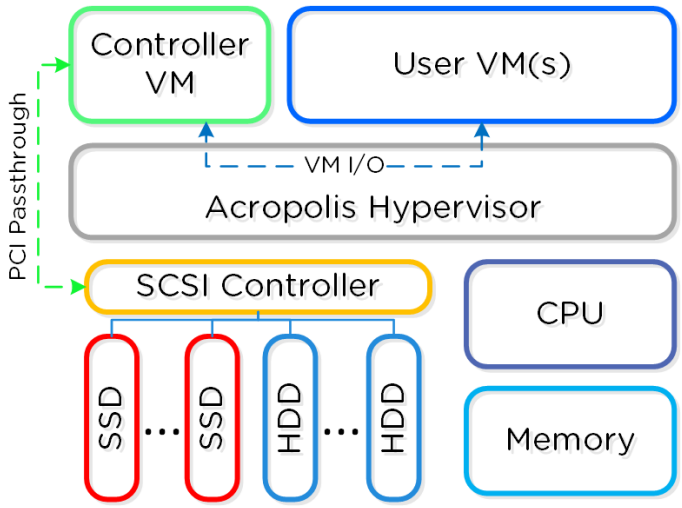
Book of AHV - AHV Architecture

[PDF generated August 17 2023. For all recent updates please see the Nutanix Bible releases notes located at https://nutanixbible.com/release_notes.html. Disclaimer: Downloaded PDFs may not always contain the latest information.]

Node Architecture

In AHV deployments, the Controller VM (CVM) runs as a VM and disks are presented using PCI passthrough. This allows the full PCI controller (and attached devices) to be passed through directly to the CVM and bypass the hypervisor. AHV is based upon CentOS KVM. Full hardware virtualization is used for guest VMs (HVM).



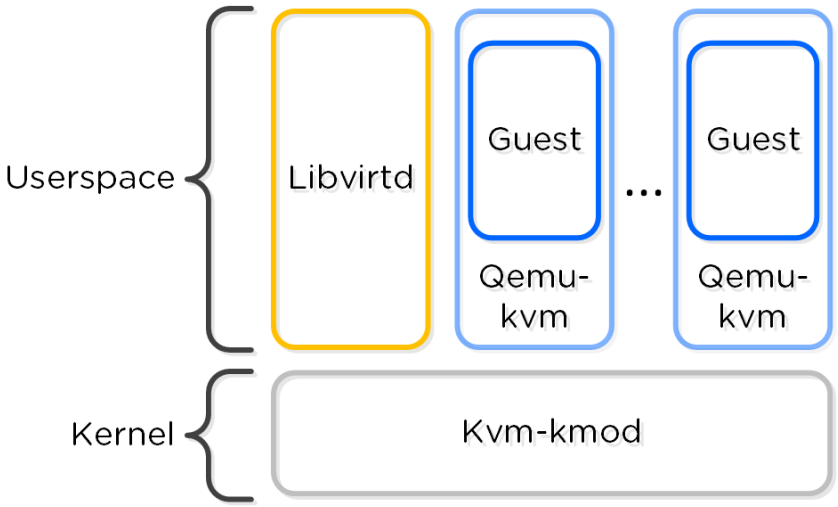
AHV Node

KVM Architecture

Within KVM there are a few main components:

- KVM-kmod
  - KVM kernel module
- Libvirtd
  - An API, daemon and management tool for managing KVM and QEMU. Communication between AOS and KVM / QEMU occurs through libvirtd.
- Qemu-kvm
  - A machine emulator and virtualizer that runs in userspace for every Virtual Machine (domain). In AHV it is used for hardware-assisted virtualization and VMs run as HVMs.

The following figure shows the relationship between the various components:



KVM Component Relationship

Communication between AOS and KVM occurs via Libvirt.

Processor generation compatibility

Similar to VMware's Enhanced vMotion Capability (EVC) which allows VMs to move between different processor generations; AHV will determine the lowest processor generation in the cluster and constrain all QEMU domains to that level. This allows mixing of processor generations within an AHV cluster and ensures the ability to live migrate between hosts.

Configuration Maximums and Scalability

The following configuration maximums and scalability limits are applicable:

- Maximum cluster size: **32**
- Maximum vCPUs per VM: **Number of physical cores per host**
- Maximum memory per VM: **4.5TB or available physical node memory**
- Maximum virtual disk size: **9EB* (Exabyte)**
- Maximum VMs per host: **N/A – Limited by memory**
- Maximum VMs per cluster: **N/A – Limited by memory**

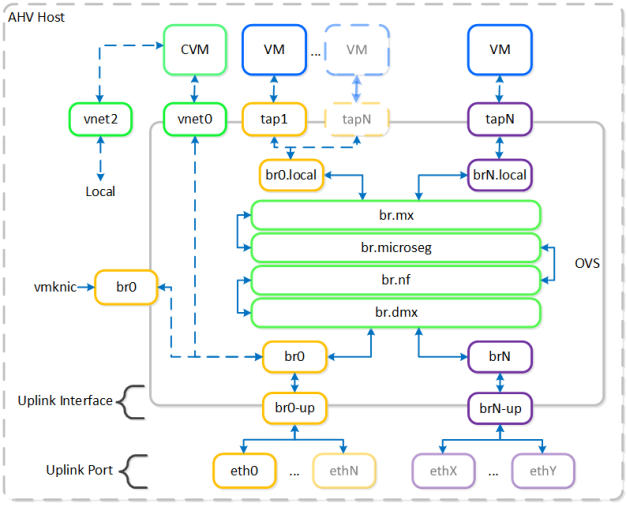
*AHV does not have a traditional storage stack like ESXi / Hyper-V; all disks are passed to the VM(s) as raw SCSI block devices. This means the maximum virtual disk size is limited by the maximum AOS vDisk size (9 Exabytes).

The above are true as of AHV 20220304.10013 and AOS 6.6. Refer to Configuration Maximums for other versions.

Networking

AHV leverages Open vSwitch (OVS) for all VM networking. VM networking is configured through Prism / ACLI and each VM nic is connected into a tap interface.

The following figure shows a conceptual diagram of the OVS architecture:



Open vSwitch Network Overview

In the prior image you see a few types of components:

Open vSwitch (OVS)

OVS is an open source software switch implemented in the Linux kernel and designed to work in a multiserver virtualization environment. By default, OVS behaves like a layer-2 learning switch that maintains a MAC address table. The hypervisor host and VMs connect to virtual ports on the switch.

OVS supports many popular switch features, including VLAN tagging, Link Aggregation Control Protocol (LACP), port mirroring, and quality of service (QoS), to name a few. Each AHV server maintains an OVS instance, and all OVS instances combine to form a single logical switch. Constructs called bridges manage the switch instances residing on the AHV hosts.

Bridge

Bridges act as virtual switches to manage network traffic between physical and virtual network interfaces. The default AHV configuration includes an OVS bridge called br0 and a native Linux bridge called virbr0. The virbr0 Linux bridge carries management and storage communication between the CVM and AHV host. All other storage, host, and VM network traffic flows through the br0 OVS bridge. The AHV host, VMs, and physical interfaces use “ports” for connectivity to the bridge.

Port

Ports are logical constructs created in a bridge that represent connectivity to the virtual switch. Nutanix uses several port types, including internal, tap, VXLAN, and bond:

- An internal port—with the same name as the default bridge (br0)—provides access for the AHV host.
- Tap ports act as bridge connections for virtual NICs presented to VMs.
- VXLAN ports are used for the IP address management functionality provided by Acropolis.
- Bonded ports provide NIC teaming for the physical interfaces of the AHV host.

Bond

Bonded ports aggregate the physical interfaces on the AHV host. By default, a bond named br0-up is created in bridge br0. After the node imaging process, all interfaces are placed within a single bond, which is a requirement for the foundation imaging process. Changes to the default bond, br0-up, often rename this to bond0. Nutanix recommends using the name br0-up to quickly identify the interface as the bridge br0 uplink.

OVS bonds allow for several load-balancing modes, including active-backup, balance-slb and balance-tcp. LACP can also be activated for a bond. The “bond_mode” setting is not specified during installation and therefore defaults to active-backup, which is the recommended configuration.

Uplink Load Balancing

Briefly mentioned in the prior section, it is possible to balance traffic across bond uplinks.

The following bond modes are available:

- active-backup
  - Default configuration which transmits all traffic over a single active adapter. If the active adapter becomes unavailable, another adapter in the bond will become active. Limits throughput to a single nic's bandwidth. (Recommended)

- balance-slb
  - Balances each VM's nic across adapters in the bond (e.g. VM A nic 1 - eth0 / nic 2 - eth1). Limits VM per-nic throughput to a single nic's bandwidth, however a VM with x nics can leverage x * adapter bandwidth (assuming x is the same for the number of VM nics and physical uplink adapters in the bond). NOTE: has caveats for multicast traffic

- balance-tcp / LACP
  - Balances each VM nic's TCP session across adapters in the bond. Limits per-nic throughput to the maximum bond bandwidth (number of physical uplink adapters * speed). Requires link aggregation and used when LACP is required.

You can find additional information on bonds in the AHV Networking guide (LINK).

VM NIC Types

AHV supports the following VM network interface types:

- Access (default)
- Trunk (4.6 and above)

By default VM nics will be created as Access interfaces (similar to what you'd see with a VM nic on a port group), however it is possible to expose a trunked interface up to the VM's OS. Trunked NICs send the primary VLAN untagged, and all additional VLANs as tags to the same vNIC on the VM. This is useful to bring multiple networks to a VM without adding vNICs.

A trunked interface can be added with the following command:

```
vm.nic_create VM_NAME vlan_mode=kTrunked trunked_networks=ALLOWED_VLANS network=NATIVE_VLAN
```

Example:

```
vm.nic_create fooVM vlan_mode=kTrunked trunked_networks=10,20,30 network=vlan.10
```

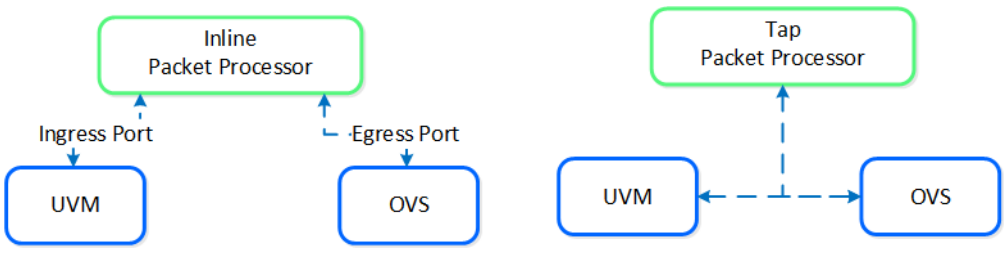
Service Chaining

AHV Service chaining allows us to intercept all traffic and forward to a packet processor (NFV, appliance, virtual appliance, etc.) functions trasparently as part of the network path.

Common uses for service chaining:

- Firewall (e.g. Palo Alto, ete.)
- Load balancer (e.g. F5, Netscaler, etc.)
- IDS/IPS/network monitors (e.g. packet capture)

Within service chaining there are two types of way:

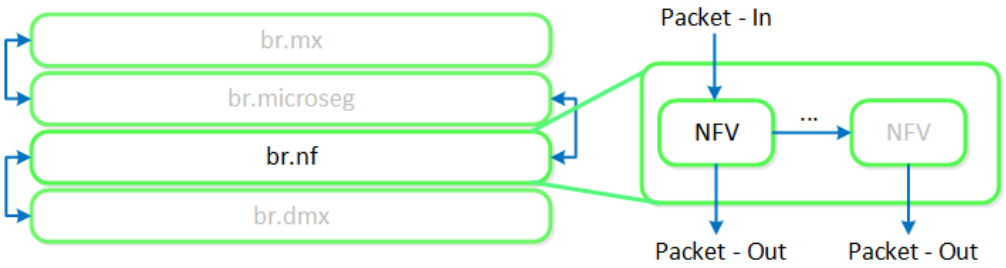


Service chain - Packet Processors

- Inline packet processor
  - Intercepts packets inline as they flow through OVS
  - Can modify and allow/deny packet
  - Common uses: firewalls and load balancers

- Tap packet processor
  - Inspects packets as they flow, can only read as it's a tap into the packet flow
  - Common uses: IDS/IPS/network monitor

Any service chaining is done after the Flow - Microsegmentation rules are applied and before the packet leaves the local OVS. This occurs in the network function bridge (br.nf):



Service Chain - Flow

NOTE: it is possible to string together multiple NFV / packet processors in a single chain.

VM Templates

AHV has always had the image library which focused on capturing the data within a single vdisk so that it could be easily cloned, but input from the admin was needed to complete the process of declaring the CPU, memory and network details. VM Templates take this concept to the next level of simplicity and provides a familiar construct for admins that have utilized templates on other hypervisors.

AHV VM Templates are created from existing virtual machines, inheriting the attributes of the defining VM such as the CPU, memory, vdisks, and networking details. The template can then be configured to customize the guest OS upon deployment and can optionally provide a Windows license key. Templates allow for multiple versions to be maintained, allowing for easy updates such as operating system and application patches to be applied without the need to create a new template. Admins can choose which version of the template is active, allowing the updates to be staged ahead of time or the ability to switch back to a previous version if needed.

Memory Overcommit

One of the central benefits of virtualization is the ability to overcommit compute resources, making it possible to provision more CPUs to VMs than are physically present on the server host. Most workloads don’t need all of their assigned CPUs 100% of the time, and the hypervisor can dynamically allocate CPU cycles to workloads that need them at each point in time.

Much like CPU or network resources, memory can be overcommitted also. At any given time, the VMs on the host may or may not use all their allocated memory, and the hypervisor can share that unused memory with other workloads. Memory overcommit makes it possible for administrators to provision a greater number of VMs per host, by combining the unused memory and allocating it to VMs that need it.

AOS 6.1 brings memory overcommit to AHV as an option to allow administrators to utilize in environments such as test and development where additional memory and VM density is required. Overcommit is disabled by default and can be defined on a per-VM basis allowing sharing to be done on all or just a subset of the VMs on a cluster.

VM Affinity Policies

Different types of applications can have requirements that dictate whether the VMs should run on the same host or different host. This is typically done for performance or availability benefits. Affinity controls enable you to govern where VMs run. AHV has two types of affinity controls:

- VM-host affinity
  - Strictly ties a VM to a host or group of hosts, so the VM only runs on that host or group. Affinity is particularly applicable for use cases that involve software licensing or VM appliances. In such cases, you often need to limit the number of hosts an application can run on or bind a VM appliance to a single host.
- Anti-affinity
  - AHV lets you declare that a given list of VMs shouldn’t run on the same hosts. Anti-affinity gives you a mechanism for allowing clustered VMs or VMs running a distributed application to run on different hosts, increasing the application’s availability and resiliency. To prefer VM availability over VM separation, the system overrides this type of rule when a cluster becomes constrained.