



Deploy the ASAv Using KVM

You can deploy the ASAv on any *server class* x86 CPU device that is capable of running the Kernel-based Virtual Machine (KVM).

- [ASAv on KVM Guidelines and Limitations, on page 1](#)
- [About ASAv Deployment Using KVM, on page 2](#)
- [Prerequisites for the ASAv and KVM, on page 2](#)
- [Prepare the Day 0 Configuration File, on page 3](#)
- [Prepare the Virtual Bridge XML Files, on page 5](#)
- [Launch the ASAv, on page 7](#)
- [Performance Tuning for the ASAv on KVM, on page 8](#)
- [CPU Usage and Reporting, on page 13](#)

ASAv on KVM Guidelines and Limitations

The specific hardware used for ASAv deployments can vary, depending on the number of instances deployed and usage requirements. Each virtual appliance you create requires a minimum resource allocation—memory, number of CPUs, and disk space—on the host machine.

Review the following guidelines and limitations before you deploy the ASAv.

ASAv on KVM System Requirements

Make sure to conform to the specifications below to ensure optimal performance. The ASAv has the following requirements:

- The host CPU must be a *server class* x86-based Intel or AMD CPU with virtualization extension.

For example, ASAv performance test labs use as minimum the following: Cisco Unified Computing System™ (Cisco UCS®) C series M4 server with the Intel® Xeon® CPU E5-2690v4 processors running at 2.6GHz.

CPU Pinning

CPU pinning is required for the ASAv to function in a KVM environment; see [Enable CPU Pinning, on page 8](#).

Failover for High Availability Guidelines

For failover deployments, make sure that the standby unit has the same model license; for example, both units should be ASAv30s.

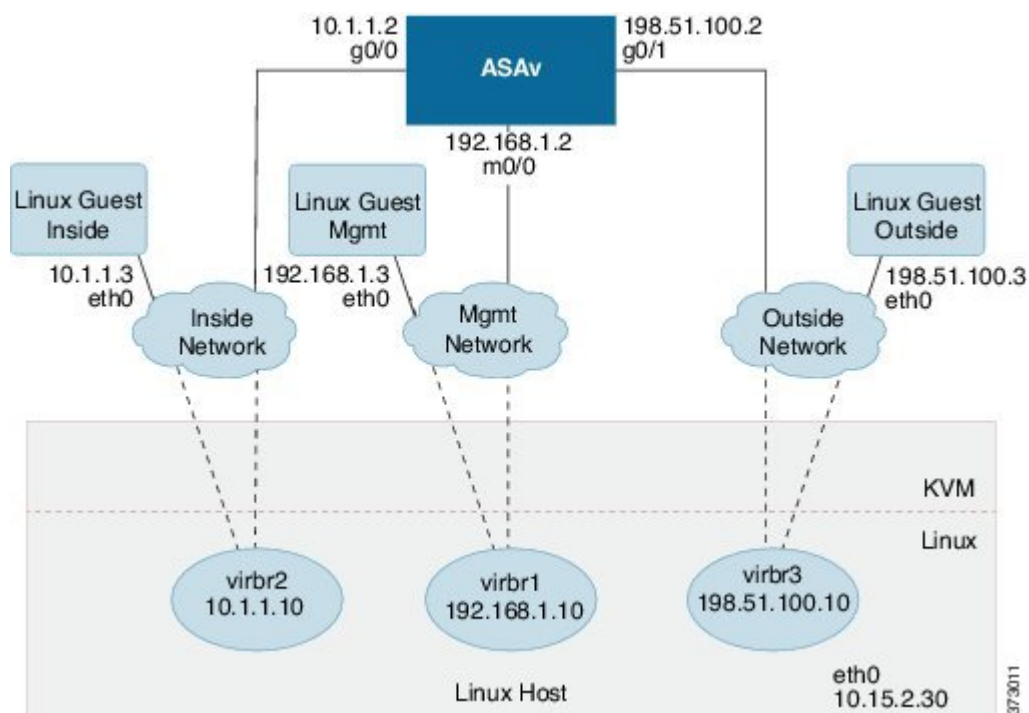


Important When creating a high availability pair using ASAv, it is necessary to add the data interfaces to each ASAv in the same order. If the exact same interfaces are added to each ASAv, but in different order, errors may be presented at the ASAv console. Failover functionality may also be affected.

About ASAv Deployment Using KVM

The following figure shows a sample network topology with ASAv and KVM. The procedures described in this chapter are based on the sample topology. The ASAv acts as the firewall between the inside and outside networks. A separate management network is also configured.

Figure 1: Sample ASAv Deployment Using KVM



Prerequisites for the ASAv and KVM

- Download the ASAv qcow2 file from Cisco.com and put it on your Linux host:

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



Note A Cisco.com login and Cisco service contract are required.

- For the purpose of the sample deployment in this document, we are assuming you are using Ubuntu 14.04 LTS. Install the following packages on top of the Ubuntu 14.04 LTS host:
 - qemu-kvm
 - libvirt-bin
 - bridge-utils
 - virt-manager
 - virtinst
 - virsh tools
 - genisoimage
- Performance is affected by the host and its configuration. You can maximize the throughput of the ASA on KVM by tuning your host. For generic host-tuning concepts, see [NFV Delivers Packet Processing Performance with Intel](#).
- Useful optimizations for Ubuntu 14.04 include the following:
 - macvtap—High performance Linux bridge; you can use macvtap instead of a Linux bridge. Note that you must configure specific settings to use macvtap instead of the Linux bridge.
 - Transparent Huge Pages—Increases memory page size and is on by default in Ubuntu 14.04.
 - Hyperthread disabled—Reduces two vCPUs to one single core.
 - txqueuelength—Increases the default txqueuelength to 4000 packets and reduces drop rate.
 - pinning—Pins qemu and vhost processes to specific CPU cores; under certain conditions, pinning is a significant boost to performance.
- For information on optimizing a RHEL-based distribution, see [Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide](#).
- For ASA software and ASA hypervisor compatibility, see [Cisco ASA Compatibility](#).

Prepare the Day 0 Configuration File

You can prepare a Day 0 configuration file before you launch the ASA. This file is a text file that contains the ASA configuration applied when the ASA is launched. This initial configuration is placed into a text file named “day0-config” in a working directory you chose, and is manipulated into a day0.iso file that is mounted and read on first boot. At the minimum, the Day 0 configuration file must contain commands to activate the management interface and set up the SSH server for public key authentication, but it can also contain a complete ASA configuration.

The day0.iso file (either your custom day0.iso or the default day0.iso) must be available during first boot:

- To automatically license the ASAv during initial deployment, place the Smart Licensing Identity (ID) Token that you downloaded from the Cisco Smart Software Manager in a text file named 'idtoken' in the same directory as the Day 0 configuration file.
- If you want to deploy the ASAv in transparent mode, you must use a known running ASA config file in transparent mode as the Day 0 configuration file. This does not apply to a Day 0 configuration file for a routed firewall.



Note We are using Linux in this example, but there are similar utilities for Windows.

Procedure

Step 1 Enter the CLI configuration for the ASAv in a text file called "day0-config." Add interface configurations for the three interfaces and any other configuration you want.

The first line should begin with the ASA version. The day0-config should be a valid ASA configuration. The best way to generate the day0-config is to copy the relevant parts of a running config from an existing ASA or ASAv. The order of the lines in the day0-config is important and should match the order seen in an existing **show running-config** command output.

Example:

```
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
```

Step 2 (Optional) For automated licensing during initial ASAv deployment, make sure the following information is in the day0-config file:

- Management interface IP address
- (Optional) HTTP proxy to use for Smart Licensing
- A **route** command that enables connectivity to the HTTP proxy (if specified) or to tools.cisco.com

- A DNS server that resolves tools.cisco.com to an IP address
- Smart Licensing configuration specifying the ASAv license you are requesting
- (Optional) A unique host name to make the ASAv easier to find in CSSM

Step 3 (Optional) Download the Smart License identity token file issued by the Cisco Smart Software Manager to your computer, copy the ID token from the download file, and put it a text file named 'idtoken' that only contains the ID token.

Step 4 Generate the virtual CD-ROM by converting the text file to an ISO file:

Example:

```
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$
```

The Identity Token automatically registers the ASAv with the Smart Licensing server.

Step 5 Repeat Steps 1 through 5 to create separate default configuration files with the appropriate IP addresses for each ASAv you want to deploy.

Prepare the Virtual Bridge XML Files

You need to set up virtual networks that connect the ASAv guests to the KVM host and that connect the guests to each other.



Note This procedure does not establish connectivity to the external world outside the KVM host.

Prepare the virtual bridge XML files on the KVM host. For the sample virtual network topology described in [Prepare the Day 0 Configuration File, on page 3](#), you need the following three virtual bridge files: virbr1.xml, virbr2.xml, and virbr3.xml (you must use these three filenames; for example, virbr0 is not allowed because it already exists). Each file has the information needed to set up the virtual bridges. You must give the virtual bridge a name and a unique MAC address. Providing an IP address is optional.

Procedure

Step 1 Create three virtual network bridge XML files. For example, virbr1.xml, virbr2.xml, and virbr3.xml:

Example:

```
<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>
```

Example:

```
<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>
```

Example:

```
<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>
```

Step 2 Create a script that contains the following (in our example, we name the script `virt_network_setup.sh`):

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

Step 3 Run this script to set up the virtual network. The script brings up the virtual networks. The networks stay up as long as the KVM host is running.

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

Note If you reload the Linux host, you must rerun the `virt_network_setup.sh` script. It does not persist over reboots.

Step 4 Verify that the virtual networks were created:

```
stack@user-ubuntu:~/KvmAsa$ brctl show
bridge name bridge id STP enabled Interfaces
virbr0 8000.00000000000000 yes
virbr1 8000.5254000056eed yes virb1-nic
virbr2 8000.5254000056eee yes virb2-nic
virbr3 8000.5254000056eec yes virb3-nic
stack@user-ubuntu:~/KvmAsa$
```

Step 5 Display the IP address assigned to the `virbr1` bridge. This is the IP address that you assigned in the XML file.

```
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
```

Launch the ASAv

Use a virt-install based deployment script to launch the ASAv.

Procedure

Step 1 Create a virt-install script called “virt_install_asav.sh.”

The name of the ASAv machine must be unique across all other VMs on this KVM host.

The ASAv supports up to 10 networks. This example uses three networks. The order of the network bridge clauses is important. The first one listed is always the management interface of the ASAv (Management 0/0), the second one listed is GigabitEthernet 0/0 of the ASAv, and the third one listed is GigabitEthernet 0/1 of the ASAv, and so on up through GigabitEthernet 0/8. The virtual NIC must be Virtio.

Example:

```
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
```

Step 2 Run the virt_install script:

Example:

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

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

A window appears displaying the console of the VM. You can see that the VM is booting. It takes a few minutes for the VM to boot. Once the VM stops booting you can issue CLI commands from the console screen.

Performance Tuning for the ASAv on KVM

Increasing Performance on KVM Configurations

You can increase the performance for an ASAv in the KVM environment by changing settings on the KVM host. These settings are independent of the configuration settings on the host server. This option is available in Red Hat Enterprise Linux 7.0 KVM.

You can improve performance on KVM configurations by enabling CPU pinning.

Enable CPU Pinning

ASAv requires that you use the KVM CPU affinity option to increase the performance of the ASAv in KVM environments. Processor affinity, or CPU pinning, enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU.

Configure host aggregates to deploy instances that use CPU pinning on different hosts from instances that do not, to avoid unpinned instances using the resourcing requirements of pinned instances.



Attention Do not deploy instances with NUMA topology on the same hosts as instances that do not have NUMA topology.

To use this option, configure CPU pinning on the KVM host.

Procedure

Step 1 In the KVM host environment, verify the host topology to find out how many vCPUs are available for pinning:

Example:

```
virsh nodeinfo
```

Step 2 Verify the available vCPU numbers:

Example:

```
virsh capabilities
```

Step 3 Pin the vCPUs to sets of processor cores:

Example:

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

The **virsh vcpupin** command must be executed for each vCPU on your ASAv. The following example shows the KVM commands needed if you have an ASAv configuration with four vCPUs and the host has eight cores:

```
virsh vcpupin asav 0 2
virsh vcpupin asav 1 3
virsh vcpupin asav 2 4
virsh vcpupin asav 3 5
```

The host core number can be any number from 0 to 7. For more information, see the KVM documentation.

Note When configuring CPU pinning, carefully consider the CPU topology of the host server. If using a server configured with multiple cores, do not configure CPU pinning across multiple sockets.

The downside of improving performance on KVM configuration is that it requires dedicated system resources.

NUMA Guidelines

Non-Uniform Memory Access (NUMA) is a shared memory architecture that describes the placement of main memory modules with respect to processors in a multiprocessor system. When a processor accesses memory that does not lie within its own node (remote memory), data must be transferred over the NUMA connection at a rate that is slower than it would be when accessing local memory.

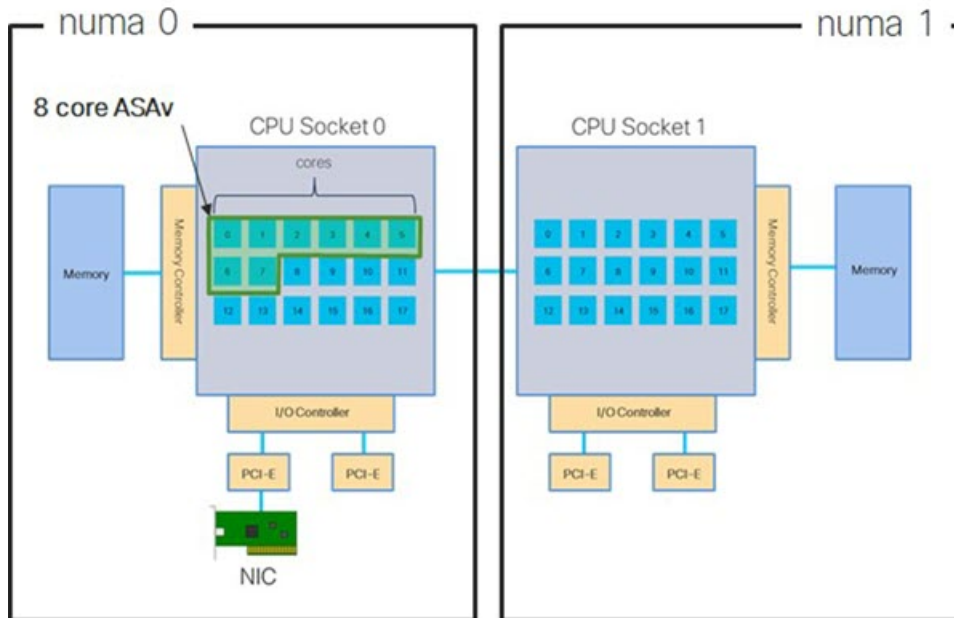
The x86 server architecture consists of multiple sockets and multiple cores within a socket. Each CPU socket along with its memory and I/O is referred to as a NUMA node. To efficiently read packets from memory, guest applications and associated peripherals (such as the NIC) should reside within the same node.

For optimum ASAv performance:

- The ASAv machine must run on a single numa node. If a single ASAv is deployed so that it runs across 2 sockets, the performance will be significantly degraded.
- An 8-core ASAv ([Figure 2: 8-Core ASAv NUMA Architecture Example, on page 10](#)) requires that each socket on the host CPU have a minimum of 8 cores per socket. Consideration must be given to other VMs running on the server.
- The NIC should be on same NUMA node as ASAv machine.

The following figure shows a server with two CPU sockets with each CPU having 18 cores. The 8-core ASAv requires that each socket on the host CPU have a minimum of 8 cores.

Figure 2: 8-Core ASAv NUMA Architecture Example



NUMA Optimization

Optimally, the ASAv machine should run on the same numa node that the NICs are running on. To do this:

1. Determine which node the NICs are on by using "lstopm" to show a diagram of the nodes. Locate the NICs and take note to which node they are attached.
2. At the KVM Host, use `virsh list` to find the ASAv.
3. Edit the VM by: `virsh edit <VM Number>`.
4. Align ASAv on the chosen node. The following examples assume 18-core nodes.

Align onto Node 0:

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

Align onto Node 1:

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

5. Save the .xml change and power cycle the ASAv machine.
6. To ensure your VM is running on the desired node, perform a `ps aux | grep <name of your ASAv VM>` to get the process ID.
7. Run `sudo numastat -c <ASAv VM Process ID>` to see if the ASAv machine is properly aligned.

More information about using NUMA tuning with KVM can be found in the RedHat document [9.3. libvirt NUMA Tuning](#).

Multiple RX Queues for Receive Side Scaling (RSS)

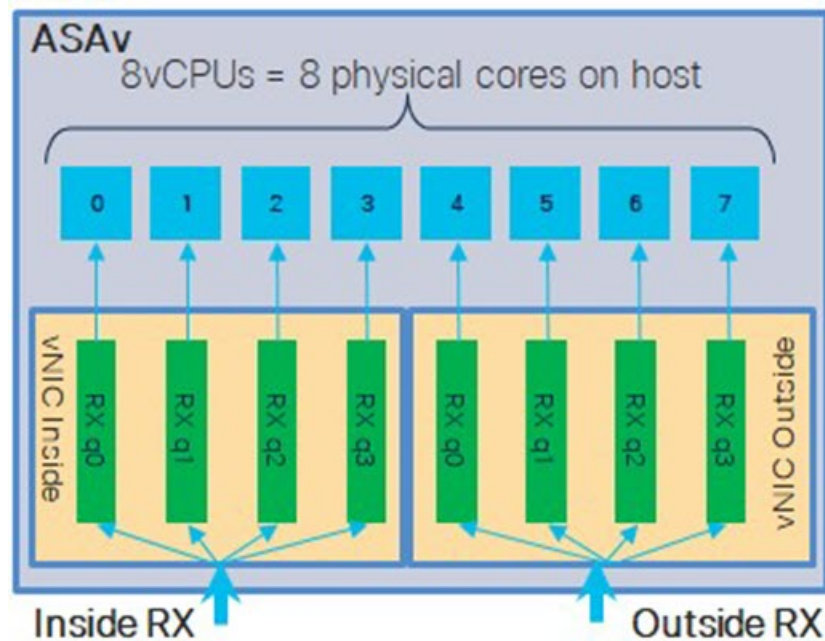
The ASAv supports Receive Side Scaling (RSS), which is a technology utilized by network adapters to distribute network receive traffic in parallel to multiple processor cores. For maximum throughput, each vCPU (core) must have its own NIC RX queue. Note that a typical RA VPN deployment might use a single inside/outside pair of interfaces.



Important You need ASAv Version 9.13(1) or greater to use multiple RX queues. For KVM, the *libvirt* version needs to be a minimum of 1.0.6.

For an 8-core VM with an inside/outside pair of interfaces, each interface will have 4 RX queues, as shown in [Figure 3: 8-Core ASAv RSS RX Queues](#), on page 11.

Figure 3: 8-Core ASAv RSS RX Queues



The following table presents the ASAv's vNICs for KVM and the number of supported RX queues. See [#unique_31 unique_31_Connect_42_section_pht_vfh_glb](#) for descriptions of the supported vNICs.

Table 1: KVM Recommended NICs/vNICs

NIC Card	vNIC Driver	Driver Technology	Number of RX Queues	Performance
x710	i40e	PCI Passthrough	8 maximum	PCI Passthrough and SR-IOV modes for the x710 offer the best performance. SR-IOV is typically preferred for virtual deployments because the NIC can be shared across multiple VMs.
	i40evf	SR-IOV	8	
x520	ixgbe	PCI Passthrough	6	The x520 NIC performs 10 to 30% lower than the x710. PCI Passthrough and SR-IOV modes for the x520 offer similar performance. SR-IOV is typically preferred for virtual deployments because the NIC can be shared across multiple VMs.
	ixgbe-vf	SR-IOV	2	
N/A	virtio	Para-virtualized	8 maximum	Not recommended for ASA v100. For other deployments, see Enable Multiqueue Support for Virtio on KVM , on page 12.

Enable Multiqueue Support for Virtio on KVM

The following example shows to configure the number of Virtio NIC RX queues to 4 using virsh to edit the libvirt xml:

```
<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>
```



Important The *libvirt* version needs to be a minimum of 1.0.6 to support multiple RX queues.

VPN Optimization

These are some additional considerations for optimizing VPN performance with the ASA v.

- IPsec has higher throughput than DTLS.
- Cipher - GCM has about 2x the throughput of CBC.

CPU Usage and Reporting

The CPU Utilization report summarizes the percentage of the CPU used within the time specified. Typically, the Core operates on approximately 30 to 40 percent of total CPU capacity during nonpeak hours and approximately 60 to 70 percent capacity during peak hours.

vCPU Usage in the ASA Virtual

The ASA virtual vCPU usage shows the amount of vCPUs used for the data path, control point, and external processes.

The vSphere reported vCPU usage includes the ASA virtual usage as described plus:

- ASA virtual idle time
- %SYS overhead used for the ASA virtual machine
- Overhead of moving packets between vSwitches, vNICs, and pNICs. This overhead can be quite significant.

CPU Usage Example

The `show cpu usage` command can be used to display CPU utilization statistics.

Example

```
Ciscoasa#show cpu usage
```

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

The following is an example in which the reported vCPU usage is substantially different:

- ASA Virtual reports: 40%
- DP: 35%
- External Processes: 5%
- ASA (as ASA Virtual reports): 40%
- ASA idle polling: 10%
- Overhead: 45%

The overhead is used to perform hypervisor functions and to move packets between NICs and vNICs using the vSwitch.

KVM CPU Usage Reporting

The

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

command provides the CPU statistical information on the specified guest virtual machine. By default, it shows the statistics for all CPUs, as well as a total. The `--total` option will only display the total statistics. The `--count` option will only display statistics for *count* CPUs.

Tools like OProfile, top etc. give the total CPU usage of a particular KVM VM which includes the CPU usage of both the hypervisor as well as VM. Similarly, tools like XenMon which are specific to Xen VMM gives total CPU usage of Xen hypervisor i.e Dom 0 but don't separate it into hypervisor usage per VM.

Apart from this, certain tools exist in cloud computing frameworks like OpenNebula which only provides coarse grained information of percentage of Virtual CPU used by a VM.

ASA Virtual and KVM Graphs

There are differences in the CPU % numbers between the ASA Virtual and KVM:

- The KVM graph numbers are always higher than the ASA Virtual numbers.
- KVM calls it %CPU usage; the ASA Virtual calls it %CPU utilization.

The terms “%CPU utilization” and “%CPU usage” mean different things:

- CPU utilization provides statistics for physical CPUs.
- CPU usage provides statistics for logical CPUs, which is based on CPU hyperthreading. But because only one vCPU is used, hyperthreading is not turned on.

KVM calculates the CPU % usage as follows:

Amount of actively used virtual CPUs, specified as a percentage of the total available CPUs

This calculation is the host view of the CPU usage, not the guest operating system view, and is the average CPU utilization over all available virtual CPUs in the virtual machine.

For example, if a virtual machine with one virtual CPU is running on a host that has four physical CPUs and the CPU usage is 100%, the virtual machine is using one physical CPU completely. The virtual CPU usage calculation is Usage in MHz / number of virtual CPUs x core frequency