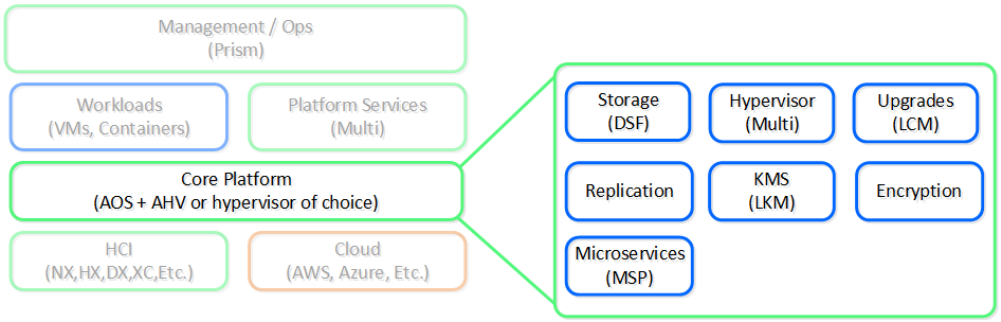
Book of AOS - AOS Architecture

[PDF generated December 08 2021. 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.]

The Acropolis Operating System (AOS) provides the core functionality leveraged by workloads and services running on the platform. This includes, but isn't limited to, things like storage services, upgrades, etc.

The figure highlights an image illustrating the conceptual nature of AOS at various layers:



High-level AOS Architecture

Building upon the distributed nature of everything Nutanix does, we're expanding this into the virtualization and resource management space. AOS is a back-end service that allows for workload and resource management, provisioning, and operations. Its goal is to abstract the facilitating resource (e.g., hypervisor, on-premise, cloud, etc.) from the workloads running, while providing a single "platform" to operate.

This gives workloads the ability to seamlessly move between hypervisors, cloud providers, and platforms.

Supported Hypervisors for VM Management

As of 4.7, AHV and ESXi are the supported hypervisors for VM management, however this may expand in the future. The Volumes API and read-only operations are still supported on all.

Acropolis Services

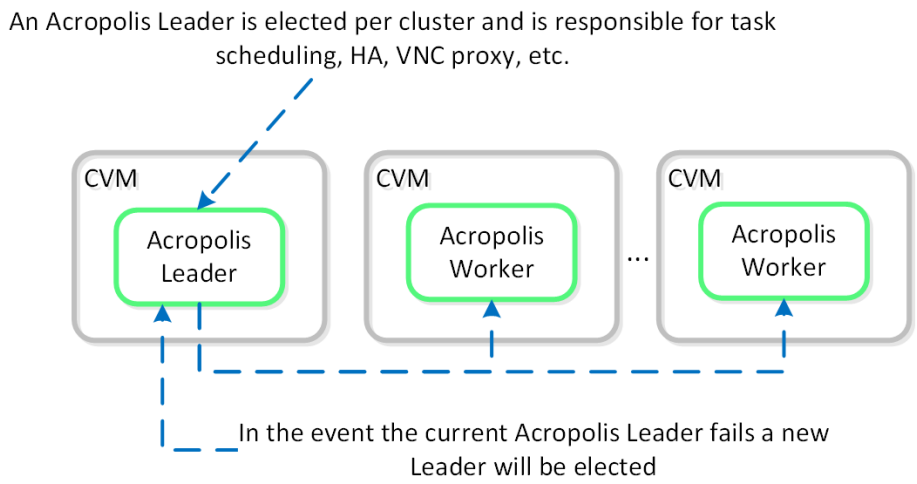
An Acropolis Worker runs on every CVM with an elected Acropolis Leader which is responsible for task scheduling, execution, IPAM, etc. Similar to other components which have a Leader, if the Acropolis Leader fails, a new one will be elected.

The role breakdown for each can be seen below:

- Acropolis Leader
  - Task scheduling & execution
  - Stat collection / publishing
  - Network Controller (for hypervisor)
  - VNC proxy (for hypervisor)
  - HA (for hypervisor)

- Acropolis Worker
  - Stat collection / publishing
  - VNC proxy (for hypervisor)

Here we show a conceptual view of the Acropolis Leader / Worker relationship:



Acropolis Services

Dynamic Scheduler

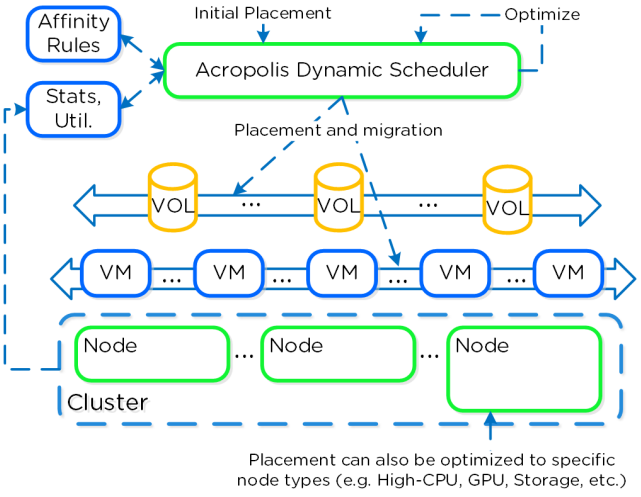
Efficient scheduling of resources is critical to ensure resources are effectively consumed. The AOS Dynamic Scheduler extends the traditional means of scheduling that relies upon compute utilization (CPU/MEM) to make placement decisions. It leverages compute, as well as storage and others to drive VM and volume (ABS) placement decisions. This ensures that resources are effectively consumed and end-user performance is optimal.

Resource scheduling can be broken down into two key areas:

- Initial placement
  - Where an item is scheduled at power-on
- Runtime Optimization
  - Movement of workloads based upon runtime metrics

The original AOS Scheduler had taken care of the initial placement decisions since its release. With its release in AOS 5.0, the AOS Dynamic Scheduler expands upon this to provide runtime resources optimization.

The figure shows a high-level view of the scheduler architecture:



AOS Dynamic Scheduler

The dynamic scheduler runs consistently throughout the day to optimize placement (currently every 15 minutes | Gflag: lazananomalydetectionperiodsecs). Estimated demand is calculated using historical utilization values and fed into a smoothing algorithm. This estimated demand is what is used to determine movement, which ensures a sudden spike will not skew decisions.

A different approach towards resource optimization

When you look at existing scheduling / optimization platforms (VMware DRS, Microsoft PRO) they are all focused on balancing workloads / VMs evenly across cluster resources. NOTE: how aggressively it tries to eliminate skew is determined by the balancing configuration (e.g. manual -> none, conservative -> some, aggressive -> more).

For example, say we had 3 hosts in a cluster, each of which is utilized 50%, 5%, 5% respectively. Typical solutions would try to re-balance workloads to get each hosts utilization ~20%. But why?

What we're really trying to do is eliminate / negate any contention for resources, not eliminate skew. Unless there is contention for resources there is no positive gain from "balancing" workloads. In fact by forcing unnecessary movement we cause additional requisite work (e.g. memory transfer, cache re-localization, etc.), all of which consumes resources.

The AOS Dynamic Scheduler does just this, it will only invoke workload movement if there is expected contention for resources, not because of skew. NOTE: DSF works in a different way and works to ensure uniform distribution of data throughout the cluster to eliminate hot spots and speed up rebuilds. To learn more of DSF, check out the 'disk balancing' section.

At power-on ADS will balance VM initial placement throughout the cluster.

At power-on ADS will balance VM initial placement throughout the cluster.

Placement Decisions

Placement decisions are based upon the following items:

- Compute utilization
  - We monitor each individual node's compute utilization. In the event where a node's expected CPU allocation breaches its threshold (currently 85% of host CPU | Gflag: lazanhostcpuusagethreshold_fraction) we will migrate VMs off those host(s) to re-balance the workload. A key thing to mention here is a migration will only be performed when there is contention. If there is skew in utilization between nodes (e.g. 3 nodes at 10% and 1 at 50%) we will not perform a migration as there is no benefit from doing so until there is contention for resources.

- Storage performance
  - Being a hyperconverged platform we manage both compute and storage resources. The scheduler will monitor each node's Stargate process utilization. In the event where certain Stargate(s) breach their allocation threshold (currently 85% of CPU allocated to Stargate | Gflag: lazanstargatecpuusagethreshold_pct) , we will migrate resources across hosts to eliminate any hot spots. Both VMs and ABS Volumes can be migrated to eliminate any hot Stargates.

- [Anti-]Affinity rules
  - Affinity or Anti-affinity constraints determine where certain resources are scheduled based upon other resources in the environment. In certain cases you want VMs to run on the same node for licensing reasons. In this case the VMs would be affined to the same host. In other cases you might want to ensure VMs run on different nodes for availability purposes. In this case the VMs would be anti-affined.

The scheduler will make its best effort to optimize workload placement based upon the prior items. The system places a penalty on movement to ensure not too many migrations are taking place. This is a key item as we want to make sure the movement doesn't have any negative impacts on the workload.

After a migration the system will judge its "effectiveness" and see what the actual benefit is. This learning model can self-optimize to ensure there is a valid basis for any migration decision.