Red Hat

OpenShift Virtualization

Red Hat Ceph Storage 5 외장 스토리지를 사용한

대규모 튜닝 및 성능

표준 아키텍처

Boaz Ben Shabat

목차

대상

이 문서의 목적은 고객, 영업 엔지니어, 현장 컨설턴트, 솔루션 아키텍트를 포함한 인프라 서비스 담당자를 지원하는 것입니다.
이 문서에서는 RHCS를 고가용성(HA) 외부 네트워크 스토리지 솔루션으로 사용하여 Red Hat OpenShift® Container Platform의 기능 중 하나인 OpenShift Virtualization을 성공적으로 대규모 배포한 예시를 보여줍니다.

핵심 요약

이 문서에서는 외부 Red Hat® Ceph® Storage(RHCS) 5 클러스터(47개 노드)와 Red Hat OpenShift Virtualization(100개 노드)을 모두 통합하는 성공적인 대규모 배포에서 Red Hat OpenShift Virtualization Performance and Scale 팀이 학습한 내용을 설명합니다. 여기에서 외부 Ceph 클러스터는 OpenShift Virtualization 가상 머신(VM)에 스토리지를 제공하여 21,400개의 포드와 함께 총 3,000개의 VM을 수용합니다.

이 표준 아키텍처에서는 RHCS와 Red Hat OpenShift Virtualization을 튜닝하기 위해 수행한 단계를 살펴보고, 이 단계를 통해 복원력이 뛰어난 100개 노드 OpenShift 클러스터를 생성할 수 있습니다.

또한 이러한 단계에 대한 추론을 설명하고 이러한 권장 사항을 모든 클러스터에 적용할 수 있는 정보를 제공합니다.

다음 표에서는 모든 운영 환경에서 발생할 수 있는 가장 중요한 시나리오에 대한 성능 결과를 보여 줍니다.

시나리오	상세 정보	결과
VM 배포	최대 **800**개의 **VM** 병렬 배포	테스트 결과에 따르면 **100**개의 **VM**을 대량 복제 시 가장 빠른 배포 시간을 달성할 수 있습니다.

VM 부팅 스톰	최대 1000개 VM의 병렬 부팅 스톰	근사 선형 부팅 시간은 100개 VM의 경우 01:42(MM:SS)에 시작되었고, 1000개 VM의 경우 17:45에 종료되었습니다.
VM 대기 시간	읽기 및 쓰기 모두에 대한 워크로드 대기 시간과 비교하여 지속된 유휴 대기 시간	유휴 VM 대기 시간은 RHCS에 액세스하는 IO 스레드 수에 영향을 받지 않으며, 100만 IOPS의 경우 읽기 대기 시간은 최대 30% 감소했고 쓰기 대기 시간은 최대 88% 증가했습니다.
VM 마이그레이션	1000개 VM 마이그레이션	1000개 VM 마이그레이션 + 7000개 포드 제거에 약 118분이 걸렸습니다(HH:MM).
VM 마이그레이션 추가 대기 시간	1000개의 Red Hat Enterprise Linux®(RHEL) VM 마이그레이션(워크로드 포함)	마이그레이션 중 IO 대기 시간은 읽기의 경우 9% 증가했고 쓰기의 경우 13% 증가했습니다. 마이그레이션 시간은 3% 증가했고, 실제 IOPS 속도에는 영향을 미치지 않았습니다.
OpenShift 클러스터 업그레이드	OpenShift 클러스터 버전 업데이트	마이너 업그레이드는 35분, 메이저 업그레이드는 136분이 걸렸습니다.

소프트웨어 구성 요소

제품	버전	상세 정보
Red Hat OpenShift	**4.9.15**	어디에 배포하든 클라우드와 유사한 경험을 지원하는 최고의 엔터프라이즈급 쿠버네티스 플랫폼입니다. **Red Hat OpenShift**를 사용하면 클라우드, 온프레미스, 네트워크 엣지 등 위치와 관계없이 일관된 경험을 통해 애플리케이션을 빌드, 배포, 실행할 위치를 선택할 수 있습니다.
Red Hat Ceph Storage	**5**	간단하며 대규모 확장이 가능한 개방형 스토리지 솔루션으로 현대적인 데이터 파이프라인에 적합합니다. 데이터 분석, 인공지능/머신 러닝(AI/ML), 이머징 워크로드를 위해 설계된

		Red Hat Ceph Storage는 고객이 선택한 업계 표준 하드웨어에 기반한 소프트웨어 정의 스토리지를 제공합니다.
Red Hat OpenShift Data Foundation	4.9.2	컨테이너를 위한 소프트웨어 정의 스토리지입니다. **Red Hat OpenShift**를 위한 데이터 및 스토리지 서비스 플랫폼으로 설계된 **Red Hat OpenShift Data Foundation**은 팀이 클라우드와 베어메탈 호스트 전반에서 빠르고 효율적으로 애플리케이션을 개발하고 배포할 수 있도록 지원합니다.
Red Hat OpenShift Virtualization	4.9.2	쿠버네티스 클러스터에서 **VM**을 실행하기 위한 **Red Hat** 솔루션입니다. **OpenShift** 가상화는 두 가지 목표를 달성하도록 설정되었습니다. 첫 번째는 모든 사용자가 하나의 플랫폼에서 워크로드를 통합하여 기존 가상 머신 사용자와 **VM** 환경을 처음 접하는 사용자 모두 컨테이너 플랫폼과 추가 가상화 플랫폼을 관리하는 데 따른 운영 오버헤드를 줄이도록 지원하는 것입니다. 두 번째는 쿠버네티스 엔진과 에코시스템의 강점을 활용하여 사용자가 전통적인 워크로드 기능, 오케스트레이션, 아키텍처를 현대화할 수 있도록 지원하는 것입니다.

Red Hat

물리적 구성 요소

RHCS 클러스터

10 * DELL PowerEdge R640 랙 서버:

구성 요소	사양	설명
CPU	40개 코어	2* Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
메모리	384GB ECC RAM	12 * SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R 듀얼 랭크 x4 모듈
SSD(루트 디스크)	446.63GB - 6Gbps	MICRON SSD MTFDDAK480TDT
SSD(스토리지)	3574GB - 12Gbps	2 * TOSHIBA SSD KPM5XVUG1T92 1787.88GB
NVME(스토리지)	2980.82GB - 8GT/s	Samsung NVME S5CXNA0N607551

37 * DELL PowerEdge R650 랙 서버:

구성 요소	사양	설명
CPU	56개 코어	2 * Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz
메모리	384GB ECC RAM	12 * SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R 듀얼 랭크 x4 모듈
SSD(루트 디스크)	446.63GB - 6Gbps	MICRON SSD MTFDDAK480TDT
SSD(스토리지)	3574GB - 12Gbps	2 * TOSHIBA SSD KPM5XVUG1T92 1787.88GB
NVME(스토리지)	2980.82GB - 8GT/s	Samsung NVME S5CXNA0N607551

참고: RHCS에 사용된 하드웨어는 RHCS 클러스터 전반의 비동종 디스크 크기와 아키텍처로 인해 작업에 완벽하게 적합하지는 않았고, 이는 Ceph 성능에 영향을 미쳤습니다. 하지만, 이는 Ceph가 제공할 수 있는 다양성을 보여주는 또 다른 방증입니다.

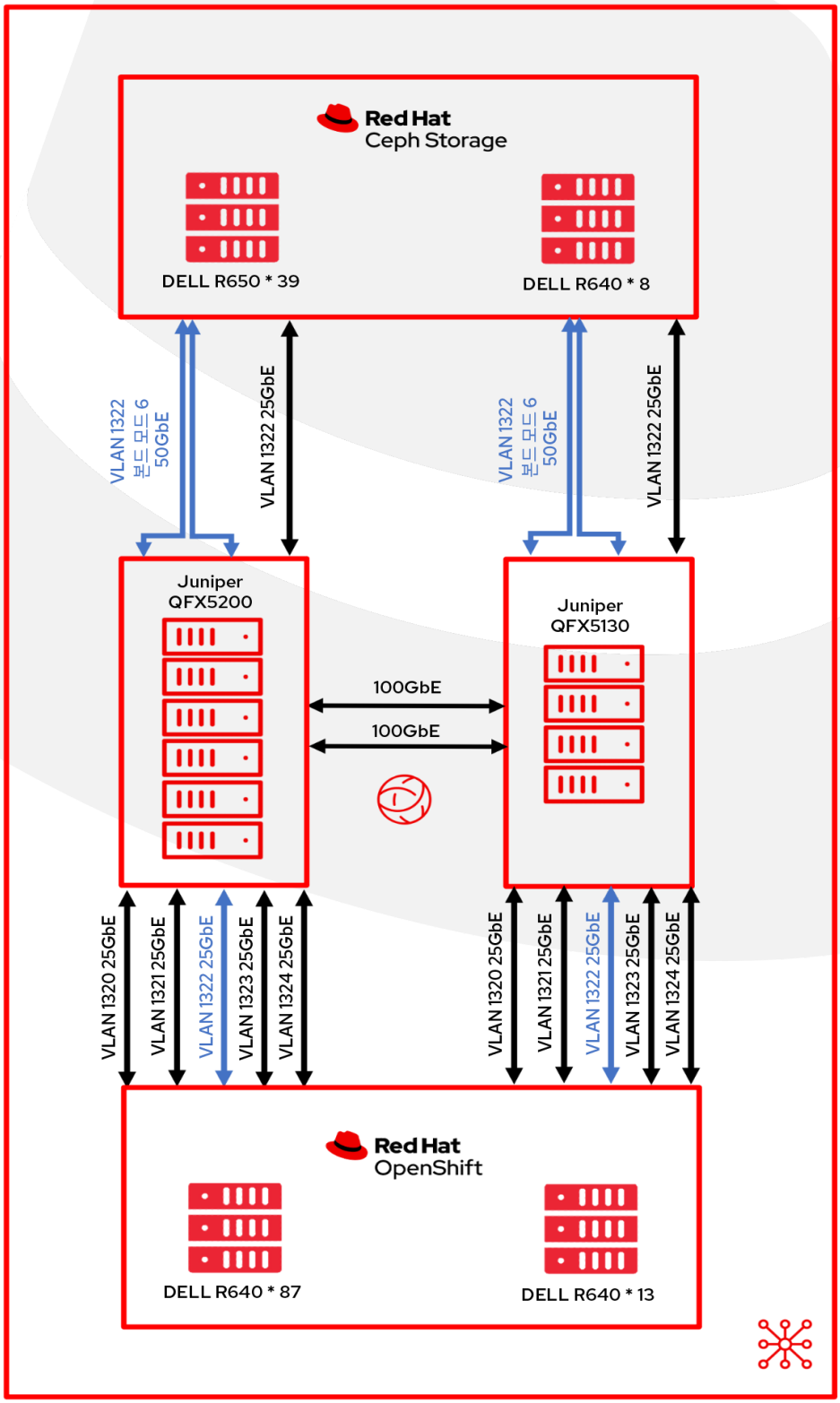
OpenShift 클러스터

100 * DELL PowerEdge R640 랙 서버:

구성 요소	사양	설명
CPU	40개 코어	2 * Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz 20개 코어
메모리	384GB ECC RAM	12 * SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R 듀얼 랭크 x4 모듈
SSD(루트 디스크)	446.63GB - 6Gbps	MICRON SSD MTFDDAK480TDT

아키텍처

이 다이어그램은 OpenShift 및 Ceph 클러스터에 대한 네트워킹 아키텍처를 보여 줍니다. 프라이빗 랩 VLAN의 Ceph와 Red Hat OpenShift Container Platform(OCP) 클러스터 간의 데이터 경로는 2 * 25GbE 포트를 포함한 balance-alb 본드를 사용합니다.



Red Hat

RHCS 네트워크 튜닝

이 섹션에서는 이러한 대규모 환경을 충족하기 위해 Ceph 노드에서 수행되는 Linux 네트워크 튜닝을 설명합니다.

Address Resolution Protocol(ARP) 튜닝

ARP 플럭스

동일한 서브넷에 여러 네트워크 인터페이스가 있는 Linux 호스트는 ARP 플럭스 문제로 인해 영향을 받을 수 있습니다. ARP 플럭스 문제는 호스트가 동일한 서브넷의 인터페이스에 대한 ARP 요청에 응답할 때 발생할 수 있습니다. 이 동작이 반드시 문제가 되는 것은 아니지만, 경우에 따라 ARP 플럭스로 인해 IPv4 주소와 MAC 주소 간 매핑이 잘못되면 일부 애플리케이션이 잘못 동작할 수 있습니다.

RHEL 기반 호스트에서는 모든 RHCS 호스트에서 /etc/sysctl.d/99.8-arp.conf를 편집하고 다음 행을 추가하여 이 동작을 수정할 수 있습니다.

```
net.ipv4.conf.all.arp_filter=1 #default value 0
net.ipv4.conf.all.arp_ignore=1 #default value 0
net.ipv4.conf.all.arp_announce=1 #default value 0
```

- **filter=1** - 이를 통해 동일한 서브넷에 여러 네트워크 인터페이스를 가질 수 있고 커널이 ARP IP에서 해당 인터페이스로 패킷을 라우팅할지 여부에 따라 각 인터페이스의 ARP에 응답할 수 있습니다(단, 이 작업을 수행하려면 소스 기반 라우팅을 사용해야 합니다). 즉, arp 요청에 응답할 카드(일반적으로 1개)를 제어할 수 있습니다.
- **ignore=1** - 대상 IP 주소가 수신 인터페이스에 구성된 로컬 주소와 일치하는 경우에만 회신합니다.
- **arp_announce=1** - 이 인터페이스의 대상 서브넷에 없는 로컬 주소는 피합니다. 이 모드는 이 인터페이스를 통해 도달할 수 있는 대상 호스트에서 수신 인터페이스에 구성된 논리적 네트워크에 속한 ARP 요청의 소스 IP 주소를 필요로 하는 경우에 유용합니다.

다음을 실행하여 새 네트워크 설정을 로드합니다.

```
$ sysctl -p /etc/sysctl.d/99.8-arp.conf
```

ARP 캐시

ARP 캐시는 IP 주소가 MAC 주소로 확인될 때 생성되는 ARP 항목 목록을 유지합니다. ARP 캐시가 모든 항목을 보유할 수 없는 대규모 경우를 방지하기 위해 **/etc/sysctl.d/99.7-arpcachesize.conf**를 편집하고 다음 행을 추가하여 ARP 캐시 크기를 늘려야 합니다.

```
net.ipv4.neigh.default.gc_thresh1 = 4096 #default value 128
net.ipv4.neigh.default.gc_thresh2 = 16384 #default value 512
net.ipv4.neigh.default.gc_thresh3 = 32768 #default value 1024
```

숫자 값은 IPv4 대상 캐시 항목에 대한 가비지 수집을 시작하는 임계값을 설정하는 것입니다. 이 값의 두 배가 되면 시스템이 새 할당을 거부합니다.

TCP/IP 튜닝

TCP 윈도우 스케일링

RHEL 기본 네트워크 설정은 일반적으로 대규모 설정에서 발견되는 대규모 병렬 작업에 대해 최적의 처리량/대기 시간 성능을 제공하지 못할 수 있습니다. Linux 네트워크 및 특정 네트워크 기기를 튜닝하여 병렬 작업 성능을 개선하는 방법은 다음과 같습니다.

고대역폭 네트워크를 더 효과적으로 사용하려면 더 큰 TCP 윈도우 크기를 사용해야 합니다.
따라서 TCP 윈도우 스케일링이 활성화되었는지 확인했습니다.
확인하는 방법은 다음과 같습니다. - **cat /proc/sys/net/ipv4/tcp_window_scaling**

```
$ sysctl -w net.ipv4.tcp_window_scaling=1
```

재부팅 시에도 지속되도록 다음과 같이 합니다.

```
$ echo "net.ipv4.tcp_window_scaling=1" >> /etc/sysctl.conf
```

버퍼 크기 튜닝

다음 단계는 소켓 "전송 버퍼 크기" 및 "수신 버퍼 크기"를 계산하는 것입니다.
일반적으로 각 소켓의 읽기/쓰기 버퍼는 최소 2개의 패킷, 기본 4개의 패킷 또는 최대 10개의 패킷을 보유할 수 있습니다. 네트워크 소켓 버퍼가 너무 작으면 버퍼가 가득 차서 유효 처리량이 감소하고, 이는 성능에 영향을 줄 수 있습니다. 네트워크 소켓 버퍼를 충분히 크게 설정하면 성능을 어느 정도 개선할 수 있습니다.

하지만 먼저 몇 가지 용어를 살펴보겠습니다.

- rmem_max - 최대 수신 버퍼 크기
- wmem_max - 최대 전송 버퍼 크기
- wmem_default - 기본 전송 버퍼 크기
- max_backlog - 수신 대기열의 최대 크기
- Netdev_budget - 한 번의 폴링 주기에서 모든 인터페이스에서 가져온 최대 패킷 수

최적의 버퍼 크기를 계산하기 위해 패킷 크기 방법을 사용했지만 대기 시간이 긴 설정의 경우 대기 시간 방법을 사용하는 것이 좋습니다.

대기 시간 방식

대기 시간을 사용하여 단일 TCP 연결의 최대 처리량을 계산하여 최적화합니다.

최적 크기 = (왕복 대기 시간(마이크로초)) x (링크 크기(Mb/s)) x 1024^2

예를 들어, 본드는 50000Mb/s로 실행됩니다. Ceph 노드에서 OpenShift 클러스터까지의 대기 시간은 0.208이며, 이를 마이크로초 단위로 변환하기 위해 1000으로 나눕니다. 그런 다음 50000을 곱하여 10.4가 되며, 이를 바이트 단위로 변환하면 10905190입니다.

또는

```
ping -Ibond0 -c 60 -q 192.168.216.90|grep avg|awk -F"/" '{printf "%f", ($5
/1000) * 50000 * 1024^2}'
```

이제 wmem_default를 설정하면 됩니다. 이 값은 10905190의 ¼를 의미하는 4개의 패킷을 포함하며, 다음과 같이 설정합니다.

```
net.core.rmem_max=10905190 #default value 212992
net.core.wmem_max=10905190 #default value 212992
net.core.wmem_default=4362076 #default value 212992
```

패킷 크기 방법

패킷 크기별 최적화 - 파일당 평균 패킷 크기를 512KB로 가정하고 각 소켓의 최적 크기는 다음과 같습니다. 최대 크기 = (패킷 크기(MB/s)) x 1024^2.

```
net.core.rmem_max=5242880 #default value 212992
net.core.wmem_max=5242880 #default value 212992
net.core.wmem_default=2097152 #default value 212992
```

정기적으로 긴 대기 시간을 경험하는 네트워크에서는 대기 시간에 따라 최적화하는 방법을 선호합니다.

어댑터 튜닝

NIC 버퍼

호스트가 여러 개 포함된 대규모 설정에서는 수신 트래픽 속도가 커널의 버퍼를 충분히 빠른 속도로 소진하는 능력을 초과할 수 있습니다. 이 경우 NIC 버퍼가 오버플로우되고, 트래픽이 손실되어 softirq 누락으로 계산됩니다. 이 시나리오를 방지하기 위해 softirq의 CPU 시간을 늘릴 수 있는데 이를 netdev_budget이라고 하며, 필요에 따라 예산을 늘릴 수 있습니다. 이 문서의 설정에서는 예산을 1000으로 늘렸으며, 이는 softirq가 CPU에서 벗어나기 전에 NIC의 메시지 1000개를 제거한다는 의미입니다.

```
net.core.netdev_budget=1000 #default value 300
```

시간이 지남에 따라 **/proc/net/softnet_stat**의 세 번째 열이 점진적으로 증가하는 경우:

```
01877e29 00000000 00000022 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005d
0c4a6107 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005e
01d05820 00000000 00000012 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005f
092b933a 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000060
```

이는 softirq가 충분한 CPU 시간을 얻지 못했음을 나타냅니다. 이 경우 예산을 증가시킬 수 있으며, 가능하면 작은 단위로 증가시키는 것이 좋습니다.

백로그 대기열

Linux 커널 내에는 트래픽이 NIC에서 수신된 후 프로토콜 스택(TCP/IP/ISCSI) 중 하나에서 처리되기 전에 트래픽이 저장되는 대기열이 있습니다.
각 CPU 코어에는 트래픽이 저장되는 백로그 대기열이 있습니다. 대기열이 이미 최대 용량에 도달한 경우에는 추가 패킷이 삭제됩니다.

시간이 지남에 따라 **/proc/net/softnet_stat**의 두 번째 열이 점진적으로 증가하는 경우:

```
04f88d2c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
023a354d 00000000 00000018 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
10df99e1 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000002
01ba2dec 00000000 00000011 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000003
```

이는 netdev 백로그 대기열이 오버플로우되고 netdev_max_backlog를 다시 증가시켜야 합니다. 이 경우에도 가능하면 작은 단위로 증가시키는 것이 좋습니다.

이 스케일 설정에서는 값을 5000으로 설정합니다.

```
net.core.netdev_max_backlog=5000 #default value 1000
```

Red Hat

RHCS 튜닝

이 섹션에서는 이러한 대규모 환경을 충족하기 위해 Ceph 노드에서 수행되는 Ceph 전용 네트워크 튜닝을 설명합니다.

배치 그룹(PG) 튜닝

PG는 OSD(오브젝트 스토리지 디바이스)에 의해 복제되는 오브젝트 모음이며, 각 오브젝트는 데이터와 메타데이터를 저장하기 위한 컨테이너입니다.

풀당 최적의 PG 수를 얻으려면 대상을 OSD당 100PG(rbd 및 libados의 모범 사례에 따라)로 설정할 수 있습니다. 그런 다음 풀의 최대 사용 용량(기본값은 85%)을 곱하고, 복제본 수로 나눈 다음, 가장 가까운 2의 거듭제곱으로 반올림합니다. - 2^(round(log2(x))).

```
( OSD당 대상 PG) x ( OSD ) x ( %데이터 (풀 최대 사용량))
-------------------------------------------------------------------------
                              ( 3개의 복제본 )
```

또는 이 설정에서는 (100 * 141 * 0.85) / 3 = 3995를 2의 거듭제곱으로 반올림하여 총 4096PG가 됩니다.

기본 계산기(bc)를 사용하여 스크립트를 작성했습니다.

```
$ echo "x=l(100*141*0.85/3)/l(2); scale=0; 2^((x+0.5)/1)" | bc -l
```

OSD당 PG 수를 더욱 늘릴 수 있으므로 클러스터 전체에서 OSD당 부하 편차를 줄일 수 있지만, 각 PG에는 해당 PG를 저장하는 OSD에 CPU와 메모리가 조금 더 필요합니다. 따라서 환경에 맞게 OSD 수를 테스트하고 튜닝해야 합니다.

다음 단계는 이 설정을 클러스터에 적용하는 것입니다.

```
$ ceph osd pool set pool_name pg_autoscaler_mode off
$ ceph osd pool set pool_name pg_num 4096
```

Red Hat

Prometheus 튜닝

Ceph 클러스터를 모니터링하기 위해 Ceph 대시보드를 사용하여 Ceph 풀 통계를 표시할 수 있습니다. 다음을 실행합니다.

```
$ ceph config set mgr mgr/prometheus/rbd_stats_pools pool_name
```

대규모 클러스터의 시스템 부하를 줄이기 위해 다음과 같이 풀 통계 수집을 조정할 수 있습니다.

```
$ ceph config set mgr mgr/prometheus/rbd_stats_pools_refresh_interval 600
#Default value 300
```

또한 Ceph 관리자가 장애물이 되어 다른 ceph-mgr 플러그인을 실행할 시간을 얻지 못하게 할 수 있는 현상을 방지하기 위해 Prometheus의 폴링 속도를 낮추는 것이 좋습니다.

이 경우 커맨드는 스크레이프 간격을 60초로 설정합니다.

```
$ ceph config set mgr mgr/prometheus/scrape_interval 60 #Default value 15
```

Red Hat

OpenShift Virtualization

소개

대규모로 OpenShift Virtualization 기능과 안정성을 시연하기 위해 다음 워크플로우를 시연하겠습니다.

- VM 배포
- VM 부팅 스톰
- 워크로드 포함/비포함 상태의 VM 추가 대기 시간
- 워크로드 포함/비포함 상태의 VM 마이그레이션

이 설정의 밀도 목표는 클러스터 전체에서 3000개의 VM과 21,400개의 포드로 설정되었습니다. 이 목표를 실현한 구성은 다음과 같습니다.

- 1,500개의 RHEL 8.5 퍼시스턴트 스토리지 VM
- 500개의 Windows10 퍼시스턴트 스토리지 VM
- 1,000개의 Fedora 일회성 스토리지 VM
- 21,400개의 유휴 포드

간단히 말하면, 노드당 30개의 VM과 214개의 포드 밀도입니다.

KubeletConfig

소개에서 언급한 확장성을 확보하기 위해 다음 KubeletConfig를 적용하여 노드당 포드 수의 기본 제한을 우회해야 했습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled
  kubeletConfig:
    maxPods: 500
    kubeAPIBurst: 200
    kubeAPIQPS: 100
```

`maxPods`를 500(기본값 250)으로 늘린 것 외에도 더 높은 버스팅 잠재력을 수용하기 위해 기본 `kubeAPIBurst`를 200(기본값 100)으로, `kubeAPIQPS`를 100(기본값 50)으로 늘렸습니다. 일반적인 비교를 위해 표준 쿠버네티스의 기본 최대 포드 수는 노드당 110개입니다.

위의 변경 사항 중 어느 것도 필수적이지는 않습니다. 장기 테스트를 수행하지 않았기 때문에 노드당 250개의 포드를 초과하는 것은 현재 권장되지 않습니다. 그러나 테스트를 수행하는 동안 밀도 관련 문제는 발생하지 않았습니다.

적용한 추가 Kubeletconfig는 BZ#1984442와 관련이 있으며, 이를 통해 모든 노드에서 VM 포드를 균등하게 분배할 수 있습니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-scheduling
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled
  kubeletConfig:
    nodeStatusMaxImages: -1
```

레이블을 사용하여 작업자 노드에 대해 두 가지 사용자 정의 Kubeletconfigs를 모두 활성화할 수 있습니다.

```
oc label machineconfigpool worker custom-kubelet=enable
```

KubeletConfig가 수정되면 연결된 노드가 재부팅됩니다.

템플릿

사용한 모든 OS 템플릿은 OpenShift Virtualization 템플릿 마법사를 통해 사용할 수 있는 기본 템플릿이며, 사용자 정의 네트워크가 일부 변경되었습니다.

Red Hat Linux

사용 중인 템플릿은 다음을 통해 검색할 수 있습니다.

```
oc get templates -n openshift rhel8-server-medium -o yaml
```

여기에 복사된 내용은 완전성을 위해 변경 사항을 포함합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: node-os-vm
  name: node-os-vm
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm:
    spec:
      terminationGracePeriodSeconds: 60
      evictionStrategy: LiveMigrate
      domain:
        cpu:
          cores: 1
          model: host-passthrough
          sockets: 1
          threads: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name:
          interfaces:
          - bridge: {}
            model: virtio
            name: nic-0
          networkInterfaceMultiqueue: true
          rng: {}
        machine:
          type: pc-q35-rhel8.4.0
        resources:
          requests:
            cpu: "1"
            memory: 4G
      networks:
      - multus:
         networkName: linux-bridge
```

```
        name: nic-0
      volumes:
      - dataVolume:
          name:
        name:
  dataVolumeTemplates:
  - metadata:
      annotations
      name:
    spec:
      pvc:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 40Gi
        volumeMode: Block
        storageClassName: ocs-external-storagecluster-ceph-rbd
      source:
        pvc:
          namespace: "default"
          name: "rhel-dv"
```

Fedora

사용 중인 템플릿은 다음을 통해 검색할 수 있습니다.

```
oc get templates -n openshift fedora-desktop-medium -o yaml
```

여기에 복사된 내용은 완전성을 위해 변경 사항을 포함합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    app:
    kubevirt-vm:
  name:
spec:
  annotations:
    descheduler.alpha.kubernetes.io/evict: "true"
    kubevirt.io/provisionOnNode:
  terminationGracePeriodSeconds: 0
  evictionStrategy: Restart
  running: true
  template:
    metadata:
      labels:
        kubevirt-vm: node-os-vm
    spec:
      domain:
        cpu:
          cores: 1
```

```
          sockets: 1
          threads: 1
        devices:
          disks:
            - disk:
                bus: virtio
              name: containerdisk
            - disk:
                bus: virtio
              name: cloudinitdisk
        machine:
          type: pc-q35-rhel8.4.0
        resources:
          requests:
            memory: 256Mi
            cpu: 100m
          limits:
            cpu: 100m
      terminationGracePeriodSeconds: 0
      volumes:
        - containerDisk:
            image: quay.io/kubevirt/fedora-container-disk-images:35
          name: containerdisk
        - cloudInitNoCloud:
            userData: |-
              #cloud-config
              Password: "password"
              chpasswd: { expire: False }
              runcmd:
                - sed -i -e "s/PasswordAuthentication.*/PasswordAuthentication yes/" /etc/ssh/sshd_config
                - systemctl restart sshd
          name: cloudinitdisk
status: {}
```

Windows

사용 중인 템플릿은 다음을 통해 검색할 수 있습니다.

```
oc get templates -n openshift windows10-desktop-medium -o yaml
```

여기에 복사된 내용은 완전성을 위해 변경 사항을 포함합니다.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm:
  name:
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm:
    spec:
```

```
    terminationGracePeriodSeconds: 0
    evictionStrategy: LiveMigrate
    domain:
      clock:
        timer:
          hpet:
            present: false
          hyperv: {}
          pit:
            tickPolicy: delay
          rtc:
            tickPolicy: catchup
        utc: {}
      cpu:
        cores: 1
        model: host-passthrough
        sockets: 1
        threads: 1
      devices:
        blockMultiQueue: false
        disks:
        - disk:
            bus: virtio
          name:
        interfaces:
        - bridge: {}
          model: virtio
          name: nic-0
      features:
        acpi: {}
        apic: {}
        hyperv:
          frequencies: {}
          ipi: {}
          reenlightenment: {}
          relaxed: {}
          reset: {}
          runtime: {}
          spinlocks:
            spinlocks: 8191
          synic: {}
          synictimer:
            direct: {}
          vapic: {}
          vpindex: {}
      machine:
        type: pc-q35-rhel8.4.0
      resources:
        requests:
          cpu: "1"
          memory: 4G
        limits:
    networks:
    - multus:
        networkName: linux-bridge
      name: nic-0
    volumes:
    - dataVolume:
        name:
      name:
dataVolumeTemplates:
- metadata:
    annotations:
    name:
```

```
  spec:
    pvc:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 40Gi
      volumeMode: Block
      storageClassName: ocs-external-storagecluster-ceph-rbd
    source:
      pvc:
        namespace: "default"
        name: "win10-dv"
```

포드

```
kind: Pod
apiVersion: v1
metadata:
  name: vdpod-pod-name
  namespace: pods-space
  labels:
    name: vdpod-density
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  restartPolicy: "Always"
  containers:
  - name: vdpod-pod-name
    image: gcr.io/google_containers/pause-amd64:3.0
    ports:
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: false
```

VM 배포

VM 배포는 모든 가상 환경의 기본입니다. 대규모를 목표로 할 때 대량의 VM을 더 빠르게 배포할수록 운영 효율성에 직접적인 영향을 미칩니다.

이 섹션에서는 Ceph CSI 복제 방법을 사용하여 골든 이미지 소스에서 여러 VM 이미지를 복제할 때 어떤 성능을 기대할 수 있는지 보여 줍니다.

CSI-clone 복제 전략을 사용하여 VM을 100개 이상 복제하는 경우 Ceph CSI가 복제본을 제거하지 않을 수 있으며, 복제본을 수동으로 삭제하지 못할 수도 있습니다(BZ#2055595). 따라서 현재는 스냅샷 복제를 방지하고 `cloneStrategy: copy`를 대신 사용하는 것이 가장 좋습니다.

CSI 스냅샷 복제를 활성화하려면 OpenShift Virtualization 스토리지 프로필을 편집해야 합니다.

```
oc edit -n openshift-cnv storageprofile <storage class name>
```

그리고 다음 사양을 추가합니다.

```
spec:
  cloneStrategy: csi-clone
```

우선 호스트 중 하나에서 데이터 볼륨(DV)으로 RHEL QCOW 이미지를 가져옵니다.

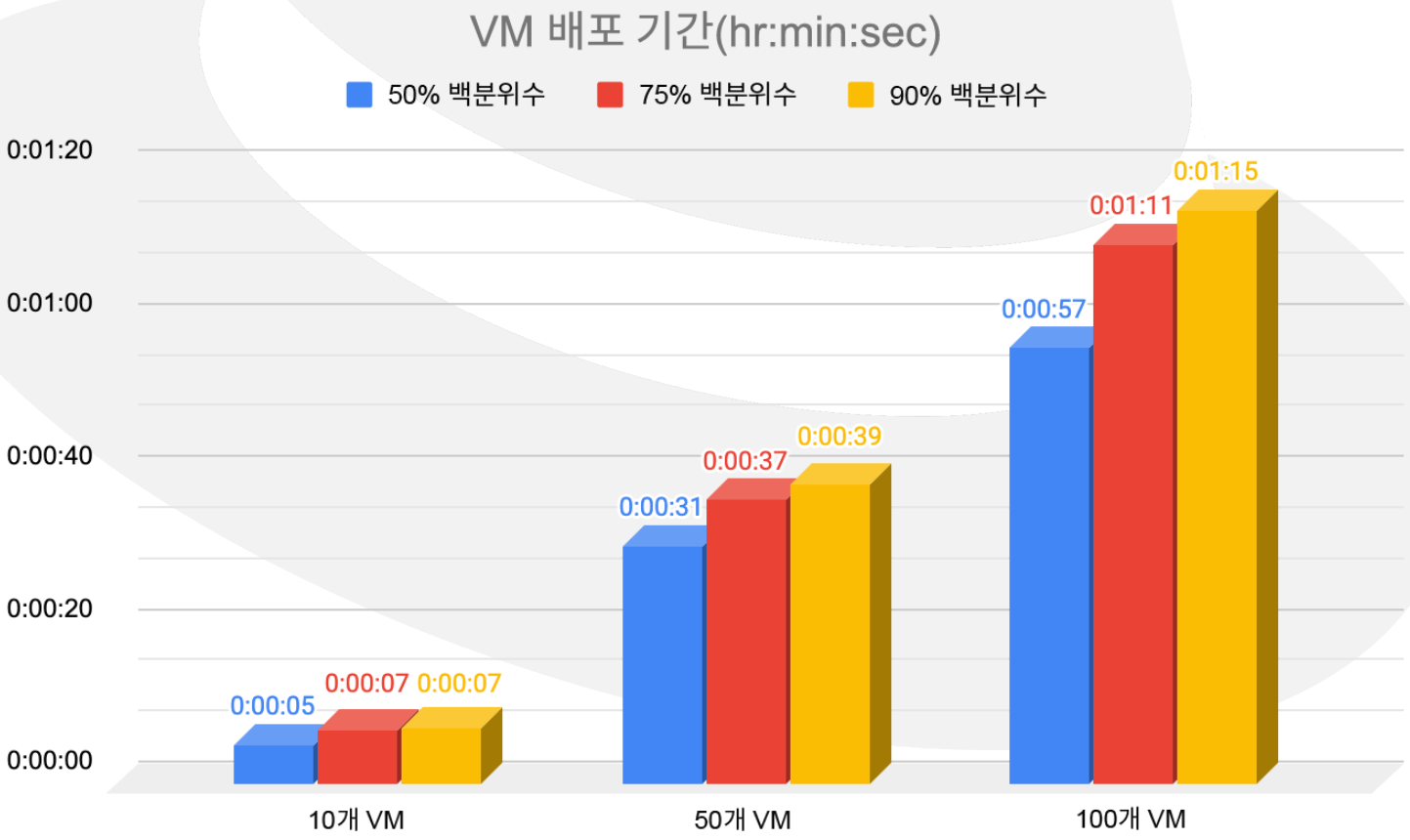
```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel-clone-dv
spec:
  source:
      http:
         url: http://internel.server.com/ISO/rhel8.qcow2
  pvc:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    volumeMode: Block
    storageClassName: ocs-external-storagecluster-ceph-rbd
```

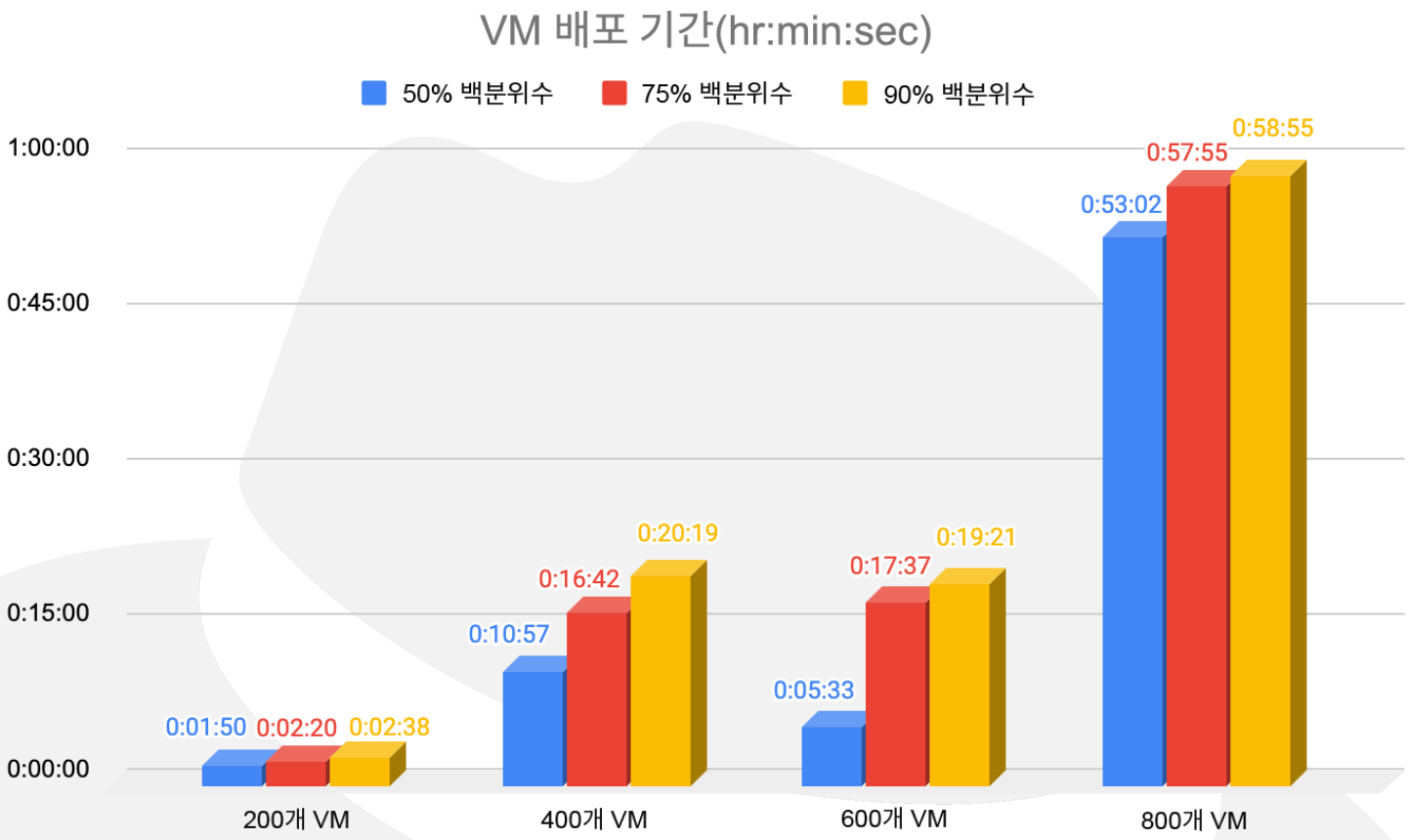
가져오기가 완료되면 원하는 수의 VM을 병렬로 배포하고 각 VM이 복제를 완료하는 데 걸리는 시간을 측정합니다. 복제가 완료될 때까지 2초 간격으로 모든 VM을 쿼리하여 이 작업을 수행합니다.

VM을 배포하는 가장 효과적인 방법은 100개씩 그룹으로 배포하는 것입니다. 즉, 100개의 VM을 배포하고 복제가 완료될 때까지 기다린 후 다음 100개의 VM을 배포합니다.

일반적으로 10개를 초과하는 VM을 병렬로 배포하면 Ceph CSI 수준에서 상위 이미지 vol-id에 대한 잠금을 획득하기 때문에 페널티가 발생합니다. 따라서 동일한 상위 이미지로부터의 복제는 병렬이 아니라 실제로는 직렬로 처리되며, 외부 프로비저닝 담당자는 언제든지 CSI 드라이버에 10gRPC 병렬 호출만 전송할 수 있습니다.

또한 복제본이 250개 이상이면 rbd 이미지가 평면화되기 시작하여 복제 페널티가 훨씬 더 증가할 것입니다. 복제본을 평면화하는 데 걸리는 시간은 스냅샷 크기에 따라 증가합니다.





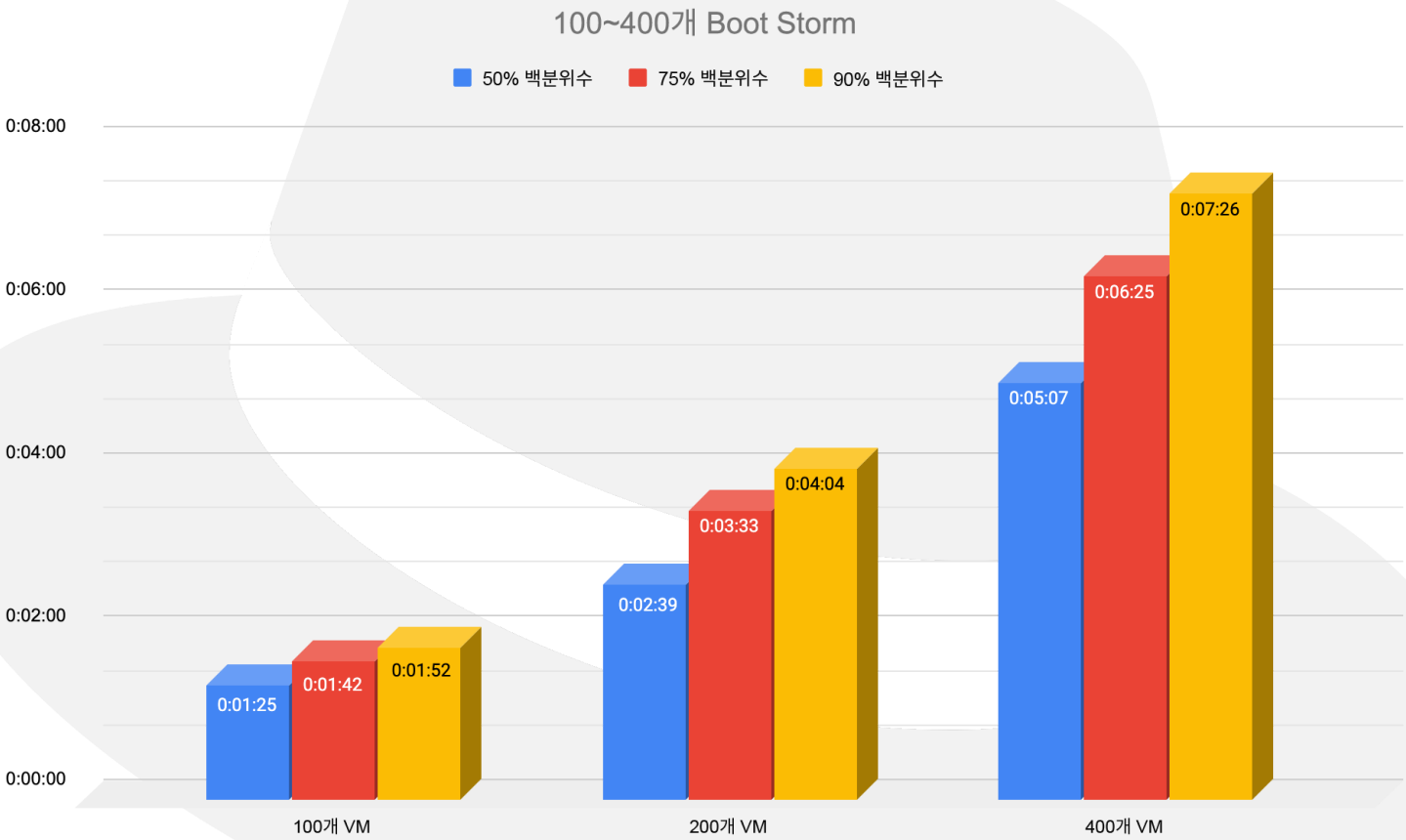
VM 부팅 스톰

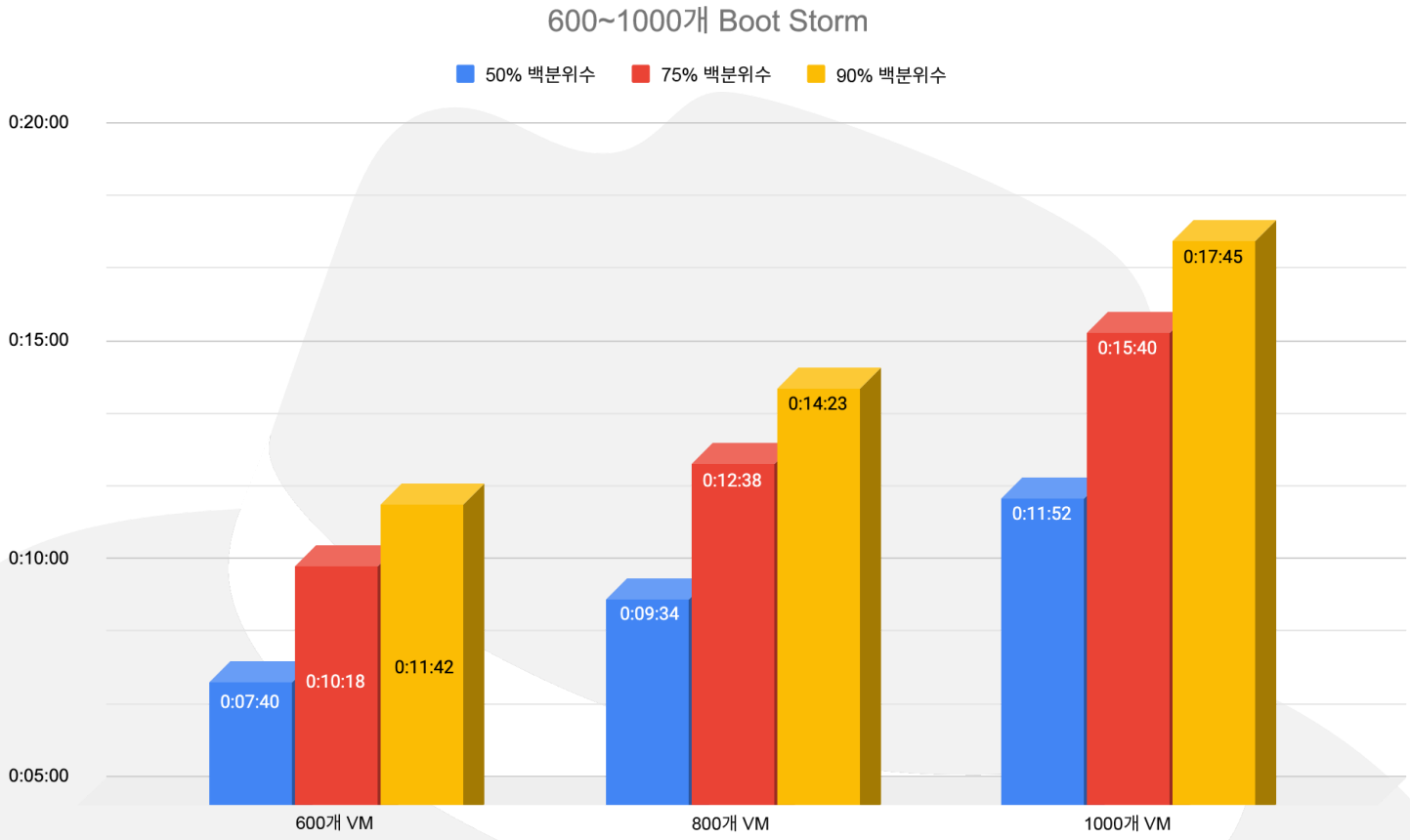
이 시나리오에서는 많은 수의 VM을 부팅하는 데 걸리는 시간을 테스트했습니다. 이를 통해 OpenShift Virtualization와 컨트롤 플레인의 복원력을 보여 줍니다. 이 시나리오는 정전과 같은 재해 상황에서 환경을 복구할 때 자주 발생하는 시나리오입니다.

측정은 모든 VM에 대해 요청 시간에 시작하여 VM이 실행 중이고 SSH 액세스를 통해 액세스 가능한 상태가 될 때(즉, 호스트가 성공적으로 부팅되고 SSH 데몬이 작동 중임을 의미) 중단됩니다. 여기에서는 각 VM에 대해 쿼리를 실행하여 시간을 측정했습니다. 상태가 "실행 중"으로 변경되면 SSH 연결이 성공할 때까지 2초마다 VM에 SSH 연결을 시도합니다.

배포 시나리오와 마찬가지로 모든 VM 시작 요청은 병렬로 실행되어 클러스터의 모든 구성 요소에 부담을 줍니다.

아래 차트에서 볼 수 있듯이, 동종 OCP 클러스터의 경우 부팅 시간이 거의 선형적으로 VM 1000개까지 증가하지만 대기열이 클수록 부팅 시간이 느려질 수 있습니다.




600~1000개 Boot Storm
50% 백분위수
75% 백분위수
90% 백분위수
0:20:00
0:15:00
0:10:00
0:05:00
0:07:40
0:10:18
0:11:42
0:09:34
0:12:38
0:14:23
0:11:52
0:15:40
0:17:45
600개 VM
800개 VM
1000개 VM

Red Hat

VM 대기 시간

각 VM에는 IO 처리를 위한 자체 IO 스레드가 있습니다. 단, 여러 개의 퍼시스턴트 볼륨 클레임(PVC)이 있고 `dedicatedIOThread:` 가 true로 설정된 경우 각 PVC는 자체 IO 스레드를 가집니다.

다음 시나리오에서는 작업자 노드당 15개의 VM을 사용하여 최대 64개의 작업자 노드 또는 960개의 VM을 테스트했으며, 여러 개의 동시 스레드가 RHCS 클러스터에 액세스하더라도 그 스레드 자체로는 대기 시간에 영향을 미치지 않는다는 것을 보여주었습니다.

이 시나리오를 위해 랜덤 읽기와 랜덤 쓰기 모두에 4KB 블록 크기를 사용하기로 선택하고 다음 테스트를 실행했습니다.

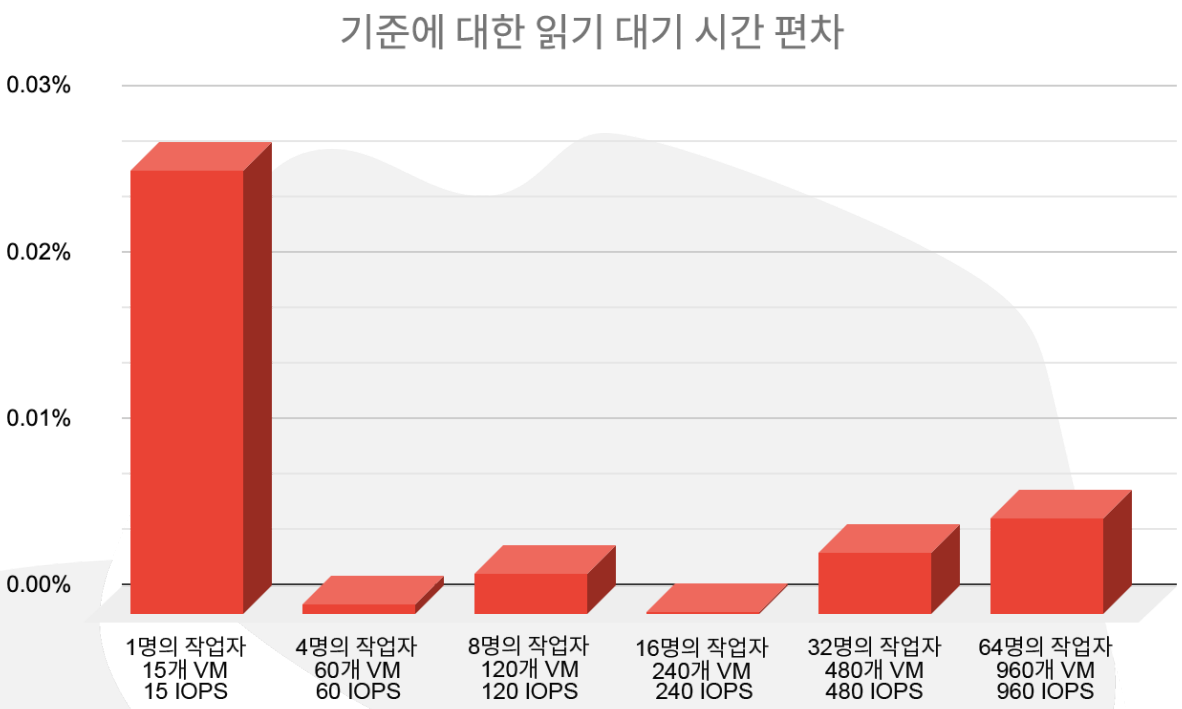
- 기준 - 모든 작업자 노드에서 15개의 VM을 사용했으며, 각 VM은 15개의 VM(15 IOPS, 단일 노드)에서 시작하고 64개의 노드에서 종료하는 단일 IOPS를 생성하여 총 960개의 VM(960 IOPS)을 생성했습니다.
- 워크로드 - 모든 작업자 노드에서 정확히 15개의 VM을 사용했으며, 각 VM은 15개의 VM(15,000 IOPS, 단일 노드)에서 시작하고 64개의 노드에서 종료하는 1000 IOPS를 생성하여 총 960개의 VM(960,000 IOPS)을 생성했습니다.

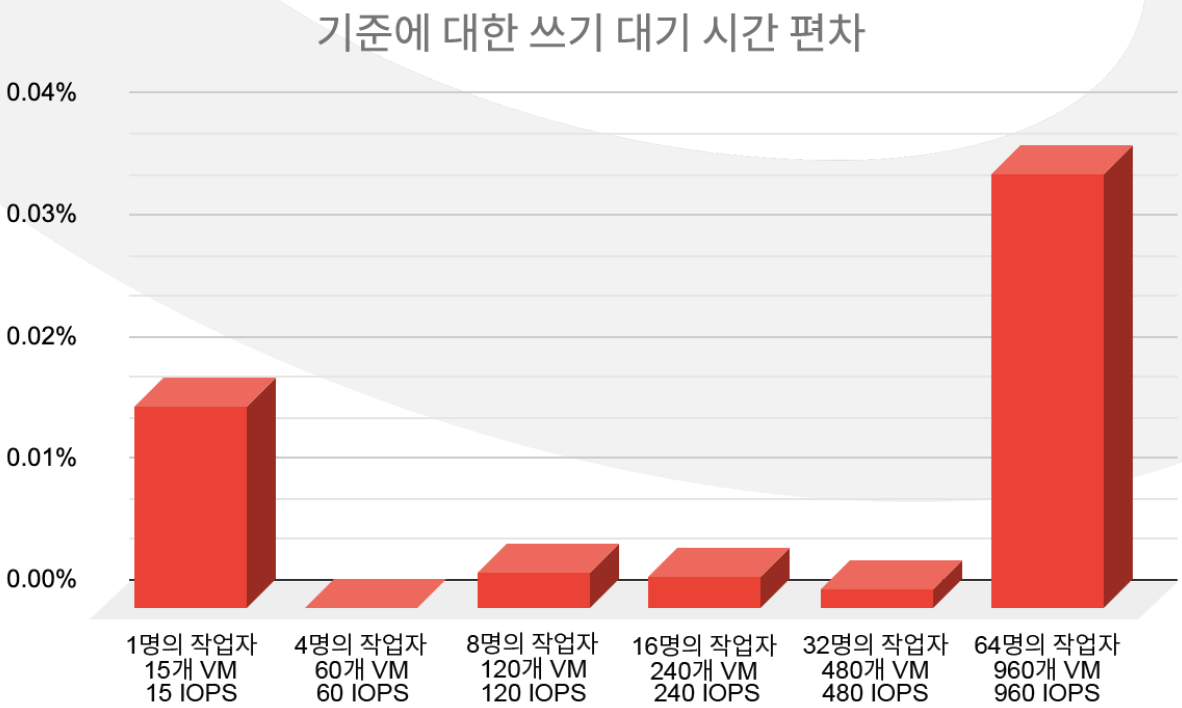
각 VM 파일 시스템 데이터세트는 300개의 디렉터리로 구성되고, 각 디렉터리에는 8개의 파일이 있으며, 각 파일의 크기는 20MiB입니다. 즉, 각 VM에는 4.8GiB 데이터세트가 있습니다.

RHCS 클러스터의 네트워킹 및 Ceph 비동종 디스크로 인해 발생할 수 있는 불일치를 최대한 방지하기 위해 작은 블록을 선택했습니다.

테스트에서 최대 960개의 VM을 사용했지만 실제로는 총 3,000개의 VM과 21,400개의 포드가 클러스터에서 실행되었습니다.

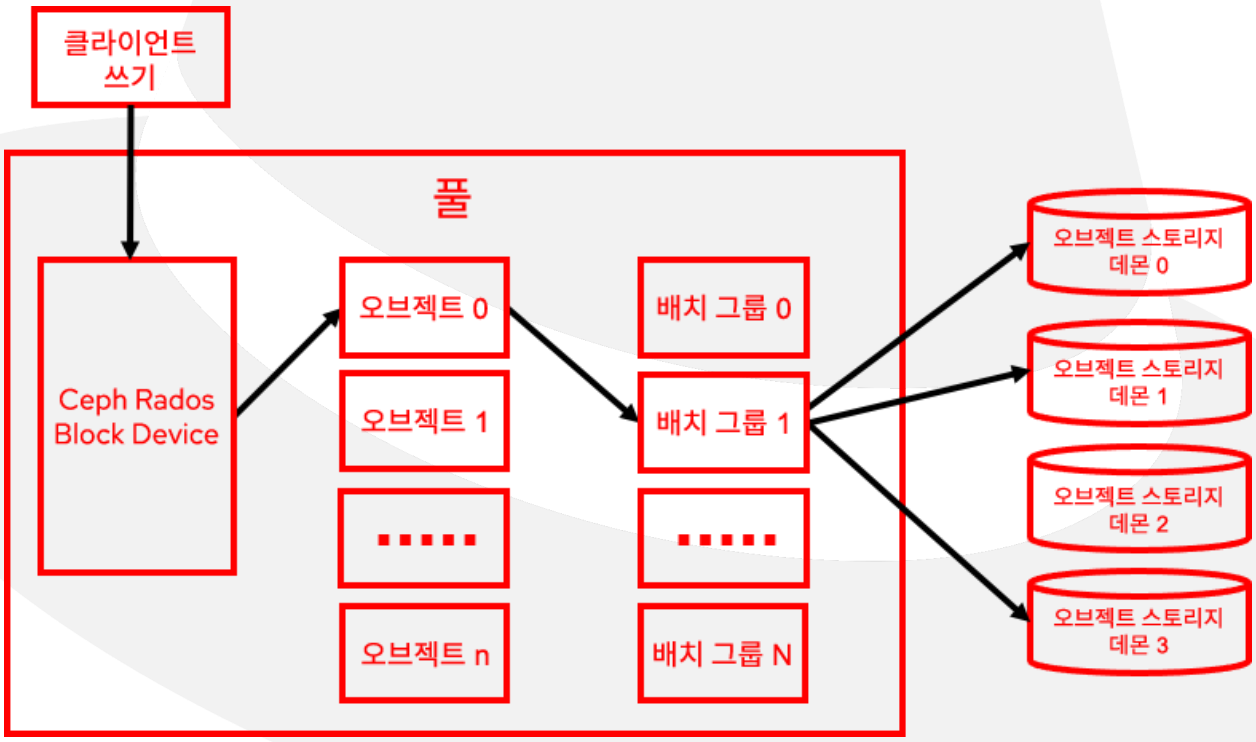
아래의 두 차트에서 볼 수 있듯이, 기준 테스트를 실행하는 동안 랜덤 읽기와 랜덤 쓰기의 대기 시간 변화는 0.04% 미만이었습니다.


기준에 대한 읽기 대기 시간 편차
0.03%
0.02%
0.01%
0.00%
1명의 작업자
15개 VM
15 IOPS
4명의 작업자
60개 VM
60 IOPS
8명의 작업자
120개 VM
120 IOPS
16명의 작업자
240개 VM
240 IOPS
32명의 작업자
480개 VM
480 IOPS
64명의 작업자
960개 VM
960 IOPS

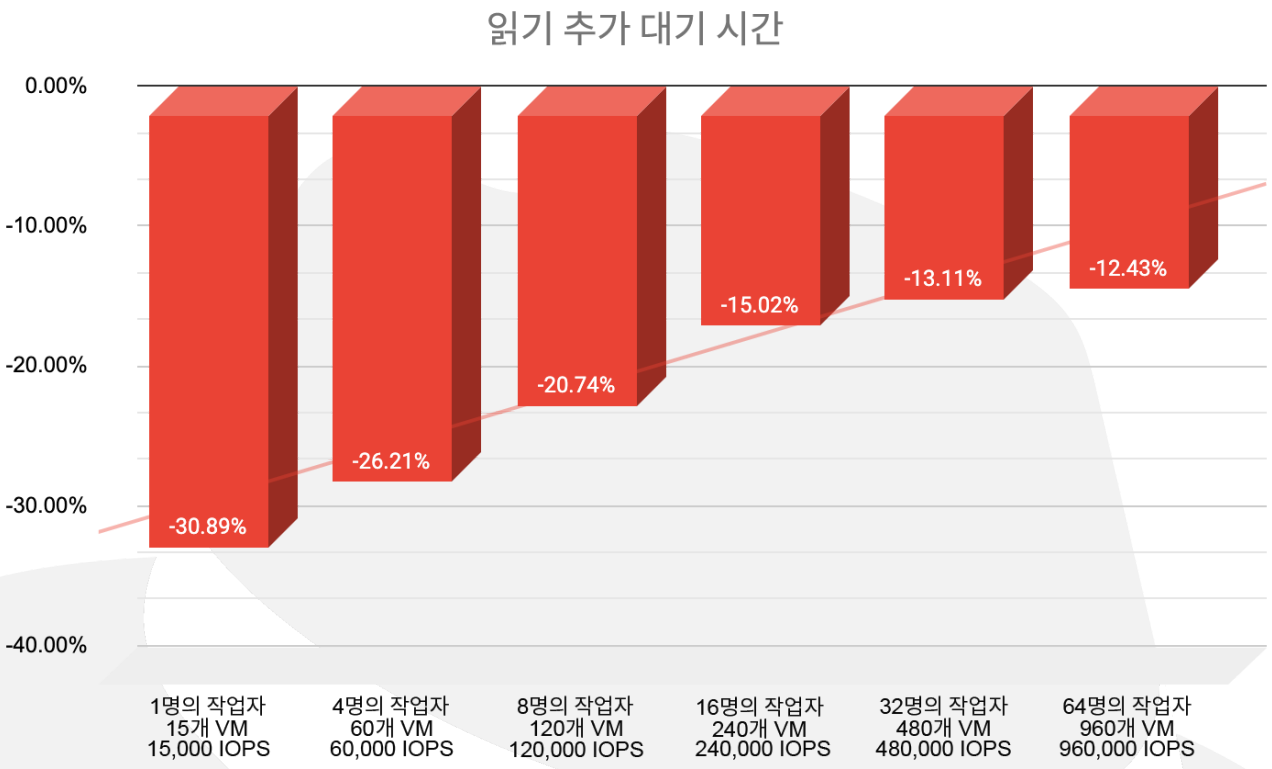

기준에 대한 쓰기 대기 시간 편차
0.04%
0.03%
0.02%
0.01%
0.00%
1명의 작업자
15개 VM
15 IOPS
4명의 작업자
60개 VM
60 IOPS
8명의 작업자
120개 VM
120 IOPS
16명의 작업자
240개 VM
240 IOPS
32명의 작업자
480개 VM
480 IOPS
64명의 작업자
960개 VM
960 IOPS

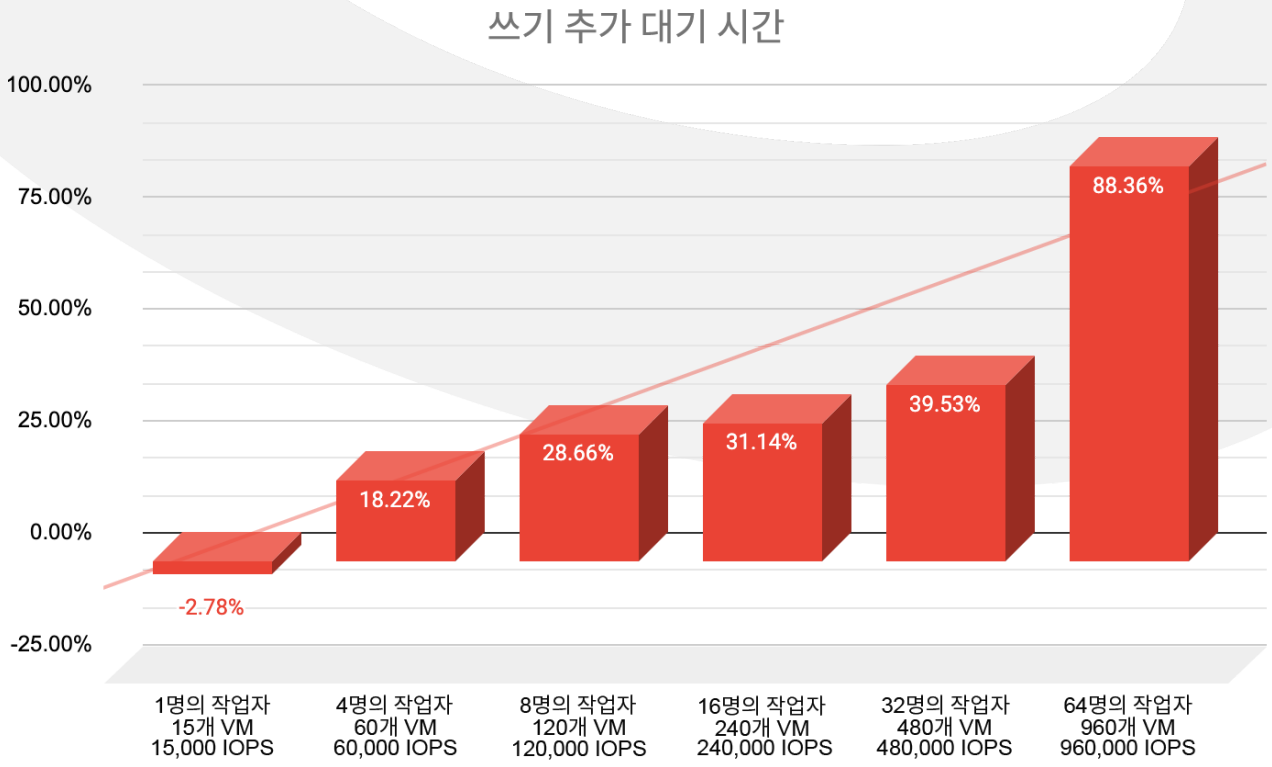
아래 차트는 기준 결과와 비교하여 워크로드 시나리오의 대기 시간 추세를 보여 줍니다(낮을수록 좋음). 읽기 성능의 경우 IOPS 속도가 더 높으면 대기 시간이 어느 정도 단축되는데, 이는 VM의 유휴/낮은 IOPS 워크로드와 비교할 때 더 높은 버스팅 동안 발생하는 리소스 할당으로 인해 발생합니다.

그러나 쓰기는 고가용성을 유지하기 위해 다른 데이터 흐름을 가집니다. 다이어그램에서 볼 수 있듯이, Ceph가 생성하는 각 쓰기에 대해 3개의 데이터 복사본이 네트워크를 통해 각 OSD로 이동합니다. 그러므로 쓰기 대기 시간이 영향을 받습니다.



대기 시간에 민감한 애플리케이션의 경우 모든 데이터 복제본에서 쓰기 대기 시간 오버헤드를 1/3로 줄일 수 있습니다.


읽기 추가 대기 시간
0.00%
-10.00%
-20.00%
-30.00%
-40.00%
-30.89%
-26.21%
-20.74%
-15.02%
-13.11%
-12.43%
1명의 작업자
15개 VM
15,000 IOPS
4명의 작업자
60개 VM
60,000 IOPS
8명의 작업자
120개 VM
120,000 IOPS
16명의 작업자
240개 VM
240,000 IOPS
32명의 작업자
480개 VM
480,000 IOPS
64명의 작업자
960개 VM
960,000 IOPS


쓰기 추가 대기 시간
100.00%
75.00%
50.00%
25.00%
0.00%
-25.00%
-2.78%
18.22%
28.66%
31.14%
39.53%
88.36%
1명의 작업자
15개 VM
15,000 IOPS
4명의 작업자
60개 VM
60,000 IOPS
8명의 작업자
120개 VM
120,000 IOPS
16명의 작업자
240개 VM
240,000 IOPS
32명의 작업자
480개 VM
480,000 IOPS
64명의 작업자
960개 VM
960,000 IOPS

Red Hat

VM 마이그레이션

다음 시나리오에서는 1000개의 VM 마이그레이션을 테스트했습니다. 실제 마이그레이션을 시뮬레이션하기 위해 VM을 마이그레이션할 뿐 아니라 해당 VM이 있는 작업자 노드도 재부팅했습니다. 이를 실현하기 위해 노드를 3개의 영역으로 나눈 다음 비어 있는 머신 구성을 특정 영역에 적용했습니다. 그러면 작업자 노드에 상주하던 모든 VM을 제거할 때 해당 영역과 연결된 모든 노드가 재부팅됩니다.

먼저 모든 작업자를 특정 영역에 레이블 지정했습니다.

```
oc label node worker01 node-role.kubernetes.io/zone-0=""
oc label node worker02 node-role.kubernetes.io/zone-1=""
oc label node worker03 node-role.kubernetes.io/zone-2=""
```

그런 다음 모든 영역에 대해 머신 구성 풀을 생성했습니다. `maxUnavailable: 10`은 주어진 영역 수명 시간에 다운될 수 있는 노드의 수를 설정합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: zone-0
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values:
[worker, zone-2]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/zone-0: ""
  paused: false
  maxUnavailable: 10
```

하이퍼컨버지드 클러스터 오퍼레이터도 편집했습니다.

```
oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

그리고 다음 마이그레이션 설정을 통해 병렬 마이그레이션의 양을 늘렸습니다.

```
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 20 # default 5
    parallelOutboundMigrationsPerNode: 4 # default 2
    progressTimeout: 150
```

테스트를 통해 virt-api 포드 수를 **VM 750개당 1개 kubevirt-api 포드** 비율로 늘리는 것이 좋습니다. 이 설정에서는 3000개의 VM을 실행하고 있었기 때문에 kubevirt API 포드 수를 4개로 확장했습니다.

이 시나리오에 대한 자동 스케일링 기능은 이미 개발 중이며 Github#7101에서 확인할 수 있습니다. 현재는 하이퍼컨버지드 오퍼레이터를 패치하여 수동으로 수행할 수 있습니다.

```
oc patch hco -n openshift-cnv kubevirt-hyperconverged --type=merge -p
'{"metadata":{"annotations":{"kubevirt.kubevirt.io/jsonpatch":"[{\"op\":
\"add\", \"path\": \"/spec/customizeComponents/patches\", \"value\":
[{\"resourceType\": \"Deployment\", \"resourceName\": \"virt-api\",
\"type\": \"json\", \"patch\": \"[{\\\"op\\\": \\\"replace\\\",
\\\"path\\\": \\\"/spec/replicas\\\", \\\"value\\\": 4}]\"}]}]"}}}'
```

그런 다음 이 머신 구성을 생성하여 마이그레이션을 트리거합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: zone_target #target zone name
  name: job_name #must be unique every time.
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
```

```
    version: 3.1.0
  networkd: {}
  passwd: {}
  storage:
    files:
    - contents:
        source: data:text/plain;charset=utf-8;base64,Zm9vCg==
        verification: {}
      filesystem: root
      mode: 420
      path: /var/tmp/tmp_dir
osImageURL: ""
```

1000개의 마이그레이션된 VM은 다음과 같이 구성되었습니다.

- 400개의 RHEL VM
- 400개의 Fedora VM
- 200개의 Windows VM

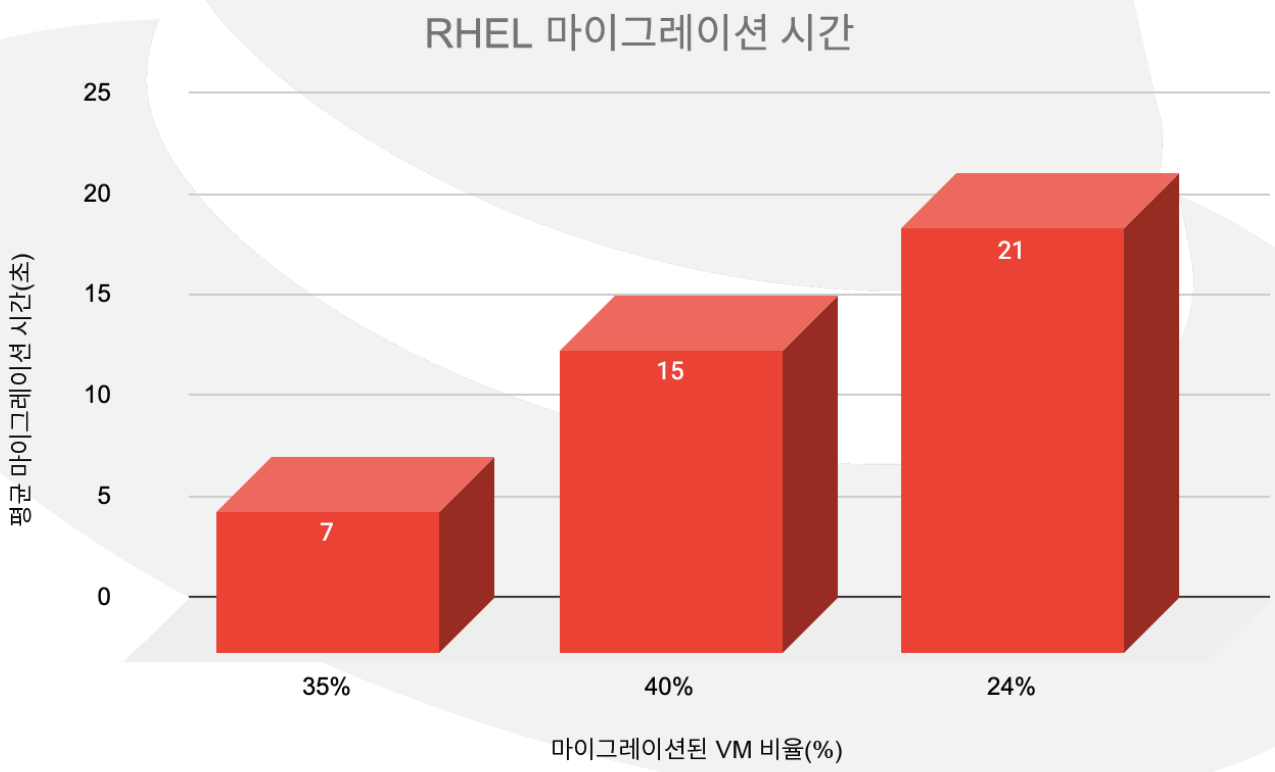
VM과 공동 배치된 7000개의 포드도 각기 다른 작업자 노드에서 다시 시작되었습니다.

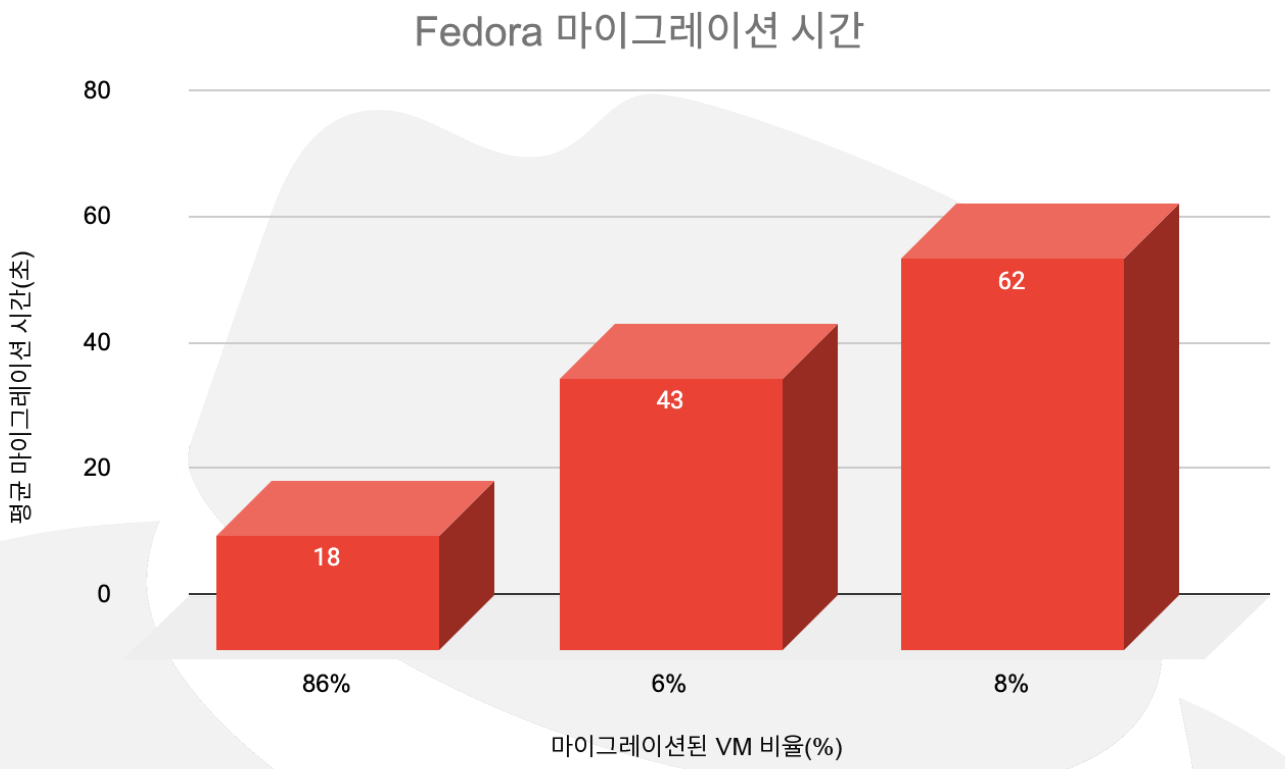
마이그레이션할 때 VMI 로그에서 모든 VM이 마이그레이션을 완료하기까지 소요한 시간을 확인할 수 있습니다.

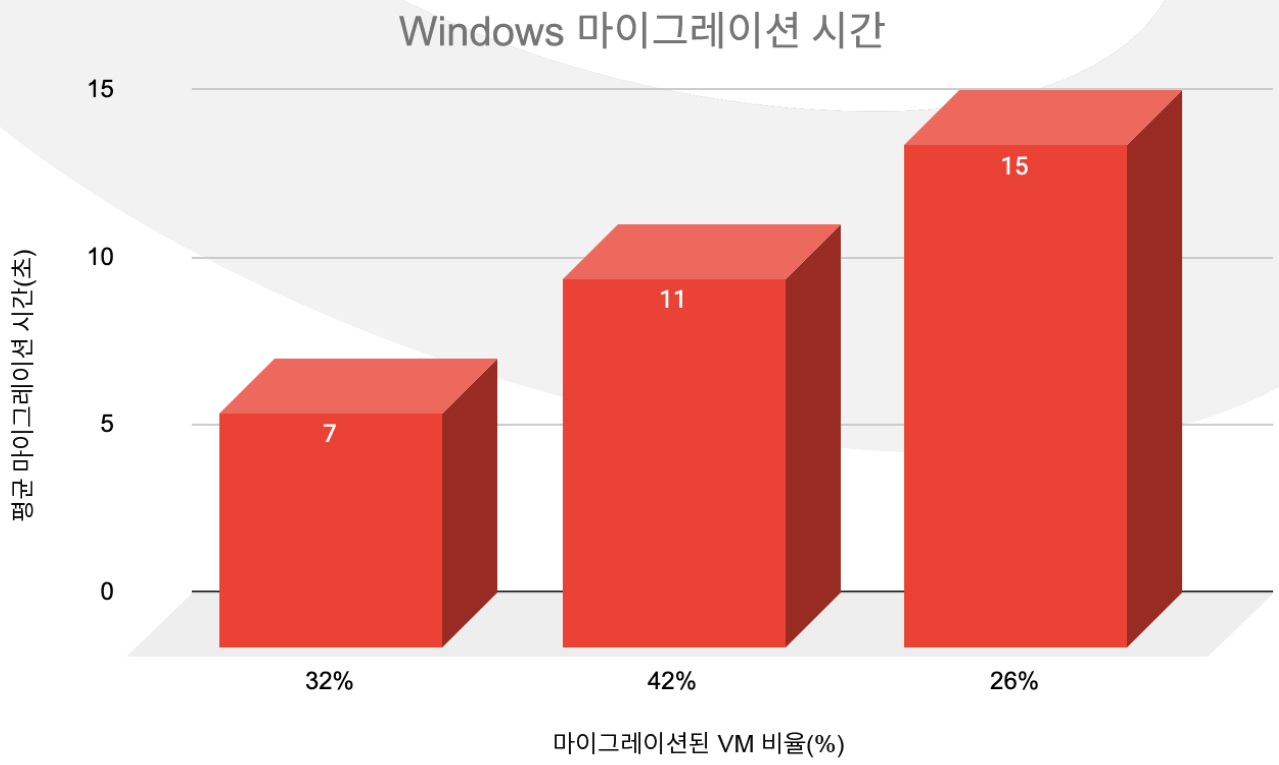
```
Phase Transition Timestamps:
  Phase:                        Scheduling
  Phase Transition Timestamp:   2022-04-10T07:15:13Z
  Phase:                        Scheduled
  Phase Transition Timestamp:   2022-04-10T07:15:23Z
  Phase:                        Running
  Phase Transition Timestamp:   2022-04-10T07:15:25Z
```

아래 차트는 분산된 각 OS의 마이그레이션 시간을 백분율로 보여줍니다. 예를 들어, RHEL VM 마이그레이션의 경우 35%의 VM이 마이그레이션을 완료되는 데 평균 7초의 시간이 걸렸습니다.

마이그레이션 시간은 반드시 OS와 관련된 것은 아니며, 해당 시간의 게스트 부하, 호스트 부하, 네트워크 부하, 스토리지 기술, 마이그레이션 정책, 이미지 크기 등과 관련되어 있습니다. 따라서 이러한 결과는 다양할 수 있습니다.




Fedora 마이그레이션 시간
80
60
40
20
0
평균 마이그레이션 시간(초)
18
43
62
86%
6%
8%
마이그레이션된 VM 비율(%)


Windows 마이그레이션 시간
15
10
5
0
평균 마이그레이션 시간(초)
7
11
15
32%
42%
26%
마이그레이션된 VM 비율(%)

OS당 평균 마이그레이션 시간:

OS	평균 마이그레이션 시간(초)	설명
RHEL	14	40GiB PVC
Fedora	23	Container disk evictionStrategy: Restart
Windows	12	40GiB PVC

요약하면, 마이그레이션은 시작부터 끝까지 총 118분 걸렸으며, 앞서 언급한 `maxUnavailable: 10`으로 인해 노드가 "Ready" 상태가 될 때까지 기다린 35분이 추가로 소요되었습니다.

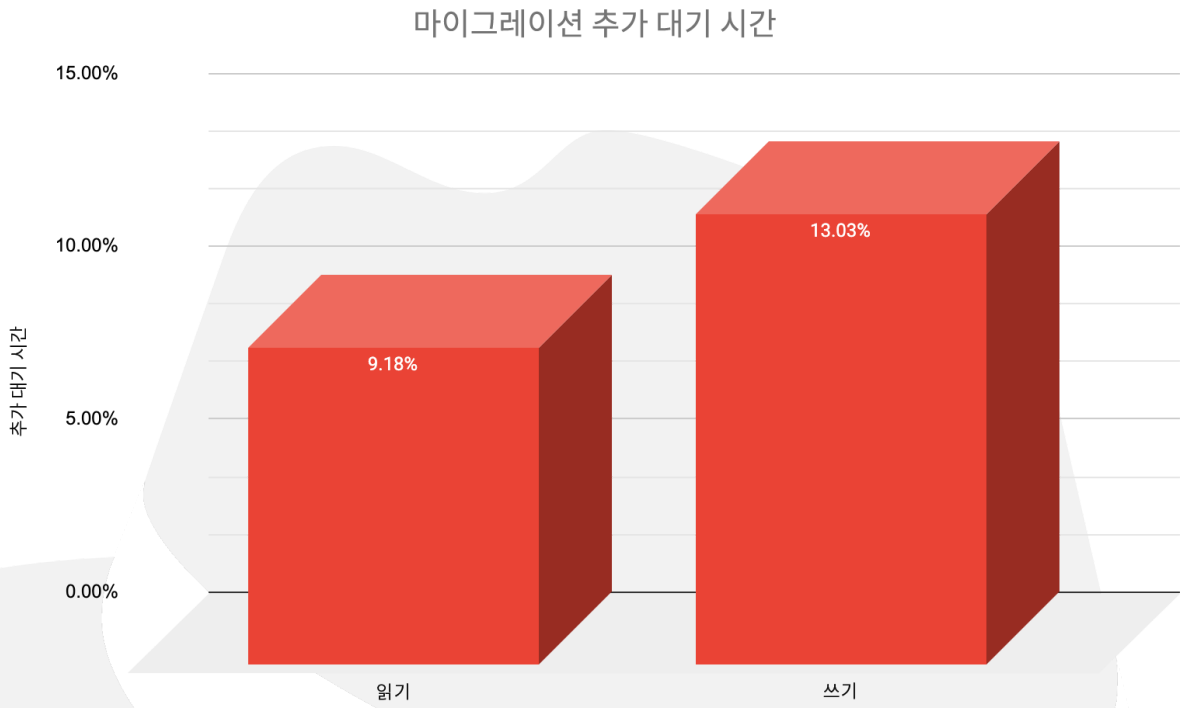
VM 마이그레이션 추가 대기 시간

다음 시나리오에서는 1000개의 VM 마이그레이션을 테스트했습니다. 이번에는 RHEL VM만 사용했습니다. 다양한 운영 체제에서 IO를 처리하는 방식으로 인해 발생할 수 있는 불일치를 방지하기 위해 RHEL VM만 사용하기로 했습니다.

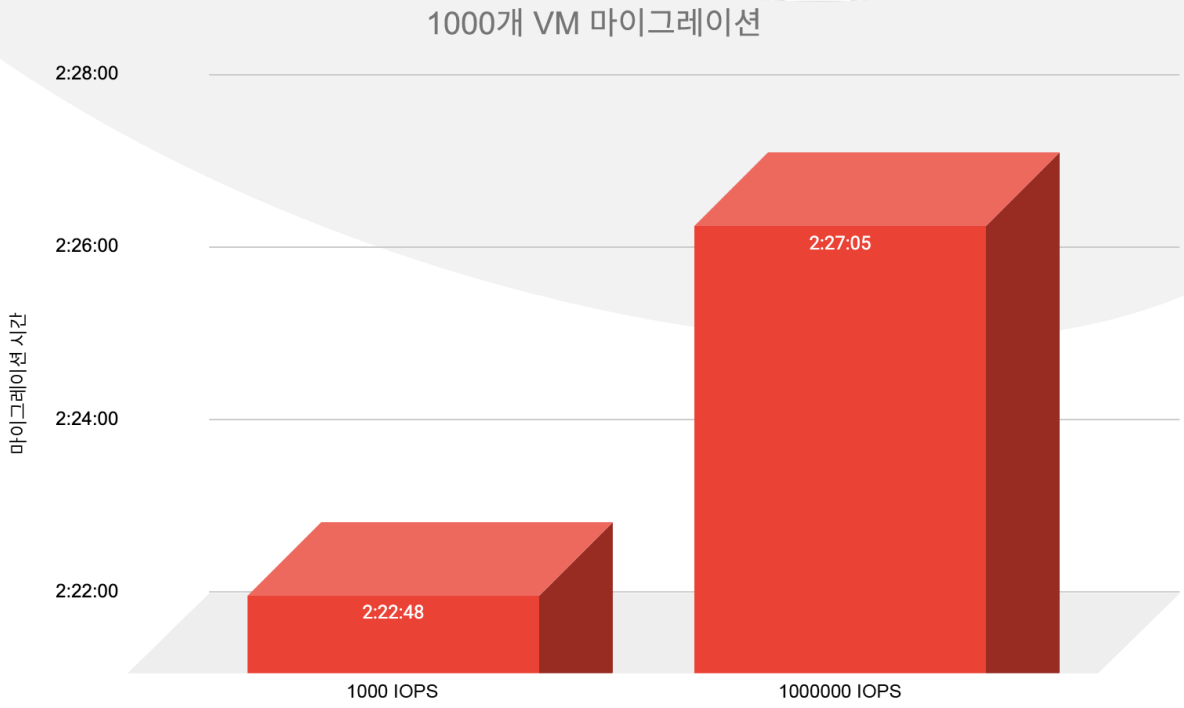
이전처럼 랜덤 읽기와 랜덤 쓰기 모두에 4KB 블록 크기를 사용하기로 선택하고 다음 테스트를 실행했습니다.

- 기준 - 각 1K VM은 단일 IOPS에서 총 1K IOPS를 생성합니다.
- 워크로드 - 각 1K VM은 초당 1000 IOPS에서 총 100만 IOPS를 생성합니다.

아래 그래프는 읽기와 쓰기 모두에 대한 마이그레이션 중 평균 대기 시간 페널티를 기준(RHEL VM만 사용한 경우)과 비교한 것입니다.



또한 아래 차트에서 볼 수 있듯이 마이그레이션 시간은 5%의 영향을 받았습니다. 마이그레이션 시간 결과는 BZ#2069098로 인해 달라질 수 있습니다.



Red Hat

규모에 따른 클러스터 업그레이드

다음 시나리오에서는 마이너 업그레이드와 메이저 업그레이드를 모두 테스트했습니다.

우선 실제 프로덕션 업그레이드를 시뮬레이션하는 동안 클러스터를 버전 4.9.15에서 4.9.23으로 업그레이드했습니다. 즉, 3000개의 VM과 21,400개의 포드가 모두 클러스터에서 실행되고 있었습니다. 또한 1500개의 VM에서 VM당 100 IOPS의 속도로 4KB의 가벼운 워크로드가 생성되었습니다.

다음을 실행하여 업그레이드를 시작했습니다.

```
$ oc adm upgrade --to 4.9.23
```

다음을 사용하여 업그레이드 진행률을 추적할 수 있습니다.

```
$ oc get clusterversion
```

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.15    True        True          25m     Working towards
4.9.23: 569 of 738 done (77% complete)
```

마이너 업그레이드 프로세스는 총 **35분** 걸렸습니다.

다음 단계에서는 클러스터를 버전 4.9.23에서 4.10.9로 업그레이드하고 이전과 동일한 조건에서 다시 업그레이드를 실행하여 메이저 업그레이드를 테스트했습니다. 다음을 실행하여 업그레이드를 시작했습니다.

```
oc adm upgrade channel candidate-4.10 --allow-explicit-channel
oc adm upgrade --to 4.10.9 --allow-explicit-upgrade
```

이전과 같이 다음을 사용하여 업그레이드 진행률을 추적할 수 있습니다.

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.23    True        True          40m     Working towards
4.10.9: 95 of 771 done (12% complete)
```

전체 메이저 업그레이드 프로세스는 총 **136분** 걸렸습니다.

일부 업그레이드에는 모든 노드가 소프트 재설정이 필요한 수정 사항이 포함될 수 있습니다. 이 경우 마이그레이션 시간이 추가되어 업그레이드 시간이 크게 늘어납니다.

결론

이 표준 아키텍처에서는 대규모 환경에서 OpenShift Virtualization 기능과 복원력을 증명했습니다. Red Hat OpenShift Container Platform의 OpenShift Virtualization 기능을 Red Hat Ceph Storage 및/또는 Red Hat OpenShift Data Foundation과 함께 사용하면 컨테이너, 가상 머신, 고가용성 스토리지를 통합하는 완벽한 프로덕션 솔루션을 제공할 수 있으며, 최소 하드웨어 요구 사항을 충족하는 모든 호스트에 배포할 수 있습니다.

이 표준 아키텍처에서는 설정한 목표를 달성하는 방법을 개략적으로 설명하지만, 주어진 환경 조건과 요구 사항에 따라 적절한 복원력, 확장성, 원활한 일상 운영을 위해 다른 아키텍처를 고려하는 것이 중요합니다. 예를 들어, 특정 한도를 초과할 경우 클러스터에서 노드 수나 워크로드가 너무 커지거나 이탈이 너무 많이 발생할 경우 멀티클러스터 접근 방식을 고려해야 합니다.

추가 리소스

시스템 및 환경 요구 사항:
https://docs.openshift.com/container-platform/3.11/install/prerequisites.html

OpenShift 템플릿:
https://docs.openshift.com/container-platform/4.9/openshift_images/using-templates.html

머신 구성 오퍼레이터:
https://docs.openshift.com/container-platform/4.9/post_installation_configuration/machine-configuration-tasks.html

실시간 마이그레이션 및 시간 제한:
https://docs.openshift.com/container-platform/4.9/virt/live_migration/virt-live-migration-limits.html

CLI를 사용한 클러스터 업데이트:
https://docs.openshift.com/container-platform/4.10/updating/updating-cluster-cli.html

Red Hat 소개

Red Hat은 세계적인 오픈소스 소프트웨어 솔루션 공급업체로서 커뮤니티 기반의 접근 방식을 통해 신뢰도 높은 고성능 Linux, 하이브리드 클라우드, 컨테이너 및 쿠버네티스 기술을 제공합니다. 또한 Red Hat은 고객이 클라우드 네이티브 애플리케이션을 개발하고, 신규 및 기존 IT 애플리케이션을 통합하고, 복잡한 환경을 자동화하고 관리할 수 있도록 지원합니다. Fortune 선정 500대 기업의 신뢰를 받는 어드바이저인 Red Hat은 전 세계 고객에게 권위 있는 어워드를 수상한 지원, 교육 및 컨설팅 서비스를 제공하여 모든 산업 분야에서 오픈 혁신의 이점을 실현할 수 있도록 최선을 다하고 있습니다. Red Hat은 기업, 파트너, 커뮤니티로 구성된 글로벌 네트워크의 허브 역할을 하며 고객들이 성장하고, 확장하고, 디지털 미래에 대비할 수 있도록 지원합니다.