: Standard_B2als_v2 (1)
# ...
```

   (1) 구성 맵에서 정의한 인스턴스 크기를 지정합니다.

   - 자동 인스턴스 크기를 사용하려면 다음 주석을 추가합니다.

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

     워크로드에서 사용할 수 있는 메모리 양을 정의합니다. 워크로드는 사용 가능한 메모리 크기에 따라 자동 인스턴스 크기에서 실행됩니다.

3. 다음 명령을 실행하여 워크로드 오브젝트에 변경 사항을 적용합니다.

```
$ oc apply -f <object.yaml>
```

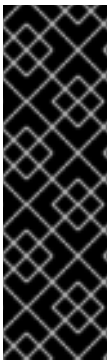
   OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata-remote** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

4장. IBM에 워크로드 배포

IBM Z® 및 IBM® LinuxONE에 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.



중요

IBM Z® 및 IBM® LinuxONE에 OpenShift 샌드박스 컨테이너 워크로드를 배포하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

클러스터 사전 요구 사항

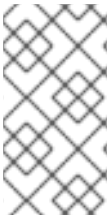
- Red Hat OpenShift Container Platform 4.14 이상을 설치했습니다.
- 클러스터에는 컨트롤 노드와 두 개의 작업자 노드가 있습니다.

배포 흐름

이 문서는 IBM Z®에 대해서만 설명하지만 모든 절차는 IBM® LinuxONE에도 적용됩니다.

다음 단계를 수행하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

1. KVM 호스트에 libvirt 볼륨을 구성합니다.
2. KVM 게스트 이미지를 생성하여 libvirt 볼륨에 업로드합니다.
3. 피어 포드 VM 이미지를 생성하여 libvirt 볼륨에 업로드합니다.
4. libvirt 공급자에 대한 시크릿을 생성합니다.
5. libvirt 공급자에 대한 구성 맵을 만듭니다.
6. KVM 호스트에 대한 SSH 키 시크릿을 생성합니다.
7. **KataConfig** CR을 생성합니다.
8. 선택 사항: 노드당 피어 Pod VM 제한을 수정합니다.
9. **kata-remote** 런타임 클래스를 사용하도록 워크로드 오브젝트를 구성합니다.



참고

- 클러스터 노드와 피어 Pod는 동일한 IBM Z® KVM 호스트 논리 파티션(LPAR)에 있어야 합니다.
- 클러스터 노드와 피어 Pod는 동일한 서브넷에 연결되어 있어야 합니다.

4.1. 환경 준비

환경을 준비하려면 다음 단계를 수행합니다.

1. 클러스터에 충분한 리소스가 있는지 확인합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 설치합니다.

4.1.1. 리소스 요구 사항

피어 Pod 가상 머신(VM)에는 다음 두 위치에 있는 리소스가 필요합니다.

- 작업자 노드입니다. 작업자 노드는 메타데이터, Kata shim 리소스(**containerd-shim-kata-v2**), remote-hypervisor 리소스(**cloud-api-adaptor**) 및 작업자 노드와 피어 Pod VM 간의 터널 설정을 저장합니다.
- 클라우드 인스턴스입니다. 클라우드에서 실행되는 실제 피어 Pod VM입니다.

Kubernetes 작업자 노드에 사용되는 CPU 및 메모리 리소스는 피어 Pod를 생성하는 데 사용되는 RuntimeClass(**kata-remote**) 정의에 포함된 Pod 오버헤드 에 의해 처리됩니다.

클라우드에서 실행되는 총 피어 Pod VM 수는 Kubernetes 노드 확장 리소스로 정의됩니다. 이 제한은 노드당이며 **peerpodConfig** CR(사용자 정의 리소스)의 **limit** 속성으로 설정됩니다.

peerpodconfig-openshift 라는 **peerpodConfig** CR은 **kataConfig** CR을 생성하고 피어 Pod를 활성화할 때 생성되며 **openshift-sandboxed-containers-operator** 네임스페이스에 있습니다.

다음 **peerpodConfig** CR 예제에서는 기본 **사양** 값을 표시합니다.

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" 1
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

1 기본 제한은 노드당 VM 10개입니다.

확장된 리소스의 이름은 **kata.peerpods.io/vm** 이며 Kubernetes 스케줄러에서 용량 추적 및 계정을 처리할 수 있습니다.

환경의 요구 사항에 따라 노드당 제한을 편집할 수 있습니다. 자세한 내용은 "Peer pods에서 노드당 VM 제한 수정"을 참조하십시오.

변경 웹 후크 는 확장된 리소스 **kata.peerpods.io/vm** 을 Pod 사양에 추가합니다. 또한 Pod 사양에서 리소스별 항목도 제거합니다(있는 경우). 이를 통해 Kubernetes 스케줄러에서 이러한 확장 리소스를 고려하여 리소스를 사용할 수 있는 경우에만 피어 Pod를 예약할 수 있습니다.

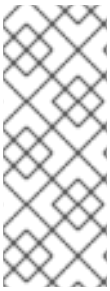
변경 웹 후크는 다음과 같이 Kubernetes Pod를 수정합니다.

- 변경 웹 후크는 **TARGET_RUNTIME_CLASS** 환경 변수에 지정된 예상 **RuntimeClassName** 값을 Pod에 확인합니다. Pod 사양의 값이 **TARGET_RUNTIME_CLASS** 의 값과 일치하지 않으면 Pod를 수정하지 않고 웹 후크가 종료됩니다.

- **RuntimeClassName** 값이 일치하는 경우 Webhook에서 Pod 사양을 다음과 같이 변경합니다.

  1. Webhook는 Pod에 있는 모든 컨테이너 및 init 컨테이너의 **resources** 필드에서 모든 리소스 사양을 제거합니다.

  2. Webhook는 Pod의 첫 번째 컨테이너의 resources 필드를 수정하여 확장 리소스 (**kata.peerpods.io/vm**)를 사양에 추가합니다. 확장된 리소스 **kata.peerpods.io/vm** 은 회계 목적으로 Kubernetes 스케줄러에서 사용합니다.



참고

변경 웹 후크는 OpenShift Container Platform의 특정 시스템 네임스페이스가 변경되지 않습니다. 해당 시스템 네임스페이스에 피어 Pod가 생성되면 Pod 사양에 확장 리소스가 포함되지 않는 한 Kubernetes 확장 리소스를 사용하는 리소스 계정이 작동하지 않습니다.

특정 네임스페이스에서 피어 Pod 생성만 허용하도록 클러스터 전체 정책을 정의하는 것이 좋습니다.

4.1.2. OpenShift 샌드박스 컨테이너 Operator 설치

OpenShift Container Platform 웹 콘솔 또는 CLI(명령줄 인터페이스)를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

4.1.2.1. 웹 콘솔을 사용하여 Operator 설치

Red Hat OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**로 이동합니다.
2. **키워드로 필터링** 필드에 **OpenShift sandboxed containers**를 입력합니다.
3. **OpenShift 샌드박스 컨테이너 Operator** 타일을 선택하고 **설치를** 클릭합니다.
4. **Operator 설치** 페이지의 사용 가능한 **업데이트 채널** 옵션 목록에서 **stable** 을 선택합니다.
5. 설치된 네임스페이스 용으로 **Operator 권장 네임스페이스** 가 선택되어 **있는지 확인합니다**. 이렇게 하면 필수 **openshift-sandboxed-containers-operator** 네임스페이스에 Operator가 설치됩니다. 이 네임스페이스가 아직 존재하지 않으면 자동으로 생성됩니다.

참고

openshift-sandboxed-containers-operator 이외의 네임스페이스에 OpenShift 샌드박스 컨테이너 Operator를 설치하려고 하면 설치가 실패합니다.

6. **승인 전략에 대해 자동 이 선택되어 있는지 확인합니다. Automatic** 은 기본값이며 새 z-stream 릴리스를 사용할 수 있을 때 OpenShift 샌드박스 컨테이너에 대한 자동 업데이트를 활성화합니다.
7. **설치**를 클릭합니다.

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

1. **Operators → 설치된 Operator**로 이동합니다.
2. OpenShift 샌드박스 컨테이너 Operator가 표시되는지 확인합니다.

추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경에 대한 Operator Lifecycle Manager에서 프록시 지원 구성 .

4.1.2.2. CLI를 사용하여 Operator 설치

CLI를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. **Namespace.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. 다음 명령을 실행하여 네임스페이스를 생성합니다.

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

3. **OperatorGroup.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

4. 다음 명령을 실행하여 operator 그룹을 생성합니다.

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

5. **Subscription.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0
```

6. 다음 명령을 실행하여 서브스크립션을 생성합니다.

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

OpenShift 샌드박스 컨테이너 Operator가 클러스터에 설치되었습니다.

검증

- 다음 명령을 실행하여 Operator가 올바르게 설치되었는지 확인합니다.

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

출력 예

```
NAME                                DISPLAY                                VERSION   REPLACES   PHASE
openshift-sandboxed-containers.v1.6.0   openshift-sandboxed-containers-operator   1.6.0     1.5.3      Succeeded
```

4.1.2.3. 추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- 연결이 끊긴 환경을 위한 Operator Lifecycle Manager에서 프록시 지원 구성

4.2. 명령줄을 사용하여 워크로드 배포

명령줄을 사용하여 OpenShift 샌드박스 컨테이너 워크로드를 배포할 수 있습니다.

4.2.1. libvirt 볼륨 구성

KVM 호스트에 libvirt 볼륨을 구성해야 합니다. 피어 Pod는 Cloud API Adaptor의 libvirt 공급자를 사용하여 가상 머신을 생성하고 관리합니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 OpenShift Container Platform 클러스터에 OpenShift 샌드박스 컨테이너 Operator를 설치했습니다.
- KVM 호스트에 대한 관리자 권한이 있습니다.

- KVM 호스트에 **podman** 을 설치했습니다.
- KVM 호스트에 **virt-customize** 를 설치했습니다.

프로세스

1. KVM 호스트에 로그인합니다.
2. 다음 명령을 실행하여 libvirt 풀의 이름을 설정합니다.

```
$ export LIBVIRT_POOL=<libvirt_pool>
```

libvirt 공급자의 시크릿을 생성하려면 **LIBVIRT_POOL** 값이 필요합니다.

3. 다음 명령을 실행하여 libvirt 풀의 이름을 설정합니다.

```
$ export LIBVIRT_VOL_NAME=<libvirt_volume>
```

libvirt 공급자의 시크릿을 생성하려면 **LIBVIRT_VOL_NAME** 값이 필요합니다.

4. 다음 명령을 실행하여 기본 스토리지 풀 위치의 경로를 설정합니다.

```
$ export LIBVIRT_POOL_DIRECTORY=<target_directory> 1
```

1 libvirt에 읽기 및 쓰기 액세스 권한이 있는지 확인하려면 libvirt 스토리지 디렉터리의 하위 디렉터리를 사용합니다. 기본값은 **/var/lib/libvirt/images/** 입니다.

5. 다음 명령을 실행하여 libvirt 풀을 생성합니다.

```
$ virsh pool-define-as $LIBVIRT_POOL --type dir --target "$LIBVIRT_POOL_DIRECTORY"
```

6. 다음 명령을 실행하여 libvirt 풀을 시작합니다.

```
$ virsh pool-start $LIBVIRT_POOL
```

7. 다음 명령을 실행하여 풀에 대한 libvirt 볼륨을 만듭니다.

```
$ virsh -c qemu:///system \
  vol-create-as --pool $LIBVIRT_POOL \
  --name $LIBVIRT_VOL_NAME \
  --capacity 20G \
  --allocation 2G \
  --prealloc-metadata \
  --format qcow2
```

4.2.2. KVM 게스트 이미지 생성

KVM 게스트 이미지를 생성하여 libvirt 볼륨에 업로드해야 합니다.

사전 요구 사항

- IBM z15 이상 또는 IBM® LinuxONE III 이상

- KVM을 사용하여 RHEL 9 이상에서 실행 중인 하나 이상의 LPAR.

프로세스

1. OpenShift Container Platform 클러스터에 로그인합니다.
2. RHEL 서브스크립션이 있는 경우 Red Hat 서브스크립션 관리에 대한 서브스크립션 환경 변수를 설정합니다.
   - 다음 명령을 실행하여 조직 ID를 설정합니다.

     ```
     $ export ORG_ID=$(cat ~/.rh_subscription/orgid)
     ```

   - 다음 명령을 실행하여 활성화 키를 설정합니다.

     ```
     $ export ACTIVATION_KEY=$(cat ~/.rh_subscription/activation_key)
     ```

3. RHEL 서브스크립션이 없는 경우 RHEL의 서브스크립션 값을 설정합니다.
   - 다음 명령을 실행하여 조직 ID를 설정합니다.

     ```
     $ export ORG_ID=<RHEL_ORGID_VALUE> 1
     ```

     1 RHEL 조직 ID를 지정합니다.

   - 다음 명령을 실행하여 활성화 키를 설정합니다.

     ```
     $ export ACTIVATION_KEY=<RHEL_ACTIVATION_KEY> 1
     ```

     1 RHEL 활성화 키를 지정합니다.

4. IBM Z® 시스템에 로그인합니다.
5. Red Hat Customer Portal 에서 libvirt 올바른 액세스 권한을 부여하려면 **s390x** RHEL KVM 게스트 이미지를 libvirt 스토리지 디렉터리로 다운로드합니다.
   기본 디렉터리는 **/var/lib/libvirt/images** 입니다. 이 이미지는 관련 바이너리를 포함하는 피어 Pod VM 이미지를 생성하는 데 사용됩니다.
6. 다음 명령을 실행하여 다운로드한 이미지의 **IMAGE_URL** 을 설정합니다.

   ```
   $ export IMAGE_URL=<path/to/image> 1
   ```

   1 KVM 게스트 이미지의 경로를 지정합니다.

7. 다음 명령을 실행하여 게스트 KVM 이미지를 등록합니다.

   ```
   $ export REGISTER_CMD="subscription-manager register --org=${ORG_ID} \
     --activationkey=${ACTIVATION_KEY}"
   ```

8. 다음 명령을 실행하여 게스트 KVM 이미지를 사용자 지정합니다.

```
$ virt-customize -v -x -a ${IMAGE_URL} --run-command "${REGISTER_CMD}"
```

9. 다음 명령을 실행하여 이미지의 체크섬을 설정합니다.

```
$ export IMAGE_CHECKSUM=$(sha256sum ${IMAGE_URL} | awk '{ print $1 }')
```

4.2.3. 피어 Pod VM 이미지 빌드

피어 Pod 가상 머신(VM) 이미지를 빌드하고 libvirt 볼륨에 업로드해야 합니다.

프로세스

1. OpenShift Container Platform 클러스터에 로그인합니다.
2. 다음 명령을 실행하여 cloud-api-adaptor 리포지토리를 복제합니다.

```
$ git clone --single-branch https://github.com/confidential-containers/cloud-api-adaptor.git
```

3. 다음 명령을 실행하여 **podvm** 디렉터리로 변경합니다.

```
$ cd cloud-api-adaptor && git checkout 8577093
```

4. 최종 QCOW2 이미지가 생성되는 빌더 이미지를 생성합니다.
   - RHEL 시스템이 서브스크립션된 경우 다음 명령을 실행합니다.

```
$ podman build -t podvm_builder_rhel_s390x \
  --build-arg ARCH="s390x" \
  --build-arg GO_VERSION="1.21.3" \
  --build-arg PROTOC_VERSION="25.1" \
  --build-arg PACKER_VERSION="v1.9.4" \
  --build-arg RUST_VERSION="1.72.0" \
  --build-arg YQ_VERSION="v4.35.1" \
  --build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1fc9
25aac08268a45" \
  -f podvm/Dockerfile.podvm_builder.rhel .
```

   - RHEL 시스템이 서브스크립션이 없는 경우 다음 명령을 실행합니다.

```
$ podman build -t podvm_builder_rhel_s390x \
  --build-arg ORG_ID=$ORG_ID \
  --build-arg ACTIVATION_KEY=$ACTIVATION_KEY \
  --build-arg ARCH="s390x" \
  --build-arg GO_VERSION="1.21.3" \
  --build-arg PROTOC_VERSION="25.1" \
  --build-arg PACKER_VERSION="v1.9.4" \
  --build-arg RUST_VERSION="1.72.0" \
  --build-arg YQ_VERSION="v4.35.1" \
  --build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1fc9
25aac08268a45" \
  -f podvm/Dockerfile.podvm_builder.rhel .
```

5. 다음 명령을 실행하여 피어 Pod를 실행하는 데 필요한 바이너리를 사용하여 중간 이미지 패키지를 생성합니다.

```
$ podman build -t podvm_binaries_rhel_s390x \
  --build-arg BUILDER_IMG="podvm_builder_rhel_s390x:latest" \
  --build-arg ARCH=s390x \
  -f podvm/Dockerfile.podvm_binaries.rhel .
```

이 프로세스는 상당한 시간이 걸립니다.

6. 다음 명령을 실행하여 바이너리를 추출하고 피어 Pod QCOW2 이미지를 빌드합니다.

```
$ podman build -t podvm_rhel_s390x \
  --build-arg ARCH=s390x \
  --build-arg CLOUD_PROVIDER=libvirt \
  --build-arg BUILDER_IMG="localhost/podvm_builder_rhel_s390x:latest" \
  --build-arg BINARIES_IMG="localhost/podvm_binaries_rhel_s390x:latest" \
  -v ${IMAGE_URL}:/tmp/rhel.qcow2:Z \
  --build-arg IMAGE_URL="/tmp/rhel.qcow2" \
  --build-arg IMAGE_CHECKSUM=${IMAGE_CHECKSUM} \
  -f podvm/Dockerfile.podvm.rhel .
```

7. 다음 명령을 실행하여 이미지 디렉터리 환경 변수를 생성합니다.

```
$ export IMAGE_OUTPUT_DIR=<image_output_directory> 1
```

1 이미지의 디렉터리를 지정합니다.

8. 다음 명령을 실행하여 이미지 디렉터리를 생성합니다.

```
$ mkdir -p $IMAGE_OUTPUT_DIR
```

9. 다음 명령을 실행하여 추출된 피어 Pod QCOW2 이미지를 저장합니다.

```
$ podman save podvm_rhel_s390x | tar -xO --no-wildcards-match-slash '*.tar' | tar -x -C
${IMAGE_OUTPUT_DIR}
```

10. 피어 Pod QCOW2 이미지를 libvirt 볼륨에 업로드합니다.

```
$ virsh -c qemu:///system vol-upload \
  --vol $LIBVIRT_VOL_NAME \
  $IMAGE_OUTPUT_DIR/podvm-*.qcow2 \
  --pool $LIBVIRT_POOL --sparse
```

4.2.4. 시크릿 생성

OpenShift Container Platform 클러스터에 **Secret** 오브젝트를 생성해야 합니다.

사전 요구 사항

- **LIBVIRT_POOL**. KVM 호스트에서 libvirt를 구성할 때 설정한 값을 사용합니다.

- **LIBVIRT_VOL_NAME**. KVM 호스트에서 libvirt를 구성할 때 설정한 값을 사용합니다.
- **LIBVIRT_URI**. 이 값은 libvirt 네트워크의 기본 게이트웨이 IP 주소입니다. libvirt 네트워크 설정을 확인하여 이 값을 가져옵니다.

참고

libvirt에서 기본 브리지 가상 네트워크를 사용하는 경우 다음 명령을 실행하여 **LIBVIRT_URI** 를 가져올 수 있습니다.

```
$ virtint=$(bridge_line=$(virsh net-info default | grep Bridge);  echo "${bridge_line//Bridge:/}" | tr -d [:blank:])

$ LIBVIRT_URI=$( ip -4 addr show $virtint | grep -oP '(?<=inet\s)\d+(\.\d+){3}')
```

프로세스

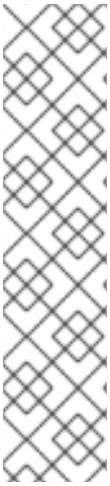
1. 다음 예에 따라 **peer-pods-secret.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  CLOUD_PROVIDER: "libvirt"
  LIBVIRT_URI: "<libvirt_gateway_uri>" 1
  LIBVIRT_POOL: "<libvirt_pool>" 2
  LIBVIRT_VOL_NAME: "<libvirt_volume>" 3
```

1. libvirt URI를 지정합니다.
2. libvirt 풀을 지정합니다.
3. libvirt 볼륨 이름을 지정합니다.

2. 매니페스트를 **적용하여 보안** 오브젝트를 생성합니다.

```
$ oc apply -f peer-pods-secret.yaml
```



참고

피어 Pod 시크릿을 업데이트하는 경우 **peerpodconfig-ctrl-caa-daemon** DaemonSet을 다시 시작하여 변경 사항을 적용해야 합니다.

시크릿을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

데몬 세트를 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod는 업데이트하지 않습니다.

4.2.5. 구성 맵 생성

libvirt 공급자의 OpenShift Container Platform 클러스터에 구성 맵을 생성해야 합니다.

프로세스

1. 다음 예에 따라 **peer-pods-cm.yaml** 매니페스트를 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "libvirt"
  PROXY_TIMEOUT: "15m"
```

2. 매니페스트를 적용하여 구성 맵을 생성합니다.

```
$ oc apply -f peer-pods-cm.yaml
```

libvirt 공급자에 대한 구성 맵이 생성됩니다.

참고

피어 Pod 구성 맵을 업데이트하는 경우 변경 사항을 적용하려면 **peerpodconfig-ctrl-caa-daemon** 데몬 세트를 다시 시작해야 합니다.

구성 맵을 업데이트한 후 매니페스트를 적용합니다. 그런 다음 다음 명령을 실행하여 **cloud-api-adaptor** Pod를 다시 시작합니다.

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

daemonset을 다시 시작하면 피어 Pod가 다시 생성됩니다. 기존 Pod를 업데이트하지 않습니다.

4.2.6. SSH 키 보안 생성

KVM 호스트에 대한 SSH 키 **시크릿** 오브젝트를 생성해야 합니다.

프로세스

1. OpenShift Container Platform 클러스터에 로그인합니다.
2. 다음 명령을 실행하여 SSH 키 쌍을 생성합니다.

```
$ ssh-keygen -f ./id_rsa -N ""
```

3. 공개 SSH 키를 KVM 호스트에 복사합니다.

```
$ ssh-copy-id -i ./id_rsa.pub <KVM_HOST_IP>
```

4. 다음 명령을 실행하여 **Secret** 오브젝트를 생성합니다.

```
$ oc create secret generic ssh-key-secret \
    -n openshift-sandboxed-containers-operator \
    --from-file=id_rsa.pub=./id_rsa.pub \
    --from-file=id_rsa=./id_rsa
```

SSH 키 시크릿이 생성됩니다.

5. 생성한 SSH 키를 삭제합니다.

```
$ shred -remove id_rsa.pub id_rsa
```

4.2.7. KataConfig 사용자 지정 리소스 생성

작업자 노드에 **kata-remote** 를 런타임 클래스로 설치하려면 **KataConfig** CR(사용자 정의 리소스)을 생성해야 합니다.

KataConfig CR을 생성하면 OpenShift 샌드박스 컨테이너 Operator가 다음을 수행합니다.

- 기본 구성을 사용하여 **kata-remote** 라는 **RuntimeClass** CR을 생성합니다. 이를 통해 사용자는 **RuntimeClassName** 필드에서 CR을 참조하여 **kata-remote** 를 런타임으로 사용하도록 워크로드를 구성할 수 있습니다. 이 CR은 런타임의 리소스 오버헤드도 지정합니다.

OpenShift 샌드박스 컨테이너는 **kata-remote** 를 기본 런타임이 아닌 클러스터의 *선택적* 런타임으로 설치합니다.

중요

KataConfig CR을 생성하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 디스크 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

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

프로세스

1. 다음 예에 따라 **cluster-kataconfig.yaml** 매니페스트 파일을 생성합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
```

2. 선택 사항: 선택한 노드에 **kata-remote** 를 설치하려면 다음 예에 따라 노드 라벨을 지정합니다.

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 선택한 노드의 라벨을 지정합니다.

3. **KataConfig** CR을 생성합니다.

```
$ oc create -f cluster-kataconfig.yaml
```

새로운 **KataConfig** CR이 생성되고 작업자 노드에 **kata-remote** 가 런타임 클래스로 설치됩니다.

설치를 확인하기 전에 **kata-remote** 설치가 완료되고 작업자 노드가 재부팅될 때까지 기다립니다.

검증

- 다음 명령을 실행하여 설치 진행 상황을 모니터링합니다.

```
$ watch "oc describe kataconfig | sed -n /^Status:/,/^Events/p"
```

kataNodes 아래의 모든 작업자의 상태가 **설치되고** 이유를 지정하지 않고 **InProgress** 조건이 **False** 이면 클러스터에 **kata-remote** 가 설치됩니다.

자세한 내용은 KataConfig 상태 메시지를 참조하십시오.

4.2.8. 선택 사항: 노드당 피어 Pod VM 수 수정

peerpodConfig CR(사용자 정의 리소스)을 편집하여 노드당 피어 Pod 가상 머신(VM) 제한을 변경할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 현재 제한을 확인합니다.

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator
\
-o jsonpath='{.spec.limit}{"\n"}'
```

2. 다음 명령을 실행하여 **peerpodConfig** CR의 **limit** 속성을 수정합니다.

```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

1 <value>를 정의할 제한으로 바꿉니다.

4.2.9. 워크로드 오브젝트 구성

다음 pod 템플릿 오브젝트의 런타임 클래스로 **kata-remote** 를 구성하여 OpenShift 샌드박스 컨테이너 워크로드를 배포합니다.

- **Pod** 오브젝트
- **ReplicaSet** 오브젝트
- **ReplicationController** 오브젝트
- **StatefulSet** 오브젝트
- **Deployment** 오브젝트
- **DeploymentConfig** 오브젝트

중요

openshift-sandboxed-containers-operator 네임스페이스에 워크로드를 배포하지 마십시오. 이러한 리소스에 대한 전용 네임스페이스를 생성합니다.

사전 요구 사항

- 공급자에 대한 보안 오브젝트를 생성했습니다.
- 공급자에 대한 구성 맵을 생성했습니다.
- **KataConfig** CR(사용자 정의 리소스)을 생성했습니다.

프로세스

1. 다음 예와 같이 **spec.runtimeClassName: kata-remote** 를 각 pod 템플릿 워크로드 오브젝트의 매니페스트에 추가합니다.

```
apiVersion: v1
kind: <object>
# ...
```

```
spec:
  runtimeClassName: kata-remote
# ...
```

OpenShift Container Platform은 워크로드 오브젝트를 생성하고 스케줄링을 시작합니다.

검증

- pod-templated 오브젝트의 **spec.runtimeClassName** 필드를 검사합니다. 값이 **kata-remote** 이면 피어 Pod를 사용하여 OpenShift 샌드박스 컨테이너에서 워크로드가 실행됩니다.

5장. 모니터링

OpenShift Container Platform 웹 콘솔을 사용하여 샌드박스된 워크로드 및 노드의 상태와 관련된 메트릭을 모니터링할 수 있습니다.

OpenShift 샌드박스 컨테이너에는 OpenShift Container Platform 웹 콘솔에서 사전 구성된 대시보드가 있습니다. 관리자는 Prometheus를 통해 원시 지표에 액세스하고 쿼리할 수도 있습니다.

5.1. 메트릭 정보

OpenShift 샌드박스 컨테이너 메트릭을 사용하면 관리자가 샌드박스 컨테이너를 실행하는 방법을 모니터링할 수 있습니다. OpenShift Container Platform 웹 콘솔에서 Metrics UI에서 이러한 메트릭을 쿼리할 수 있습니다.

OpenShift 샌드박스 컨테이너 지표는 다음 카테고리에 대해 수집됩니다.

Kata 에이전트 메트릭

Kata 에이전트 메트릭은 샌드박스 컨테이너에 포함된 VM에서 실행되는 kata 에이전트 프로세스에 대한 정보를 표시합니다. 이러한 메트릭에는 **/proc/<pid>/[io, stat, status]** 의 데이터가 포함됩니다.

Kata 게스트 운영 체제 메트릭

Kata 게스트 운영 체제 메트릭은 샌드박스 컨테이너에서 실행되는 게스트 운영 체제의 데이터를 표시합니다. 이러한 메트릭에는 **/proc/[stats, diskstats, meminfo, vmstats]** 및 **/proc/net/dev** 의 데이터가 포함됩니다.

하이퍼바이저 지표

하이퍼바이저 지표는 샌드박스 컨테이너에 포함된 VM을 실행하는 하이퍼바이저와 관련된 데이터를 표시합니다. 이러한 메트릭에는 주로 **/proc/<pid>/[io, stat, status]** 의 데이터가 포함됩니다.

Kata 모니터 메트릭

Kata 모니터는 지표 데이터를 수집하여 Prometheus에서 사용할 수 있도록 하는 프로세스입니다. kata 모니터 메트릭에는 kata-monitor 프로세스 자체의 리소스 사용량에 대한 자세한 정보가 표시됩니다. 이러한 메트릭에는 Prometheus 데이터 수집의 카운터도 포함됩니다.

Kata containerd shim v2 메트릭

Kata containerd shim v2 메트릭에는 kata shim 프로세스에 대한 자세한 정보가 표시됩니다. 이러한 메트릭에는 **/proc/<pid>/[io, stat, status]** 및 세부 리소스 사용량 메트릭의 데이터가 포함됩니다.

5.2. 메트릭 보기

OpenShift Container Platform 웹 콘솔의 **Metrics** 페이지에서 OpenShift 샌드박스 컨테이너의 메트릭에 액세스할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할 또는 모든 프로젝트에 대한 보기 권한이 있는 사용자로 클러스터에 액세스할 수 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **모니터링 → 메트릭** 으로 이동합니다.
2. 입력 필드에 모니터링할 지표 쿼리를 입력합니다.
   모든 카타 관련 메트릭은 **카타** 로 시작합니다. **kata** 를 입력하면 사용 가능한 모든 카타 메트릭 목록이 표시됩니다.

쿼리의 메트릭은 페이지에 시각화됩니다.

추가 리소스

- 메트릭 쿼리.
- 클러스터에 대한 데이터 수집.

6장. 설치 제거

OpenShift Container Platform 웹 콘솔 또는 명령줄을 사용하여 OpenShift 샌드박스 컨테이너를 설치 제거할 수 있습니다.

워크플로우 설치 제거

1. 워크로드 Pod를 삭제합니다.
2. **KataConfig** 사용자 지정 리소스를 삭제합니다.
3. OpenShift 샌드박스 컨테이너 Operator를 설치 제거합니다.
4. **KataConfig** 사용자 지정 리소스 정의를 삭제합니다.

6.1. 웹 콘솔을 사용하여 설치 제거

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너를 설치 제거할 수 있습니다.

6.1.1. 워크로드 Pod 삭제

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 워크로드 Pod를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift 샌드박스 컨테이너 런타임 클래스를 사용하는 Pod 목록이 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **워크로드 → Pod** 로 이동합니다.
2. **이름으로 검색 필드에 삭제할 Pod의 이름을** 입력합니다.
3. 포드 이름을 클릭하여 엽니다.
4. **세부 정보** 페이지에서 **런타임 클래스에** 대해 **kata** 또는 **kata-remote** 가 표시되는지 확인합니다.
5. **옵션** 메뉴  를 클릭하고 **Pod 삭제** 를 선택합니다.
6. **삭제**를 클릭합니다.

6.1.2. KataConfig CR 삭제

웹 콘솔을 사용하여 **KataConfig** CR(사용자 정의 리소스)을 삭제할 수 있습니다. **KataConfig** CR을 삭제하면 클러스터에서 **kata** 런타임 및 관련 리소스가 제거되고 제거됩니다.

중요

KataConfig CR을 삭제하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **kata** 를 **runtimeClass** 로 사용하는 실행 중인 모든 Pod를 삭제했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
2. **이름으로 검색 필드를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 검색합니다.**
3. Operator를 클릭하여 엽니다. 그런 다음 **KataConfig** 탭을 선택합니다.
4. **KataConfig** 리소스의 **옵션** 메뉴 를 클릭한 다음 **KataConfig삭제** 를 선택합니다.
5. 확인 창에서 **삭제**를 클릭합니다.

다음 단계를 진행하기 전에 **kata** 런타임 및 리소스가 제거되고 작업자 노드가 재부팅될 때까지 기다립니다.

6.1.3. Operator 설치 제거

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치 제거할 수 있습니다. Operator를 설치 제거하면 해당 Operator의 카탈로그 서브스크립션, Operator group, CSV(클러스터 서비스 버전)가 제거됩니다. 그런 다음 **openshift-sandboxed-containers-operator** 네임스페이스를 삭제합니다.

사전 요구 사항

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

프로세스

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**로 이동합니다.
2. **이름으로 검색 필드에 OpenShift 샌드박스 컨테이너 Operator 이름을**입력합니다.

3. Operator의 **옵션** 메뉴 를 클릭하고 **Operator 설치 제거**를 선택합니다.
4. 확인 창에서 **설치 제거**를 클릭합니다.
5. **이름으로 검색 필드에 openshift-sandboxed-containers-operator 이름을** 입력합니다.
6. 네임스페이스의 **옵션** 메뉴 를 클릭하고 네임스페이스 **삭제를 선택합니다**.

참고

네임스페이스 삭제 옵션을 사용할 수 없으면 네임스페이스를 삭제할 수 있는 권한이 없음을 의미합니다.

7. **네임스페이스 삭제** 창에서 **openshift-sandboxed-containers-operator** 를 입력하고 **삭제** 를 클릭합니다.
8. **삭제**를 클릭합니다.

6.1.4. KataConfig CRD 삭제

OpenShift Container Platform 웹 콘솔을 사용하여 **KataConfig** CRD(사용자 정의 리소스 정의)를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **KataConfig** CR을 삭제했습니다.
- OpenShift 샌드박스 컨테이너 Operator를 설치 제거했습니다.

프로세스

1. 웹 콘솔에서 **Administration → CustomResourceDefinitions** 로 이동합니다.
2. **이름으로 검색 필드에 KataConfig 이름을** 입력합니다.
3. **KataConfig** CRD의 **옵션** 메뉴 를 클릭하고 **CustomResourceDefinition 삭제** 를 선택합니다.
4. 확인 창에서 **삭제**를 클릭합니다.
5. **KataConfig** CRD가 목록에서 사라질 때까지 기다립니다. 이 작업은 몇 분 정도 걸릴 수 있습니다.

6.2. CLI를 사용하여 설치 제거

CLI(명령줄 인터페이스)를 사용하여 OpenShift 샌드박스 컨테이너를 설치 제거할 수 있습니다.

6.2.1. 워크로드 Pod 삭제

CLI를 사용하여 OpenShift 샌드박스 컨테이너 워크로드 Pod를 삭제할 수 있습니다.

사전 요구 사항

- JSON 프로세서(**jq**) 유틸리티가 설치되어 있습니다.

프로세스

1. 다음 명령을 실행하여 Pod를 검색합니다.

```
$ oc get pods -A -o json | jq -r '.items[] | \
  select(.spec.runtimeClassName == "<runtime>").metadata.name' 1
```

1 베어 메탈 배포에는 **kata** 를 지정합니다. 퍼블릭 클라우드, IBM Z® 및 IBM® LinuxONE 배포에는 **kata-remote** 를 지정합니다.

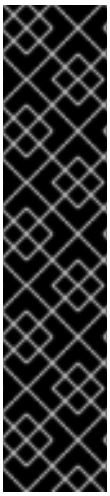
2. 다음 명령을 실행하여 각 Pod를 삭제합니다.

```
$ oc delete pod <pod>
```

6.2.2. KataConfig CR 삭제

명령줄을 사용하여 **KataConfig** CR(사용자 정의 리소스)을 삭제할 수 있습니다.

KataConfig CR을 삭제하면 클러스터에서 런타임 및 관련 리소스가 제거됩니다.



중요

KataConfig CR을 삭제하면 작업자 노드가 자동으로 재부팅됩니다. 재부팅에는 10분에서 60분 이상 걸릴 수 있습니다. 재부팅 시간을 방해하는 요소는 다음과 같습니다.

- 더 많은 작업자 노드가 있는 대규모 OpenShift Container Platform 배포
- BIOS 및 Cryostat 유틸리티 활성화.
- SSD가 아닌 하드 드라이브에 배포합니다.
- 가상 노드가 아닌 베어 메탈과 같은 물리적 노드에 배포됩니다.
- 느린 CPU 및 네트워크입니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

- 다음 명령을 실행하여 **KataConfig** CR을 삭제합니다.

```
$ oc delete kataconfig <kataconfig>
```

OpenShift 샌드박스된 컨테이너 Operator는 클러스터에서 런타임을 활성화하기 위해 처음 생성된 모든 리소스를 제거합니다.

검증

KataConfig CR을 삭제하면 모든 작업자 노드가 재부팅될 때까지 CLI가 응답하지 않습니다. 확인을 수행하기 전에 삭제 프로세스를 완료해야 합니다.

- **KataConfig** 사용자 지정 리소스가 삭제되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get kataconfig <kataconfig>
```

출력 예

```
No KataConfig instances exist
```

6.2.3. Operator 설치 제거

CLI를 사용하여 OpenShift 샌드박스 컨테이너 Operator를 설치 제거할 수 있습니다. Operator 서브스크립션, Operator group, CSV(클러스터 서비스 버전) 및 네임스페이스를 삭제하여 Operator를 설치 제거합니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- 명령줄 JSON 프로세서(**jq**)를 설치했습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. 다음 명령을 실행하여 서브스크립션에서 OpenShift 샌드박스 컨테이너의 CSV(클러스터 서비스 버전) 이름을 가져옵니다.

```
CSV_NAME=$(oc get csv -n openshift-sandboxed-containers-operator -o=custom-columns=:metadata.name)
```

2. 다음 명령을 실행하여 OLM(Operator Lifecyle Manager)에서 Operatorsubscription을 삭제합니다.

```
$ oc delete subscription sandboxed-containers-operator -n openshift-sandboxed-containers-operator
```

3. 다음 명령을 실행하여 OpenShift 샌드박스 컨테이너의 CSV 이름을 삭제합니다.

```
$ oc delete csv ${CSV_NAME} -n openshift-sandboxed-containers-operator
```

4. 다음 명령을 실행하여 Operator 그룹 이름을 가져옵니다.

```
$ OG_NAME=$(oc get operatorgroup -n openshift-sandboxed-containers-operator -o=jsonpath={..name})
```

5. 다음 명령을 실행하여 Operator 그룹 이름을 삭제합니다.

```
$ oc delete operatorgroup ${OG_NAME} -n openshift-sandboxed-containers-operator
```

6. 다음 명령을 실행하여 Operator 네임스페이스를 삭제합니다.

```
$ oc delete namespace openshift-sandboxed-containers-operator
```

6.2.4. KataConfig CRD 삭제

명령줄을 사용하여 **KataConfig** CRD(사용자 정의 리소스 정의)를 삭제할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **KataConfig** CR을 삭제했습니다.
- OpenShift 샌드박스 컨테이너 Operator를 설치 제거했습니다.

프로세스

1. 다음 명령을 실행하여 **KataConfig** CRD를 삭제합니다.

```
$ oc delete crd kataconfigs.kataconfiguration.openshift.io
```

검증

- **KataConfig** CRD가 삭제되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get crd kataconfigs.kataconfiguration.openshift.io
```

출력 예

```
Unknown CR KataConfig
```

7장. 업그레이드

OpenShift 샌드박스 컨테이너 구성 요소의 업그레이드는 다음 세 단계로 구성됩니다.

1. OpenShift Container Platform을 업그레이드하여 **Kata** 런타임 및 해당 종속 항목을 업데이트합니다.
2. OpenShift 샌드박스 컨테이너 Operator를 업그레이드하여 Operator 서브스크립션을 업데이트합니다.

아래에 언급된 한 가지 예외를 제외하고 OpenShift 샌드박스 컨테이너 Operator 업그레이드 전이나 후에 OpenShift Container Platform을 업그레이드할 수 있습니다. OpenShift 샌드박스 컨테이너 Operator를 업그레이드한 직후 **KataConfig** 패치를 항상 적용합니다.

7.1. 리소스 업그레이드

OpenShift 샌드박스 컨테이너 리소스는 RHCOS(Red Hat Enterprise Linux CoreOS) 확장을 사용하여 클러스터에 배포됩니다.

RHCOS 확장 **샌드박스 컨테이너에** 는 Kata 컨테이너 런타임, 하이퍼바이저 QEMU 및 기타 종속 항목과 같은 OpenShift 샌드박스 컨테이너를 실행하는 데 필요한 구성 요소가 포함되어 있습니다. 클러스터를 새 OpenShift Container Platform 릴리스로 업그레이드하여 확장을 업그레이드합니다.

OpenShift Container Platform 업그레이드에 대한 자세한 내용은 클러스터 업데이트를 참조하십시오.

7.2. OPERATOR 업그레이드

OLM(Operator Lifecycle Manager)을 사용하여 OpenShift 샌드박스 컨테이너 Operator를 수동 또는 자동으로 업그레이드합니다. 초기 배포 중에 수동 또는 자동 업그레이드 중 하나를 선택하면 향후 업그레이드 모드가 결정됩니다. 수동 업그레이드의 경우 OpenShift Container Platform 웹 콘솔에는 클러스터 관리자가 설치할 수 있는 사용 가능한 업데이트가 표시됩니다.

OLM(Operator Lifecycle Manager)에서 OpenShift 샌드박스 컨테이너 Operator를 업그레이드하는 방법에 대한 자세한 내용은 설치된 Operator 업데이트를 참조하십시오.

8장. 문제 해결

OpenShift 샌드박스 컨테이너의 문제를 해결할 때 지원 케이스를 열고 **must-gather** 툴을 사용하여 디버깅 정보를 제공할 수 있습니다.

클러스터 관리자인 경우 로그를 직접 검토하여 더 자세한 로그 수준을 활성화할 수도 있습니다.

8.1. RED HAT 지원을 위한 데이터 수집

지원 사례를 여는 경우 클러스터에 대한 디버깅 정보를 Red Hat 지원에 제공하면 도움이 됩니다.

must-gather 툴을 사용하면 가상 머신 및 OpenShift 샌드박스 컨테이너와 관련된 기타 데이터를 포함하여 OpenShift Container Platform 클러스터에 대한 진단 정보를 수집할 수 있습니다.

즉각 지원을 받을 수 있도록 OpenShift Container Platform 및 OpenShift 샌드박스 컨테이너 둘 다에 대한 진단 정보를 제공하십시오.

must-gather 툴 사용

oc adm must-gather CLI 명령은 다음을 포함하여 문제를 디버깅하는 데 필요할 가능성이 높은 클러스터에서 정보를 수집합니다.

- 리소스 정의
- 서비스 로그

기본적으로 **oc adm must-gather** 명령은 기본 플러그인 이미지를 사용하고 **./must-gather.local** 에 씁니다.

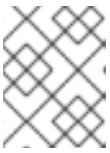
또는 다음 섹션에 설명된 대로 적절한 인수로 명령을 실행하여 특정 정보를 수집할 수 있습니다.

- 하나 이상의 특정 기능과 관련된 데이터를 수집하려면 다음 섹션에 나열된 대로 이미지와 함께 **--image** 인수를 사용합니다.
  예를 들면 다음과 같습니다.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

- 감사 로그를 수집하려면 다음 섹션에 설명된 대로 **-- /usr/bin/gather_audit_logs** 인수를 사용하십시오.
  예를 들면 다음과 같습니다.

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```



참고

감사 로그는 파일 크기를 줄이기 위해 기본 정보 세트의 일부로 수집되지 않습니다.

oc adm must-gather 를 실행하면 클러스터의 새 프로젝트에 임의의 이름이 있는 새 Pod가 생성됩니다. 해당 Pod에 대한 데이터가 수집되어 **must-gather.local**로 시작하는 새 디렉터리에 저장됩니다. 이 디렉터리는 현재 작업 중인 디렉터리에 생성되어 있습니다.

예를 들면 다음과 같습니다.

■

```
NAMESPACE                 NAME              READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running     0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running     0          72s
...
```

필요한 경우 **--run-namespace** 옵션을 사용하여 특정 네임스페이스에서 **oc adm must-gather** 명령을 실행할 수 있습니다.

예를 들면 다음과 같습니다.

```
$ oc adm must-gather --run-namespace <namespace> --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

8.2. 로그 데이터 수집

OpenShift 샌드박스 컨테이너와 관련된 기능 및 오브젝트는 다음과 같습니다.

- OpenShift 샌드박스 컨테이너 리소스에 속하는 모든 네임스페이스 및 해당 하위 오브젝트
- 모든 OpenShift 샌드박스 컨테이너 CRD(사용자 정의 리소스 정의)

kata 런타임으로 실행되는 각 Pod에 대해 다음 구성 요소 로그를 수집할 수 있습니다.

- Kata 에이전트 로그
- Kata 런타임 로그
- QEMU 로그
- 감사 로그
- CRI-O 로그

8.2.1. CRI-O 런타임의 디버그 로그 활성화

KataConfig CR에서 **logLevel** 필드를 업데이트하여 디버그 로그를 활성화할 수 있습니다. 이렇게 하면 OpenShift 샌드박스 컨테이너를 실행하는 작업자 노드의 CRI-O 런타임의 로그 수준이 변경됩니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

1. 기존 **KataConfig** CR의 **logLevel** 필드를 **debug** 로 변경합니다.

   ```
   $ oc patch kataconfig <kataconfig> --type merge --patch '{"spec":{"logLevel":"debug"}}'
   ```

2. **UPDATED** 의 값이 **True** 가 될 때까지 **kata-oc** 머신 구성 풀을 모니터링하여 모든 작업자 노드가 업데이트됨을 나타냅니다.

```
$ oc get mcp kata-oc
```

출력 예

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

검증

1. 머신 구성 풀에서 노드로 디버그 세션을 시작합니다.

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

2. root 디렉토리를 **/host** 로 변경합니다.

```
# chroot /host
```

3. **crio.conf** 파일의 변경 사항을 확인합니다.

```
# crio config | egrep 'log_level
```

출력 예

```
log_level = "debug"
```

8.2.2. 구성 요소의 디버그 로그 보기

클러스터 관리자는 디버그 로그를 사용하여 문제를 해결할 수 있습니다. 각 노드의 로그는 노드 저널에 출력됩니다.

다음 OpenShift 샌드박스 컨테이너 구성 요소의 로그를 검토할 수 있습니다.

- Kata 에이전트
- Kata 런타임 (**containerd-shim-kata-v2**)
- **virtiofsd**

QEMU는 경고 및 오류 로그만 생성합니다. 이러한 경고 및 오류는 추가 **qemuPid** 필드를 사용하여 Kata 런타임 로그와 CRI-O 로그 모두에서 노드 저널에 출력됩니다.

QEMU 로그 예

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.587116986Z"
level=info msg="Start logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu

Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.607339014Z"
```

```
level=error msg="qemu-kvm: -machine q35,accel=kvm,kernel_irqchip=split,foo: Expected '=' after
parameter 'foo'" name=containerd-shim-v2 pid=2241647 qemuPid=2241693
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu

Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.60890737Z"
level=info msg="Stop logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

Kata 런타임은 QEMU가 시작될 때 **로깅 QEMU 시작** 및 QEMU가 **중지되면 로깅 QEMU** 를 중지합니다. 이러한 두 로그 메시지 사이에 **qemuPid** 필드가 있는 오류가 표시됩니다. QEMU의 실제 오류 메시지가 빨간색으로 표시됩니다.

QEMU 게스트의 콘솔도 노드 저널에 출력됩니다. Kata 에이전트 로그와 함께 게스트 콘솔 로그를 볼 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

프로세스

- Kata 에이전트 로그 및 게스트 콘솔 로그를 검토하려면 다음 명령을 실행합니다.

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g “reading guest
console”
```

- Kata 런타임 로그를 검토하려면 다음 명령을 실행합니다.

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata
```

- **virtiofsd** 로그를 검토하려면 다음 명령을 실행합니다.

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t virtiofsd
```

- QEMU 로그를 검토하려면 다음 명령을 실행합니다.

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "qemuPid=\d+"
```

추가 리소스

- OpenShift Container Platform 문서에서 클러스터에 대한 데이터 수집

부록 A. KATACONFIG 상태 메시지

다음 표에는 두 개의 작업자 노드가 있는 클러스터의 **KataConfig** CR(사용자 정의 리소스)의 상태 메시지가 표시됩니다.

표 A.1. KataConfig 상태 메시지

상태	설명
초기 설치 **KataConfig** CR이 생성되고 두 작업자 모두에 **kata** 설치를 시작하면 다음 상태가 몇 초 동안 표시됩니다.	conditions: message: Performing initial installation of kata on cluster reason: Installing status: 'True' type: InProgress kataNodes: nodeCount: 0 readyNodeCount: 0
설치 몇 초 내에 상태가 변경됩니다.	kataNodes: nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0 - worker-1
설치 (Worker-1 설치 시작) 잠시 동안 하나의 노드가 **kata** 설치를 시작한 것을 나타내는 상태가 변경되고 다른 하나는 대기 상태입니다. 이는 한 번에 하나의 노드만 사용할 수 없기 때문입니다. 두 노드 모두 결국 **kata** 를 수신하기 때문에 **nodeCount** 는 2로 남아 있지만 **readyNodeCount** 는 현재 해당 상태에 도달하지 않았기 때문에 현재 0입니다.	kataNodes: installing: - worker-1 nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0
설치 (Worker-1 설치, worker-0 설치 시작) 잠시 후 **worker-1** 은 설치를 완료하여 상태가 변경됩니다. **readyNodeCount** 가 1로 업데이트되어 **worker-1** 이 이제 **kata** 워크로드를 실행할 준비가 되었음을 나타냅니다. 설치 프로세스 종료 시 런타임 클래스가 생성될 때까지 **kata** 워크로드를 예약하거나 실행할 수 없습니다.	kataNodes: installed: - worker-1 installing: - worker-0 nodeCount: 2 readyNodeCount: 1

상태	설명

설치됨
 	
설치하는 경우 두 작업자 모두 설치된 것으로 나열되고 **InProgress** 조건이 클러스터에 **kata** 를 성공적으로 설치하는 것을 나타내는 이유를 지정하지 않고 **False** 로 전환됩니다.
	conditions:
	  message: ""
	  reason: ""
	  status: 'False'
	  type: InProgress
	kataNodes:
	  installed:
	  - worker-0
	  - worker-1
	  nodeCount: 2
	  readyNodeCount: 2

상태	설명

초기 설치 제거
 	
kata 가 두 작업자 모두에 설치되어 있고**KataConfig** 를 삭제하여 클러스터에서 **kata** 를 제거하면 두 작업자 모두 잠시 대기 상태를 간략하게 입력합니다.
	conditions:
	  message: Removing kata from cluster
	  reason: Uninstalling
	  status: 'True'
	  type: InProgress
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  waitingToUninstall:
	  - worker-0
	  - worker-1

설치 제거
 	
몇 초 후에 작업자 중 하나가 제거를 시작합니다.
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  uninstalling:
	  - worker-1
	  waitingToUninstall:
	  - worker-0

설치 제거
 	
worker-1이 완료되고 worker-0이 제거를 시작합니다.
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  uninstalling:
	  - worker-0

참고

reason 필드는 다음 원인도 보고할 수 있습니다.

- **Failed**: 노드가 전환을 완료할 수 없는 경우 보고됩니다. **상태는 True** 이고 **메시지는 Node <node_name> Degraded: <error_message_from_the_node** >입니다.
- **BlockedByExistingKataPods**: 카타를 제거하는 동안 **kata** 런타임을 사용하는 클러스터에서 실행 중인 Pod가 있는 경우 보고됩니다. **status** 필드는 **False** 이고 **메시지는 kata" RuntimeClass를 사용하는 기존 Pod입니다. KataConfig 삭제의 경우 Pod를 수동으로 삭제하여 다음을 진행합니다.**. 클러스터 컨트롤 플레인과의 통신에 실패하는 경우 **Failed to list kata pods: <error_message** >와 같은 기술 오류 메시지가 표시될 수도 있습니다.