KVM을 사용하여 ASA 가상 구축

KVM(Kernel-based Virtual Machine)을 실행할 수 있는 모든 서버 클래스 x86 CPU 디바이스에 ASA 가상을 구축할 수 있습니다.

중요 ASA 가상의 최소 메모리 요구 사항은 2GB입니다. 현재 ASA 가상이 2GB 미만의 메모리로 실행되는 경우에는 ASA 가상 머신의 메모리를 늘리지 않고는 이전 버전에서 9.13(1) 이상으로 업그레이드할 수 없습니다. 최신 버전의 새 ASA 가상 머신을 재구축할 수도 있습니다.

KVM 지침 및 제한 사항에서의 ASA 가상

ASA 가상 구축에 사용되는 특정 하드웨어는 구축된 인스턴스 수 및 사용 요구 사항에 따라 달라질 수 있습니다. 생성하는 각 가상 어플라이언스는 호스트 머신에서 최소 리소스 할당(메모리, CPU 수 및 디스크 공간)을 필요로 합니다.

중요 ASA 가상는 8GB 디스크 스토리지 크기로 구축됩니다. 디스크 공간의 리소스 할당은 변경할 수 없습니다.

ASA 가상을 구축하기 전에 다음 지침 및 제한 사항을 검토하십시오.

KVM 시스템 요구 사항에서의 ASA 가상

최적의 성능을 보장하려면 아래 사양을 준수해야 합니다. ASA 가상에는 다음 요구 사항이 적용됩니다.

- 호스트 CPU는 가상화 확장을 사용하는 서버 클래스 x86 기반 Intel 또는 AMD CPU여야 합니다.

  예를 들어 ASA 가상 성능 테스트 랩에서는 2.6GHz에서 실행되는 Intel® Xeon® CPU E5-2690v4 프로세서를 사용하는 Cisco UCS®(Unified Computing System™) C 시리즈 M4 서버가 최소 요구 사항입니다.

권장 vNIC

최적의 성능을 위해 다음 vNIC를 사용하는 것이 좋습니다.

- PCI 패스스루의 i40e - 서버의 물리적 NIC를 VM 전용으로 지정하고 DMA(Direct Memory Access)를 통해 NIC와 VM 간에 패킷 데이터를 전송합니다. 패킷 이동에는 CPU 사이클이 필요하지 않습니다.
- i40evf/ixgbe-vf - 위와 사실상 동일하지만(NIC와 VM 간의 DMA 패킷) 여러 VM에서 NIC를 공유할 수 있습니다. 뛰어난 구축 유연성 때문에 일반적으로 SR-IOV를 선호합니다. 확인
- virtio - 10Gbps 작업을 지원하지만 CPU 사이클이 필요한 반가상화 네트워크 드라이버입니다.

참고 KVM 시스템에서 실행 중인 ASA 가상 인스턴스는 vNIC 드라이버 i40e 버전 2.11.25를 사용하는 SR-IOV 인터페이스에서 데이터 연결 문제가 발생할 수 있습니다. 이 문제를 해결하려면 vNIC 버전을 다른 버전으로 업그레이드하는 것이 좋습니다.

성능 최적화

ASA 가상에서 최상의 성능을 얻으려면 VM과 호스트를 모두 조정할 수 있습니다. 자세한 내용은 KVM의 ASA 가상에 대한 성능 조정, 10 페이지를 참조하십시오.

- **NUMA** - 게스트 VM의 CPU 리소스를 단일 NUMA(Non-Uniform Memory Access) 노드로 격리하면 ASA 가상 성능을 개선할 수 있습니다. 자세한 내용은 NUMA 지침, 11 페이지를 참조하십시오.
- **Receive Side Scaling** — ASA 가상는 RSS(Receive Side Scaling)를 지원합니다. RSS는 네트워크 수신 트래픽을 여러 프로세서 코어로 분산하기 위해 네트워크 어댑터에서 활용하는 기술입니다. 자세한 내용은 RSS(Receive Side Scaling)를 위한 다중 RX 대기열, 13 페이지를 참조하십시오.
- **VPN** 최적화 - ASA 가상을 사용한 VPN 성능 최적화를 위한 추가 고려 사항은 VPN 최적화, 15 페이지를 참조하십시오.

클러스터링

버전 9.17부터는 KVM에 구축된 ASA 가상 인스턴스에서 클러스터링이 지원됩니다. 자세한 내용은 ASAv용 ASA 클러스터를 참조하십시오.

CPU 고정

ASA 가상이 KVM 환경에서 작동하려면 CPU 고정이 필요합니다. CPU 피닝 활성화, 10 페이지를 참조하십시오.

고가용성을 위한 장애 조치 지침

장애 조치 구축의 경우, 대기 유닛에 동일한 라이선스 엔타이틀먼트가 있는지 확인합니다. 예를 들어 두 유닛 모두 2Gbps 엔타이틀먼트가 있어야 합니다.

중요 ASA 가상을 이용해 고가용성 쌍을 만들 때는 동일한 순서로 각 ASA 가상에 데이터 인터페이스를 추가해야 합니다. 각 ASA 가상에 동일한 인터페이스를 추가했지만 순서가 다른 경우 ASA 가상 콘솔에서 오류가 나타날 수 있습니다. 장애 조치 기능도 영향을 받을 수 있습니다.

Proxmox VE에서의 ASA 가상

Proxmox VE(Virtual Environment)는 KVM 가상 머신을 관리할 수 있는 오픈 소스 서버 가상화 플랫폼입니다. Proxmox VE는 웹 기반 관리 인터페이스도 제공합니다.

Proxmox VE에서 ASA 가상을 구축할 때는 에뮬레이션된 시리얼 포트를 갖도록 VM을 구성해야 합니다. 시리얼 포트가 없으면 ASA 가상은 부팅 프로세스 중에 루프에 빠집니니다. 모든 관리 작업은 Proxmox VE 웹 기반 관리 인터페이스를 사용하여 수행할 수 있습니다.

참고 Unix 쉘 또는 Windows Powershell에 익숙한 고급 사용자를 위해, Proxmox VE는 가상 환경의 모든 구성 요소를 관리할 수 있는 명령줄 인터페이스를 제공합니다. 이 명령줄 인터페이스는 지능형 탭 완성 기능과 UNIX 매뉴얼 페이지 형식의 전체 설명서를 제공합니다.

ASA 가상이 올바르게 부팅하려면 VM에 시리얼 디바이스가 구성되어 있어야 합니다.

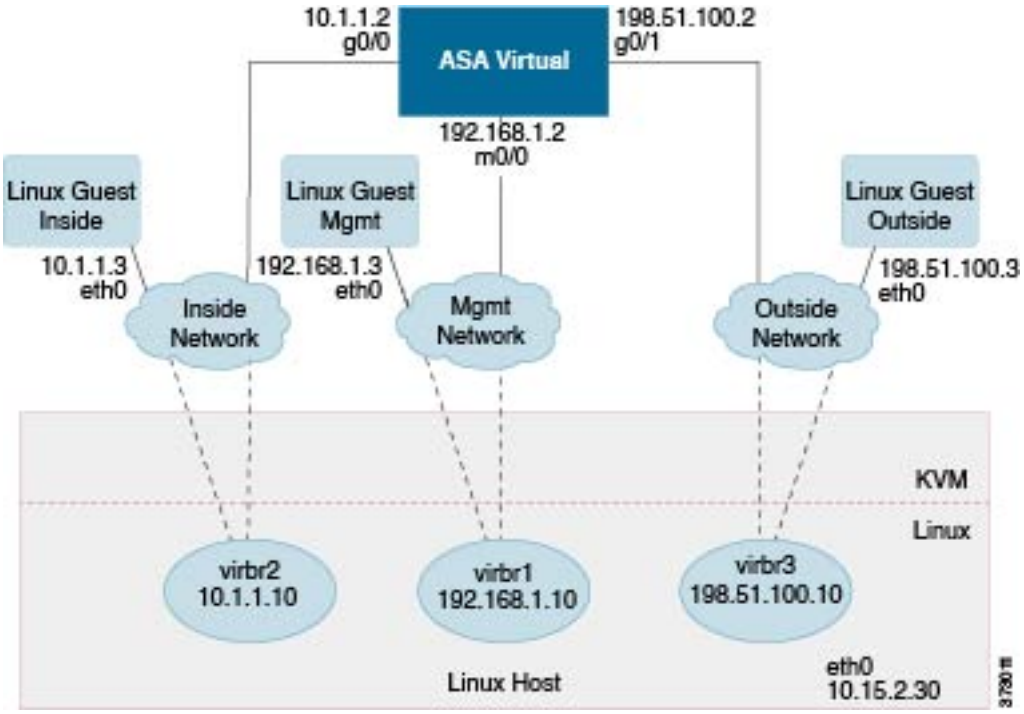
1. 기본 관리 센터의 왼쪽 탐색 트리에서 ASA 가상 머신을 선택합니다.
2. 가상 머신의 전원을 끕니다.
3. **Hardware(**하드웨어**)** > **Add(**추가**)** > **Network Device(**네트워크 디바이스**)**를 선택하고 시리얼 포트를 추가합니다.
4. 가상 머신을 켭니다.
5. Xterm.js를 사용하여 ASA 가상 머신에 액세스합니다.

게스트/서버에서 터미널을 설정하고 활성화하는 자세한 방법은 Proxmox Serial Terminal(시리얼 터미널) 페이지를 참조하십시오.

KVM을 사용한 ASA 가상 구축 정보

다음 그림에서는 ASA 가상 및 KVM을 사용하면 샘플 네트워크 토폴로지를 확인할 수 있습니다. 이 장에 설명된 절차는 샘플 토폴로지를 기반으로 합니다. ASA 가상는 내부 및 외부 네트워크 사이의 방화벽으로 작동합니다. 별도의 관리 네트워크도 구성됩니다.

그림 *1: KVM*을 사용하여 샘플 *ASA* 가상 구축



ASA 가상 및 KVM에 대한 사전 요건

- Cisco.com에서 ASA 가상 qcow2 파일을 다운로드하고 이를 Linux 호스트에 넣습니다.

  http://www.cisco.com/go/asa-software

참고 Cisco.com 로그인 및 Cisco 서비스 계약이 필요합니다.

- 이 문서에 나와 있는 샘플 구축의 경우, 사용자가 Ubuntu 18.04 LTS를 사용한다고 가정합니다. Ubuntu 18.04 LTS 호스트의 상위에 다음 패키지를 설치합니다.
  - qemu-kvm
  - libvirt-bin

- bridge-utils
- virt-manager
- virtinst
- virsh tools
- genisoimage

- 성능은 호스트 및 해당 컨피그레이션에 영향을 받습니다. 호스트를 조정하여 KVM에서 ASA 가상의 처리량을 극대화할 수 있습니다. 일반적인 호스트 조정 개념은 NFV가 Intel을 이용하여 전달하는 패킷 처리 성능을 참조하십시오.
- Ubuntu 18.04에 유용한 최적화는 다음과 같습니다.
  - macvtap - 고성능 Linux 브리지로, Linux 브리지 대신 macvtap을 사용할 수 있습니다. Linux 브리지 대신 macvtap을 사용하려면 특정 설정을 구성해야 합니다.
  - Transparent Huge Pages - 메모리 페이지 크기를 늘리며 18.04에서 기본적으로 설정됩니다.

    Hyperthread 비활성화 - 두 개의 vCPU를 단일 코어로 줄입니다.
  - txqueuelength - 기본 txqueuelength를 4000 패킷으로 늘이고 삭제율을 줄입니다.
  - 고정 - qemu 및 vhost 프로세스를 특정 CPU 코어에 고정합니다. 특정 조건에서 고정 기능을 사용하면 성능이 대폭 향상됩니다.
- RHEL 기반 배포 최적화에 대한 자세한 내용은 Red Hat Enterprise Linux 7 가상화 조정 및 최적화 가이드를 참조하십시오.
- ASA 소프트웨어 및 ASA 가상 하이퍼바이저 호환성에 대해서는 Cisco Secure Firewall ASA 호환성을 참조하십시오.

Day 0 컨피그레이션 파일 준비

ASA 가상를 실행하기 전에 Day 0 구성 파일을 준비할 수 있습니다. 이 파일은 ASA 가상을 시작할 때 적용한 ASA 가상 컨피그레이션이 포함된 텍스트 파일입니다. 이 초기 컨피그레이션은 사용자가 선택하는 작업 디렉토리의 “day0-config”라는 이름의 텍스트 파일에 위치하며, 이 파일은 최초 부팅 시 마운트되고 읽히는 day0.iso 파일로 조작됩니다. Day 0 컨피그레이션 파일에는 최소한 관리 인터페이스를 활성화하고 공용 키 인증용 SSH 서버를 설정하는 명령이 포함되어야 할 뿐만 아니라, 완전한 ASA 컨피그레이션도 포함되어야 합니다.

최초 부팅 동안 day0.iso 파일(사용자 정의 day0.iso 또는 기본 day0.iso)을 사용할 수 있어야 합니다.

- 초기 구축 동안 ASA 가상 라이센스를 자동으로 적용하려면, Cisco Smart Software Manager에서 다운로드한 Smart Licensing ID(Identity) Token을 Day 0 컨피그레이션 파일과 같은 디렉토리에 있는 ‘idtoken’이라는 이름의 텍스트 파일로 가져옵니다.

- 가상 VGA 콘솔 대신 하이퍼바이저의 시리얼 포트에서 ASA 가상에 액세스하고 구성하려면, Day 0 컨피그레이션 파일에 콘솔 시리얼 설정을 포함하여 첫 부팅 시 시리얼 포트를 사용해야 합니다.
- 투명 모드에서 ASA 가상을 구축하려는 경우, 투명 모드에서 실행 중인 알려진 ASA 컨피그레이션 파일을 Day 0 컨피그레이션 파일로 사용해야 합니다. 이 사항은 라우팅 방화벽용 Day 0 컨피그레이션 파일에는 적용되지 않습니다.

참고 이 예에서는 Linux를 사용하지만 Windows의 경우에도 유사한 유틸리티가 있습니다.

단계 **1** "day0 config"라는 텍스트 파일에 ASA 가상에 대한 CLI 컨피그레이션을 입력합니다. 3개의 인터페이스에 대한 인터페이스 컨피그레이션 및 원하는 기타 모든 컨피그레이션을 추가합니다.

첫 줄은 ASAv 버전으로 시작해야 합니다. day0-config는 유효한 ASA 컨피그레이션이어야 합니다. day0-config를 생성하는 가장 좋은 방법은 기존 ASA 또는 ASA 가상에서 실행 중인 컨피그레이션 중 관련 부분을 복사하는 것입니다. day0-config에서의 줄의 순서가 중요하며 기존 **show running-config** 명령 출력의 순서와 일치해야 합니다.

예제:

```
ASA Version 9.4.1
!
console serial
interface management0/0
nameif management
security-level 100
ip address 192.168.1.2 255.255.255.0
no shutdown
interface gigabitethernet0/0
nameif inside
security-level 100
ip address 10.1.1.2 255.255.255.0
no shutdown
interface gigabitethernet0/1
nameif outside
security-level 0
ip address 198.51.100.2 255.255.255.0
no shutdown
http server enable
http 192.168.1.0 255.255.255.0 management
crypto key generate rsa modulus 1024
username AdminUser password paSSw0rd
ssh 192.168.1.0 255.255.255.0 management
aaa authentication ssh console LOCAL
```

단계 **2** (선택 사항) 초기 ASA 가상 구축 동안 라이센싱이 자동으로 이루어진 경우, day0-config 파일에 다음 정보가 포함되어 있는지 확인합니다.

- 관리 인터페이스 IP 주소
- (선택 사항) Smart Licensing에 사용할 HTTP 프록시
- HTTP 프록시(지정된 경우) 또는 tools.cisco.com에 대한 연결을 지원하는 **route** 명령

- tools.cisco.com을 IP 주소에 확인하는 DNS 서버
- 사용자가 요청하는 ASA 가상 라이센스를 지정하는 Smart Licensing 컨피그레이션
- (선택 사항) ASA 가상이 CSSM에서 검색을 더욱 쉽게 수행할 수 있도록 하는 고유한 호스트 이름

단계 **3** (선택 사항) Cisco Smart Software Manager에서 발급한 Smart License ID 토큰 파일을 컴퓨터에 다운로드하고, 다운로드 파일에서 ID 토큰을 복사한 다음 ID 토큰만 포함하는 'idtoken'이라는 텍스트 파일에 붙여넣습니다.

단계 **4** 텍스트 파일을 ISO 파일로 전환하여 가상 CD-ROM을 생성합니다.

예제:

```
stack@user-ubuntu:-/KvmAsa$ sudo genisoimage -r -o day0.iso day0-config idtoken
I: input-charset not specified, using utf-8 (detected in locale settings)
Total translation table size: 0
Total rockridge attributes bytes: 252
Total directory bytes: 0
Path table size (bytes): 10
Max brk space used 0
176 extents written (0 MB)
stack@user-ubuntu:-/KvmAsa$
```

ID 토큰은 ASA 가상을 Smart Licensing 서버에 자동으로 등록합니다.

단계 **5** 1단계~5단계를 반복하여 구축하려는 각 ASA 가상에 대해 적절한 IP 주소가 포함된 별도의 기본 컨피그레이션 파일을 만듭니다.

가상 브리지 XML 파일 준비

ASA 가상 게스트를 KVM 호스트에 연결하고 게스트를 서로 연결하는 가상 네트워크를 설정해야 합니다.

참고 이 절차는 KVM 호스트 밖의 외부 환경에 대한 연결을 설정하지 않습니다.

KVM 호스트에서 가상 브리지 XML 파일을 준비합니다. Day 0 컨피그레이션 파일 준비, 5 페이지에 설명된 샘플 가상 네트워크 토폴로지 경우, virbr1.xml, virbr2.xml, virbr3.xml이라는 3개의 가상 브리지 파일이 필요합니다(이러한 3개의 파일 이름을 사용해야 함. 예를 들어, virbr0은 이미 존재하므로 사용할 수 없음). 각 파일에는 가상 브리지를 설정하는 데 필요한 정보가 포함되어 있습니다. 가상 브리지에 이름과 고유한 MAC 주소를 제공해야 합니다. IP 주소를 제공하는 것은 선택 사항입니다.

단계 **1** 가상 네트워크 브리지 XML 파일 3개를 만듭니다. 예: virbr1.xml, virbr2.xml, virbr3.xml:

예제:

```
<network>
<name>virbr1</name>
<bridge name='virbr1' stp='on' delay='0' />
```

```
<mac address='52:54:00:05:6e:00' />
<ip address='192.168.1.10' netmask='255.255.255.0' />
</network>
```

예제:

```
<network>
<name>virbr2</name>
<bridge name='virbr2' stp='on' delay='0' />
<mac address='52:54:00:05:6e:01' />
<ip address='10.1.1.10' netmask='255.255.255.0' />
</network>
```

예제:

```
<network>
<name>virbr3</name>
<bridge name='virbr3' stp='on' delay='0' />
<mac address='52:54:00:05:6e:02' />
<ip address='198.51.100.10' netmask='255.255.255.0' />
</network>
```

단계 **2** 다음 정보가 포함된 스크립트를 만듭니다(이 예에서는 스크립트 이름을 virt_network_setup.sh로 지정함).

```
virsh net-create virbr1.xml
virsh net-create virbr2.xml
virsh net-create virbr3.xml
```

단계 **3** 이 스크립트를 실행하여 가상 네트워크를 설정합니다. 이 스크립트는 가상 네트워크를 불러옵니다. 네트워크는 KVM 호스트가 실행되는 동안 계속 가동됩니다.

```
stack@user-ubuntu:-/KvmAsa$ virt_network_setup.sh
```

참고 Linux 호스트를 다시 로드할 경우, virt_network_setup.sh 스크립트를 다시 실행해야 합니다. 재부팅되면 스크립트가 지속되지 않습니다.

단계 **4** 가상 네트워크가 만들어졌는지 확인합니다.

```
stack@user-ubuntu:-/KvmAsa$ brctl show
bridge name bridge id STP enabled Interfaces
virbr0 8000.0000000000000 yes
virbr1 8000.5254000056eed yes virb1-nic
virbr2 8000.5254000056eee yes virb2-nic
virbr3 8000.5254000056eec yes virb3-nic
stack@user-ubuntu:-/KvmAsa$
```

단계 **5** virbr1 브리지에 할당된 IP 주소가 표시됩니다. 이는 XML 파일에 할당한 IP 주소입니다.

```
stack@user-ubuntu:-/KvmAsa$ ip address show virbr1
S: virbr1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
link/ether 52:54:00:05:6e:00 brd ff:ff:ff:ff:ff:ff
inet 192.168.1.10/24 brd 192.168.1.255 scope global virbr1
valid_lft forever preferred_lft forever
```

실행 ASA 가상

virt-install 기반 구축 스크립트를 사용하여 ASA 가상을 시작합니다.

단계 **1** "virt_install_asav.sh"라는 virt-install 스크립트를 만듭니다.

ASA 가상 머신의 이름은 이 KVM 호스트의 모든 기타 VM에서 고유해야 합니다.

ASA 가상은 최대 10개의 네트워크를 지원합니다. 이 예에서는 3개의 네트워크를 사용합니다. 네트워크 브리지 절의 순서가 중요합니다. 첫 번째 줄의 항목은 항상 ASA 가상의 관리 인터페이스(Management 0/0)이고, 두 번째 줄의 항목은 ASA 가상의 GigabitEthernet 0/0이며, 세 번째 줄의 항목은 ASA 가상의 GigabitEthernet 0/1입니다. 이런 식으로 GigabitEthernet 0/8까지 이어집니다. 가상 NIC는 Virtio여야 합니다.

예제:

```
virt-install \
--connect=qemu:///system \
--network network=default,model=virtio \
--network network=default,model=virtio \
--network network=default,model=virtio \
--name=asav \
--cpu host \
--arch=x86_64 \
--machine=pc-1.0 \
--vcpus=1 \
--ram=2048 \
--os-type=linux \
--virt-type=kvm \
--import \
--disk path=/home/kvmperf/Images/desmo.qcow2,format=qcow2,device=disk,bus=virtio,cache=none \
--disk path=/home/kvmperf/asav_day0.iso,format=iso,device=cdrom \
--console pty,target_type=virtio \
--serial tcp,host=127.0.0.1:4554,mode=bind,protocol=telnet
```

단계 **2** virt_install 스크립트를 실행합니다.

예제:

```
stack@user-ubuntu:-/KvmAsa$ ./virt_install_asav.sh

Starting install...
Creating domain...
```

VM의 콘솔을 표시하는 창이 나타납니다. VM이 부팅 중인 것으로 표시됩니다. VM이 부팅될 때까지 몇 분 정도 소요됩니다. VM이 부팅을 멈추면 콘솔 화면에서 CLI 명령을 발급할 수 있습니다.

KVM의 ASA 가상에 대한 성능 조정

KVM 컨피그레이션의 성능 향상

KVM 호스트의 설정을 변경하여 KVM 환경에서 ASA 가상의 성능을 높일 수 있습니다. 이러한 설정은 호스트 서버의 컨피그레이션 설정과 무관합니다. 이 옵션은 Red Hat Enterprise Linux 7.0 KVM에서 사용할 수 있습니다.

CPU 고정을 활성화하여 KVM 컨피그레이션에서 성능을 개선할 수 있습니다.

CPU 피닝 활성화

ASA 가상을 사용하려면 KVM CPU 선호도 옵션을 사용하여 KVM 환경에서 ASA 가상 성능을 높여야 합니다. 프로세서 선호도 또는 CPU 고정을 사용하면 프로세스 또는 스레드를 CPU(중앙 처리 장치) 또는 CPU 범위에 바인딩하거나 바인딩 해제하여, 프로세스나 스레드가 아무 CPU가 아닌 지정된 CPU에서만 실행되게 할 수 있습니다.

고정되지 않은 인스턴스가 고정된 인스턴스의 리소스 요구 사항을 사용하지 못하다록, CPU 고정을 사용하는 인스턴스를 CPU 고정을 사용하지 않는 인스턴스와 다른 호스트에 구축하도록 호스트 집계를 구성하십시오.

주의 NUMA 토폴로지가 없는 인스턴스와 동일한 호스트에 NUMA 토폴로지가 있는 인스턴스를 구축하지 마십시오.

이 옵션을 사용하려면 KVM 호스트에서 CPU 고정을 구성합니다.

단계 **1** KVM 호스트 환경에서 호스트 토폴로지를 확인하여 고정에 사용할 수 있는 vCPU 수를 확인합니다.

예제:

```
virsh nodeinfo
```

단계 **2** 사용 가능한 vCPU 수를 확인합니다.

예제:

```
virsh capabilities
```

단계 **3** 프로세서 코어 집합에 vCPU를 고정합니다.

예제:

```
virsh vcpupin <vm-name> <vcpu-number> <host-core-number>
```

virsh vcpupin 명령은 ASA 가상의 각 vCPU를 대상으로 실행해야 합니다. 다음 예는 ASA 가상 컨피그레이션에 vCPU 4개가 있고 호스트에 8개의 코어가 있는 경우 필요한 KVM 명령을 보여줍니다.

```
virsh vcpupin asav 0 2
virsh vcpupin asav 1 3
```

```
virsh vcpupin asav 2 4
virsh vcpupin asav 3 5
```

호스트 코어 숫자는 0~7일 수 있습니다. 자세한 내용은 KVM 설명서를 참조하십시오.

참고 CPU 고정을 구성할 때는 호스트 서버의 CPU 토폴로지를 신중하게 고려하십시오. 여러 코어로 구성된 서버를 사용하는 경우, 여러 소켓에서 CPU 고정을 구성하지 마십시오.

KVM 컨피그레이션에서의 성능 개선은 전용 시스템 리소스가 필요하다는 단점이 있습니다.

NUMA 지침

NUMA(Non-Uniform Memory Access)는 멀티프로세서 시스템의 프로세서와 관련한 기본 메모리 모듈의 배치를 설명하는 공유 메모리 아키텍처입니다. 프로세서가 자체 노드 내에 있지 않은 메모리(원격 메모리)에 액세스하는 경우, 데이터를 로컬 메모리에 액세스할 때보다 느린 속도로 NUMA 연결을 통해 전송해야 합니다.

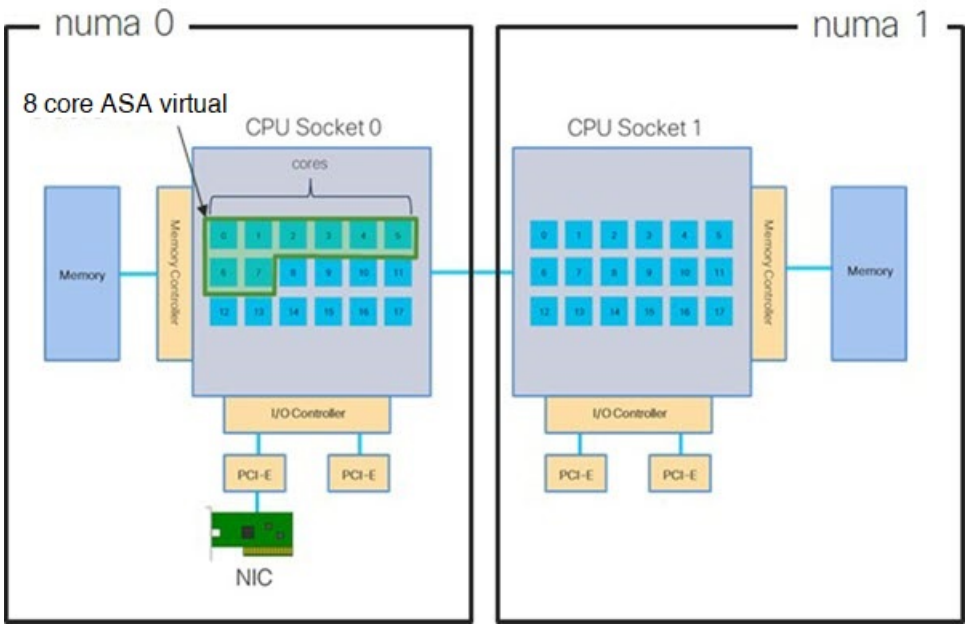
x86 서버 아키텍처는 여러 소켓 및 단일 소켓 내의 여러 코어로 구성됩니다. 각 CPU 소켓과 소켓의 메모리 및 I/O를 NUMA 노드라고 합니다. 메모리에서 패킷을 효율적으로 읽으려면 게스트 애플리케이션 및 관련 주변 장치(예: NIC)가 동일한 노드 내에 있어야 합니다.

최적의 ASA 가상 성능을 얻으려면 다음 조건을 충족해야 합니다.

- ASA 가상 머신은 단일 NUMA 노드에서 실행해야 합니다. 단일 ASA 가상이 소켓 2개에서 실행되도록 구축되면 성능이 크게 저하됩니다.
- 8코어 ASA 가상(그림 2: 8코어 ASA 가상 NUMA 아키텍처 예시, 12 페이지)을 사용하려면 호스트 CPU의 각 소켓에 최소 8개의 코어가 있어야 합니다. 서버에서 실행 중인 다른 VM도 고려해야 합니다.
- 16코어 ASA 가상(그림 3: 16코어 ASA 가상 NUMA 아키텍처 예시, 12 페이지)을 사용하려면 호스트 CPU의 각 소켓에 최소 16개의 코어가 있어야 합니다. 서버에서 실행 중인 다른 VM도 고려해야 합니다.
- NIC는 ASA 가상 머신과 동일한 NUMA 노드에 있어야 합니다.

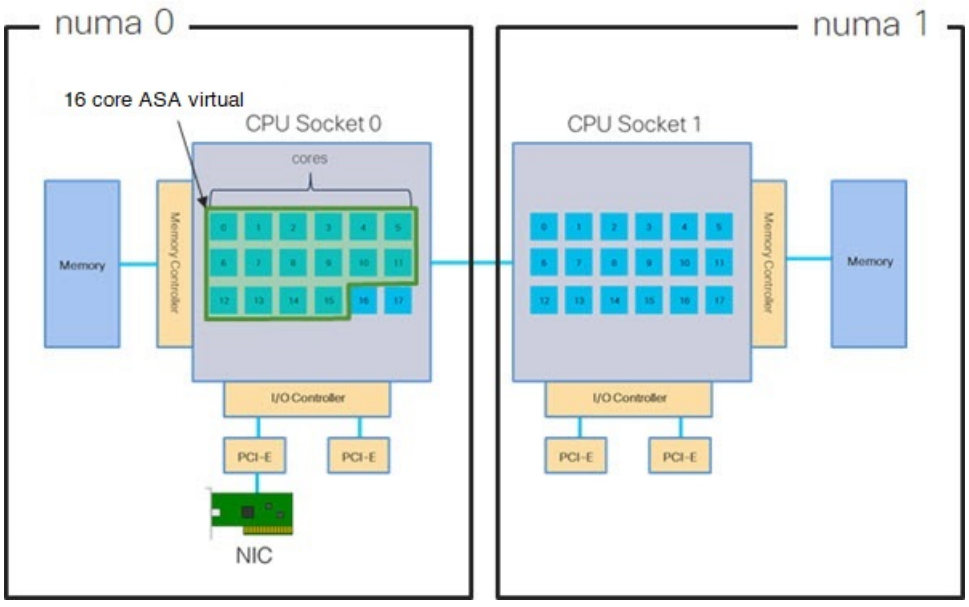
다음 그림에서는 CPU 소켓 2개가 있으며 각 CPU에 코어 18개가 있는 서버를 확인할 수 있습니다. 8코어 ASA 가상을 사용하려면 호스트 CPU의 각 소켓에 최소 8개의 코어가 있어야 합니다.

그림 *2: 8*코어 *ASA* 가상 *NUMA* 아키텍처 예시



다음 그림에서는 CPU 소켓 2개가 있으며 각 CPU에 코어 18개가 있는 서버를 확인할 수 있습니다. 16코어 ASA 가상을 사용하려면 호스트 CPU의 각 소켓에 최소 8개의 코어가 있어야 합니다.

그림 *3: 16*코어 *ASA* 가상 *NUMA* 아키텍처 예시



NUMA 최적화

NIC가 실행 중인 것과 동일한 numa 노드에서 ASA 가상 머신을 실행하는 것이 가장 좋습니다. 이렇게 하려면 다음을 수행합니다.

1. 노드의 다이어그램을 표시하려면 "lstopo"를 사용하여 NIC가 있는 노드를 결정합니다. NIC를 찾아 연결된 노드를 기록해 둡니다.
2. KVM 호스트에서 `virsh 목록`을 사용하여 ASA 가상을 찾습니다.
3. `virsh edit <VM Number>`를 이용해 VM을 편집합니다.
4. 선택한 노드에서 ASA 가상을 정렬합니다. 다음 예에서는 18 코어 노드를 가정합니다.

   노드 0에 정렬:

   ```
   <vcpu placement='static' cpuset='0-17'>16</vcpu>
   <numatune>
       <memory mode='strict' nodeset='0'/>
   </numatune>
   ```

   노드 1에 정렬:

   ```
   <vcpu placement='static' cpuset='18-35'>16</vcpu>
   <numatune>
       <memory mode='strict' nodeset='1'/>
   </numatune>
   ```

5. .xml 변경 사항을 저장하고 ASA 가상 머신 전원을 껐다가 켭니다.
6. 원하는 노드에서 VM이 실행 중인지 확인하려면 `ps aux | grep<name of your ASAv VM>`을 실행하여 프로세스 ID를 가져옵니다.
7. `sudo numastat -c <ASAv VM Process ID>`를 실행하여 ASA 가상 머신이 올바르게 정렬되었는지 확인합니다.

KVM를 이용한 NUMA 조정에 관한 자세한 내용은 RedHat 문서 9.3. libvirt NUMA 조정에서 확인할 수 있습니다.

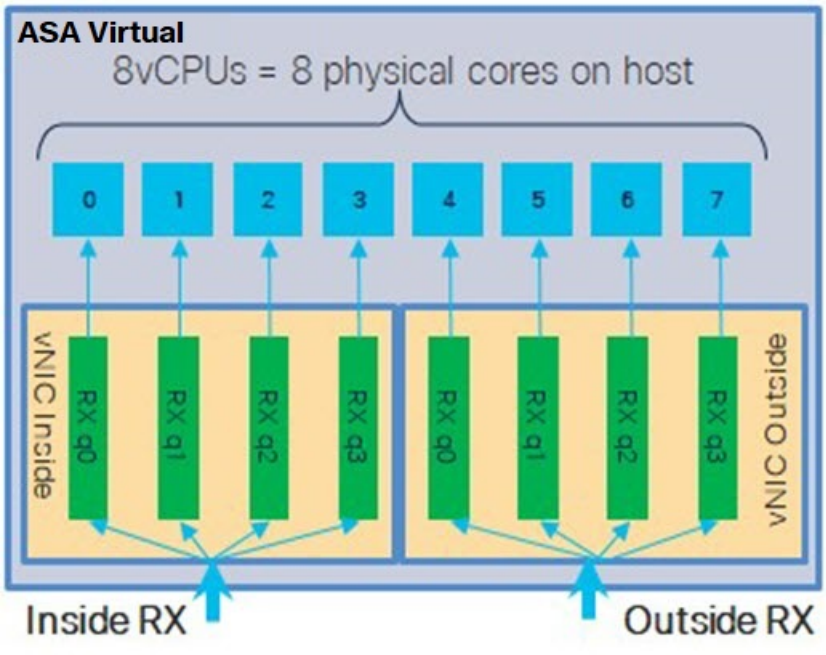
RSS(Receive Side Scaling)를 위한 다중 RX 대기열

ASA 가상은 RSS(Receive Side Scaling)를 지원합니다. RSS는 네트워크 수신 트래픽을 여러 프로세서 코어에 병렬로 분산하기 위해 네트워크 어댑터에서 활용하는 기술입니다. 최대 처리량을 확보하려면 각 vCPU(코어)에 자체 NIC RX 대기열이 있어야 합니다. 일반적인 RA VPN 구축에서는 단일 내부/외부 인터페이스 쌍을 사용합니다.

중요 여러 RX 대기열을 사용하려면 ASA 가상 버전 9.13(1) 이상이 필요합니다. KVM의 경우 *libvirt* 버전이 1.0.6 이상이어야 합니다.

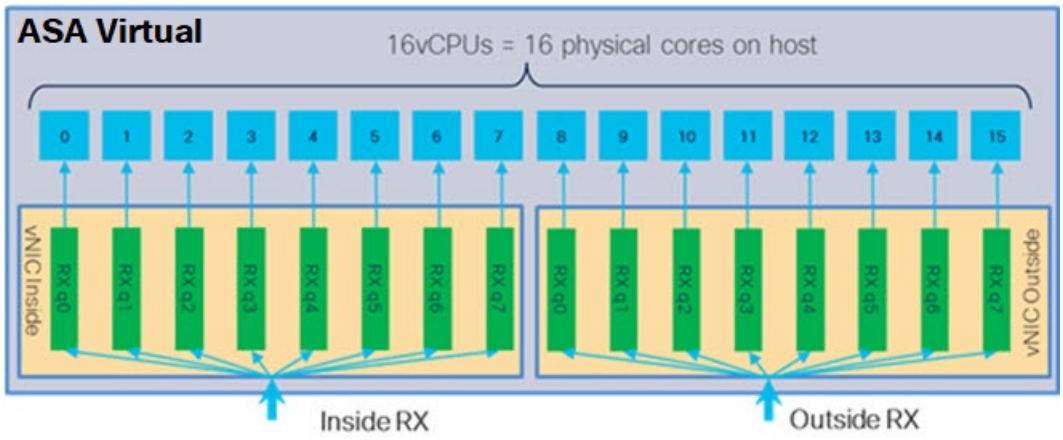
내부/외부 인터페이스 쌍이 있는 8코어 VM의 경우, 각 인터페이스에는 그림 4: 8코어 ASA 가상 RSS RX 대기열, 14 페이지에서처럼 RX 대기열 4개가 있습니다.

그림 *4: 8*코어 *ASA* 가상 *RSS RX* 대기열



내부/외부 인터페이스 쌍이 있는 16코어 VM의 경우, 각 인터페이스에는 그림 5: 16코어 ASA 가상 RSS RX 대기열, 14 페이지에서처럼 RX 대기열 8개가 있습니다.

그림 *5: 16*코어 *ASA* 가상 *RSS RX* 대기열



다음 표에는 KVM에 대한 의 vNIC 및 지원되는 RX 대기열 수가 나와 있습니다.ASA 가상 지원되는 vNIC에 대한 설명은 권장 vNIC, 2 페이지를 참조하십시오.

표 *1: KVM* 권장 *NIC/vNIC*

NIC카드	vNIC 드라이버	드라이버 기술	RX 대기열 수	성능
x710	i40e	PCI 패스스루	최대 8	x710의 PCI 통과 및 SR-IOV 모드는 최상의 성능을 제공합니다. NIC는 여러 VM에서 공유할 수 있으므로 가상 구축에는 일반적으로 SR-IOV가 선호됩니다.
	i40evf	SR-IOV	8	
x520	ixgbe	PCI 패스스루	6	x520 NIC는 x710보다 성능이 10~30% 낮습니다. x520의 PCI 통과 및 SR-IOV 모드는 유사한 성능을 제공합니다. NIC는 여러 VM에서 공유할 수 있으므로 가상 구축에는 일반적으로 SR-IOV가 선호됩니다.
	ixgbe-vf	SR-IOV	2	
해당 없음	virtio	반가상화	최대 8	ASAv100에는 권장되지 않습니다. 다른 구축에 대해서는 KVM에서 Virtio에 대한 다중 대기열 지원 활성화, 15 페이지를 참조하십시오.

KVM에서 **Virtio**에 대한 다중 대기열 지원 활성화

다음 예에서는 virsh를 사용하여 libvirt xml을 편집하여 Virtio NIC RX 대기열 수를 4로 구성하는 방법을 보여 줍니다.

```
<interface type='bridge'>
 <mac address='52:54:00:43:6e:3f'/>
 <source bridge='clients'/>
 <model type='virtio'/>
 <driver name='vhost' queues='4'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</interface>
```

중요 여러 RX 대기열을 지원하려면 *libvirt* 버전이 1.0.6 이상이어야 합니다.

VPN 최적화

다음은 ASA 가상을 사용하여 VPN 성능을 최적화하기 위한 몇 가지 추가 고려 사항입니다.

- IPSec은 DTLS보다 처리량이 높습니다.
- 암호 - GCM의 처리량은 CBC의 약 2배입니다.

SR-IOV 인터페이스 프로비저닝

SR-IOV를 사용하면 여러 VM이 호스트 내에서 단일 PCIe 네트워크 어댑터를 공유할 수 있습니다. SR-IOV는 다음 기능을 정의합니다.

- PF(물리적 기능) - PF는 SR-IOV 기능을 포함하는 전체 PCIe 기능입니다. 이러한 기능은 호스트 서버에서 일반 고정 NIC로 표시됩니다.
- VF(가상 기능) - VF는 데이터 전송을 지원하는 경량 PCIe 기능입니다. VF는 PF에서 파생되어 관리됩니다.

VF는 가상화된 운영 체제 프레임워크 내에서 ASA 가상 머신에 최대 10Gbps의 연결을 제공할 수 있습니다. 이 섹션에서는 KVM 환경에서 VF를 구성하는 방법을 설명합니다. ASA 가상에서의 SR-IOV 지원은 ASA 가상 및 SR-IOV 인터페이스 프로비저닝에서 설명합니다.

SR-IOV 인터페이스 프로비저닝 요구 사항

SR-IOV를 지원하는 물리적 NIC가 있다면 SR-IOV가 활성화된 VF 또는 vNIC(Virtual NIC)를 ASA 가상 인스턴스에 연결할 수 있습니다. 또한 SR-IOV는 BIOS 및 하드웨어에서 실행 중인 운영 체제 인스턴스/하이퍼바이저에서의 지원을 필요로 합니다. 다음은 KVM 환경에서 실행되는 ASA 가상의 SR-IOV 인터페이스 프로비저닝에 대한 일반 지침 목록입니다.

- 호스트 서버에 SR-IOV을 지원하는 물리적 NIC가 있어야 합니다. SR-IOV 인터페이스에 대한 지침 및 제한 사항를 참조하십시오.
- 호스트 서버의 BIOS에서 가상화를 활성화해야 합니다. 자세한 내용은 벤더 설명서를 참조하십시오.
- 호스트 서버의 BIOS에서 SR-IOV에 대한 IOMMU 전역 지원을 활성화해야 합니다. 자세한 내용은 하드웨어 벤더 설명서를 참조하십시오.

KVM 호스트 BIOS 및 호스트 OS 수정

이 섹션에서는 KVM 머신에서 SR-IOV 인터페이스를 프로비저닝하는 다양한 설정 및 구성 단계를 설명합니다. 이 섹션의 정보는 Intel Ethernet 서버 어댑터 X520-DA2를 사용하는 Cisco UCS C 시리즈 서버에서 Ubuntu 14.04를 사용하는 특정 랩 환경의 디바이스에서 생성되었습니다.

시작하기 전에

- SR-IOV 호환 NIC(네트워크 인터페이스 카드)가 설치되어 있는지 확인합니다.
- Intel VT-x(Virtualization Technology) 및 VT-d 기능이 활성화되어 있는지 확인합니다.

참고 일부 시스템 제조업체는 기본적으로 이러한 확장을 비활성화합니다. 시스템마다 BIOS 설정에 액세스하고 변경하는 방법이 다르므로, 벤더 설명서를 참조하여 프로세스를 확인하는 것이 좋습니다.

- 운영 체제를 설치하는 동안 모든 Linux KVM 모듈, 라이브러리, 사용자 툴 및 유틸리티가 설치되었는지 확인합니다. ASA 가상 및 KVM에 대한 사전 요건, 4 페이지를 참조하십시오.
- 물리적 인터페이스가 UP(작동) 상태인지 확인합니다. ifconfig <*ethname*>을 사용하여 확인합니다.

단계 **1** "root" 사용자 계정 및 비밀번호를 사용하여 시스템에 로그인합니다.

단계 **2** Intel VT-d가 활성화되어 있는지 확인합니다.

예제:

```
kvmuser@kvm-host:/$ dmesg | grep -e DMAR -e IOMMU
[    0.000000] ACPI: DMAR 0x000000006F9A4C68 000140 (v01 Cisco0 CiscoUCS 00000001 INTL 20091013)
[    0.000000] DMAR: IOMMU enabled
```

마지막 줄은 VT-d가 활성화되었음을 나타냅니다.

단계 **3** *intel_iommu=on* 매개변수를 */etc/default/grub* 컨피그레이션 파일의 GRUB_CMDLINE_LINUX 항목에 추가하여 커널에서 Intel VT-d를 활성화합니다.

예제:

```
# vi /etc/default/grub
...
GRUB_CMDLINE_LINUX="nofb splash=quiet console=tty0 ... intel_iommu=on"
...
```

참고 AMD 프로세서를 사용하는 경우 대신 부팅 매개변수에 *amd_iommu=on*을 추가합니다.

단계 **4** iommu 변경 사항을 적용하려면 서버를 재부팅합니다.

예제:

```
> shutdown -r now
```

단계 **5** 다음 형식을 사용하여 *sysfs* 인터페이스를 통해 *sriov_numvfs* 매개변수에 적절한 값을 작성하여 VF를 생성합니다.

```
#echo n > /sys/class/net/device name/device/sriov_numvfs
```

서버의 전원을 껐다가 켤 때마다 원하는 수의 VF가 생성되게 하려면 */etc/rc.d/* 디렉터리에 있는 *rc.local* 파일에 위의 명령을 추가합니다. Linux OS는 부팅 프로세스가 끝날 때 *rc.local* 스크립트를 실행합니다.

예를 들어 다음은 포트당 하나의 VF를 생성하는 방법을 보여줍니다. 인터페이스는 설정에 따라 다릅니다.

예제:

```
echo '1' > /sys/class/net/eth4/device/sriov_numvfs
echo '1' > /sys/class/net/eth5/device/sriov_numvfs
echo '1' > /sys/class/net/eth6/device/sriov_numvfs
echo '1' > /sys/class/net/eth7/device/sriov_numvfs
```

단계 **6** 서버를 재부팅합니다.

예제:

```
> shutdown -r now
```

단계 **7** *lspci*를 사용하여 VF가 생성되었는지 확인합니다.

예제:

```
 > lspci | grep -i "Virtual Function"
kvmuser@kvm-racetrack:~$ lspci | grep -i "Virtual Function"
0a:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
0a:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
0a:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
0a:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
```

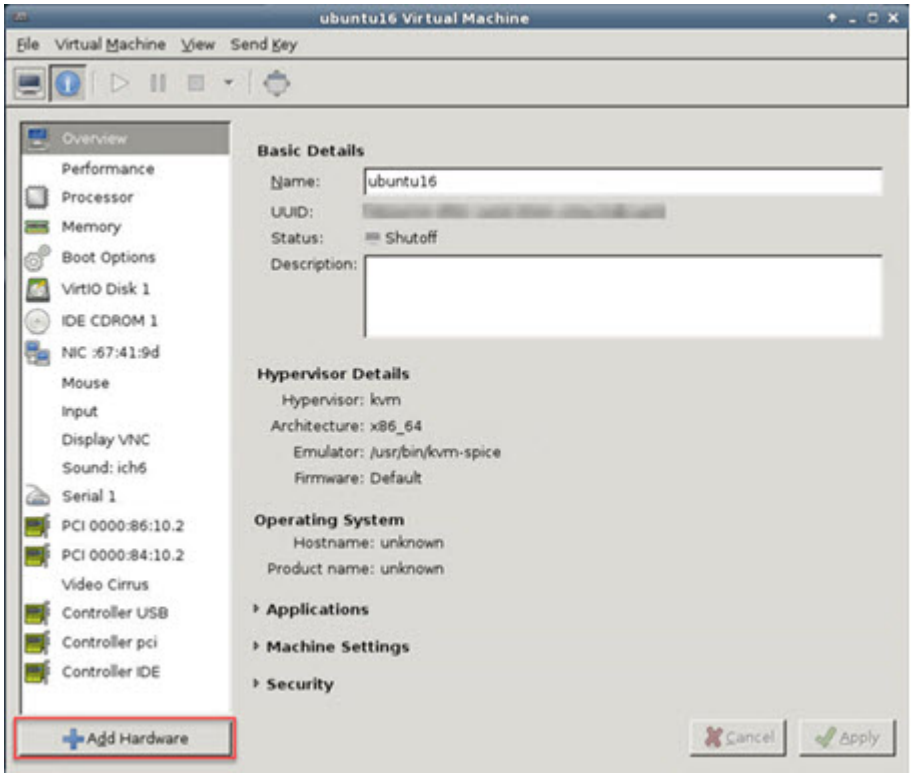
참고 **ifconfig** 명령을 사용하면 추가 인터페이스가 표시됩니다.

ASA 가상에 PCI 디바이스 할당

VF를 생성한 후에는 PCI 디바이스를 추가할 때처럼 ASA 가상에 VF를 추가할 수 있습니다. 다음 예에서는 그래픽 **virt-manager** 툴을 사용하여 이더넷 VF 컨트롤러를 ASA 가상에 추가하는 방법을 설명합니다.

단계 **1** ASA 가상을 열고 **Add Hardware(**하드웨어 추가**)** 버튼을 클릭하여 가상 머신에 새 디바이스를 추가합니다.

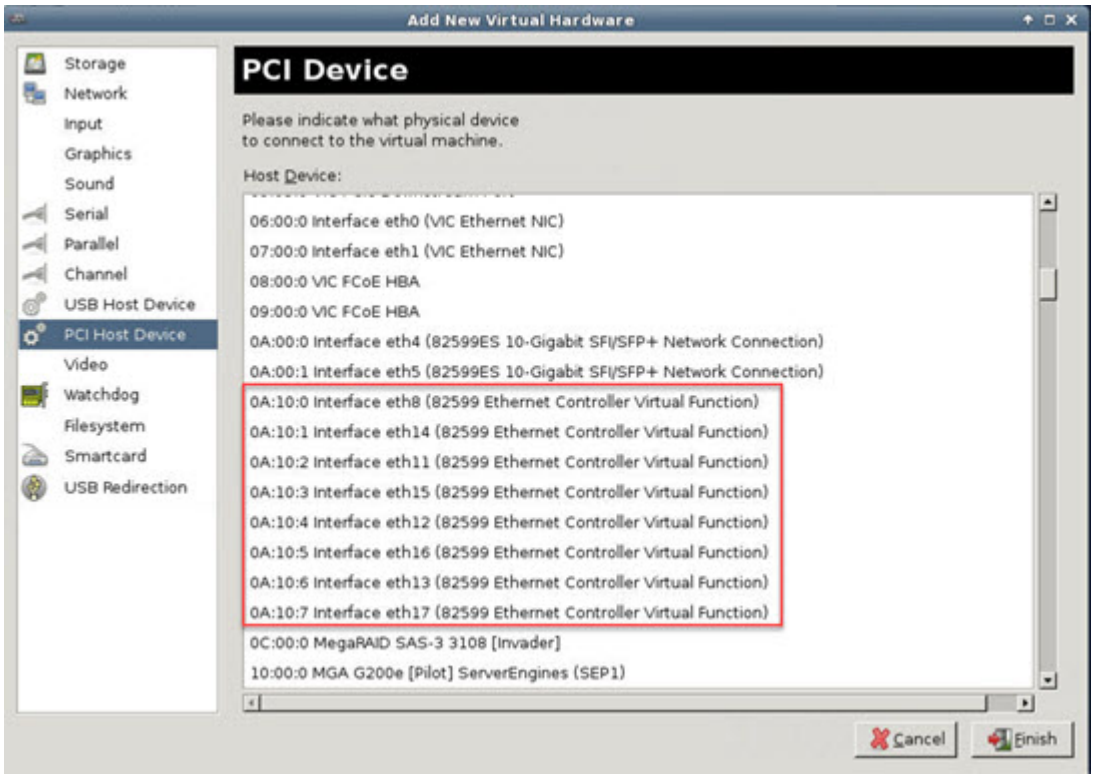
그림 ***6:*** 하드웨어 추가



단계 **2** 왼쪽 창의 **Hardware(**하드웨어**)** 목록에서 **PCI Host Device(PCI** 호스트 디바이스**)**를 클릭합니다.

VF를 포함한 PCI 디바이스 목록이 중앙 창에 표시됩니다.

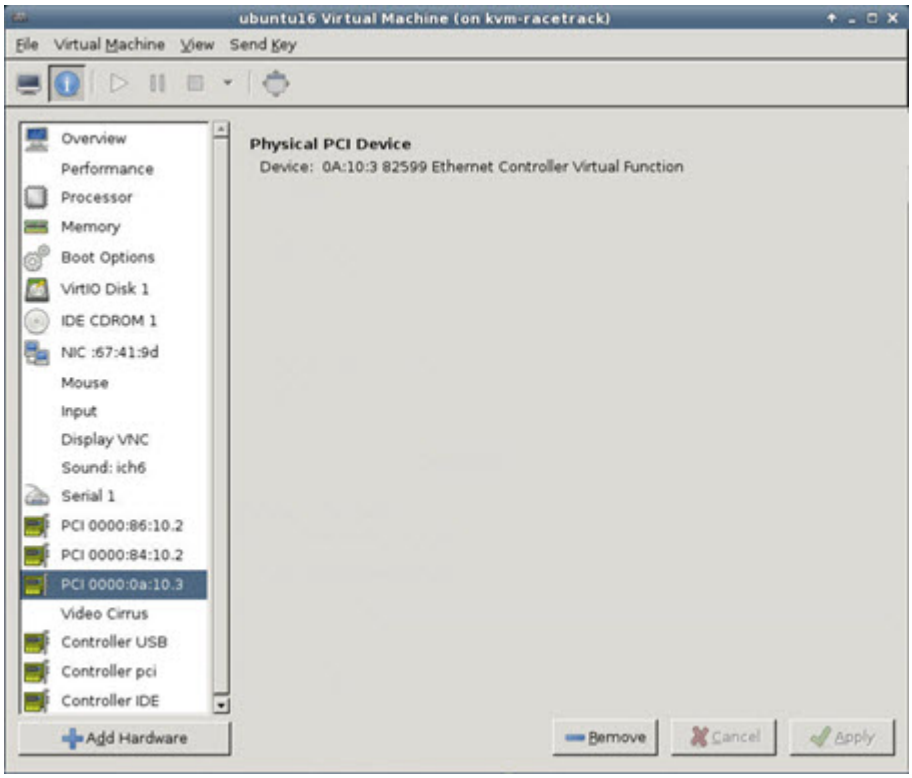
그림 7: 가상 기능 목록



단계 **3** 사용 가능한 가상 기능 중 하나를 선택하고 **Finish(**마침**)**를 클릭합니다.

PCI Device(PCI 디바이스)가 Hardware List(하드웨어 목록)에 표시됩니다. 이더넷 컨트롤러 가상 기능 역할을 하는 디바이스의 설명을 확인하십시오.

그림 *8:* 가상 기능 추가됨



다음에 수행할 작업

- ASA 가상명령줄에서 **show interface**명령을 사용하여, 새로 구성된 인터페이스를 확인합니다.
- 트래픽 전송 및 수신을 위해 인터페이스를 구성하고 활성화하려면 ASA 가상에서 인터페이스 구성 모드를 사용해야 합니다. 자세한 내용은 Cisco Secure Firewall ASA Series 일반적인 작업 CLI 구성 가이드의 기본 인터페이스 구성 장을 참조하십시오.

CPU 사용량 및 보고

CPU Utilization(CPU 사용률) 보고서에는 지정된 시간 내에 사용된 CPU의 백분율이 요약되어 있습니다. 일반적으로 코어는 사용량이 적은 시간에는 총 CPU 용량의 약 30~40%, 사용량이 많은 시간에는 약 60~70%로 작동합니다.

중요 9.13(1)부터 모든 ASA Virtual 라이선스는 지원되는 모든 ASA Virtual vCPU/메모리 구성에서 사용할 수 있습니다. 따라서 ASA Virtual 고객은 다양한 VM 리소스 사용 공간에서 실행할 수 있습니다.

ASA Virtual의 vCPU 사용량

ASA virtual vCPU 사용량에서는 데이터 경로, 제어 지점, 외부 프로세스에 사용된 vCPU 양을 확인할 수 있습니다.

vSphere에서 보고하는 vCPU 사용량에는 앞서 설명한 ASA virtual 사용량과 함께 다음 항목도 포함되어 있습니다.

- ASA virtual 유휴 시간
- ASA 가상 머신에 사용된 %SYS 오버헤드
- vSwitch, vNIC, pNIC 간 패킷 이동의 오버헤드. 이 오버헤드가 상당히 클 수 있습니다.

CPU 사용량의 예

show cpu usage 명령을 사용하여 CPU 사용률 통계를 표시할 수 있습니다.

예

```
Ciscoasa#show cpu usage

CPU utilization for 5 seconds = 1%; 1 minute: 2%; 5 minutes: 1%
```

다음은 보고된 vCPU 사용량이 상당한 차이를 보이는 예입니다.

- ASA Virtual 보고서: 40%
- DP: 35%
- 외부 프로세스: 5%
- ASA(ASA Virtual 보고서): 40%
- ASA 유휴 폴링: 10%
- 오버헤드: 45%

이 오버헤드는 하이퍼바이저 기능을 수행하고 vSwitch를 사용하여 NIC와 vNIC 간에 패킷을 이동하는 데 사용됩니다.

KVM CPU 사용량 보고

제공

`virsh cpu-stats` *`domain`* `--total start count`

명령은 지정된 게스트 가상 머신에 대한 CPU 통계 정보를 제공합니다. 기본적으로 모든 CPU에 대한 통계와 총계를 표시합니다. `--total` 옵션은 전체 통계만 표시합니다. `--count` 옵션은 카운트 CPU에 대한 통계만 표시합니다.

OProfile과 top 등의 툴은 하이퍼바이저와 VM 모두의 CPU 사용량을 포함하는 특정 KVM VM의 총 CPU 사용량을 제공합니다. 마찬가지로 Xen VMM 전용인 XenMon 같은 도구는 Xen 하이퍼바이저(예: Dom 0)의 총 CPU 사용량을 제공하지만 VM당 하이퍼바이저 사용량으로 분리하지는 않습니다.

또한 OpenNebula와 같은 클라우드 컴퓨팅 프레임워크에는 VM에서 사용하는 가상 CPU의 백분율 정보만 제공하는 특정 툴이 있습니다.

ASA Virtual 및 KVM 그래프

ASA Virtual과 KVM의 CPU % 수치가 다릅니다.

- KVM 그래프 수치가 항상 ASA Virtual 수치보다 높습니다.
- KVM에서는 이를 %CPU usage, ASA Virtual에서는 %CPU utilization이라고 부릅니다.

용어 "%CPU utilization"과 "%CPU usage"의 의미는 서로 다릅니다.

- CPU utilization은 물리적 CPU의 통계를 제공합니다.
- CPU usage는 논리적 CPU의 통계로서 CPU 하이퍼스레딩을 기반으로 합니다. 그러나 단 하나의 vCPU가 사용되므로 하이퍼스레딩은 켜져 있지 않습니다.

KVM는 %CPU usage를 다음과 같이 계산합니다.

활발하게 사용 중인 가상 CPU의 양 - 총 가용 CPU 기준 백분율로 표시

이 계산은 게스트 운영 체제가 아닌 호스트의 관점에서 본 CPU 사용량입니다. 그리고 가상 머신에 있는 사용 가능한 모든 가상 CPU의 평균 CPU 사용률입니다.

예를 들어, 가상 CPU 1개를 사용하는 가상 시스템이 4개의 물리적 CPU를 가진 호스트에서 실행되는 중이고 CPU usage가 100%라면 가상 머신에서 하나의 물리적 CPU를 온전히 사용하는 것입니다. 가상 CPU 사용량 계산: 사용량(MHz) / 가상 CPU 수 x 코어 주파수