OpenShift Container Platform 4.19

etcd

etcd를 사용하여 중복성 제공

Last Updated: 2025-12-13

OpenShift Container Platform 4.19 etcd

etcd를 사용하여 중복성 제공

Legal Notice

Abstract

이 문서에서는 OpenShift Container Platform에서 클러스터 구성과 복원력에 대한 안정적인 접근 방식을 보장하는 etcd 사용에 대한 지침을 제공합니다.

Table of Contents

1장. ETCD 개요

etcd(발음은 et-see-dee)는 일관되고 분산된 키-값 저장소로, 전체가 메모리에 들어갈 수 있는 머신 클러스터에 소량의 데이터를 저장합니다. etcd는 많은 프로젝트의 핵심 구성 요소이지만 Kubernetes의 기본 데이터 저장소이며 이는 컨테이너 오케스트레이션의 표준 시스템입니다.

etcd를 사용하면 다음과 같은 여러 가지 방법으로 이점을 누릴 수 있습니다.

- 클라우드 네이티브 애플리케이션의 일관된 가동 시간을 유지하고 개별 서버가 실패하더라도 계속 작동합니다.
- Kubernetes의 모든 클러스터 상태 저장 및 복제
- 구성 데이터를 배포하여 노드 구성에 중복 및 복원력 제공

중요

기본 etcd 구성은 컨테이너 오케스트레이션을 최적화합니다. 최상의 결과를 얻으려면 설계된 대로 사용하세요.

1.1. ETCD 작동 방식

클러스터 구성 및 관리에 대한 안정적인 접근 방식을 보장하기 위해 etcd는 etcd Operator를 사용합니다. Operator는 OpenShift Container Platform과 같은 Kubernetes 컨테이너 플랫폼에서 etcd 사용을 간소화합니다.

또한 etcd Operator를 사용하여 OpenShift Container Platform 제어 평면에 대한 etcd 클러스터를 배포하고 관리할 수 있습니다. etcd 운영자는 다음과 같은 방법으로 클러스터 상태를 관리합니다.

- Kubernetes API를 사용하여 클러스터 상태를 관찰합니다.
- 현재 상태와 필요한 상태의 차이점을 분석합니다.
- etcd 클러스터 관리 API, Kubernetes API 또는 둘 다를 통해 차이점을 수정합니다.

참고

etcd에는 클러스터 상태가 있으며 이는 지속적으로 업데이트됩니다. 이 상태는 지속적으로 유지되므로 높은 빈도로 인해 작은 변화가 많이 발생합니다. 따라서 빠르고 대기 시간이 짧은 I/O로 etcd 클러스터 멤버를 백업하는 것이 중요합니다. etcd의 모범 사례에 대한 자세한 내용은 "etcd에 대한 권장 사례"를 참조하십시오.

추가 리소스

- etcd 관련 권장 사례

1.2. ETCD 성능 이해

복제된 노드의 클러스터로 작동하는 일관된 분산 키-값 저장소인 etcd는 한 노드를 리더로, 다른 노드를 팔로워로 선출하는 Raft 알고리즘을 따릅니다. 리더는 시스템의 현재 상태를 유지하고 팔로워가 최신 정보를 받도록 보장합니다.

리더 노드는 로그 복제를 담당합니다. 클라이언트로부터 들어오는 쓰기 트랜잭션을 처리하고 팔로워에게 브로드캐스트하는 Raft 로그 항목을 작성합니다.

kube-apiserver 와 같은 etcd 클라이언트가 쿼럼이 필요한 작업(예: 값 쓰기)을 요청하는 etcd 멤버에 연결할 때, 해당 etcd 멤버가 팔로워인 경우 트랜잭션을 리더에게 전달해야 한다는 메시지를 반환합니다.

etcd 클라이언트가 쿼럼이 필요한 작업(예: 값 쓰기)을 리더에게 요청하면 리더는 로컬 Raft 로그를 쓰는 동안 클라이언트 연결을 열어두고, 로그를 팔로워에게 브로드캐스트하고, 대다수의 팔로워가 실패 없이 로그를 커밋했다는 확인을 기다릴 때까지 기다립니다. 리더는 etcd 클라이언트에게 확인응답을 보내고 세션을 닫습니다. 팔로워로부터 실패 알림을 받고 합의가 이루어지지 않으면 리더는 오류 메시지를 클라이언트에게 반환하고 세션을 닫습니다.

etcd에 대한 OpenShift 컨테이너 플랫폼 타이머 조건

OpenShift Container Platform은 각 플랫폼에 최적화된 etcd 타이머를 유지 관리합니다. OpenShift Container Platform은 각 플랫폼 제공업체에 맞게 최적화된 검증된 값을 규정하고 있습니다. **platform=none** 또는 **platform=metal** 값이 있는 기본 **etcd 타이머** 매개변수는 다음과 같습니다.

```
- name: ETCD_ELECTION_TIMEOUT (1)
  value: "1000"
  ...
- name: ETCD_HEARTBEAT_INTERVAL (2)
  value: "100"
```

(1) 이 시간 초과는 팔로워 노드가 리더가 되기 위해 시도하기 전에 하트비트를 듣지 않고 기다리는 시간입니다.

(2) 리더가 팔로워들에게 자신이 여전히 리더임을 알리는 빈도입니다.

이러한 매개변수는 제어 평면이나 etcd에 대한 모든 정보를 제공하지 않습니다. etcd 클러스터는 디스크 지연 시간에 민감합니다. etcd는 제안을 로그에 유지해야 하므로 다른 프로세스의 디스크 활동으로 인해 **fsync** 대기 시간이 길어질 수 있습니다. 결과적으로 etcd가 하트비트를 놓칠 수 있고, 이로 인해 요청 시간이 초과되고 일시적인 리더 손실이 발생할 수 있습니다. 리더 손실 및 재선출 동안 Kubernetes API는 서비스에 영향을 미치는 이벤트와 클러스터 불안정성을 유발하는 요청을 처리할 수 없습니다.

디스크 지연 시간이 etcd에 미치는 영향

etcd 클러스터는 디스크 지연 시간에 민감합니다. 제어 평면 환경에서 etcd가 경험하는 디스크 지연 시간을 파악하려면 Flexible I/O Tester(fio) 테스트나 제품군을 실행하여 OpenShift Container Platform에서 etcd 디스크 성능을 확인하세요.

중요

특정 시점의 디스크 지연 시간을 측정하려면 **fio** 테스트만 사용하세요. 이 테스트는 장기적인 디스크 동작과 프로덕션 환경에서 etcd로 인해 발생하는 기타 디스크 작업 부하를 고려하지 않습니다.

다음 예에서 보듯이 최종 보고서에서 디스크가 etcd에 적합한 것으로 분류되었는지 확인하세요.

```
...
99th percentile of fsync is 5865472 ns
99th percentile of the fsync is within the suggested threshold: - 20 ms, the disk can be used to host
etcd
```

대기 시간이 긴 디스크를 사용하면 다음 예와 같이 해당 디스크가 etcd에 권장되지 않는다는 메시지가 표시됩니다.

```
...
99th percentile of fsync is 15865472 ns
99th percentile of the fsync is greater than the suggested value which is 20 ms, faster disks are
suggested to host etcd for better performance
```

클러스터 배포가 제안된 지연 시간을 충족하지 못하는 etcd용 디스크를 사용하는 여러 데이터 센터에 걸쳐 있는 경우 서비스에 영향을 미치는 오류가 발생할 수 있습니다. 또한, 제어 평면이 견딜 수 있는 네트워크 지연 시간이 획기적으로 감소합니다.

네트워크 지연 및 지터가 etcd에 미치는 영향

최대 전송 단위(MTU) 검색 및 검증 섹션에 설명된 도구를 사용하여 평균 및 최대 네트워크 지연 시간을 얻습니다.

하트비트 간격의 값은 멤버 간 평균 왕복 시간(RTT)의 최대값과 거의 같아야 하며, 일반적으로 왕복 시간의 약 1.5배입니다. OpenShift Container Platform의 기본 하트비트 간격이 100ms인 경우, 제어 평면 노드 간의 권장 RTT는 33ms 미만이며, 최대값은 66ms 미만입니다(66ms x 1.5 = 99ms). 네트워크 지연 시간이 길어지면 서비스에 영향을 미치는 이벤트와 클러스터 불안정이 발생할 수 있습니다.

네트워크 지연 시간은 구리, 광섬유, 무선 또는 위성과 같은 전송 네트워크의 기술, 전송 네트워크의 네트워크 장치 수와 품질 및 기타 요소를 포함한 요소에 의해 결정됩니다.

정확한 계산을 위해 네트워크 지터와 네트워크 지연 시간을 고려하세요. *네트워크 지터* 는 네트워크 지연 시간의 변화 또는 수신 패킷의 지연 시간의 변화입니다. 네트워크가 효율적인 환경에서는 지터가 0이어야 합니다. 네트워크 지터는 etcd의 네트워크 지연 시간 계산에 영향을 미칩니다. 왜냐하면 시간에 따른 실제 네트워크 지연 시간은 RTT에 지터를 더하거나 뺀 값이기 때문입니다.

예를 들어, 최대 지연 시간이 80ms이고 지터가 30ms인 네트워크는 지연 시간이 110ms가 되는데, 이는 etcd가 하트비트를 놓치는 것을 의미합니다. 이러한 조건으로 인해 요청 시간 초과 및 일시적인 리더 손실이 발생합니다. 리더 손실 및 재선출 동안 Kubernetes API는 서비스에 영향을 미치는 이벤트와 클러스터 불안정성을 유발하는 요청을 처리할 수 없습니다.

etcd에 대한 합의 지연의 영향

해당 절차는 활성 클러스터에서만 실행될 수 있습니다. 클러스터 배포를 계획하는 동안 디스크 또는 네트워크 테스트를 완료해야 합니다. 해당 테스트는 배포 후 클러스터 상태를 검증하고 모니터링합니다.

etcdctl CLI를 사용하면 etcd가 합의에 도달하는 데 걸리는 지연 시간을 관찰할 수 있습니다. etcd 포드 중 하나를 식별한 다음 엔드포인트 상태를 검색해야 합니다.

etcd 피어 왕복 시간이 성능에 미치는 영향

etcd 피어 왕복 시간은 네트워크 왕복 시간과 다릅니다. 이 계산은 구성원 간에 복제가 얼마나 빨리 발생할 수 있는지에 대한 종단 간 테스트 지표입니다.

Etcd 피어 왕복 시간은 Etcd가 모든 Etcd 멤버 간에 클라이언트 요청을 복제하는 데 걸리는 지연 시간을 보여주는 지표입니다. OpenShift Container Platform 콘솔은 다양한 etcd 메트릭을 시각화하는 대시보드를 제공합니다. 콘솔에서 **관찰 → 대시보드를** 클릭합니다. 드롭다운 목록에서 **etcd를** 선택합니다.

etcd 피어의 왕복 시간을 요약한 플롯은 etcd **대시보드** 페이지의 끝 부분에 있습니다.

etcd에 대한 데이터베이스 크기의 영향

etcd 데이터베이스 크기는 etcd 조각 모음 프로세스를 완료하는 데 걸리는 시간에 직접적인 영향을 미칩니다. OpenShift Container Platform은 최소 45%의 조각화를 감지하면 한 번에 하나의 etcd 멤버에 대해 etcd 조각 모음을 자동으로 실행합니다. 조각 모음 프로세스 동안 etcd 멤버는 어떠한 요청도 처리할 수 없

습니다. 작은 etcd 데이터베이스에서는 조각 모음 프로세스가 1초 이내에 완료됩니다. 대규모 etcd 데이터베이스의 경우 디스크 지연 시간이 조각화 시간에 직접적인 영향을 미쳐 조각 모음이 진행되는 동안 작업이 차단되어 추가 지연 시간이 발생합니다.

etcd 데이터베이스의 크기는 네트워크 파티션이 일정 기간 동안 제어 평면 노드를 격리하고 통신이 재설정된 후 제어 평면을 동기화해야 하는 경우 고려해야 할 요소입니다.

etcd 데이터베이스의 크기를 제어하기 위한 옵션은 시스템의 운영자와 애플리케이션에 따라 달라지므로 최소한의 옵션만 존재합니다. 시스템이 작동하는 대기 시간 범위를 고려할 때 etcd 데이터베이스 크기별로 동기화나 조각 모음의 효과를 고려하세요.

효과의 규모는 배포에 따라 다릅니다. 조각 모음을 완료하는 데 걸리는 시간으로 인해 etcd 멤버가 조각 모음 프로세스 동안 업데이트를 수락할 수 없으므로 트랜잭션 속도가 저하됩니다. 마찬가지로, 높은 변경률을 지닌 대규모 데이터베이스의 etcd 재동기화에 걸리는 시간은 시스템의 트랜잭션 속도와 트랜잭션 지연 시간에 영향을 미칩니다. 계획해야 할 영향의 유형에 대해 다음 두 가지 예를 살펴보세요.

데이터베이스 크기에 따른 etcd 조각 모음 효과에 대한 첫 번째 예는 1GB의 etcd 데이터베이스를 초당 80MB의 속도로 느린 7200RPM 디스크에 쓰는 데 약 1분 40초가 걸린다는 것입니다. 이런 시나리오에서 조각 모음 프로세스는 적어도 조각 모음을 완료하는 데 이 정도 시간이 걸립니다.

etcd 동기화에 대한 데이터베이스 크기의 영향에 대한 두 번째 예는 제어 평면 노드 중 하나가 연결 해제되는 동안 etcd 데이터베이스의 10%가 변경되면 동기화에서 최소 100MB를 전송해야 한다는 것입니다. 1Gbps 링크를 통해 100MB를 전송하는 데 800ms가 걸립니다. Kubernetes API를 사용하여 정기적인 트랜잭션이 있는 클러스터에서 etcd 데이터베이스 크기가 클수록 네트워크 불안정성이 커지고 제어 평면 불안정성이 발생합니다.

OpenShift Container Platform에서 etcd 대시보드에는 etcd 데이터베이스의 크기를 보고하는 플롯이 있습니다. 또는 **etcdctl** 도구를 사용하여 CLI에서 데이터베이스 크기를 얻을 수 있습니다.

```
# oc get pods -n openshift-etcd -l app=etcd
```

출력 예

```
NAME     READY   STATUS    RESTARTS   AGE
etcd-m0   4/4     Running   4          22h
etcd-m1   4/4     Running   4          22h
etcd-m2   4/4     Running   4          22h
```

```
# oc exec -t etcd-m0 -- etcdctl endpoint status -w simple | cut -d, -f 1,3,4
```

출력 예

```
https://198.18.111.12:2379, 3.5.6, 1.1 GB
https://198.18.111.13:2379, 3.5.6, 1.1 GB
https://198.18.111.14:2379, 3.5.6, 1.1 GB
```

Kubernetes API 트랜잭션 속도가 etcd에 미치는 영향

확장된 제어 평면을 사용하는 경우 Kebernetes API 트랜잭션 속도는 특정 배포의 특성에 따라 달라집니다. 이는 etcd 디스크 지연 시간, etcd 왕복 시간, API에 기록되는 객체 크기의 조합에 따라 달라집니다. 결과적으로, 확장된 제어 평면을 사용할 때 클러스터 관리자는 해당 환경에서 가능한 지속적인 트랜잭션 속도를 확인하기 위해 환경을 테스트해야 합니다. 이 목적으로 **kube-burner** 도구를 사용할 수 있습니다.

사용자 환경에 맞는 Kubernetes API 트랜잭션 속도 결정

Kubernetes API의 거래율을 측정하지 않고는 알 수 없습니다. 제어 평면의 부하 테스트에 사용되는 도구 중 하나는 **kube-burner** 입니다. 이 바이너리는 OpenShift Container Platform 클러스터를 테스트하기 위한 OpenShift Container Platform 래퍼를 제공합니다. 클러스터나 노드 밀도를 테스트하는 데 사용됩니다. 제어 평면을 테스트하기 위해 **kube-burner ocp에는 cluster-density** , **cluster-density-v2** , **cluster-density-ms의** 세 가지 워크로드 프로필이 있습니다. 각 작업 부하 프로필은 제어를 로드하도록 설계된 일련의 리소스를 생성합니다.

2장. ETCD 관련 권장 사례

다음 문서에서는 etcd에 권장되는 성능 및 확장성 관행에 대한 정보를 제공합니다.

2.1. ETCD에 대한 저장 관행

etcd는 디스크에 데이터를 쓰고 제안을 디스크에 저장하므로 성능은 디스크 성능에 따라 달라집니다. etcd는 특별히 I/O를 많이 사용하지는 않지만 최적의 성능과 안정성을 위해 저지연 블록 장치가 필요합니다. etcd의 합의 프로토콜은 메타데이터를 로그(WAL)에 지속적으로 저장하는 데 의존하므로 etcd는 디스크 쓰기 지연에 민감합니다. 디스크가 느리거나 다른 프로세스에서 디스크 활동이 발생하면 fsync 대기 시간이 길어질 수 있습니다.

이러한 지연 시간으로 인해 etcd가 하트비트를 놓치고, 새로운 제안을 제때 디스크에 커밋하지 못하고, 궁극적으로 요청 시간 초과와 일시적인 리더 손실을 경험할 수 있습니다. 쓰기 대기 시간이 길어지면 OpenShift API 속도가 느려서 클러스터 성능에 영향을 미칩니다. 이러한 이유로 I/O에 민감하거나 집약적이며 동일한 기본 I/O 인프라를 공유하는 제어 평면 노드에 다른 작업 부하를 함께 배치하지 마십시오.

fdatasync를 포함하여 10ms 이내에 8KB의 50 IOPS 이상을 순차적으로 쓸 수 있는 블록 장치에서 etcd를 실행합니다. 부하가 심한 클러스터의 경우 8000바이트(2ms)의 연속 500 IOPS가 권장됩니다. 이러한 숫자를 측정하려면 **fio** 명령과 같은 벤치마킹 도구를 사용할 수 있습니다.

이러한 성능을 달성하려면 낮은 지연 시간과 높은 처리량을 제공하는 SSD 또는 NVMe 디스크로 지원되는 머신에서 etcd를 실행하세요. 메모리 셀 당 1비트를 제공하고 내구성과 안정성이 뛰어나며 쓰기 작업이 많은 작업에 적합한 SLC(Single-Level Cell) 솔리드 스테이트 드라이브(SSD)를 생각해 보세요.

참고

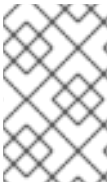
etcd의 부하는 노드와 Pod 수와 같은 정적 요소와 Pod 자동 확장, Pod 재시작, 작업 실행 및 기타 작업 관련 이벤트로 인한 엔드포인트 변경을 포함한 동적 요소로 인해 발생합니다. etcd 설정의 크기를 정확하게 조정하려면 워크로드의 특정 요구 사항을 분석해야 합니다. etcd의 부하에 영향을 미치는 노드, 포드 및 기타 관련 요소의 수를 고려하세요.

다음 하드 드라이브 관행은 최적의 etcd 성능을 제공합니다.

- 전용 etcd 드라이브를 사용합니다. iSCSI와 같이 네트워크를 통해 통신하는 드라이브는 사용하지 마세요. etcd 드라이브에 로그 파일이나 기타 무거운 작업 부하를 두지 마세요.
- 빠른 읽기 및 쓰기 작업을 지원하려면 대기 시간이 짧은 드라이브를 선호합니다.
- 더 빠른 압축 및 조각 모음을 위해 고대역폭 쓰기를 선호합니다.
- 장애 발생 시 빠른 복구를 위해 고대역폭 읽기를 선호합니다.
- 최소한으로 솔리드 스테이트 드라이브를 선택하세요. 프로덕션 환경에서는 NVMe 드라이브를 선호합니다.
- 안정성을 높이려면 서버급 하드웨어를 사용하세요.
- NAS 또는 SAN 설정 및 회전 드라이브를 방지합니다. Ceph Rados Block Device(RBD) 및 기타 유형의 네트워크 연결 스토리지는 예측할 수 없는 네트워크 지연을 초래할 수 있습니다. 대규모로 etcd 노드에 빠른 스토리지를 제공하려면 PCI 패스스루를 사용하여 NVM 장치를 노드에 직접 전달합니다.
- **fio** 와 같은 유틸리티를 사용하여 항상 벤치마크를 실시하세요. 이러한 유틸리티를 사용하면 클러스터 성능이 향상되는 것을 지속적으로 모니터링할 수 있습니다.

- NFS(네트워크 파일 시스템) 프로토콜이나 기타 네트워크 기반 파일 시스템을 사용하지 마세요.

배포된 OpenShift Container Platform 클러스터에서 모니터링할 몇 가지 주요 지표는 etcd 디스크 쓰기 전 로그 기간과 etcd 리더 변경 횟수입니다. 이러한 지표를 추적하려면 Prometheus를 사용하십시오.



참고

일반 작업 중에 클러스터 내의 etcd 멤버 데이터베이스 크기가 달라질 수 있습니다. 리더 크기가 다른 멤버와 다르더라도 이러한 차이는 클러스터 업그레이드에 영향을 미치지 않습니다.

2.2. ETCD에 대한 하드웨어 검증

OpenShift Container Platform 클러스터를 생성하기 전이나 후에 etcd에 대한 하드웨어를 검증하려면 fio를 사용할 수 있습니다.

사전 요구 사항

- Podman이나 Docker와 같은 컨테이너 런타임은 테스트 중인 머신에 설치됩니다.
- 데이터는 **/var/lib/etcd** 경로에 기록됩니다.

프로세스

- Fio를 실행하고 결과를 분석합니다.
  - Podman을 사용하는 경우 다음 명령을 실행하세요.

```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

  - Docker를 사용하는 경우 다음 명령을 실행하세요.

```
$ sudo docker run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

실행에서 캡처된 fsync 지표의 99번째 백분위수를 비교한 후 속도가 10ms 미만인지를 확인하여 디스크 속도가 etcd를 호스팅하는 데 충분한지 여부가 출력에 표시됩니다. I/O 성능에 영향을 받을 수 있는 가장 중요한 etcd 메트릭은 다음과 같습니다.

- **etcd_disk_wal_fsync_duration_seconds_bucket** 메트릭은 etcd의 WAL fsync 기간을 보고합니다.
- **etcd_disk_backend_commit_duration_seconds_bucket** 메트릭은 etcd 백엔드 커밋 대기 시간을 보고합니다.
- **etcd_server_leader_changes_seen_total** 메트릭은 리더 변경 사항을 보고합니다.

etcd는 모든 멤버 간에 요청을 복제하므로 성능은 네트워크 입력/출력(IO) 대기 시간에 따라 달라집니다. 네트워크 지연 시간이 길어 지면 etcd 하트비트가 선택 시간 초과보다 오래 걸릴 수 있으며 이로 인해 리더 선택이 발생하여 클러스터가 손상될 수 있습니다. 배포된 OpenShift Container Platform 클러스터에서 모니터링되는 주요 메트릭은 각 etcd 클러스터 멤버에서 etcd 네트워크 피어 대기 시간의 99번째 백분위수입니다. 이러한 메트릭을 추적하려면 Prometheus를 사용하십시오.

histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[2m])) 메트릭은 etcd가 멤버 간 클라이언트 요청 복제를 완료하는 데 걸리는 왕복 시간을 보고합니다. 50ms 이하인지 확인하세요.

추가 리소스

- OpenShift Container Platform에서 **fio**를 사용하여 etcd 디스크 성능을 확인하는 방법
- OpenShift 컨테이너 플랫폼을 위한 etcd 성능 문제 해결 가이드

3장. ETCD에 대한 성능 고려 사항

OpenShift Container Platform에서 etcd의 최적의 성능과 확장성을 보장하려면 다음 연습을 완료할 수 있습니다.

3.1. ETCD에 대한 노드 확장

일반적으로 클러스터에는 3개의 제어 평면 노드가 있어야 합니다. 하지만 클러스터가 베어 메탈 플랫폼에 설치된 경우 최대 5개의 제어 평면 노드를 가질 수 있습니다. 기존 베어 메탈 클러스터에 제어 평면 노드가 5개 미만인 경우 설치 후 작업으로 클러스터를 확장할 수 있습니다.

예를 들어, 설치 후 제어 평면 노드를 3개에서 4개로 확장하려면 호스트를 추가하고 이를 제어 평면 노드로 설치할 수 있습니다. 그런 다음 etcd 운영자는 추가 제어 평면 노드를 고려하여 그에 맞게 확장됩니다.

클러스터를 4개 또는 5개의 제어 평면 노드로 확장하는 것은 베어 메탈 플랫폼에서만 가능합니다.

Assisted Installer를 사용하여 제어 평면 노드를 확장하는 방법에 대한 자세한 내용은 "API를 사용하여 호스트 추가" 및 "정상적인 클러스터에서 제어 평면 노드 교체"를 참조하세요.

다음 표는 다양한 크기의 클러스터에 대한 실패 허용 범위를 보여줍니다.

표 3.1. 클러스터 크기에 따른 실패 허용 범위

클러스터 크기	다수	고장 허용 범위
노드	1	0
노드	2	1
노드	3	1
노드	3	2

쿼럼 손실에서 복구하는 방법에 대한 자세한 내용은 "이전 클러스터 상태로 복원"을 참조하세요.

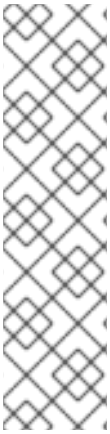
추가 리소스

- API를 사용하여 호스트 추가
- 정상 클러스터에서 제어 평면 노드 교체
- 클러스터 확장
- 이전 클러스터 상태로 복원

3.2. ETCD를 다른 디스크로 이동

성능 문제를 방지하거나 해결하기 위해 공유 디스크에서 별도의 디스크로 etcd를 이동할 수 있습니다.

MCO(Machine Config Operator)는 OpenShift Container Platform 4.19 컨테이너 스토리지를 위한 보조 디스크를 마운트하는 역할을 합니다.



참고

이 인코딩된 스크립트는 다음 장치 유형에 대한 장치 이름만 지원합니다.

SCSI 또는 SATA

/dev/sd*

가상 장치

/dev/vd*

NVMe

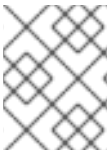
/dev/nvme*[0-9]*n*

제한

- 새 디스크가 클러스터에 연결되면 etcd 데이터베이스가 루트 마운트의 일부가 됩니다. 기본 노드가 다시 생성될 때 이는 보조 디스크나 의도된 디스크의 일부가 아닙니다. 결과적으로 기본 노드는 별도의 **/var/lib/etcd** 마운트를 생성하지 않습니다.

사전 요구 사항

- 클러스터의 etcd 데이터를 백업했습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 권한이 있는 클러스터에 액세스할 수 있습니다.
- 머신 구성을 업로드하기 전에 추가 디스크를 추가하세요.
- **MachineConfigPool**은 **metadata.labels[machineconfiguration.openshift.io/role]** 와 일치해야 합니다. 이는 컨트롤러, 워커 또는 사용자 정의 풀에 적용됩니다.



참고

이 절차는 **/var/** 과 같은 루트 파일 시스템의 일부를 설치된 노드의 다른 디스크나 파티션으로 이동하지 않습니다.

중요

제어 평면 머신 세트를 사용하는 경우 이 절차는 지원되지 않습니다.

프로세스

1. 클러스터에 새 디스크를 연결하고 디버그 셸에서 **lsblk** 명령을 실행하여 디스크가 노드에서 감지되는지 확인합니다.

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

```
# lsblk
```

   lsblk 명령에서 보고된 새 디스크의 장치 이름을 기록해 보세요.

2. 다음 스크립트를 만들고 **etcd-find-secondary-device.sh** 라고 이름을 지정합니다.

```
#!/bin/bash
set -uo pipefail

for device in <device_type_glob>; do 1
/usr/sbin/blkid "${device}" &> /dev/null
 if [ $? == 2  ]; then
    echo "secondary device found ${device}"
    echo "creating filesystem for etcd mount"
    mkfs.xfs -L var-lib-etcd -f "${device}" &> /dev/null
    udevadm settle
    touch /etc/var-lib-etcd-mount
    exit
 fi
done
echo "Couldn't find secondary block device!" >&2
exit 77
```

1 **<device_type_glob>**을 블록 장치 유형에 맞는 셸 글로브로 바꾸세요. SCSI 또는 SATA 드라이브의 경우 **/dev/sd***를 사용하고, 가상 드라이브의 경우 **/dev/vd***를 사용하고, NVMe 드라이브의 경우 **/dev/nvme*[0-9]*n***을 사용합니다.

3. **etcd-find-secondary-device.sh** 스크립트에서 base64로 인코딩된 문자열을 만들고 내용을 기록해 둡니다.

```
$ base64 -w0 etcd-find-secondary-device.sh
```

4. 다음과 같은 내용을 포함하는 **etcd-mc.yml** 이라는 **MachineConfig** YAML 파일을 만듭니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:
  config:
    ignition:
      version: 3.5.0
    storage:
      files:
        - path: /etc/find-secondary-device
          mode: 0755
          contents:
            source: data:text/plain;charset=utf-8;base64,
<encoded_etcd_find_secondary_device_script> 1
    systemd:
      units:
        - name: find-secondary-device.service
          enabled: true
          contents: |
            [Unit]
            Description=Find secondary device
            DefaultDependencies=false
            After=systemd-udev-settle.service
```

```
    Before=local-fs-pre.target
    ConditionPathExists=!/etc/var-lib-etcd-mount

    [Service]
    RemainAfterExit=yes
    ExecStart=/etc/find-secondary-device

    RestartForceExitStatus=77

    [Install]
    WantedBy=multi-user.target
- name: var-lib-etcd.mount
  enabled: true
  contents: |
    [Unit]
    Before=local-fs.target

    [Mount]
    What=/dev/disk/by-label/var-lib-etcd
    Where=/var/lib/etcd
    Type=xfs
    TimeoutSec=120s

    [Install]
    RequiredBy=local-fs.target
- name: sync-var-lib-etcd-to-etcd.service
  enabled: true
  contents: |
    [Unit]
    Description=Sync etcd data if new mount is empty
    DefaultDependencies=no
    After=var-lib-etcd.mount var.mount
    Before=crio.service

    [Service]
    Type=oneshot
    RemainAfterExit=yes
    ExecCondition=/usr/bin/test ! -d /var/lib/etcd/member
    ExecStart=/usr/sbin/setsebool -P rsync_full_access 1
    ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
    ExecStart=/usr/sbin/semanage fcontext -a -t container_var_lib_t '/var/lib/etcd(/.*)?'
    ExecStart=/usr/sbin/setsebool -P rsync_full_access 0
    TimeoutSec=0

    [Install]
    WantedBy=multi-user.target graphical.target
- name: restorecon-var-lib-etcd.service
  enabled: true
  contents: |
    [Unit]
    Description=Restore recursive SELinux security contexts
    DefaultDependencies=no
    After=var-lib-etcd.mount
    Before=crio.service

    [Service]
```

```
Type=oneshot
RemainAfterExit=yes
ExecStart=/sbin/restorecon -R /var/lib/etcd/
TimeoutSec=0

[Install]
WantedBy=multi-user.target graphical.target
```

1 **<encoded_etcd_find_secondary_device_script>**를 귀하가 기록해 둔 인코딩된 스크립트 내용으로 바꾸세요.

5. 생성된 **MachineConfig** YAML 파일을 적용합니다.

```
$ oc create -f etcd-mc.yml
```

검증 단계

- 노드의 디버그 셸에서 **grep /var/lib/etcd /proc/mounts** 명령을 실행하여 디스크가 마운트되었는지 확인하세요.

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

```
# grep -w "/var/lib/etcd" /proc/mounts
```

출력 예

```
/dev/sdb /var/lib/etcd xfs rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,noquota
0 0
```

추가 리소스

- RHCOS(Red Hat Enterprise Linux CoreOS)

3.3. ETCD 데이터 조각 모음

크기가 크고 밀집된 클러스터의 경우 키 공간이 지나치게 커져서 공간 할당량을 초과하면 etcd 성능이 저하될 수 있습니다. 주기적으로 etcd를 유지 관리하고 조각 모음을 수행하여 데이터 저장소의 공간을 확보합니다. Prometheus에서 etcd 메트릭을 모니터링하고 필요한 경우 조각 모음을 수행합니다. 그렇지 않으면 etcd가 클러스터 전체에 알람을 발생시켜 클러스터를 키 읽기와 삭제만 허용하는 유지 관리 모드로 전환할 수 있습니다.

다음과 같은 주요 지표를 모니터링하세요.

- 현재 할당량 한도인 **etcd_server_quota_backend_bytes**
- **etcd_mvcc_db_total_size_in_use_in_bytes** 는 기록 압축 후 실제 데이터베이스 사용량을 나타냅니다.
- **etcd_mvcc_db_total_size_in_bytes** 는 조각 모음을 기다리는 여유 공간을 포함한 데이터베이스 크기를 보여줍니다.

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

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

조각 모음이 자동으로 수행되지만 수동으로 트리거할 수도 있습니다.

참고

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

3.3.1. 자동 조각 모음

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

다음 로그 중 하나를 확인하여 조각 모음 프로세스가 성공했는지 확인합니다.

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

주의

자동 조각 모음은 Kubernetes 컨트롤러 관리자 등 다양한 OpenShift 핵심 구성 요소에서 리더 선출 실패를 일으킬 수 있으며, 이로 인해 실패한 구성 요소가 다시 시작됩니다. 재시작은 무해하며, 다음 실행 인스턴스로 장애 조치를 시작하거나 재시작 후 구성 요소가 다시 작업을 재개합니다.

성공적인 조각 모음에 대한 로그 출력 예

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

실패한 조각 모음에 대한 로그 출력 예

```
failed defrag on member: <member_name>, memberID: <member_id>: <error_message>
```

3.3.2. 수동 조각 모음

Prometheus 경고는 수동 조각 모음을 사용해야 하는 경우를 나타냅니다. 경고는 두 가지 경우에 표시됩니다.

- etcd가 사용 가능한 공간의 50% 이상을 10분 이상 사용하는 경우
- etcd가 10분 이상 전체 데이터베이스 크기의 50% 미만을 적극적으로 사용하는 경우

또한 PromQL 표현식 **((etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024**를 사용하여 조각 모음으로 해제될 etcd 데이터베이스 크기(MB)를 확인하여 조각 모음이 필요한지 여부를 판단할 수 있습니다.

주의

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

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

사전 요구 사항

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

프로세스

1. 리더가 최종 조각화 처리를 수행하므로 어떤 etcd 멤버가 리더인지 확인합니다.
   a. etcd pod 목록을 가져옵니다.

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

출력 예

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

   b. Pod를 선택하고 다음 명령을 실행하여 어떤 etcd 멤버가 리더인지 확인합니다.

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

출력 예

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see all of the containers in this pod.
+---------------------------+------------------+---------+---------+-----------+------------+-----------+
+------------+-------------------+--------+
|         ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+------------------+---------+---------+-----------+------------+-----------+
+------------+-------------------+--------+
```

```
| https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |           91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |           91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |           91624 |        |
+--------------------------+------------------+---------+---------+-----------+-----------+-----------
+------------+-------------------+--------+
```

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

2. etcd 멤버를 분리합니다.

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

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

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

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

   c. etcd 멤버를 분리합니다.

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

   출력 예

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

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

   d. 데이터베이스 크기가 감소되었는지 확인합니다.

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

   출력 예

   ```
   +--------------------------+------------------+---------+---------+-----------+-----------+-----------
   +------------+-------------------+--------+
   |        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
   RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
   +--------------------------+------------------+---------+---------+-----------+-----------+-----------
   +------------+-------------------+--------+
   | https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
   7 |      91624 |           91624 |        |
   | https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |   41 MB |     false |      false |
   7 |      91624 |           91624 |        | 1
   | https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
   7 |      91624 |           91624 |        |
   +--------------------------+------------------+---------+---------+-----------+-----------+-----------
   +------------+-------------------+--------+
   ```

이 예에서는 etcd 멤버의 데이터베이스 크기가 시작 크기인 104MB와 달리 현재 41MB임을 보여줍니다.

e. 다음 단계를 반복하여 다른 etcd 멤버에 연결하고 조각 모음을 수행합니다. 항상 리더의 조각 모음을 마지막으로 수행합니다.
etcd pod가 복구될 수 있도록 조각 모음 작업에서 1분 이상 기다립니다. etcd pod가 복구될 때까지 etcd 멤버는 응답하지 않습니다.

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

a. **NOSPACE** 경고가 있는지 확인합니다.

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

출력 예

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

b. 경고를 지웁니다.

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

3.4. ETCD에 대한 튜닝 매개변수 설정

제어 평면 하드웨어 속도를 **"표준"**, **"느림"** 또는 기본값인 **""** 으로 설정할 수 있습니다.

기본 설정에서는 시스템이 사용할 속도를 결정합니다. 이 값을 사용하면 시스템이 이전 버전의 값을 선택할 수 있으므로 해당 기능이 없는 버전에서 업그레이드할 수 있습니다.

다른 값 중 하나를 선택하면 기본값이 재정의됩니다. 시간 초과나 하트비트 누락으로 인해 리더 선거가 자주 발생하는 경우, 시스템이 **""** 또는 **"표준"** 으로 설정되어 있다면 하드웨어 속도를 **"느리게"** 로 설정하여 시스템이 지연 시간 증가에 더 잘 견딜 수 있도록 하세요.

3.4.1. 하드웨어 속도 허용 범위 변경

etcd의 하드웨어 속도 허용 범위를 변경하려면 다음 단계를 완료하세요.

프로세스

1. 다음 명령을 입력하여 현재 값을 확인하세요.

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

출력 예

```
Control Plane Hardware Speed:  <VALUE>
```

참고

출력이 비어 있으면 필드가 설정되지 않았으므로 기본값("")으로 간주해야 합니다.

2. 다음 명령을 입력하여 값을 변경합니다. **<value>**를 유효한 값 중 하나인 **""** , **"Standard"** 또는 **"Slower"** 로 바꾸세요.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"controlPlaneHardwareSpeed": "<value>"}}'
```

다음 표는 각 프로필에 대한 하트비트 간격과 리더 선출 시간 초과를 나타냅니다. 이러한 값은 변경될 수 있습니다.

프로필	ETCD_HEARTBEAT_INTERVAL	ETCD_LEADER_ELECTION_TIMEOUT
""	플랫폼에 따라 다름	플랫폼에 따라 다름
Standard	100	1000
더 느리게	500	2500

3. 출력을 확인합니다.

출력 예

```
etcd.operator.openshift.io/cluster patched
```

유효한 값 이외의 값을 입력하면 오류 출력이 표시됩니다. 예를 들어, 값으로 **"Faster"**를 입력한 경우 출력은 다음과 같습니다.

출력 예

```
The Etcd "cluster" is invalid: spec.controlPlaneHardwareSpeed: Unsupported value: "Faster": supported values: "", "Standard", "Slower"
```

4. 다음 명령을 입력하여 값이 변경되었는지 확인하세요.

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

출력 예

```
Control Plane Hardware Speed:  ""
```

5. etcd 포드가 롤아웃될 때까지 기다리세요.

```
$ oc get pods -n openshift-etcd -w
```

다음 출력은 master-0에 대한 예상 항목을 보여줍니다. 계속하기 전에 모든 마스터의 상태가 **4/4 Running** 으로 표시될 때까지 기다리세요.

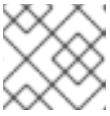
출력 예

```
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0      0/1     Pending     0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0      0/1     Pending     0          0s
```

```
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     ContainerCreating   0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     ContainerCreating   0          1s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          1/1     Running             0          2s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     Completed           0          34s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     Completed           0          36s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     Completed           0          36s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0           0/1     Running             0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 0/4     Pending             0          0s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 0/4     Init:1/3            0          1s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 0/4     Init:2/3            0          2s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 0/4     PodInitializing     0          3s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 3/4     Running             0          4s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0           1/1     Running             0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 3/4     Running             0          20s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                 4/4     Running             0          20s
```

6. 다음 명령을 입력하여 값을 검토하세요.

```
$ oc describe -n openshift-etcd pod/<ETCD_PODNAME> | grep -e HEARTBEAT_INTERVAL
-e ELECTION_TIMEOUT
```



참고

이러한 값은 기본값에서 변경되지 않을 수 있습니다.

추가 리소스

- FeatureGate 이해

3.5. ETCD에 대한 데이터베이스 크기 늘리기

각 etcd 인스턴스에 대해 기비바이트(GiB) 단위로 디스크 할당량을 설정할 수 있습니다. etcd 인스턴스에 디스크 할당량을 설정하는 경우 8~32 사이의 정수 값을 지정할 수 있습니다. 기본값은 0입니다. 증가하는 값만 지정할 수 있습니다.

디스크 **공간 부족** 경고가 발생하면 디스크 할당량을 늘리는 것이 좋습니다. 이 경고는 자동 압축 및 조각 모음에도 불구하고 클러스터가 너무 커서 etcd에 맞지 않음을 나타냅니다. 이 경고가 표시되면 디스크 할당량을 즉시 늘려야 합니다. etcd에서 공간이 부족해지면 쓰기가 실패하기 때문입니다.

디스크 할당량을 늘려야 할 또 다른 상황은 **과도한 데이터베이스 증가** 경고가 발생하는 경우입니다. 이 경고는 다음 4시간 동안 데이터베이스가 너무 커질 수 있다는 경고입니다. 이런 경우 디스크 할당량을 늘리는 것을 고려해 보세요. 그러면 결국 **공간 부족** 경고가 발생하고 쓰기가 실패할 가능성이 있습니다.

디스크 할당량을 늘리면 지정한 디스크 공간이 즉시 예약되지 않습니다. 대신 필요한 경우 etcd를 해당 크기로 확장할 수 있습니다. etcd가 디스크 할당량에 지정한 값보다 큰 전용 디스크에서 실행 중인지 확인하세요.

대규모 etcd 데이터베이스의 경우 제어 평면 노드에는 추가 메모리와 스토리지가 필요합니다. API 서버 캐시를 고려해야 하므로 필요한 최소 메모리는 etcd 데이터베이스의 구성된 크기의 최소 3배입니다.

중요

etcd의 데이터베이스 크기를 늘리는 것은 기술 미리 보기 기능에만 해당됩니다. 기술 미리 보기 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 Technology Preview 기능의 지원 범위에 대한 자세한 내용은 다음 링크를 참조하세요.

- 기술 프리뷰 기능 지원 범위

3.5.1. etcd 데이터베이스 크기 변경

etcd의 데이터베이스 크기를 변경하려면 다음 단계를 완료하세요.

프로세스

1. 다음 명령을 입력하여 각 etcd 인스턴스의 디스크 할당량의 현재 값을 확인하세요.

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

출력 예

```
Backend Quota Gi B: <value>
```

2. 다음 명령을 입력하여 디스크 할당량 값을 변경합니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": <value>}}'
```

출력 예

```
etcd.operator.openshift.io/cluster patched
```

검증

1. 다음 명령을 입력하여 디스크 할당량의 새 값이 설정되었는지 확인하세요.

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

etcd Operator는 새 값으로 etcd 인스턴스를 자동으로 롤아웃합니다.

2. 다음 명령을 입력하여 etcd 포드가 작동 중인지 확인하세요.

```
$ oc get pods -n openshift-etcd
```

다음 출력은 예상 항목을 보여줍니다.

출력 예

```
NAME                                          READY   STATUS     RESTARTS   AGE
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-0       4/4     Running    0          39m
```

```
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-1                 4/4     Running     0          37m
etcd-ci-ln-b6kfsw2-72292-mzwbq-master-2                 4/4     Running     0          41m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-0           1/1     Running     0          51m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-1           1/1     Running     0          49m
etcd-guard-ci-ln-b6kfsw2-72292-mzwbq-master-2           1/1     Running     0          54m
installer-5-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          51m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-0          0/1     Completed   0          46m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          44m
installer-7-ci-ln-b6kfsw2-72292-mzwbq-master-2          0/1     Completed   0          49m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-0          0/1     Completed   0          40m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-1          0/1     Completed   0          38m
installer-8-ci-ln-b6kfsw2-72292-mzwbq-master-2          0/1     Completed   0          42m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-0    0/1     Completed   0          43m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-1    0/1     Completed   0          43m
revision-pruner-7-ci-ln-b6kfsw2-72292-mzwbq-master-2    0/1     Completed   0          43m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-0    0/1     Completed   0          42m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-1    0/1     Completed   0          42m
revision-pruner-8-ci-ln-b6kfsw2-72292-mzwbq-master-2    0/1     Completed   0          42m
```

3. 다음 명령을 입력하여 etcd Pod의 디스크 할당량 값이 업데이트되었는지 확인하세요.

```
$ oc describe -n openshift-etcd pod/<etcd_podname> | grep
"ETCD_QUOTA_BACKEND_BYTES"
```

해당 값은 기본값 **8** 에서 변경되지 않았을 수 있습니다.

출력 예

```
ETCD_QUOTA_BACKEND_BYTES:                8589934592
```

참고

설정한 값은 GiB 단위의 정수이지만, 출력에 표시되는 값은 바이트로 변환됩니다.

3.5.2. 문제 해결

etcd의 데이터베이스 크기를 늘리려고 할 때 문제가 발생하면 다음 문제 해결 단계가 도움이 될 수 있습니다.

3.5.2.1. 값이 너무 작습니다

지정한 값이 **8** 보다 작으면 다음과 같은 오류 메시지가 표시됩니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 5}}'
```

오류 메시지의 예

```
The Etcd "cluster" is invalid:
* spec.backendQuotaGiB: Invalid value: 5: spec.backendQuotaGiB in body should be greater than or
equal to 8
* spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased
```

이 문제를 해결하려면 **8** ~ **32** 사이의 정수를 지정하세요.

3.5.2.2. 값이 너무 큽니다

지정한 값이 **32** 보다 큰 경우 다음과 같은 오류 메시지가 표시됩니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 64}}'
```

오류 메시지의 예

```
The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: 64: spec.backendQuotaGiB in body should be less than or equal to 32
```

이 문제를 해결하려면 **8** ~ **32** 사이의 정수를 지정하세요.

3.5.2.3. 가치가 감소하고 있습니다

값이 **8** ~ **32** 사이의 유효한 값으로 설정된 경우 값을 줄일 수 없습니다. 그렇지 않으면 오류 메시지가 표시됩니다.

1. 다음 명령을 입력하여 현재 값을 확인하세요.

   ```
   $ oc describe etcd/cluster | grep "Backend Quota"
   ```

   출력 예

   ```
   Backend Quota Gi B: 10
   ```

2. 다음 명령을 입력하여 디스크 할당량 값을 줄이세요.

   ```
   $ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 8}}'
   ```

   오류 메시지의 예

   ```
   The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased
   ```

3. 이 문제를 해결하려면 **10** 보다 큰 정수를 지정하세요.

4장. ETCD 데이터 백업 및 복원

4.1. ETCD 데이터 백업 및 복원

OpenShift Container Platform의 키-값 저장소인 etcd는 모든 리소스 객체의 상태를 유지합니다.

클러스터의 etcd 데이터를 정기적으로 백업하고 안전한 위치(이상적으로는 OpenShift Container Platform 환경 외부)에 저장하세요. 설치 후 24 시간 내에 발생하는 첫 번째 인증서 교체가 완료되기 전까지 etcd 백업을 수행하지 마십시오. 인증서 교체가 완료되기 전에 실행하면 백업에 만료된 인증서가 포함됩니다. etcd 스냅샷은 I/O 비용이 높기 때문에 사용량이 많지 않은 시간에 etcd 백업을 수행하는 것이 좋습니다.

클러스터를 업데이트하기 전에 etcd 백업을 꼭 하세요. 업데이트하기 전에 백업을 하는 것이 중요한 이유는 클러스터를 복원할 때 동일한 z-stream 릴리스에서 가져온 etcd 백업을 사용해야 하기 때문입니다. 예를 들어, OpenShift Container Platform 4.17.5 클러스터는 4.17.5에서 가져온 etcd 백업을 사용해야 합니다.

중요

제어 평면 호스트에서 백업 스크립트를 한 번 호출하여 클러스터의 etcd 데이터를 백업합니다. 클러스터의 각 컨트롤 플레인 호스트마다 백업을 수행하지 마십시오.

etcd 백업 후 이전 클러스터 상태로 복원 할 수 있습니다.

4.1.1. etcd 데이터 백업

다음 단계에 따라 etcd 스냅샷을 작성하고 정적 pod의 리소스를 백업하여 etcd 데이터를 백업합니다. 이 백업을 저장하여 etcd를 복원해야하는 경우 나중에 사용할 수 있습니다.

중요

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

사전 요구 사항

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

  작은 정보

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

프로세스

1. 제어 평면 노드의 루트로 디버그 세션을 시작합니다.

   ```
   $ oc debug --as-root node/<node_name>
   ```

2. 디버그 셸에서 루트 디렉토리를 **/host** 로 변경하세요.

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

3. 클러스터 전체 프록시가 활성화된 경우 다음 명령을 실행하여 **NO_PROXY** , **HTTP_PROXY** 및 **HTTPS_PROXY** 환경 변수를 내보냅니다.

```
$ export HTTP_PROXY=http://<your_proxy.example.com>:8080
```

```
$ export HTTPS_PROXY=https://<your_proxy.example.com>:8080
```

```
$ export NO_PROXY=<example.com>
```

4. 디버그 셸에서 **cluster-backup.sh** 스크립트를 실행하고 백업을 저장할 위치를 전달합니다.

작은 정보

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

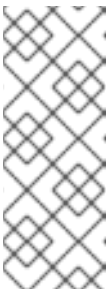
```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

스크립트 출력 예

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

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

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



참고

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

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

추가 리소스

- 비정상적인 etcd 클러스터 복구

4.1.2. 자동화된 etcd 백업 생성

etcd의 자동화된 백업 기능은 반복 및 단일 백업을 모두 지원합니다. 반복 백업은 작업이 트리거될 때마다 단일 백업을 시작하는 cron 작업을 생성합니다.

중요

etcd 백업 자동화는 기술 프리뷰 기능 전용입니다. 기술 미리 보기 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 Technology Preview 기능의 지원 범위에 대한 자세한 내용은 다음 링크를 참조하세요.

- 기술 프리뷰 기능 지원 범위

다음 단계에 따라 etcd에 대한 자동 백업을 활성화합니다.

주의

클러스터에 **TechPreviewNoUpgrade** 기능 세트를 활성화하면 마이너 버전 업데이트가 수행되지 않습니다. **TechPreviewNoUpgrade** 기능 세트를 비활성화할 수 없습니다. 프로덕션 클러스터에서 이 기능 세트를 활성화하지 마십시오.

사전 요구 사항

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

프로세스

1. 다음 콘텐츠를 사용하여 **enable-tech-preview-no-upgrade.yaml** 이라는 **FeatureGate** CR(사용자 정의 리소스) 파일을 생성합니다.

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

2. CR을 적용하고 자동화된 백업을 활성화합니다.

```
$ oc apply -f enable-tech-preview-no-upgrade.yaml
```

3. 관련 API를 활성화하는 데 시간이 걸립니다. 다음 명령을 실행하여 CRD(사용자 정의 리소스 정의) 생성을 확인합니다.

```
$ oc get crd | grep backup
```

출력 예

```
backups.config.openshift.io 2023-10-25T13:32:43Z
etcdbackups.operator.openshift.io 2023-10-25T13:32:04Z
```

4.1.2.1. 단일 자동화된 etcd 백업 생성

다음 단계에 따라 CR(사용자 정의 리소스)을 생성하고 적용하여 단일 etcd 백업을 생성합니다.

사전 요구 사항

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

프로세스

- 동적으로 프로비저닝된 스토리지를 사용할 수 있는 경우 다음 단계를 완료하여 단일 자동화된 etcd 백업을 만듭니다.
  a. 다음 예와 같은 콘텐츠를 사용하여 **etcd-backup-pvc.yaml** 이라는 PVC(영구 볼륨 클레임)를 생성합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
```

```
    requests:
      storage: 200Gi 1
  volumeMode: Filesystem
```

1 1 PVC에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

b. 다음 명령을 실행하여 PVC를 적용합니다.

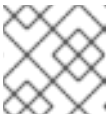
```
$ oc apply -f etcd-backup-pvc.yaml
```

c. 다음 명령을 실행하여 PVC 생성을 확인합니다.

```
$ oc get pvc
```

출력 예

```
NAME              STATUS    VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
etcd-backup-pvc   Bound                                      51s
```



참고

동적 PVC는 마운트될 때까지 **Pending** 상태로 유지됩니다.

d. 다음 예와 같은 콘텐츠를 사용하여 **etcd-single-backup.yaml** 이라는 CR 파일을 만듭니다.

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc 1
```

1 백업을 저장할 PVC의 이름입니다. 환경에 따라 이 값을 조정합니다.

e. CR을 적용하여 단일 백업을 시작합니다.

```
$ oc apply -f etcd-single-backup.yaml
```

- 동적으로 프로비저닝된 스토리지를 사용할 수 없는 경우 다음 단계를 완료하여 단일 자동화된 etcd 백업을 만듭니다.

a. 다음 콘텐츠를 사용하여 **etcd-backup-local-storage.yaml** 이라는 **StorageClass** CR 파일을 만듭니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: etcd-backup-local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

b. 다음 명령을 실행하여 **StorageClass** CR을 적용합니다.

```
$ oc apply -f etcd-backup-local-storage.yaml
```

c. 다음 예시와 같은 내용으로 **etcd-backup-pv-fs.yaml** 이라는 PV를 만듭니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-backup-pv-fs
spec:
  capacity:
    storage: 100Gi (1)
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: etcd-backup-local-storage
  local:
    path: /mnt
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - <example_master_node> (2)
```

(1) <.> PV에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

(2) 이 값을 이 PV를 연결할 노드로 바꾸세요.

d. 다음 명령을 실행하여 PV 생성을 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS              REASON   AGE
etcd-backup-pv-fs      100Gi      RWO            Retain           Available           etcd-backup-
local-storage           10s
```

e. 다음 예와 같은 콘텐츠를 사용하여 **etcd-backup-pvc.yaml** 이라는 PVC를 만듭니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
```

```
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi 1
```

1 PVC에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

f. 다음 명령을 실행하여 PVC를 적용합니다.

```
$ oc apply -f etcd-backup-pvc.yaml
```

g. 다음 예와 같은 콘텐츠를 사용하여 **etcd-single-backup.yaml** 이라는 CR 파일을 만듭니다.

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc 1
```

1 <.> 백업을 저장할 PVC(영구 볼륨 클레임)의 이름입니다. 환경에 따라 이 값을 조정합니다.

h. CR을 적용하여 단일 백업을 시작합니다.

```
$ oc apply -f etcd-single-backup.yaml
```

4.1.2.2. 반복적인 자동화된 etcd 백업 생성

다음 단계에 따라 etcd의 자동 반복 백업을 생성합니다.

동적으로 프로비저닝된 스토리지를 사용하여 가능한 경우 생성된 etcd 백업 데이터를 안전한 외부 위치에 보관합니다. 동적으로 프로비저닝된 스토리지를 사용할 수 없는 경우 NFS 공유에 백업 데이터를 저장하여 백업 복구에 더 쉽게 액세스할 수 있도록 하는 것이 좋습니다.

사전 요구 사항

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

프로세스

1. 동적으로 프로비저닝된 스토리지를 사용할 수 있는 경우 다음 단계를 완료하여 자동화된 반복 백업을 생성합니다.

   a. 다음 예와 같은 콘텐츠를 사용하여 **etcd-backup-pvc.yaml** 이라는 PVC(영구 볼륨 클레임)를

생성합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 200Gi 1
  volumeMode: Filesystem
  storageClassName: etcd-backup-local-storage
```

1 PVC에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

참고

다음 공급자마다 **accessModes** 및 **storageClassName** 키를 변경해야 합니다.

공급자	accessModes 값	storageClassName value
버전ed-installer-efc_operator-ci 프로필이 있는 AWS	**- ReadWriteMany**	**efs-sc**
Google Cloud	**- ReadWriteMany**	**filestore-csi**
Microsoft Azure	**- ReadWriteMany**	**azurefile-csi**

b. 다음 명령을 실행하여 PVC를 적용합니다.

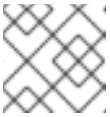
```
$ oc apply -f etcd-backup-pvc.yaml
```

c. 다음 명령을 실행하여 PVC 생성을 확인합니다.

```
$ oc get pvc
```

출력 예

```
NAME           STATUS   VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
etcd-backup-pvc   Bound                                      51s
```



참고

동적 PVC는 마운트될 때까지 **Pending** 상태로 유지됩니다.

2. 동적으로 프로비저닝된 스토리지를 사용할 수 없는 경우 다음 단계를 완료하여 로컬 스토리지 PVC를 생성합니다.

주의

저장된 백업 데이터가 포함된 노드에 대한 액세스 권한을 삭제하거나 손실하면 데이터가 손실될 수 있습니다.

a. 다음 콘텐츠를 사용하여 **etcd-backup-local-storage.yaml** 이라는 **StorageClass** CR 파일을 만듭니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: etcd-backup-local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

b. 다음 명령을 실행하여 **StorageClass** CR을 적용합니다.

```
$ oc apply -f etcd-backup-local-storage.yaml
```

c. 다음 예와 같은 콘텐츠를 사용하여 적용된 **StorageClass** 에서 **etcd-backup-pv-fs.yaml** 이라는 PV를 만듭니다.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-backup-pv-fs
spec:
  capacity:
    storage: 100Gi (1)
  volumeMode: Filesystem
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: etcd-backup-local-storage
  local:
    path: /mnt/
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
```

```
      operator: In
      values:
      - <example_master_node> 2
```

1 <.> PV에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

2 이 값을 이 PV를 연결할 마스터 노드로 바꾸세요.

작은 정보

다음 명령을 실행하여 사용 가능한 노드를 나열합니다.

```
$ oc get nodes
```

d. 다음 명령을 실행하여 PV 생성을 확인합니다.

```
$ oc get pv
```

출력 예

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS            REASON   AGE
etcd-backup-pv-fs   100Gi      RWX            Delete           Available
etcd-backup-local-storage          10s
```

e. 다음 예와 같은 콘텐츠를 사용하여 **etcd-backup-pvc.yaml** 이라는 PVC를 만듭니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
spec:
  accessModes:
  - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi 1
  storageClassName: etcd-backup-local-storage
```

1 PVC에서 사용할 수 있는 스토리지 용량입니다. 요구 사항에 맞게 이 값을 조정합니다.

f. 다음 명령을 실행하여 PVC를 적용합니다.

```
$ oc apply -f etcd-backup-pvc.yaml
```

3. **etcd-recurring-backups.yaml** 이라는 CRD(사용자 정의 리소스 정의) 파일을 생성합니다. 생성된 CRD의 내용은 자동 백업의 일정 및 보존 유형을 정의합니다.

- 15개의 보존된 백업이 있는 보존 **Number** 의 기본 보존 유형은 다음 예와 같은 내용을 사용합니다.

```
apiVersion: config.openshift.io/v1alpha1
kind: Backup
metadata:
  name: etcd-recurring-backup
spec:
  etcd:
    schedule: "20 4 * * *" 1
    timeZone: "UTC"
    pvcName: etcd-backup-pvc
```

1 반복 백업의 **CronTab** 스케줄입니다. 요구 사항에 맞게 이 값을 조정합니다.

- 최대 백업 수에 따라 보존을 사용하려면 다음 키-값 쌍을 **etcd** 키에 추가합니다.

```
spec:
  etcd:
    retentionPolicy:
      retentionType: RetentionNumber 1
      retentionNumber:
        maxNumberOfBackups: 5 2
```

1 <.> 보존 유형. 지정되지 않은 경우 기본값은 **RetentionNumber** 입니다.

2 보존할 백업의 최대 수입니다. 요구 사항에 맞게 이 값을 조정합니다. 지정되지 않은 경우 기본값은 15 백업입니다.

주의

알려진 문제로 인해 보존된 백업 수가 구성된 값보다 큰 문제가 됩니다.

- 백업 파일 크기에 따라 보존하려면 다음을 사용합니다.

```
spec:
  etcd:
    retentionPolicy:
      retentionType: RetentionSize
      retentionSize:
        maxSizeOfBackupsGb: 20 1
```

1 보관된 백업의 최대 파일 크기(GB)입니다. 요구 사항에 맞게 이 값을 조정합니다. 지정되지 않은 경우 기본값은 10GB입니다.

주의

알려진 문제로 인해 보존 백업의 최대 크기가 구성된 값보다 최대 10GB 가 됩니다.

4. 다음 명령을 실행하여 CRD에서 정의한 cron 작업을 생성합니다.

```
$ oc create -f etcd-recurring-backup.yaml
```

5. 생성된 cron 작업을 찾으려면 다음 명령을 실행합니다.

```
$ oc get cronjob -n openshift-etcd
```

4.2. 비정상적인 ETCD 멤버 교체

단일의 비정상적인 etcd 멤버를 교체하는 프로세스는 머신이 실행 중이 아니거나 노드가 준비되지 않았기 때문에 etcd 멤버가 비정상적인지, 아니면 etcd Pod가 크래시 루프를 겪고 있는지에 따라 달라집니다.

참고

대부분의 제어 평면 호스트를 잃은 경우 이 절차 대신 재해 복구 절차에 따라 이전 클러스터 상태로 복원하세요 .

교체된 멤버에서 컨트롤 플레인 인증서가 유효하지 않은 경우 이 프로세스 대신 만료된 컨트롤 플레인 인증서 복구 절차를 따라야합니다.

컨트롤 플레인 노드가 유실되고 새 노드가 생성되는 경우 etcd 클러스터 Operator는 새 TLS 인증서 생성 및 노드를 etcd 멤버로 추가하는 프로세스를 진행합니다.

4.2.1. 비정상 etcd 멤버 식별

클러스터에 비정상적인 etcd 멤버가 있는지 여부를 확인할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- etcd 백업이 수행되었습니다. 자세한 내용은 "etcd 데이터 백업"을 참조하세요.

프로세스

1. 다음 명령을 사용하여 **EtcdMembersAvailable** 상태의 상태 조건을 확인하십시오.

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

2. 출력을 확인합니다.

```
2 of 3 members are available, ip-10-0-131-183.ec2.internal is unhealthy
```

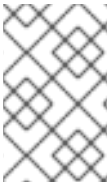
이 출력 예는 **ip-10-0-131-183.ec2.internal** etcd 멤버가 비정상임을 보여줍니다.

4.2.2. 비정상적인 etcd 멤버의 상태 확인

비정상적인 etcd 멤버를 교체하는 프로세스는 etcd가 다음의 어떤 상태에 있는지에 따라 달라집니다.

- 컴퓨터가 실행 중이 아니거나 노드가 준비되지 않았습니다.
- etcd pod가 크래시 루프 상태에 있습니다.

다음 프로세스에서는 etcd 멤버가 어떤 상태에 있는지를 확인합니다. 이를 통해 비정상 etcd 멤버를 대체하기 위해 수행해야하는 단계를 확인할 수 있습니다.



참고

시스템이 실행되고 있지 않거나 노드가 준비되지 않았지만 곧 정상 상태로 돌아올 것으로 예상되는 경우 etcd 멤버를 교체하기 위한 절차를 수행할 필요가 없습니다. etcd 클러스터 Operator는 머신 또는 노드가 정상 상태로 돌아 오면 자동으로 동기화됩니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- 비정상적인 etcd 멤버를 식별하고 있습니다.

프로세스

1. **시스템이 실행되고 있지 않은지**를 확인합니다.

```
$ oc get machines -A -ojsonpath='{range .items[*]}{@.status.nodeRef.name}{"\t"}
{@.status.providerStatus.instanceState}{"\n"}' | grep -v running
```

출력 예

```
ip-10-0-131-183.ec2.internal  stopped 1
```

1 이 출력은 노드와 노드 시스템의 상태를 나열합니다. 상태가 **running**이 아닌 경우 **시스템은 실행되지 않습니다**.

시스템이 실행되고 있지 않은경우, *시스템이 실행되고 있지 않거나 노드가 준비되지 않은 비정상적인 etcd 멤버 교체* 프로세스를 수행하십시오.

2. **노드가 준비되지 않았는지** 확인합니다.
다음 조건 중 하나에 해당하면 **노드가 준비되지 않은 것**입니다.

- 시스템이 실행중인 경우 노드에 액세스할 수 있는지 확인하십시오.

```
$ oc get nodes -o jsonpath='{range .items[*]}{"\n"}{.metadata.name}{"\t"}{range
.spec.taints[*]}{.key}{" "}' | grep unreachable
```

출력 예

```
ip-10-0-131-183.ec2.internal node-role.kubernetes.io/master
node.kubernetes.io/unreachable node.kubernetes.io/unreachable 1
```

1 **unreachable** 상태의 노드가 나열되면 **노드가 준비되지 않은 것** 입니다.

- 노드에 여전히 액세스할 수 있는 경우 노드가 **NotReady**로 나열되어 있는지 확인하십시오.

```
$ oc get nodes -l node-role.kubernetes.io/master | grep "NotReady"
```

출력 예

```
ip-10-0-131-183.ec2.internal   NotReady   master   122m   v1.32.3 1
```

1 노드가 **NotReady**로 표시되면 **노드가 준비되지 않은 것**입니다.

노드가 준비되지 않은 경우 *시스템이 실행되고 있지 않거나 노드가 준비되지 않은 비정상적인 etcd 멤버 교체* 프로세스를 수행하십시오.

3. **etcd pod가 크래시 루프 상태**인지 확인합니다.
시스템이 실행되고 있고 노드가 준비된 경우 etcd pod가 크래시 루프 상태인지 확인하십시오.

a. 모든 제어 평면 노드가 **준비** 로 나열되어 있는지 확인하세요.

```
$ oc get nodes -l node-role.kubernetes.io/master
```

출력 예

```
NAME                         STATUS   ROLES    AGE     VERSION
ip-10-0-131-183.ec2.internal   Ready    master   6h13m   v1.32.3
ip-10-0-164-97.ec2.internal    Ready    master   6h13m   v1.32.3
ip-10-0-154-204.ec2.internal   Ready    master   6h13m   v1.32.3
```

b. etcd pod의 상태가 **Error** 또는 **CrashloopBackoff**인지 확인하십시오.

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

출력 예

```
etcd-ip-10-0-131-183.ec2.internal          2/3     Error       7          6h9m 1
etcd-ip-10-0-164-97.ec2.internal           3/3     Running     0          6h6m
etcd-ip-10-0-154-204.ec2.internal          3/3     Running     0          6h6m
```

1 이 pod의 상태는 **Error**이므로 **etcd pod는 크래시 루프 상태**입니다.

etcd pod가 크래시 루프상태인 경우*etcd pod가 크래시 루프 상태인 비정상적인 etcd 멤버 교체* 프로세스를 수행하십시오.

4.2.3. 비정상적인 etcd 멤버 교체

비정상적인 etcd 멤버의 상태에 따라 다음 절차 중 하나를 사용합니다.

- 시스템이 실행되고 있지 않거나 노드가 준비되지 않은 비정상적인 etcd 멤버 교체
- 비정상 클러스터에 기본 컨트롤 플레인 노드 설치
- etcd pod가 크래시 루프 상태인 비정상적인 etcd 멤버 교체
- 비정상적으로 중지된 베어메탈 etcd 멤버 교체

4.2.3.1. 시스템이 실행되고 있지 않거나 노드가 준비되지 않은 비정상적인 etcd 멤버 교체

다음에서는 시스템이 실행되고 있지 않거나 노드가 준비되지 않은 경우의 비정상적인 etcd 멤버를 교체하는 프로세스에 대해 자세히 설명합니다.

참고

클러스터가 컨트롤 플레인 머신 세트를 사용하는 경우 etcd 복구 절차에 대한 "컨트롤 플레인 머신 세트 문제 해결"의 "성능이 저하된 etcd Operator 복구"를 참조하세요.

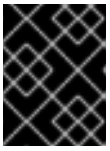
사전 요구 사항

- 비정상적인 etcd 멤버를 식별했습니다.
- 시스템이 실행되고 있지 않거나 노드가 준비되지 않았음을 확인했습니다.

중요

다른 제어 평면 노드의 전원을 끄면 기다려야 합니다. 제어 평면 노드는 건강에 해로운 etcd 멤버의 교체가 완료될 때까지 전원이 꺼져 있어야 합니다.

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



중요

이 절차를 수행하기 전에 etcd 백업을 수행하여 문제가 발생할 경우 클러스터를 복원할 수 있도록 하세요.

프로세스

1. 비정상적인 멤버를 제거합니다.
   a. 영향을 받는 노드에 없는 Pod를 선택하세요.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

```
etcd-ip-10-0-131-183.ec2.internal             3/3     Running     0          123m
```

```
etcd-ip-10-0-164-97.ec2.internal              3/3     Running     0          123m
etcd-ip-10-0-154-204.ec2.internal             3/3     Running     0          124m
```

b. 실행중인 etcd 컨테이너에 연결하고 영향을 받는 노드에 없는 pod 이름을 전달합니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
```

c. 멤버 목록을 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+----------------------------+--------------------------+---------------
-----------+
|        ID        | STATUS  |            NAME            |        PEER ADDRS        |      CLIENT
ADDRS       |
+------------------+---------+----------------------------+--------------------------+---------------
-----------+
| 6fc1e7c9db35841d | started | ip-10-0-131-183.ec2.internal | https://10.0.131.183:2380 |
https://10.0.131.183:2379 |
| 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+------------------+---------+----------------------------+--------------------------+---------------
-----------+
```

이러한 값은 프로세스의 뒷부분에서 필요하므로 비정상 etcd 멤버의 ID와 이름을 기록해 두십시오. **$ etcdctl endpoint health** 명령은 교체 절차가 완료되고 새 멤버가 추가될 때까지 제거된 멤버를 나열합니다.

d. **etcdctl member remove** 명령에 ID를 지정하여 비정상적인 etcd 멤버를 제거합니다.

```
sh-4.2# etcdctl member remove 6fc1e7c9db35841d
```

출력 예

```
Member 6fc1e7c9db35841d removed from cluster ead669ce1fbfb346
```

e. 멤버 목록을 다시 표시하고 멤버가 제거되었는지 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+----------------------------+--------------------------+---------------
-----------+
|        ID        | STATUS  |            NAME            |        PEER ADDRS        |      CLIENT
ADDRS       |
+------------------+---------+----------------------------+--------------------------+---------------
```

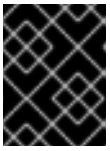
```
----------+
| 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+-----------------+---------+----------------------------+------------------------+---------------
----------+
```

이제 노드 쉘을 종료할 수 있습니다.

2. 다음 명령을 입력하여 쿼럼 가드를 끕니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

이 명령을 사용하면 비밀을 성공적으로 다시 생성하고 정적 포드를 롤아웃할 수 있습니다.



중요

쿼럼 가드를 끄면 나머지 etcd 인스턴스가 구성 변경을 반영하기 위해 재부팅되는 동안 잠시 동안 클러스터에 접근하지 못할 수 있습니다.

참고

두 개의 멤버로 실행할 경우 etcd는 추가 멤버 실패를 허용할 수 없습니다. 나머지 멤버를 다시 시작하면 쿼럼이 끊어지고 클러스터가 다운타임됩니다. 쿼럼 가드는 구성 변경으로 인해 etcd가 다시 시작되어 가동 중지되는 것을 방지하므로 이 절차를 완료하려면 쿼럼 가드를 비활성화해야 합니다.

3. 다음 명령을 실행하여 영향을 받은 노드를 삭제합니다.

```
$ oc delete node <node_name>
```

명령 예

```
$ oc delete node ip-10-0-131-183.ec2.internal
```

4. 삭제된 비정상 etcd 멤버의 이전 암호를 제거합니다.

   a. 삭제된 비정상 etcd 멤버의 시크릿(secrets)을 나열합니다.

```
$ oc get secrets -n openshift-etcd | grep ip-10-0-131-183.ec2.internal 1
```

1 이 프로세스의 앞부분에서 기록한 비정상 etcd 멤버의 이름을 전달합니다.

다음 출력에 표시된대로 피어, 서빙 및 메트릭 시크릿이 있습니다.

출력 예

```
etcd-peer-ip-10-0-131-183.ec2.internal             kubernetes.io/tls                     2      47m
etcd-serving-ip-10-0-131-183.ec2.internal          kubernetes.io/tls                     2      47m
etcd-serving-metrics-ip-10-0-131-183.ec2.internal  kubernetes.io/tls                     2
```

```
47m
```

b. 제거된 비정상 etcd 멤버의 시크릿을 삭제합니다.

i. 피어 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-peer-ip-10-0-131-183.ec2.internal
```

ii. 서빙 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-serving-ip-10-0-131-183.ec2.internal
```

iii. 메트릭 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-serving-metrics-ip-10-0-131-183.ec2.internal
```

5. 다음 명령을 입력하여 제어 평면 머신 세트가 있는지 확인하세요.

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

- 제어 평면 머신 세트가 존재하는 경우 제어 평면 머신을 삭제하고 다시 생성합니다. 이 시스템을 다시 만든 후에는 새 버전이 강제 실행되고 etcd는 자동으로 확장됩니다. 자세한 내용은 "실행 중이 아닌 머신이나 준비되지 않은 노드가 있는 비정상적인 etcd 멤버 교체"를 참조하세요.
  설치 프로그램에서 제공한 인프라를 실행 중이거나 Machine API를 사용하여 컴퓨터를 만든 경우 다음 단계를 수행합니다. 그렇지 않은 경우 원래 제어 평면을 만드는 데 사용한 것과 동일한 방법을 사용하여 새 제어 평면을 만들어야 합니다.

a. 비정상 멤버의 컴퓨터를 가져옵니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

```
NAME                                PHASE   TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-0          Running m4.xlarge   us-east-1   us-east-1a   3h37m   ip-10-0-131-183.ec2.internal   aws:///us-east-1a/i-0ec2782f8287dfb7e   stopped (1)
clustername-8qw5l-master-1          Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2          Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running m4.large    us-east-1   us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running m4.large    us-east-1   us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-0cb45ac45a166173b   running
```

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

1 이는 비정상 노드의 컨트롤 플레인 시스템 **ip-10-0-131-183.ec2.internal**입니다.

b. 비정상 멤버의 시스템을 삭제합니다.

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

1 비정상 노드의 컨트롤 플레인 시스템의 이름을 지정합니다.

건강에 문제가 있는 멤버의 컴퓨터를 삭제하면 자동으로 새 컴퓨터가 프로비저닝됩니다.

c. 새로운 머신이 생성되었는지 확인하세요.

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

출력 예

```
NAME                                PHASE         TYPE        REGION      ZONE         AGE     NODE                           PROVIDERID                              STATE
clustername-8qw5l-master-1          Running       m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2          Running       m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-master-3          Provisioning  m4.xlarge   us-east-1   us-east-1a   85s     ip-10-0-133-53.ec2.internal    aws:///us-east-1a/i-015b0888fe17bc2c8   running 1
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large  us-east-1  us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large  us-east-1  us-east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large  us-east-1  us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-06861c00007751b0a  running
```

1 새 시스템 **clustername-8qw5l-master-3**이 생성되고 단계가 **Provisioning**에서 **Running**으로 변경되면 시스템이 준비 상태가 됩니다.

새 시스템을 만드는 데 몇 분이 소요될 수 있습니다. etcd 클러스터 운영자는 머신이나 노드가 정상 상태로 돌아오면 자동으로 동기화합니다.

참고

사용 중인 머신 세트에 대한 서브넷 ID를 확인하여 올바른 가용성 영역에 속하는지 확인하세요.

- 제어 평면 머신 세트가 존재하지 않으면 제어 평면 머신을 삭제하고 다시 생성합니다. 이 시스템을 다시 만든 후에는 새 버전이 강제 실행되고 etcd는 자동으로 확장됩니다.
  설치 프로그램에서 제공한 인프라를 실행 중이거나 Machine API를 사용하여 컴퓨터를 만든 경우 다음 단계를 수행합니다. 그렇지 않은 경우 원래 제어 평면을 만드는 데 사용한 것과 동일한 방법을 사용하여 새 제어 평면을 만들어야 합니다.

  a. 비정상 멤버의 컴퓨터를 가져옵니다.
  클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

  출력 예

  ```
  NAME                                PHASE   TYPE        REGION      ZONE         AGE
  NODE                           PROVIDERID                          STATE
  clustername-8qw5l-master-0          Running m4.xlarge   us-east-1   us-east-1a
  3h37m   ip-10-0-131-183.ec2.internal   aws:///us-east-1a/i-0ec2782f8287dfb7e
  stopped (1)
  clustername-8qw5l-master-1          Running m4.xlarge   us-east-1   us-east-1b
  3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631
  running
  clustername-8qw5l-master-2          Running m4.xlarge   us-east-1   us-east-1c
  3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba
  running
  clustername-8qw5l-worker-us-east-1a-wbtgd  Running m4.large   us-east-1   us-
  east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-
  010ef6279b4662ced   running
  clustername-8qw5l-worker-us-east-1b-lrdxb   Running m4.large   us-east-1   us-
  east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-
  0cb45ac45a166173b   running
  clustername-8qw5l-worker-us-east-1c-pkg26   Running m4.large   us-east-1   us-
  east-1c   3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-
  06861c00007751b0a   running
  ```

  (1) 이는 비정상 노드의 컨트롤 플레인 시스템 **ip-10-0-131-183.ec2.internal**입니다.

  b. 시스템 설정을 파일 시스템의 파일에 저장합니다.

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

  (1) 비정상 노드의 컨트롤 플레인 시스템의 이름을 지정합니다.

  c. 이전 단계에서 만든 **new-master-machine.yaml** 파일을 편집하여 새 이름을 할당하고 불필요한 필드를 제거합니다.

  i. 전체 **status** 섹션을 삭제합니다.

  ```
  status:
  ```

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

ii. **metadata.name** 필드를 새 이름으로 변경합니다.
이전 기계와 동일한 기본 이름을 유지하고, 마지막 번호를 다음으로 사용 가능한 번호로 변경합니다. 이 예에서 **clustername-8qw5l-master-0**을 **clustername-8qw5l-master-3**으로 변경되어 있습니다.

예를 들면 다음과 같습니다.

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

iii. **spec.providerID** 필드를 삭제합니다.

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

d. 비정상 멤버의 시스템을 삭제합니다.

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

1 비정상 노드의 컨트롤 플레인 시스템의 이름을 지정합니다.

e. 시스템이 삭제되었는지 확인합니다.

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

출력 예

```
NAME                                 PHASE   TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1           Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2           Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-worker-us-east-1a-wbtgd   Running m4.large    us-east-1   us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running m4.large    us-east-1   us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26   Running m4.large    us-east-1   us-east-1c   3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-06861c00007751b0a   running
```

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

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

g. 새로운 머신이 생성되었는지 확인하세요.

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

출력 예

```
NAME                                 PHASE   TYPE        REGION      ZONE         AGE     NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1           Running m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-154-204.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2           Running m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-164-97.ec2.internal    aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-master-3           Provisioning m4.xlarge   us-east-1   us-east-1a   85s     ip-10-0-133-53.ec2.internal    aws:///us-east-1a/i-015b0888fe17bc2c8   running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd   Running m4.large    us-east-1   us-east-1a   3h28m   ip-10-0-129-226.ec2.internal   aws:///us-east-1a/i-010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb   Running m4.large    us-east-1   us-east-1b   3h28m   ip-10-0-144-248.ec2.internal   aws:///us-east-1b/i-0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26   Running m4.large    us-east-1   us-east-1c   3h28m   ip-10-0-170-181.ec2.internal   aws:///us-east-1c/i-06861c00007751b0a   running
```

(1) 새 시스템 **clustername-8qw5l-master-3**이 생성되고 단계가 **Provisioning**에서 **Running**으로 변경되면 시스템이 준비 상태가 됩니다.

새 시스템을 만드는 데 몇 분이 소요될 수 있습니다. etcd 클러스터 운영자는 머신이나 노드가 정상 상태로 돌아오면 자동으로 동기화합니다.

6. 다음 명령을 입력하여 쿼럼 가드를 다시 켭니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

7. 다음 명령을 입력하면 **지원되지 않는 ConfigOverrides** 섹션이 개체에서 제거되었는지 확인할 수 있습니다.

```
$ oc get etcd/cluster -oyaml
```

8. 단일 노드 OpenShift를 사용하는 경우 노드를 다시 시작합니다. 그렇지 않으면 etcd 클러스터 운영자에서 다음 오류가 발생할 수 있습니다.

출력 예

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply your changes to the latest version and try again]
```

검증

1. 모든 etcd pod가 올바르게 실행되고 있는지 확인합니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

```
etcd-ip-10-0-133-53.ec2.internal             3/3     Running     0          7m49s
etcd-ip-10-0-164-97.ec2.internal             3/3     Running     0          123m
etcd-ip-10-0-154-204.ec2.internal            3/3     Running     0          124m
```

이전 명령의 출력에 두 개의 pod만 나열되는 경우 수동으로 etcd 재배포를 강제 수행할 수 있습니다. 클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$( date --rfc-3339=ns )"'"}}' --type=merge (1)
```

(1) **forceRedeploymentReason** 값은 고유해야하므로 타임 스탬프가 추가됩니다.

2. 정확히 세 개의 etcd 멤버가 있는지 확인합니다.

a. 실행중인 etcd 컨테이너에 연결하고 영향을 받는 노드에 없는 pod 이름을 전달합니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
```

b. 멤버 목록을 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+--------------------------+--------------------------+----------
----------+
|        ID        | STATUS  |          NAME            |        PEER ADDRS        |      CLIENT
ADDRS       |
+------------------+---------+--------------------------+--------------------------+----------
----------+
| 5eb0d6b8ca24730c | started |  ip-10-0-133-53.ec2.internal |  https://10.0.133.53:2380 |
https://10.0.133.53:2379 |
| 757b6793e2408b6c | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+------------------+---------+--------------------------+--------------------------+----------
----------+
```

이전 명령의 출력에 세 개 이상의 etcd 멤버가 나열된 경우 원하지 않는 멤버를 신중하게 제거해야 합니다.

> **주의**
>
> 올바른 etcd 멤버를 제거하십시오. etcd 멤버를 제거하면 쿼럼이 손실될 수 있습니다.

추가 리소스

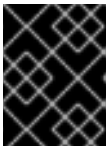
- 성능 저하된 etcd Operator 복구
- 비정상 클러스터에 기본 컨트롤 플레인 노드 설치

4.2.3.2. etcd pod가 크래시 루프 상태인 비정상적인 etcd 멤버 교체

이 단계에서는 etcd pod가 크래시 루프 상태에 있는 경우 비정상 etcd 멤버를 교체하는 방법을 설명합니다.

전제 조건

- 비정상적인 etcd 멤버를 식별했습니다.
- etcd pod가 크래시 루프 상태에 있는것으로 확인되었습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있습니다.
- etcd 백업이 수행되었습니다.



중요

문제가 발생할 경우 클러스터를 복원할 수 있도록 이 프로세스를 수행하기 전에 etcd 백업을 수행해야합니다.

프로세스

1. 크래시 루프 상태에 있는 etcd pod를 중지합니다.

   a. 크래시 루프 상태의 노드를 디버깅합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

   ```
   $ oc debug node/ip-10-0-131-183.ec2.internal 1
   ```

   1 이를 비정상 노드의 이름으로 변경합니다.

   b. 루트 디렉토리를 **/host** 로 변경합니다.

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

   c. kubelet 매니페스트 디렉토리에서 기존 etcd pod 파일을 이동합니다.

   ```
   sh-4.2# mkdir /var/lib/etcd-backup
   ```

   ```
   sh-4.2# mv /etc/kubernetes/manifests/etcd-pod.yaml /var/lib/etcd-backup/
   ```

   d. etcd 데이터 디렉토리를 다른 위치로 이동합니다.

   ```
   sh-4.2# mv /var/lib/etcd/ /tmp
   ```

   이제 노드 쉘을 종료할 수 있습니다.

2. 비정상적인 멤버를 제거합니다.

   a. 영향을 받는 노드에 *없는* pod를 선택합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

   출력 예

   ```
   etcd-ip-10-0-131-183.ec2.internal           2/3     Error     7          6h9m
   etcd-ip-10-0-164-97.ec2.internal            3/3     Running   0          6h6m
   etcd-ip-10-0-154-204.ec2.internal           3/3     Running   0          6h6m
   ```

   b. 실행중인 etcd 컨테이너에 연결하고 영향을 받는 노드에 없는 pod 이름을 전달합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

   ```
   $ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
   ```

   c. 멤버 목록을 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
|        ID        | STATUS  |            NAME            |       PEER ADDRS         |      CLIENT
ADDRS       |
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
| 62bcf33650a7170a | started | ip-10-0-131-183.ec2.internal | https://10.0.131.183:2380 |
https://10.0.131.183:2379 |
| b78e2856655bc2eb | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| d022e10b498760d5 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380
| https://10.0.154.204:2379 |
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
```

이러한 값은 프로세스의 뒷부분에서 필요하므로 비정상 etcd 멤버의 ID와 이름을 기록해 두십시오.

d. **etcdctl member remove** 명령에 ID를 지정하여 비정상적인 etcd 멤버를 제거합니다.

```
sh-4.2# etcdctl member remove 62bcf33650a7170a
```

출력 예

```
Member 62bcf33650a7170a removed from cluster ead669ce1fbfb346
```

e. 멤버 목록을 다시 표시하고 멤버가 제거되었는지 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
|        ID        | STATUS  |            NAME            |       PEER ADDRS         |      CLIENT
ADDRS       |
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
| b78e2856655bc2eb | started |  ip-10-0-164-97.ec2.internal |  https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| d022e10b498760d5 | started | ip-10-0-154-204.ec2.internal | https://10.0.154.204:2380
| https://10.0.154.204:2379 |
+------------------+---------+----------------------------+--------------------------+----------------
-----------+
```

이제 노드 쉘을 종료할 수 있습니다.

3. 다음 명령을 입력하여 쿼럼 가드를 끕니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

이 명령을 사용하면 비밀을 성공적으로 다시 생성하고 정적 포드를 롤아웃할 수 있습니다.

4. 삭제된 비정상 etcd 멤버의 이전 암호를 제거합니다.
   a. 삭제된 비정상 etcd 멤버의 시크릿(secrets)을 나열합니다.

```
$ oc get secrets -n openshift-etcd | grep ip-10-0-131-183.ec2.internal 1
```

1 이 프로세스의 앞부분에서 기록한 비정상 etcd 멤버의 이름을 전달합니다.

다음 출력에 표시된대로 피어, 서빙 및 메트릭 시크릿이 있습니다.

출력 예

```
etcd-peer-ip-10-0-131-183.ec2.internal             kubernetes.io/tls                     2      47m
etcd-serving-ip-10-0-131-183.ec2.internal          kubernetes.io/tls                     2      47m
etcd-serving-metrics-ip-10-0-131-183.ec2.internal  kubernetes.io/tls                     2
47m
```

   b. 제거된 비정상 etcd 멤버의 시크릿을 삭제합니다.
      i. 피어 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-peer-ip-10-0-131-183.ec2.internal
```

      ii. 서빙 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-serving-ip-10-0-131-183.ec2.internal
```

      iii. 메트릭 시크릿을 삭제합니다.

```
$ oc delete secret -n openshift-etcd etcd-serving-metrics-ip-10-0-131-
183.ec2.internal
```

5. etcd를 강제로 재배포합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "single-master-recovery-
'"$( date --rfc-3339=ns )"'"}}' --type=merge 1
```

1 **forceRedeploymentReason** 값은 고유해야하므로 타임 스탬프가 추가됩니다.

etcd 클러스터 Operator가 재배포를 수행하면 모든 마스터 노드에서 etcd pod가 작동하는지 확인합니다.

6. 다음 명령을 입력하여 쿼럼 가드를 다시 켭니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

7. 다음 명령을 입력하면 **지원되지 않는 ConfigOverrides** 섹션이 개체에서 제거되었는지 확인할 수 있습니다.

```
$ oc get etcd/cluster -oyaml
```

8. 단일 노드 OpenShift를 사용하는 경우 노드를 다시 시작합니다. 그렇지 않으면 etcd 클러스터 운영자에서 다음 오류가 발생할 수 있습니다.

출력 예

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply your changes to the latest version and try again]
```

검증

- 새 멤버가 사용 가능하고 정상적인 상태에 있는지 확인합니다.

a. 실행중인 etcd 컨테이너에 다시 연결합니다.
cluster-admin 사용자로 클러스터에 액세스할 수 있는 터미널에서 다음 명령을 실행합니다.

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal
```

b. 모든 멤버가 정상인지 확인합니다.

```
sh-4.2# etcdctl endpoint health
```

출력 예

```
https://10.0.131.183:2379 is healthy: successfully committed proposal: took = 16.671434ms
https://10.0.154.204:2379 is healthy: successfully committed proposal: took = 16.698331ms
https://10.0.164.97:2379 is healthy: successfully committed proposal: took = 16.621645ms
```

4.2.3.3. 머신이 실행 중이 아니거나 노드가 준비되지 않은 비정상적 베어 메탈 etcd 멤버 교체

다음에서는 시스템이 실행되고 있지 않거나 노드가 준비되지 않은 경우의 비정상적인 etcd 멤버를 교체하는 프로세스에 대해 자세히 설명합니다.

설치 프로그램에서 제공한 인프라를 실행 중이거나 Machine API를 사용하여 컴퓨터를 만든 경우 다음 단계를 수행합니다. 그렇지 않은 경우 원래 생성에 사용한 것과 동일한 방법을 사용하여 새 제어 평면 노드를 생성해야 합니다.

사전 요구 사항

- 건강에 해로운 베어 메탈 etcd 멤버를 식별했습니다.
- 시스템이 실행되고 있지 않거나 노드가 준비되지 않았음을 확인했습니다.

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

중요

문제가 발생할 경우 클러스터를 복원할 수 있도록 이 프로세스를 수행하기 전에 etcd 백업을 수행해야합니다.

프로세스

1. 건강에 해로운 멤버를 확인하고 제거하세요.
   a. 영향을 받는 노드에 *없는*pod를 선택합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

```
etcd-openshift-control-plane-0  5/5  Running  11  3h56m  192.168.10.9  openshift-control-plane-0  <none>  <none>
etcd-openshift-control-plane-1  5/5  Running  0   3h54m  192.168.10.10  openshift-control-plane-1  <none>  <none>
etcd-openshift-control-plane-2  5/5  Running  0   3h58m  192.168.10.11  openshift-control-plane-2  <none>  <none>
```

   b. 실행중인 etcd 컨테이너에 연결하고 영향을 받는 노드에 없는 pod 이름을 전달합니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc rsh -n openshift-etcd etcd-openshift-control-plane-0
```

   c. 멤버 목록을 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+--------------------+---------------------------+---------------------------+--------------------+
| ID               | STATUS  | NAME               | PEER ADDRS                | CLIENT ADDRS              | IS LEARNER |
+------------------+---------+--------------------+---------------------------+---------------------------+--------------------+
| 7a8197040a5126c8 | started | openshift-control-plane-2 | https://192.168.10.11:2380/ | https://192.168.10.11:2379/ | false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380/ | https://192.168.10.10:2379/ | false |
| cc3830a72fc357f9 | started | openshift-control-plane-0 | https://192.168.10.9:2380/ | https://192.168.10.9:2379/   | false |
+------------------+---------+--------------------+---------------------------+---------------------------+--------------------+
```

이러한 값은 프로세스의 뒷부분에서 필요하므로 비정상 etcd 멤버의 ID와 이름을 기록해 두십시오. **etcdctl 엔드포인트 상태** 명령은 교체 절차가 완료되고 새 멤버가 추가될 때까지 제거된 멤버를 나열합니다.

d. **etcdctl member remove** 명령에 ID를 지정하여 비정상적인 etcd 멤버를 제거합니다.

주의

올바른 etcd 멤버를 제거하십시오. etcd 멤버를 제거하면 쿼럼이 손실될 수 있습니다.

```
sh-4.2# etcdctl member remove 7a8197040a5126c8
```

출력 예

```
Member 7a8197040a5126c8 removed from cluster b23536c33f2cdd1b
```

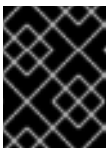
e. 멤버 목록을 다시 표시하고 멤버가 제거되었는지 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+-------------------+--------------------------+--------------------------
-+------------------------+
| ID               | STATUS  | NAME                | PEER ADDRS               | CLIENT
ADDRS              | IS LEARNER |
+------------------+---------+-------------------+--------------------------+--------------------------
-+------------------------+
| cc3830a72fc357f9 | started | openshift-control-plane-2 | https://192.168.10.11:2380/ |
https://192.168.10.11:2379/ | false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380/ |
https://192.168.10.10:2379/ | false |
+------------------+---------+-------------------+--------------------------+--------------------------
-+------------------------+
```

이제 노드 쉘을 종료할 수 있습니다.



중요

멤버를 제거한 후에는 나머지 etcd 인스턴스가 재부팅되는 동안 잠시 동안 클러스터에 접속할 수 없을 수 있습니다.

2. 다음 명령을 입력하여 쿼럼 가드를 끕니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

이 명령을 사용하면 비밀을 성공적으로 다시 생성하고 정적 포드를 롤아웃할 수 있습니다.

3. 다음 명령을 실행하여 제거된 비정상적인 etcd 멤버에 대한 이전 비밀을 제거합니다.

   a. 삭제된 비정상 etcd 멤버의 시크릿(secrets)을 나열합니다.

   ```
   $ oc get secrets -n openshift-etcd | grep openshift-control-plane-2
   ```

   이 프로세스의 앞부분에서 기록한 비정상 etcd 멤버의 이름을 전달합니다.

   다음 출력에 표시된대로 피어, 서빙 및 메트릭 시크릿이 있습니다.

   ```
   etcd-peer-openshift-control-plane-2              kubernetes.io/tls   2   134m
   etcd-serving-metrics-openshift-control-plane-2   kubernetes.io/tls   2   134m
   etcd-serving-openshift-control-plane-2           kubernetes.io/tls   2   134m
   ```

   b. 제거된 비정상 etcd 멤버의 시크릿을 삭제합니다.

      i. 피어 시크릿을 삭제합니다.

      ```
      $ oc delete secret etcd-peer-openshift-control-plane-2 -n openshift-etcd

      secret "etcd-peer-openshift-control-plane-2" deleted
      ```

      ii. 서빙 시크릿을 삭제합니다.

      ```
      $ oc delete secret etcd-serving-metrics-openshift-control-plane-2 -n openshift-etcd

      secret "etcd-serving-metrics-openshift-control-plane-2" deleted
      ```

      iii. 메트릭 시크릿을 삭제합니다.

      ```
      $ oc delete secret etcd-serving-openshift-control-plane-2 -n openshift-etcd

      secret "etcd-serving-openshift-control-plane-2" deleted
      ```

4. 비정상 멤버의 컴퓨터를 가져옵니다.
   클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

   출력 예

   ```
   NAME                              PHASE     TYPE   REGION   ZONE   AGE     NODE
   PROVIDERID                                                              STATE
   examplecluster-control-plane-0    Running                          3h11m   openshift-control-plane-0
   baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-
   0aa41222964e   externally provisioned (1)
   examplecluster-control-plane-1    Running                          3h11m   openshift-control-plane-1
   baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
   03b20280a3e1   externally provisioned
   examplecluster-control-plane-2    Running                          3h11m   openshift-control-plane-2
   baremetalhost:///openshift-machine-api/openshift-control-plane-2/3354bdac-61d8-410f-be5b-
   ```

```
6a395b056135   externally provisioned
examplecluster-compute-0         Running                        165m   openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f         provisioned
examplecluster-compute-1         Running                        165m   openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9         provisioned
```

1 이는 비정상 노드 **(examplecluster-control-plane-2)** 에 대한 제어 플레인 머신입니다.

5. 다음 명령을 실행하여 Bare Metal Operator를 사용할 수 있는지 확인하세요.

```
$ oc get clusteroperator baremetal
```

출력 예

```
NAME        VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
baremetal   4.19.0    True        False         False      3d15h
```

6. 다음 명령을 실행하여 이전 **BareMetalHost** 객체를 제거합니다.

```
$ oc delete bmh openshift-control-plane-2 -n openshift-machine-api
```

출력 예

```
baremetalhost.metal3.io "openshift-control-plane-2" deleted
```

7. 다음 명령을 실행하여 문제가 있는 멤버의 컴퓨터를 삭제합니다.

```
$ oc delete machine -n openshift-machine-api examplecluster-control-plane-2
```

BareMetalHost 및 **Machine** 오브젝트를 제거한 후 **Machine** 컨트롤러에서 **Node** 오브젝트를 자동으로 삭제합니다.

어떤 이유로든 머신 삭제가 지연되거나 명령이 방해를 받아 지연되는 경우, 머신 객체 종료자 필드를 제거하여 강제로 삭제할 수 있습니다.

중요

Ctrl+c를 눌러 기계 삭제를 중단하지 마세요. 명령이 완료될 때까지 진행되도록 허용해야 합니다. 새 터미널 창을 열어 종료자 필드를 편집하고 삭제합니다.

건강에 문제가 있는 멤버의 컴퓨터를 삭제하면 자동으로 새 컴퓨터가 프로비저닝됩니다.

a. 다음 명령을 실행하여 머신 구성을 편집합니다.

```
$ oc edit machine -n openshift-machine-api examplecluster-control-plane-2
```

b. **머신** 사용자 지정 리소스에서 다음 필드를 삭제한 다음 업데이트된 파일을 저장합니다.

```
  finalizers:
  - machine.machine.openshift.io
```

출력 예

```
machine.machine.openshift.io/examplecluster-control-plane-2 edited
```

8. 다음 명령을 실행하여 머신이 삭제되었는지 확인하세요.

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

출력 예

```
NAME                              PHASE     TYPE   REGION   ZONE   AGE     NODE
PROVIDERID                                                        STATE
examplecluster-control-plane-0    Running                          3h11m   openshift-control-plane-0
baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-
0aa41222964e   externally provisioned
examplecluster-control-plane-1    Running                          3h11m   openshift-control-plane-1
baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
03b20280a3e1   externally provisioned
examplecluster-compute-0          Running                          165m    openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f          provisioned
examplecluster-compute-1          Running                          165m    openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9          provisioned
```

9. 다음 명령을 실행하여 노드가 삭제되었는지 확인하세요.

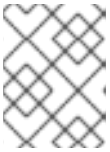
```
$ oc get nodes

NAME                    STATUS ROLES    AGE   VERSION
openshift-control-plane-0 Ready master 3h24m v1.32.3
openshift-control-plane-1 Ready master 3h24m v1.32.3
openshift-compute-0       Ready worker 176m v1.32.3
openshift-compute-1       Ready worker 176m v1.32.3
```

10. 새로운 **BareMetalHost** 개체와 BMC 자격 증명을 저장할 비밀번호를 만듭니다.

```
$ cat <<EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: openshift-control-plane-2-bmc-secret
  namespace: openshift-machine-api
data:
  password: <password>
  username: <username>
type: Opaque
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
```

```
metadata:
  name: openshift-control-plane-2
  namespace: openshift-machine-api
spec:
  automatedCleaningMode: disabled
  bmc:
    address: redfish://10.46.61.18:443/redfish/v1/Systems/1
    credentialsName: openshift-control-plane-2-bmc-secret
    disableCertificateVerification: true
  bootMACAddress: 48:df:37:b0:8a:a0
  bootMode: UEFI
  externallyProvisioned: false
  online: true
  rootDeviceHints:
    deviceName: /dev/disk/by-id/scsi-<serial_number>
  userData:
    name: master-user-data-managed
    namespace: openshift-machine-api
EOF
```



참고

사용자 이름과 비밀번호는 다른 베어 메탈 호스트의 비밀에서 찾을 수 있습니다. **bmc:address** 에 사용할 프로토콜은 다른 bmh 객체에서 가져올 수 있습니다.

중요

기존 제어 평면 호스트에서 **BareMetalHost** 개체 정의를 재사용하는 경우 **externallyProvisioned** 필드를 **true** 로 설정된 상태로 두지 마세요.

기존 제어 평면 **BareMetalHost** 개체는 OpenShift Container Platform 설치 프로그램에서 프로비저닝된 경우 **externallyProvisioned** 플래그가 **true** 로 설정될 수 있습니다.

검사가 완료되면 **BareMetalHost** 개체가 생성되어 프로비저닝할 수 있습니다.

11. 사용 가능한 **BareMetalHost** 오브젝트를 사용하여 생성 프로세스를 확인합니다.

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

NAME                 STATE                  CONSUMER                        ONLINE ERROR   AGE
openshift-control-plane-0 externally provisioned examplecluster-control-plane-0 true
4h48m
openshift-control-plane-1 externally provisioned examplecluster-control-plane-1 true
4h48m
openshift-control-plane-2 available          examplecluster-control-plane-3 true     47m
openshift-compute-0       provisioned        examplecluster-compute-0       true     4h48m
openshift-compute-1       provisioned        examplecluster-compute-1       true     4h48m
```

a. 새로운 머신이 생성되었는지 확인하세요.

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

출력 예

```
NAME                         PHASE    TYPE   REGION   ZONE   AGE    NODE
PROVIDERID                                                          STATE
examplecluster-control-plane-0       Running                          3h11m   openshift-control-
plane-0   baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-
3ff2-41c5-b099-0aa41222964e   externally provisioned   (1)
examplecluster-control-plane-1       Running                          3h11m   openshift-control-
plane-1   baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-
329c-475e-8d81-03b20280a3e1   externally provisioned
examplecluster-control-plane-2       Running                          3h11m   openshift-control-
plane-2   baremetalhost:///openshift-machine-api/openshift-control-plane-2/3354bdac-
61d8-410f-be5b-6a395b056135   externally provisioned
examplecluster-compute-0             Running                          165m    openshift-compute-
0         baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-
4bb3-80ec-13a31858241f        provisioned
examplecluster-compute-1             Running                          165m    openshift-compute-
1         baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-
4241-91dc-e7ea72ab13b9        provisioned
```

(1) 새 시스템 **clustername-8qw5l-master-3**이 생성되고 단계가 **Provisioning**에서 **Running**으로 변경되면 시스템이 준비 상태가 됩니다.

새 시스템을 만드는 데 몇 분이 소요될 수 있습니다. etcd 클러스터 Operator는 머신 또는 노드가 정상 상태로 돌아 오면 자동으로 동기화됩니다.

b. 다음 명령을 실행하여 베어 메탈 호스트가 프로비저닝되고 오류가 보고되지 않았는지 확인하세요.

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

출력 예

```
$ oc get bmh -n openshift-machine-api
NAME                 STATE                  CONSUMER                  ONLINE ERROR AGE
openshift-control-plane-0 externally provisioned examplecluster-control-plane-0 true
4h48m
openshift-control-plane-1 externally provisioned examplecluster-control-plane-1 true
4h48m
openshift-control-plane-2 provisioned          examplecluster-control-plane-3 true
47m
openshift-compute-0       provisioned          examplecluster-compute-0       true
4h48m
openshift-compute-1       provisioned          examplecluster-compute-1       true
4h48m
```

c. 다음 명령을 실행하여 새 노드가 추가되었고 준비 상태인지 확인하세요.

```
$ oc get nodes
```

출력 예

```
$ oc get nodes
NAME                 STATUS ROLES   AGE   VERSION
openshift-control-plane-0 Ready master 4h26m v1.32.3
```

```
openshift-control-plane-1 Ready master 4h26m v1.32.3
openshift-control-plane-2 Ready master 12m   v1.32.3
openshift-compute-0       Ready worker 3h58m v1.32.3
openshift-compute-1       Ready worker 3h58m v1.32.3
```

12. 다음 명령을 입력하여 쿼럼 가드를 다시 켭니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

13. 다음 명령을 입력하면 **지원되지 않는 ConfigOverrides** 섹션이 개체에서 제거되었는지 확인할 수 있습니다.

```
$ oc get etcd/cluster -oyaml
```

14. 단일 노드 OpenShift를 사용하는 경우 노드를 다시 시작합니다. 그렇지 않으면 etcd 클러스터 운영자에서 다음 오류가 발생할 수 있습니다.

출력 예

```
EtcdCertSignerControllerDegraded: [Operation cannot be fulfilled on secrets "etcd-peer-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-sno-0": the object has been modified; please apply your changes to the latest version and try again, Operation cannot be fulfilled on secrets "etcd-serving-metrics-sno-0": the object has been modified; please apply your changes to the latest version and try again]
```

검증

1. 모든 etcd pod가 올바르게 실행되고 있는지 확인합니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

```
etcd-openshift-control-plane-0     5/5     Running     0     105m
etcd-openshift-control-plane-1     5/5     Running     0     107m
etcd-openshift-control-plane-2     5/5     Running     0     103m
```

이전 명령의 출력에 두 개의 pod만 나열되는 경우 수동으로 etcd 재배포를 강제 수행할 수 있습니다. 클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$( date --rfc-3339=ns )""}}' --type=merge 1
```

1 **forceRedeploymentReason** 값은 고유해야하므로 타임 스탬프가 추가됩니다.

정확히 3개의 etcd 멤버가 있는지 확인하려면 실행 중인 etcd 컨테이너에 연결하고 영향을 받은 노드에 없는 Pod의 이름을 전달합니다. 클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

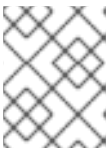
```
$ oc rsh -n openshift-etcd etcd-openshift-control-plane-0
```

2. 멤버 목록을 확인합니다.

```
sh-4.2# etcdctl member list -w table
```

출력 예

```
+------------------+---------+-------------------+--------------------------+--------------------------+---
--------------+
|        ID        | STATUS  |       NAME        |        PEER ADDRS        |       CLIENT ADDRS
|    IS LEARNER    |
+------------------+---------+-------------------+--------------------------+--------------------------+---
--------------+
| 7a8197040a5126c8 | started | openshift-control-plane-2 | https://192.168.10.11:2380 |
https://192.168.10.11:2379 |    false |
| 8d5abe9669a39192 | started | openshift-control-plane-1 | https://192.168.10.10:2380 |
https://192.168.10.10:2379 |    false |
| cc3830a72fc357f9 | started | openshift-control-plane-0 | https://192.168.10.9:2380 |
https://192.168.10.9:2379 |     false |
+------------------+---------+-------------------+--------------------------+--------------------------+---
--------------+
```



참고

이전 명령의 출력에 세 개 이상의 etcd 멤버가 나열된 경우 원하지 않는 멤버를 신중하게 제거해야 합니다.

3. 다음 명령을 실행하여 모든 etcd 멤버가 정상인지 확인하세요.

```
# etcdctl endpoint health --cluster
```

출력 예

```
https://192.168.10.10:2379 is healthy: successfully committed proposal: took = 8.973065ms
https://192.168.10.9:2379 is healthy: successfully committed proposal: took = 11.559829ms
https://192.168.10.11:2379 is healthy: successfully committed proposal: took = 11.665203ms
```

4. 다음 명령을 실행하여 모든 노드가 최신 개정 버전인지 확인하세요.

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

```
AllNodesAtLatestRevision
```

추가 리소스

- 머신 라이프사이클 후크를 통한 쿼럼 보호

4.3. 재해 복구

재해 복구 문서에서는 관리자에게 OpenShift Container Platform 클러스터에서 발생할 수있는 여러 재해 상황을 복구하는 방법에 대한 정보를 제공합니다. 관리자는 클러스터를 작동 상태로 복원하려면 다음 절차 중 하나 이상을 수행해야합니다.

중요

재해 복구를 위해서는 최소한 하나의 정상적인 제어 평면 호스트가 필요합니다.

4.3.1. 정족수 회복

쿼럼 손실로 인해 오프라인이 된 클러스터에서 etcd 쿼럼을 복원하려면 **quorum-restore.sh** 스크립트를 사용할 수 있습니다. 쿼럼이 손실되면 OpenShift Container Platform API는 읽기 전용이 됩니다. 쿼럼이 복구되면 OpenShift Container Platform API는 읽기/쓰기 모드로 돌아갑니다.

4.3.1.1. 고가용성 클러스터를 위한 **etcd** 쿼럼 복원

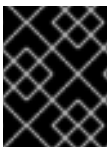
쿼럼 손실로 인해 오프라인이 된 클러스터에서 etcd 쿼럼을 복원하려면 **quorum-restore.sh** 스크립트를 사용할 수 있습니다. 쿼럼이 손실되면 OpenShift Container Platform API는 읽기 전용이 됩니다. 쿼럼이 복구되면 OpenShift Container Platform API는 읽기/쓰기 모드로 돌아갑니다.

quorum-restore.sh 스크립트는 로컬 데이터 디렉토리를 기반으로 새로운 단일 멤버 etcd 클러스터를 즉시 다시 가져오고 이전 클러스터 식별자를 폐기하여 다른 모든 멤버를 유효하지 않은 것으로 표시합니다. 제어 평면을 복원하는 데 사전 백업이 필요하지 않습니다.

HA(고가용성) 클러스터의 경우 3 노드 HA 클러스터를 종료하여 클러스터 분할을 방지하기 위해 두 호스트에서 etcd를 종료해야 합니다. 4-node 및 5-노드 HA 클러스터에서 호스트 3개를 종료해야 합니다. 쿼럼에는 노드의 단순 과반수가 필요합니다. 3-노드 HA 클러스터에서 쿼럼에 필요한 최소 노드 수는 2개입니다. 4-node 및 5-노드 HA 클러스터에서 쿼럼에 필요한 최소 노드 수는 3개입니다. 복구 호스트에서 백업을 통해 새 클러스터를 시작하면 다른 etcd 멤버가 여전히 쿼럼을 형성하고 서비스를 계속할 수 있습니다.

주의

복원을 실행하는 호스트에 모든 데이터가 복제되지 않은 경우 데이터 손실이 발생할 수 있습니다.



중요

쿼럼 복원은 복원 프로세스 외부에서 노드 수를 줄이는 데 사용되어서는 안 됩니다. 노드 수를 줄이면 지원되지 않는 클러스터 구성이 발생합니다.

사전 요구 사항

- 쿼럼을 복원하는 데 사용된 노드에 SSH 액세스 권한이 있습니다.

프로세스

1. 복구 호스트로 사용할 컨트롤 플레인 호스트를 선택합니다. 이 호스트에서 복원 작업을 실행합니다.
    a. 다음 명령을 실행하여 실행 중인 etcd 포드를 나열합니다.

```
$ oc get pods -n openshift-etcd -l app=etcd --field-selector="status.phase==Running"
```

b. 포드를 선택하고 다음 명령을 실행하여 IP 주소를 얻습니다.

```
$ oc exec -n openshift-etcd <etcd-pod> -c etcdctl -- etcdctl endpoint status -w table
```

학습자가 아니고 Raft 인덱스가 가장 높은 회원의 IP 주소를 기록해 보세요.

c. 다음 명령을 실행하고 선택한 etcd 멤버의 IP 주소에 해당하는 노드 이름을 기록해 둡니다.

```
$ oc get nodes -o jsonpath='{range .items[*]}[{.metadata.name},{.status.addresses[?
(@.type=="InternalIP")].address}]{end}'
```

2. SSH를 사용하여 선택한 복구 노드에 연결하고 다음 명령을 실행하여 etcd 쿼럼을 복원합니다.

```
$ sudo -E /usr/local/bin/quorum-restore.sh
```

몇 분 후, 다운된 노드는 복구 스크립트가 실행된 노드와 자동으로 동기화됩니다. 남아 있는 온라인 노드는 **quorum-restore.sh** 스크립트에서 생성된 새 etcd 클러스터에 자동으로 다시 가입합니다. 이 과정은 몇 분 정도 걸립니다.

3. SSH 세션을 종료합니다.

4. 노드가 오프라인인 경우 3-노드 구성으로 돌아갑니다. 오프라인인 각 노드에 대해 다음 단계를 반복하여 삭제하고 다시 생성합니다. 머신이 다시 생성된 후 새로운 개정판이 강제로 적용되고 etcd가 자동으로 확장됩니다.

- 사용자 제공 베어 메탈 설치를 사용하는 경우 원래 생성에 사용한 것과 동일한 방법을 사용하여 제어 평면 머신을 다시 생성할 수 있습니다. 자세한 내용은 "베어메탈에 사용자 프로비저닝 클러스터 설치"를 참조하세요.

주의

복구 호스트의 머신을 삭제했다가 다시 생성하지 마세요.

- 설치 프로그램에서 제공한 인프라를 실행 중이거나 Machine API를 사용하여 컴퓨터를 만든 경우 다음 단계를 수행합니다.

주의

복구 호스트의 머신을 삭제했다가 다시 생성하지 마세요.

설치 프로그램이 제공하는 인프라에서 베어 메탈을 설치하는 경우 제어 플레인 머신은 다시 생성되지 않습니다. 자세한 내용은 "베어 메탈 제어 평면 노드 교체"를 참조하세요.

a. 오프라인 노드 중 하나에 대한 머신을 얻습니다.
클러스터에 액세스할 수 있는 터미널에서 **cluster-admin** 사용자로 다음 명령을 실행합니다.

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

출력 예

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

(1) 이는 오프라인 노드 **ip-10-0-131-183.ec2.internal** 의 제어 플레인 머신입니다.

b. 다음을 실행하여 오프라인 노드의 머신을 삭제합니다.

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

(1) 오프라인 노드에 대한 제어 플레인 머신의 이름을 지정합니다.

오프라인 노드의 머신을 삭제하면 자동으로 새로운 머신이 프로비저닝됩니다.

5. 다음을 실행하여 새 머신이 생성되었는지 확인하세요.

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

출력 예

```
NAME                                 PHASE     TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
clustername-8qw5l-master-1           Running   m4.xlarge   us-east-1   us-east-1b   3h37m   ip-10-0-143-125.ec2.internal   aws:///us-east-1b/i-096c349b700a19631   running
clustername-8qw5l-master-2           Running   m4.xlarge   us-east-1   us-east-1c   3h37m   ip-10-0-154-194.ec2.internal   aws:///us-east-1c/i-02626f1dba9ed5bba   running
clustername-8qw5l-master-3           Provisioning   m4.xlarge   us-east-1   us-east-1a   85s
```

```
ip-10-0-173-171.ec2.internal    aws:///us-east-1a/i-015b0888fe17bc2c8  running (1)
clustername-8qw5l-worker-us-east-1a-wbtgd  Running      m4.large  us-east-1  us-east-1a
3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-010ef6279b4662ced  running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running      m4.large  us-east-1  us-east-1b
3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-0cb45ac45a166173b  running
clustername-8qw5l-worker-us-east-1c-pkg26  Running      m4.large  us-east-1  us-east-1c
3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-06861c00007751b0a  running
```

(1) 새 시스템 **clustername-8qw5l-master-3**이 생성되고 단계가 **Provisioning**에서 **Running**으로 변경되면 시스템이 준비 상태가 됩니다.

새 시스템을 만드는 데 몇 분이 소요될 수 있습니다. etcd 클러스터 Operator는 머신 또는 노드가 정상 상태로 돌아 오면 자동으로 동기화됩니다.

a. 오프라인 상태인 각 노드에 대해 이 단계를 반복합니다.

6. 다음 명령을 실행하여 제어 평면이 복구될 때까지 기다리세요.

```
$ oc adm wait-for-stable-cluster
```

참고

제어 평면이 복구되는 데 최대 15분이 걸릴 수 있습니다.

문제 해결

- etcd 정적 포드 롤아웃에 진행 상황이 보이지 않으면 다음 명령을 실행하여 etcd 클러스터 운영자에서 강제로 재배포할 수 있습니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$(date --rfc-3339=ns )"'"}}' --type=merge
```

추가 리소스

- 베어 메탈에 사용자 프로비저닝 클러스터 설치
- 베어 메탈 컨트롤 플레인 노드 교체

참고

제어 평면 노드의 대부분이 여전히 사용 가능하고 etcd 쿼럼이 있는 경우, 비정상 상태인 단일 etcd 멤버를 교체합니다 .

4.3.2. 이전 클러스터 상태로 복원

클러스터를 이전 상태로 복구하려면 스냅샷을 작성하여 **etcd 데이터 백업**을 수행해야합니다. 이 스냅샷을 사용하여 클러스터 상태를 복구합니다. 자세한 내용은 "etcd 데이터 백업"을 참조하세요.

해당되는 경우 만료 된 컨트롤 플레인 인증서 복구 를 수행해야 할 수도 있습니다.

주의

이전 클러스터 상태로 복원하는 것은 실행 중인 클러스터에서 수행하기에 위험하고 불안정한 작업입니다. 이 절차는 마지막 수단으로만 사용해야 합니다.

복원을 수행하기 전에 "이전 클러스터 상태로 복원하는 방법"을 참조하여 클러스터에 미치는 영향에 대한 자세한 내용을 확인하세요.

4.3.2.1. 이전 클러스터 상태로의 복원 정보

클러스터를 이전 상태로 복구하려면 스냅샷을 작성하여 **etcd 데이터 백업**을 수행해야합니다. 이 스냅샷을 사용하여 클러스터 상태를 복구합니다. 자세한 내용은 "etcd 데이터 백업"을 참조하세요.

etcd 백업을 사용하여 클러스터를 이전 상태로 복원할 수 있습니다. 이를 사용하여 다음과 같은 상황에서 복구할 수 있습니다.

- 클러스터에서 대부분의 컨트롤 플레인 호스트가 손실되었습니다(쿼럼 손실).
- 관리자가 중요한 것을 삭제했으며 클러스터를 복구하려면 복원해야 합니다.

주의

이전 클러스터 상태로 복원하는 것은 실행 중인 클러스터에서 수행하기에 위험하고 불안정한 작업입니다. 이는 마지막 수단으로만 사용해야 합니다.

Kubernetes API 서버를 사용하여 데이터를 검색할 수 있는 경우 etcd를 사용할 수 있으며 etcd 백업을 사용하여 복원할 수 없습니다.

etcd를 복원하려면 클러스터를 효율적으로 복원하는 데 시간이 걸리며 모든 클라이언트가 충돌하는 병렬 기록이 발생합니다. 이는 네트워크 운영자를 포함하여 kubelet, Kubernetes 컨트롤러 관리자, 영구 볼륨 컨트롤러, OpenShift Container Platform 운영자와 같은 구성 요소를 감시하는 동작에 영향을 미칠 수 있습니다.

이로 인해 etcd의 콘텐츠가 디스크의 실제 콘텐츠와 일치하지 않을 때 Operator가 문제가 발생하여 디스크의 파일이 etcd의 콘텐츠와 충돌할 때 Kubernetes API 서버, Kubernetes 컨트롤러 관리자, Kubernetes 스케줄러 및 etcd의 Operator가 중단될 수 있습니다. 여기에는 문제를 해결하기 위해 수동 작업이 필요할 수 있습니다.

극단적인 경우 클러스터에서 영구 볼륨 추적을 손실하고, 더 이상 존재하지 않는 중요한 워크로드를 삭제하고, 시스템을 다시 이미지화하고, 만료된 인증서로 CA 번들을 다시 작성할 수 있습니다.

4.3.2.2. 단일 노드에 대한 이전 클러스터 상태로 복원

저장된 etcd 백업을 사용하여 단일 노드에서 이전 클러스터 상태를 복원할 수 있습니다.

중요

클러스터를 복원할 때 동일한 z-stream 릴리스에서 가져온 etcd 백업을 사용해야 합니다. 예를 들어, OpenShift Container Platform 4.19.2 클러스터는 4.19.2에서 가져온 etcd 백업을 사용해야 합니다.

사전 요구 사항

- 설치 중에 사용된 것과 같은 인증서 기반 **kubeconfig** 파일을 통해 **cluster-admin** 역할이 있는 사용자로 클러스터에 액세스합니다.
- 제어 평면 호스트에 SSH 액세스 권한이 있습니다.
- 동일한 백업에서 가져온 etcd 스냅샷과 정적 pod 리소스가 모두 포함된 백업 디렉토리입니다. 디렉토리의 파일 이름은 **snapshot_<datetimestamp>.db** 및 **static_kuberesources_<datetimestamp>.tar.gz** 형식이어야합니다.

프로세스

1. SSH를 사용하여 단일 노드에 연결하고 다음 명령을 실행하여 etcd 백업을 **/home/core** 디렉터리에 복사합니다.

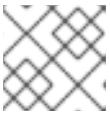
```
$ cp <etcd_backup_directory> /home/core
```

2. 이전 백업에서 클러스터를 복원하려면 단일 노드에서 다음 명령을 실행하세요.

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/<etcd_backup_directory>
```

3. SSH 세션을 종료합니다.
4. 다음 명령을 실행하여 제어 평면의 복구 진행 상황을 모니터링합니다.

```
$ oc adm wait-for-stable-cluster
```



참고

제어 평면이 복구되는 데 최대 15분이 걸릴 수 있습니다.

4.3.2.3. 두 개 이상의 노드에 대해 이전 클러스터 상태로 복원

저장된 etcd 백업을 사용하여 이전 클러스터 상태를 복원하거나 대부분의 제어 평면 호스트를 손실한 클러스터를 복원할 수 있습니다.

HA(고가용성) 클러스터의 경우 3 노드 HA 클러스터를 종료하여 클러스터 분할을 방지하기 위해 두 호스트에서 etcd를 종료해야 합니다. 4-node 및 5-노드 HA 클러스터에서 호스트 3개를 종료해야 합니다. 쿼럼에는 노드의 단순 과반수가 필요합니다. 3-노드 HA 클러스터에서 쿼럼에 필요한 최소 노드 수는 2개입니다. 4-node 및 5-노드 HA 클러스터에서 쿼럼에 필요한 최소 노드 수는 3개입니다. 복구 호스트에서 백업을 통해 새 클러스터를 시작하면 다른 etcd 멤버가 여전히 쿼럼을 형성하고 서비스를 계속할 수 있습니다.

참고

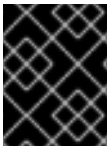
클러스터가 컨트롤 플레인 머신 세트를 사용하는 경우 etcd 복구 절차에 대한 "컨트롤 플레인 머신 세트 문제 해결"의 "성능이 저하된 etcd Operator 복구"를 참조하세요. 단일 노드의 OpenShift Container Platform의 경우 "단일 노드의 이전 클러스터 상태로 복원"을 참조하세요.

중요

클러스터를 복원할 때 동일한 z-stream 릴리스에서 가져온 etcd 백업을 사용해야 합니다. 예를 들어, OpenShift Container Platform 4.19.2 클러스터는 4.19.2에서 가져온 etcd 백업을 사용해야 합니다.

사전 요구 사항

- 설치 중에 사용된 것과 같은 인증서 기반 **kubeconfig** 파일을 통해 **cluster-admin** 역할이 있는 사용자로 클러스터에 액세스합니다.
- 복구 호스트로 사용할 정상적인 컨트롤 플레인 호스트가 있어야 합니다.
- 제어 평면 호스트에 SSH 액세스 권한이 있습니다.
- 동일한 백업에서 가져온 **etcd** 스냅샷과 정적 pod의 리소스가 모두 포함된 백업 디렉토리입니다. 디렉토리의 파일 이름은 **snapshot_<datetimestamp>.db** 및 **static_kuberesources_<datetimestamp>.tar.gz** 형식이어야합니다.
- 노드는 접근 가능하거나 부팅 가능해야 합니다.



중요

복구가 불가능한 제어 평면 노드의 경우 SSH 연결을 설정하거나 정적 포드를 중지할 필요가 없습니다. 다른 비복구 제어 평면 머신을 하나씩 삭제하고 다시 생성할 수 있습니다.

프로세스

1. 복구 호스트로 사용할 컨트롤 플레인 호스트를 선택합니다. 이는 복원 작업을 실행하는 호스트입니다.
2. 복구 호스트를 포함하여 각 컨트롤 플레인 노드에 SSH 연결을 설정합니다.
   복구 프로세스가 시작된 후에는 **kube-apiserver** 에 액세스할 수 없게 되어 제어 평면 노드에 액세스할 수 없습니다. 따라서 다른 터미널에서 각 컨트롤 플레인 호스트에 대한 SSH 연결을 설정하는 것이 좋습니다.

   중요

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

3. SSH를 사용하여 각 제어 평면 노드에 연결하고 다음 명령을 실행하여 etcd를 비활성화합니다.

   ```
   $ sudo -E /usr/local/bin/disable-etcd.sh
   ```

4. etcd 백업 디렉토리를 복구 컨트롤 플레인 호스트에 복사합니다.

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

5. SSH를 사용하여 복구 호스트에 연결하고 다음 명령을 실행하여 이전 백업에서 클러스터를 복원합니다.

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

6. SSH 세션을 종료합니다.

7. API가 응답하면 다음 명령을 실행하여 etcd Operator 쿼럼 가드를 끕니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
{"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

8. 다음 명령을 실행하여 컨트롤 플레인의 복구 진행 상황을 모니터링합니다.

```
$ oc adm wait-for-stable-cluster
```

참고

컨트롤 플레인을 복구하는 데 최대 15분이 걸릴 수 있습니다.

9. 복구되면 다음 명령을 실행하여 쿼럼 가드를 활성화합니다.

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

문제 해결

etcd 정적 pod를 롤아웃하는 진행 상황이 표시되지 않으면 다음 명령을 실행하여 **cluster-etcd-operator** 에서 강제로 재배포할 수 있습니다.

```
$ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$(date --rfc-3339=ns
)""}}' --type=merge
```

추가 리소스

- 성능 저하된 etcd Operator 복구

4.3.2.4. etcd 백업에서 수동으로 클러스터 복원

"이전 클러스터 상태로 복원" 섹션에 설명된 복원 절차:

- UPI 설치 시 컨트롤 플레인 노드에 대한 **Machine** 또는 **ControlPlaneMachineset** 을 생성하지 않으므로 UPI 설치 방법을 사용하여 설치된 클러스터에 대한 2개의 컨트롤 플레인 노드를 완전히 다시 생성해야 합니다.

- 새 단일 멤버 etcd 클러스터를 시작한 /usr/local/bin/cluster-restore.sh 스크립트를 사용하여 세 멤버로 확장합니다.

이와 반대로 이 절차는 다음과 같습니다.

- 컨트롤 플레인 노드를 다시 생성할 필요가 없습니다.

- 3개의 멤버 etcd 클러스터를 직접 시작합니다.

클러스터에서 컨트롤 플레인에 **MachineSet** 을 사용하는 경우 etcd 복구 절차를 위해 "Restoring to a previous cluster state"를 사용하는 것이 좋습니다.

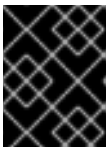
클러스터를 복원할 때 동일한 z-stream 릴리스에서 가져온 etcd 백업을 사용해야 합니다. 예를 들어 OpenShift Container Platform 4.7.2 클러스터는 4.7.2에서 가져온 etcd 백업을 사용해야 합니다.

사전 요구 사항

- **cluster-admin** 역할(예: **kubeadmin** 사용자)을 사용하여 사용자로 클러스터에 액세스합니다.
- 호스트 사용자를 사용하여 *모든* 컨트롤 플레인 호스트에 대한 SSH 액세스 **(** 예: 기본 **코어** 호스트 사용자)
- 이전 etcd 스냅샷과 동일한 백업의 정적 pod 리소스가 모두 포함된 백업 디렉터리입니다. 디렉토리의 파일 이름은 **snapshot_<datetimestamp>.db** 및 **static_kuberesources_<datetimestamp>.tar.gz** 형식이어야합니다.

프로세스

1. SSH를 사용하여 각 제어 평면 노드에 연결합니다.
   복구 프로세스가 시작된 후에는 Kubernetes API 서버에 액세스할 수 없으므로 컨트롤 플레인 노드에 액세스할 수 없습니다. 이러한 이유로 액세스하는 각 제어 평면 호스트에 대해 별도의 터미널에서 SSH 연결을 사용하는 것이 좋습니다.

   중요

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

2. etcd 백업 디렉터리를 각 컨트롤 플레인 호스트에 복사합니다.
   이 단계에서는 etcd 스냅샷 및 정적 pod의 리소스가 포함된 **backup** 디렉터리를 복구 컨트롤 플레인 호스트의 **/home/core/** 디렉터리에 복사하는 것을 전제로하고 있습니다. 해당 **자산** 폴더가 아직 없다면 만들어야 할 수도 있습니다.

3. 모든 제어 평면 노드에서 정적 포드를 한 번에 한 호스트씩 중지합니다.

   a. 기존 Kubernetes API 서버 정적 Pod 매니페스트를 kubelet 매니페스트 디렉토리 밖으로 옮깁니다.

   ```
   $ mkdir -p /root/manifests-backup
   ```

   ```
   $ mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /root/manifests-backup/
   ```

   b. 다음 명령을 사용하여 Kubernetes API 서버 컨테이너가 중지되었는지 확인하세요.

   ```
   $ crictl ps | grep kube-apiserver | grep -E -v "operator|guard"
   ```

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

   c. Kubernetes API 서버 컨테이너가 계속 실행 중이면 다음 명령을 사용하여 수동으로 종료합니다.

```
$ crictl stop <container_id>
```

d. **kube-controller-manager-pod.yaml** , **kube-scheduler-pod.yaml** , 마지막으로 **etcd-pod.yaml** 에 대해 동일한 단계를 반복합니다.

i. 다음 명령을 사용하여 **kube-controller-manager** 포드를 중지합니다.

```
$ mv /etc/kubernetes/manifests/kube-controller-manager-pod.yaml /root/manifests-backup/
```

ii. 다음 명령을 사용하여 컨테이너가 중지되었는지 확인하세요.

```
$ crictl ps | grep kube-controller-manager | grep -E -v "operator|guard"
```

iii. 다음 명령을 사용하여 **kube-scheduler** pod를 중지합니다.

```
$ mv /etc/kubernetes/manifests/kube-scheduler-pod.yaml /root/manifests-backup/
```

iv. 다음 명령을 사용하여 컨테이너가 중지되었는지 확인하세요.

```
$ crictl ps | grep kube-scheduler | grep -E -v "operator|guard"
```

v. 다음 명령을 사용하여 **etcd** Pod를 중지합니다.

```
$ mv /etc/kubernetes/manifests/etcd-pod.yaml /root/manifests-backup/
```

vi. 다음 명령을 사용하여 컨테이너가 중지되었는지 확인하세요.

```
$ crictl ps | grep etcd | grep -E -v "operator|guard"
```

4. 각 제어 평면 호스트에서 현재 **etcd** 데이터를 **백업** 폴더로 이동하여 저장합니다.

```
$ mkdir /home/core/assets/old-member-data
```

```
$ mv /var/lib/etcd/member /home/core/assets/old-member-data
```

이 데이터는 **etcd** 백업 복원이 작동하지 않고 **etcd** 클러스터를 현재 상태로 복원해야 하는 경우에 유용합니다.

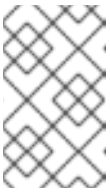
5. 각 제어 평면 호스트에 대한 올바른 etcd 매개변수를 찾습니다.

a. **<ETCD_NAME>** 의 값은 각 제어 평면 호스트마다 고유하며, 특정 제어 평면 호스트의 매니페스트 **/etc/kubernetes/static-pod-resources/etcd-certs/configmaps/restore-etcd-pod/pod.yaml** 파일에 있는 **ETCD_NAME** 변수의 값과 같습니다. 다음 명령을 사용하여 찾을 수 있습니다.

```
RESTORE_ETCD_POD_YAML="/etc/kubernetes/static-pod-resources/etcd-certs/configmaps/restore-etcd-pod/pod.yaml"
cat $RESTORE_ETCD_POD_YAML | \
  grep -A 1 $(cat $RESTORE_ETCD_POD_YAML | grep 'export ETCD_NAME' | grep -Eo 'NODE_.+_ETCD_NAME') | \
  grep -Po '(?<=value: ").+(?=")'
```

b. **<UUID>** 값은 다음 명령을 사용하여 제어 평면 호스트에서 생성할 수 있습니다.

```
$ uuidgen
```



참고

<UUID> 값은 한 번만 생성되어야 합니다. 한 제어 평면 호스트에서 **UUID**를 생성한 후 다른 제어 평면 호스트에서 다시 생성하지 마세요. 다음 단계에서는 모든 제어 평면 호스트에서 동일한 **UUID**가 사용됩니다.

c. **ETCD_NODE_PEER_URL** 의 값은 다음 예와 같이 설정해야 합니다.

```
https://<IP_CURRENT_HOST>:2380
```

다음 명령을 사용하여 특정 제어 평면 호스트의 **<ETCD_NAME>** 에서 올바른 IP를 찾을 수 있습니다.

```
$ echo <ETCD_NAME> | \
  sed -E 's/[.-]/_/g' | \
  xargs -I {} grep {} /etc/kubernetes/static-pod-resources/etcd-certs/configmaps/etcd-scripts/etcd.env | \
  grep "IP" | grep -Po '(?<=").+(?=")'
```

d. **<ETCD_INITIAL_CLUSTER>** 의 값은 다음과 같이 설정해야 합니다. 여기서 **<ETCD_NAME_n>** 은 각 제어 평면 호스트의 **<ETCD_NAME>** 입니다.

참고

사용되는 포트는 2379가 아닌 2380이어야 합니다. 포트 2379는 etcd 데이터베이스 관리에 사용되며 컨테이너의 etcd 시작 명령에서 직접 구성됩니다.

출력 예

```
<ETCD_NAME_0>=<ETCD_NODE_PEER_URL_0>,<ETCD_NAME_1>=<ETCD_NODE_PEER_URL_1>,<ETCD_NAME_2>=<ETCD_NODE_PEER_URL_2>
```
(1)

(1) 각 제어 평면 호스트에서 **ETCD_NODE_PEER_URL** 값을 지정합니다.

<ETCD_INITIAL_CLUSTER> 값은 모든 제어 평면 호스트에서 동일하게 유지됩니다. 다음 단계에서는 모든 제어 평면 호스트에 동일한 값이 필요합니다.

6. 백업에서 etcd 데이터베이스를 다시 생성합니다.
이러한 작업은 각 제어 평면 호스트에서 실행되어야 합니다.

a. 다음 명령을 사용하여 **etcd** 백업을 **/var/lib/etcd** 디렉토리에 복사합니다.

```
$ cp /home/core/assets/backup/<snapshot_yyyy-mm-dd_hhmmss>.db /var/lib/etcd
```

b. 계속하기 전에 올바른 **etcdctl** 이미지를 식별합니다. 다음 명령을 사용하여 Pod 매니페스트 백업에서 이미지를 검색합니다.

```
$ jq -r '.spec.containers[]|select(.name=="etcdctl")|.image' /root/manifests-backup/etcd-pod.yaml
```

```
$ podman run --rm -it --entrypoint="/bin/bash" -v /var/lib/etcd:/var/lib/etcd:z <image-hash>
```

c. **etcdctl** 도구의 버전이 백업이 생성된 **etcd** 서버의 버전인지 확인하세요.

```
$ etcdctl version
```

d. 현재 호스트에 맞는 올바른 값을 사용하여 **etcd** 데이터베이스를 다시 생성하려면 다음 명령을 실행하세요.

```
$ ETCDCTL_API=3 /usr/bin/etcdctl snapshot restore /var/lib/etcd/<snapshot_yyyy-mm-dd_hhmmss>.db \
  --name "<ETCD_NAME>" \
  --initial-cluster="<ETCD_INITIAL_CLUSTER>" \
  --initial-cluster-token "openshift-etcd-<UUID>" \
  --initial-advertise-peer-urls "<ETCD_NODE_PEER_URL>" \
  --data-dir="/var/lib/etcd/restore-<UUID>" \
  --skip-hash-check=true
```

참고

etcd 데이터베이스를 다시 생성할 때는 따옴표가 필수입니다.

7. **추가된 멤버** 로그에 인쇄된 값을 기록하세요. 예:

출력 예

```
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "56cd73b614699e7", "added-peer-peer-urls": ["https://10.0.91.5:2380"], "added-peer-is-learner": false}
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "1f63d01b31bb9a9e", "added-peer-peer-urls": ["https://10.0.90.221:2380"], "added-peer-is-learner": false}
2022-06-28T19:52:43Z info membership/cluster.go:421 added member {"cluster-id": "c5996b7c11c30d6b", "local-member-id": "0", "added-peer-id": "fdc2725b3b70127c", "added-peer-peer-urls": ["https://10.0.94.214:2380"], "added-peer-is-learner": false}
```

a. 컨테이너에서 나오세요.

b. 다른 제어 평면 호스트에서도 이 단계를 반복하여 **추가된 멤버** 로그에 인쇄된 값이 모든 제어 평면 호스트에서 동일한지 확인합니다.

8. 재생성된 **etcd** 데이터베이스를 기본 위치로 이동합니다.
이러한 작업은 각 제어 평면 호스트에서 실행되어야 합니다.

a. 재생성된 데이터베이스(이전 **etcdctl 스냅샷 복원** 명령으로 생성된 **멤버** 폴더)를 기본 etcd 위치 **/var/lib/etcd** 로 이동합니다.

```
$ mv /var/lib/etcd/restore-<UUID>/member /var/lib/etcd
```

b. **/var/lib/** etcd 디렉토리의 **/var/lib/etcd** /member 폴더에 대한 SELinux 컨텍스트를 복원합니다.

```
$ restorecon -vR /var/lib/etcd/
```

c. 남아 있는 파일과 디렉토리를 제거합니다.

```
$ rm -rf /var/lib/etcd/restore-<UUID>
```

```
$ rm /var/lib/etcd/<snapshot_yyyy-mm-dd_hhmmss>.db
```

중요

작업이 끝나면 **/var/lib/etcd** 디렉토리에는 **member** 폴더만 포함되어야 합니다.

d. 다른 컨트롤 플레인 호스트에서 이 단계를 반복합니다.

9. etcd 클러스터를 다시 시작합니다.

a. 한 번에 하나의 호스트이지만 모든 컨트롤 플레인 호스트에서 다음 단계를 실행해야 합니다.

b. kubelet이 관련 컨테이너를 시작하도록 하기 위해 **etcd** 정적 포드 매니페스트를 kubelet 매니페스트 디렉토리로 다시 이동합니다.

```
$ mv /root/manifests-backup/etcd-pod.yaml /etc/kubernetes/manifests
```

c. 모든 **etcd** 컨테이너가 시작되었는지 확인하세요.

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

출력 예

```
38c814767ad983
f79db5a8799fd2c08960ad9ee22f784b9fbe23babe008e8a3bf68323f004c840
28 seconds ago      Running             etcd-health-monitor                 2
fe4b9c3d6483c
e1646b15207c6
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago  Running             etcd-metrics                        0
fe4b9c3d6483c
08ba29b1f58a7
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago  Running             etcd                                0
fe4b9c3d6483c
2ddc9eda16f53
9d28c15860870e85c91d0e36b45f7a6edd3da757b113ec4abb4507df88b17f06
About a minute ago  Running             etcdctl
```

이 명령의 출력이 비어 있지 않으면 몇 분 기다렸다가 다시 확인하십시오.

10. **etcd** 클러스터의 상태를 확인합니다.

a. 모든 제어 평면 호스트에서 다음 명령을 사용하여 **etcd** 클러스터의 상태를 확인하세요.

```
$ crictl exec -it $(crictl ps | grep etcdctl | awk '{print $1}') etcdctl endpoint status -w table
```

출력 예

```
+------------------------+-----------------+---------+---------+-----------+-----------+-----------
+-----------+------------------+--------+
|        ENDPOINT        |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+------------------------+-----------------+---------+---------+-----------+-----------+-----------
+-----------+------------------+--------+
| https://10.0.89.133:2379 | 682e4a83a0cec6c0 |   3.5.0 |   67 MB |      true |      false |
2 |       218 |              218 |        |
|  https://10.0.92.74:2379 | 450bcf6999538512 |   3.5.0 |   67 MB |     false |      false |
2 |       218 |              218 |        |
| https://10.0.93.129:2379 | 358efa9c1d91c3d6 |   3.5.0 |   67 MB |     false |      false |
2 |       218 |              218 |        |
+------------------------+-----------------+---------+---------+-----------+-----------+-----------
+-----------+------------------+--------+
```

11. 다른 정적 포드를 다시 시작합니다.
한 번에 하나의 호스트이지만 모든 컨트롤 플레인 호스트에서 다음 단계를 실행해야 합니다.

a. 다음 명령을 사용하여 kubelet이 관련 컨테이너를 시작하도록 하려면 Kubernetes API Server 정적 포드 매니페스트를 kubelet 매니페스트 디렉토리로 다시 이동합니다.

```
$ mv /root/manifests-backup/kube-apiserver-pod.yaml /etc/kubernetes/manifests
```

b. 모든 Kubernetes API 서버 컨테이너가 시작되었는지 확인하세요.

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

참고

다음 명령의 출력이 비어 있으면 몇 분 기다렸다가 다시 확인합니다.

c. **kube-controller-manager-pod.yaml** 및 **kube-scheduler-pod.yaml** 파일에 대해서도 동일한 단계를 반복합니다.

i. 다음 명령을 사용하여 모든 노드에서 kubelet을 다시 시작합니다.

```
$ systemctl restart kubelet
```

ii. 다음 명령을 사용하여 나머지 제어 평면 포드를 시작합니다.

```
$ mv /root/manifests-backup/kube-* /etc/kubernetes/manifests/
```

iii. **kube-apiserver** , **kube-scheduler** 및 **kube-controller-manager** 포드가 올바르게 시작되는지 확인하세요.

```
$ crictl ps | grep -E 'kube-(apiserver|scheduler|controller-manager)' | grep -v -E
'operator|guard'
```

iv. 다음 명령을 사용하여 OVN 데이터베이스를 지웁니다.

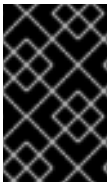
```
for NODE in  $(oc get node -o name | sed 's:node/::g')
do
  oc debug node/${NODE} -- chroot /host /bin/bash -c  'rm -f /var/lib/ovn-
ic/etc/ovn*.db && systemctl restart ovs-vswitchd ovsdb-server'
  oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-node --field-
selector=spec.nodeName=${NODE} --wait
  oc -n openshift-ovn-kubernetes wait pod -l app=ovnkube-node --field-
selector=spec.nodeName=${NODE} --for condition=ContainersReady --
timeout=600s
done
```

추가 리소스

- etcd 데이터 백업
- 베어 메탈에 사용자 프로비저닝 클러스터 설치
- 설치 관리자 프로비저닝 인프라 클러스터에서 Amazon Web Services의 호스트에 액세스
- 베어 메탈 컨트롤 플레인 노드 교체

4.3.2.5. 영구 스토리지 상태 복원을 위한 문제 및 해결 방법

OpenShift Container Platform 클러스터에서 모든 형식의 영구저장장치를 사용하는 경우 일반적으로 클러스터의 상태가 etcd 외부에 저장됩니다. etcd 백업에서 복원하면 OpenShift Container Platform의 워크로드 상태도 복원됩니다. 그러나 etcd 스냅샷이 오래된 경우 상태가 유효하지 않거나 오래되었을 수 있습니다.



중요

PV(영구 볼륨)의 내용은 etcd 스냅샷의 일부가 아닙니다. etcd 스냅샷에서 OpenShift Container Platform 클러스터를 복원할 때 중요하지 않은 워크로드가 중요한 데이터에 액세스할 수 있으며 그 반대의 경우로도 할 수 있습니다.

다음은 사용되지 않는 상태를 생성하는 몇 가지 예제 시나리오입니다.

- MySQL 데이터베이스는 PV 오브젝트에서 지원하는 pod에서 실행됩니다. etcd 스냅샷에서 OpenShift Container Platform을 복원해도 스토리지 공급자의 볼륨을 다시 가져오지 않으며 pod를 반복적으로 시작하려고 하지만 실행 중인 MySQL pod는 생성되지 않습니다. 스토리지 공급자에서 볼륨을 복원한 다음 새 볼륨을 가리키도록 PV를 편집하여 이 Pod를 수동으로 복원해야 합니다.
- Pod P1에서는 노드 X에 연결된 볼륨 A를 사용합니다. 다른 pod가 노드 Y에서 동일한 볼륨을 사용하는 동안 etcd 스냅샷을 가져오는 경우 etcd 복원이 수행되면 해당 볼륨이 여전히 Y 노드에 연결되어 있으므로 Pod P1이 제대로 시작되지 않을 수 있습니다. OpenShift Container Platform은 연결을 인식하지 못하고 자동으로 연결을 분리하지 않습니다. 이 경우 볼륨이 노드 X에 연결된 다음 Pod P1이 시작될 수 있도록 노드 Y에서 볼륨을 수동으로 분리해야 합니다.

- etcd 스냅샷을 만든 후 클라우드 공급자 또는 스토리지 공급자 인증 정보가 업데이트되었습니다. 이로 인해 해당 인증 정보를 사용하는 CSI 드라이버 또는 Operator가 작동하지 않습니다. 해당 드라이버 또는 Operator에 필요한 인증 정보를 수동으로 업데이트해야 할 수 있습니다.

- etcd 스냅샷을 만든 후 OpenShift Container Platform 노드에서 장치가 제거되거나 이름이 변경됩니다. Local Storage Operator는 **/dev/disk/by-id** 또는 **/dev** 디렉터리에서 관리하는 각 PV에 대한 심볼릭 링크를 생성합니다. 이 경우 로컬 PV가 더 이상 존재하지 않는 장치를 참조할 수 있습니다.
  이 문제를 해결하려면 관리자가 다음을 수행해야 합니다.

  1. 잘못된 장치가 있는 PV를 수동으로 제거합니다.

  2. 각 노드에서 심볼릭 링크를 제거합니다.

  3. **LocalVolume** 또는 **LocalVolumeSet** 오브젝트를 삭제합니다 (*스토리지 → 영구 스토리지 구성 → 로컬 볼륨을 사용하는 영구 스토리지 → Local Storage Operator 리소스 삭제* 참조).

4.3.3. 만료된 컨트롤 플레인 인증서 복구

클러스터는 만료된 컨트롤 플레인 인증서에서 자동으로 복구될 수 있습니다.

그러나 kubelet 인증서를 복구하려면 대기 중인 **node-bootstrapper** 인증서 서명 요청(CSR)을 수동으로 승인해야 합니다. 사용자 프로비저닝 설치의 경우 보류 중인 kubelet 서비스 CSR을 승인해야 할 수도 있습니다.

보류중인 CSR을 승인하려면 다음 단계를 수행합니다.

프로세스

1. 현재 CSR의 목록을 가져옵니다.

```
$ oc get csr
```

출력 예

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

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

2 보류 중인 **node-bootstrapper** CSR입니다.

2. CSR의 세부 사항을 검토하여 CSR이 유효한지 확인합니다.

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

```
```

1 **<csr_name>**은 현재 CSR 목록에 있는 CSR의 이름입니다.

3. 각각의 유효한 **node-bootstrapper** CSR을 승인합니다.

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

4. 사용자 프로비저닝 설치의 경우 CSR을 제공하는 각 유효한 kubelet을 승인합니다.

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

4.3.4. 복구 절차 테스트

복구 절차를 테스트하는 것은 자동화와 워크로드가 새로운 클러스터 상태를 원활하게 처리하는지 확인하는 데 중요합니다. etcd 쿼럼의 복잡한 특성과 etcd 운영자가 자동으로 복구를 시도하는 탓에 클러스터를 복구할 수 있을 만큼 손상된 상태로 올바르게 되돌리는 게 어려운 경우가 많습니다.

주의

클러스터에 SSH 액세스 권한이 있어야 **합니다**. SSH 접근이 없으면 클러스터가 완전히 손실될 수 있습니다.

사전 요구 사항

- 제어 평면 호스트에 SSH 액세스 권한이 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. SSH를 사용하여 각 비복구 노드에 연결하고 다음 명령을 실행하여 etcd와 **kubelet** 서비스를 비활성화합니다.
   a. 다음 명령을 실행하여 etcd를 비활성화합니다.

```
$ sudo /usr/local/bin/disable-etcd.sh
```

   b. 다음 명령을 실행하여 etcd의 변수 데이터를 삭제합니다.

```
$ sudo rm -rf /var/lib/etcd
```

   c. 다음 명령을 실행하여 **kubelet** 서비스를 비활성화합니다.

```
$ sudo systemctl disable kubelet.service
```

2. 모든 SSH 세션을 종료합니다.

3. 다음 명령을 실행하여 복구되지 않은 노드가 **준비되지 않음** 상태인지 확인하세요.

```
$ oc get nodes
```

4. 클러스터를 복원하려면 "이전 클러스터 상태로 복원"의 단계를 따르세요.

5. 클러스터를 복구하고 API가 응답하면 SSH를 사용하여 복구되지 않은 각 노드에 연결하고 **kubelet** 서비스를 활성화합니다.

```
$ sudo systemctl enable kubelet.service
```

6. 모든 SSH 세션을 종료합니다.

7. 다음 명령을 실행하여 노드가 **READY** 상태로 돌아오는 것을 관찰하세요.

```
$ oc get nodes
```

8. etcd를 사용할 수 있는지 확인하려면 다음 명령을 실행하세요.

```
$ oc get pods -n openshift-etcd
```

추가 리소스

- 이전 클러스터 상태로 복원

5장. ETCD 암호화 활성화

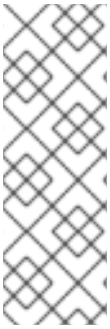
5.1. ETCD 암호화 정보

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

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

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

etcd 암호화를 활성화하면 암호화 키가 생성됩니다. etcd 백업에서 복원하려면 이 키가 있어야 합니다.



참고

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

백업 중에 etcd 암호화가 활성화된 경우 ***static_kuberesources_<datetimestamp>.tar.gz*** 파일에 etcd 스냅샷에 대한 암호화 키가 포함됩니다. 보안상의 이유로 이 파일은 etcd 스냅샷과 별도로 저장하세요. 하지만 이 파일은 해당 etcd 스냅샷에서 etcd의 이전 상태를 복원하는 데 필요합니다.

5.2. 지원되는 암호화 유형

OpenShift Container Platform에서 etcd 데이터를 암호화하는 데 지원되는 암호화 유형은 다음과 같습니다.

AES-CBC

PKCS#7 패딩과 32바이트 키를 사용하는 AES-CBC를 사용하여 암호화를 수행합니다. 암호화 키는 매주 교체됩니다.

AES-GCM

임의의 비ce 및 32바이트 키와 함께 AES-GCM을 사용하여 암호화를 수행합니다. 암호화 키는 매주 교체됩니다.

5.3. ETCD 암호화 활성화

etcd 암호화를 활성화하여 클러스터에서 중요한 리소스를 암호화할 수 있습니다.

주의

초기 암호화 프로세스가 완료될 때까지 etcd 리소스를 백업하지 마세요. 암호화 프로세스가 완료되지 않은 경우 백업은 부분적으로만 암호화될 수 있습니다.

etcd 암호화를 활성화하면 다음과 같은 여러 가지 변경 사항이 발생할 수 있습니다.

- etcd 암호화는 몇몇 리소스의 메모리 소비에 영향을 미칠 수 있습니다.
- 리더가 백업을 담당해야 하므로 백업 성능에 일시적인 영향이 있을 수 있습니다.
- 디스크 I/O는 백업 상태를 수신하는 노드에 영향을 미칠 수 있습니다.

AES-GCM 또는 AES-CBC 암호화로 etcd 데이터베이스를 암호화할 수 있습니다.

참고

etcd 데이터베이스를 한 암호화 유형에서 다른 암호화 유형으로 마이그레이션하려면 API 서버의 **spec.encryption.type** 필드를 수정하면 됩니다. etcd 데이터는 자동으로 새로운 암호화 유형으로 마이그레이션됩니다.

사전 요구 사항

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

프로세스

1. **APIServer** 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. **spec.encryption.type** 필드를 **aesgcm** 또는 **aescbc** 로 설정합니다.

```
spec:
  encryption:
    type: aesgcm (1)
```

(1) AES-GCM 암호화의 경우 **aesgcm** 으로 설정하고, AES-CBC 암호화의 경우 **aescbc로 설정**합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.
암호화 프로세스가 시작됩니다. etcd 데이터베이스 크기에 따라 이 프로세스를 완료하는 데 20분 이상 걸릴 수 있습니다.

4. etcd 암호화에 성공했는지 확인합니다.

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

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

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

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

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

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

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

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

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

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

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

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

암호화에 성공하면 출력에 **EncryptionCompleted**가 표시됩니다.

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

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

5.4. ETCD 암호화 비활성화

클러스터에서 etcd 데이터의 암호화를 비활성화할 수 있습니다.

사전 요구 사항

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

프로세스

1. **APIServer** 오브젝트를 수정합니다.

```
$ oc edit apiserver
```

2. **암호화** 필드 유형을 **identity**로 설정합니다.

```
spec:
  encryption:
    type: identity (1)
```

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

3. 파일을 저장하여 변경 사항을 적용합니다.
암호 해독 프로세스가 시작됩니다. 클러스터 크기에 따라 이 프로세스를 완료하는 데 20분 이상 걸릴 수 있습니다.

4. etcd 암호 해독에 성공했는지 확인합니다.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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