Red Hat OpenStack Platform 16.2

인스턴스 생성용 Compute 서비스 구성

인스턴스 생성을 위한 Red Hat OpenStack Platform Compute(nova) 서비스 구성 및 관리 가이드

Last Updated: 2024-04-22

Red Hat OpenStack Platform 16.2 인스턴스 생성용 Compute 서비스 구성

인스턴스 생성을 위한 Red Hat OpenStack Platform Compute(nova) 서비스 구성 및 관리 가이드

OpenStack Team
rhos-docs@redhat.com

법적 공지

초록

이 가이드에서는 클라우드 관리자가 OpenStack 클라이언트 CLI를 사용하여 Red Hat OpenStack Platform Compute(nova) 서비스를 구성하고 관리할 수 있는 개념과 절차를 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 더 나은지 알려주십시오.

직접 문서 피드백(DDF) 기능 사용

피드백 추가 DDF 기능을 사용하여 특정 문장, 단락 또는 코드 블록에 대한 직접 의견을 제출할 수 있습니다.

1. *다중 페이지 HTML* 형식으로 문서를 봅니다.
2. 문서의 오른쪽 상단에 **피드백** 버튼이 표시되는지 확인합니다.
3. 주석 처리하려는 텍스트 부분을 강조 표시합니다.
4. **피드백 추가를** 클릭합니다.
5. 코멘트를 사용하여 **피드백 추가** 필드를 완료합니다.
6. 선택 사항: 문서 팀이 문제에 대한 자세한 설명을 위해 연락을 드릴 수 있도록 이메일 주소를 추가합니다.
7. **Submit(제출)을** 클릭합니다.

1장. COMPUTE 서비스(NOVA) 기능

Compute(nova) 서비스를 사용하여 RHOSP(Red Hat OpenStack Platform) 환경에서 가상 머신 인스턴스와 베어 메탈 서버를 생성, 프로비저닝 및 관리합니다. 계산 서비스는 기본 호스트 플랫폼에 대한 세부 사항을 노출하지 않고 실행되는 기본 하드웨어를 추상화합니다. 예를 들어, 호스트에서 실행 중인 CPU의 유형과 토폴로지를 노출하는 대신, 계산 서비스에서 여러 가상 CPU(vCPU)를 노출하고 이러한 vCPU를 오버 커밋할 수 있습니다.

계산 서비스는 KVM 하이퍼바이저를 사용하여 계산 서비스 워크로드를 실행합니다. libvirt 드라이버는 QEMU와 상호 작용하여 KVM과의 모든 상호 작용을 처리하고 가상 시스템 인스턴스 생성을 활성화합니다. 인스턴스를 생성하고 프로비저닝하기 위해 계산 서비스는 다음 RHOSP 서비스와 상호 작용합니다.

- 인증을 위한 ID(keystone) 서비스.
- 리소스 인벤토리 추적 및 선택을 위한 배치 서비스.
- 디스크 및 인스턴스 이미지의 Image 서비스(glance).
- 부팅 시 인스턴스가 연결되는 가상 또는 물리적 네트워크를 프로비저닝하기 위한 네트워킹(neutron) 서비스입니다.

계산 서비스는 **nova-*** 라는 데몬 프로세스 및 서비스로 구성됩니다. 핵심 Compute 서비스는 다음과 같습니다.

Compute 서비스(nova-compute)

이 서비스는 KVM 또는 QEMU 하이퍼바이저 API에 libvirt를 사용하여 인스턴스를 생성, 관리 및 종료하고, 인스턴스 상태로 데이터베이스를 업데이트합니다.

Compute Conductor(nova-conductor)

이 서비스는 계산 서비스와 데이터베이스 간의 상호 작용을 중재하여 계산 노드를 직접 데이터베이스 액세스로부터 보호합니다. **nova-compute** 서비스가 실행되는 노드에 이 서비스를 배포하지 마십시오.

컴퓨팅 스케줄러(nova-scheduler)

이 서비스는 대기열에서 인스턴스 요청을 가져와 인스턴스를 호스팅할 계산 노드를 결정합니다.

컴퓨팅 API(nova-api)

이 서비스는 사용자에게 외부 REST API를 제공합니다.

API 데이터베이스

이 데이터베이스는 인스턴스 위치 정보를 추적하고 빌드되지만 예약되지 않은 인스턴스의 임시 위치를 제공합니다. 다중 셀 배포에서 이 데이터베이스에는 각 셀의 데이터베이스 연결을 지정하는 셀 매핑도 포함되어 있습니다.

셀 데이터베이스

이 데이터베이스에는 인스턴스에 대한 대부분의 정보가 포함되어 있습니다. API 데이터베이스, 컨덕터 및 계산 서비스에서 사용합니다.

메세지 큐

이 메시징 서비스는 모든 서비스가 셀 내에서 및 글로벌 서비스와 통신하는 데 사용됩니다.

컴퓨팅 메타데이터

이 서비스는 인스턴스 고유의 데이터를 저장합니다. 인스턴스는 http://169.254.169.254 또는 링크-로컬 주소 fe80::a9fe:a9fe의 IPv6를 통해 메타데이터 서비스에 액세스합니다. 네트워킹(neutron) 서비스는 요청을 메타데이터 API 서버로 전달합니다. **NeutronMetadataProxySharedSecret** 매개변수를 사용하여 서비스가 통신할 수 있도록 네트워킹 서비스와 계산 서비스의 구성에 secret 키워드를 설정해야 합니다. 계산 메타데이터 서비스는 계산 API의 일부로 또는 각 셀에서 전역적으로 실행할 수 있습니다.

두 개 이상의 컴퓨팅 노드를 배포할 수 있습니다. 인스턴스를 작동하는 하이퍼바이저는 각 컴퓨팅 노드에서 실행됩니다. 각 컴퓨팅 노드에는 최소 두 개의 네트워크 인터페이스가 필요합니다. 계산 노드는 인스턴스를 가상 네트워크에 연결하고 보안 그룹을 통해 인스턴스에 방화벽 서비스를 제공하는 네트워킹 서비스 에이전트도 실행합니다.

기본적으로 director는 모든 컴퓨팅 노드에 대한 단일 셀로 오버클라우드를 설치합니다. 이 셀에는 가상 시스템 인스턴스를 제어 및 관리하는 모든 계산 서비스 및 데이터베이스와 모든 인스턴스 및 인스턴스 메타데이터가 포함됩니다. 대규모 배포의 경우 여러 셀이 있는 오버클라우드를 배포하여 더 많은 수의 컴퓨팅 노드를 수용할 수 있습니다. 새 오버클라우드를 설치할 때 또는 나중에 언제든지 셀을 환경에 추가할 수 있습니다. 자세한 내용은 Compute Cells를 사용한 배포 스케일링 을 참조하십시오.

2장. COMPUTE 서비스(NOVA) 구성

클라우드 관리자는 환경 파일을 사용하여 Compute(nova) 서비스를 사용자 지정합니다. Puppet은 이 구성을 **/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf** 파일에 생성하여 저장합니다. 다음 구성 방법을 사용하여 다음 순서의 우선 순위로 계산 서비스 구성을 사용자 지정합니다.

1. **Heat 매개변수** - *Overcloud Parameters 가이드의 Compute(nova) 매개 변수* 섹션에 자세히 설명되어 있습니다. 다음 예제에서는 heat 매개변수를 사용하여 기본 스케줄러 필터를 설정하고 Compute 서비스의 NFS 백엔드를 구성합니다.

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
AggregateInstanceExtraSpecsFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter
  NovaNfsEnabled: true
  NovaNfsShare: '192.0.2.254:/export/nova'
  NovaNfsOptions: 'context=system_u:object_r:nfs_t:s0'
  NovaNfsVersion: '4.2'
```

2. **Puppet 매개변수** - **/etc/puppet/modules/nova/manifests/*** 에 정의된 대로 :

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_raw_images: True
```

참고

동등한 heat 매개변수가 없는 경우에만 이 방법을 사용합니다.

3. **수동 hieradata 덮어쓰기** - heat 또는 Puppet 매개 변수가 없는 경우 매개변수를 사용자 정의하는 데 사용됩니다. 예를 들어 다음은 Compute 역할의 **[DEFAULT]** 섹션에 **timeout_nbd** 를 설정합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      DEFAULT/timeout_nbd:
        value: '20'
```

주의

heat 매개변수가 있으면 Puppet 매개변수 대신 사용합니다. Puppet 매개 변수가 있지만 heat 매개 변수가 아닌 경우 수동으로 재정의 방법 대신 Puppet 매개 변수를 사용합니다. 동일한 heat 또는 Puppet 매개 변수가 없는 경우에만 수동 재정의 방법을 사용합니다.

작은 정보

특정 구성을 사용자 지정하는 데 heat 또는 Puppet 매개변수를 사용할 수 있는지 확인하기 위해 수정할 매개변수 식별 지침에 따릅니다.

오버클라우드 서비스 설정 방법에 대한 자세한 내용은 *Advanced Overcloud Customization* 가이드의 Heat 매개변수를 참조하십시오.

2.1. 과다 할당을 위한 메모리 구성

메모리 과다 할당(**NovaRAMAllocationRatio** >= 1.0)을 사용하는 경우 할당 비율을 지원하기에 충분한 스왑 공간이 있는 오버클라우드를 배포해야 합니다.

참고

NovaRAMAllocationRatio 매개변수가 **<1** 로 설정된 경우 스왑 크기에 대한 RHEL 권장 사항을 따르십시오. 자세한 내용은 RHEL *Managing Storage Devices* 가이드의 권장 시스템 스왑 공간을 참조하십시오.

사전 요구 사항

- 노드에 필요한 스왑 크기를 계산했습니다. 자세한 내용은 스왑 크기 계산에서 참조하십시오.

절차

1. **/usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml** 파일을 환경 파일 디렉터리에 복사합니다.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml
/home/stack/templates/enable-swap.yaml
```

2. **enable-swap.yaml 파일에 다음 매개변수를 추가하여 스왑 크기를 구성합니다.**

```
parameter_defaults:
  swap_size_megabytes: <swap size in MB>
  swap_path: <full path to location of swap, default: /swap>
```

3. 다른 환경 파일과 함께 **enable_swap.yaml** 환경 파일을 스택에 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/enable-swap.yaml
```

2.2. 컴퓨팅 노드에서 예약된 호스트 메모리 계산

호스트 프로세스에 예약할 총 RAM 크기를 확인하려면 다음 각각에 충분한 메모리를 할당해야 합니다.

- 호스트에서 실행되는 리소스(예: OSD는 3GB 메모리를 사용합니다.
- 인스턴스를 호스팅하는 데 필요한 에뮬레이터 오버헤드.
- 각 인스턴스의 하이퍼바이저.

메모리에 대한 추가 요구 사항을 계산한 후 다음 공식을 사용하여 각 노드의 호스트 프로세스에 예약할 메모리 양을 결정하는 데 도움이 됩니다.

```
NovaReservedHostMemory = total_RAM - ( (vm_no * (avg_instance_size + overhead)) + (resource1 * resource_ram) + (resourcen * resource_ram))
```

- **vm_no** 를 인스턴스 수로 바꿉니다.
- **avg_instance_size** 를 각 인스턴스에서 사용할 수 있는 평균 메모리 양으로 바꿉니다.
- **오버헤드** 를 각 인스턴스에 필요한 하이퍼바이저 오버헤드로 바꿉니다.
- **resource1** 및 **<resourcen>** 의 모든 리소스를 노드의 리소스 유형 수로 바꿉니다.
- **resource_ram** 을 이 유형의 각 리소스에 필요한 RAM 양으로 바꿉니다.

2.3. 스왑 크기 계산

할당된 스왑 크기는 메모리 과다 할당을 처리할 수 있을 만큼 커야 합니다. 다음 공식을 사용하여 노드에 필요한 스왑 크기를 계산할 수 있습니다.

- overcommit_ratio = **NovaRAMAllocationRatio** - 1
- 최소 스왑 크기(MB) = **(total_RAM * overcommit_ratio) + RHEL_min_swap**
- 권장(최대) 스왑 크기(MB) = **total_RAM * (overcommit_ratio + percentage_of_RAM_to_use_for_swap)**

percentage_of_RAM_to_use_for_swap 변수는 QEMU 오버헤드 및 운영 체제 또는 호스트 서비스에서 사용하는 기타 리소스를 설명하는 버퍼를 생성합니다.

예를 들어 스왑에 사용 가능한 RAM의 25%, 총 RAM 64GB, **NovaRAMAllocationRatio** 를 **1** 로 설정하려면 다음을 수행합니다.

- 권장(최대) 스왑 크기 = 6MB * (0 + 0.25) = 16000 MB

NovaReservedHostMemory 값을 계산하는 방법에 대한 자세한 내용은 Compute 노드에서 예약된 호스트 메모리 계산에서 참조하십시오.

RHEL_min_swap 값을 결정하는 방법에 대한 자세한 내용은 RHEL *Managing Storage Devices* 가이드의 권장 시스템 스왑 공간을 참조하십시오.

3장. 성능을 위해 컴퓨팅 노드 구성

클라우드 관리자는 NFV 및 HPC(고성능 컴퓨팅)를 포함한 특수 워크로드를 대상으로 하는 맞춤형 플레이버를 만들어 최적의 성능을 위해 인스턴스의 스케줄링 및 배치를 구성할 수 있습니다.

다음 기능을 사용하여 최적의 성능을 위해 인스턴스를 조정합니다.

- **CPU 고정**: 가상 CPU를 물리적 CPU에 고정.
- **에뮬레이터 스레드**: 인스턴스와 연결된 에뮬레이터 스레드를 물리적 CPU에 고정합니다.
- **대규모 페이지**: 일반 메모리(4k 페이지) 및 대규모 페이지(2MB 또는 1GB 페이지)에 대해 인스턴스 메모리 할당 정책을 조정합니다.

참고

이러한 기능을 구성하면 NUMA 토폴로지가 이미 없는 경우 인스턴스에 암시적 NUMA 토폴로지가 생성됩니다.

3.1. 컴퓨팅 노드에서 CPU 고정 구성

컴퓨팅 노드에서 CPU 고정을 활성화하여 전용 호스트 CPU에서 실행되도록 각 인스턴스 CPU 프로세스를 구성할 수 있습니다. 인스턴스에서 CPU 고정을 사용하는 경우 각 인스턴스 vCPU 프로세스에 다른 인스턴스 vCPU 프로세스가 사용할 수 없는 고유한 호스트 pCPU가 할당됩니다. CPU 고정이 활성화된 Compute 노드에서 실행되는 인스턴스에는 NUMA 토폴로지가 있습니다. 인스턴스 NUMA 토폴로지의 각 NUMA 노드는 호스트 Compute 노드의 NUMA 노드에 매핑됩니다.

동일한 컴퓨팅 노드에서 공유(유동) CPU를 사용하여 전용(고정) CPU를 사용하여 인스턴스를 예약하도록 Compute 스케줄러를 구성할 수 있습니다. NUMA 토폴로지가 있는 컴퓨팅 노드에서 CPU 고정을 구성하려면 다음을 완료해야 합니다.

1. CPU 고정을 위해 컴퓨팅 노드 지정.
2. 고정 인스턴스 vCPU 프로세스, 유동 인스턴스 vCPU 프로세스 및 호스트 프로세스에 대한 호스트 코어를 예약하도록 계산 노드를 구성합니다.
3. Overcloud를 배포합니다.
4. CPU 고정이 필요한 인스턴스를 시작하기 위한 플레이버를 만듭니다.
5. 공유 또는 유동 CPU를 사용하는 인스턴스를 시작하기 위한 플레이버를 만듭니다.

3.1.1. 사전 요구 사항

- Compute 노드의 NUMA 토폴로지를 알고 있습니다.

3.1.2. CPU 고정을 위해 컴퓨팅 노드 지정

고정 CPU로 인스턴스에 대해 컴퓨팅 노드를 지정하려면 새 역할 파일을 만들어 CPU 고정 역할을 구성하고, CPU 고정을 위해 사용할 새 오버클라우드 플레이버 및 CPU 고정 리소스 클래스를 구성해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. **Controller,Compute 및 Compute CPUPinning** 역할이 포함된 **roles_data_cpu_pinning.yaml** 이라는 새 역할 데이터 파일을 생성합니다.

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_cpu_pinning.yaml \
 Compute:ComputeCPUPinning Compute Controller
```

4. **roles_data_cpu_pinning.yaml** 을 열고 다음 매개변수 및 섹션을 편집하거나 추가합니다.

섹션/패랜드	현재 값	새 값
역할 설명	**역할: 컴퓨팅**	**역할: ComputeCPUPinning**
역할 이름	**name: 컴퓨팅**	**name: ComputeCPUPinning**
description	**기본 컴퓨팅 노드 역할**	**CPU 고정 컴퓨팅 노드 역할**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepinning-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-cpu-pinning.yaml**

5. 노드 정의 템플릿 node. **json 또는 node. yaml에 추가하여 오버클라우드의 CPU 고정 컴퓨팅 노드를** 등록합니다. 자세한 내용은 *Director 설치 및 사용* 가이드 의 오버클라우드 노드 등록을 참조하십시오.

6. 노드 하드웨어를 검사합니다.

```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

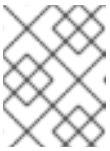
자세한 내용은 *Director 설치 및 사용* 가이드 의 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

7. CPU 고정 컴퓨팅 **노드의 compute-cpu-pinning** 오버클라우드 플레이버를 생성합니다.

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> compute-cpu-pinning
```

- **<ram_size_mb>** 를 베어 메탈 노드의 RAM(MB)으로 바꿉니다.
- **<disk_size_gb>** 를 베어 메탈 노드의 디스크 크기(GB)로 바꿉니다.

- **<no_vcpus>** 를 베어 메탈 노드의 CPU 수로 바꿉니다.



참고

이러한 속성은 인스턴스를 예약하는 데 사용되지 않습니다. 그러나 계산 스케줄러는 디스크 크기를 사용하여 루트 파티션 크기를 결정합니다.

8. 사용자 정의 CPU 고정 리소스 클래스를 사용하여 CPU 고정을 지정할 각 베어 메탈 노드에 태그를 지정합니다.

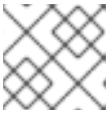
```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.CPU-PINNING <node>
```

<node> 를 베어 메탈 노드의 ID로 바꿉니다.

9. **compute-cpu-pinning** 플레이버를 사용자 지정 CPU 고정 리소스 클래스와 연결합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_CPU_PINNING=1 \
 compute-cpu-pinning
```

베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 각 문장 부호 표시를 밑줄로 바꾸고 접두사는 **CUSTOM_** 로 바꿉니다.



참고

플레이버는 베어 메탈 리소스 클래스의 인스턴스 하나만 요청할 수 있습니다.

10. Compute 스케줄러가 베어 메탈 플레이버 속성을 사용하여 인스턴스를 예약하지 못하도록 다음 플레이버 속성을 설정합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 \
 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 compute-cpu-pinning
```

11. 선택 사항: **ComputeCPUPinning 역할의 네트워크 토폴로지가 Compute** 역할의 네트워크 토폴로지와 다른 경우 사용자 지정 네트워크 인터페이스 템플릿을 **생성합니다**. 자세한 내용은 *Advanced Overcloud Customization* 가이드의 Custom network interface templates 를 참조하십시오.
ComputeCPUPinning 역할의 네트워크 토폴로지가 **Compute** 역할과 동일한 경우 **compute.yaml** 에 정의된 기본 네트워크 토폴로지를 사용할 수 있습니다.

12. **network-environment.yaml 파일에서 ComputeCPUPinning 역할의 Net::SoftwareConfig 를 등록합니다.**

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeCPUPinning::Net::SoftwareConfig: /home/stack/templates/nic-
```

```
configs/<cpu_pinning_net_top>.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

<cpu_pinning_net_top> 을 **ComputeCPUPinning** 역할의 네트워크 토폴로지가 포함된 파일 이름으로 바꿉니다(예: 기본 네트워크 토폴로지를 사용하려면 **compute.yaml**).

13. 다음 매개변수를 **node-info.yaml** 파일에 추가하여 CPU 고정 컴퓨팅 노드 수와 CPU 고정 지정된 컴퓨팅 노드에 사용할 플레이버를 지정합니다.

```
parameter_defaults:
  OvercloudComputeCPUPinningFlavor: compute-cpu-pinning
  ComputeCPUPinningCount: 3
```

14. 역할이 생성되었는지 확인하려면 다음 명령을 입력합니다.

```
(undercloud)$ openstack baremetal node list --long -c "UUID" \
 -c "Instance UUID" -c "Resource Class" -c "Provisioning State" \
 -c "Power State" -c "Last Error" -c "Fault" -c "Name" -f json
```

출력 예:

```
[
  {
    "Fault": null,
    "Instance UUID": "e8e60d37-d7c7-4210-acf7-f04b245582ea",
    "Last Error": null,
    "Name": "compute-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "baremetal.CPU-PINNING",
    "UUID": "b5a9ac58-63a7-49ba-b4ad-33d84000ccb4"
  },
  {
    "Fault": null,
    "Instance UUID": "3ec34c0b-c4f5-4535-9bd3-8a1649d2e1bd",
    "Last Error": null,
    "Name": "compute-1",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "compute",
    "UUID": "432e7f86-8da2-44a6-9b14-dfacdf611366"
  },
  {
    "Fault": null,
    "Instance UUID": "4992c2da-adde-41b3-bef1-3a5b8e356fc0",
    "Last Error": null,
    "Name": "controller-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "controller",
    "UUID": "474c2fc8-b884-4377-b6d7-781082a3a9c0"
  }
]
```

3.1.3. CPU 고정을 위한 컴퓨팅 노드 구성

노드의 NUMA 토폴로지를 기반으로 Compute 노드에서 CPU 고정을 구성합니다. 호스트 프로세스의 효율성을 위해 모든 NUMA 노드에서 일부 CPU 코어를 예약합니다. 나머지 CPU 코어를 인스턴스 관리에 할당합니다.

이 절차에서는 CPU 고정 구성 방법을 설명하기 위해 8개의 CPU 코어가 2개의 NUMA 노드에 분산되어 있는 다음 NUMA 토폴로지를 사용합니다.

표 3.1. NUMA 토폴로지 예

NUMA 노드 0		NUMA 노드 1	
코어 0	코어 1	코어 2	코어 3
코어 4	코어 5	코어 6	코어 7

이 절차에서는 코어 0 및 4를 호스트 프로세스, 코어 1, 3, 5, 7을 CPU 고정이 필요한 인스턴스에 대해 예약하고, CPU 고정이 필요하지 않은 유동 인스턴스에 대해 코어 2 및 6을 예약합니다.

절차

1. 환경 파일을 만들어 고정된 인스턴스, 유동 인스턴스 및 호스트 프로세스(예: **cpu_pinning.yaml**)의 코어를 예약하도록 컴퓨팅 노드를 구성합니다.

2. NUMA 사용 가능한 Compute 노드에 NUMA 토폴로지를 사용하여 인스턴스를 예약하려면 아직 없는 경우 Compute 환경 **파일의 NovaSchedulerDefaultFilter 매개변수에 NUMATopology** Filter를 추가합니다.

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','Serve
rGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```

   NUMATopologyFilter 에 대한 자세한 내용은 Compute Scheduler filters를 참조하십시오.

3. 전용 인스턴스의 물리적 CPU 코어를 예약하려면 **cpu_pinning.yaml에 다음 구성을 추가합니다.**

```
parameter_defaults:
  ComputeCPUPinningParameters:
    NovaComputeCpuDedicatedSet: 1,3,5,7
```

4. 공유 인스턴스의 물리적 CPU 코어를 예약하려면 **cpu_pinning.yaml에 다음 구성을 추가합니다.**

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    NovaComputeCpuSharedSet: 2,6
```

5. 호스트 프로세스에 예약할 RAM 크기를 지정하려면 **cpu_pinning.yaml에 다음 설정을 추가합니다.**

```
parameter_defaults:
```

```
  ComputeCPUPinningParameters:
    ...
    NovaReservedHostMemory: <ram>
```

<ram> 을 MB로 예약할 RAM 양으로 바꿉니다.

6. 호스트 프로세스가 인스턴스에 예약된 CPU 코어에서 실행되지 않도록 하려면 **IsolCpusList** 매개변수를 인스턴스에 예약한 CPU 코어로 설정합니다.

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    IsolCpusList: 1-3,5-7
```

쉼표로 구분된 CPU 인덱스의 목록 또는 범위를 사용하여 **IsolCpusList** 매개변수 값을 지정합니다.

7. 다른 환경 파일을 사용하여 스택에 새 역할 및 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -r /home/stack/templates/roles_data_cpu_pinning.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/cpu_pinning.yaml \
  -e /home/stack/templates/node-info.yaml
```

3.1.4. 인스턴스의 전용 CPU 플레이버 생성

클라우드 사용자가 전용 CPU가 있는 인스턴스를 만들 수 있도록 하려면 인스턴스를 시작하기 위한 전용 CPU 정책으로 플레이버를 만들 수 있습니다.

사전 요구 사항

- 호스트에서 동시 멀티스레딩(SMT)이 활성화됩니다.
- 컴퓨팅 노드는 CPU 고정을 허용하도록 구성되어 있습니다. 자세한 내용은 Compute 노드에서 CPU 고정 구성을 참조하십시오.

절차

1. **overcloudrc** 파일을 소싱합니다.

```
(undercloud)$ source ~/overcloudrc
```

2. CPU 고정이 필요한 인스턴스의 플레이버를 생성합니다.

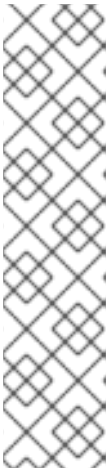
```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_reserved_vcpus> pinned_cpus
```

3. 고정된 CPU를 요청하려면 플레이버의 **hw:cpu_policy** 속성을 전용으로 설정합니다.

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_policy=dedicated pinned_cpus
```

4. 스레드 시블링에 각 **vCPU**를 배치하려면 다음을 요구 하도록 플레이버의 **hw:cpu_thread_policy** 속성을 설정합니다.

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_thread_policy=require pinned_cpus
```



참고

- 호스트에 **SMT** 아키텍처 또는 사용 가능한 스레드 스레딩이 충분한 **CPU** 코어가 없는 경우 예약에 실패합니다. 이를 방지하려면 **require** 대신 **hw:cpu_thread_policy** 를 **prefer** 로 설정합니다. **prefer** 정책은 사용 가능한 경우 스레드를 사용하도록 하는 기본 정책입니다.

- **hw:cpu_thread_policy=isolate** 을 사용하는 경우 **SMT**를 비활성화하거나 **SMT**를 지원하지 않는 플랫폼을 사용해야 합니다.

검증

1. 플레이버가 전용 **CPU**가 있는 인스턴스를 생성하는지 확인하려면 새 플레이버를 사용하여 인스턴스를 시작합니다.

```
(overcloud)$ openstack server create --flavor pinned_cpus \
 --image <image> pinned_cpu_instance
```

2. 새 인스턴스의 올바른 배치를 확인하려면 다음 명령을 입력하고 출력에 **OS-EXT-SRV-ATTR:hypervisor_hostname** 을 확인합니다.

```
(overcloud)$ openstack server show pinned_cpu_instance
```

3.1.5. 인스턴스의 공유 CPU 플레이버 생성

클라우드 사용자가 공유 또는 유동 **CPU**를 사용하는 인스턴스를 만들 수 있도록 하려면 공유 **CPU** 정책으로 플레이버를 만들어 인스턴스를 시작할 수 있습니다.

사전 요구 사항

- 컴퓨팅 노드는 공유 **CPU**에 대해 물리적 **CPU** 코어를 예약하도록 구성되어 있습니다. 자세한 내용은 **Compute** 노드에서 **CPU** 고정 구성을 참조하십시오.

절차

1. **overcloudrc** 파일을 소싱합니다.

```
(undercloud)$ source ~/overcloudrc
```

2. **CPU** 고정이 필요하지 않은 인스턴스의 플레이버를 생성합니다.

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_reserved_vcpus> floating_cpus
```

3. 유동 **CPU**를 요청하려면 플레이버의 **hw:cpu_policy** 속성을 **shared** 로 설정합니다.

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_policy=shared floating_cpus
```

검증

1. 플레이버가 공유 **CPU**를 사용하는 인스턴스를 생성하는지 확인하려면 새 플레이버를 사용하여 인스턴스를 시작합니다.

```
(overcloud)$ openstack server create --flavor floating_cpus \
 --image <image> floating_cpu_instance
```

2. 새 인스턴스의 올바른 배치를 확인하려면 다음 명령을 입력하고 출력에 **OS-EXT-SRV-ATTR:hypervisor_hostname** 을 확인합니다.

```
(overcloud)$ openstack server show floating_cpu_instance
```

3.1.6. SMT(동시 멀티스레딩)를 사용하여 컴퓨팅 노드에서 CPU 고정 구성

컴퓨팅 노드가 **SMT(**동시 멀티스레딩**)**를 지원하는 경우 전용 또는 공유 집합에 그룹 스레드를 함께 사용합니다. 스레드 스레딩은 몇 가지 일반적인 하드웨어를 공유하므로 한 스레드 시블링에서 프로세스를 실행하여 다른 스레드 스레딩의 성능에 영향을 줄 수 있습니다.

예를 들어 호스트는 **SMT**가 있는 듀얼 코어 **CPU**에서 **4**개의 논리적 **CPU** 코어를 식별합니다. **0, 1, 2** 및 **3.** 이 **4**개 중 두 쌍의 스레드 시블링이 있습니다.

- 스레드 시블링 1: 논리 CPU 코어 0 및 2

- 스레드 시블링 2: 논리 CPU 코어 1 및 3

이 시나리오에서는 논리 CPU 코어 0과 1을 전용으로 할당하지 말고 2 및 3을 공유로 할당하지 마십시오. 대신 0과 2를 전용으로 할당하고 1 및 3을 공유로 할당합니다.

파일 /sys/devices/system/cpu/cpuN/topology/thread_siblings_list 여기서 N 은 논리 CPU 번호이며 스레드 쌍을 포함합니다. 다음 명령을 사용하여 스레드 에이전트인 논리적 CPU 코어를 식별할 수 있습니다.

```
# grep -H . /sys/devices/system/cpu/cpu*/topology/thread_siblings_list | sort -n -t ':' -k 2 -u
```

다음 출력은 논리 CPU 코어 0 및 논리적 CPU 코어 2가 동일한 코어의 스레드임을 나타냅니다.

```
/sys/devices/system/cpu/cpu0/topology/thread_siblings_list:0,2
/sys/devices/system/cpu/cpu2/topology/thread_siblings_list:1,3
```

3.1.7. 추가 리소스

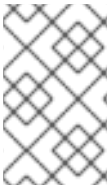
- *Network Functions Virtualization Planning and Configuration Guide* 에서 NUMA 노드 토폴로지를 검색합니다.

- *Network Functions Virtualization Product Guide* 의 CPU 및 NUMA 노드

3.2. 에뮬레이터 스레드 구성

계산 노드에는 에뮬레이터 스레드라고 하는 각 인스턴스의 하이퍼바이저와 연결된 오버헤드 작업이 있습니다. 기본적으로 에뮬레이터 스레드는 인스턴스와 동일한 CPU에서 실행되므로 인스턴스의 성능에 영향을 미칩니다.

인스턴스가 사용하는 사용자에 대해 별도의 CPU에서 에뮬레이터 스레드를 실행하도록 에뮬레이터 스레드 정책을 구성할 수 있습니다.



참고

패킷 손실을 피하려면 NFV 배포에서 vCPU를 선점해서는 안 됩니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.

2. 컴퓨팅 환경 파일을 엽니다.

3. CPU 고정이 필요한 인스턴스에 대해 물리적 CPU 코어를 예약하려면 Compute 환경 파일에서 NovaComputeCpuDedicatedSet 매개변수를 구성합니다. 예를 들어 다음 구성은 32코어 CPU가 있는 컴퓨팅 노드에서 전용 CPU를 설정합니다.

```
parameter_defaults:
  ...
  NovaComputeCpuDedicatedSet: 2-15,18-31
  ...
```

자세한 내용은 Compute 노드에서 CPU 고정 구성을 참조하십시오.

4. 에뮬레이터 스레드에 대한 물리적 CPU 코어를 예약하려면 Compute 환경 파일에서 NovaComputeCpuSharedSet 매개 변수를 구성합니다. 예를 들어 다음 구성은 32코어 CPU가 있는 컴퓨팅 노드에서 공유 CPU를 설정합니다.

```
parameter_defaults:
  ...
  NovaComputeCpuSharedSet: 0,1,16,17
  ...
```

참고

계산 스케줄러는 공유 또는 유동 CPU에서 실행되는 인스턴스의 공유 세트의 CPU도 사용합니다. 자세한 내용은 Compute 노드에서 CPU 고정 구성을참조하십시오.

5. Compute 스케줄러 필터 NUMATopologyFilter 를 NovaSchedulerDefaultFilters 매개변수에 추가합니다(아직 없는 경우).

6. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

7. **NovaComputeCpuSharedSet** 를 사용하여 구성된 공유 **CPU**에서 선택되는 전용 **CPU**에서 인스턴스에 대해 에뮬레이터 스레드를 실행하는 플레이버를 구성합니다.

```
(overcloud)$ openstack flavor set --property hw:cpu_policy=dedicated \
 --property hw:emulator_threads_policy=share \
 dedicated_emulator_threads
```

hw:emulator_threads_policy 에 대한 구성 옵션에 대한 자세한 내용은 **Flavor** 메타데이터의 **RuntimeClass** 스레드 정책을 참조하십시오.

3.3. 컴퓨팅 노드에서 HUGE PAGE 구성

클라우드 관리자는 인스턴스가 대규모 페이지를 요청할 수 있도록 컴퓨팅 노드를 구성할 수 있습니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 각 **NUMA** 노드에서 인스턴스가 아닌 프로세스를 예약하도록 대규모 페이지 메모리의 양을 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaReservedHugePages: ["node:0,size:1GB,count:1","node:1,size:1GB,count:1"]
```

- 각 노드의 크기 값을 할당된 대규모 페이지의 크기로 바꿉니다. 다음 유효한 값 중 하나로 설정합니다.
  - **2048 (2MB**의 경우)

- **1GB**

- 각 노드의 개수 값을 **NUMA** 노드당 **OVS**에서 사용하는 대규모 페이지 수로 바꿉니다. 예를 들어 **Open vSwitch**에서 사용하는 소켓 메모리 **4096**의 경우 이 값을 **2**로 설정합니다.

3. 컴퓨팅 노드에서 대규모 페이지를 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    ...
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32"
```

참고

여러 대규모 페이지 크기를 구성하는 경우 첫 번째 부팅 중에 대규모 페이지 폴더를 마운트해야 합니다. 자세한 내용은 처음 부팅할 때 여러 대규모 페이지 폴더 마운트를 참조하십시오.

4. 선택 사항: 인스턴스가 **1GB**대 페이지를 할당할 수 있도록 **CPU** 기능 플래그 **NovaLibvirtCPUModelExtraFlags** 를 **include pdpe1gb** 로 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'Haswell-noTSX'
    NovaLibvirtCPUModelExtraFlags: 'vmx, pdpe1gb'
```

참고

- 인스턴스가 **2MB**의 대규모 페이지만 요청할 수 있도록 **CPU** 기능 플래그를 구성할 필요는 없습니다.
- 호스트가 **1G** 대규모 페이지 할당을 지원하는 경우에만 인스턴스에 **1G** 대규모 페이지를 할당할 수 있습니다.
- **NovaLibvirtCPUMode**가 **host-model** 또는 **custom** 로 설정된 경우 **NovaLibvirtCPUMode**를 **NovaLibvirtCPU ModelExtraFlags to pdpe1gb** 만 설정하면 됩니다.
- 호스트 **support pdpe1gb** 및 **host-passthrough** 가 **NovaLibvirtCPUMode** 로 사용되는 경우 **set pdpe1gb** 를 **NovaLibvirtCPUModelExtraFlags** 로 사용할 필요가 없습니다. **The pdpe1gb** 플래그는 **Opteron_G4** 및 **Opteron_G5 CPU** 모델에만 포함되어 있으며 **QEMU**에서 지원하는 **Intel CPU** 모델에 포함되지 않습니다.
- **MDS(Microarchitectural Data Sampling)**와 같은 **CPU** 하드웨어 문제를 완화하려면 다른 **CPU** 플래그를 구성해야 할 수 있습니다. 자세한 내용은 **RHOS Mitigation for MDS("Microarchitectural Data Sampling") 보안 결함**을 참조하십시오.

5. **Meltdown** 보호를 적용한 후 성능 손실을 방지하려면 **+pcid** 를 포함하도록 **NovaLibvirtCPUModelExtraFlags CPU** 기능 플래그를 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'Haswell-noTSX'
    NovaLibvirtCPUModelExtraFlags: 'vmx, pdpe1gb, +pcid'
```

작은 정보

자세한 내용은 **"PCID" CPU 기능 플래그를 사용하여 OpenStack 게스트의 *Meltdown* CVE 픽스 성능에 미치는 영향** 감소를 참조하십시오.

6. 아직 없는 경우 **NovaSchedulerDefaultFilters** 매개변수에 **NUMATopology Filter**를 추가합니다.

7. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files]  \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

3.3.1. 인스턴스의 대규모 페이지 플레이버 생성

클라우드 사용자가 대규모 페이지를 사용하는 인스턴스를 생성할 수 있도록 하려면 인스턴스를 시작하기 위한 **hw:mem_page_size** 추가 사양 키로 플레이버를 생성할 수 있습니다.

사전 요구 사항

- 컴퓨팅 노드는 대규모 페이지에 대해 구성됩니다. 자세한 내용은 컴퓨팅 노드에서 대규모 페이지 구성을 참조하십시오.

절차

1. 대규모 페이지가 필요한 인스턴스의 플레이버를 생성합니다.

```
$ openstack flavor create --ram <size_mb> --disk <size_gb> \
 --vcpus <no_reserved_vcpus> huge_pages
```

2. 대규모 페이지를 요청하려면 플레이버의 **hw:mem_page_size** 속성을 필수 크기로 설정합니다.

```
$ openstack flavor set huge_pages --property hw:mem_page_size=1GB
```

다음 유효한 값 중 하나로 **hw:mem_page_size** 를 설정합니다.

- **Large -** 호스트에서 지원되는 가장 큰 페이지 크기를 선택합니다. 이 크기는 **x86_64** 시스템에서 **2MB** 또는 **1GB**일 수 있습니다.
- **small -** (기본값) 호스트에서 지원되는 최소 페이지 크기를 선택합니다. **x86_64** 시스템에서는 **4 kB**(일반 페이지)입니다.
-

any - libvirt 드라이버가 결정한 대로 사용 가능한 가장 큰 대규모 페이지 크기를 선택합니다.

- **<pagesize>:** (문자열) 워크로드에 특정 요구 사항이 있는 경우 명시적 페이지 크기를 설정합니다. 페이지 크기(KB) 또는 표준 접미사에 정수 값을 사용합니다. 예를 들면 다음과 같습니다. **4KB, 2MB, 2048, 1GB.**

3. 플레이버가 대규모 페이지가 있는 인스턴스를 생성하는지 확인하려면 새 플레이버를 사용하여 인스턴스를 시작합니다.

```
$ openstack server create --flavor huge_pages \
 --image <image> huge_pages_instance
```

계산 스케줄러는 인스턴스의 메모리를 백업하는 데 필요한 크기의 사용 가능한 대규모 페이지가 충분한 호스트를 식별합니다. 스케줄러가 충분한 페이지가 있는 호스트 및 **NUMA** 노드를 찾을 수 없는 경우 요청이 **NoValidHost** 오류로 인해 실패합니다.

3.3.2. 첫 번째 부팅 중에 여러 개의 대규모 페이지 폴더 마운트

첫 번째 부팅 프로세스의 일부로 여러 페이지 크기를 처리하도록 **Compute** 서비스(**nova**)를 구성할 수 있습니다. 첫 번째 부팅 프로세스는 노드를 처음 부팅할 때 모든 노드에 **heat** 템플릿 설정을 추가합니다. 오버클라우드 스택 업데이트와 같은 이러한 템플릿이 나중에 포함되면 이러한 스크립트가 실행되지 않습니다.

절차

1. 스크립트를 실행하여 대규모 페이지 폴더에 대한 마운트를 생성하는 첫 번째 부팅 템플릿 파일 **hugepages.yaml** 을 생성합니다. **OS::TripleO::MultipartMime** 리소스 유형을 사용하여 구성 스크립트를 보낼 수 있습니다.

```
heat_template_version: <version>

description: >
  Huge pages configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: hugepages_config}

  hugepages_config:
    type: OS::Heat::SoftwareConfig
```

```
  properties:
    config: |
      #!/bin/bash
      hostname | grep -qiE 'co?mp' || exit 0
      systemctl mask dev-hugepages.mount || true
      for pagesize in 2M 1G;do
        if ! [ -d "/dev/hugepages${pagesize}" ]; then
          mkdir -p "/dev/hugepages${pagesize}"
          cat << EOF > /etc/systemd/system/dev-hugepages${pagesize}.mount
      [Unit]
      Description=${pagesize} Huge Pages File System
      Documentation=https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt
      Documentation=https://www.freedesktop.org/wiki/Software/systemd/APIFileSystems
      DefaultDependencies=no
      Before=sysinit.target
      ConditionPathExists=/sys/kernel/mm/hugepages
      ConditionCapability=CAP_SYS_ADMIN
      ConditionVirtualization=!private-users

      [Mount]
      What=hugetlbfs
      Where=/dev/hugepages${pagesize}
      Type=hugetlbfs
      Options=pagesize=${pagesize}

      [Install]
      WantedBy = sysinit.target
      EOF
        fi
      done
      systemctl daemon-reload
      for pagesize in 2M 1G;do
        systemctl enable --now dev-hugepages${pagesize}.mount
      done

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

이 템플릿의 구성 스크립트는 다음 작업을 수행합니다.

a. **'co?mp'** 와 일치하는 호스트 이름을 지정하여 대규모 페이지 폴더에 대한 마운트를 생성하도록 호스트를 필터링합니다. 필요에 따라 특정 계산에 대해 필터 **grep** 패턴을 업데이트할 수 있습니다.

b. 기본 **dev-hugepages.mount systemd** 장치 파일을 연결하여 페이지 크기를 사용하여 새 마운트를 생성할 수 있도록 합니다.

c.
폴더가 먼저 생성되었는지 확인합니다.

d.
각 **pagesize** 에 대해 **systemd** 마운트 단위를 생성합니다.

e.
첫 번째 루프 후에 **systemd daemon-reload** 를 실행하여 새로 생성된 장치 파일을 포함합니다.

f.
2M 및 **1G** 페이지 크기에 대한 각 마운트를 활성화합니다. 필요에 따라 추가 페이지 크기를 포함하도록 이 루프를 업데이트할 수 있습니다.

2.
선택 사항: **/dev** 폴더는 **nova_compute** 및 **nova_libvirt** 컨테이너에 자동으로 바인딩됩니다. 대규모 페이지 마운트에 다른 대상을 사용한 경우, **nova_compute** 및 **nova_libvirt** 컨테이너에 마운트를 전달해야 합니다.

```
parameter_defaults
  NovaComputeOptVolumes:
    - /opt/dev:/opt/dev
  NovaLibvirtOptVolumes:
    - /opt/dev:/opt/dev
```

3.
~/templates/firstboot.yaml 환경 파일에 **OS::TripleO::NodeUserData** 리소스 유형으로 **heat** 템플릿을 등록합니다.

```
resource_registry:
  OS::TripleO::NodeUserData: ./hugepages.yaml
```

중요

NodeUserData 리소스를 각 리소스에 대해 하나의 **heat** 템플릿에만 등록할 수 있습니다. 후속 사용은 사용할 **heat** 템플릿을 덮어씁니다.

4.
첫 번째 부팅 환경 파일을 다른 환경 파일과 함께 스택에 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/firstboot.yaml \
  ...
```

3.4. 인스턴스에 파일 지원 메모리를 사용하도록 컴퓨팅 노드 구성

libvirt 메모리 지원 디렉터리 내의 파일을 인스턴스 메모리로 할당하여 파일 지원 메모리를 사용하여 컴퓨팅 노드 메모리 용량을 확장할 수 있습니다. 인스턴스 메모리에 사용할 수 있는 호스트 디스크의 양과 인스턴스 메모리 파일의 디스크 위치를 구성할 수 있습니다.

계산 서비스는 파일 지원 메모리에 구성된 용량을 배치 서비스에 총 시스템 메모리 용량으로 보고합니다. 이렇게 하면 **Compute** 노드에서 일반적으로 시스템 메모리에 맞는 것보다 더 많은 인스턴스를 호스팅할 수 있습니다.

인스턴스에 파일 지원 메모리를 사용하려면 컴퓨팅 노드에서 파일 지원 메모리를 활성화해야 합니다.

제한

- 파일 지원 메모리가 활성화된 컴퓨팅 노드와 파일 지원 메모리가 활성화되어 있지 않은 컴퓨팅 노드 간에 인스턴스를 실시간 마이그레이션할 수 없습니다.
- 파일 지원 메모리는 대규모 페이지와 호환되지 않습니다. 대규모 페이지를 사용하는 인스턴스는 파일 지원 메모리가 활성화된 컴퓨팅 노드에서 시작할 수 없습니다. 호스트 집계를 사용하여 대규모 페이지를 사용하는 인스턴스가 파일 지원 메모리가 활성화된 컴퓨팅 노드에 배치되지 않았는지 확인합니다.
- 파일 지원 메모리는 메모리 과다 할당과 호환되지 않습니다.
- **NovaReservedHostMemory** 를 사용하여 호스트 프로세스의 메모리를 예약할 수 없습니다. 파일 지원 메모리가 사용 중인 경우 예약된 메모리는 파일 지원 메모리에 대해 별도로 설정되지 않은 디스크 공간에 해당합니다. 파일 지원 메모리는 캐시 메모리로 사용된 총 시스템 메모리로 배치 서비스에 보고됩니다.

사전 요구 사항

- **NovaRAMAllocationRatio** 는 노드에서 **"1.0"**으로 설정하고 노드가 추가된 호스트를 집계해야 합니다.
- **NovaReservedHostMemory** 를로 설정해야 합니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. **Compute** 환경 파일에 다음 매개 변수를 추가하여 **RAM** 인스턴스에 사용할 수 있도록 호스트 디스크 공간**(MiB)**을 구성합니다.

```
parameter_defaults:
  NovaLibvirtFileBackedMemory: 102400
```

3. 선택 사항**:** 메모리 백업 파일을 저장하도록 디렉터리를 구성하려면 컴퓨팅 환경 파일에 **QemuMemoryBackingDir** 매개변수를 설정합니다. 설정되지 않은 경우 메모리 지원 디렉토리의 기본값은 **/var/lib/libvirt/qemu/ram/** 입니다.

참고

기본 디렉토리 위치 **/var/lib/libvirt/qemu/ram/** 의 디렉토리에서 백업 저장소를 찾아야 합니다.

백업 저장소의 호스트 디스크를 변경할 수도 있습니다. 자세한 내용은 메모리 백업 디렉터리 호스트 디스크 변경을 참조하십시오.

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

3.4.1. 메모리 백업 디렉터리 호스트 디스크 변경

메모리 지원 디렉터리를 기본 기본 디스크 위치에서 대체 디스크로 이동할 수 있습니다.

절차

1. 대체 백업 장치에서 파일 시스템을 만듭니다. 예를 들어 **/dev/sdb** 에 **ext4** 파일 시스템을 생성하려면 다음 명령을 입력합니다.

```
# mkfs.ext4 /dev/sdb
```

2. **백업 장치를 마운트합니다. 예를 들어 기본 libvirt 메모리 백업 디렉터리에 /dev/sdb 를 마운트하려면 다음 명령을 입력합니다.**

```
# mount /dev/sdb /var/lib/libvirt/qemu/ram
```

참고

마운트 지점은 QemuMemoryBackingDir 매개변수 값과 일치해야 합니다.

4장. 계산 서비스 스토리지 구성

계산 서비스에서 이미지(glance) 서비스에서 복사하고 Compute 노드에 로컬로 캐시하는 기본 이미지에서 인스턴스를 생성합니다. 인스턴스의 백엔드인 인스턴스 디스크도 기본 이미지를 기반으로 합니다.

호스트 Compute 노드에 로컬로 임시 인스턴스 디스크 데이터를 저장하거나 NFS 공유 또는 Ceph 클러스터에서 원격으로 계산 서비스를 구성할 수 있습니다. 또는 인스턴스 디스크 데이터를 블록 스토리지(Cinder) 서비스에서 제공하는 영구 스토리지에 저장하도록 계산 서비스를 구성할 수도 있습니다.

환경에 대한 이미지 캐싱을 구성하고 인스턴스 디스크의 성능 및 보안을 구성할 수 있습니다. Image 서비스(glance)에서 Red Hat Ceph RADOS Block Device(RBD)를 백엔드로 사용하는 경우 Image 서비스 API를 사용하지 않고 RBD 이미지 리포지토리에서 직접 이미지를 다운로드하도록 계산 서비스를 구성할 수도 있습니다.

4.1. 이미지 캐싱을 위한 구성 옵션

다음 표에 설명된 매개 변수를 사용하여 계산 노드에서 이미지 캐시를 구현하고 관리하는 방법을 구성합니다.

표 4.1. Compute(nova) 서비스 이미지 캐시 매개변수

구성 방법	매개변수	설명
Puppet	**nova::compute::image_cache::manager_interval**	컴퓨팅 노드에서 기본 이미지 캐싱을 관리하는 이미지 캐시 관리자의 실행 간격을 대기하는 시간(초)을 지정합니다. **nova::compute::image_cache::remove_unused_base_images** 가 **True** 로 설정된 경우 계산 서비스는 이 기간을 사용하여 사용되지 않은 캐시된 이미지를 자동으로 제거합니다. 기본 지표 간격 60초(권장되지 않음)에서 실행되도록 **0** 으로 설정합니다. 이미지 캐시 관리자를 비활성화하려면 **-1** 로 설정합니다. 기본값: **2400**

구성 방법	매개변수	설명
Puppet	**nova::compute::image_cache::precache_concurrency**	이미지를 병렬로 사전 캐시할 수 있는 최대 컴퓨팅 노드 수를 지정합니다.  **참고** ● 이 매개 변수를 높은 수로 설정하면 pre-cache 성능이 저하될 수 있으며 이미지 서비스에서 플레이버가 발생할 수 있습니다. ● 이 매개 변수를 낮은 숫자로 설정하면 이미지 서비스의 부하가 줄어들지만, 캐시가 더 순차적으로 수행되므로 더 긴 런타임이 완료될 수 있습니다. 기본값: **1**
Puppet	**nova::compute::image_cache::remove_unused_base_images**	**manager_interval** 을 사용하여 구성한 간격으로 사용되지 않는 기본 이미지를 캐시에서 자동으로 제거하려면 **True** 로 설정합니다. **NovaImageCacheTTL** 을 사용하여 지정한 시간 동안 액세스하지 않은 경우 이미지는 사용되지 않은 것으로 정의됩니다. 기본값: **True**
Puppet	**nova::compute::image_cache::remove_unused_resized_minimum_age_seconds**	사용되지 않은 크기 조정 기본 이미지를 캐시에서 제거해야 하는 최소 기간(초)을 지정합니다. 이보다 오래된 사용되지 않은 크기 조정된 기본 이미지는 제거되지 않습니다. disable를 설정하려면 **undef** 로 설정합니다. 기본값: **3600**
Puppet	**nova::compute::image_cache::subdirectory_name**	**$instances_path** 를 기준으로 캐시된 이미지가 저장된 폴더의 이름을 지정합니다. 기본값: **_base**

구성 방법	매개변수	설명
Heat	**NovaImageCacheTTL**	Compute 노드에서 더 이상 사용하지 않는 경우 이미지를 계속 캐싱해야 하는 시간(초)을 지정합니다. 계산 서비스는 캐시 디렉터리에서 구성된 수명보다 오래된 컴퓨팅 노드에 캐시된 이미지를 다시 필요할 때까지 삭제합니다. 기본값: 86400 (24 시간)

4.2. 인스턴스 임시 스토리지 속성에 대한 구성 옵션

다음 표에 자세히 설명된 매개 변수를 사용하여 인스턴스에서 사용하는 임시 스토리지의 성능 및 보안을 구성합니다.

참고

RHOSP(Red Hat OpenStack Platform)는 인스턴스 디스크의 **LVM** 이미지 유형을 지원하지 않습니다. 따라서 인스턴스가 삭제될 때 임시 디스크를 제거하는 **[libvirt]/volume_clear** 구성 옵션은 인스턴스 디스크 이미지 유형이 **LVM**인 경우에만 적용되므로 지원되지 않습니다.

표 **4.2. Compute(nova)** 서비스 인스턴스 임시 스토리지 매개변수

구성 방법	매개변수	설명
Puppet	**nova::compute::default_ephemeral_format**	새 임시 볼륨에 사용되는 기본 형식을 지정합니다. 다음 유효한 값 중 하나로 설정합니다. • **ext2** • **ext3** • **ext4** **ext4** 형식은 새 대규모 디스크에 대해 ext3보다 훨씬 빠른 초기화 시간을 제공합니다. 기본값: **ext4**

구성 방법	매개변수	설명
Puppet	**nova::compute::force_raw_images**	공개되지 않은 캐시된 기본 이미지를 원시 형식으로 변환하려면 **True** 로 설정합니다. raw 이미지 형식은 qcow2와 같은 다른 이미지 형식보다 더 많은 공간을 사용합니다. 비종료 이미지 형식은 압축에 더 많은 CPU를 사용합니다. **False** 로 설정하면 계산 서비스에서 CPU 병목 현상이 발생하지 않도록 압축 중에 기본 이미지에서 압축을 제거합니다. 입력 대역폭을 줄이기 위해 느린 I/O 또는 사용 가능한 공간이 있는 시스템이 있는 경우 **False** 로 설정합니다. 기본값: **True**
Puppet	**nova::compute::use_cow_images**	인스턴스 디스크에 대해 qcow2 형식의 CoW(Copy on Write) 이미지를 사용하려면 **True** 로 설정합니다. CoW를 사용하면 백업 저장소와 호스트 캐싱에 따라 각 인스턴스가 고유한 사본에서 작동하도록 함으로써 동시성이 향상될 수 있습니다. 원시 형식을 사용하려면 **False** 로 설정합니다. 원시 형식은 디스크 이미지의 공통 부분에 더 많은 공간을 사용합니다. 기본값: **True**
Puppet	**nova::compute::libvirt::preallocate_images**	인스턴스 디스크에 대한 사전 할당 모드를 지정합니다. 다음 유효한 값 중 하나로 설정합니다. ● **none** - 인스턴스를 시작할 때 스토리지가 프로비저닝되지 않습니다. ● **space** - 계산 서비스는 인스턴스 디스크 이미지에서 **fallocate(1)** 를 실행하여 시작할 때 인스턴스의 스토리지를 완전히 할당합니다. 이렇게 하면 CPU 오버헤드 및 파일 조각화가 줄어들고 I/O 성능이 향상되며 필요한 디스크 공간이 보장됩니다. 기본값: **none**

구성 방법	매개변수	설명
hieradata 덮어쓰기	**DEFAULT/resize_fs_using_block_device**	블록 장치를 통해 이미지에 액세스하여 기본 이미지의 직접 크기 조정을 활성화하려면 **True** 로 설정합니다. 이는 자체 크기를 조정할 수 없는 이전 버전의 cloud-init가 있는 이미지에만 필요합니다. 보안상의 이유로 비활성화될 수 있는 이미지를 직접 마운트할 수 있으므로 이 매개변수는 기본적으로 활성화되어 있지 않습니다. 기본값: **False**
hieradata 덮어쓰기	**[libvirt]/images_type**	인스턴스 디스크에 사용할 이미지 유형을 지정합니다. 다음 유효한 값 중 하나로 설정합니다. ● **raw** ● **qcow2** ● **flat** ● **rbd** ● **default**  **참고** RHOSP는 인스턴스 디스크에 LVM 이미지 유형을 지원하지 않습니다. **기본값** 이외의 유효한 값으로 설정하면 이미지 유형이 **use_cow_images** 구성을 대체합니다. **default** 가 지정되면 **use_cow_images 구성에서** 이미지 유형을 결정합니다. ● **use_cow_images** 가 **True** (기본값)로 설정된 경우 이미지 유형은 **qcow2** 입니다. ● **use_cow_images** 를 **False** 로 설정하면 이미지 유형은 **Flat** 입니다. 기본값은 **NovaEnableRbdBackend** 구성으로 결정됩니다. ● **NovaEnableRbdBackend: False** 기본값: **default** ● **NovaEnableRbdBackend: True** 기본값: **rbd**

4.3. 공유 인스턴스 스토리지 구성

기본적으로 인스턴스를 시작하면 인스턴스 디스크가 인스턴스 디렉터리 /var/lib/nova/instances 에 파일로 저장됩니다. 이러한 인스턴스 파일을 공유 NFS 스토리지에 저장하도록 Compute 서비스의 NFS 스토리지 백엔드를 구성할 수 있습니다.

사전 요구 사항

- NFSv4 이상을 사용해야 합니다. RHOSP(Red Hat OpenStack Platform)는 이전 버전의 NFS를 지원하지 않습니다. 자세한 내용은 Red Hat Knowledgebase 솔루션 RHOS NFSv4-Only Support Notes 에서 참조하십시오.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.

2. stackrc 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. 환경 파일을 만들어 공유 인스턴스 스토리지를 구성합니다(예: nfs_instance_disk_backend.yaml).

4. 인스턴스 파일의 NFS 백엔드를 구성하려면 nfs_instance_disk_backend.yaml에 다음 구성을 추가합니다.

```
parameter_defaults:
  ...
  NovaNfsEnabled: True
  NovaNfsShare: <nfs_share>
```

<nfs_share> 를 인스턴스 파일 스토리지용으로 마운트할 NFS 공유 디렉토리(예: '192.168.122.1:/export/nova' 또는 '192.168.24.1:/var/nfs' 로 바꿉니다. IPv6를 사용하는 경우 이중 및 단일 따옴표를 모두 사용합니다(예: '"[fdd0::1]:/export/nova' ").

5. 선택 사항: NFS 백엔드 스토리지가 활성화된 경우 NFS 스토리지에 대한 기본 SELinux 컨텍스트는 'context=system_u:object_r:nfs_t:s0' 입니다. 다음 매개 변수를 추가하여 NFS 인스턴스 파일 스토리지 마운트 지점에 대한 마운트 옵션을 수정합니다.

```
parameter_defaults:
  ...
  NovaNfsOptions: 'context=system_u:object_r:nfs_t:s0,
<additional_nfs_mount_options>'
```

<additional_nfs_mount_options> 를 **NFS** 인스턴스 파일 스토리지에 사용할 마운트 옵션의 쉼표로 구분된 목록으로 바꿉니다. 사용 가능한 마운트 옵션에 대한 자세한 내용은 **mount** 도움말 페이지를 참조하십시오.

```
$ man 8 mount.
```

6. 업데이트를 환경 파일에 저장합니다.

7. 다른 환경 파일을 사용하여 스택에 새 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/nfs_instance_disk_backend.yaml
```

4.4. RED HAT CEPH RADOS BLOCK DEVICE (RBD)에서 직접 이미지 다운로드 구성

이미지 서비스(glance)에서 **Red Hat Ceph RADOS Block Device(RBD)**를 백엔드로 사용하고 **Compute** 서비스에서 로컬 파일 기반 임시 스토리지를 사용하는 경우, 이미지 서비스 **API**를 사용하지 않고 **RBD** 이미지 리포지토리에서 직접 이미지를 다운로드하도록 **Compute** 서비스를 구성할 수 있습니다. 이렇게 하면 인스턴스 부팅 시 이미지를 **Compute** 노드 이미지 캐시에 다운로드하는 데 걸리는 시간이 단축되어 인스턴스 시작 시간이 개선됩니다.

사전 요구 사항

- 이미지 서비스 백엔드는 **Red Hat Ceph RADOS** 블록 장치(RBD)입니다.
- 계산 서비스는 이미지 캐시 및 인스턴스 디스크에 로컬 파일 기반 임시 저장소를 사용하고 있습니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. 컴퓨팅 환경 파일을 엽니다.

3. **RBD** 백엔드에서 직접 이미지를 다운로드하려면 **Compute** 환경 파일에 다음 구성을 추가합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaGlanceEnableRbdDownload: True
    NovaEnableRbdBackend: False
    ...
```

4. 선택 사항**:** 여러 **Red Hat Ceph Storage** 백엔드를 사용하도록 이미지 서비스가 구성된 경우 **Compute** 환경 파일에 다음 구성을 추가하여 이미지를 다운로드할 **RBD** 백엔드를 식별합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaGlanceEnableRbdDownload: True
    NovaEnableRbdBackend: False
    NovaGlanceRbdDownloadMultistoreID: <rbd_backend_id>
    ...
```

<rbd_backend_id> 를 **GlanceMultistoreConfig** 구성에서 백엔드를 지정하는 데 사용되는 **ID(**예**: rbd2_store)**로 바꿉니다.

5. **Compute** 환경 파일에 다음 구성을 추가하여 이미지 서비스 **RBD** 백엔드와 계산 서비스에서 이미지 서비스 **RBD** 백엔드에 연결하도록 대기하는 최대 시간을 초 단위로 지정합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      glance/rbd_user:
        value: 'glance'
      glance/rbd_pool:
        value: 'images'
      glance/rbd_ceph_conf:
        value: '/etc/ceph/ceph.conf'
      glance/rbd_connect_timeout:
        value: '5'
```

6. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

7. 계산 서비스가 **RBD**에서 직접 이미지를 다운로드하는지 확인하려면 인스턴스를 만든 다음 인스턴스 디버그 로그에서 **"Attempting to export RBD image:"** 항목이 있는지 확인합니다.

4.5. 추가 리소스

- **Compute 서비스(nova) 구성**

5장. PCI 패스스루 구성

PCI 통과를 사용하여 그래픽 카드 또는 네트워크 장치와 같은 물리적 PCI 장치를 인스턴스에 연결할 수 있습니다. 장치에 PCI 패스스루를 사용하는 경우 인스턴스는 작업을 수행하기 위해 장치에 대한 전용 액세스를 예약하고 호스트에서 장치를 사용할 수 없습니다.

중요

라우팅된 공급자 네트워크에서 PCI 패스스루 사용

계산 서비스는 여러 프로바이더 네트워크에 걸쳐 있는 단일 네트워크를 지원하지 않습니다. 네트워크에 여러 물리적 네트워크가 포함된 경우 계산 서비스에서는 첫 번째 실제 네트워크만 사용합니다. 따라서 라우팅된 공급자 네트워크를 사용하는 경우 모든 컴퓨팅 노드에서 동일한 **physical_network** 이름을 사용해야 합니다.

VLAN 또는 플랫 네트워크에서 라우팅된 공급자 네트워크를 사용하는 경우 모든 세그먼트에 동일한 **physical_network** 이름을 사용해야 합니다. 그런 다음 네트워크에 대한 여러 세그먼트를 만들고 세그먼트를 적절한 서브넷에 매핑합니다.

클라우드 사용자가 PCI 장치가 연결된 인스턴스를 생성할 수 있도록 하려면 다음을 완료해야 합니다.

1. PCI 패스스루를 위해 컴퓨팅 노드를 지정합니다.
2. 필요한 PCI 장치가 있는 PCI 패스스루에 대해 컴퓨팅 노드를 구성합니다.
3. Overcloud를 배포합니다.
4. PCI 장치가 연결된 인스턴스를 시작하기 위한 플레이버를 만듭니다.

사전 요구 사항

- 컴퓨팅 노드에는 필수 PCI 장치가 있습니다.

5.1. PCI 패스스루를 위한 컴퓨팅 노드 지정

물리적 PCI 장치가 연결된 인스턴스용으로 Compute 노드를 지정하려면 새 역할 파일을 생성하여 PCI 패스스루 역할을 구성하고 PCI 패스스루를 위해 Compute 노드에 태그를 지정하는 데 사용할 새 오버클라우드 플레이버 및 PCI 통과 리소스 클래스를 구성해야 합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.

2. stackrc 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. Controller,Compute 및 Compute PCI 역할이 포함된 roles_data_pci_passthrough.yaml 이라는 새 역할 데이터 파일을 생성합니다.

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_pci_passthrough.yaml \
 Compute:ComputePCI Compute Controller
```

4. roles_data_pci_passthrough.yaml 을 열고 다음 매개변수 및 섹션을 편집하거나 추가합니다.

섹션/패랜드	현재 값	새 값
역할 설명	역할: 컴퓨팅	역할: ComputePCI
역할 이름	name: 컴퓨팅	name: ComputePCI
description	기본 컴퓨팅 노드 역할	PCI 통과 계산 노드 역할
HostnameFormatDefault	%stackname%-novacompute-%index%	%stackname%-novacomputepci-%index%
deprecated_nic_config_name	compute.yaml	compute-pci-passthrough.yaml

5. 노드 정의 템플릿 node. json 또는 node. yaml에 추가하여 오버클라우드의 PCI 패스스루 컴퓨팅 노드를 등록합니다. 자세한 내용은 *Director 설치 및 사용* 가이드 의 오버클라우드 노드 등록을 참조하십시오.

6. 노드 하드웨어를 검사합니다.

```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

자세한 내용은 ***Director 설치 및 사용*** 가이드 의 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

7. **PCI** 패스스루 컴퓨팅 노드의 **compute-pci-passthrough** 오버클라우드 플레이버를 생성합니다.

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> compute-pci-passthrough
```

- **<ram_size_mb>** 를 베어 메탈 노드의 **RAM(MB)**으로 바꿉니다.
- **<disk_size_gb>** 를 베어 메탈 노드의 디스크 크기**(GB)**로 바꿉니다.
- **<no_vcpus>** 를 베어 메탈 노드의 **CPU** 수로 바꿉니다.

참고

이러한 속성은 인스턴스를 예약하는 데 사용되지 않습니다. 그러나 계산 스케줄러는 디스크 크기를 사용하여 루트 파티션 크기를 결정합니다.

8. 사용자 정의 **PCI** 패스스루 리소스 클래스를 사용하여 **PCI** 패스스루를 지정하려는 각 베어 메탈 노드에 태그를 지정합니다.

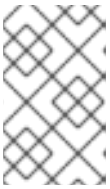
```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.PCI-PASSTHROUGH <node>
```

<node> 를 베어 메탈 노드의 **ID**로 바꿉니다.

9. **compute-pci-passthrough** 플레이버를 사용자 지정 **PCI** 패스스루 리소스 클래스와 연결합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_PCI_PASSTHROUGH=1 \
 compute-pci-passthrough
```

베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 모든 문장 부호를 밑줄로 바꾸고 접두사는 **CUSTOM_** 로 바꿉니다.



참고

플레이버는 베어 메탈 리소스 클래스의 인스턴스 하나만 요청할 수 있습니다.

10. **Compute** 스케줄러가 베어 메탈 플레이버 속성을 사용하여 인스턴스를 예약하지 못하도록 다음 플레이버 속성을 설정합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 compute-pci-passthrough
```

11. **node-info.yaml** 파일에 다음 매개변수를 추가하여 **PCI** 패스스루 컴퓨팅 노드 수와 **PCI** 패스스루 지정 컴퓨팅 노드에 사용할 플레이버를 지정합니다.

```
parameter_defaults:
  OvercloudComputePCIFlavor: compute-pci-passthrough
  ComputePCICount: 3
```

12. 역할이 생성되었는지 확인하려면 다음 명령을 입력합니다.

```
(undercloud)$ openstack overcloud profiles list
```

5.2. PCI 패스스루 컴퓨팅 노드 구성

클라우드 사용자가 **PCI** 장치가 연결된 인스턴스를 생성할 수 있도록 하려면 **PCI** 장치와 컨트롤러 노드가 있는 컴퓨팅 노드 모두를 구성해야 합니다.

절차

1. 환경 파일을 생성하여 **PCI** 패스스루의 오버클라우드에서 컨트롤러 노드를 구성합니다(예:

pci_passthrough_controller.yaml).

2. **pci_pass through_controller.yaml**의 **NovaSchedulerDefaultFilters** 매개변수에 **PciPass throughFilter**를 추가합니다.

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','Serve
rGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```

3. 컨트롤러 노드에서 장치의 **PCI** 별칭을 지정하려면 **pci_passthrough_controller.yaml**에 다음 구성을 추가합니다.

```
parameter_defaults:
  ...
  ControllerExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
```

device_type 필드 구성에 대한 자세한 내용은 **PCI** 패스스루 장치 유형 필드를 참조하십시오.

참고

nova-api 서비스가 **Controller** 역할과 다른 역할에서 실행 중인 경우 **Controller ExtraConfig** 를 **<Role>ExtraConfig** 형식의 사용자 역할로 교체합니다.

4. 선택 사항**: PCI** 패스스루 장치에 대한 기본 **NUMA** 선호도 정책을 설정하려면 **3**단계의 **nova::pci::aliases:** 구성에 **numa_policy** 를 추가합니다.

```
parameter_defaults:
  ...
  ControllerExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
        numa_policy: "preferred"
```

5.

PCI 패스스루에 대해 오버클라우드에서 **Compute** 노드를 구성하려면 환경 파일(예: **pci_passthrough_compute.yaml**)을 생성합니다.

6.

Compute 노드에서 장치에 사용 가능한 **PCI**를 지정하려면 **vendor_id** 및 **product_id** 옵션을 사용하여 인스턴스 통과에 사용할 수 있는 **PCI** 장치 풀에 일치하는 모든 **PCI** 장치를 추가합니다. 예를 들어 인스턴스에 대한 패스스루에 사용할 수 있는 **PCI** 장치 풀에 **Intel®** 이더넷 컨트롤러 **X710** 장치를 추가하려면 **pci_passthrough_compute.yaml**에 다음 구성을 추가합니다.

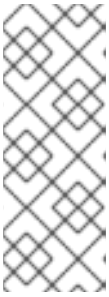
```
parameter_defaults:
  ...
  ComputePCIParameters:
    NovaPCIPassthrough:
      - vendor_id: "8086"
        product_id: "1572"
```

NovaPCIPassthrough 구성 방법에 대한 자세한 내용은 **NovaPCIPassthrough** 구성을 위한 지침을 참조하십시오 .

7.

인스턴스 마이그레이션 및 크기 조정 작업을 위해 컴퓨팅 노드에 **PCI** 별칭 사본을 생성해야 합니다. **PCI** 패스스루 컴퓨팅 노드에서 장치의 **PCI** 별칭을 지정하려면 **pci_passthrough_compute.yaml**에 다음을 추가합니다.

```
parameter_defaults:
  ...
  ComputePCIExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
```



참고

컴퓨팅 노드 별칭은 컨트롤러 노드의 별칭과 동일해야 합니다. 따라서 **pci_passthrough _ controller.yaml**의 **nova::pci::aliases** 에 **numa_affinity** 를 추가한 경우 **pci_ passthrough_compute.yaml** 의 **nova::pci::aliases** 에도 추가해야 합니다.

8.

Compute 노드의 서버 **BIOS**에서 **IOMMU**를 활성화하여 **PCI** 패스스루를 지원하려면 **KernelArgs** 매개변수를 **pci_passthrough_compute.yaml** 에 추가합니다. 예를 들어 다음 **KernalArgs** 설정을 사용하여 **Intel IOMMU**를 활성화합니다.

```
parameter_defaults:
```

```
...
ComputePCIParameters:
  KernelArgs: "intel_iommu=on iommu=pt"
```

AMD IOMMU를 사용하려면 **KernelArgs** 를 **"amd_iommu=on iommu=pt"** 로 설정합니다.

참고

KernelArgs 매개변수를 역할 구성에 처음 추가하면 오버클라우드 노드가 자동으로 재부팅됩니다. 필요한 경우 노드 자동 재부팅을 비활성화하고 대신 각 오버클라우드 배포 후 노드를 수동으로 재부팅할 수 있습니다. 자세한 내용은 **KernelArgs**를 정의하도록 수동 노드 재부팅 구성을 참조하십시오.

9. 다른 환경 파일을 사용하여 스택에 사용자 지정 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/pci_passthrough_controller.yaml \
  -e /home/stack/templates/pci_passthrough_compute.yaml \
```

10. 클라우드 사용자가 **PCI** 장치를 요청하는 데 사용할 수 있는 플레이버를 생성하고 구성합니다. 다음 예제에서는 각각 **7**단계에 정의된 별칭을 사용하여 공급업체 **ID**가 **8086** 이고 제품 **ID**가 **1572** 인 두 개의 장치를 요청합니다.

```
(overcloud)# openstack flavor set \
 --property "pci_passthrough:alias"="a1:2" device_passthrough
```

11. 선택 사항: **PCI** 통과 장치의 기본 **NUMA** 선호도 정책을 재정의하려면 플레이버 또는 이미지에 **NUMA** 선호도 정책 키를 추가할 수 있습니다.

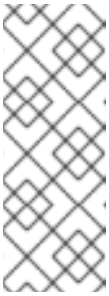
- 플레이버를 사용하여 기본 **NUMA** 선호도 정책을 재정의하려면 **hw:pci_numa_affinity_policy** 속성 키를 추가합니다.

```
(overcloud)# openstack flavor set \
 --property "hw:pci_numa_affinity_policy"="required" \
 device_passthrough
```

hw:pci_numa_affinity_policy 의 유효한 값에 대한 자세한 내용은 **Flavor metadata** 를 참조하십시오.

- 이미지를 사용하여 기본 **NUMA** 선호도 정책을 재정의하려면 **hw_pci_numa_affinity_policy** 속성 키를 추가합니다.

```
(overcloud)# openstack image set \
 --property hw_pci_numa_affinity_policy=required \
 device_passthrough_image
```



참고

이미지와 플레이버 모두에 **NUMA** 선호도 정책을 설정하면 속성 값이 일치해야 합니다. 플레이버 설정이 이미지와 기본 설정보다 우선합니다. 따라서 이미지에 대한 **NUMA** 선호도 정책의 구성은 플레이버에 특성이 설정되지 않은 경우에만 적용됩니다.

검증

1. **PCI** 패스스루 장치를 사용하여 인스턴스를 생성합니다.

```
# openstack server create --flavor device_passthrough \
 --image <image> --wait test-pci
```

2. 클라우드 사용자로 인스턴스에 로그인합니다. 자세한 내용은 인스턴스에 연결을 참조하십시오.

3. 인스턴스에서 **PCI** 장치에 액세스할 수 있는지 확인하려면 인스턴스에서 다음 명령을 입력합니다.

```
$ lspci -nn | grep <device_name>
```

5.3. PCI 패스스루 장치 유형 필드

계산 서비스는 장치가 보고하는 기능에 따라 **PCI** 장치를 세 가지 유형 중 하나로 분류합니다. 다음은 **device_type** 필드를 로 설정할 수 있는 유효한 값을 나열합니다.

type-PF

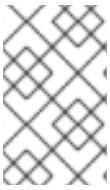
장치는 **SR-IOV**를 지원하며 상위 또는 루트 장치입니다. 전체에서 **SR-IOV**를 지원하는 장치를 통과하도록 이 장치 유형을 지정합니다.

type-VF

장치는 **SR-IOV**를 지원하는 장치의 하위 장치입니다.

type-PCI

장치는 **SR-IOV**를 지원하지 않습니다. **device_type** 필드가 설정되지 않은 경우 기본 장치 유형입니다.



참고

동일한 **device_type** 으로 컴퓨팅 및 컨트롤러 노드를 구성해야 합니다.

5.4. NOVAPCIPASSTHROUGH구성을 위한 지침

- **NIC**의 장치 이름이 변경될 수 있으므로 **PCI** 패스스루를 구성할 때 **devname** 매개변수를 사용하지 마십시오. 대신, 더 안정적이므로 **vendor_id** 및 **product_id** 를 사용하거나 **NIC** 주소를 사용합니다.
- 특정 **PF**(물리 기능)를 통과하려면 **PCI** 주소는 각 장치에 고유하므로 **address** 매개변수를 사용할 수 있습니다. 또는 **product_id** 매개변수를 사용하여 **PF**를 통과할 수 있지만 동일한 유형의 **PF**가 여러 개인 경우 **PF**의 주소를 지정해야 합니다.
- 모든 **VF**(가상 기능)를 통과하려면 **PCI** 패스스루에 사용하려는 **VF**의 **product _id** 및 **vendor_id** 만 지정합니다. **NIC** 파티션에 **SRIOV**를 사용하고 **VF**에서 **OVS**를 실행하는 경우 **VF**의 주소 도 지정해야 합니다.
- **PF** 자체는 아닌 **PF**에 대한 **VF**만 전달하려면 **address** 매개변수를 사용하여 **PF** 및 **product_id** 의 **PCI** 주소를 지정하여 **VF**의 제품 **ID**를 지정할 수 있습니다.

address 매개변수 구성

address 매개 변수는 장치의 **PCI** 주소를 지정합니다. **String** 또는 **dict** 매핑을 사용하여 **address** 매개변수의 값을 설정할 수 있습니다.

문자열 형식

문자열을 사용하여 주소를 지정하는 경우 다음 예와 같이 와일드카드(*)를 포함할 수 있습니다.

```
NovaPCIPassthrough:
 -
```

```
    address: "*:0a:00.*"
    physical_network: physnet1
```

사전 형식

사전 형식을 사용하여 주소를 지정하는 경우 다음 예와 같이 정규식 구문을 포함할 수 있습니다.

```
NovaPCIPassthrough:
 -
    address:
      domain: ".*"
      bus: "02"
      slot: "01"
      function: "[0-2]"
    physical_network: net1
```

참고

Compute 서비스는 **address** 필드의 구성을 다음 최대값으로 제한합니다.

- **domain - 0xFFFF**
- **버스 - 0xFF**
- **슬롯 - 0x1F**
- **함수 - 0x7**

Compute 서비스는 **16**비트 주소 도메인이 있는 **PCI** 장치를 지원합니다. **Compute** 서비스는 **32**비트 주소 도메인을 사용하여 **PCI** 장치를 무시합니다.

6장. 호스트 집계 생성 및 관리

클라우드 관리자는 성능 또는 관리상의 목적으로 계산 배포를 논리 그룹으로 분할할 수 있습니다. **RHOSP(Red Hat OpenStack Platform)**는 논리 그룹을 파티셔닝하기 위한 다음과 같은 메커니즘을 제공합니다.

호스트 집계

호스트 집계는 하드웨어 또는 성능 특성과 같은 속성을 기반으로 계산 노드를 논리 단위로 그룹화하는 것입니다. 하나 이상의 호스트 집계에 컴퓨팅 노드를 할당할 수 있습니다.

호스트 집계에서 메타데이터를 설정한 다음 플레이버 추가 사양 또는 이미지 메타데이터 속성을 호스트 집계에 일치시켜 플레이버 및 이미지를 호스트 집계에 매핑할 수 있습니다. 계산 스케줄러는 이 메타데이터를 사용하여 필요한 필터를 활성화할 때 인스턴스를 예약할 수 있습니다. 호스트 집계에 지정하는 메타데이터는 해당 호스트의 사용을 플레이버 또는 이미지에 지정된 동일한 메타데이터를 가진 인스턴스로 제한합니다.

호스트 집계 메타데이터에서 **xxx_weight_multiplier** 구성 옵션을 설정하여 각 호스트 집계의 가중치 승수를 구성할 수 있습니다.

호스트 집계를 사용하여 부하 분산을 처리하거나, 물리적 격리 또는 중복을 적용하거나, 공통 속성을 사용하여 서버를 그룹화하거나, 하드웨어 클래스를 구분할 수 있습니다.

호스트 집계를 생성할 때 영역 이름을 지정할 수 있습니다. 이 이름은 클라우드 사용자에게 선택할 수 있는 가용성 영역으로 제공됩니다.

가용성 영역

가용 영역은 호스트 집계의 클라우드 사용자 보기입니다. 클라우드 사용자는 가용성 영역에서 컴퓨팅 노드를 보거나 가용성 영역의 메타데이터를 볼 수 없습니다. 클라우드 사용자는 가용성 영역의 이름만 볼 수 있습니다.

각 컴퓨팅 노드를 하나의 가용성 영역에만 할당할 수 있습니다. 클라우드 사용자가 영역을 지정하지 않을 때 인스턴스가 예약되는 기본 가용성 영역을 구성할 수 있습니다. 특정 기능이 있는 가용 영역을 사용하도록 클라우드 사용자에게 지시할 수 있습니다.

6.1. 호스트 집계에서 예약 활성화

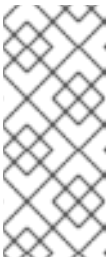
특정 특성이 있는 호스트 집계에 인스턴스를 예약하려면 **Compute Scheduler**의 구성을 업데이트하여 호스트 집계 메타데이터에 따라 필터링을 활성화합니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 다음 값을 **NovaSchedulerDefaultFilters** 매개변수에 추가합니다.

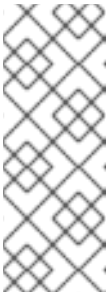
- **AggregateInstanceExtraSpecsFilter:** 이 값을 추가하여 플레이버 추가 사양과 일치하는 호스트 집계 메타데이터로 **Compute** 노드를 필터링합니다.



참고

이 필터가 예상대로 수행하려면 **extra_specs** 키 앞에 **aggregate_instance_extra_specs:** 네임스페이스를 추가하여 플레이버 추가 사양의 범위를 지정해야 합니다.

- **AggregateImagePropertiesIsolation:** 이 값을 추가하여 이미지 메타데이터 속성과 일치하는 호스트 집계 메타데이터로 컴퓨팅 노드를 필터링합니다.



참고

이미지 메타데이터 속성을 사용하여 호스트 집계 메타데이터를 필터링하려면 호스트 집계 메타데이터 키가 유효한 이미지 메타데이터 속성과 일치해야 합니다. 유효한 이미지 메타데이터 속성에 대한 자세한 내용은 이미지 메타데이터 를 참조하십시오.

- **AvailabilityZoneFilter:** 인스턴스를 시작할 때 가용성 영역별로 필터링하려면 이 값을 추가합니다.

참고

AvailabilityZoneFilter Compute 스케줄러 서비스 필터를 사용하는 대신 배치 서비스를 사용하여 가용성 영역 요청을 처리할 수 있습니다. 자세한 내용은 **Placement** 서비스를 사용하여 가용성 영역으로 필터링을 참조하십시오.

3. 업데이트를 **Compute** 환경 파일에 저장합니다.

4. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

6.2. 호스트 집계 생성

클라우드 관리자는 필요한 만큼 많은 호스트 집계를 만들 수 있습니다.

절차

1. 호스트 집계를 생성하려면 다음 명령을 입력합니다.

```
(overcloud)# openstack aggregate create <aggregate_name>
```

<aggregate_name> 을 호스트 집계에 할당할 이름으로 바꿉니다.

2. 호스트 집계에 메타데이터를 추가합니다.

```
(overcloud)# openstack aggregate set \
 --property <key=value> \
 --property <key=value> \
 <aggregate_name>
```

- **<key=value>** 를 메타데이터 키-값 쌍으로 바꿉니다. **AggregateInstanceExtraSpecsFilter** 필터를 사용하는 경우 키는 임의의 문자열(예: **ssd=true**)일 수 있습니다. **AggregateImagePropertiesIsolation** 필터를 사용하는 경우 키는 유효한 이미지 메타데이터 속성과 일치해야 합니다. 유효한 이미지 메타데이터 속성에 대한 자세한 내용은 이미지 메타데이터 를 참조하십시오.

- **<aggregate_name>** 을 호스트 집계 이름으로 바꿉니다.

3. 호스트 집계에 컴퓨팅 노드를 추가합니다.

```
(overcloud)# openstack aggregate add host \
 <aggregate_name> \
 <host_name>
```

- **<aggregate_name>** 을 컴퓨팅 노드를 추가할 호스트 집계의 이름으로 바꿉니다.
- **<host_name>** 을 호스트 집계에 추가할 컴퓨팅 노드의 이름으로 바꿉니다.

4. 호스트 집계에 대한 플레이버 또는 이미지를 생성합니다.

- 플레이버를 생성합니다.

```
(overcloud)$ openstack flavor create \
  --ram <size_mb> \
  --disk <size_gb> \
  --vcpus <no_reserved_vcpus> \
  host-agg-flavor
```

- 이미지를 생성합니다.

```
(overcloud)$ openstack image create host-agg-image
```

5. 호스트 집계의 키-값 쌍과 일치하는 플레이버 또는 이미지에 하나 이상의 키-값 쌍을 설정합니다.

- 플레이버에서 키-값 쌍을 설정하려면 **aggregate_instance_extra_specs** 범위를 사용합니다.

```
(overcloud)# openstack flavor set \
 --property aggregate_instance_extra_specs:ssd=true \
 host-agg-flavor
```

- 이미지에 키-값 쌍을 설정하려면 유효한 이미지 메타데이터 속성을 키로 사용합니다.

```
(overcloud)# openstack image set \
 --property os_type=linux \
 host-agg-image
```

6.3. 가용성 영역 생성

클라우드 관리자는 클라우드 사용자가 인스턴스를 만들 때 선택할 수 있는 가용성 영역을 만들 수 있습니다.

절차

1. 가용성 영역을 생성하려면 새 가용성 영역 호스트 집계를 생성하거나 기존 호스트가 가용성 영역을 집계하도록 할 수 있습니다.

   a. 새 가용성 영역 호스트 집계를 생성하려면 다음 명령을 입력합니다.

   ```
   (overcloud)# openstack aggregate create \
    --zone <availability_zone> \
    <aggregate_name>
   ```

   - **<availability_zone>** 을 가용성 영역에 할당할 이름으로 바꿉니다.
   - **<aggregate_name>** 을 호스트 집계에 할당할 이름으로 바꿉니다.

   b. 기존 호스트가 가용성 영역을 집계하려면 다음 명령을 입력합니다.

   ```
   (overcloud)# openstack aggregate set --zone <availability_zone> \
    <aggregate_name>
   ```

   - **<availability_zone>** 을 가용성 영역에 할당할 이름으로 바꿉니다.
   - **<aggregate_name>** 을 호스트 집계 이름으로 바꿉니다.

2. 선택 사항: 가용성 영역에 메타데이터를 추가합니다.

```
(overcloud)# openstack aggregate set --property <key=value> \
 <aggregate_name>
```

- **<key=value>** 를 메타데이터 키-값 쌍으로 바꿉니다. 필요한 만큼 키-값 속성을 추가할 수 있습니다.
- **<aggregate_name>** 을 가용성 영역 호스트 집계의 이름으로 바꿉니다.

3. 가용성 영역 호스트 집계에 컴퓨팅 노드를 추가합니다.

```
(overcloud)# openstack aggregate add host <aggregate_name> \
 <host_name>
```

- **<aggregate_name>** 을 컴퓨팅 노드를 추가할 가용성 영역 호스트 집계의 이름으로 바꿉니다.
- **<host_name>** 을 가용성 영역에 추가할 컴퓨팅 노드 이름으로 바꿉니다.

6.4. 호스트 집계 삭제

호스트 집계를 삭제하려면 먼저 호스트 집계에서 모든 컴퓨팅 노드를 제거합니다.

절차

1. 호스트 집계에 할당된 모든 컴퓨팅 노드 목록을 보려면 다음 명령을 입력합니다.

```
(overcloud)# openstack aggregate show <aggregate_name>
```

2. 호스트 집계에서 할당된 모든 컴퓨팅 노드를 제거하려면 각 컴퓨팅 노드에 대해 다음 명령을 입력합니다.

```
(overcloud)# openstack aggregate remove host <aggregate_name> \
 <host_name>
```

- **<aggregate_name>** 을 **Compute** 노드를 제거할 호스트 집계의 이름으로 바꿉니다.

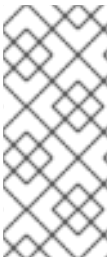
- **<host_name>** 을 호스트 집계에서 제거할 컴퓨팅 노드의 이름으로 바꿉니다.

3. 호스트 집계에서 모든 컴퓨팅 노드를 제거한 후 다음 명령을 입력하여 호스트 집계를 삭제합니다.

```
(overcloud)# openstack aggregate delete <aggregate_name>
```

6.5. 프로젝트 분리 호스트 집계 생성

특정 프로젝트에서만 사용할 수 있는 호스트 집계를 생성할 수 있습니다. 호스트 집계에 할당한 프로젝트만 호스트 집계에서 인스턴스를 시작할 수 있습니다.



참고

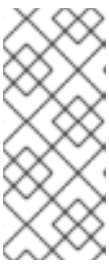
프로젝트 격리는 배치 서비스를 사용하여 각 프로젝트의 호스트 집계를 필터링합니다. 이 프로세스는 **AggregateMultiTenancyIsolation** 필터의 기능을 대체합니다. 따라서 **AggregateMultiTenancyIsolation** 필터를 사용할 필요가 없습니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 프로젝트 분리 호스트 집계에 프로젝트 인스턴스를 예약하려면 **Compute** 환경 파일에서 **NovaSchedulerLimitTenantsToPlacementAggregate** 매개 변수를 **True** 로 설정합니다.

3. 선택 사항: 호스트 집계에 할당한 프로젝트만 클라우드에 인스턴스를 생성할 수 있도록 하려면 **NovaSchedulerPlacementAggregateRequiredForTenants** 매개변수를 **True** 로 설정합니다.



참고

NovaSchedulerPlacementAggregateRequiredForTenants 는 기본적으로 **False** 입니다. 이 매개 변수가 **False** 인 경우 호스트 집계에 할당되지 않은 프로젝트는 모든 호스트 집계에 인스턴스를 생성할 수 있습니다.

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/<compute_environment_file>.yaml \
```

6. 호스트 집계를 생성합니다.

7. 프로젝트 **ID** 목록을 검색합니다.

```
(overcloud)# openstack project list
```

8. **filter_tenant_id<suffix>** 메타데이터 키를 사용하여 프로젝트를 호스트 집계에 할당합니다.

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id<ID0>=<project_id0> \
 --property filter_tenant_id<ID1>=<project_id1> \
 ...
 --property filter_tenant_id<IDn>=<project_idn> \
 <aggregate_name>
```

- **<ID0>, <ID1>** 및 **<IDn>** 까지 모든 **ID**를 생성하려는 각 프로젝트 필터의 고유 값으로 바꿉니다.
- **<project_id0>, <project_id1>** 및 **<project_idn>** 까지 모든 프로젝트 **ID**를 호스트 집계에 할당하려는 각 프로젝트의 **ID**로 바꿉니다.
- **<aggregate_name>** 을 **project-isolated** 호스트 집계의 이름으로 바꿉니다.

예를 들어 다음 구문을 사용하여 프로젝트 **78f1,9d3t** 및 **aa29** 를 호스트 집계 **project-isolated-aggregate** 에 할당합니다.

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id0=78f1 \
 --property filter_tenant_id1=9d3t \
 --property filter_tenant_id2=aa29 \
 project-isolated-aggregate
```

작은 정보

filter_tenant_id 메타데이터 키에서 접미사를 생략하여 단일 특정 프로젝트에서만 사용할 수 있는 호스트 집계를 생성할 수 있습니다.

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id=78f1 \
 single-project-isolated-aggregate
```

추가 리소스

- 호스트 집계 생성에 대한 자세한 내용은 호스트 집계 생성 및 관리를 참조하십시오.

7장. 인스턴스 스케줄링 및 배치 구성

계산 스케줄러 서비스는 인스턴스를 배치할 컴퓨팅 노드 또는 호스트 집계를 결정합니다. **Compute(nova)** 서비스에서 인스턴스를 시작하거나 이동하라는 요청을 수신하면 요청, 플레이버 및 이미지를 사용하여 적절한 호스트를 찾습니다. 예를 들어 플레이버는 인스턴스에 스토리지 디스크 유형 또는 **Intel CPU** 명령 집합 확장 등 호스트가 보유해야 하는 특성을 지정할 수 있습니다.

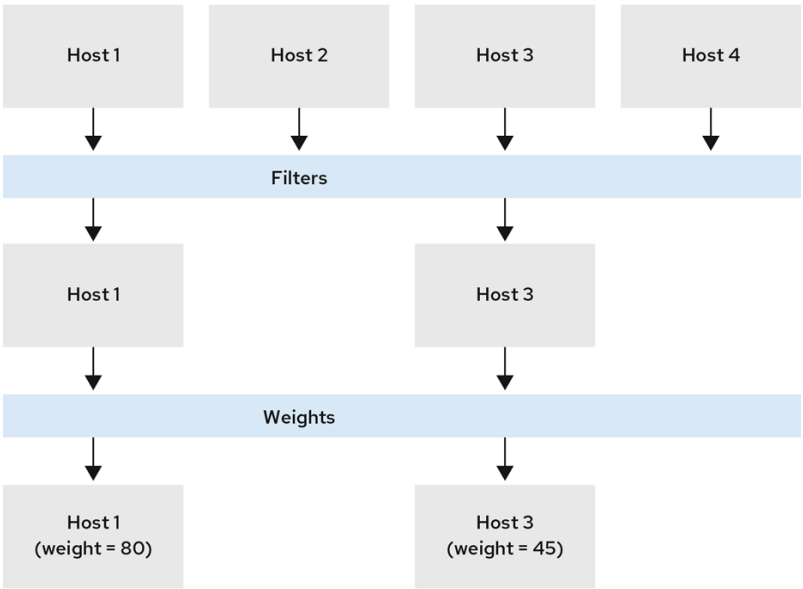
계산 스케줄러 서비스는 다음 구성 요소의 구성을 사용하여 인스턴스를 시작하거나 이동할 계산 노드를 결정합니다.

1. 배치 서비스 사전 필터 : 계산 스케줄러 서비스는 배치 서비스를 사용하여 특정 속성을 기반으로 후보 컴퓨팅 노드 집합을 필터링합니다. 예를 들어 배치 서비스는 비활성화된 **Compute** 노드를 자동으로 제외합니다.

2. 필터: 계산 스케줄러 서비스에서 사용하여 인스턴스를 시작할 계산 노드의 초기 집합을 결정합니다.

3. 가중치: 계산 스케줄러 서비스는 가중치 시스템을 사용하여 필터링된 컴퓨팅 노드의 우선 순위를 지정합니다. 가장 높은 가중치는 우선 순위가 가장 높습니다.

다음 다이어그램에서 호스트 **1** 및 **3**은 필터링 후 사용할 수 있습니다. 호스트 **1**은 가중치가 가장 높으므로 스케줄링에 가장 높은 우선 순위가 있습니다.



7.1. 배치 서비스를 사용한 사전 필터링

Compute 서비스(**nova**)는 인스턴스를 생성하고 관리할 때 배치 서비스와 상호 작용합니다. 배치 서비스는 계산 노드, 공유 스토리지 풀 또는 **IP** 할당 풀, 사용 가능한 **vCPU**와 같은 리소스 프로바이더의 인벤토리 및 사용을 추적합니다. 리소스의 선택 및 사용을 관리해야 하는 서비스는 배치 서비스를 사용할 수 있습니다.

배치 서비스는 리소스 프로바이더가 보유한 스토리지 디스크 특성 유형과 같은 리소스 프로바이더에 대한 사용 가능한 정성적 리소스 매핑도 추적합니다.

배치 서비스는 배치 서비스 리소스 프로바이더 인벤토리 및 특성을 기반으로 후보 컴퓨팅 노드 세트에 사전 필터를 적용합니다. 다음 기준에 따라 사전 필터를 생성할 수 있습니다.

- 지원되는 이미지 유형
- **traits**
- 프로젝트 또는 테넌트
- 가용성 영역

7.1.1. 요청된 이미지 유형 지원으로 필터링

인스턴스를 시작하는 데 사용되는 이미지의 디스크 형식을 지원하지 않는 컴퓨팅 노드를 제외할 수 있습니다. 이 기능은 환경에서 **Red Hat Ceph Storage**를 임시 백엔드로 사용하여 **QCOW2** 이미지를 지원하지 않는 경우에 유용합니다. 이 기능을 사용하면 스케줄러에서 **QCOW2** 이미지를 사용하여 **Red Hat Ceph Storage**에서 지원하는 계산 노드로 인스턴스를 시작하도록 요청을 보내지 않습니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.
2. 인스턴스를 시작하는 데 사용되는 이미지의 디스크 형식을 지원하지 않는 컴퓨팅 노드를 제

외하려면 **Compute** 환경 파일에서 **NovaSchedulerQueryImageType** 매개변수를 **True** 로 설정합니다.

3. 업데이트를 **Compute** 환경 파일에 저장합니다.

4. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

7.1.2. 리소스 공급자 특성별 필터링

각 리소스 프로바이더에는 일련의 특성이 있습니다. 특성은 리소스 프로바이더의 질적 측면입니다(예: 스토리지 디스크 유형 또는 **Intel CPU** 명령 집합 확장).

계산 노드는 해당 기능을 특성으로 배치 서비스에 보고합니다. 인스턴스는 이러한 특성 중 필요한 특성 또는 리소스 프로바이더에 없어야 하는 특성을 지정할 수 있습니다. 계산 스케줄러는 이러한 특성을 사용하여 인스턴스를 호스팅하는 데 적합한 컴퓨팅 노드 또는 호스트 집계를 식별할 수 있습니다.

클라우드 사용자가 특정 특성이 있는 호스트에 인스턴스를 만들 수 있도록 하려면 특정 특성을 요구하거나 금지하는 플레이버를 정의하고 특정 특성을 사용하거나 금지하는 이미지를 생성할 수 있습니다.

사용 가능한 특성 목록은 **os-traits** 라이브러리 를 참조하십시오. 필요에 따라 사용자 지정 특성을 만들 수도 있습니다.

7.1.2.1. 리소스 공급자 특성을 필요하거나 금지하는 이미지 생성

클라우드 사용자가 특정 특성을 가진 호스트에서 인스턴스를 시작하는 데 사용할 수 있는 인스턴스 이미지를 생성할 수 있습니다.

절차

1. 새 이미지를 생성합니다.

```
(overcloud)$ openstack image create ... trait-image
```

2. 호스트 또는 호스트 집계가 있어야 하는 특성을 식별합니다. 기존 특성을 선택하거나 새 특성을 생성할 수 있습니다.

- 기존 특성을 사용하려면 기존 특성을 나열하여 특성 이름을 검색합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait list
```

- 새 특성을 생성하려면 다음 명령을 입력합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
 create CUSTOM_TRAIT_NAME
```

사용자 지정 특성은 접두사 **CUSTOM_** 로 시작해야 하며 **A~Z** 문자, 숫자 **0~9**자, 밑줄 **"_"** 문자만 포함해야 합니다.

3. 각 호스트의 기존 리소스 공급자 특성을 수집합니다.

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

4. 다음과 같이 호스트 또는 호스트 집계가 필요한 특성의 기존 리소스 공급자 특성을 확인합니다.

```
(overcloud)$ echo $existing_traits
```

5. 필요한 특성이 리소스 공급자에 아직 추가되지 않은 경우 각 호스트의 리소스 공급자에 기존 특성 및 필요한 특성을 추가합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

<TRAIT_NAME> 을 리소스 공급자에 추가할 특성의 이름으로 바꿉니다. 필요에 따라 **--trait** 옵션을 두 번 이상 사용하여 추가 특성을 추가할 수 있습니다.

참고

이 명령은 리소스 공급자에 대한 특성을 완전히 대체합니다. 따라서 호스트의 기존 리소스 공급자 특성 목록을 검색하고 다시 설정하여 제거되지 않도록 해야 합니다.

6. 필수 특성을 가진 호스트 또는 호스트 집계에서 인스턴스를 예약하려면 이미지 추가 사양에 특성을 추가합니다. 예를 들어 플레이버**-512**를 지원하는 호스트 또는 호스트 집계에 인스턴스를 예약하려면 이미지 추가 사양에 다음 특성을 추가합니다.

```
(overcloud)$ openstack image set \
 --property trait:HW_CPU_X86_AVX512BW=required \
 trait-image
```

7. 금지된 특성이 있는 호스트 또는 호스트 집계를 필터링하려면 해당 특성을 이미지에 추가 사양에 추가합니다. 예를 들어 다중 연결 볼륨을 지원하는 호스트 또는 호스트 집계에 인스턴스가 예약되지 않도록 하려면 이미지 추가 사양에 다음 특성을 추가합니다.

```
(overcloud)$ openstack image set \
 --property trait:COMPUTE_VOLUME_MULTI_ATTACH=forbidden \
 trait-image
```

7.1.2.2. 리소스 공급자 특성을 필요하거나 금지하는 플레이버 생성

클라우드 사용자가 특정 특성을 가진 호스트에서 인스턴스를 시작하는 데 사용할 플레이버를 만들 수 있습니다.

절차

1. 플레이버를 생성합니다.

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 \
 --disk 2 trait-flavor
```

2. 호스트 또는 호스트 집계가 있어야 하는 특성을 식별합니다. 기존 특성을 선택하거나 새 특성을 생성할 수 있습니다.

- 기존 특성을 사용하려면 기존 특성을 나열하여 특성 이름을 검색합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait list
```

- 새 특성을 생성하려면 다음 명령을 입력합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
 create CUSTOM_TRAIT_NAME
```

사용자 지정 특성은 접두사 **CUSTOM_** 로 시작해야 하며 **A~Z** 문자, 숫자 **0~9**자, 밑줄 "_" 문자만 포함해야 합니다.

3. 각 호스트의 기존 리소스 공급자 특성을 수집합니다.

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

4. 다음과 같이 호스트 또는 호스트 집계가 필요한 특성의 기존 리소스 공급자 특성을 확인합니다.

```
(overcloud)$ echo $existing_traits
```

5. 필요한 특성이 리소스 공급자에 아직 추가되지 않은 경우 각 호스트의 리소스 공급자에 기존 특성 및 필요한 특성을 추가합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

<TRAIT_NAME> 을 리소스 공급자에 추가할 특성의 이름으로 바꿉니다. 필요에 따라 **--trait** 옵션을 두 번 이상 사용하여 추가 특성을 추가할 수 있습니다.

참고

이 명령은 리소스 공급자에 대한 특성을 완전히 대체합니다. 따라서 호스트의 기존 리소스 공급자 특성 목록을 검색하고 다시 설정하여 제거되지 않도록 해야 합니다.

6. 필수 특성을 가진 호스트 또는 호스트 집계에 인스턴스를 예약하려면 플레이버 추가 사양에 특성을 추가합니다. 예를 들어 플레이버 추가 사양에 다음 특성을 추가하는 호스트 또는 호스트 집계에서 인스턴스를 예약하려면 다음 특성을 추가합니다.

```
(overcloud)$ openstack flavor set \
 --property trait:HW_CPU_X86_AVX512BW=required \
 trait-flavor
```

7. 금지된 특성이 있는 호스트 또는 호스트 집계를 필터링하려면 플레이버 추가 사양에 특성을 추가합니다. 예를 들어 다중 연결 볼륨을 지원하는 호스트 또는 호스트 집계에 인스턴스를 예약하지 않으려면 다음 특성을 플레이버 추가 사양에 추가합니다.

```
(overcloud)$ openstack flavor set \
 --property trait:COMPUTE_VOLUME_MULTI_ATTACH=forbidden \
 trait-flavor
```

7.1.3. 호스트 집계를 격리하여 필터링

호스트 집계의 예약을 플레이버 및 이미지 특성이 호스트 집계의 메타데이터와 일치하는 인스턴스로만 제한할 수 있습니다. 플레이버 및 이미지 메타데이터의 조합에는 해당 호스트 집계의 계산 노드에 예약할 수 있는 모든 호스트 집계 특성이 있어야 합니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 플레이버 및 이미지 특성이 집계 메타데이터와 일치하는 인스턴스를 호스팅하도록 호스트 집계를 격리하려면 **Compute** 환경 파일에서 **NovaSchedulerEnableIsolatedAggregateFiltering** 매개변수를 **True** 로 설정합니다.

3. 업데이트를 **Compute** 환경 파일에 저장합니다.

4. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

5. 호스트 집계를 격리할 특성을 식별합니다. 기존 특성을 선택하거나 새 특성을 생성할 수 있습니다.

- 기존 특성을 사용하려면 기존 특성을 나열하여 특성 이름을 검색합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait list
```

- 새 특성을 생성하려면 다음 명령을 입력합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
 create CUSTOM_TRAIT_NAME
```

사용자 지정 특성은 접두사 **CUSTOM_** 로 시작해야 하며 **A~Z** 문자, 숫자 **0~9**자, 밑줄 "_" 문자만 포함해야 합니다.

6. 각 컴퓨팅 노드의 기존 리소스 공급자 특성을 수집합니다.

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

7. 에 대한 호스트 집계를 격리할 특성의 기존 리소스 공급자 특성을 확인합니다.

```
(overcloud)$ echo $existing_traits
```

8. 필요한 특성이 리소스 공급자에 아직 추가되지 않은 경우 호스트 집계의 각 컴퓨팅 노드의 리소스 공급자에 기존 특성 및 필요한 특성을 추가합니다.

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

<TRAIT_NAME> 을 리소스 공급자에 추가할 특성의 이름으로 바꿉니다. 필요에 따라 **--trait** 옵션을 두 번 이상 사용하여 추가 특성을 추가할 수 있습니다.

참고

이 명령은 리소스 공급자에 대한 특성을 완전히 대체합니다. 따라서 호스트의 기존 리소스 공급자 특성 목록을 검색하고 다시 설정하여 제거되지 않도록 해야 합니다.

9. 호스트 집계의 각 컴퓨팅 노드에 대해 **6~8**단계를 반복합니다.

10. 특성의 **metadata** 속성을 호스트 집계에 추가합니다.

```
(overcloud)$ openstack --os-compute-api-version 2.53 aggregate set \
 --property trait:<TRAIT_NAME>=required <aggregate_name>
```

11. 플레이버 또는 이미지에 특성을 추가합니다.

```
(overcloud)$ openstack flavor set \
 --property trait:<TRAIT_NAME>=required <flavor>
(overcloud)$ openstack image set \
 --property trait:<TRAIT_NAME>=required <image>
```

7.1.4. 배치 서비스를 사용하여 가용성 영역으로 필터링

배치 서비스를 사용하여 가용성 영역 요청을 이행할 수 있습니다. 배치 서비스를 사용하여 가용성 영역별로 필터링하려면 가용성 영역 호스트 집계의 멤버십 및 **UUID**와 일치하는 배치 집계가 있어야 합니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 배치 서비스를 사용하여 가용성 영역별로 필터링하려면 **Compute** 환경 파일에서 **NovaSchedulerQueryPlacementForAvailabilityZone** 매개 변수를 **True** 로 설정합니다.

3. **NovaSchedulerDefaultFilters** 매개변수에서 **AvailabilityZoneFilter** 필터를 제거합니다.

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

추가 리소스

- 가용성 영역으로 사용할 호스트 집계를 만드는 방법에 대한 자세한 내용은 가용성 영역 생성을 참조하십시오.

7.2. COMPUTE 스케줄러 서비스의 필터 및 가중치 구성

인스턴스를 시작할 컴퓨팅 노드의 초기 집합을 결정하려면 **Compute** 스케줄러 서비스의 필터 및 가중치를 구성해야 합니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 스케줄러가 **NovaSchedulerDefaultFilters** 매개변수에 사용할 필터를 추가합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
AggregateInstanceExtraSpecsFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter
```

3. 각 컴퓨팅 노드의 가중치를 계산하는 데 사용할 속성을 지정합니다. 예를 들면 다음과 같습니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      filter_scheduler/weight_classes:
        value: nova.scheduler.weights.all_weighers
```

사용 가능한 속성에 대한 자세한 내용은 **Compute** 스케줄러 가중치를 참조하십시오.

4. 선택 사항: 각 가중치에 맞게 **multiplier**를 구성합니다. 예를 들어 **Compute** 노드의 사용 가능한 **RAM**에 다른 기본 가중치보다 높은 가중치가 있고 **Compute** 스케줄러에서 사용 가능한 **RAM**보다 더 많은 컴퓨팅 노드를 선호하는 경우 다음 구성을 사용합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      filter_scheduler/weight_classes:
```

```
      value: nova.scheduler.weights.all_weighers
    filter_scheduler/ram_weight_multiplier:
      value: 2.0
```

작은 정보

multipliers를 음수 값으로 설정할 수도 있습니다. 위의 예에서 사용 가능한 **RAM**보다 사용 가능한 **RAM**보다 적은 컴퓨팅 노드를 선호하려면 **ram_weight_multiplier** 를 **-2.0** 으로 설정합니다.

5. 업데이트를 **Compute** 환경 파일에 저장합니다.

6. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

추가 리소스

- 사용 가능한 **Compute** 스케줄러 서비스 필터 목록은 **Compute** 스케줄러 필터를 참조하십시오.
- 사용 가능한 가중치 구성 옵션 목록은 **Compute** 스케줄러 가중치를 참조하십시오.

7.3. 컴퓨팅 스케줄러 필터

Compute 환경 파일에 **NovaSchedulerDefaultFilters** 매개변수를 구성하여 **Compute** 스케줄러에서 인스턴스를 호스팅할 적절한 컴퓨팅 노드를 선택할 때 적용해야 하는 필터를 지정합니다. 기본 구성은 다음 필터에 적용됩니다.

- **AvailabilityZoneFilter:** 컴퓨팅 노드는 요청된 가용성 영역에 있어야 합니다.
- **ComputeFilter:** 컴퓨팅 노드는 요청을 서비스할 수 있습니다.
- **ComputeCapabilitiesFilter:** 컴퓨팅 노드는 플레이버 추가 사양을 충족합니다.

- **ImagePropertiesFilter:** 컴퓨팅 노드는 요청된 이미지 속성을 충족합니다.
- **ServerGroupAntiAffinityFilter:** 컴퓨팅 노드는 지정된 그룹의 인스턴스를 이미 호스팅하지 않습니다.
- **ServerGroupAffinityFilter:** 컴퓨팅 노드는 이미 지정된 그룹의 인스턴스를 호스팅하고 있습니다.

필터를 추가하고 제거할 수 있습니다. 다음 테이블에서는 사용 가능한 모든 필터를 설명합니다.

표 7.1. 컴퓨팅 스케줄러 필터

필터	설명
AggregateImagePropertiesIsolation	이 필터를 사용하여 인스턴스의 이미지 메타데이터와 호스트 집계 메타데이터를 일치시킵니다. 호스트 집계 메타데이터 중 하나라도 이미지의 메타데이터와 일치하는 경우 해당 호스트 집계에 속하는 컴퓨팅 노드가 해당 이미지에서 인스턴스를 시작하기 위한 후보입니다. 스케줄러는 유효한 이미지 메타데이터 속성만 고려합니다. 유효한 이미지 메타데이터 속성에 대한 자세한 내용은 이미지 메타데이터 속성 을 참조하십시오.
AggregateInstanceExtraSpecsFilter	이 필터를 사용하여 호스트 집계 메타데이터와 인스턴스의 플레이버 추가 사양에 정의된 네임스페이스의 속성을 일치시킵니다. **aggregate_instance_extra _specs: 네임스페이스를 접두사로 지정하여 extra_specs** 키의 범위를 지정해야 합니다. 호스트 집계 메타데이터 중 하나라도 플레이버 추가 사양의 메타데이터와 일치하는 경우 해당 호스트 집계에 속하는 컴퓨팅 노드가 해당 이미지에서 인스턴스를 시작하기 위한 후보입니다.
AggregateIoOpsFilter	이 필터를 사용하여 I/O 작업별로 **filter_scheduler/max_io_ops_per_host** 값을 사용하여 호스트를 필터링합니다. 집계별 값을 찾을 수 없는 경우 이 값은 전역 설정으로 대체됩니다. 호스트가 두 개 이상의 집계에 있고 둘 이상의 값이 발견되면 스케줄러는 최솟값을 사용합니다.

필터	설명
AggregateMultiTenancyIsolation	이 필터를 사용하여 project-isolated 호스트 집계의 컴퓨팅 노드 가용성을 지정된 프로젝트 집합으로 제한합니다. **filter_tenant_id** 메타데이터 키를 사용하여 지정된 프로젝트만 호스트 집계의 컴퓨팅 노드에서 인스턴스를 시작할 수 있습니다. 자세한 내용은 Creating a project-isolated host aggregate를 참조하십시오.  **참고** 프로젝트는 계속 다른 호스트에 인스턴스를 배치할 수 있습니다. 이를 제한하려면 **NovaSchedulerPlacementAggregateRequiredForTenants** 매개변수를 사용합니다.
AggregateNumInstancesFilter	이 필터를 사용하여 집계된 각 컴퓨팅 노드의 인스턴스 수를 제한할 수 있습니다. **filter_scheduler/max_instances_per_host** 매개변수를 사용하여 집계당 최대 인스턴스 수를 구성할 수 있습니다. 집계별 값을 찾을 수 없는 경우 이 값은 전역 설정으로 대체됩니다. 컴퓨팅 노드가 두 개 이상의 집계에 있는 경우 스케줄러는 가장 낮은 **max_instances_per_host** 값을 사용합니다.
AggregateTypeAffinityFilter	플레이버 메타데이터 키가 설정되지 않았거나 플레이버 집계 메타데이터 값에 요청된 플레이버의 이름이 포함된 경우 이 필터를 사용하여 호스트를 전달합니다. 플레이버 메타데이터 항목의 값은 m1.nano 또는 **m1.nano,m1.small** 과 같이 단일 플레이버 이름 또는 쉼표로 구분된 플레이버 이름 목록을 포함할 수 있는 문자열입니다.
AllHostsFilter	이 필터를 사용하여 인스턴스 예약에 사용 가능한 모든 컴퓨팅 노드를 고려하십시오.  **참고** 이 필터를 사용하면 다른 필터가 비활성화되지 않습니다.
AvailabilityZoneFilter	이 필터를 사용하여 인스턴스에서 지정한 가용성 영역의 컴퓨팅 노드에서 인스턴스를 시작합니다.
ComputeCapabilitiesFilter	이 필터를 사용하여 계산 노드 기능과 비교하여 플레이버의 추가 사양에 정의된 네임스페이스의 속성을 일치시킵니다. 플레이버 추가 사양 앞에 **capabilities:** namespace를 추가해야 합니다. **ComputeCapabilitiesFilter** 필터를 사용하는 보다 효율적인 대안은 배치 서비스에 보고되는 플레이버의 CPU 특성을 사용하는 것입니다. 특성은 CPU 기능에 대한 일관된 명명을 제공합니다. 자세한 내용은 리소스 공급자 특성을 사용하여 필터링을 참조하십시오.
ComputeFilter	이 필터를 사용하여 작동 및 활성화된 모든 컴퓨팅 노드를 전달합니다. 이 필터가 항상 존재해야 합니다.

필터	설명

DifferentHostFilter	이 필터를 사용하여 특정 인스턴스 집합의 다른 컴퓨팅 노드에서 인스턴스 예약을 활성화합니다. 인스턴스를 시작할 때 이러한 인스턴스를 지정하려면 **different_host** 를 키로 사용하여 **--hint** 인수를 사용하고 인스턴스 UUID를 값으로 사용합니다.
	
	``$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \
	  --flavor 1 --hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \
	  --hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1```

ImagePropertiesFilter	이 필터를 사용하여 인스턴스 이미지에 정의된 다음 속성을 기반으로 컴퓨팅 노드를 필터링합니다. - **hw_architecture** - 호스트 아키텍처(예: x86, ARM, Power)에 대한 응답. - **img_hv_type** - 하이퍼바이저 유형(예: KVM, QEMU, Xen, LXC)에 대한 응답. - **img_hv_requested_version** - Compute 서비스가 보고하는 하이퍼바이저 버전에 대한 응답. - **hw_vm_mode** - 하이퍼바이저 유형(예: hvm, xen, satellitel 또는 exe)에 대한 응답. 인스턴스에 포함된 지정된 이미지 속성을 지원할 수 있는 컴퓨팅 노드는 스케줄러로 전달됩니다. 이미지 속성에 대한 자세한 내용은 이미지 메타데이터 속성 을 참조하십시오.

IsolatedHostsFilter	이 필터를 사용하여 격리된 컴퓨팅 노드에 격리된 이미지가 있는 인스턴스만 예약합니다. **filter_scheduler/restrict_isolated_hosts_to_isolated_images** 를 구성하여 격리된 컴퓨팅 노드에 인스턴스를 빌드하는 데 격리되지 않은 이미지를 사용하지 않도록 할 수도 있습니다.
	
	분리된 이미지 및 호스트 세트를 지정하려면 **filter_scheduler/isolated_hosts** 및 **filter_scheduler/isolated_images** 구성 옵션을 사용합니다. 예를 들면 다음과 같습니다.
	
	``parameter_defaults:
	  ComputeExtraConfig:
	  nova::config::nova_config:
	  filter_scheduler/isolated_hosts:
	  value: server1, server2
	  filter_scheduler/isolated_images:
	  value: 342b492c-128f-4a42-8d3a-c5088cf27d13, ebd267a6-ca86-4d6c-9a0e-bd132d6b7d09```

필터	설명
IoOpsFilter	이 필터를 사용하여 호스트에서 실행할 수 있는 최대 I/O 집약 인스턴스 수를 지정하는 구성된 **filter_scheduler/max_io_ops_per_host** 를 초과하는 동시 I/O 작업이 있는 호스트를 필터링합니다.

MetricsFilter	이 필터를 사용하여 metrics **/weight_setting** 을 사용하여 구성한 지표를 보고하는 컴퓨팅 노드로 예약을 제한합니다.
	
	이 필터를 사용하려면 Compute 환경 파일에 다음 구성을 추가합니다.
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  DEFAULT/compute_monitors:`
	`  value: 'cpu.virt_driver'`
	
	기본적으로 계산 스케줄러 서비스는 60초마다 지표를 업데이트합니다. 메트릭이 최신 상태인지 확인하려면 **update_resources_interval** 구성 옵션을 사용하여 지표 데이터가 새로 고침되는 빈도를 늘릴 수 있습니다. 예를 들어 다음 구성을 사용하여 2초마다 지표 데이터를 새로 고칩니다.
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  DEFAULT/update_resources_interval:`
	`  value: '2'`

NUMATopologyFilter	이 필터를 사용하여 NUMA 사용 가능한 Compute 노드에서 NUMA 토폴로지가 있는 인스턴스를 예약합니다. **extra_specs** 플레이버 및 이미지 속성을 사용하여 인스턴스의 NUMA 토폴로지를 지정합니다. 필터는 각 호스트 NUMA 셀에 대한 초과 서브스크립션 제한을 고려하여 Compute 노드 토폴로지에 인스턴스 NUMA 토폴로지를 일치시키려고 합니다.
NumInstancesFilter	이 필터를 사용하여 **max_instances_per_host** 옵션에서 지정한 것보다 많은 인스턴스가 실행 중인 컴퓨팅 노드를 필터링합니다.
PciPassthroughFilter	이 필터를 사용하여 **extra_specs** 플레이버를 사용하여 인스턴스가 요청하는 장치가 있는 컴퓨팅 노드에 인스턴스를 예약합니다. 요청하는 인스턴스에 대해 일반적으로 비용이 많이 들고 제한된 PCI 장치가 있는 노드를 예약하려면 이 필터를 사용합니다.

필터	설명

SameHostFilter	이 필터를 사용하여 특정 인스턴스 집합과 동일한 컴퓨팅 노드에서 인스턴스 예약을 활성화합니다. 인스턴스를 시작할 때 이러한 인스턴스를 지정하려면 키로 **same_host** 를 사용하고 인스턴스 UUID를 값으로 **--hint** 인수를 사용합니다.
	`$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \`
	`--flavor 1 --hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \`
	`--hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1`

ServerGroupAffinityFilter	이 필터를 사용하여 동일한 컴퓨팅 노드의 선호도 서버 그룹에 인스턴스를 예약합니다. 서버 그룹을 생성하려면 다음 명령을 입력합니다.
	`$ openstack server group create --policy affinity <group_name>`
	이 그룹에서 인스턴스를 시작하려면 그룹에 **--hint** 인수를 **키로** 사용하고 그룹 UUID를 값으로 사용합니다.
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint group=<group_uuid> <instance_name>`

ServerGroupAntiAffinityFilter	이 필터를 사용하여 다른 컴퓨팅 노드에서 anti-affinity 서버 그룹에 속하는 인스턴스를 예약합니다. 서버 그룹을 생성하려면 다음 명령을 입력합니다.
	`$ openstack server group create --policy anti-affinity <group_name>`
	이 그룹에서 인스턴스를 시작하려면 그룹에 **--hint** 인수를 **키로** 사용하고 그룹 UUID를 값으로 사용합니다.
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint group=<group_uuid> <instance_name>`

SimpleCIDRAffinityFilter	이 필터를 사용하여 특정 IP 서브넷 범위가 있는 컴퓨팅 노드에 인스턴스를 예약합니다. 필요한 범위를 지정하려면 인스턴스를 시작할 때 **--hint** 인수를 사용하여 **build_near_host_ip** 키와 **cidr** 키를 전달합니다.
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint build_near_host_ip=<ip_address> \`
	`--hint cidr=<subnet_mask> <instance_name>`

7.4. 컴퓨팅 스케줄러 가중치

각 컴퓨팅 노드에는 스케줄러에서 인스턴스 예약 우선 순위를 지정하는 데 사용할 수 있는 가중치가 있

습니다. **Compute** 스케줄러가 필터를 적용한 후 나머지 후보 컴퓨팅 노드에서 가장 큰 가중치가 있는 **Compute** 노드를 선택합니다.

Compute 스케줄러는 다음 작업을 수행하여 각 컴퓨팅 노드의 가중치를 결정합니다.

1. 스케줄러는 각 가중치를 **0.0**에서 **1.0** 사이의 값으로 정규화합니다.

2. 스케줄러는 **weighter multiplier**로 정규화된 가중치를 곱합니다.

Compute 스케줄러는 후보 컴퓨팅 노드에서 리소스 가용성에 대한 하위 및 상위 값을 사용하여 각 리소스 유형의 가중치 정규화를 계산합니다.

- 리소스 가용성이 가장 낮은 노드에 **'0'**이 할당됩니다.

- 리소스 가용성**(maxval)**이 가장 높은 노드에는 **'1'**이 할당됩니다.

- **minval - maxval** 범위 내에 리소스 가용성이 있는 노드에는 다음 수식을 사용하여 계산된 정규화된 가중치가 할당됩니다.

  ```
  (node_resource_availability - minval) / (maxval - minval)
  ```

모든 컴퓨팅 노드에 리소스에 대한 가용성이 동일한 경우 모두 **0**으로 정규화됩니다.

예를 들어 스케줄러는 다음과 같이 각각 사용 가능한 **vCPU** 수 **10**개에서 사용 가능한 **vCPU**에 대한 정규화된 가중치를 계산합니다.

컴퓨팅 노드	1	2	3	4	5	6	7	8	9	10
vCPU 없음	5	5	10	10	15	20	20	15	10	5
정규화된 가중치	0	0	0.33	0.33	0.67	1	1	0.67	0.33	0

컴퓨팅 스케줄러는 다음 공식을 사용하여 컴퓨팅 노드의 가중치를 계산합니다.

```
(w1_multiplier * norm(w1)) + (w2_multiplier * norm(w2)) + ...
```

다음 표에서는 가중치에 사용할 수 있는 구성 옵션을 설명합니다.

참고

다음 표에 설명된 옵션과 동일한 이름의 집계 메타데이터 키를 사용하여 호스트 집계에 가중치를 설정할 수 있습니다. 호스트 집계에 설정된 경우 호스트 집계 값이 우선합니다.

표 7.2. 컴퓨팅 스케줄러 가중치

설정 옵션	유형	설명
filter_scheduler/weight_classes	문자열	이 매개변수를 사용하여 각 컴퓨팅 노드의 가중치를 계산하는 데 사용할 속성을 구성합니다. ● **nova.scheduler.weights.ram.RAMWeigher** - 컴퓨팅 노드에서 사용 가능한 RAM을 조정합니다. ● **nova.scheduler.weights.cpu.CPUWeigher** - 컴퓨팅 노드에서 사용 가능한 CPU를 대기열에 넣습니다. ● **nova.scheduler.weights.disk.DiskWeigher** - 계산 노드에서 사용 가능한 디스크를 포함합니다. ● **nova.scheduler.weights.metrics.MetricsWeigher** - 컴퓨팅 노드의 메트릭을 결합합니다. ● **nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher** - 계산 노드의 근접성을 지정된 인스턴스 그룹의 다른 노드에 연결합니다. ● **nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher** - 계산 노드의 근접성을 지정된 인스턴스 그룹의 다른 노드에 배치합니다. ● **nova.scheduler.weights.compute.BuildFailureWeigher** - 컴퓨팅 노드를 최근 실패한 부팅 시도 횟수로 조정합니다. ● **nova.scheduler.weights.io_ops.IoOpsWeigher** - 계산 노드를 워크로드에 따라 조정합니다. ● **nova.scheduler.weights.pci.PCIWeigher** - 계산 노드를 PCI 가용성에 따라 조정합니다. ● **nova.scheduler.weights.cross_cell.CrossCellWeigher** - 인스턴스를 이동할 때 소스 셀에서 컴퓨팅 노드를 선호하는 셀을 기반으로 계산 노드를 조정합니다. ● **nova.scheduler.weights.all_weighers** - (기본값) 위의 모든 가중치를 사용합니다.

설정 옵션	유형	설명
filter_scheduler/ram_weight_multiplier	부동 소수점	이 매개 변수를 사용하여 사용 가능한 RAM에 따라 호스트를 가중치를 지정하는 데 사용할 승수를 지정합니다. 여러 호스트에 인스턴스를 분산하는 사용 가능한 RAM이 많은 호스트를 선호하도록 양수로 설정합니다. 사용 가능한 RAM이 적은 호스트를 선호하는 음수 값으로 설정합니다. 이 값은 덜 사용되는 호스트로 예약하기 전에 가능한 한 많은 호스트를 채웁니다(스택). 절대 값은 양수이든 음수든 관계없이 RAM 가중치가 다른 계량기와 비교하여 얼마나 강력한지 제어합니다. 기본값: 1.0 - 스케줄러는 모든 호스트에 인스턴스를 균등하게 분배합니다.
filter_scheduler/disk_weight_multiplier	부동 소수점	이 매개 변수를 사용하여 사용 가능한 디스크 공간을 기준으로 호스트를 가중치를 지정하는 데 사용할 승수를 지정합니다. 여러 호스트에 인스턴스를 분산하는 사용 가능한 디스크 공간이 많은 호스트를 선호하는 양수 값으로 설정합니다. 사용 가능한 디스크 공간이 적은 호스트를 선호하는 음수 값으로 설정합니다. 이 값은 덜 사용되는 호스트로 예약하기 전에 가능한 한 많은 호스트를 채웁니다(스택). 절대 값은, 양수이든 음수이든, 디스크 가중의 강점이 다른 계량기와 상대적인지를 제어합니다. 기본값: 1.0 - 스케줄러는 모든 호스트에 인스턴스를 균등하게 분배합니다.
filter_scheduler/cpu_weight_multiplier	부동 소수점	이 매개 변수를 사용하여 사용 가능한 vCPU를 기반으로 호스트를 가중치를 지정하는 데 사용할 승수를 지정합니다. 여러 호스트에 인스턴스를 분산하는 사용 가능한 vCPU가 더 많은 호스트를 선호하도록 양수로 설정합니다. 사용 가능한 vCPU가 적은 호스트를 선호하는 음수 값으로 설정합니다. 이 값은 덜 사용되는 호스트로 예약하기 전에 가능한 한 많은 호스트를 채웁니다(스택). 절대 값은 vCPU 가벼움이 다른 가중과 비교되는 정도를 제어합니다. 기본값: 1.0 - 스케줄러는 모든 호스트에 인스턴스를 균등하게 분배합니다.

설정 옵션	유형	설명
filter_scheduler/io_ops_weight_multiplier	부동 소수점	이 매개 변수를 사용하여 호스트 워크로드를 기준으로 호스트를 가중치를 조정하는 데 사용할 승수를 지정합니다. 더 적은 워크로드가 있는 호스트를 선호하려면 음수로 설정하여 더 많은 호스트에 워크로드를 분산합니다. 대규모 워크로드가 있는 호스트를 선호하도록 양수로 설정합니다. 이 값은 이미 사용 중인 호스트에 인스턴스를 예약합니다. 절대 값은 I/O 작업 가중치가 다른 가중치와 관련하여 얼마나 강력한지 제어합니다. 기본값: -1.0 - 스케줄러는 더 많은 호스트에 워크로드를 분산합니다.
filter_scheduler/build_failure_weight_multiplier	부동 소수점	이 매개변수를 사용하여 최근 빌드 실패에 따라 호스트를 가중치를 조정하는 데 사용할 승수를 지정합니다. 최근에 호스트에서 보고한 빌드 실패의 중요도를 높이기 위해 양수 값으로 설정합니다. 그러면 최근 빌드 오류가 있는 호스트를 선택할 가능성이 줄어듭니다. 최근 실패 횟수에 따라 계산 호스트의 가중치를 비활성화하려면 **0** 으로 설정합니다. 기본값: 1000000.0
filter_scheduler/cross_cell_move_weight_multiplier	부동 소수점	이 매개 변수를 사용하여 교차 셀 이동 중에 호스트를 가중치를 지정하는 데 사용할 승수를 지정합니다. 이 옵션은 인스턴스를 이동할 때 동일한 소스 셀 내에 있는 호스트에 배치되는 가중치 양을 결정합니다. 기본적으로 스케줄러는 인스턴스를 마이그레이션할 때 동일한 소스 셀 내의 호스트를 선호합니다. 인스턴스가 현재 실행 중인 동일한 셀 내의 호스트를 선호하도록 양수 값으로 설정합니다. 인스턴스가 현재 실행 중인 다른 셀에 있는 호스트를 선호하도록 음수 값으로 설정합니다. 기본값: 1000000.0

설정 옵션	유형	설명
filter_scheduler/pci_weight_multiplier	양수 부동점	이 매개 변수를 사용하여 호스트의 PCI 장치 수 및 인스턴스에서 요청한 PCI 장치 수에 따라 호스트를 가중치하는 데 사용할 승수를 지정합니다. 인스턴스가 PCI 장치를 요청하면 Compute 노드가 계산 노드에 할당된 가중치가 더 높은 PCI 장치가 증가합니다. 예를 들어, 여러 PCI 장치가 있고 PCI 장치가 없고 PCI 장치가 없는 단일 PCI 장치가 있는 호스트 3개가 사용 가능한 경우 Compute 스케줄러는 인스턴스의 요구에 따라 이러한 호스트에 우선 순위를 지정합니다. 인스턴스에서 PCI 장치를 하나씩 요청하는 경우 스케줄러에서 첫 번째 호스트를 선호해야 하며, 인스턴스에 여러 PCI 장치가 필요한 경우 두 번째 호스트와 인스턴스에서 PCI 장치를 요청하지 않는 경우 세 번째 호스트를 선호해야 합니다. PCI가 아닌 인스턴스가 PCI 장치가 있는 호스트의 리소스를 차지하지 못하도록 이 옵션을 구성합니다. 기본값: 1.0
filter_scheduler/host_subset_size	정수	이 매개 변수를 사용하여 호스트를 선택할 필터링된 호스트의 하위 집합 크기를 지정합니다. 이 옵션을 1 이상으로 설정해야 합니다. 값 1은 가중치 함수에서 반환한 첫 번째 호스트를 선택합니다. 스케줄러는 1보다 작은 값을 무시하고 대신 1을 사용합니다. 여러 스케줄러 프로세스가 동일한 호스트를 선택한 유사한 요청을 처리하여 잠재적인 경합 조건을 생성하지 않도록 하려면 1보다 큰 값으로 설정합니다. 요청에 가장 적합한 N 호스트에서 임의로 호스트를 선택하면 충돌 가능성이 줄어듭니다. 그러나 이 값을 설정하면 선택한 호스트가 해당 요청에 덜 적합할 수 있습니다. 기본값: 1
filter_scheduler/soft_affinity_weight_multiplier	양수 부동점	이 매개 변수를 사용하여 소프트 선호도 그룹에 대한 호스트의 가중치를 지정하는 데 사용할 승수를 지정합니다.  **참고** 이 정책으로 그룹을 생성할 때 마이크로버전을 지정해야 합니다. `$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name>` 기본값: 1.0

설정 옵션	유형	설명
filter_scheduler/soft_anti_affinity_weight_multiplier	양수 부동점	이 매개 변수를 사용하여 soft-anti-affinity 그룹에 대한 호스트를 조정하는 데 사용할 승수를 지정합니다.  **참고** 이 정책으로 그룹을 생성할 때 마이크로버전을 지정해야 합니다. `$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name>` 기본값: 1.0
metrics/weight_multiplier	부동 소수점	가중치 메트릭에 사용할 승수를 지정하려면 이 매개변수를 사용합니다. 기본적으로 가능한 호스트에 인스턴스를 분배하는 **weight_multiplier=1.0** 입니다. 전체 가중치에 대한 지표의 효과를 높이기 위해 1.0보다 큰 숫자로 설정합니다. 전체 가중치에 대한 지표의 영향을 줄이기 위해 0.0에서 1.0 사이의 숫자로 설정합니다. 지표 값을 무시하고 **weight_of_unavailable** 옵션의 값을 반환하려면 0.0으로 설정합니다. 더 낮은 지표로 호스트의 우선 순위를 지정하고 호스트의 스택 인스턴스를 사용하려면 음수로 설정합니다. 기본값: 1.0

설정 옵션	유형	설명
metrics/weight_setting	콤마로 구분된 **metric=ratio** 쌍 목록	이 매개변수를 사용하여 가중치에 사용할 메트릭과 각 메트릭의 가중치를 계산하는 데 사용할 비율을 지정합니다. 유효한 메트릭 이름: ● **cpu.frequency** - CPU 빈도 ● **cpu.user.time** - CPU 사용자 모드 시간 ● **cpu.kernel.time** - CPU 커널 시간 ● **cpu.idle.time** - CPU 유휴 시간 ● **cpu.iowait.time** - CPU I/O 대기 시간 ● **cpu.user.percent** - CPU 사용자 모드 백분율 ● **cpu.kernel.percent** - CPU 커널 백분율 ● **cpu.idle.percent** - CPU 유휴 백분율 ● **cpu.iowait.percent** - CPU I/O 대기 백분율 ● **cpu.percent** - 일반 CPU 사용 예: **weight_setting=cpu.user.time=1.0**
메트릭/필수	부울	이 매개변수를 사용하여 사용할 수 없는 구성된 **메트릭/값 설정 메트릭을** 처리하는 방법을 지정합니다. ● **True** - 메트릭이 필요합니다. 지표를 사용할 수 없는 경우 예외가 발생합니다. 예외를 방지하려면 **NovaSchedulerDefault Filters에서 Metrics Filter** 필터를 사용합니다. ● **False** - 사용할 수 없는 메트릭이 가중치 프로세스에서 부정 요인으로 처리됩니다. **weight_of_unavailable** 구성 옵션을 사용하여 반환된 값을 설정합니다.
metrics/weight_of_unavailable	부동 소수점	이 매개변수를 사용하여 **metrics/weight_setting 메트릭**을 사용할 수 없는 경우 사용할 가중치와 **metrics/required=False** 를 지정합니다. 기본값: -10000.0

8장. 인스턴스 시작을 위한 플레이버 만들기

인스턴스 플레이버는 인스턴스의 가상 하드웨어 프로필을 지정하는 리소스 템플릿입니다. 클라우드 사용자는 인스턴스를 시작할 때 플레이버를 지정해야 합니다.

플레이버는 계산 서비스에서 인스턴스에 할당해야 하는 다음 리소스의 수량을 지정할 수 있습니다.

- **vCPU** 수.
- **RAM(MB).**
- 루트 디스크**(GB).**
- 보조 임시 스토리지 및 스왑 디스크를 포함한 가상 스토리지.

모든 프로젝트에 플레이버를 공용으로 만들거나 특정 프로젝트 또는 도메인에 대한 개인을 만들어 플레이버를 사용할 수 있는 사람을 지정할 수 있습니다.

플레이버는 **"extra specs"**라고도 하는 메타데이터를 사용하여 인스턴스 하드웨어 지원 및 할당량을 지정할 수 있습니다. 플레이버 메타데이터는 인스턴스 배치, 리소스 사용 제한 및 성능에 영향을 미칩니다. 사용 가능한 메타데이터 속성의 전체 목록은 플레이버 메타데이터를 참조하십시오.

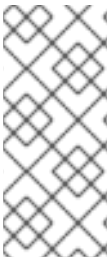
호스트 집계의 **extra_specs** 메타데이터와 일치하여 플레이버 메타데이터 키를 사용하여 인스턴스를 호스팅할 적절한 호스트 집계를 찾을 수도 있습니다. 호스트 집계에서 인스턴스를 예약하려면 **extra_specs** 키 앞에 **aggregate_ instance_extra_specs:** 네임스페이스를 추가하여 플레이버 메타데이터의 범위를 지정해야 합니다. 자세한 내용은 호스트 집계 생성 및 관리를 참조하십시오.

RHOSP(Red Hat OpenStack Platform) 배포에는 클라우드 사용자가 사용할 수 있는 다음과 같은 기본 공용 플레이버 세트가 포함되어 있습니다.

표 **8.1.** 기본 플레이버

이름	vCPU	RAM	루트 디스크 크기
m1.nano	1	128MB	1GB

이름	vCPU	RAM	루트 디스크 크기
m1.micro	1	192MB	1GB



참고

플레이버 속성을 사용하여 설정된 동작은 이미지를 사용하여 설정된 동작을 재정의합니다. 클라우드 사용자가 인스턴스를 시작하면 지정한 플레이버의 속성이 지정한 이미지의 속성을 재정의합니다.

8.1. 플레이버 생성

다음과 같이 특정 기능이나 동작을 위한 특수 플레이버를 생성하고 관리할 수 있습니다.

- 기본 메모리 및 용량을 변경하여 기본 하드웨어 요구 사항에 맞춥니다.
- 메타데이터를 추가하여 인스턴스에 특정 I/O 속도를 강제 적용하거나 호스트 집계와 일치시킵니다.

절차

1. 인스턴스에 사용할 수 있도록 기본 리소스를 지정하는 플레이버를 생성합니다.

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_vcpus> \
 [--private --project <project_id>] <flavor_name>
```

- **<size_mb>** 를 이 플레이버로 만든 인스턴스에 할당할 **RAM** 크기로 바꿉니다.
- **<size_gb>** 를 이 플레이버로 만든 인스턴스에 할당할 루트 디스크 크기로 바꿉니다.
- **<no_vcpus>** 를 이 플레이버로 생성된 인스턴스에 예약할 **vCPU** 수로 바꿉니다.
- 선택 사항: 특정 프로젝트 또는 사용자 그룹에서만 플레이버에 액세스할 수 있도록 **--private** 및 **--project** 옵션을 지정합니다. **<project_id>** 를 이 플레이버를 사용하여 인스턴스

를 생성할 수 있는 프로젝트의 **ID**로 바꿉니다. 접근성을 지정하지 않으면 플레이버가 기본적으로 **public**으로 설정되므로 모든 프로젝트에서 사용할 수 있습니다.

참고

생성 후에는 공용 플레이버를 개인용으로 만들 수 없습니다.

- **<flavor_name>** 을 플레이버의 고유 이름으로 바꿉니다.

  플레이버 인수에 대한 자세한 내용은 플레이버 인수를 참조하십시오.

2. 선택 사항: 플레이버 메타데이터를 지정하려면 키-값 쌍을 사용하여 필요한 속성을 설정합니다.

```
(overcloud)$ openstack flavor set \
 --property <key=value> --property <key=value> ... <flavor_name>
```

- **<key>** 를 이 플레이버로 생성된 인스턴스에 할당할 속성의 메타데이터 키로 바꿉니다. 사용 가능한 메타데이터 키 목록은 **Flavor metadata** 를 참조하십시오.

- **<value>** 를 이 플레이버로 생성된 인스턴스에 할당할 메타데이터 키의 값으로 바꿉니다.

- **<flavor_name>** 을 플레이버 이름으로 바꿉니다.

  예를 들어 다음 플레이버를 사용하여 시작되는 인스턴스에는 각각 두 개의 **CPU** 소켓이 있습니다.

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_sockets=2 \
 --property hw:cpu_cores=2 processor_topology_flavor
```

8.2. 플레이버 인수

openstack flavor create 명령에는 새 플레이버의 이름을 지정하기 위해 위치 인수 **<flavor_name>** 하나가 있습니다.

다음 표에는 새 플레이버를 생성할 때 필요에 따라 지정할 수 있는 선택적 인수가 자세히 나와 있습니다.

표 8.2. 선택적 플레이버 인수

선택적 인수	설명
--id	플레이버의 고유 ID입니다. 기본값인 **auto** 는 UUID4 값을 생성합니다. 이 인수를 사용하여 수동으로 정수 또는 UUID4 값을 지정할 수 있습니다.
--ram	(필수) 인스턴스에 사용할 메모리 크기(MB)입니다. 기본값: 256MB
--disk	(필수) 루트(/) 파티션에 사용할 디스크 공간(GB)입니다. 루트 디스크는 기본 이미지가 복사되는 임시 디스크입니다. 인스턴스가 영구 볼륨에서 부팅되면 루트 디스크가 사용되지 않습니다.  **참고** **--disk** 를 **0** 으로 설정한 플레이버를 사용하여 인스턴스를 만들려면 인스턴스가 볼륨에서 부팅되어야 합니다. 기본값: 0GB
--ephemeral	임시 디스크에 사용할 디스크 공간(GB)입니다. 기본값은 0GB이므로 보조 임시 디스크가 생성되지 않습니다. 임시 디스크는 인스턴스의 라이프사이클에 연결된 시스템 로컬 디스크 스토리지를 제공합니다. 임시 디스크는 어떠한 스냅샷에도 포함되지 않습니다. 이 디스크는 삭제되고 인스턴스가 삭제될 때 모든 데이터가 손실됩니다. 기본값: 0GB
--swap	스왑 디스크 크기(MB). Compute 서비스 백엔드 스토리지가 로컬 스토리지가 아닌 경우 플레이버에 **스왑** 을 지정하지 마십시오. 기본값: 0GB
--vcpus	(필수) 인스턴스의 가상 CPU 수입니다. 기본값: 1
--public	플레이버는 모든 프로젝트에서 사용할 수 있습니다. 기본적으로 플레이버는 공용이며 모든 프로젝트에서 사용할 수 있습니다.
--private	플레이버는 **--project** 옵션을 사용하여 지정한 프로젝트에서만 사용할 수 있습니다. 개인 플레이버를 생성하지만 프로젝트에 추가하지 않으면 클라우드 관리자만 플레이버를 사용할 수 있습니다.

선택적 인수	설명
--property	다음 형식의 키-값 쌍을 사용하여 지정하는 metadata 또는 "extra specs"입니다. **--property <key=value>** 이 옵션을 반복하여 여러 속성을 설정합니다.
--project	개인 플레이버를 사용할 수 있는 프로젝트를 지정합니다. private **옵션과** 함께 이 인수를 사용해야 합니다. 프로젝트를 지정하지 않으면 플레이버는 **admin** 사용자만 볼 수 있습니다. 이 옵션을 반복하여 여러 프로젝트에 대한 액세스를 허용합니다.
--project-domain	개인 플레이버를 사용할 수 있는 프로젝트 도메인을 지정합니다. private **옵션과** 함께 이 인수를 사용해야 합니다. 이 옵션을 반복하여 여러 프로젝트 도메인에 액세스할 수 있도록 허용합니다.
--description	플레이버에 대한 설명입니다. 길이가 65535개로 제한됩니다. 인쇄 가능한 문자만 사용할 수 있습니다.

8.3. 플레이버 메타데이터

플레이버를 생성할 때 --property 옵션을 사용하여 플레이버 메타데이터를 지정합니다. 플레이버 메타데이터를 *추가 사양*이라고도 합니다. 플레이버 메타데이터는 인스턴스 배치, 인스턴스 제한 및 성능에 영향을 미치는 인스턴스 하드웨어 지원 및 할당량을 결정합니다.

인스턴스 리소스 사용

다음 표의 속성 키를 사용하여 인스턴스별로 CPU, 메모리 및 디스크 I/O 사용량에 대한 제한을 구성합니다.

표 8.3. 리소스 사용을 위한 플레이버 메타데이터

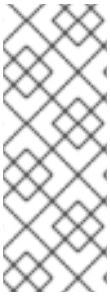
키	설명
quota:cpu_shares	도메인의 CPU 시간의 비례 가중치 공유를 지정합니다. 기본적으로 제공되는 OS 기본값입니다. 계산 스케줄러는 동일한 도메인의 다른 인스턴스에서 이 속성의 설정을 기준으로 이 값의 가중치를 적용합니다. 예를 들어 **quota:cpu_shares=2048** 로 구성된 인스턴스에 **quota:cpu_shares=1024** 로 구성된 인스턴스로 CPU 시간이 두 배나 할당됩니다.

키	설명
quota:cpu_period	**cpu_quota** 를 마이크로초 단위로 적용할 기간을 지정합니다. **cpu_period** 내에서 각 vCPU는 런타임의 **cpu_quota** 이상을 사용할 수 없습니다. 1000 - 1000000 범위의 값으로 설정합니다. 비활성화하려면 **0** 으로 설정합니다.

quota:cpu_quota	각 **cpu_period** 에서 vCPU에 대해 허용되는 최대 대역폭을 마이크로초 단위로 지정합니다.
	● 1000 - 1844674407370955 범위의 값으로 설정합니다.
	● 비활성화하려면 **0** 으로 설정합니다.
	● 무한 대역폭을 허용하려면 음수 값으로 설정합니다.
	cpu_quota 및 **cpu_period** 를 사용하여 모든 vCPU가 동일한 속도로 실행되는지 확인할 수 있습니다. 예를 들어 다음 플레이버를 사용하여 물리적 CPU 컴퓨팅 기능의 최대 50% CPU만 사용할 수 있는 인스턴스를 시작할 수 있습니다.
	`$ openstack flavor set cpu_limits_flavor \`
	`--property quota:cpu_quota=10000 \`
	`--property quota:cpu_period=20000`

인스턴스 디스크 튜닝

다음 표의 속성 키를 사용하여 인스턴스 디스크 성능을 조정합니다.



참고

Compute 서비스는 계산 서비스에서 프로비저닝한 스토리지(예: 임시 스토리지)에 다음과 같은 서비스 설정을 적용합니다. Block Storage(cinder) 볼륨의 성능을 조정하려면 볼륨 유형에 대해 QoS(Quality-of-Service) 값도 구성해야 합니다. 자세한 내용은 *스토리지 가이드* 의 서비스 품질 사양 사용을 참조하십시오.

표 8.4. 디스크 튜닝을 위한 플레이버 메타데이터

키	설명
quota:disk_read_bytes_sec	인스턴스에서 사용할 수 있는 최대 디스크 읽기 수를 초당 바이트 단위로 지정합니다.
quota:disk_read_iops_sec	IOPS에서 인스턴스에서 사용할 수 있는 최대 디스크 읽기를 지정합니다.

키	설명
quota:disk_write_bytes_sec	인스턴스에 사용할 수 있는 최대 디스크 쓰기 수를 초당 바이트 단위로 지정합니다.
quota:disk_write_iops_sec	IOPS에서 인스턴스에서 사용할 수 있는 최대 디스크 쓰기 수를 지정합니다.
quota:disk_total_bytes_sec	인스턴스에 사용할 수 있는 최대 I/O 작업(초당 바이트)을 지정합니다.
quota:disk_total_iops_sec	IOPS에서 인스턴스에서 사용할 수 있는 최대 I/O 작업을 지정합니다.

인스턴스 네트워크 트래픽 대역폭

다음 표의 속성 키를 사용하여 VIF I/O 옵션을 구성하여 인스턴스 네트워크 트래픽에 대한 대역폭 제한을 구성합니다.

참고

할당량 :vif_* 속성은 더 이상 사용되지 않습니다. 대신 Networking(neutron) 서비스 품질(QoS) 정책을 사용해야 합니다. QoS 정책에 대한 자세한 내용은 *네트워킹 가이드* 의 QoS(Quality of Service) 정책 구성을 참조하십시오. quota:vif_* 속성은 NeutronOVSFirewallDriver가 iptables_hybrid 로 설정된 ML2/OVS 메커니즘 드라이버를 사용하는 경우에만 지원됩니다.

표 8.5. 대역폭 제한에 대한 플레이버 메타데이터

키	설명
quota:vif_inbound_average	(폐기됨) kbps에서 인스턴스로 들어오는 트래픽에 필요한 평균 비트 속도를 지정합니다.
quota:vif_inbound_burst	(더 이상 사용되지 않음) 최대 수신 트래픽 양을 KB 단위로 최고 속도로 버스트할 수 있습니다.
quota:vif_inbound_peak	(폐기됨) 인스턴스가 kbps에서 들어오는 트래픽을 수신할 수 있는 최대 속도를 지정합니다.
quota:vif_outbound_average	(폐기됨) 인스턴스에서 나가는 트래픽에 필요한 평균 비트 속도를 kbps로 지정합니다.
quota:vif_outbound_burst	(더 이상 사용되지 않음) 최고 속도(KB)로 버스트할 수 있는 나가는 트래픽의 최대 양을 지정합니다.

키	설명
quota:vif_outbound_peak	(폐기됨) 인스턴스가 kbps에서 나가는 트래픽을 보낼 수 있는 최대 속도를 지정합니다.

하드웨어 비디오 RAM

다음 표의 속성 키를 사용하여 비디오 장치에 사용할 인스턴스 RAM에 대한 제한을 구성합니다.

표 8.6. 비디오 장치의 플레이버 메타데이터

키	설명
hw_video:ram_max_mb	비디오 장치에 사용할 최대 RAM(MB)을 지정합니다. **hw_video_ram** 이미지 속성과 함께 사용합니다. **hw_video_ram** 은 **hw_video:ram_max_mb** 보다 작거나 같아야 합니다.

워치독 동작

다음 표의 속성 키를 사용하여 인스턴스에서 가상 하드웨어 워치독 장치를 활성화합니다.

표 8.7. 워치독 동작의 플레이버 메타데이터

키	설명

키	설명
hw:watchdog_action	가상 하드웨어 워치독 장치를 활성화하고 동작을 설정하도록 를 지정합니다. 인스턴스가 중지되거나 실패한 경우 워치독 장치는 구성된 작업을 수행합니다. 워치독은 PCI Intel 6300ESB를 에뮬레이트하는 i6300esb 장치를 사용합니다. **hw:watchdog_action** 이 지정되지 않은 경우 워치독이 비활성화됩니다. 다음 유효한 값 중 하나로 설정합니다. ● **disabled**: (기본값) 장치가 연결되어 있지 않습니다. ● **reset**: 강제로 인스턴스를 재설정합니다. ● **poweroff**: 인스턴스를 강제 종료합니다. ● **pause**: 인스턴스를 일시 중지합니다. ● **none**: 워치독을 활성화하지만 인스턴스가 중지되거나 실패하면 아무 작업도 수행하지 않습니다.  **참고** 특정 이미지의 속성을 사용하여 설정한 워치독 동작은 플레이버를 사용하여 설정한 동작을 재정의합니다.

임의 번호 생성기 (RNG)

다음 표의 속성 키를 사용하여 인스턴스에서 RNG 장치를 활성화합니다.

표 8.8. RNG의 플레이버 메타데이터

키	설명
hw_rng:allowed	이미지 속성을 통해 인스턴스에 추가된 RNG 장치를 비활성화하려면 **False** 로 설정합니다. 기본값: **True**
hw_rng:rate_bytes	인스턴스가 호스트의 엔트로피에서 읽을 수 있는 최대 바이트 수를 기간별로 지정합니다.
hw_rng:rate_period	읽기 기간(밀리초)을 지정합니다.

가상 성능 모니터링 단위(vPMU)

다음 표의 속성 키를 사용하여 인스턴스에 vPMU를 활성화합니다.

표 8.9. vPMU의 플레이버 메타데이터

키	설명
hw:pmu	인스턴스에 vPMU를 활성화하려면 **True** 로 설정합니다. **perf** 와 같은 툴은 인스턴스에서 vPMU를 사용하여 더 정확한 정보를 제공하고 인스턴스 성능을 모니터링합니다. 실시간 워크로드의 경우 vPMU의 에뮬레이션에 바람직하지 않을 수 있는 추가 대기 시간이 발생할 수 있습니다. 원격 분석이 필요하지 않은 경우 **hw:pmu=False** 를 설정합니다.

인스턴스 CPU 토폴로지

다음 테이블의 속성 키를 사용하여 인스턴스에서 프로세서의 토폴로지를 정의합니다.

표 8.10. CPU 토폴로지의 플레이버 메타데이터

키	설명
hw:cpu_sockets	인스턴스의 기본 소켓 수를 지정합니다. default: 요청된 vCPU 수
hw:cpu_cores	인스턴스의 소켓당 기본 코어 수를 지정합니다. 기본값: **1**
hw:cpu_threads	인스턴스의 기본 코어 수를 지정합니다. 기본값: **1**
hw:cpu_max_sockets	이미지 속성을 사용하여 사용자가 해당 인스턴스에 대해 선택할 수 있는 최대 소켓 수를 지정합니다. Example: **hw:cpu_max_sockets=2**
hw:cpu_max_cores	사용자가 이미지 속성을 사용하여 해당 인스턴스에 대해 선택할 수 있는 소켓당 최대 코어 수를 지정합니다.
hw:cpu_max_threads	사용자가 이미지 속성을 사용하여 인스턴스에 대해 선택할 수 있는 코어당 최대 스레드 수를 지정합니다.

직렬 포트

다음 표의 속성 키를 사용하여 인스턴스당 직렬 포트 수를 구성합니다.

표 8.11. 직렬 포트의 플레이버 메타데이터

키	설명
hw:serial_port_count	인스턴스당 최대 직렬 포트 수.

CPU 고정 정책

기본적으로 인스턴스 가상 **CPU(vCPU)**는 하나의 코어와 하나의 스레드가 있는 소켓입니다. 속성을 사용하여 인스턴스의 **vCPU**를 호스트의 물리적 **CPU** 코어**(pCPU)**에 고정하는 플레이버를 만들 수 있습니다. 하나 이상의 코어에 스레드 스레딩이 있는 **SMT(**동시 멀티스레딩**)** 아키텍처에서 하드웨어 **CPU** 스레드 동작을 구성할 수도 있습니다.

다음 테이블의 속성 키를 사용하여 인스턴스의 **CPU** 고정 정책을 정의합니다.

표 8.12. CPU 고정을 위한 플레이버 메타데이터

키	설명
hw:cpu_policy	사용할 CPU 정책을 지정합니다. 다음 유효한 값 중 하나로 설정합니다. ● **shared**: (기본값) 인스턴스 vCPU는 호스트 pCPU 간에 유동합니다. ● **전용**: 인스턴스 vCPU를 호스트 pCPU 집합에 고정합니다. 그러면 인스턴스가 고정되는 CPU 토폴로지와 일치하는 인스턴스 CPU 토폴로지가 생성됩니다. 이 옵션은 1.0의 과다 할당 비율을 의미합니다.

키	설명

hw:cpu_thread_policy	**hw:cpu_policy=dedicated** 경우 사용할 CPU 스레드 정책을 지정합니다. 다음 유효한 값 중 하나로 설정합니다.
	● **prefer**: (기본값) 호스트에 SMT 아키텍처가 있을 수도 있고 없을 수도 있습니다. SMT 아키텍처가 있는 경우 Compute 스케줄러는 스레드 에이전트를 선호합니다.
	● **격리**: 호스트에 SMT 아키텍처가 없거나 SMT 이외의 아키텍처를 에뮬레이션해야 합니다. 이 정책을 사용하면 계산 스케줄러에서 **HW_CPU_HYPERTHREADING** 특성을 보고하지 않는 호스트를 요청하여 SMT 없이 호스트에 인스턴스를 배치할 수 있습니다. 다음 속성을 사용하여 이 특성을 명시적으로 요청할 수도 있습니다.
	`--property trait:HW_CPU_HYPERTHREADING=forbidden`
	호스트에 SMT 아키텍처가 없는 경우 계산 서비스는 각 vCPU를 예상대로 다른 코어에 배치합니다. 호스트에 SMT 아키텍처가 있는 경우 **[workarounds]/disable_fallback_pcpu_query** 매개변수 구성으로 동작을 결정합니다.
	○ **True**: SMT 아키텍처가 포함된 호스트가 사용되지 않으며 스케줄링이 실패합니다.
	○ **False**: 계산 서비스는 각 vCPU를 다른 물리적 코어에 배치합니다. 계산 서비스는 동일한 코어에 있는 다른 인스턴스의 vCPU를 배치하지 않습니다. 사용된 각 코어에 대해 하나의 스레드를 제외하고 모두 사용할 수 없으므로 사용할 수 없게 됩니다.
	● **요구 사항**: 호스트에는 SMT 아키텍처가 있어야 합니다. 이 정책을 사용하면 계산 스케줄러에서 **HW_CPU_HYPERTHREADING** 특성을 보고하는 호스트를 요청하여 SMT가 있는 호스트에 인스턴스를 배치할 수 있습니다. 다음 속성을 사용하여 이 특성을 명시적으로 요청할 수도 있습니다.
	`--property trait:HW_CPU_HYPERTHREADING=required`
	계산 서비스는 스레드 시블링에 각 vCPU를 할당합니다. 호스트에 SMT 아키텍처가 없는 경우 사용되지 않습니다. 호스트에 SMT 아키텍처가 있지만 사용 가능한 스레드 스레딩이 충분한 코어가 없는 경우 스케줄링에 실패합니다.

인스턴스 PCI NUMA 선호도 정책

다음 표에서 속성 키를 사용하여 **PCI** 패스스루 장치 및 **SR-IOV** 인터페이스에 대한 **NUMA** 선호도 정책을 지정하는 플레이버를 만듭니다.

표 8.13. PCI NUMA 선호도 정책의 플레이버 메타데이터

키	설명

hw:pci_numa_affinity_policy	PCI 통과 장치 및 SR-IOV 인터페이스에 대한 NUMA 선호도 정책을 지정합니다. 다음 유효한 값 중 하나로 설정합니다.
	● **필수** 항목 : 계산 서비스는 인스턴스의 NUMA 노드 중 하나 이상이 PCI 장치와 선호도가 있는 경우에만 PCI 장치를 요청하는 인스턴스를 생성합니다. 이 옵션은 최고의 성능을 제공합니다.
	● **preferred**: 계산 서비스는 NUMA 선호도를 기반으로 PCI 장치를 가장 효과적으로 선택하려고 시도합니다. 이 작업을 수행할 수 없는 경우 Compute 서비스는 PCI 장치와의 선호도가 없는 NUMA 노드에 인스턴스를 예약합니다.
	● **레거시**: (기본값) 계산 서비스는 다음 사례 중 하나에서 PCI 장치를 요청하는 인스턴스를 생성합니다.
	  ○ PCI 장치에는 NUMA 노드 중 하나와 선호도가 있습니다.
	  ○ PCI 장치는 NUMA 선호도에 대한 정보를 제공하지 않습니다.

인스턴스 NUMA 토폴로지

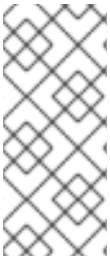
속성을 사용하여 인스턴스 vCPU 스레드에 대한 호스트 NUMA 배치 및 호스트 NUMA 노드에서 인스턴스 vCPU 및 메모리 할당을 정의하는 플레이버를 생성할 수 있습니다.

인스턴스의 NUMA 토폴로지를 정의하면 메모리 및 vCPU 할당이 계산 호스트의 NUMA 노드 크기보다 큰 플레이버의 인스턴스 OS의 성능이 향상됩니다.

계산 스케줄러는 이러한 속성을 사용하여 인스턴스에 적합한 호스트를 결정합니다. 예를 들어 클라우드 사용자는 다음 플레이버를 사용하여 인스턴스를 시작합니다.

```
$ openstack flavor set numa_top_flavor \
  --property hw:numa_nodes=2 \
  --property hw:numa_cpus.0=0,1,2,3,4,5 \
  --property hw:numa_cpus.1=6,7 \
  --property hw:numa_mem.0=3072 \
  --property hw:numa_mem.1=1024
```

계산 스케줄러는 3GB RAM과 CPU 6개를 실행하는 기능과 1GB의 RAM과 CPU 2개가 있는 2개의 NUMA 노드가 있는 호스트를 검색합니다. 호스트에 8개의 CPU와 4GB RAM을 실행하는 기능이 있는 단일 NUMA 노드가 있는 경우 Compute 스케줄러는 유효한 일치 것으로 간주하지 않습니다.



참고

플레이버에서 정의하는 **NUMA** 토폴로지는 이미지에서 정의하는 **NUMA** 토폴로지로 재정의할 수 없습니다. 이미지 **NUMA** 토폴로지가 플레이버 **NUMA** 토폴로지와 충돌하는 경우 계산 서비스에서 **ImageNUMATopologyForbidden** 오류가 발생합니다.

경고

이 기능을 사용하여 특정 호스트 **CPU** 또는 **NUMA** 노드로 인스턴스를 제한할 수 없습니다. 이 기능은 광범위한 테스트 및 성능 측정을 완료한 후에만 사용하십시오. 대신 **hw:pci_numa_affinity_policy** 속성을 사용할 수 있습니다.

다음 테이블의 속성 키를 사용하여 인스턴스 **NUMA** 토폴로지를 정의합니다.

표 **8.14. NUMA** 토폴로지의 플레이버 메타데이터

키	설명
hw:numa_nodes	인스턴스 vCPU 스레드의 실행을 제한할 호스트 NUMA 노드 수를 지정합니다. 지정하지 않으면 vCPU 스레드를 사용 가능한 호스트 NUMA 노드 수에서 실행할 수 있습니다.
hw:numa_cpus.N	인스턴스 NUMA 노드 N에 매핑할 인스턴스 vCPU의 쉼표로 구분된 목록입니다. 이 키를 지정하지 않으면 사용 가능한 NUMA 노드 간에 vCPU가 균등하게 나뉩니다. N은 0부터 시작합니다. *.N 값을 주의해서 사용하고 2개 이상의 NUMA 노드가 있는 경우에만 사용합니다. 이 속성은 **hw:numa_nodes** 를 설정한 경우에만 유효하며, 인스턴스의 NUMA 노드에 CPU 및 RAM의 비대칭 할당이 있는 경우에만 필요합니다. 이 속성은 일부 NFV 워크로드에 중요한 CPU 및 RAM의 비대칭 할당이 있는 경우에만 유효합니다.
hw:numa_mem.N	인스턴스 NUMA 노드 N에 매핑할 인스턴스 메모리의 수입니다. 이 키를 지정하지 않으면 사용 가능한 NUMA 노드 간에 메모리가 균등하게 나뉩니다. N은 0부터 시작합니다. *.N 값을 주의해서 사용하고 2개 이상의 NUMA 노드가 있는 경우에만 사용합니다. 이 속성은 **hw:numa_nodes** 를 설정한 경우에만 유효하며, 인스턴스의 NUMA 노드에 CPU 및 RAM의 비대칭 할당이 있는 경우에만 필요합니다. 이 속성은 일부 NFV 워크로드에 중요한 CPU 및 RAM의 비대칭 할당이 있는 경우에만 유효합니다.

주의

hw:numa_cpus.N 또는 **hw:numa_mem.N** 의 결합된 값이 각각 사용 가능한 **CPU** 또는 메모리 수보다 크면 계산 서비스에서 예외가 발생합니다.

인스턴스 메모리 암호화

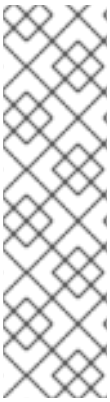
다음 표의 속성 키를 사용하여 인스턴스 메모리의 암호화를 활성화합니다.

표 8.15. 메모리 암호화를 위한 플레이버 메타데이터

키	설명
hw:mem_encryption	인스턴스의 메모리 암호화를 요청하려면 **True** 로 설정합니다. 자세한 내용은 Configuring AMD SEV Compute nodes to provide memory encryption for instances를 참조하십시오.

CPU 실시간 정책

다음 테이블의 속성 키를 사용하여 인스턴스에서 프로세서의 실시간 정책을 정의합니다.



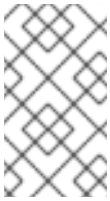
참고

- 대부분의 인스턴스 **vCPU**는 실시간 정책으로 실행할 수 있지만 최소 하나 이상의 **vCPU**를 비실시간 게스트 프로세스 및 에뮬레이터 오버헤드 프로세스에 사용하도록 표시해야 합니다.
- 이 추가 사양을 사용하려면 고정된 **CPU**를 활성화해야 합니다.

표 8.16. CPU 실시간 정책의 플레이버 메타데이터

키	설명
hw:cpu_realtime	인스턴스 vCPU에 실시간 정책을 할당하는 플레이버를 만들려면 **yes** 로 설정합니다. 기본값: **no**

키	설명

hw:cpu_realtime_mask	실시간 정책을 할당하지 않을 vCPU를 지정합니다. 마스크 값 앞에 캐럿 기호(^)를 추가해야 합니다. 다음 예제는 vCPU 0과 1을 제외한 모든 vCPU에 실시간 정책이 있음을 나타냅니다.
	`$ openstack flavor set <flavor> \`
	`--property hw:cpu_realtime="yes" \`
	`--property hw:cpu_realtime_mask=^0-1`
	 
	참고
	이미지에 **hw_cpu_realtime_mask** 속성이 설정된 경우 플레이버에 설정된 **hw:cpu_realtime_mask** 속성보다 우선합니다.

에뮬레이터 스레드 정책

인스턴스에 pCPU를 할당하여 에뮬레이터 스레드에 사용할 수 있습니다. 에뮬레이터 스레드는 인스턴스와 직접 관련이 없는 에뮬레이터 프로세스입니다. 실시간 워크로드에는 전용 에뮬레이터 스레드 pCPU가 필요합니다. 에뮬레이터 스레드 정책을 사용하려면 다음 속성을 설정하여 고정된 CPU를 활성화해야 합니다.

```
--property hw:cpu_policy=dedicated
```

다음 테이블에서 속성 키를 사용하여 인스턴스의 에뮬레이터 스레드 정책을 정의합니다.

표 8.17. 에뮬레이터 스레드 정책의 플레이버 메타데이터

키	설명

키	설명
hw:emulator_threads_policy	인스턴스에 사용할 에뮬레이터 스레드 정책을 지정합니다. 다음 유효한 값 중 하나로 설정합니다. ● **share**: 에뮬레이터 스레드는 **NovaComputeCpuSharedSet** heat 매개변수에 정의된 pCPU 전반에서 유동됩니다. **NovaComputeCpuSharedSet** 이 구성되지 않은 경우 에뮬레이터 스레드는 인스턴스와 연결된 고정된 CPU를 통과합니다. ● **격리**: 에뮬레이터 스레드를 위해 인스턴스당 추가 전용 pCPU를 예약합니다. 리소스를 과다적으로 많이 사용하므로 이 정책을 주의해서 사용하십시오. ● 설정 해제: (기본값) 에뮬레이터 스레드 정책이 활성화되지 않고, 인스턴스와 연결된 고정된 CPU에서 에뮬레이터 스레드 유동을 수행합니다.

인스턴스 메모리 페이지 크기

다음 테이블의 속성 키를 사용하여 명시적 메모리 페이지 크기가 있는 인스턴스를 생성합니다.

표 8.18. 메모리 페이지 크기에 대한 플레이버 메타데이터

키	설명
hw:mem_page_size	인스턴스를 백업하는 데 사용할 큰 페이지 크기를 지정합니다. 이 옵션을 사용하면 **hw:numa_nodes** 에서 지정하지 않는 한 1 NUMA 노드의 암시적 NUMA 토폴로지가 생성됩니다. 다음 유효한 값 중 하나로 설정합니다. ● **large**: 호스트에서 지원되는 최소 페이지 크기보다 큰 페이지 크기를 선택합니다. x86_64 시스템에서 2MB 또는 1GB일 수 있습니다. ● **small**: 호스트에서 지원되는 최소 페이지 크기를 선택합니다. x86_64 시스템에서는 4 kB(일반 페이지)입니다. ● **모든**: libvirt 드라이버에 따라 사용 가능한 가장 큰 대규모 페이지 크기를 선택합니다. ● **<pagesize>**: (문자열) 워크로드에 특정 요구 사항이 있는 경우 명시적 페이지 크기를 설정합니다. 페이지 크기(KB) 또는 표준 접미사에 정수 값을 사용합니다. 예를 들면 다음과 같습니다. **4KB**, **2MB**, **2048**, **1GB**. ● 설정 해제: (기본값) 대규모 페이지는 인스턴스를 지원하는 데 사용되지 않으며 암시적인 NUMA 토폴로지가 생성되지 않습니다.

PCI 통과

다음 표의 속성 키를 사용하여 그래픽 카드 또는 네트워크 장치와 같은 물리적 PCI 장치를 인스턴스에 연결합니다. PCI 패스스루 사용에 대한 자세한 내용은 PCI 패스스루 구성을 참조하십시오.

표 8.19. PCI 패스스루의 플레이버 메타데이터

키	설명
pci_passthrough:alias	다음 형식을 사용하여 인스턴스에 할당할 PCI 장치를 지정합니다. <alias>:<count> • <alias> 를 특정 PCI 장치 클래스에 해당하는 별칭으로 바꿉니다. • <count> 를 인스턴스에 할당할 유형의 PCI 장치 수로 바꿉니다.

하이퍼바이저 서명

다음 표의 속성 키를 사용하여 인스턴스에서 하이퍼바이저 서명을 숨깁니다.

표 8.20. 하이퍼바이저 서명을 숨기기 위한 플레이버 메타데이터

키	설명
hide_hypervisor_id	인스턴스에서 하이퍼바이저 서명을 숨기려면 True 로 설정합니다. 모든 드라이버가 인스턴스에서 로드하고 작업할 수 있습니다.

인스턴스 리소스 특성

각 리소스 프로바이더에는 일련의 특성이 있습니다. 특성은 리소스 프로바이더의 질적 측면입니다(예: 스토리지 디스크 유형 또는 Intel CPU 명령 집합 확장). 인스턴스에서 이러한 특성 중 필요한 특성을 지정할 수 있습니다.

지정할 수 있는 특성은 os-traits 라이브러리에 정의되어 있습니다. 예제 특성에는 다음이 포함됩니다.

- COMPUTE_TRUSTED_CERTS
- COMPUTE_NET_ATTACH_INTERFACE_WITH_TAG

- **COMPUTE_IMAGE_TYPE_RAW**
- **HW_CPU_X86_AVX**
- **HW_CPU_X86_AVX512VL**
- **HW_CPU_X86_AVX512CD**

os-traits 라이브러리 사용 방법에 대한 자세한 내용은 **https://docs.openstack.org/os-traits/latest/user/index.html** 을 참조하십시오.

다음 테이블의 속성 키를 사용하여 인스턴스의 리소스 특성을 정의합니다.

표 8.21. 리소스 특성의 플레이버 메타데이터

키	설명
trait:<trait_name>	컴퓨팅 노드 특성을 지정합니다. 특성을 다음 유효한 값 중 하나로 설정합니다. • **필수** 항목 : 인스턴스를 호스팅하도록 선택한 컴퓨팅 노드에는 특성이 있어야 합니다. • **허용되지** 않음 : 인스턴스를 호스팅하도록 선택한 컴퓨팅 노드에는 특성이 없어야 합니다. 예제: `$ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512-flavor`

인스턴스 베어메탈 리소스 클래스

다음 테이블의 속성 키를 사용하여 인스턴스에 대한 베어 메탈 리소스 클래스를 요청합니다.

표 8.22. 베어 메탈 리소스 클래스의 플레이버 메타데이터

키	설명

키	설명

resources:<resource_class_name>	이 속성을 사용하여 표준 베어 메탈 리소스 클래스를 지정하여 해당 값을 재정의하거나 인스턴스에 필요한 사용자 지정 베어 메탈 리소스 클래스를 지정합니다.
	
	재정의할 수 있는 표준 리소스 클래스는 **VCPU**,**MEMORY_MB** 및 **DISK_GB** 입니다. 계산 스케줄러가 인스턴스 예약에 베어 메탈 플레이버 속성을 사용하지 않도록 하려면 표준 리소스 클래스의 값을 **0** 으로 설정합니다.
	
	사용자 정의 리소스 클래스의 이름은 **CUSTOM_** 로 시작해야 합니다. 베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 모든 문장 부호를 밑줄로 바꾸고 접두사는 CUSTOM_로 바꿉니다.
	
	예를 들어 **--resource-class baremetal.SMALL** 이 있는 노드에 인스턴스를 예약하려면 다음 플레이버를 생성합니다.
	
	`$ openstack flavor set \`
	`--property`
	`resources:CUSTOM_BAREMETAL_SMALL=1 \`
	`--property resources:VCPU=0 --property`
	`resources:MEMORY_MB=0 \`
	`--property resources:DISK_GB=0 compute-small`

9장. 인스턴스에 메타데이터 추가

계산(nova) 서비스는 메타데이터를 사용하여 시작 시 구성 정보를 인스턴스에 전달합니다. 인스턴스는 구성 드라이브 또는 메타데이터 서비스를 사용하여 메타데이터에 액세스할 수 있습니다.

설정 드라이브

구성 드라이브는 부팅 시 인스턴스에 연결할 수 있는 특수 드라이브입니다. 구성 드라이브는 인스턴스에 읽기 전용 드라이브로 표시됩니다. 인스턴스에서 이 드라이브를 마운트하고 파일을 읽어 메타데이터 서비스를 통해 일반적으로 사용할 수 있는 정보를 가져올 수 있습니다.

메타데이터 서비스

계산 서비스는 인스턴스와 관련된 데이터를 검색하는 데 사용할 수 있는 REST API로 메타데이터 서비스를 제공합니다. 인스턴스는 169.254.169.254 또는 fe80::a9fe:a9fe에서 이 서비스에 액세스합니다.

9.1. 인스턴스 메타데이터 유형

클라우드 사용자, 클라우드 관리자 및 계산 서비스는 메타데이터를 인스턴스에 전달할 수 있습니다.

클라우드 사용자가 데이터 제공

클라우드 사용자는 인스턴스가 부팅 시 실행되는 쉘 스크립트와 같이 인스턴스를 시작할 때 사용할 추가 데이터를 지정할 수 있습니다. 클라우드 사용자는 사용자 데이터 기능을 사용하고 인스턴스를 만들거나 업데이트할 때 키-값 쌍을 필수 속성으로 전달하여 인스턴스에 데이터를 전달할 수 있습니다.

클라우드 관리자가 데이터 제공

RHOSP 관리자는 vendordata 기능을 사용하여 데이터를 인스턴스에 전달합니다. 계산 서비스는 관리자가 인스턴스에 메타데이터를 전달할 수 있도록 vendordata 모듈 StaticJSON 및 DynamicJSON 을 제공합니다.

- StaticJSON: (기본값) 모든 인스턴스에 대해 동일한 메타데이터에 사용합니다.
- DynamicJSON: 각 인스턴스에 따라 다른 메타데이터에 를 사용합니다. 이 모듈은 외부 REST 서비스에 요청하여 인스턴스에 추가할 메타데이터를 결정합니다.

Vendordata 구성은 인스턴스의 다음 읽기 전용 파일 중 하나에 있습니다.

- /openstack/{version}/vendor_data.json

- /openstack/{version}/vendor_data2.json

Compute 서비스 제공 데이터

계산 서비스는 메타데이터 서비스의 내부 구현을 사용하여 인스턴스의 요청된 호스트 이름 및 인스턴스가 속한 가용성 영역과 같은 정보를 인스턴스에 전달합니다. 이러한 작업은 기본적으로 수행되며 클라우드 사용자 또는 관리자가 구성이 필요하지 않습니다.

9.2. 모든 인스턴스에 구성 드라이브 추가

관리자는 인스턴스에 대한 구성 드라이브를 항상 생성하고 배포와 관련된 메타데이터로 구성 드라이브를 채우도록 계산 서비스를 구성할 수 있습니다. 예를 들어 다음과 같은 이유로 구성 드라이브를 사용할 수 있습니다.

- 배포에서 **DHCP**를 사용하지 않는 경우 네트워킹 구성을 전달하여 인스턴스에 **IP** 주소를 할당합니다. 인스턴스의 네트워크 설정을 구성하기 전에 인스턴스가 마운트하고 액세스할 수 있는 **config** 드라이브를 통해 인스턴스의 **IP** 주소 구성을 전달할 수 있습니다.

- 인스턴스를 시작하는 사용자에게 알 수 없는 인스턴스에 데이터를 전달하려면 예를 들어 **Active Directory** 사후 부팅으로 인스턴스를 등록하는 데 사용할 암호화 토큰입니다.

- 로컬 캐시된 디스크 읽기를 생성하여 인스턴스 요청의 부하를 관리하려면 메타데이터 서버에 액세스하는 인스턴스의 영향을 정기적으로 줄여 팩트를 확인하고 빌드합니다.

ISO 9660 또는 **VFAT** 파일 시스템을 마운트할 수 있는 모든 인스턴스 운영 체제는 구성 드라이브를 사용할 수 있습니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 인스턴스를 시작할 때 항상 구성 드라이브를 연결하려면 다음 매개 변수를 **True** 로 설정합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
```

3. 선택 사항: 구성 드라이브 형식을 **iso9660** 의 기본값에서 **vfat** 로 변경하려면 **config_drive_format** 매개변수를 구성에 추가합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
    nova::compute::config_drive_format: vfat
```

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml \
```

검증

1. 인스턴스를 생성합니다.

```
(overcloud)$ openstack server create --flavor m1.tiny \
 --image cirros test-config-drive-instance
```

2. 인스턴스에 로그인합니다.

3. 구성 드라이브를 마운트합니다.

- 인스턴스 **OS**에서 **udev** 를 사용하는 경우 **:**

```
# mkdir -p /mnt/config
# mount /dev/disk/by-label/config-2 /mnt/config
```

- 인스턴스 **OS**가 **udev** 를 사용하지 않는 경우 먼저 구성 드라이브에 해당하는 블록 장치를 식별해야 합니다.

```
# blkid -t LABEL="config-2" -odevice
/dev/vdb
# mkdir -p /mnt/config
# mount /dev/vdb /mnt/config
```

4. 메타데이터의 마운트된 구성 드라이브 디렉터리인 **mnt/config/openstack/{version}/** 의 파일을 검사합니다.

9.3. 인스턴스에 정적 메타데이터 추가

배포의 모든 인스턴스에서 정적 메타데이터를 사용할 수 있습니다.

절차

1. 메타데이터의 **JSON** 파일을 생성합니다.

2. 컴퓨팅 환경 파일을 엽니다.

3. 환경 파일에 대한 **JSON** 파일의 경로를 추가합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      ...
      api/vendordata_jsonfile_path:
        value: <path_to_the_JSON_file>
```

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml \
```

9.4. 인스턴스에 동적 메타데이터 추가

인스턴스별 메타데이터를 생성하고 **JSON** 파일을 통해 해당 인스턴스에서 메타데이터를 사용할 수 있도록 배포를 구성할 수 있습니다.

작은 정보

언더클라우드에서 동적 메타데이터를 사용하여 **director**를 **Red Hat IdM(Identity Management)** 서버와 통합할 수 있습니다. **IdM** 서버를 인증 기관으로 사용하고 오버클라우드에서 **SSL/TLS**를 활성화하면 오버클라우드 인증서를 관리할 수 있습니다. 자세한 내용은 **IdM**에 언더클라우드 추가를 참조하십시오.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. **vendordata** 공급자 모듈에 **DynamicJSON** 을 추가합니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      ...
      api/vendordata_providers:
        value: StaticJSON,DynamicJSON
```

3. 메타데이터를 생성하려면 연결할 **REST** 서비스를 지정합니다. 다음과 같이 필요한 만큼 대상 **REST** 서비스를 지정할 수 있습니다.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      ...
      api/vendordata_providers:
        value: StaticJSON,DynamicJSON
      api/vendordata_dynamic_targets:
        value: target1@http://127.0.0.1:125
      api/vendordata_dynamic_targets:
        value: target2@http://127.0.0.1:126
```

계산 서비스는 구성된 대상 서비스에서 검색한 메타데이터를 포함하도록 **JSON** 파일 **vendordata2.json** 을 생성하여 **config** 드라이브 디렉터리에 저장합니다.

참고

대상 서비스에 동일한 이름을 두 번 이상 사용하지 마십시오.

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

10장. 인스턴스의 CPU 기능 플래그 구성

호스트 컴퓨팅 노드의 설정을 변경하고 컴퓨팅 노드를 재부팅하지 않고 인스턴스에 대한 CPU 기능 플래그를 활성화하거나 비활성화할 수 있습니다. 인스턴스에 적용되는 표준 CPU 기능 플래그 세트를 구성하면 컴퓨팅 노드 전체에서 실시간 마이그레이션 호환성을 달성할 수 있습니다. 또한 특정 CPU 모델을 사용하여 인스턴스의 보안 또는 성능 성능에 부정적인 영향을 미치는 플래그를 비활성화하거나 보안 문제로부터 완화하거나 성능 문제를 완화하는 플래그를 활성화하여 인스턴스의 성능 및 보안을 관리할 수 있습니다.

10.1. 사전 요구 사항

- 호스트 컴퓨팅 노드의 하드웨어 및 소프트웨어에서 CPU 모델 및 기능 플래그를 지원해야 합니다.
  - 호스트에서 지원하는 하드웨어를 확인하려면 컴퓨팅 노드에 다음 명령을 입력합니다.

```
$ cat /proc/cpuinfo
```

  - 호스트에서 지원되는 CPU 모델을 확인하려면 컴퓨팅 노드에 다음 명령을 입력합니다.

```
$ sudo podman exec -it nova_libvirt virsh cpu-models <arch>
```

    <arch> 를 아키텍처의 이름으로 바꿉니다(예: x86_64).

10.2. 인스턴스의 CPU 기능 플래그 구성

특정 vCPU 모델이 있는 인스턴스에 CPU 기능 플래그를 적용하도록 계산 서비스를 구성합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. 컴퓨팅 환경 파일을 엽니다.

4. 인스턴스 CPU 모드를 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: <cpu_mode>
```

<cpu_mode> 를 Compute 노드에 있는 각 인스턴스의 CPU 모드로 바꿉니다. 다음 유효한 값 중 하나로 설정합니다.

- host-model: (기본값) 호스트 컴퓨팅 노드의 CPU 모델을 사용합니다. 이 CPU 모드를 사용하여 중요한 CPU 플래그를 인스턴스에 자동으로 추가하여 보안 결함으로부터 완화를 제공합니다.

- custom: 를 사용하여 각 인스턴스에서 사용해야 하는 특정 CPU 모델을 구성합니다.

참고

CPU 모드를 host-passthrough 로 설정하여 해당 컴퓨팅 노드에 호스팅된 인스턴스의 컴퓨팅 노드와 동일한 CPU 모델 및 기능 플래그를 사용할 수도 있습니다.

5. 선택 사항: NovaLibvirtCPUMode 를 사용자 정의로 설정하면 사용자 정의 하려는 인스턴스 CPU 모델을 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: <cpu_model>
```

<cpu_model> 을 호스트에서 지원하는 CPU 모델 쉼표로 구분된 목록으로 바꿉니다. 목록에 일반적인 및 덜 고급 CPU 모델을 배치하고 더 많은 기능이 풍부한 CPU 모델을 순서대로 나열하십시오(예: SandyBridge, IvyBridge,swell, Broadwell). 모델 이름 목록은 /usr/share/libvirt/cpu_map.xml 을 참조하거나 호스트 컴퓨팅 노드에 다음 명령을 입력합니다.

```
$ sudo podman exec -it nova_libvirt virsh cpu-models <arch>
```

<arch> 를 컴퓨팅 노드의 아키텍처 이름으로 바꿉니다(예: **x86_64**).

6. 지정된 **CPU** 모델을 사용하여 인스턴스의 **CPU** 기능 플래그를 구성합니다.

```
parameter_defaults:
  ComputeParameters:
    ...
    NovaLibvirtCPUModelExtraFlags: <cpu_feature_flags>
```

<cpu_feature_flags> 를 활성화 또는 비활성화할 쉼표로 구분된 기능 플래그 목록으로 바꿉니다. 각 플래그 앞에 **"+"**를 추가하여 플래그를 활성화하거나 **"-"**를 추가하여 비활성화합니다. 접두사를 지정하지 않으면 플래그가 활성화됩니다. 지정된 **CPU** 모델에 대해 사용 가능한 기능 플래그 목록은 **/usr/share/libvirt/cpu_map/*.xml** 을 참조하십시오.

다음 예제에서는 **IvyBridge** 및 **Cascadelake-Server** 모델에 대해 **CPU** 기능 플래그 **pcid** 및 **ssbd** 를 활성화하고 기능 플래그 **mtrr** 를 비활성화합니다.

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'IvyBridge','Cascadelake-Server'
    NovaLibvirtCPUModelExtraFlags: 'pcid,+ssbd,-mtrr'
```

7. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

11장. KERNELARGS를 정의하도록 수동 노드 재부팅 구성

오버클라우드 배포에 **KernelArgs** 를 처음 설정하는 작업이 포함된 경우 오버클라우드 노드가 자동으로 재부팅됩니다. 이미 프로덕션 중인 배포에 **KernelArgs** 를 추가하는 경우 기존 워크로드에 노드 재부팅이 문제가 될 수 있습니다. 배포를 업데이트할 때 노드의 자동 재부팅을 비활성화하고 대신 각 오버클라우드 배포 후 노드 재부팅을 수동으로 수행할 수 있습니다.

참고

자동 재부팅을 비활성화한 다음 배포에 새 컴퓨팅 노드를 추가하면 새 노드가 초기 프로비저닝 중에 재부팅되지 않습니다. 이로 인해 **KernelArgs** 구성이 재부팅된 후에만 적용되므로 배포 오류가 발생할 수 있습니다.

11.1. KERNELARGS를 정의하도록 수동 노드 재부팅 구성

KernelArgs 를 처음 구성할 때 노드의 자동 재부팅을 비활성화하고 대신 노드를 수동으로 재부팅할 수 있습니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. 사용자 지정 환경 파일에서 **KernelArgsDeferReboot** 역할 매개변수를 활성화합니다(예: **kernelargs_manual_reboot.yaml**):

```
parameter_defaults:
  <Role>Parameters:
    KernelArgsDeferReboot: True
```

4. 사용자 지정 환경 파일을 다른 환경 파일과 함께 스택에 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/kernelargs_manual_reboot.yaml
```

5. 컴퓨팅 노드 목록을 검색하여 재부팅할 노드의 호스트 이름을 확인합니다.

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

6. 재부팅할 컴퓨팅 노드에서 **Compute** 서비스를 비활성화하여 **Compute** 스케줄러에서 새 인스턴스를 노드에 할당하지 않도록 합니다.

```
(overcloud)$ openstack compute service set <node> nova-compute --disable
```

& lt;node >를 **Compute** 서비스를 비활성화하려는 노드의 호스트 이름으로 바꿉니다.

7. 마이그레이션하려는 컴퓨팅 노드에서 호스팅되는 인스턴스 목록을 검색합니다.

```
(overcloud)$ openstack server list --host <node_UUID> --all-projects
```

8. 인스턴스를 다른 컴퓨팅 노드로 마이그레이션 인스턴스 마이그레이션에 대한 자세한 내용은 컴퓨팅 노드 간에 가상 머신 인스턴스 마이그레이션을 참조하십시오.

9. 재부팅하려는 노드에 로그인합니다.

10. 노드를 재부팅합니다.

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

11. 노드가 부팅될 때까지 기다립니다.

12. 컴퓨팅 노드를 다시 활성화합니다.

```
(overcloud)$ openstack compute service set <node_UUID>  nova-compute --enable
```

13. 컴퓨팅 노드가 활성화되었는지 확인합니다.

```
(overcloud)$ openstack compute service list
```

12장. 인스턴스의 메모리 암호화를 제공하도록 AMD SEV 컴퓨팅 노드 구성

클라우드 관리자는 클라우드 사용자에게 메모리 암호화가 활성화된 SEV 가능 컴퓨팅 노드에서 실행되는 인스턴스를 생성하는 기능을 제공할 수 있습니다.

이 기능은 2세대 AMD EPYC™ 7002 시리즈("Rome")에서 사용할 수 있습니다.

클라우드 사용자가 메모리 암호화를 사용하는 인스턴스를 생성하도록 하려면 다음 작업을 수행해야 합니다.

1. 메모리 암호화를 위해 AMD SEV 컴퓨팅 노드를 지정합니다.
2. 메모리 암호화를 위해 컴퓨팅 노드를 구성합니다.
3. Overcloud를 배포합니다.
4. 메모리 암호화로 인스턴스를 시작하기 위한 플레이버 또는 이미지를 생성합니다.

작은 정보

AMD SEV 하드웨어가 제한된 경우 AMD SEV 컴퓨팅 노드에서 예약을 최적화하도록 호스트 집계를 구성할 수도 있습니다. AMD SEV Compute 노드에서 메모리 암호화를 요청하는 인스턴스만 예약하려면 AMD SEV 하드웨어가 있는 컴퓨팅 노드의 호스트 집계를 생성하고 호스트 집계에 메모리 암호화를 요청하는 인스턴스만 배치하도록 Compute 스케줄러를 구성합니다. 자세한 내용은 호스트 집계를 격리 하여 호스트 집계 및 필터링 생성 및 관리를 참조하십시오.

12.1. 보안 암호화 가상화 (SEV)

AMD에서 제공하는 SELinux(Secure Encrypted Virtualization)는 실행 중인 가상 머신 인스턴스가 사용 중인 DRAM의 데이터를 보호합니다. SEV는 고유한 키를 사용하여 각 인스턴스의 메모리를 암호화합니다.

SEV는 비발성 메모리 기술(NVDIMM)을 사용할 때 보안을 강화합니다. 이는 하드 드라이브와 유사하게 데이터가 없는 시스템에서 NVDIMM 칩을 물리적으로 제거할 수 있기 때문입니다. 암호화를 사용하지 않으면 민감한 데이터, 암호 또는 비밀 키와 같은 저장된 정보를 손상시킬 수 있습니다.

자세한 내용은 *AMD Secure Encrypted Virtualization(SEV)* 문서를 참조하십시오.

메모리 암호화를 사용하는 인스턴스의 제한 사항

- 실시간 마이그레이션 또는 메모리 암호화를 사용하여 인스턴스를 일시 중지 및 재개할 수 없습니다.
- PCI 통과를 사용하여 메모리 암호화가 있는 인스턴스의 장치에 직접 액세스할 수 없습니다.
- RHEL-8.1.0(Red Hat Enterprise Linux) 커널이 있는 메모리 암호화가 있는 인스턴스의 부팅 디스크로 virtio-blk 를 사용할 수 없습니다.

참고

virtio-scsi 또는 SATA 를 부팅 디스크로 사용하거나 부팅되지 않은 디스크의 경우 virtio-blk 를 사용할 수 있습니다.

- 암호화된 인스턴스에서 실행되는 운영 체제는 SEV 지원을 제공해야 합니다. 자세한 내용은 RHEL 8에서 AMD Secure Encrypted Virtualization 활성화를 참조하십시오.
- SEV를 지원하는 시스템은 암호화 키를 저장하기 위해 메모리 컨트롤러에 제한된 수의 슬롯이 있습니다. 암호화된 메모리로 실행 중인 각 인스턴스는 이러한 슬롯 중 하나를 사용합니다. 따라서 동시에 실행할 수 있는 메모리 암호화가 있는 인스턴스 수는 메모리 컨트롤러의 슬롯 수로 제한됩니다. 예를 들어 1차 AMD EPYC™ 7001 시리즈("Naples")에서는 제한이 16이고 2차 AMD EPYC™ 7002 시리즈("Rome")의 경우 제한은 255입니다.
- RAM의 메모리 암호화 고정 페이지가 있는 인스턴스. 계산 서비스는 이러한 페이지를 스왑할 수 없으므로 메모리 암호화로 인스턴스를 호스팅하는 컴퓨팅 노드에 메모리를 과다 할당할 수 없습니다.
- NUMA 노드가 여러 개 있는 인스턴스는 메모리 암호화를 사용할 수 없습니다.

12.2. 메모리 암호화를 위해 AMD SEV 컴퓨팅 노드 지정

메모리 암호화를 사용하는 인스턴스에 AMD SEV 컴퓨팅 노드를 지정하려면 새 역할 파일을 생성하여 AMD SEV 역할을 구성하고, 메모리 암호화를 위해 Compute 노드에 태그를 지정하는 데 사용할 새 오버

클라우드 플레이버 및 **AMD SEV** 리소스 클래스를 구성해야 합니다.

절차

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. **ComputeAMDSEV** 역할을 포함하는 새 역할 데이터 파일을 오버클라우드에 필요한 다른 역할과 함께 생성합니다. 다음 예제에서는 **Controller** 및 **ComputeAMDSEV** 역할이 포함된 **roles** 데이터 파일 **roles_data_amd_sev.yaml** 을 생성합니다.

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_amd_sev.yaml \
 Compute:ComputeAMDSEV Controller
```

4. **roles_data_amd_sev.yaml** 을 열고 다음 매개변수 및 섹션을 편집하거나 추가합니다.

섹션/패랜드	현재 값	새 값
역할 설명	**역할: 컴퓨팅**	**역할: ComputeAMDSEV**
역할 이름	**name: 컴퓨팅**	**name: ComputeAMDSEV**
description	기본 컴퓨팅 노드 역할	**AMD SEV Compute Node** 역할
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputeamdsev-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-amd-sev.yaml**

5. 노드 정의 템플릿 **node. json** 또는 **node. yaml**에 추가하여 오버클라우드의 **AMD SEV** 컴퓨팅 노드를 등록합니다. 자세한 내용은 ***Director 설치 및 사용*** 가이드 의 오버클라우드 노드 등록을 참조하십시오.

6. 노드 하드웨어를 검사합니다.

```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

자세한 내용은 ***Director 설치 및 사용*** 가이드 의 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

7. **AMD SEV** 컴퓨팅 노드의 **compute-amd-sev** 오버클라우드 플레이버를 생성합니다.

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> compute-amd-sev
```

- **<ram_size_mb>** 를 베어 메탈 노드의 **RAM(MB)**으로 바꿉니다.
- **<disk_size_gb>** 를 베어 메탈 노드의 디스크 크기**(GB)**로 바꿉니다.
- **<no_vcpus>** 를 베어 메탈 노드의 **CPU** 수로 바꿉니다.

참고

이러한 속성은 인스턴스를 예약하는 데 사용되지 않습니다. 그러나 계산 스케줄러는 디스크 크기를 사용하여 루트 파티션 크기를 결정합니다.

8. 노드 목록을 검색하여 **UUID**를 확인합니다.

```
(undercloud)$ openstack baremetal node list
```

9. 사용자 정의 **AMD SEV** 리소스 클래스를 사용하여 메모리 암호화를 지정할 각 베어 메탈 노드에 태그를 지정합니다.

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.AMD-SEV <node>
```

<node> 를 베어 메탈 노드의 **ID**로 바꿉니다.

10. **compute-amd-sev** 플레이버를 사용자 지정 **AMD SEV** 리소스 클래스와 연결합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_AMD_SEV=1 \
  compute-amd-sev
```

베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 각 문장 부호 표시를 밑줄로 바꾸고 접두사는 **CUSTOM_** 로 바꿉니다.

참고

플레이버는 베어 메탈 리소스 클래스의 인스턴스 하나만 요청할 수 있습니다.

11. **Compute** 스케줄러가 베어 메탈 플레이버 속성을 사용하여 인스턴스를 예약하지 못하도록 다음 플레이버 속성을 설정합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 compute-amd-sev
```

12. 선택 사항: **ComputeAMDSEV** 역할의 네트워크 토폴로지가 **Compute** 역할의 네트워크 토폴로지와 다른 경우 사용자 지정 네트워크 인터페이스 템플릿을 생성합니다. 자세한 내용은 ***Advanced Overcloud Customization*** 가이드의 **Custom network interface templates** 를 참조하십시오.

ComputeAMDSEV 역할의 네트워크 토폴로지가 **Compute** 역할과 동일한 경우 **compute.yaml** 에 정의된 기본 네트워크 토폴로지를 사용할 수 있습니다.

13. **network-environment.yaml** 파일에 **ComputeAMDSEV** 역할의 **Net::SoftwareConfig** 를 등록합니다.

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeCPUPinning::Net::SoftwareConfig: /home/stack/templates/nic-
```

```
configs/<amd_sev_net_top>.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

<amd_sev_net_top> 을 **ComputeAMDSEV** 역할의 네트워크 토폴로지가 포함된 파일 이름으로 바꿉니다(예**:** 기본 네트워크 토폴로지를 사용하려면 **compute.yaml).**

14. **node-info.yaml** 파일에 다음 매개변수를 추가하여 **AMD SEV** 컴퓨팅 노드 수와 **AMD SEV** 지정 컴퓨팅 노드에 사용하려는 플레이버를 지정합니다**.**

```
parameter_defaults:
  OvercloudComputeAMDSEVFlavor: compute-amd-sev
  ComputeAMDSEVCount: 3
```

15. 역할이 생성되었는지 확인하려면 다음 명령을 입력합니다**.**

```
(undercloud)$ openstack baremetal node list --long -c "UUID" \
 -c "Instance UUID" -c "Resource Class" -c "Provisioning State" \
 -c "Power State" -c "Last Error" -c "Fault" -c "Name" -f json
```

출력 예**:**

```
[
  {
    "Fault": null,
    "Instance UUID": "e8e60d37-d7c7-4210-acf7-f04b245582ea",
    "Last Error": null,
    "Name": "compute-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "baremetal.AMD-SEV",
    "UUID": "b5a9ac58-63a7-49ba-b4ad-33d84000ccb4"
  },
  {
    "Fault": null,
    "Instance UUID": "3ec34c0b-c4f5-4535-9bd3-8a1649d2e1bd",
    "Last Error": null,
    "Name": "compute-1",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "compute",
    "UUID": "432e7f86-8da2-44a6-9b14-dfacdf611366"
  },
  {
    "Fault": null,
    "Instance UUID": "4992c2da-adde-41b3-bef1-3a5b8e356fc0",
    "Last Error": null,
```

```
    "Name": "controller-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "controller",
    "UUID": "474c2fc8-b884-4377-b6d7-781082a3a9c0"
  }
]
```

12.3. 메모리 암호화를 위한 AMD SEV 컴퓨팅 노드 구성

클라우드 사용자가 메모리 암호화를 사용하는 인스턴스를 생성할 수 있도록 하려면 **AMD SEV** 하드웨어가 있는 컴퓨팅 노드를 구성해야 합니다.

사전 요구 사항

- 배포에 **AMD EPYC CPU**와 같이 **SEV**를 지원할 수 있는 **AMD** 하드웨어에서 실행되는 컴퓨팅 노드가 포함되어야 합니다. 다음 명령을 사용하여 배포가 **SEV-**사용 가능한지 확인할 수 있습니다.

```
$ lscpu | grep sev
```

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 선택 사항**: Compute** 환경 파일에 다음 구성을 추가하여 **AMD SEV** 컴퓨팅 노드가 동시에 호스팅할 수 있는 최대 메모리 암호화 인스턴스 수를 지정합니다.

```
parameter_defaults:
  ComputeAMDSEVExtraConfig:
    nova::config::nova_config:
      libvirt/num_memory_encrypted_guests:
        value: 15
```

참고

libvirt/num_memory_encrypted_guests 매개 변수의 기본값은 **none** 입니다. 사용자 지정 값을 설정하지 않으면 **AMD SEV Compute** 노드는 노드가 동시에 호스팅할 수 있는 메모리 암호화 인스턴스 수에 제한을 지정하지 않습니다. 대신, 하드웨어에서 **AMD SEV** 컴퓨팅 노드가 동시에 호스팅할 수 있는 최대 메모리 암호화 인스턴스 수를 결정하여 일부 메모리 암호화 인스턴스가 시작되지 않을 수 있습니다.

3. 선택 사항: 모든 **x86_64** 이미지가 기본적으로 **q35** 시스템 유형을 사용하도록 지정하려면 **Compute** 환경 파일에 다음 구성을 추가합니다.

```
parameter_defaults:
  ComputeAMDSEVParameters:
    NovaHWMachineType: x86_64=q35
```

이 매개변수 값을 지정하는 경우 모든 **AMD SEV** 인스턴스 이미지에서 **hw_machine_type** 속성을 **q35** 로 설정할 필요가 없습니다.

4. **AMD SEV Compute** 노드에서 호스트 수준 서비스가 작동할 충분한 메모리를 예약하려면 잠재적인 각 **AMD SEV** 인스턴스에 대해 **16MB**를 추가합니다.

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
    NovaReservedHostMemory: <libvirt/num_memory_encrypted_guests * 16>
```

5. **AMD SEV** 컴퓨팅 노드의 커널 매개변수를 구성합니다.

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
    KernelArgs: "hugepagesz=1GB hugepages=32 default_hugepagesz=1GB
mem_encrypt=on kvm_amd.sev=1"
```

참고

KernelArgs 매개변수를 역할 구성에 처음 추가하면 오버클라우드 노드가 자동으로 재부팅됩니다. 필요한 경우 노드 자동 재부팅을 비활성화하고 대신 각 오버클라우드 배포 후 노드를 수동으로 재부팅할 수 있습니다. 자세한 내용은 **KernelArgs**를 정의하도록 수동 노드 재부팅 구성을 참조하십시오.

6. 업데이트를 **Compute** 환경 파일에 저장합니다.

7. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

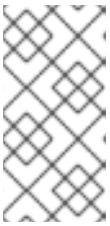
12.4. 메모리 암호화를 위한 이미지 생성

오버클라우드에 **AMD SEV** 컴퓨팅 노드가 포함된 경우 클라우드 사용자가 메모리 암호화가 있는 인스턴스를 시작하는 데 사용할 수 있는 **AMD SEV** 인스턴스 이미지를 생성할 수 있습니다.

절차

1. 메모리 암호화를 위한 새 이미지를 생성합니다.

```
(overcloud)$ openstack image create ...  \
 --property hw_firmware_type=uefi amd-sev-image
```



참고

기존 이미지를 사용하는 경우 이미지에 **hw_firmware_type** 속성이 **uefi** 로 설정되어 있어야 합니다.

2. 선택 사항: 이미지에 **hw_mem_encryption=True** 속성을 추가하여 이미지에서 **AMD SEV** 메모리 암호화를 활성화합니다.

```
(overcloud)$ openstack image set  \
 --property hw_mem_encryption=True amd-sev-image
```

작은 정보

플레이버에서 메모리 암호화를 활성화할 수 있습니다. 자세한 내용은 메모리 암호화에 대한 플레이버 생성 을 참조하십시오.

3. 선택 사항: 컴퓨팅 노드 구성에 아직 설정되지 않은 경우 머신 유형을 **q35** 로 설정합니다.

```
(overcloud)$ openstack image set  \
 --property hw_machine_type=q35 amd-sev-image
```

4. 선택 사항: **SEV** 가능 호스트 집계에서 메모리 암호화 인스턴스를 예약하려면 이미지 추가 사

양에 다음 특성을 추가합니다.

```
(overcloud)$ openstack image set  \
 --property trait:HW_CPU_X86_AMD_SEV=required amd-sev-image
```

작은 정보

플레이버에서 이 특성을 지정할 수도 있습니다. 자세한 내용은 메모리 암호화에 대한 플레이버 생성 을 참조하십시오.

12.5. 메모리 암호화를 위한 플레이버 생성

오버클라우드에 **AMD SEV** 컴퓨팅 노드가 포함된 경우 클라우드 사용자가 메모리 암호화가 있는 인스턴스를 시작하는 데 사용할 수 있는 **AMD SEV** 플레이버를 하나 이상 생성할 수 있습니다.

참고

AMD SEV 플레이버는 이미지에 **hw_mem_encryption** 속성이 설정되지 않은 경우에만 필요합니다.

절차

1. 메모리 암호화를 위한 플레이버를 생성합니다.

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2  \
 --property hw:mem_encryption=True m1.small-amd-sev
```

2. **SEV** 가능 호스트 집계에서 메모리 암호화 인스턴스를 예약하려면 플레이버 추가 사양에 다음 특성을 추가합니다.

```
(overcloud)$ openstack flavor set  \
 --property trait:HW_CPU_X86_AMD_SEV=required m1.small-amd-sev
```

12.6. 메모리 암호화를 사용하여 인스턴스 시작

메모리 암호화가 활성화된 **AMD SEV** 컴퓨팅 노드에서 인스턴스를 시작하려면 메모리 암호화 플레이버 또는 이미지를 사용하여 인스턴스를 생성합니다.

절차

1. **AMD SEV 플레이버 또는 이미지를 사용하여 인스턴스를 생성합니다. 다음 예제에서는 메모리 암호화에 플레이버를 만들고 메모리 암호화를 위해 이미지 생성에 생성된 플레이버 를 사용하여 인스턴스를 생성합니다.**

```
(overcloud)$ openstack server create --flavor m1.small-amd-sev \
 --image amd-sev-image amd-sev-instance
```

2. **클라우드 사용자로 인스턴스에 로그인합니다.**

3. **인스턴스에서 메모리 암호화를 사용하는지 확인하려면 인스턴스에서 다음 명령을 입력합니다.**

```
$ dmesg | grep -i sev
AMD Secure Encrypted Virtualization (SEV) active
```

13장. 인스턴스에 영구 메모리를 제공하도록 NVDIMM 컴퓨팅 노드 구성

비발성 듀얼 인라인 메모리 모듈(NVDIMM)은 DRAM에 PMEM(영구 메모리)을 제공하는 기술입니다. 표준 컴퓨터 메모리는 전기 전력 손실 후 데이터를 손실합니다. NVDIMM은 전기 전력 손실 후에도 데이터를 유지 관리합니다. PMEM을 사용하는 인스턴스는 전원 주기 동안 애플리케이션 데이터를 유지하는 대규모 연속 메모리 세그먼트를 로드할 수 있는 기능을 애플리케이션에 제공할 수 있습니다. 이 기능은 대량의 메모리를 요청하는 HPC(고성능 컴퓨팅)에 유용합니다.

클라우드 관리자는 NVDIMM 하드웨어가 있는 컴퓨팅 노드에서 PMEM 네임스페이스를 생성하고 구성하여 가상 PMEM(vPMEM)으로 인스턴스에서 PMEM을 사용할 수 있습니다. 그런 다음 클라우드 사용자는 종료 후 인스턴스 콘텐츠를 유지해야 할 때 vPMEM을 요청하는 인스턴스를 생성할 수 있습니다.

클라우드 사용자가 PMEM을 사용하는 인스턴스를 생성할 수 있도록 하려면 다음 절차를 완료해야 합니다.

1. PMEM을 위한 컴퓨팅 노드 지정.
2. NVDIMM 하드웨어가 있는 PMEM의 컴퓨팅 노드를 구성합니다.
3. Overcloud를 배포합니다.
4. vPMEM이 있는 인스턴스를 시작하기 위한 PMEM 플레이버를 만듭니다.

작은 정보

NVDIMM 하드웨어가 제한된 경우 PMEM 컴퓨팅 노드에 예약을 최적화하도록 호스트 집계를 구성할 수도 있습니다. PMEM 계산 노드에서 vPMEM을 요청하는 인스턴스만 예약하려면 NVDIMM 하드웨어가 있는 컴퓨팅 노드의 호스트 집계를 생성하고 호스트 집계에 PMEM 인스턴스만 배치하도록 Compute 스케줄러를 구성합니다. 자세한 내용은 호스트 집계를 격리 하여 호스트 집계 및 필터링 생성 및 관리를 참조하십시오.

사전 요구 사항

- 컴퓨팅 노드에는 Intel® Optane™ DC Persistent Memory와 같은 영구 메모리 하드웨어가 있습니다.
-

PMEM 하드웨어 장치에서 backend NVDIMM 리전에 PMEM 네임스페이스를 생성하도록 구성했습니다. Intel에서 제공하는 ipmctl 도구를 사용하여 PMEM 하드웨어를 구성할 수 있습니다.

PMEM 장치 사용 시 제한 사항

- vPMEM을 사용하는 인스턴스를 콜드 마이그레이션, 실시간 마이그레이션, 크기 조정 또는 일시 중단할 수 없습니다.
- RHEL8이 실행되는 인스턴스만 vPMEM을 사용할 수 있습니다.
- vPMEM 인스턴스를 다시 빌드할 때 인스턴스의 초기 상태를 복원하도록 영구 메모리 네임스페이스가 제거됩니다.
- 새 플레이버를 사용하여 인스턴스 크기 조정 시 원래 가상 영구 메모리의 콘텐츠가 새 가상 영구 메모리에 복사되지 않습니다.
- 가상 영구 메모리 핫플러그는 지원되지 않습니다.
- vPMEM 인스턴스의 스냅샷을 생성할 때 가상 영구 이미지가 포함되지 않습니다.

13.1. PMEM을 위한 컴퓨팅 노드 지정

PMEM 워크로드에 대해 컴퓨팅 노드를 지정하려면 새 역할 파일을 생성하여 PMEM 역할을 구성하고, PMEM의 새 오버클라우드 플레이버 및 리소스 클래스를 구성하여 NVDIMM 컴퓨팅 노드를 태그해야 합니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.
2. stackrc 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. Controller,Compute 및 Compute PMEM 역할이 포함된 roles_data_pmem.yaml 이라는 새

역할 데이터 파일을 생성합니다.

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_pmem.yaml \
 Compute:ComputePMEM Compute Controller
```

4. **roles_data_pmem.yaml** 을 열고 다음 매개변수 및 섹션을 편집하거나 추가합니다.

섹션/패랜드	현재 값	새 값
역할 설명	**역할: 컴퓨팅**	**역할: ComputePMEM**
역할 이름	**name: 컴퓨팅**	**name: ComputePMEM**
description	기본 컴퓨팅 노드 역할	**PMEM** 계산 노드 역할
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepmem-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-pmem.yaml**

5. 노드 정의 템플릿 **node. json** 또는 **node. yaml**에 추가하여 오버클라우드에 **NVDIMM** 컴퓨팅 노드를 등록합니다. 자세한 내용은 ***Director** 설치 및 사용* 가이드 의 오버클라우드 노드 등록을 참조하십시오.

6. 노드 하드웨어를 검사합니다.

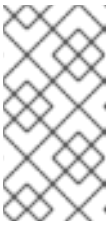
```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

자세한 내용은 ***Director** 설치 및 사용* 가이드 의 베어 메탈 노드 하드웨어 인벤토리 생성을 참조하십시오.

7. **PMEM** 컴퓨팅 노드의 **compute-pmem** 오버클라우드 플레이버를 생성합니다.

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> compute-pmem
```

- **<ram_size_mb>** 를 베어 메탈 노드의 **RAM(MB)**으로 바꿉니다.
- **<disk_size_gb>** 를 베어 메탈 노드의 디스크 크기**(GB)**로 바꿉니다.
- **<no_vcpus>** 를 베어 메탈 노드의 **CPU** 수로 바꿉니다.



참고

이러한 속성은 인스턴스를 예약하는 데 사용되지 않습니다. 그러나 계산 스케줄러는 디스크 크기를 사용하여 루트 파티션 크기를 결정합니다.

8. 사용자 정의 **PMEM** 리소스 클래스를 사용하여 **PMEM** 워크로드를 지정하려는 각 베어 메탈 노드에 태그를 지정합니다.

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.PMEM <node>
```

<node> 를 베어 메탈 노드의 **ID**로 바꿉니다.

9. **compute-pmem** 플레이버를 사용자 지정 **PMEM** 리소스 클래스와 연결합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_PMEM=1 \
  compute-pmem
```

베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 모든 문장 부호를 밑줄로 바꾸고 접두사는 **CUSTOM_** 로 바꿉니다.

참고

플레이버는 베어 메탈 리소스 클래스의 인스턴스 하나만 요청할 수 있습니다.

10. **Compute** 스케줄러가 베어 메탈 플레이버 속성을 사용하여 인스턴스를 예약하지 못하도록 다음 플레이버 속성을 설정합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 compute-pmem
```

11. 다음 매개변수를 **node-info.yaml** 파일에 추가하여 **PMEM** 컴퓨팅 노드 수와 **PMEM** 지정 컴퓨팅 노드에 사용할 플레이버를 지정합니다.

```
parameter_defaults:
  OvercloudComputePMEMFlavor: compute-pmem
  ComputePMEMCount: 3 #set to the no of NVDIMM devices you have
```

12. 역할이 생성되었는지 확인하려면 다음 명령을 입력합니다.

```
(undercloud)$ openstack overcloud profiles list
```

13.2. PMEM 컴퓨팅 노드 구성

클라우드 사용자가 **vPMEM**을 사용하는 인스턴스를 생성할 수 있도록 하려면 **NVDIMM** 하드웨어가 있는 컴퓨팅 노드를 구성해야 합니다.

절차

1. **NVDIMM** 컴퓨팅 노드를 구성하는 새 **Compute** 환경 파일을 만듭니다(예: **env_pmem.yaml**).

2. **NVDIMM** 지역을 인스턴스에서 사용할 수 있는 **PMEM** 네임스페이스로 분할하려면 **Compute** 환경 파일의 **PMEM** 역할에 **NovaPMEMNamespaces** 역할별 매개 변수를 추가하고 다음 형식을 사용하여 값을 설정합니다.

```
<size>:<namespace_name>[,<size>:<namespace_name>]
```

다음 접미사를 사용하여 크기를 나타냅니다.

- **KiB**의 경우 **"K"** 또는 **"K"**
- **MiB**의 경우 **"m"** 또는 **"M"**

- **GiB**의 경우 **"G"** 또는 **"G"**
- **TiB**의 경우 **"t"** 또는 **"T"**

예를 들어 다음 구성에서는 크기가 **6GiB**이고 크기가 **100GiB**인 네 개의 네임스페이스를 생성합니다.

```
parameter_defaults:
  ComputePMEMParameters:
    NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
```

3. 플레이버에서 사용할 수 있는 레이블에 **PMEM** 네임스페이스를 매핑하려면 **Compute** 환경 파일의 **PMEM** 역할에 **NovaPMEMMappings** 역할 특정 매개 변수를 추가하고 다음 형식을 사용하여 값을 설정합니다.

```
<label>:<namespace_name>[|<namespace_name>][,<label>:<namespace_name>[|
<namespace_name>]].
```

예를 들어 다음 구성은 **3**개의 **6GiB** 네임스페이스를 **"6GB"** 레이블에 매핑하고 **100GiB** 네임스페이스를 **"LARGE"** 레이블에 매핑합니다.

```
parameter_defaults:
  ComputePMEMParameters:
    NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
    NovaPMEMMappings: "6GB:ns0|ns1|ns2,LARGE:ns3"
```

4. 업데이트를 **Compute** 환경 파일에 저장합니다.

5. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -r /home/stack/templates/roles_data_pmem.yaml \
 -e /home/stack/templates/node-info.yaml \
 -e [your environment files] \
 -e /home/stack/templates/env_pmem.yaml
```

6. 클라우드 사용자가 **vPMEM**이 있는 인스턴스를 시작하는 데 사용할 수 있는 플레이버를 만들고 구성합니다. 다음 예제에서는 **3**단계에 매핑된 대로 작은 **PMEM** 장치 **6GB**를 요청하는 플레이

버를 만듭니다.

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2  \
 --property hw:pmem='6GB' small_pmem_flavor
```

검증

1. PMEM 플레이버 중 하나를 사용하여 인스턴스를 생성합니다.

```
(overcloud)$ openstack flavor list
(overcloud)$ openstack server create --flavor small_pmem_flavor \
 --image rhel8 pmem_instance
```

2. 클라우드 사용자로 인스턴스에 로그인합니다. 자세한 내용은 인스턴스에 연결을 참조하십시오.

3. 인스턴스에 연결된 모든 디스크 장치를 나열합니다.

```
$ sudo fdisk -l /dev/pmem0
```

나열된 장치 중 하나가 NVDIMM인 경우 인스턴스에 vPMEM이 있습니다.

14장. 인스턴스의 가상 GPU 구성

인스턴스에서 GPU 기반 렌더링을 지원하려면 사용 가능한 물리 GPU 장치 및 하이퍼바이저 유형에 따라 가상 GPU(vGPU) 리소스를 정의하고 관리할 수 있습니다. 이 구성을 사용하여 모든 물리적 GPU 장치 간에 렌더링 워크로드를 보다 효율적으로 분할하고 vGPU 지원 인스턴스 예약을 보다 효과적으로 제어할 수 있습니다.

Compute(nova) 서비스에서 vGPU를 활성화하려면 클라우드 사용자가 vGPU 장치가 있는 RHEL(Red Hat Enterprise Linux) 인스턴스를 만드는 데 사용할 수 있는 플레이버를 만듭니다. 그러면 각 인스턴스에서 물리 GPU 장치에 해당하는 가상 GPU 장치가 있는 GPU 워크로드를 지원할 수 있습니다.

계산 서비스는 각 호스트에서 정의한 각 GPU 프로필에 사용할 수 있는 vGPU 장치 수를 추적합니다. 계산 서비스는 플레이버를 기반으로 이러한 호스트에 인스턴스를 예약하고, 장치를 연결하며, 사용량을 지속적으로 모니터링합니다. 인스턴스가 삭제되면 계산 서비스는 vGPU 장치를 다시 사용 가능한 풀에 추가합니다.

중요

Red Hat은 지원 예외에 대한 요구 사항 없이 RHOSP에서 NVIDIA vGPU를 사용할 수 있습니다. 그러나 Red Hat은 NVIDIA vGPU 드라이버에 대한 기술 지원을 제공하지 않습니다. NVIDIA vGPU 드라이버는 NVIDIA에서 제공하고 지원합니다. NVIDIA vGPU 소프트웨어에 대한 NVIDIA Enterprise 지원을 받으려면 NVIDIA Certified Support Services 서브스크립션이 필요합니다. 지원되는 구성 요소에서 문제를 재현할 수 없는 NVIDIA vGPU를 사용하면 다음과 같은 지원 정책이 적용됩니다.

- Red Hat이 이 문제와 관련하여 타사 구성 요소와 관련이 없는 경우 일반 지원 범위 및 Red Hat SLA 가 적용됩니다.
- Red Hat은 타사 구성 요소와 관련된 것으로 의심되는 경우 고객은 Red Hat의 타사 지원 및 인증 정책에 따라 NVIDIA로 이동합니다. 자세한 내용은 Obtaining Support from NVIDIA 에서 참조하십시오.

14.1. 지원되는 구성 및 제한 사항

지원되는 GPU 카드

지원되는 NVIDIA GPU 카드 목록은 NVIDIA 웹 사이트의 Virtual GPU 소프트웨어 지원 제품을 참조하십시오.

vGPU 장치 사용 시 제한 사항

- 각 컴퓨팅 노드에서 하나의 **vGPU** 유형만 활성화할 수 있습니다.
- 각 인스턴스는 하나의 **vGPU** 리소스만 사용할 수 있습니다.
- 호스트 간 **vGPU** 인스턴스의 실시간 마이그레이션은 지원되지 않습니다.
- **vGPU** 인스턴스 비우기가 지원되지 않습니다.
- **vGPU** 인스턴스를 호스팅하는 컴퓨팅 노드를 재부팅해야 하는 경우 **vGPU**는 재생성된 인스턴스에 자동으로 다시 할당되지 않습니다. 컴퓨팅 노드를 재부팅하기 전에 인스턴스를 콜드 마이그레이션하거나 재부팅 후 각 **vGPU**를 올바른 인스턴스에 수동으로 할당해야 합니다. 각 **vGPU**를 수동으로 할당하려면 재부팅하기 전에 컴퓨팅 노드에서 실행되는 각 **vGPU** 인스턴스의 **XML**에서 **mdev UUID**를 검색해야 합니다. 다음 명령을 사용하여 각 인스턴스의 **mdev UUID**를 검색할 수 있습니다.

```
# virsh dumpxml <instance_name> | grep mdev
```

  <instance_name>을 **Compute API**에 **/servers** 요청에 반환된 **libvirt** 인스턴스 이름인 **OS-EXT-SRV-ATTR:instance_name**으로 바꿉니다.

- **libvirt** 제한으로 인해 **vGPU** 지원 인스턴스에서 일시 중지 작업이 지원되지 않습니다. 대신 인스턴스를 스냅샷하거나 보류할 수 있습니다.
- 기본적으로 컴퓨팅 호스트의 **vGPU** 유형은 **API** 사용자에게 노출되지 않습니다. 액세스 권한을 부여하려면 호스트를 호스트 집계에 추가합니다. 자세한 내용은 호스트 집계 생성 및 관리를 참조하십시오.
- **NVIDIA** 액셀러레이터 하드웨어를 사용하는 경우 **NVIDIA** 라이센스 요구 사항을 준수해야 합니다. 예를 들어 **NVIDIA vGPU GRID**에는 라이센스 서버가 필요합니다. **NVIDIA** 라이센스 요구 사항에 대한 자세한 내용은 **NVIDIA** 웹 사이트의 **NVIDIA** 라이센스 서버 릴리스 정보를 참조하십시오.

14.2. 컴퓨팅 노드에 VGPU 구성

클라우드 사용자가 **vGPU**(가상 **GPU**)를 사용하는 인스턴스를 생성할 수 있도록 하려면 물리적 **GPU**가 있는 컴퓨팅 노드를 구성해야 합니다.

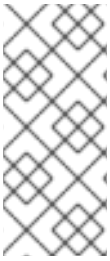
1. vGPU에 대해 컴퓨팅 노드를 지정하기 위해 GPU 역할, 프로필 및 플레이버를 준비합니다.

2. vGPU의 컴퓨팅 노드를 구성합니다.

3. Overcloud를 배포합니다.

작은 정보

GPU 하드웨어가 제한된 경우 vGPU 컴퓨팅 노드에 예약을 최적화하도록 호스트 집계를 구성할 수도 있습니다. vGPU 컴퓨팅 노드에서 vGPU를 요청하는 인스턴스만 예약하려면 vGPU 컴퓨팅 노드의 호스트 집계를 생성하고 호스트 집계에 vGPU 인스턴스만 배치하도록 Compute 스케줄러를 구성합니다. 자세한 내용은 호스트 집계를 격리 하여 호스트 집계 및 필터링 생성 및 관리를 참조하십시오.



참고

NVIDIA GRID vGPU를 사용하려면 NVIDIA GRID 라이센스 요구 사항을 준수해야 하며 자체 호스팅 라이센스 서버의 URL이 있어야 합니다. 자세한 내용은 NVIDIA 라이센스 서버 릴리스 정보 웹 페이지를 참조하십시오.

14.2.1. 사전 요구 사항

- NVIDIA 웹사이트에서 GPU 장치에 해당하는 NVIDIA GRID 호스트 드라이버 RPM 패키지를 다운로드했습니다. 필요한 드라이버를 확인하려면 NVIDIA 드라이버 다운로드 포털 을 참조하십시오. 포털에서 드라이버를 다운로드하려면 등록된 NVIDIA 고객이어야 합니다.

- NVIDIA GRID 호스트 드라이버가 설치된 사용자 지정 오버클라우드 이미지를 빌드했습니다. 사용자 지정 오버클라우드 이미지 빌드에 대한 자세한 내용은 오버클라우드 이미지 작업을 참조하십시오.

14.2.2. vGPU용 컴퓨팅 노드 지정

vGPU 워크로드에 대해 컴퓨팅 노드를 지정하려면 새 역할 파일을 생성하여 vGPU 역할을 구성하고 GPU 사용 컴퓨팅 노드의 태그를 지정하는 데 사용할 새 오버클라우드 플레이버 및 리소스 클래스를 구성해야 합니다.

절차

1.

1. **stack** 사용자로 언더클라우드에 로그인합니다.

2. **stackrc** 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. **Controller,Compute** 및 **Compute Gpu** 역할을 포함하는 **roles_data_gpu.yaml** 이라는 새 역할 데이터 파일을 생성합니다.

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_gpu.yaml \
 Compute:ComputeGpu Compute Controller
```

4. **roles_data_gpu.yaml** 을 열고 다음 매개변수 및 섹션을 편집하거나 추가합니다.

섹션/패랜드	현재 값	새 값
역할 설명	**역할: 컴퓨팅**	**역할: ComputeGpu**
역할 이름	**name: 컴퓨팅**	**name: ComputeGpu**
description	기본 컴퓨팅 노드 역할	**GPU** 컴퓨팅 노드 역할
ImageDefault	해당 없음	**overcloud-full-gpu**
HostnameFormatDefault	**-compute-**	**-computegpu-**
deprecated_nic_config_name	**compute.yaml**	**compute-gpu.yaml**

5. 노드 정의 템플릿 **node. json** 또는 **node. yaml**에 추가하여 오버클라우드의 **GPU** 지원 컴퓨팅 노드를 등록합니다. 자세한 내용은 ***Director** 설치 및 사용* 가이드 의 오버클라우드 노드 등록을 참조하십시오.

6. 노드 하드웨어를 검사합니다.

```
(undercloud)$ openstack overcloud node introspect --all-manageable \
 --provide
```

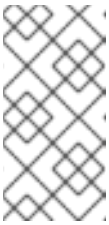
자세한 내용은 ***Director** 설치 및 사용* 가이드 의 베어 메탈 노드 하드웨어 인벤토리 생성을 참

조하십시오.

7. **vGPU** 컴퓨팅 노드의 **compute-vgpu-nvidia** 오버클라우드 플레이버를 생성합니다.

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> compute-vgpu-nvidia
```

- **<ram_size_mb>** 를 베어 메탈 노드의 **RAM(MB)**으로 바꿉니다.
- **<disk_size_gb>** 를 베어 메탈 노드의 디스크 크기**(GB)**로 바꿉니다.
- **<no_vcpus>** 를 베어 메탈 노드의 **CPU** 수로 바꿉니다.



참고

이러한 속성은 인스턴스를 예약하는 데 사용되지 않습니다. 그러나 계산 스케줄러는 디스크 크기를 사용하여 루트 파티션 크기를 결정합니다.

8. **GPU** 워크로드를 위해 사용자 정의 **GPU** 리소스 클래스를 지정하려는 각 베어 메탈 노드에 태그를 지정합니다.

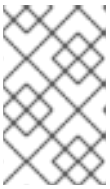
```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.GPU <node>
```

<node> 를 **baremetal** 노드의 **ID**로 바꿉니다.

9. **compute-vgpu-nvidia** 플레이버를 사용자 지정 **GPU** 리소스 클래스와 연결합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_GPU=1 \
 compute-vgpu-nvidia
```

베어 메탈 서비스 노드의 리소스 클래스에 해당하는 사용자 지정 리소스 클래스의 이름을 확인하려면 리소스 클래스를 대문자로 변환하려면 모든 문장 부호를 밑줄로 바꾸고 접두사는 **CUSTOM_** 로 바꿉니다.



참고

플레이버는 베어 메탈 리소스 클래스의 인스턴스 하나만 요청할 수 있습니다.

10. **Compute** 스케줄러가 베어 메탈 플레이버 속성을 사용하여 인스턴스를 예약하지 못하도록 다음 플레이버 속성을 설정합니다.

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 compute-vgpu-nvidia
```

11. 역할이 생성되었는지 확인하려면 다음 명령을 입력합니다.

```
(undercloud)$ openstack overcloud profiles list
```

14.2.3. vGPU의 컴퓨팅 노드 구성 및 오버클라우드 배포

사용자 환경의 물리적 **GPU** 장치에 해당하는 **vGPU** 유형을 검색 및 할당하고, **vGPU**의 컴퓨팅 노드를 구성하도록 환경 파일을 준비해야 합니다.

절차

1. 임시 컴퓨팅 노드에 **Red Hat Enterprise Linux** 및 **NVIDIA GRID** 드라이버를 설치하고 노드를 시작합니다.

2. 컴퓨팅 노드에서 활성화하려는 물리 **GPU** 장치의 **vGPU** 유형을 찾습니다. **libvirt**의 경우 가상 **GPU**는 조정된 장치 또는 **mdev** 유형 장치입니다. 지원되는 **mdev** 장치를 검색하려면 다음 명령을 입력합니다.

```
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/
nvidia-11  nvidia-12  nvidia-13  nvidia-14  nvidia-15  nvidia-16  nvidia-17  nvidia-18  nvidia-19
nvidia-20  nvidia-21  nvidia-210  nvidia-22

[root@overcloud-computegpu-0 ~]# cat
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/nvidia-18/description
num_heads=4, frl_config=60, framebuffer=2048M, max_resolution=4096x2160,
max_instance=4
```

3. **network-environment.yaml** 파일에 **ComputeGpu** 역할의 **Net::SoftwareConfig** 를 등록합니다.

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeGpu::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute-gpu.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

4. 다음 매개변수를 **node-info.yaml** 파일에 추가하여 **GPU** 컴퓨팅 노드 수와 **GPU** 지정 컴퓨팅 노드에 사용할 플레이버를 지정합니다.

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudComputeGpuFlavor: compute-vgpu-nvidia
  ControllerCount: 1
  ComputeCount: 0
  ComputeGpuCount: 1
```

5. **gpu.yaml** 파일을 생성하여 **GPU** 장치의 **vGPU** 유형을 지정합니다.

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-18
```

참고

각 물리 **GPU**는 하나의 가상 **GPU** 유형만 지원합니다. 이 속성에 **vGPU** 유형을 여러 개 지정하는 경우 첫 번째 유형만 사용됩니다.

6. 업데이트를 **Compute** 환경 파일에 저장합니다.

7. 다른 환경 파일을 사용하여 스택에 새 역할 및 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -r /home/stack/templates/roles_data_gpu.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/gpu.yaml \
  -e /home/stack/templates/node-info.yaml
```

14.3. 사용자 정의 GPU 인스턴스 이미지 생성

클라우드 사용자가 vGPU(가상 GPU)를 사용하는 인스턴스를 생성할 수 있도록 하려면 인스턴스 시작을 위한 사용자 지정 vGPU 지원 이미지를 만들 수 있습니다. 다음 절차에 따라 NVIDIA GRID 게스트 드라이버 및 라이센스 파일을 사용하여 vGPU 지원 인스턴스 이미지를 생성합니다.

사전 요구 사항

- GPU 사용 컴퓨팅 노드를 사용하여 오버클라우드를 구성하고 배포했습니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.

2. overcloudrc 인증 정보 파일을 소싱합니다.

```
$ source ~/overcloudrc
```

3. vGPU 인스턴스에 필요한 하드웨어 및 소프트웨어 프로필을 사용하여 인스턴스를 생성합니다.

```
(overcloud)$ openstack server create --flavor <flavor> \
 --image <image> temp_vgpu_instance
```

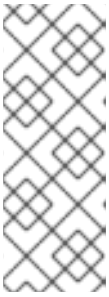
   - <flavor> 를 vGPU 인스턴스에 필요한 하드웨어 프로필이 있는 플레이버의 이름 또는 ID로 바꿉니다. vGPU 플레이버 생성에 대한 자세한 내용은 인스턴스의 vGPU 플레이버 생성을 참조하십시오.

   - <image> 를 vGPU 인스턴스에 필요한 소프트웨어 프로필이 있는 이미지의 이름 또는 ID로 바꿉니다. RHEL 클라우드 이미지를 다운로드하는 방법에 대한 자세한 내용은 이미지 서비스를 참조하십시오.

4. cloud-user로 인스턴스에 로그인합니다.

5. NVIDIA 지침에 따라 인스턴스에 gridd.conf NVIDIA GRID 라이센스 파일을 만듭니다. 구성 파일을 사용하여 Linux에서 NVIDIA vGPU 라이센스.

6. 인스턴스에 GPU 드라이버를 설치합니다. NVIDIA 드라이버 설치에 대한 자세한 내용은 Linux에 NVIDIA vGPU 소프트웨어 그래픽 드라이버 설치를 참조하십시오.



참고

hw_video_model 이미지 속성을 사용하여 GPU 드라이버 유형을 정의합니다. vGPU 인스턴스에 대해 에뮬레이트된 GPU를 비활성화하려면 none 을 선택할 수 있습니다. 지원되는 드라이버에 대한 자세한 내용은 이미지 메타데이터 를 참조하십시오.

7. 인스턴스의 이미지 스냅샷을 생성합니다.

```
(overcloud)$ openstack server image create \
 --name vgpu_image temp_vgpu_instance
```

8. 선택 사항: 인스턴스를 삭제합니다.

14.4. 인스턴스의 VGPU 플레이버 생성

클라우드 사용자가 GPU 워크로드용 인스턴스를 생성할 수 있도록 하려면 vGPU 인스턴스를 시작하는 데 사용할 수 있는 GPU 플레이버를 생성하고 해당 플레이버에 vGPU 리소스를 할당할 수 있습니다.

사전 요구 사항

- GPU 지정 컴퓨팅 노드를 사용하여 오버클라우드를 구성하고 배포했습니다.

절차

1. NVIDIA GPU 플레이버를 생성합니다. 예를 들면 다음과 같습니다.

```
(overcloud)$ openstack flavor create --vcpus 6 \
 --ram 8192 --disk 100 m1.small-gpu
+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
| disk                       | 100                                 |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                        |
```

```
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 8192                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 6                                    |
+----------------------------+--------------------------------------+
```

2. **vGPU 리소스를 생성한 플레이버에 할당합니다. 각 인스턴스에 대해 하나의 vGPU만 할당할 수 있습니다.**

```
(overcloud)$ openstack flavor set m1.small-gpu \
 --property "resources:VGPU=1"

(overcloud)$ openstack flavor show m1.small-gpu
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| access_project_ids         | None                                 |
| disk                       | 100                                  |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                         |
| os-flavor-access:is_public | True                                 |
| properties                 | resources:VGPU='1'                   |
| ram                        | 8192                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 6                                    |
+----------------------------+--------------------------------------+
```

14.5. VGPU 인스턴스 시작

GPU 워크로드에 사용할 GPU 지원 인스턴스를 생성할 수 있습니다.

절차

1. **GPU 플레이버 및 이미지를 사용하여 인스턴스를 생성합니다. 예를 들면 다음과 같습니다.**

```
(overcloud)$ openstack server create --flavor m1.small-gpu \
 --image vgpu_image --security-group web --nic net-id=internal0 \
 --key-name lambda vgpu-instance
```

2. **cloud-user로 인스턴스에 로그인합니다.**

3. 인스턴스에서 GPU에 액세스할 수 있는지 확인하려면 인스턴스에서 다음 명령을 입력합니다.

```
$ lspci -nn | grep <gpu_name>
```

14.6. GPU 장치에 PCI 패스스루 활성화

PCI 통과를 사용하여 그래픽 카드와 같은 물리적 PCI 장치를 인스턴스에 연결할 수 있습니다. 장치에 PCI 패스스루를 사용하는 경우 인스턴스는 작업을 수행하기 위해 장치에 대한 전용 액세스를 예약하고 호스트에서 장치를 사용할 수 없습니다.

사전 요구 사항

- pciutils 패키지는 PCI 카드가 있는 물리적 서버에 설치됩니다.
- GPU 장치의 드라이버는 장치가 전달되는 인스턴스에 설치해야 합니다. 따라서 필수 GPU 드라이버가 설치된 사용자 정의 인스턴스 이미지를 생성해야 합니다. GPU 드라이버를 설치한 사용자 정의 인스턴스 이미지를 생성하는 방법에 대한 자세한 내용은 Creating a custom GPU 인스턴스 이미지 생성을 참조하십시오.

절차

1. 각 패스스루 장치 유형의 벤더 ID와 제품 ID를 확인하려면 PCI 카드가 있는 물리적 서버에 다음 명령을 입력합니다.

```
# lspci -nn | grep -i <gpu_name>
```

예를 들어 NVIDIA GPU의 벤더 및 제품 ID를 확인하려면 다음 명령을 입력합니다.

```
# lspci -nn | grep -i nvidia
3b:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1eb8] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1db4] (rev a1)
```

2. 각 PCI 장치에 SR-IOV(Single Root I/O Virtualization) 기능이 있는지 확인하려면 PCI 카드가 있는 물리적 서버에 다음 명령을 입력합니다.

```
# lspci -v -s 3b:00.0
3b:00.0 3D controller: NVIDIA Corporation TU104GL [Tesla T4] (rev a1)
        ...
        Capabilities: [bcc] Single Root I/O Virtualization (SR-IOV)
        ...
```

-

3. **PCI 패스스루를 위해 오버클라우드에서 컨트롤러 노드를 구성하려면 환경 파일(예: pci_passthru_controller.yaml)을 만듭니다.**

4. **pci_passthru_controller.yaml 의 NovaSchedulerDefaultFilters 매개변수에 PciPassthrough Filter를 추가합니다.**

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','Serve
rGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```

5. **컨트롤러 노드에서 장치의 PCI 별칭을 지정하려면 pci_passthru_controller.yaml에 다음 구성을 추가합니다.**

- **PCI 장치에 SR-IOV 기능이 있는 경우:**

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

- **PCI 장치에 SR-IOV 기능이 없는 경우:**

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```

device_type 필드 구성에 대한 자세한 내용은 PCI 패스스루 장치 유형 필드를 참조하십시오.

참고

nova-api 서비스가 컨트롤러 이외의 역할에서 실행 중인 경우 **<Role> ExtraConfig** 형식으로 **Controller ExtraConfig** 를 사용자 역할로 바꿉니다.

6. **PCI** 패스스루에 대해 오버클라우드에서 **Compute** 노드를 구성하려면 환경 파일(예: **pci_passthru_compute.yaml**)을 생성합니다.

7. **Compute** 노드에서 장치에 사용 가능한 **PCI**를 지정하려면 **pci_passthru_compute.yaml**에 다음을 추가합니다.

```
parameter_defaults:
  NovaPCIPassthrough:
    - vendor_id: "10de"
      product_id: "1eb8"
```

8. 인스턴스 마이그레이션 및 크기 조정 작업을 위해 컴퓨팅 노드에 **PCI** 별칭 사본을 생성해야 합니다. 컴퓨팅 노드에서 장치의 **PCI** 별칭을 지정하려면 **pci_passthru_compute.yaml**에 다음을 추가합니다.

- **PCI** 장치에 **SR-IOV** 기능이 있는 경우:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

- **PCI** 장치에 **SR-IOV** 기능이 없는 경우:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```

참고

컴퓨팅 노드 별칭은 컨트롤러 노드의 별칭과 동일해야 합니다.

9. **Compute** 노드의 서버 **BIOS**에서 **IOMMU**를 활성화하여 **PCI** 패스스루를 지원하려면 **KernelArgs** 매개변수를 **pci_passthru_compute.yaml**에 추가합니다.

```
parameter_defaults:
  ...
  ComputeParameters:
    KernelArgs: "intel_iommu=on iommu=pt"
```

참고

KernelArgs 매개변수를 역할 구성에 처음 추가하면 오버클라우드 노드가 자동으로 재부팅됩니다. 필요한 경우 노드 자동 재부팅을 비활성화하고 대신 각 오버클라우드 배포 후 노드를 수동으로 재부팅할 수 있습니다. 자세한 내용은 **KernelArgs**를 정의하도록 수동 노드 재부팅 구성을 참조하십시오.

10. 다른 환경 파일을 사용하여 스택에 사용자 지정 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/pci_passthru_controller.yaml \
  -e /home/stack/templates/pci_passthru_compute.yaml
```

11. **PCI** 장치를 요청하도록 플레이버를 구성합니다. 다음 예제에서는 각각 공급업체 **ID**가 **10de** 이고 제품 **ID**가 **13f2** 인 두 개의 장치를 요청합니다.

```
# openstack flavor set m1.large \
  --property "pci_passthrough:alias"="t4:2"
```

검증

1. **PCI** 패스스루 장치를 사용하여 인스턴스를 생성합니다.

```
# openstack server create --flavor m1.large \
  --image <custom_gpu> --wait test-pci
```

<custom_gpu> 를 필수 **GPU** 드라이버가 설치된 사용자 정의 인스턴스 이미지의 이름으로 바꿉니다.

2. 클라우드 사용자로 인스턴스에 로그인합니다.

3. 인스턴스에서 **GPU**에 액세스할 수 있는지 확인하려면 인스턴스에서 다음 명령을 입력합니다.

```
$ lspci -nn | grep <gpu_name>
```

4. **NVIDIA** 시스템 관리 인터페이스 상태를 확인하려면 인스턴스에서 다음 명령을 입력합니다.

```
$ nvidia-smi
```

출력 예:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================
====|
|   0  Tesla T4            Off  | 00000000:01:00.0 Off |                    0 |
| N/A   43C    P0    20W /  70W |      0MiB / 15109MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================
==|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

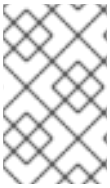
15장. 실시간 컴퓨팅 구성

클라우드 관리자는 대기 시간이 짧은 정책을 준수하고 실시간 처리를 수행하기 위해 컴퓨팅 노드에 인스턴스가 필요할 수 있습니다. 실시간 컴퓨팅 노드에는 실시간 지원 커널, 특정 가상화 모듈 및 최적화된 배포 매개 변수가 포함되어 실시간 처리 요구 사항을 지원하고 대기 시간을 최소화합니다.

실시간 계산을 활성화하는 프로세스에는 다음이 포함됩니다.

- 컴퓨팅 노드의 BIOS 설정 구성
- 실시간 커널 및 실시간 KVM(RT-KVM) 커널 모듈로 실시간 이미지 빌드
- ComputeRealTime 역할을 컴퓨팅 노드에 할당

NFV 워크로드에 대한 실시간 Compute 배포의 사용 사례 예제는 example를 참조하십시오. *네트워크 기능 가상화 계획 및 구성 가이드의 OVS-DPDK 구성 ODL 및 VXLAN 터널링* 섹션.



참고

실시간 컴퓨팅 노드는 Red Hat Enterprise Linux 버전 7.5 이상에서만 지원됩니다.

15.1. 실시간 컴퓨팅 노드 준비

오버클라우드에 실시간 계산을 배포하려면 먼저 Red Hat Enterprise Linux Real-Time KVM(RT-KVM)을 활성화하고 실시간 오버클라우드 이미지를 지원하도록 BIOS를 구성해야 합니다.

사전 요구 사항

- RT-KVM 컴퓨팅 노드에 Red Hat 인증 서버를 사용해야 합니다. 자세한 내용은 Red Hat Enterprise Linux for Real Time 7 certified servers 를 참조하십시오.
- rhel-8-for-x86_64-nfv-rpms 리포지토리에 액세스하려면 별도의 *Red Hat OpenStack Platform for Real Time* 서브스크립션이 필요합니다. 언더클라우드의 리포지토리 및 서브스크립션 관리에 대한 자세한 내용은 *Director 설치 및 사용 가이드의 언더클라우드 등록 및 서브스크립션 연결*을 참조하십시오.

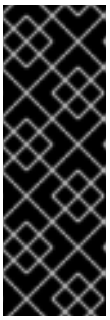
절차

1. **실시간 오버클라우드 이미지를 빌드하려면 RT-KVM에 대해 rhel-8-for-x86_64-nfv-rpms 리포지토리를 활성화해야 합니다. 리포지토리에서 설치할 패키지를 확인하려면 다음 명령을 입력합니다.**

```
$ dnf repo-pkgs rhel-8-for-x86_64-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
kernel-rt.x86_64                4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-rpms
kernel-rt-debug.x86_64          4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-devel.x86_64    4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-kvm.x86_64      4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-devel.x86_64          4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-doc.noarch            4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-rpms
kernel-rt-kvm.x86_64            4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
[ output omitted…]
```

2. **실시간 컴퓨팅 노드의 오버클라우드 이미지를 빌드하려면 언더클라우드에 libguestfs-tools 패키지를 설치하여 virt-customize 툴을 가져옵니다.**

```
(undercloud)$ sudo dnf install libguestfs-tools
```



중요

언더클라우드에 libguestfs-tools 패키지를 설치하는 경우 iscsid.socket 을 비활성화하여 언더클라우드의 tripleo_iscsid 서비스와 포트 충돌을 방지합니다.

```
$ sudo systemctl disable --now iscsid.socket
```

3. **이미지를 추출합니다.**

```
(undercloud)$ tar -xf /usr/share/rhosp-director-images/overcloud-full.tar
(undercloud)$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent.tar
```

4. **기본 이미지를 복사합니다.**

```
(undercloud)$ cp overcloud-full.qcow2 overcloud-realtime-compute.qcow2
```

5. 이미지를 등록하고 필요한 서브스크립션을 구성합니다.

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager register --username=<username> --password=<password>'
[  0.0] Examining the guest ...
[ 10.0] Setting a random seed
[ 10.0] Running: subscription-manager register --username=<username> --password=
<password>
[ 24.0] Finishing off
```

사용자 이름 및 암호 값을 **Red Hat** 고객 계정 세부 정보로 바꿉니다.

실시간 오버클라우드 이미지 빌드에 대한 일반적인 정보는 기술 자료 문서 **virt-customize**로 **Red Hat Enterprise Linux OpenStack Platform Overcloud** 이미지 수정을 참조하십시오.

6. ***Red Hat OpenStack Platform for Real Time*** 서브스크립션의 **SKU**를 찾아보십시오. **SKU**는 동일한 계정 및 인증 정보를 사용하여 이미 **Red Hat Subscription Manager**에 등록된 시스템에 있을 수 있습니다.

```
$ sudo subscription-manager list
```

7. ***Red Hat OpenStack Platform for Real Time*** 서브스크립션을 이미지에 연결합니다.

```
(undercloud)$  virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager attach --pool [subscription-pool]'
```

8. 이미지에 **rt** 를 구성하는 스크립트를 생성합니다.

```
(undercloud)$ cat rt.sh
 #!/bin/bash

 set -eux

 subscription-manager repos --enable=[REPO_ID]
 dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
 dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host

 # END OF SCRIPT
```

9. 스크립트를 실행하여 실시간 이미지를 구성합니다.

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee
virt-customize.log
```

10. **SELinux 레이블을 다시 지정합니다.**

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --selinux-relabel
```

11. **vmlinuz 및 initrd 추출. 예를 들면 다음과 같습니다.**

```
(undercloud)$ mkdir image
(undercloud)$ guestmount -a overcloud-realtime-compute.qcow2 -i --ro image
(undercloud)$ cp image/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 ./overcloud-
realtime-compute.vmlinuz
(undercloud)$ cp image/boot/initramfs-4.18.0-80.7.1.rt9.153.el8_0.x86_64.img ./overcloud-
realtime-compute.initrd
(undercloud)$ guestunmount image
```

참고

vmlinuz 및 initramfs 파일 이름의 소프트웨어 버전은 커널 버전에 따라 다릅니다.

12. **이미지를 업로드합니다.**

```
(undercloud)$ openstack overcloud image upload \
 --update-existing --os-image-name
 overcloud-realtime-compute.qcow2
```

이제 선택 컴퓨팅 노드에 대해 ComputeRealTime 구성 가능 역할과 함께 사용할 수 있는 실시간 이미지가 있습니다.

13. **실시간 컴퓨팅 노드의 대기 시간을 줄이려면 컴퓨팅 노드에서 BIOS 설정을 수정해야 합니다. 컴퓨팅 노드 BIOS 설정에서 다음 구성 요소에 대한 모든 옵션을 비활성화해야 합니다.**

- 전원 관리
- 하이퍼-스레딩

- CPU 절전 상태

- 논리 프로세서

이러한 설정에 대한 설명 및 비활성화에 미치는 영향은 BIOS 매개 변수 설정을 참조하십시오. BIOS 설정을 변경하는 방법에 대한 자세한 내용은 하드웨어 제조업체 설명서를 참조하십시오.

15.2. 실시간 컴퓨팅 역할 배포

RHOSP(Red Hat OpenStack Platform) director는 실시간 컴퓨팅 노드를 배포하는 데 사용할 수 있는 ComputeRealTime 역할에 대한 템플릿을 제공합니다. 실시간 컴퓨팅 노드를 지정하려면 추가 단계를 수행해야 합니다.

절차

1. /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml 파일을 기반으로 ComputeRealTime 역할에 대한 매개 변수를 설정하는 compute-real-time.yaml 환경 파일을 생성합니다.

```
cp /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml /home/stack/templates/compute-real-time.yaml
```

파일에는 다음 매개변수의 값이 포함되어야 합니다.

- IsolCpusList 및 NovaComputeCpuDedicatedSet: 실시간 워크로드를 위해 예약할 격리된 CPU 코어 및 가상 CPU 고정 목록입니다. 이 값은 실시간 컴퓨팅 노드의 CPU 하드웨어에 따라 다릅니다.

- NovaComputeCpuSharedSet: 에뮬레이터 스레드를 예약할 호스트 CPU 목록입니다.

- KernelArgs: 실시간 컴퓨팅 노드의 커널에 전달할 인수입니다. 예를 들어 default_hugepagesz=1G hugepagesz=1G hugepages=<number_of_1G_pages_to_reserve> hugepagesz=2M hugepages=<number_of_2M_pages> 를 사용하여 여러 크기의 대규모 페이지가 있는 게스트의 메모리 요구 사항을 정의할 수 있습니다. 이 예에서 기본 크기는 1GB이지만 2M 대규모 페이지를 예약할 수도 있습니다.

- **NovaComputeDisableIrqBalance: tuned** 서비스가 **irqbalance** 서비스가 아닌 실시간 배포에 대한 **IRQ** 분산을 관리하므로 이 매개변수가 **Real-time Compute** 노드에 대해 **true** 로 설정되어 있는지 확인합니다.

2. **ComputeRealTime** 역할을 역할 데이터 파일에 추가하고 파일을 다시 생성합니다. 예를 들면 다음과 같습니다.

```
$ openstack overcloud roles generate -o /home/stack/templates/rt_roles_data.yaml Controller
Compute ComputeRealTime
```

이 명령은 다음 예제와 유사한 내용을 사용하여 **ComputeRealTime** 역할을 생성하고 **ImageDefault** 옵션도 **overcloud-realtime-compute** 로 설정합니다.

```
- name: ComputeRealTime
  description: |
    Compute role that is optimized for real-time behaviour. When using this role
    it is mandatory that an overcloud-realtime-compute image is available and
    the role specific parameters IsolCpusList, NovaComputeCpuDedicatedSet and
    NovaComputeCpuSharedSet are set accordingly to the hardware of the real-time compute
nodes.
  CountDefault: 1
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computerealtime-%index%'
  ImageDefault: overcloud-realtime-compute
  RoleParametersDefault:
    TunedProfileName: "realtime-virtual-host"
    KernelArgs: ""      # these must be set in an environment file
    IsolCpusList: ""    # or similar according to the hardware
    NovaComputeCpuDedicatedSet: ""  # of real-time nodes
    NovaComputeCpuSharedSet: ""     #
    NovaLibvirtMemStatsPeriodSeconds: 0
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
```

```
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
```

사용자 지정 역할 및 **roles-data.yaml** 에 대한 일반 정보는 **Roles** 를 참조하십시오.

3. **compute-realtime** 플레이버를 만들어 실시간 워크로드를 지정하려는 노드에 태그를 지정합니다. 예를 들면 다음과 같습니다.

```
$ source ~/stackrc
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute-realtime
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property "capabilities:profile"="compute-realtime"
compute-realtime
```

4. **compute-realtime** 프로필을 사용하여 실시간 워크로드를 지정할 각 노드에 태그를 지정합니다.

```
$ openstack baremetal node set --property capabilities='profile:compute-
realtime,boot_option:local' <node_uuid>
```

5. 다음 콘텐츠가 포함된 환경 파일을 만들어 **ComputeRealTime** 역할을 **compute-realtime** 플레이버에 매핑합니다.

```
parameter_defaults:
  OvercloudComputeRealTimeFlavor: compute-realtime
```

6. 다른 환경 파일과 함께 스택에 환경 파일과 새 역할 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -r /home/stack/templates/rt~/my_roles_data.yaml \
 -e home/stack/templates/compute-real-time.yaml
```

15.3. 샘플 배포 및 테스트 시나리오

다음 예제 절차에서는 간단한 단일 노드 배포를 사용하여 환경 변수 및 기타 지원 구성이 올바르게 설정되었는지 테스트합니다. 클라우드에 배포하는 노드 및 인스턴스 수에 따라 실제 성능 결과가 다를 수 있습니다.

절차

1. 다음 매개변수를 사용하여 **compute-real-time.yaml** 파일을 생성합니다.

```
parameter_defaults:
  ComputeRealTimeParameters:
    IsolCpusList: "1"
    NovaComputeCpuDedicatedSet: "1"
    NovaComputeCpuSharedSet: "0"
    KernelArgs: "default_hugepagesz=1G hugepagesz=1G hugepages=16"
    NovaComputeDisableIrqBalance: true
```

2. **ComputeRealTime** 역할을 사용하여 새 **rt_roles_data.yaml** 파일을 생성합니다.

```
$ openstack overcloud roles generate \
 -o ~/rt_roles_data.yaml Controller ComputeRealTime
```

3. 다른 환경 파일이 있는 스택에 **compute-real-time .yaml** 및 **rt_roles_data.yaml** 을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -r /home/stack/rt_roles_data.yaml \
  -e [your environment files] \
  -e /home/stack/templates/compute-real-time.yaml
```

이 명령은 컨트롤러 노드 1개와 실시간 컴퓨팅 노드 1개를 배포합니다.

4. **실시간 컴퓨팅 노드에 로그인하여 다음 매개변수를 확인합니다.**

```
[root@overcloud-computerealtime-0 ~]# uname -a
Linux overcloud-computerealtime-0 4.18.0-80.7.1.rt9.153.el8_0.x86_64 #1 SMP PREEMPT
RT Wed Dec 13 13:37:53 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@overcloud-computerealtime-0 ~]# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 root=UUID=45ae42d0-
58e7-44fe-b5b1-993fe97b760f ro console=tty0 crashkernel=auto console=ttyS0,115200
default_hugepagesz=1G hugepagesz=1G hugepages=16
[root@overcloud-computerealtime-0 ~]# tuned-adm active
Current active profile: realtime-virtual-host
[root@overcloud-computerealtime-0 ~]# grep ^isolated_cores /etc/tuned/realtime-virtual-host-
variables.conf
isolated_cores=1
[root@overcloud-computerealtime-0 ~]# cat /usr/lib/tuned/realtime-virtual-
host/lapic_timer_adv_ns
4000 # The returned value must not be 0
[root@overcloud-computerealtime-0 ~]# cat
/sys/module/kvm/parameters/lapic_timer_advance_ns
4000 # The returned value must not be 0
# To validate hugepages at a host level:
[root@overcloud-computerealtime-0 ~]# cat /proc/meminfo | grep -E
HugePages_Total|Hugepagesize
HugePages_Total:      64
Hugepagesize:    1048576 kB
# To validate hugepages on a per NUMA level (below example is a two NUMA compute
host):
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
32
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node1/hugepages/hugepages-1048576kB/nr_hugepages
32
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_dedicated_set
1
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_shared_set
0
[root@overcloud-computerealtime-0 ~]# systemctl status irqbalance
● irqbalance.service - irqbalance daemon
```

```
   Loaded: loaded (/usr/lib/systemd/system/irqbalance.service; enabled; vendor preset:
enabled)
   Active: inactive (dead) since Tue 2021-03-30 13:36:31 UTC; 2s ago
```

15.4. 실시간 인스턴스 시작 및 튜닝

실시간 컴퓨팅 노드를 배포하고 구성한 후 해당 노드에서 실시간 인스턴스를 시작할 수 있습니다. CPU 고정, NUMA 토폴로지 필터 및 대규모 페이지를 사용하여 이러한 실시간 인스턴스를 추가로 구성할 수 있습니다.

사전 요구 사항

- 실시간 Compute 역할 배포에 설명된 대로 compute-realtime 플레이버가 오버클라우드에 있습니다.

절차

1. 실시간 인스턴스를 시작합니다.

```
# openstack server create  --image <rhel> \
 --flavor r1.small --nic net-id=<dpdk_net> test-rt
```

2. 선택 사항: 인스턴스에서 할당된 에뮬레이터 스레드를 사용하는지 확인합니다.

```
# virsh dumpxml <instance_id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='5'/>
  <vcpupin vcpu='3' cpuset='7'/>
  <emulatorpin cpuset='0-1'/>
  <vcpusched vcpus='2-3' scheduler='fifo'
  priority='1'/>
</cputune>
```

CPU 고정 및 에뮬레이터 스레드 정책 설정

실시간 워크로드를 위해 각 실시간 계산 노드에 충분한 CPU가 있는지 확인하려면 인스턴스에 대해 하나 이상의 가상 CPU(vCPU)를 호스트의 물리적 CPU(pCPU)에 고정해야 합니다. 그 vCPU의 에뮬레이터 스레드는 해당 pCPU 전용으로 유지됩니다.

전용 CPU 정책을 사용하도록 플레이버를 구성합니다. 이렇게 하려면 hw:cpu_policy 매개 변수를 플

레이버 전용으로 설정합니다. 예를 들면 다음과 같습니다.

```
# openstack flavor set --property hw:cpu_policy=dedicated 99
```

참고

리소스 할당량에 사용할 실시간 컴퓨팅 노드에 충분한 pCPU가 있는지 확인합니다.

네트워크 설정 최적화

배포 요구 사항에 따라 network-environment.yaml 파일에서 매개변수를 설정해야 특정 실시간 워크로드에 대해 네트워크를 튜닝해야 할 수 있습니다.

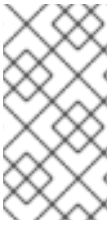
OVS-DPDK에 최적화된 구성 예를 검토하려면 *Network Functions Virtualization Planning and Configuration Guide* 의 OVS-DPDK 매개변수 구성 섹션을 참조하십시오.

대규모 페이지 구성

기본 대규모 페이지 크기를 1GB로 설정하는 것이 좋습니다. 그렇지 않으면 TLB 플러시에서 vCPU 실행에 지터를 만들 수 있습니다. 대규모 페이지 사용에 대한 일반 정보는 Running DPDK 애플리케이션 웹 페이지를 참조하십시오.

PMU(성능 모니터링 단위) 에뮬레이션 비활성화

인스턴스는 vPMU로 이미지 또는 플레이버를 지정하여 PMU 지표를 제공할 수 있습니다. PMU 지표를 제공하면 대기 시간이 발생합니다.



참고

NovaLibvirtCPUMode 가 host-passthrough 로 설정된 경우 vPMU는 기본적으로 활성화됩니다.

PMU 지표가 필요하지 않은 경우 인스턴스를 생성하는 데 사용되는 이미지 또는 플레이버에서 PMU 속성을 "False"로 설정하여 vPMU를 비활성화하여 대기 시간을 줄입니다.

- 이미지: hw_pmu=False

- **Flavor: hw:pmu=False**

16장. 인스턴스 관리

클라우드 관리자는 클라우드에서 실행되는 인스턴스를 모니터링하고 관리할 수 있습니다.

16.1. 인스턴스의 VNC 콘솔에 대한 연결 보안

VNC 프록시 서비스에 대한 들어오는 클라이언트 연결을 강제 적용하도록 허용되는 TLS 암호 및 최소 프로토콜 버전을 구성하여 인스턴스의 VNC 콘솔에 대한 연결을 보호할 수 있습니다.

절차

1. stack 사용자로 언더클라우드에 로그인합니다.

2. stackrc 파일을 소싱합니다.

```
[stack@director ~]$ source ~/stackrc
```

3. 컴퓨팅 환경 파일을 엽니다.

4. 인스턴스에 대한 VNC 콘솔 연결에 사용할 최소 프로토콜 버전을 구성합니다.

```
parameter_defaults:
  ...
  NovaVNCProxySSLMinimumVersion: <version>
```

<version> 을 허용되는 최소 SSL/TLS 프로토콜 버전으로 바꿉니다. 다음 유효한 값 중 하나로 설정합니다.

- 기본값: 기본 시스템 OpenSSL 기본값을 사용합니다.
- tlsv1_1: 이후 버전을 지원하지 않는 클라이언트가 있는 경우 를 사용합니다.

참고

TLS 1.0 및 **TLS 1.1**은 **RHEL 8**에서 더 이상 사용되지 않으며 **RHEL 9**에서는 지원되지 않습니다.

- **tlsv1_2:** 인스턴스에 대한 **VNC** 콘솔 연결에 사용할 **SSL/TLS** 암호를 구성하려면 를 사용합니다.

5. 허용된 최소 **SSL/TLS** 프로토콜 버전을 **tlsv1_2** 로 설정하는 경우 인스턴스에 **VNC** 콘솔 연결에 사용할 **SSL/TLS** 암호를 구성합니다.

```
parameter_defaults:
  NovaVNCProxySSLCiphers: <ciphers>
```

<ciphers> 를 허용할 암호화 제품군의 콜론으로 구분된 목록으로 바꿉니다. **openssl** 에서 사용 가능한 암호 목록을 검색합니다.

6. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files]  \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

16.2. 데이터베이스 정리

계산 서비스에는 데이터베이스 스키마 적용, 업그레이드 중 온라인 데이터 마이그레이션 수행, 데이터베이스 관리 및 데이터베이스 관리 및 정리와 같은 배포, 업그레이드, 정리 및 유지 관리 관련 작업을 수행하는 데 사용할 수 있는 관리 도구 **nova-manage** 가 포함되어 있습니다.

director는 **cron**을 사용하여 오버클라우드에서 다음 데이터베이스 관리 작업을 자동화합니다.

- 삭제된 행을 프로덕션 테이블에서 섀도우 테이블로 이동하여 아카이브에서 인스턴스 레코드를 삭제했습니다.
- 아카이브가 완료된 후 섀도우 테이블에서 삭제된 행을 제거합니다.

16.2.1. 데이터베이스 관리 구성

cron 작업은 기본 설정을 사용하여 데이터베이스 관리 작업을 수행합니다. 기본적으로 데이터베이스 아카이브 **cron** 작업은 **00:01**에서 매일 실행되며 데이터베이스 제거 **cron** 작업은 매일 **05:00**에 실행되며 둘 다 **0~3600**초 사이의 지터가 있습니다. **heat** 매개변수를 사용하여 필요에 따라 이러한 설정을 수정할 수 있습니다.

절차

1. 컴퓨팅 환경 파일을 엽니다.

2. 추가하거나 수정할 **cron** 작업을 제어하는 **heat** 매개변수를 추가합니다. 예를 들어 보관된 직후 새도우 테이블을 제거하려면 다음 매개 변수를 **"True"**로 설정합니다.

```
parameter_defaults:
  ...
  NovaCronArchiveDeleteRowsPurge: True
```

데이터베이스 **cron** 작업을 관리하는 **heat** 매개변수 전체 목록은 **Compute** 서비스 자동화된 데이터베이스 관리에 대한 구성 옵션을 참조하십시오.

3. 업데이트를 **Compute** 환경 파일에 저장합니다.

4. 다른 환경 파일을 사용하여 스택에 **Compute** 환경 파일을 추가하고 오버클라우드를 배포합니다.

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/<compute_environment_file>.yaml
```

16.2.2. 계산 서비스 자동 데이터베이스 관리를 위한 구성 옵션

다음 **heat** 매개 변수를 사용하여 데이터베이스를 관리하는 자동화된 **cron** 작업을 활성화하고 수정합니다.

표 **16.1.** 컴퓨팅**(nova)** 서비스 **cron** 매개변수

매개변수	설명

매개변수	설명
NovaCronArchiveDeleteAllCells	이 매개 변수를 "True"로 설정하여 모든 셀에서 삭제된 인스턴스 레코드를 아카이브로 설정합니다. 기본값: **True**
NovaCronArchiveDeleteRowsAge	이 매개 변수를 사용하여 수명(일)에 따라 인스턴스 레코드가 삭제된 아카이브를 보관합니다. 새도우 테이블에서 오늘 이전의 데이터를 보관하려면 **0** 으로 설정합니다. 기본값: **90**
NovaCronArchiveDeleteRowsDestination	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 로깅할 파일을 구성합니다. 기본값: **/var/log/nova/nova-rowsflush.log**
NovaCronArchiveDeleteRowsHour	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 다른 테이블로 이동하도록 cron 명령을 실행할 시간을 구성합니다. 기본값: **0**
NovaCronArchiveDeleteRowsMaxDelay	삭제된 인스턴스 레코드를 다른 테이블로 이동하기 전에 이 매개 변수를 사용하여 최대 지연 시간(초)을 구성합니다. 기본값: **3600**
NovaCronArchiveDeleteRowsMaxRows	이 매개 변수를 사용하여 다른 테이블로 이동할 수 있는 삭제된 최대 인스턴스 레코드 수를 구성합니다. 기본값: **1000**
NovaCronArchiveDeleteRowsMinute	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 다른 테이블로 이동하도록 cron 명령을 실행하는 시간 후를 구성합니다. 기본값: **1**
NovaCronArchiveDeleteRowsMonthday	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 다른 테이블로 이동하도록 cron 명령을 실행할 날짜를 구성합니다. 기본값: ***** (모든 일)
NovaCronArchiveDeleteRowsMonth	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 다른 테이블로 이동하도록 cron 명령을 실행할 월을 구성합니다. 기본값: ***** (매월)

매개변수	설명
NovaCronArchiveDeleteRowsPurge	예약된 아카이브 직후 섀도우 테이블을 제거하도록 이 매개변수를 "True"로 설정합니다. 기본값: **False**
NovaCronArchiveDeleteRowsUntilComplete	모든 레코드가 이동될 때까지 삭제된 인스턴스 레코드를 다른 테이블로 이동하려면 이 매개변수를 "True"로 설정합니다. 기본값: **True**
NovaCronArchiveDeleteRowsUser	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 보관하는 crontab을 소유하고 crontab에서 사용하는 로그 파일에 대한 액세스 권한이 있는 사용자를 구성합니다. 기본값: **nova**
NovaCronArchiveDeleteRowsWeekday	이 매개 변수를 사용하여 삭제된 인스턴스 레코드를 다른 테이블로 이동하도록 cron 명령을 실행할 요일을 구성합니다. 기본값: ***** (모든 일)
NovaCronPurgeShadowTablesAge	이 매개변수를 사용하여 수명(일)에 따라 섀도우 테이블을 제거합니다. **0** 으로 설정하여 오늘보다 오래된 섀도우 테이블을 제거합니다. 기본값: **14**
NovaCronPurgeShadowTablesAllCells	모든 셀에서 섀도우 테이블을 제거하도록 이 매개변수를 "True"로 설정합니다. 기본값: **True**
NovaCronPurgeShadowTablesDestination	이 매개 변수를 사용하여 삭제된 섀도우 테이블을 로깅할 파일을 구성합니다. 기본값: **/var/log/nova/nova-rowspurge.log**
NovaCronPurgeShadowTablesHour	이 매개 변수를 사용하여 섀도우 테이블을 제거하기 위해 cron 명령을 실행할 시간을 구성합니다. 기본값: **5**
NovaCronPurgeShadowTablesMaxDelay	섀도우 테이블을 제거하기 전에 이 매개변수를 사용하여 최대 지연 시간(초)을 구성합니다. 기본값: **3600**

매개변수	설명
NovaCronPurgeShadowTablesMinute	이 매개 변수를 사용하여 새도우 테이블을 제거하기 위해 cron 명령을 실행하는 시간 후를 구성합니다. 기본값: **0**
NovaCronPurgeShadowTablesMonth	이 매개 변수를 사용하여 cron 명령을 실행하여 새도우 테이블을 제거하는 월을 구성합니다. 기본값: ***** (매월)
NovaCronPurgeShadowTablesMonthday	이 매개 변수를 사용하여 cron 명령을 실행하여 새도우 테이블을 제거하는 날을 구성합니다. 기본값: ***** (모든 일)
NovaCronPurgeShadowTablesUser	이 매개 변수를 사용하여 crontab에서 사용하는 로그 파일에 대한 액세스 권한이 있는 crontab을 소유한 사용자를 구성합니다. 기본값: **nova**
NovaCronPurgeShadowTablesVerbose	삭제된 새도우 테이블의 로그 파일에서 자세한 로깅을 활성화하려면 이 매개변수를 사용합니다. 기본값: **False**
NovaCronPurgeShadowTablesWeekday	이 매개 변수를 사용하여 cron 명령을 실행하여 새도우 테이블을 제거하기 위해 요일을 구성합니다. 기본값: ***** (모든 일)

16.3. 컴퓨팅 노드 간 가상 머신 인스턴스 마이그레이션

오버클라우드의 한 컴퓨팅 노드에서 다른 컴퓨팅 노드로 인스턴스를 마이그레이션하여 유지보수를 수행하거나 워크로드를 리밸런스하거나 장애가 발생한 노드를 교체해야 하는 경우가 있습니다.

컴퓨팅 노드 유지보수

예를 들어 하드웨어 유지 관리 또는 복구, 커널 업그레이드 및 소프트웨어 업데이트를 수행하기 위해 Compute 노드를 일시적으로 사용해야 하는 경우 컴퓨팅 노드에서 실행 중인 인스턴스를 다른 컴퓨팅 노드로 마이그레이션할 수 있습니다.

컴퓨팅 노드 실패

컴퓨팅 노드에 오류가 발생하고 서비스를 제공하거나 교체해야 하는 경우 실패한 컴퓨팅 노드에서 정상적인 컴퓨팅 노드로 인스턴스를 마이그레이션할 수 있습니다.

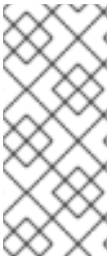
실패한 컴퓨팅 노드

컴퓨팅 노드가 이미 실패하면 인스턴스를 비울 수 있습니다. 동일한 이름, UUID, 네트워크 주소 및 컴퓨팅 노드가 실패하기 전에 할당된 기타 리소스를 사용하여 다른 컴퓨팅 노드의 원래 이미지에서 인스턴스를 다시 빌드할 수 있습니다.

워크로드 리밸런싱

하나 이상의 인스턴스를 다른 컴퓨팅 노드로 마이그레이션하여 워크로드를 리밸런싱할 수 있습니다. 예를 들어 컴퓨팅 노드의 인스턴스를 통합하여 전원을 절약하고, 인스턴스를 다른 네트워크 리소스에 물리적으로 더 가까운 컴퓨팅 노드로 마이그레이션하여 대기 시간을 줄이거나, 인스턴스를 컴퓨팅 노드에 배포하여 핫스팟을 방지하고 복원력을 높일 수 있습니다.

director는 안전한 마이그레이션을 제공하기 위해 모든 컴퓨팅 노드를 구성합니다. 모든 컴퓨팅 노드에는 마이그레이션 프로세스 중 다른 컴퓨팅 노드에 대한 액세스 권한을 각 호스트의 사용자에게 제공하기 위해 공유 **SSH** 키가 필요합니다. **director**는 **OS::TripleO::Services::NovaCompute** 구성 가능 서비스를 사용하여 이 키를 생성합니다. 이 구성 가능 서비스는 기본적으로 모든 계산 역할에 포함된 기본 서비스 중 하나입니다. 자세한 내용은 ***Advanced Overcloud Customization*** 가이드의 **Composable Services and Custom Roles** 를 참조하십시오.



참고

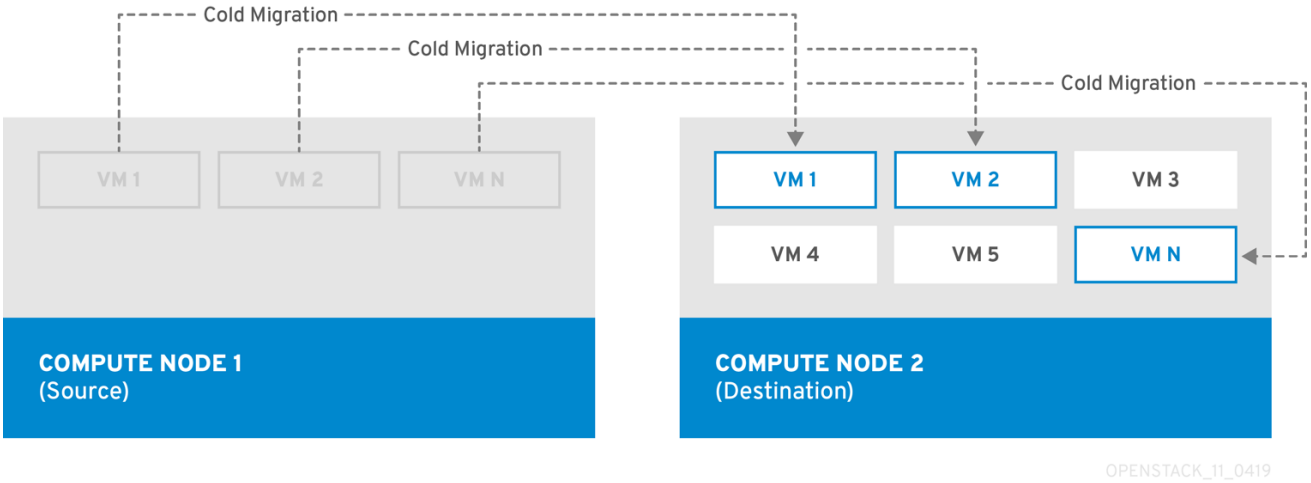
작동 중인 컴퓨팅 노드가 있고 백업 목적으로 인스턴스 사본을 만들거나 인스턴스를 다른 환경에 복사하려는 경우 ***Director 설치 및 사용*** 가이드의 오버클라우드로 가상 머신을 가져오는 절차를 따르십시오.

16.3.1. 마이그레이션 유형

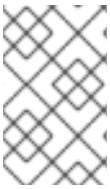
RHOSP(Red Hat OpenStack Platform)는 다음과 같은 유형의 마이그레이션을 지원합니다.

콜드 마이그레이션

콜드 마이그레이션 또는 실시간이 아닌 마이그레이션에는 소스 컴퓨팅 노드에서 대상 컴퓨팅 노드로 마이그레이션하기 전에 실행 중인 인스턴스를 종료해야 합니다.



콜드 마이그레이션에는 인스턴스에 다운타임이 있습니다. 마이그레이션된 인스턴스는 동일한 볼륨 및 IP 주소에 대한 액세스를 유지 관리합니다.

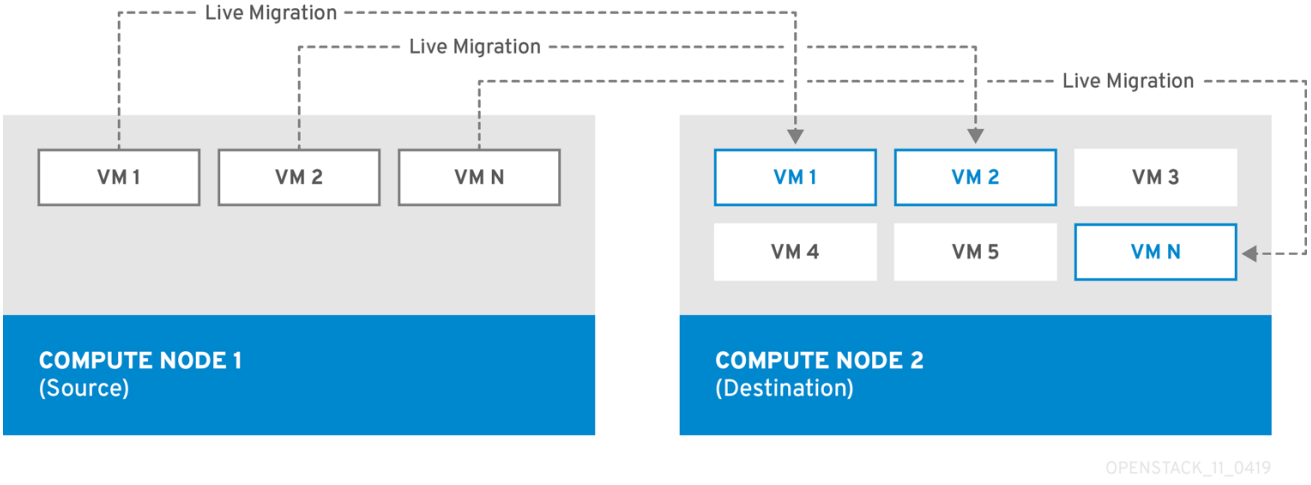


참고

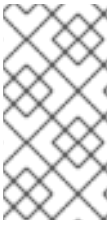
콜드 마이그레이션에는 소스 및 대상 컴퓨팅 노드가 모두 실행 중이어야 합니다.

실시간 마이그레이션

실시간 마이그레이션에는 인스턴스를 종료하지 않고 소스 컴퓨팅 노드에서 대상 컴퓨팅 노드로 이동하고 상태 일관성을 유지 관리하는 작업이 포함됩니다.



실시간 인스턴스 마이그레이션에는 다운타임이 거의 발생하지 않거나 발생하지 않습니다. 그러나 실시간 마이그레이션은 마이그레이션 작업 기간 동안 성능에 영향을 미칩니다. 따라서 인스턴스를 마이그레이션하는 동안 중요한 경로에서 가져와야 합니다.



참고

실시간 마이그레이션을 수행하려면 소스 및 대상 컴퓨팅 노드가 모두 실행 중이어야 합니다.

경우에 따라 인스턴스는 실시간 마이그레이션을 사용할 수 없습니다. 자세한 내용은 *마이그레이션 제한 조건을* 참조하십시오.

비우기

소스 컴퓨팅 노드가 이미 실패했으므로 인스턴스를 마이그레이션해야 하는 경우 인스턴스를 비울 수 있습니다.

16.3.2. 마이그레이션 제약 조건

마이그레이션 제한 조건은 일반적으로 블록 마이그레이션, 구성 디스크 또는 하나 이상의 인스턴스가 컴퓨팅 노드의 물리 하드웨어에 액세스하는 경우에 발생합니다.

CPU 제약 조건

소스 및 대상 컴퓨팅 노드에 동일한 CPU 아키텍처가 있어야 합니다. 예를 들어 Red Hat은 x86_64 CPU에서 ppc64le CPU로 인스턴스 마이그레이션을 지원하지 않습니다.

다른 CPU 모델 간 마이그레이션은 지원되지 않습니다. 경우에 따라 CPU 호스트 패스스루를 사용하는 인스턴스와 같이 소스 및 대상 컴퓨팅 노드의 CPU가 정확히 일치해야 합니다. 모든 경우에 대상 노드의 CPU 기능은 소스 노드의 CPU 기능의 상위 세트여야 합니다.

메모리 제약 조건

대상 컴퓨팅 노드에는 사용 가능한 충분한 RAM이 있어야 합니다. 메모리 초과 서브스크립션으로 인해 마이그레이션이 실패할 수 있습니다.

블록 마이그레이션 제약 조건

컴퓨팅 노드에 로컬로 저장된 디스크를 사용하는 인스턴스를 마이그레이션하는 데 Red Hat Ceph Storage와 같이 공유 스토리지를 사용하는 볼륨 지원 인스턴스를 마이그레이션하는 것보다 시간이 훨씬 오래 걸립니다. 이 대기 시간은 기본적으로 OpenStack Compute(nova)가 컨트롤 플레인 네트워크를 통해 컴퓨팅 노드 간에 블록 단위로 로컬 디스크를 마이그레이션하기 때문에 발생합니다. 반대로, 공유 스토리지를 사용하는 볼륨 기반 인스턴스(예: Red Hat Ceph Storage)는 각 컴퓨팅 노드에서 이미 공유 스토리지에 액세스할 수 있기 때문에 볼륨을 마이그레이션할 필요가 없습니다.

참고

많은 RAM을 사용하는 로컬 디스크 또는 인스턴스를 마이그레이션하여 컨트롤 플레인 네트워크의 네트워크 정체는 RabbitMQ와 같은 컨트롤 플레인 네트워크를 사용하는 다른 시스템의 성능에 영향을 줄 수 있습니다.

읽기 전용 드라이브 마이그레이션 제약 조건

드라이브에 읽기 및 쓰기 기능이 모두 있는 경우에만 드라이브 마이그레이션이 지원됩니다. 예를 들어 OpenStack Compute(nova)는 CD-ROM 드라이브 또는 읽기 전용 구성 드라이브를 마이그레이션할 수 없습니다. 그러나 OpenStack Compute(nova)는 vfat 와 같은 드라이브 형식의 구성 드라이브를 포함하여 읽기 및 쓰기 기능을 모두 사용하여 드라이브를 마이그레이션할 수 있습니다.

실시간 마이그레이션 제약 조건

실시간 마이그레이션 인스턴스에는 추가 제약이 있는 경우도 있습니다.

마이그레이션 중에 새 작업 없음

소스 및 대상 노드의 인스턴스 복사본 간에 상태 일관성을 달성하려면 RHOSP에서 실시간 마이그레이션 중에 새 작업을 방지해야 합니다. 그렇지 않으면 실시간 마이그레이션에서 메모리 상태를 복제하는 속도보다 메모리에 쓰기 속도가 더 빠르게 발생하는 경우 실시간 마이그레이션에 시간이 오래 걸리거나 잠재적으로 끝나지 않을 수 있습니다.

NUMA를 사용하여 CPU 고정

Compute 구성의 NovaSchedulerDefaultFilters 매개변수에는 AggregateInstanceExtraSpecsFilter 및 NUMATopologyFilter 값이 포함되어야 합니다.

다중 셀 클라우드

다중 셀 클라우드에서는 동일한 셀의 다른 호스트로 인스턴스를 실시간으로 마이그레이션할 수 있지만, 셀 간에는 인스턴스를 마이그레이션할 수 없습니다.

유동 인스턴스

유동 인스턴스를 실시간으로 마이그레이션하는 경우 대상 컴퓨팅 노드의 NovaComputeCpuSharedSet 구성이 소스 컴퓨팅 노드의 NovaComputeCpuSharedSet 구성과 다른 경우 대상 컴퓨팅 노드에 공유(고정 해제) 인스턴스에 구성된 CPU에 인스턴스가 할당되지 않습니다. 따라서 유동 인스턴스를 실시간 마이그레이션해야 하는 경우 전용(고정) 및 공유(고정되지 않은) 인스턴스에 대해 동일한 CPU 매핑으로 모든 컴퓨팅 노드를 구성하거나 공유 인스턴스에 호스트 집계를 사용해야 합니다.

대상 컴퓨팅 노드 용량

대상 컴퓨팅 노드에는 마이그레이션할 인스턴스를 호스팅할 충분한 용량이 있어야 합니다.

SR-IOV 실시간 마이그레이션

SR-IOV 기반 네트워크 인터페이스가 있는 인스턴스를 실시간 마이그레이션할 수 있습니다. 직접 모드 **SR-IOV** 네트워크 인터페이스가 있는 실시간 마이그레이션 인스턴스로 인해 네트워크 중단이 발생합니다. 이는 마이그레이션 중에 직접 모드 인터페이스를 분리하고 다시 연결해야 하기 때문입니다.

ML2/OVN 배포에서 패킷 손실

ML2/OVN은 패킷 손실 없이 실시간 마이그레이션을 지원하지 않습니다. 이는 **OVN**에서 여러 포트 바인딩을 처리할 수 없기 때문에 포트 마이그레이션 시기를 알 수 없기 때문입니다.

실시간 마이그레이션 중에 패키지 손실을 최소화하려면 마이그레이션이 완료되면 대상 호스트에서 인스턴스를 표시하도록 **ML2/OVN** 배포를 구성하십시오.

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      workarounds/enable_qemu_monitor_announce_self:
        value: 'True'
```

ML2/OVS 배포에서 실시간 마이그레이션

ML2/OVS 배포에서 실시간 마이그레이션 시 패키지 손실을 최소화하려면 **ML2/OVS** 배포를 구성하여 **Networking** 서비스**(neutron)** 실시간 마이그레이션 이벤트를 활성화하고 마이그레이션이 완료되면 대상 호스트에서 인스턴스를 발표합니다.

```
parameter_defaults:
  NetworkExtraConfig:
    neutron::config::neutron_config:
      nova/live_migration_events:
        value: 'True'
  ComputeExtraConfig:
    nova::config::nova_config:
      workarounds/enable_qemu_monitor_announce_self:
        value: 'True'
```

실시간 마이그레이션을 방지하는 제약 조건

다음 기능을 사용하는 인스턴스를 실시간 마이그레이션할 수 없습니다.

PCI 통과

QEMU/KVM 하이퍼바이저는 컴퓨팅 노드의 **PCI** 장치를 인스턴스에 연결할 수 있도록 지원합니다. **PCI** 통과를 사용하여 **PCI** 장치에 대한 인스턴스 전용 액세스를 제공합니다. **PCI** 장치는 인스턴스의 운영 체제에 물리적으로 연결된 것처럼 표시되고 작동합니다. 그러나 **PCI** 통과에는 물리적 장치에 대한 직접 액세스 권한이 필요하기 때문에 **QEMU/KVM**은 **PCI** 패스스루를 사용하여 인스턴스의 실시간 마이그레이션을 지원하지 않습니다.

포트 리소스 요청

보장된 최소 대역폭 QoS 정책과 같이 리소스 요청이 있는 포트를 사용하는 인스턴스를 실시간 마이그레이션할 수 없습니다. 다음 명령을 사용하여 포트에 리소스 요청이 있는지 확인합니다.

```
$ openstack port show <port_name/port_id>
```

16.3.3. 마이그레이션 준비

하나 이상의 인스턴스를 마이그레이션하기 전에 마이그레이션할 인스턴스의 컴퓨팅 노드 이름과 ID를 확인해야 합니다.

절차

1. 소스 컴퓨팅 노드 호스트 이름과 대상 컴퓨팅 노드 호스트 이름을 식별합니다.

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

2. 소스 컴퓨팅 노드의 인스턴스를 나열하고 마이그레이션할 인스턴스 또는 인스턴스의 ID를 찾습니다.

```
(overcloud)$ openstack server list --host <source> --all-projects
```

<source> 를 소스 컴퓨팅 노드의 이름 또는 ID로 바꿉니다.

3. 선택 사항: 노드에서 유지보수를 수행하기 위해 소스 컴퓨팅 노드에서 인스턴스를 마이그레이션하는 경우 스케줄러가 유지보수 중에 새 인스턴스를 노드에 할당하지 않도록 노드를 비활성화해야 합니다.

```
(overcloud)$ openstack compute service set <source> nova-compute --disable
```

<source> 를 소스 컴퓨팅 노드의 호스트 이름으로 바꿉니다.

이제 마이그레이션을 수행할 준비가 되었습니다. 인스턴스 또는 라이브 마이그레이션 인스턴스에 대한 자세한 내용은 Cold 마이그레이션에 설명된 필수 절차를 따르십시오.

16.3.4. 인스턴스 콜드 마이그레이션

인스턴스를 콜드 마이그레이션하는 경우 인스턴스를 중지하고 다른 컴퓨팅 노드로 이동합니다. 콜드 마이그레이션은 **PCI** 통과를 사용하는 인스턴스 마이그레이션과 같이 실시간 마이그레이션이 용이하지 않은 마이그레이션 시나리오에 유용합니다. 스케줄러는 대상 컴퓨팅 노드를 자동으로 선택합니다. 자세한 내용은 마이그레이션 제한 조건을 참조하십시오.

절차

1. 인스턴스를 콜드 마이그레이션하려면 다음 명령을 입력하여 인스턴스의 전원을 끄고 이동합니다.

```
(overcloud)$ openstack server migrate <instance> --wait
```

- **<instance>** 를 마이그레이션할 인스턴스의 이름 또는 **ID**로 바꿉니다.
- 로컬에 저장된 볼륨을 마이그레이션하는 경우 **--block-migration** 플래그를 지정합니다.

2. 마이그레이션이 완료될 때까지 기다립니다. 인스턴스 마이그레이션이 완료될 때까지 기다리는 동안 마이그레이션 상태를 확인할 수 있습니다. 자세한 내용은 마이그레이션 상태 확인 을 참조하십시오.

3. 인스턴스의 상태를 확인합니다.

```
(overcloud)$ openstack server list --all-projects
```

"VERIFY_RESIZE" 상태는 마이그레이션을 확인하거나 되돌려야 함을 나타냅니다.

- 마이그레이션이 예상대로 작동된 경우 이를 확인합니다.

```
(overcloud)$ openstack server resize --confirm <instance>
```

<instance> 를 마이그레이션할 인스턴스의 이름 또는 **ID**로 바꿉니다. **"ACTIVE"** 상태는 인스턴스가 사용할 준비가 되었음을 나타냅니다.

- 마이그레이션이 예상대로 작동하지 않으면 이를 되돌립니다.

```
(overcloud)$ openstack server resize --revert <instance>
```

<instance> 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

4. 인스턴스를 다시 시작하십시오.

```
(overcloud)$ openstack server start <instance>
```

<instance> 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

5. 선택 사항: 유지보수를 위해 소스 컴퓨팅 노드를 비활성화한 경우 새 인스턴스를 할당할 수 있도록 노드를 다시 활성화해야 합니다.

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

<source> 를 소스 컴퓨팅 노드의 호스트 이름으로 바꿉니다.

16.3.5. 인스턴스 실시간 마이그레이션

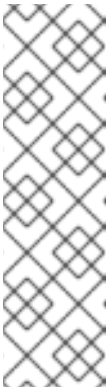
실시간 마이그레이션은 다운타임을 최소화하여 소스 컴퓨팅 노드에서 대상 컴퓨팅 노드로 인스턴스를 이동합니다. 실시간 마이그레이션이 모든 인스턴스에 적합하지 않을 수 있습니다. 자세한 내용은 마이그레이션 제한 조건을 참조하십시오.

절차

1. 인스턴스를 실시간 마이그레이션하려면 인스턴스 및 대상 컴퓨팅 노드를 지정합니다.

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait
```

- **<instance>** 를 인스턴스의 이름 또는 **ID**로 바꿉니다.
- **<dest>** 를 대상 컴퓨팅 노드의 이름 또는 **ID**로 바꿉니다.



참고

openstack server migrate 명령은 기본 공유 스토리지로 인스턴스를 마이그레이션하는 작업을 다룹니다. 로컬에 저장된 볼륨을 마이그레이션하려면 **--block-migration** 플래그를 지정합니다.

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host
<dest>] --wait --block-migration
```

2. 인스턴스가 마이그레이션 중인지 확인합니다.

```
(overcloud)$ openstack server show <instance>

+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| ...                  | ...                                  |
| status               | MIGRATING                            |
| ...                  | ...                                  |
+----------------------+--------------------------------------+
```

3. 마이그레이션이 완료될 때까지 기다립니다. 인스턴스 마이그레이션이 완료될 때까지 기다리는 동안 마이그레이션 상태를 확인할 수 있습니다. 자세한 내용은 마이그레이션 상태 확인 을 참조하십시오.

4. 인스턴스의 상태를 확인하여 마이그레이션에 성공했는지 확인합니다.

```
(overcloud)$ openstack server list --host <dest> --all-projects
```

<dest> 를 대상 컴퓨팅 노드의 이름 또는 **ID**로 바꿉니다.

5. 선택 사항: 유지보수를 위해 소스 컴퓨팅 노드를 비활성화한 경우 새 인스턴스를 할당할 수 있도록 노드를 다시 활성화해야 합니다.

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

<source> 를 소스 컴퓨팅 노드의 호스트 이름으로 바꿉니다.

16.3.6. 마이그레이션 상태 확인

마이그레이션이 완료되기 전에 마이그레이션에는 여러 상태 전환이 포함됩니다. 정상적인 마이그레이션 과정에서 마이그레이션 상태는 일반적으로 다음과 같이 전환됩니다.

1. 대기 중: 계산 서비스에서 인스턴스 마이그레이션 요청을 수락했으며 마이그레이션이 보류 중입니다.

2. 준비 중: 계산 서비스에서 인스턴스를 마이그레이션할 준비가 되어 있습니다.

3. 실행 중: 계산 서비스에서 인스턴스를 마이그레이션합니다.

4. 마이그레이션 후: 계산 서비스는 대상 컴퓨팅 노드에 인스턴스를 빌드했으며 소스 컴퓨팅 노드에 리소스를 해제하고 있습니다.

5. 완료됨: 계산 서비스는 인스턴스 마이그레이션을 완료하고 소스 컴퓨팅 노드에서 리소스 릴리스를 완료했습니다.

절차

1. 인스턴스의 마이그레이션 **ID** 목록을 검색합니다.

```
$ nova server-migration-list <instance>

+----+-------------+-----------  (...)
| Id | Source Node | Dest Node | (...)
+----+-------------+-----------+ (...)
| 2  | -           | -         | (...)
+----+-------------+-----------+ (...)
```

<instance> 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

2. 마이그레이션 상태를 표시합니다.

```
$ nova server-migration-show <instance> <migration_id>
```

- **<instance>** 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

- **<migration_id>** 를 마이그레이션 **ID**로 바꿉니다.

nova server-migration-show 명령을 실행하면 다음 예제 출력이 반환됩니다.

```
+------------------------+--------------------------------------+
| Property               | Value                                |
+------------------------+--------------------------------------+
| created_at             | 2017-03-08T02:53:06.000000           |
| dest_compute           | controller                           |
| dest_host              | -                                    |
| dest_node              | -                                    |
| disk_processed_bytes   | 0                                    |
| disk_remaining_bytes   | 0                                    |
| disk_total_bytes       | 0                                    |
| id                     | 2                                    |
| memory_processed_bytes | 65502513                             |
| memory_remaining_bytes | 786427904                            |
| memory_total_bytes     | 1091379200                           |
| server_uuid            | d1df1b5a-70c4-4fed-98b7-423362f2c47c |
| source_compute         | compute2                             |
| source_node            | -                                    |
| status                 | running                              |
| updated_at             | 2017-03-08T02:53:47.000000           |
+------------------------+--------------------------------------+
```

작은 정보

OpenStack 계산 서비스는 마이그레이션 진행 상황을 복사할 나머지 메모리 바이트 수만큼 측정합니다. 시간이 지남에 따라 이 수가 감소하지 않으면 마이그레이션을 완료할 수 없으며 계산 서비스에서 중단될 수 있습니다.

인스턴스 마이그레이션에 시간이 오래 걸리거나 오류가 발생하는 경우가 있습니다. 자세한 내용은 마이그레이션 문제 해결을 참조하십시오.

16.3.7. 인스턴스 비우기

Compute 노드에서 동일한 환경의 새 호스트로 인스턴스를 이동하거나 종료하려면 해당 인스턴스를 비울 수 있습니다.

비우기 프로세스에서는 원래 인스턴스를 삭제하고 원본 이미지, 인스턴스 이름, **UUID**, 네트워크 주소 및 원래 인스턴스가 할당된 기타 리소스를 사용하여 다른 컴퓨팅 노드에 다시 빌드합니다.

인스턴스에서 공유 스토리지를 사용하는 경우 대상 **Compute** 노드에서 디스크에 계속 액세스할 수 있으므로 비우기 프로세스 중에 인스턴스 루트 디스크가 다시 빌드되지 않습니다. 인스턴스가 공유 스토리지를 사용하지 않으면 인스턴스 루트 디스크도 대상 컴퓨팅 노드에 다시 빌드됩니다.

참고

- 컴퓨팅 노드가 펜싱된 경우에만 비우기를 수행할 수 있으며 **API**에서 컴퓨팅 노드의 상태가 "다운" 또는 "강제 종료"임을 보고합니다. 컴퓨팅 노드가 "다운" 또는 "강제 종료"로 보고되지 않으면 **evacuate** 명령이 실패합니다.
- 비우기를 수행하려면 클라우드 관리자여야 합니다.

16.3.7.1. 하나의 인스턴스 비우기

한 번에 하나씩 인스턴스를 비울 수 있습니다.

절차

1. 관리자로 실패한 컴퓨팅 노드에 로그인합니다.
2. 컴퓨팅 노드를 비활성화합니다.

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <host> <service> --disable
```

- **<host>** 를 인스턴스를 비울 컴퓨팅 노드의 이름으로 바꿉니다.
- **<service>** 를 비활성화할 서비스의 이름으로 바꿉니다(예: **nova-compute**).

3. 인스턴스를 비우려면 다음 명령을 입력합니다.

```
(overcloud)[stack@director ~]$ nova evacuate [--password <pass>] <instance> [<dest>]
```

- 비워진 인스턴스에 대해 설정하려면 **<pass>** 를 관리자 암호로 바꿉니다. 암호를 지정하지 않으면 비우기가 완료되면 임의 암호가 생성되고 출력됩니다.

- **<instance>** 를 비울 인스턴스의 이름 또는 **ID**로 바꿉니다.

- **<dest>** 를 인스턴스를 비울 컴퓨팅 노드의 이름으로 바꿉니다. 대상 컴퓨팅 노드를 지정하지 않으면 컴퓨팅 스케줄러에서 노드를 선택합니다. 다음 명령을 사용하여 가능한 컴퓨팅 노드를 찾을 수 있습니다.

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

16.3.7.2. 호스트의 모든 인스턴스 비우기

지정된 컴퓨팅 노드의 모든 인스턴스를 비울 수 있습니다.

절차

1. 관리자로 실패한 컴퓨팅 노드에 로그인합니다.

2. 컴퓨팅 노드를 비활성화합니다.

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <host> <service> --disable
```

- 인스턴스를 비울 컴퓨팅 노드의 이름으로 **<host>** 를 바꿉니다.

- **<service>** 를 비활성화할 서비스의 이름으로 바꿉니다(예: **nova-compute**).

3. 지정된 컴퓨팅 노드의 모든 인스턴스를 비웁니다.

```
(overcloud)[stack@director ~]$ nova host-evacuate [--target_host <dest>] <host>
```

- **<dest>** 를 인스턴스를 비우려면 대상 컴퓨팅 노드의 이름으로 교체합니다. 대상을 지정하지 않으면 **Compute** 스케줄러에서 대상을 선택합니다. 다음 명령을 사용하여 가능한 컴퓨팅 노드를 찾을 수 있습니다.

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

- 인스턴스를 비울 컴퓨팅 노드의 이름으로 **<host>** 를 바꿉니다.

16.3.8. 마이그레이션 문제 해결

인스턴스 마이그레이션 중에 다음과 같은 문제가 발생할 수 있습니다.

- 마이그레이션 프로세스에서 오류 발생
- 마이그레이션 프로세스가 종료되지 않음
- 마이그레이션 후 인스턴스의 성능이 저하됩니다.

16.3.8.1. 마이그레이션 중 오류

다음과 같은 문제가 발생하면 마이그레이션 작업이 **error** 상태로 전환합니다.

- 다른 버전의 **RHOSP(Red Hat OpenStack Platform)**로 클러스터 실행.
- 찾을 수 없는 인스턴스 **ID** 지정.
- 마이그레이션하려는 인스턴스가 **error** 상태입니다.
- **Compute** 서비스가 종료됨
- 경합 상태 발생
- 실시간 마이그레이션이 **failed** 상태로 전환

실시간 마이그레이션이 **failed** 상태로 전환된 후에는 일반적으로 **error** 상태가 됩니다. 다음과 같은 일반적인 문제로 인해 **failed** 상태가 발생할 수 있습니다.

- 대상 **Compute** 호스트를 사용할 수 없음
- 스케줄러 예외 발생
- 컴퓨팅 리소스가 부족하여 재빌드 프로세스 실패
- 서버 그룹 확인 실패
- 대상 컴퓨팅 노드로 마이그레이션하기 전에 소스 컴퓨팅 노드의 인스턴스가 삭제됩니다.

16.3.8.2. 끝나지 않는 실시간 마이그레이션

실시간 마이그레이션이 완료되지 않아 마이그레이션이 영구적으로 실행 중 상태가 됩니다. 실시간 마이그레이션이 완료되지 않는 일반적인 이유는 소스 컴퓨팅 노드에서 실행 중인 인스턴스에 대한 클라이언트 요청이 **Compute** 서비스에서 대상 컴퓨팅 노드에 복제하는 속도보다 빠르게 발생하는 변경 사항을 생성하기 때문입니다.

다음 방법 중 하나를 사용하여 이 상황을 해결하십시오.

- 실시간 마이그레이션 중지
- 실시간 마이그레이션 강제 완료

실시간 마이그레이션 중지

인스턴스 상태가 마이그레이션 프로세스에서 대상 노드에 복사하는 속도보다 빨리 변경되고 인스턴스 작업을 일시적으로 중단하지 않으려는 경우 실시간 마이그레이션을 중단할 수 있습니다.

절차

1. 인스턴스의 마이그레이션 목록을 검색합니다.

```
$ nova server-migration-list <instance>
```

<instance> 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

2. 실시간 마이그레이션을 중지합니다.

```
$ nova live-migration-abort <instance> <migration_id>
```

- **<instance>** 를 인스턴스의 이름 또는 **ID**로 바꿉니다.
- **<migration_id>** 를 마이그레이션 **ID**로 바꿉니다.

실시간 마이그레이션 강제 완료

인스턴스 상태가 마이그레이션 프로세스에서 대상 노드에 복사하는 속도보다 빠르게 변경되고 인스턴스 작업을 일시적으로 일시 중단하여 마이그레이션을 강제 완료하려는 경우 실시간 마이그레이션 절차를 강제 완료할 수 있습니다.

중요

실시간 마이그레이션을 강제 완료하면 상당한 다운 타임이 발생할 수 있습니다.

절차

1. 인스턴스의 마이그레이션 목록을 검색합니다.

```
$ nova server-migration-list <instance>
```

<instance> 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

2. 실시간 마이그레이션을 강제 완료합니다.

```
$ nova live-migration-force-complete <instance> <migration_id>
```

- **<instance>** 를 인스턴스의 이름 또는 **ID**로 바꿉니다.

- <migration_id> 를 마이그레이션 ID로 바꿉니다.

16.3.8.3. 마이그레이션 후 인스턴스 성능 저하

NUMA 토폴로지를 사용하는 인스턴스의 경우 소스 및 대상 Compute 노드에 동일한 NUMA 토폴로지 및 구성이 있어야 합니다. 대상 Compute 노드의 NUMA 토폴로지에는 사용 가능한 충분한 리소스가 있어야 합니다. 소스와 대상 Compute 노드 간의 NUMA 구성이 동일하지 않은 경우 인스턴스 성능이 저하되는 동안 실시간 마이그레이션이 성공할 수 있습니다. 예를 들어 소스 Compute 노드는 NIC 1을 NUMA 노드 0에 매핑하지만 대상 Compute 노드는 NIC 1을 NUMA 노드 5에 매핑하는 경우, 마이그레이션 후 인스턴스에서 트래픽을 NIC 1로 라우팅하기 위해 버스를 통해 첫 번째 CPU의 네트워크 트래픽을 NUMA 노드 5가 있는 두 번째 CPU로 라우팅할 수 있습니다. 이로 인해 예상된 동작이 발생할 수 있지만 성능이 저하될 수 있습니다. 마찬가지로 소스 Compute 노드의 NUMA 노드 0에 사용 가능한 CPU 및 RAM이 충분하지만 대상 Compute 노드의 NUMA 노드 0에 일부 리소스를 사용하는 인스턴스가 이미 있는 경우 인스턴스가 올바르게 실행되고 성능이 저하될 수 있습니다. 자세한 내용은 마이그레이션 제한 조건을 참조하십시오.