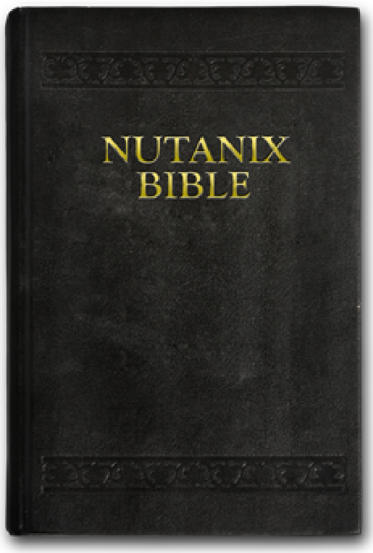


The Nutanix Bible

Intro

Welcome to The Nutanix Bible! I work the with Nutanix platform on a daily basis – trying to find issues, push its limits as well as administer it for my production benchmarking lab. This page is being produced to serve as a living document outlining tips and tricks used every day by myself and a variety of engineers at Nutanix. This will also include summary items discussed as part of the Advanced Nutanix series. NOTE: This is not an official reference so tread at your own risk!

Book of Nutanix

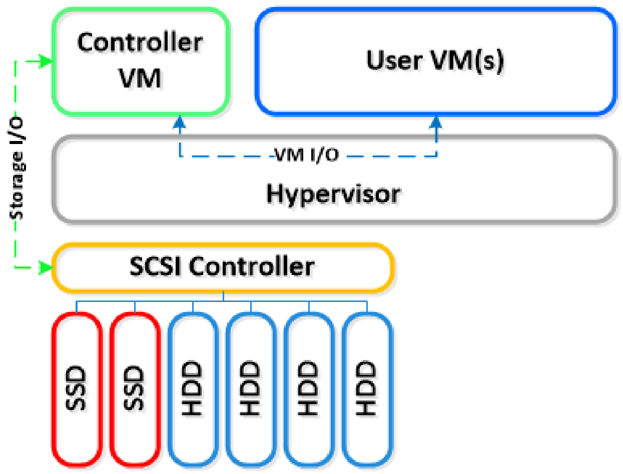
Architecture

Converged Platform

The Nutanix solution is a converged storage + compute solution which leverages local components and creates a distributed platform for virtualization aka virtual computing platform. The solution is a bundled hardware + software appliance which houses 2 (6000/7000 series) or 4 nodes (1000/2000/3000/3050 series) in a 2U footprint.

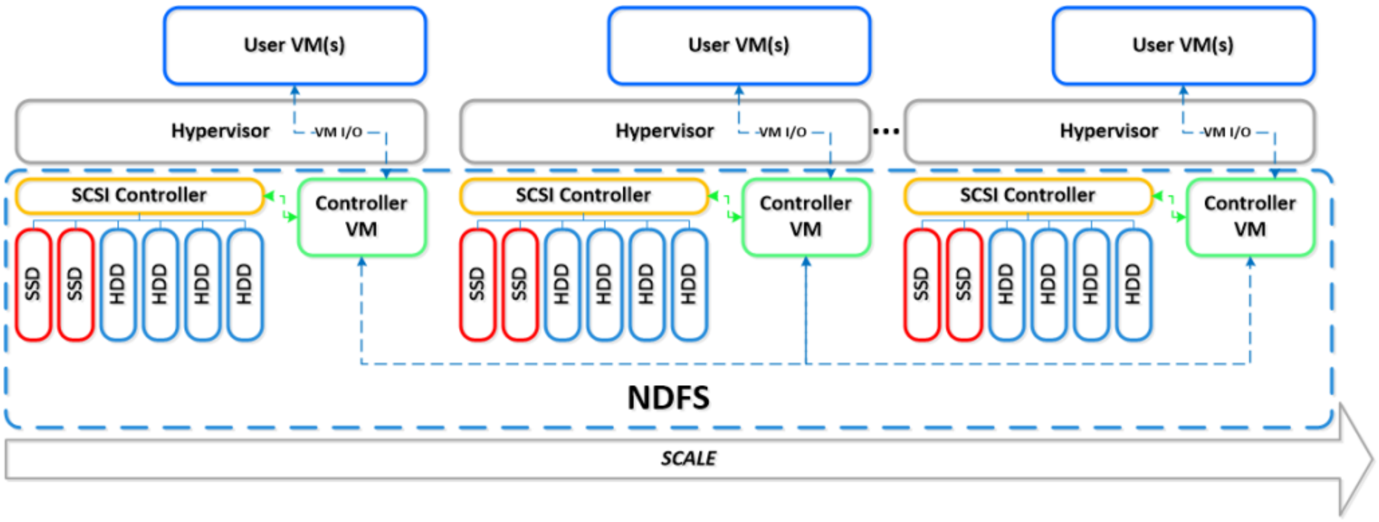
Each node runs an industry standard hypervisor (ESXi, KVM, Hyper-V currently) and the Nutanix Controller VM (CVM). The Nutanix CVM is what runs the Nutanix software and serves all of the I/O operations for the hypervisor and all VMs running on that host. For the Nutanix units running VMware vSphere, the SCSI controller, which manages the SSD and HDD devices, is directly passed to the CVM leveraging VM-Direct Path (Intel VT-d). In the case of Hyper-V the storage devices are passed through to the CVM.

Below is an example of what a typical node logically looks like:



Together, a group of Nutanix Nodes forms a distributed platform called the Nutanix Distributed Filesystem (NDFS). NDFS appears to the hypervisor like any centralized storage array, however all of the I/Os are handled locally to provide the highest performance. More detail on how these nodes form a distributed system can be found below.

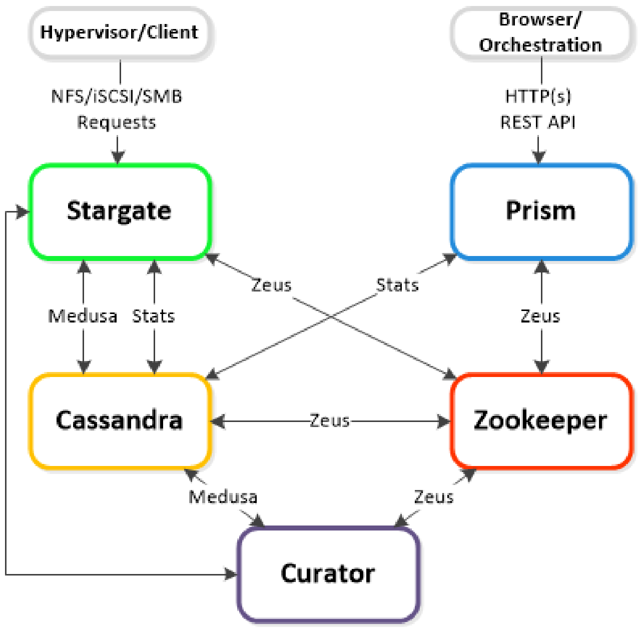
Below is an example of how these Nutanix nodes form NDFS:



top

Cluster Components

The Nutanix platform is composed of the following high-level components:



Cassandra

- **Key Role:** Distributed metadata store
- **Description:** Cassandra stores and manages all of the cluster metadata in a distributed ring like manner based upon a heavily modified Apache Cassandra. The Paxos algorithm is utilized to enforce strict consistency. This service runs on every node in the cluster. Cassandra is accessed via an interface called Medusa.

Zookeeper

- **Key Role:** Cluster configuration manager
- **Description:** Zeus stores all of the cluster configuration including hosts, IPs, state, etc. and is based upon Apache Zookeeper. This service runs on three nodes in the cluster, one of which is elected as a leader. The leader receives all requests and forwards them to the peers. If the leader fails to respond a new leader is automatically elected. Zookeeper is accessed via an interface called Zeus.

Stargate

- **Key Role:** Data I/O manager
- **Description:** Stargate is responsible for all data management and I/O operations and is the main interface from the hypervisor (via NFS, iSCSI or SMB). This service runs on every node in the cluster in order to serve localized I/O.

Curator

- **Key Role:** Map reduce cluster management and cleanup
- **Description:** Curator is responsible for managing and distributing tasks throughout the cluster including disk balancing, proactive scrubbing, and many more items. Curator runs on every node and is controlled by an elected Curator Master who is responsible for the task and job delegation.

Prism

- **Key Role:** UI and API
- **Description:** Prism is the management gateway for component and administrators to configure and monitor the Nutanix cluster. This includes Ncli, the HTML5 UI and REST API. Prism runs on every node in the cluster and uses an elected leader like all components in the cluster.

Genesis

- **Key Role:** Cluster component & service manager
- **Description:** Genesis is a process which runs on each node and is responsible for any services interactions (start/stop/etc.) as well as for the initial configuration. Genesis is a process which runs independently of the cluster and does not require the cluster to be configured/running. The only requirement for genesis to be running is that Zookeeper is up and running. The cluster_init and cluster_status pages are displayed by the genesis process.

Chronos

- **Key Role:** Job and Task scheduler
- **Description:** Chronos is responsible for taking the jobs and tasks resulting from a Curator scan and scheduling/throttling tasks among nodes. Chronos runs on every node and is controlled by an elected Chronos Master who is responsible for the task and job delegation and runs on the same node as the Curator Master.

Cerebro

- **Key Role:** Replication/DR manager
- **Description:** Cerebro is responsible for the replication and DR capabilities of NDFS. This includes the scheduling of snapshots, the replication to remote sites, and the site migration/failover. Cerebro runs on every node in the Nutanix cluster and all nodes participate in replication to remote clusters/sites.

Pithos

- **Key Role:** vDisk configuration manager
- **Description:** Pithos is responsible for vDisk (NDFS file) configuration data. Pithos runs on every node and is built on top of Cassandra.

top

Data Structure

The Nutanix Distributed Filesystem is composed of the following high-level structs:

Storage Pool

- **Key Role:** Group of physical devices
- **Description:** A storage pool is a group of physical storage devices including PCIe SSD, SSD, and HDD devices for the cluster. The storage pool can span multiple Nutanix nodes and is expanded as the cluster scales. In most configurations only a single storage pool is leveraged.

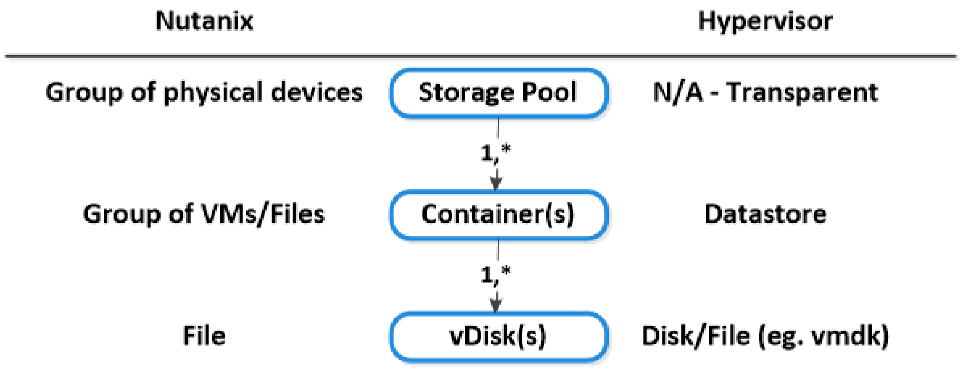
Container

- **Key Role:** Group of VMs/files
- **Description:** A container is a logical segmentation of the Storage Pool and contains a group of VM or files (vDisks). Some configuration options (eg. RF) are configured at the container level, however are applied at the individual VM/file level. Containers typically have a 1 to 1 mapping with a datastore (in the case of NFS/SMB).

vDisk

- **Key Role:** vDisk
- **Description:** A vDisk is any file over 512KB on NDFS including .vmdks and VM hard disks. vDisks are composed of extents which are grouped and stored on disk as an extent group.

Below we show how these map between NDFS and the hypervisor:



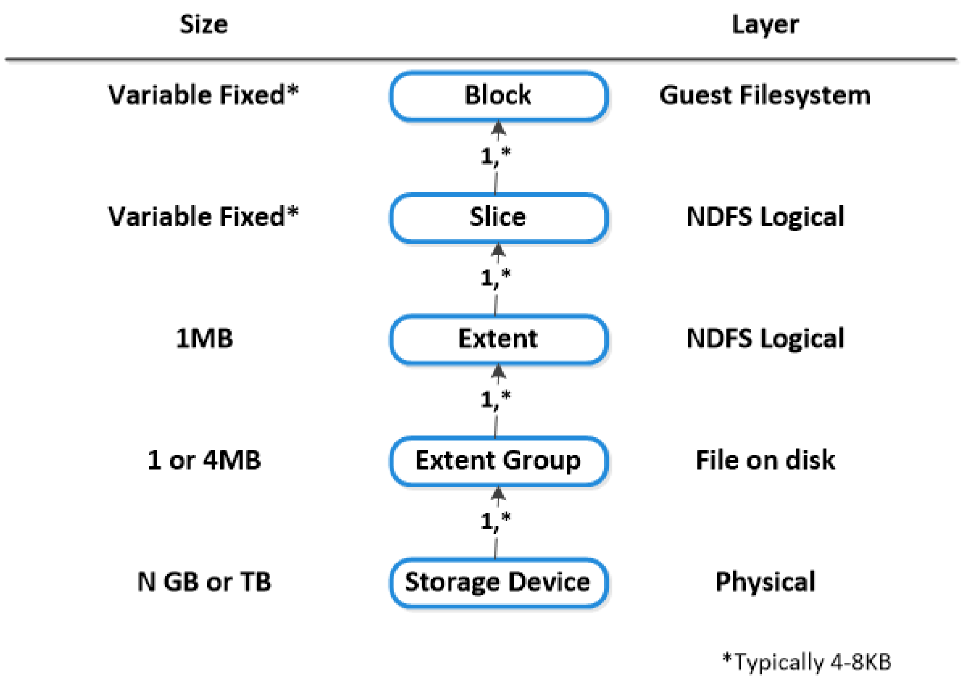
Extent

- **Key Role:** Logically contiguous data
- **Description:** A extent is a 1MB piece of logically contiguous data which consists of n number of contiguous blocks (varies depending on guest OS block size). Extents are written/read/modified on a sub-extent basis (aka slice) for granularity and efficiency. An extent's slice may be trimmed when moving into the cache depending on the amount of data being read/cached.

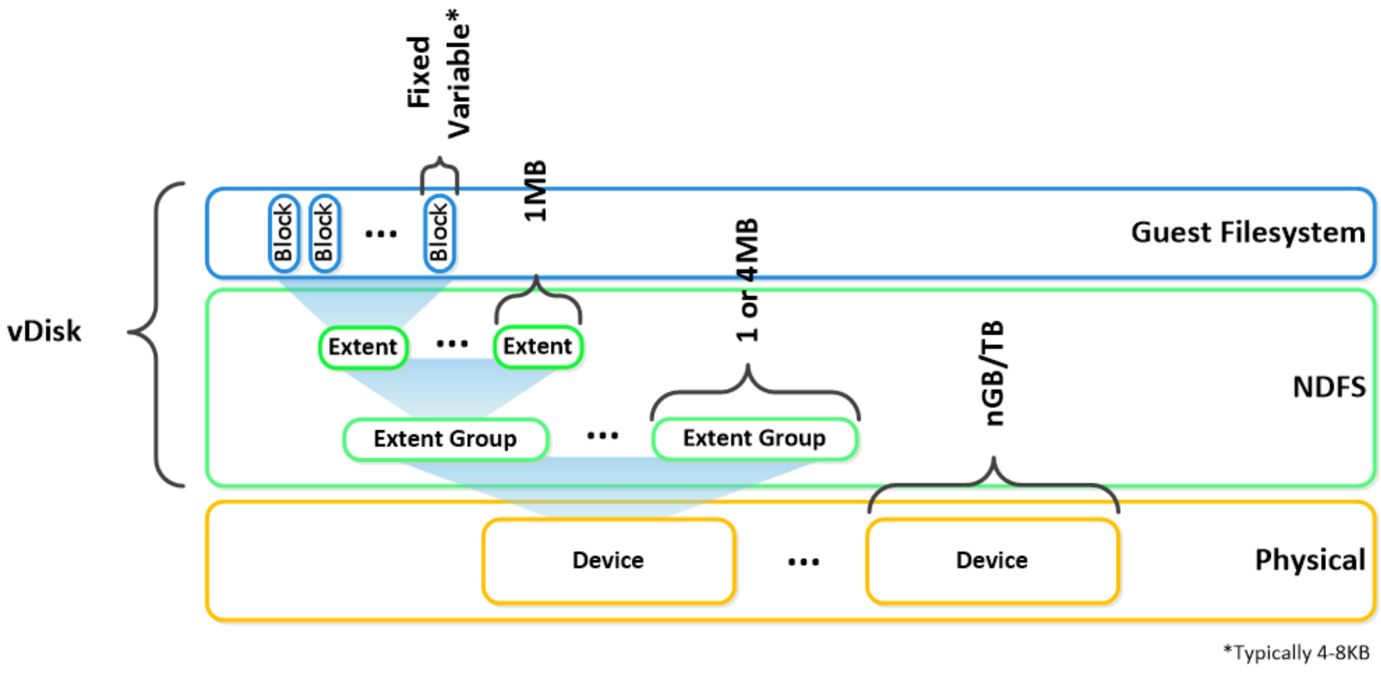
Extent Group

- **Key Role:** Physically contiguous stored data
- **Description:** A extent group is a 1MB or 4MB piece of physically contiguous stored data. This data is stored as a file on the storage device owned by the CVM. Extents are dynamically distributed among extent groups to provide data striping across nodes/disks to improve performance. NOTE: as of 4.0 extent groups can now be either 1MB or 4MB depending on dedupe.

Below we show how these structs relate between the various filesystems:



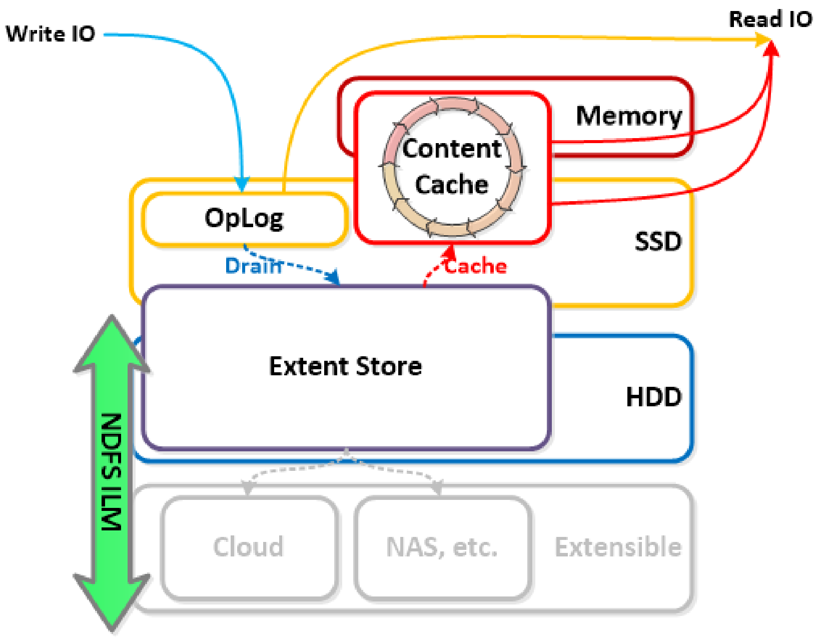
Here is another graphical representation of how these units are logically related:



top

I/O Path Overview

The Nutanix I/O path is composed of the following high-level components:



OpLog

- **Key Role:** Persistent write buffer
- **Description:** The Oplog is similar to a filesystem journal and is built to handle bursty writes, coalesce them and then sequentially drain the data to the extent store. Upon a write the OpLog is synchronously replicated to another n number of CVM's OpLog before the write is acknowledged for data availability purposes. All CVM OpLogs partake in the replication and are dynamically chosen based upon load. The OpLog is stored on the SSD tier on the CVM to provide extremely fast write I/O performance, especially for random I/O workloads. For sequential workloads the OpLog is bypassed and the writes go directly to the extent store. If data is currently sitting in the OpLog and has not been drained, all read requests will be directly fulfilled from the OpLog until they have been drain where they would then be served by the extent store/content cache. For containers where fingerprinting (aka Dedupe) has been enabled, all write I/Os will be fingerprinted using a hashing scheme allowing them to be deduped based upon fingerprint in the content cache.

Extent Store

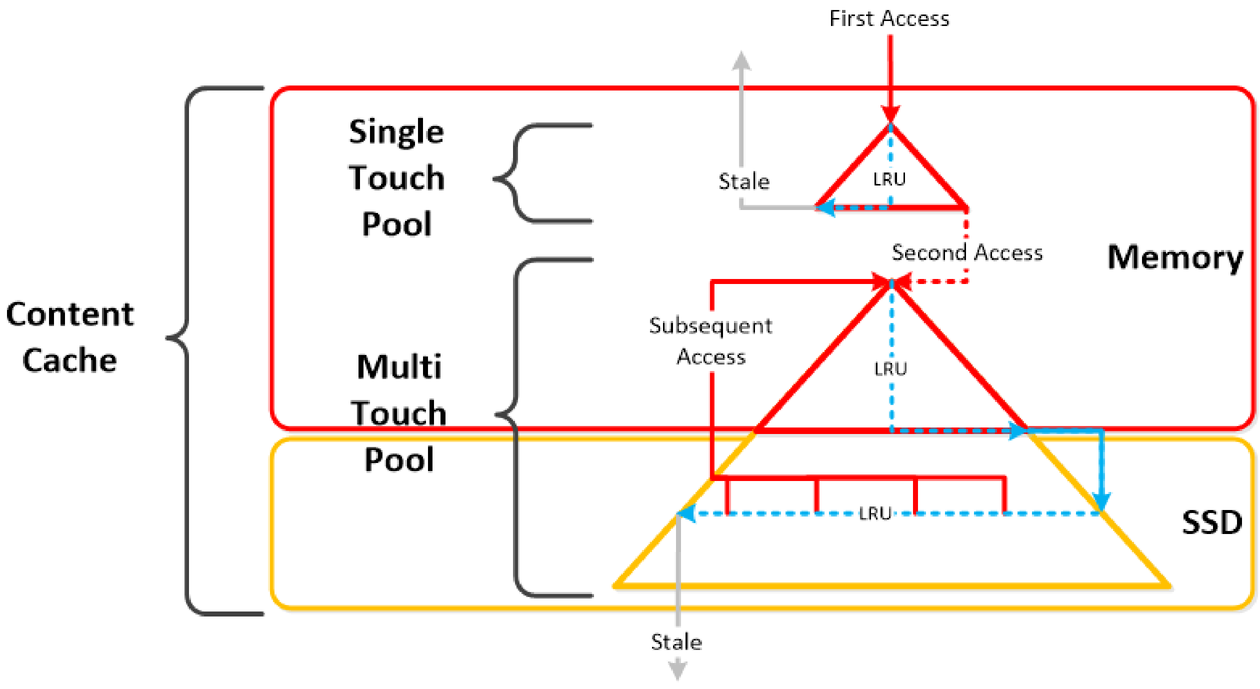
- **Key Role:** Persistent data storage
- **Description:** The Extent Store is the persistent bulk storage of NDFS and spans SSD and HDD and is extensible to facilitate additional devices/tiers. Data entering the extent store is either being A) drained from the OpLog or B) is sequential in nature and has bypassed the OpLog directly. Nutanix ILM will determine tier placement dynamically based upon I/O patterns and will move data between tiers.

Content Cache

- **Key Role:** Dynamic read cache
- **Description:** The Content Cache (aka "Elastic Dedupe Engine") is a deduped read cache which spans both the CVM's memory and SSD. Upon a read request of data not in the cache (or based upon a particular fingerprint) the data will be placed in to the single-touch pool of the content cache which completely sits in memory where it will use LRU until it is ejected from the cache. Any subsequent read request will "move" (no data is actually moved, just cache metadata) the data into the memory portion of the multi-touch pool which consists of both memory and SSD. From here there are two LRU cycles, one for the in-memory piece upon which eviction will move the data to the SSD section of the multi-touch pool where a new LRU counter is assigned. Any read request for data in the

multi-touch pool will cause the data to go to the peak of the multi-touch pool where it will be given a new LRU counter. Fingerprinting is configured at the container level and can be configured via the UI. By default fingerprinting is disabled.

- Below we show a high-level overview of the Content Cache:



Extent Cache

- **Key Role:** In-memory read cache
- **Description:** The Extent Cache is an in-memory read cache that is completely in the CVM's memory. This will store non-fingerprinted extents for containers where fingerprinting and dedupe is disabled. As of version 3.5 this is separate from the Content Cache, however these will be merging in a subsequent release.

top

How It Works

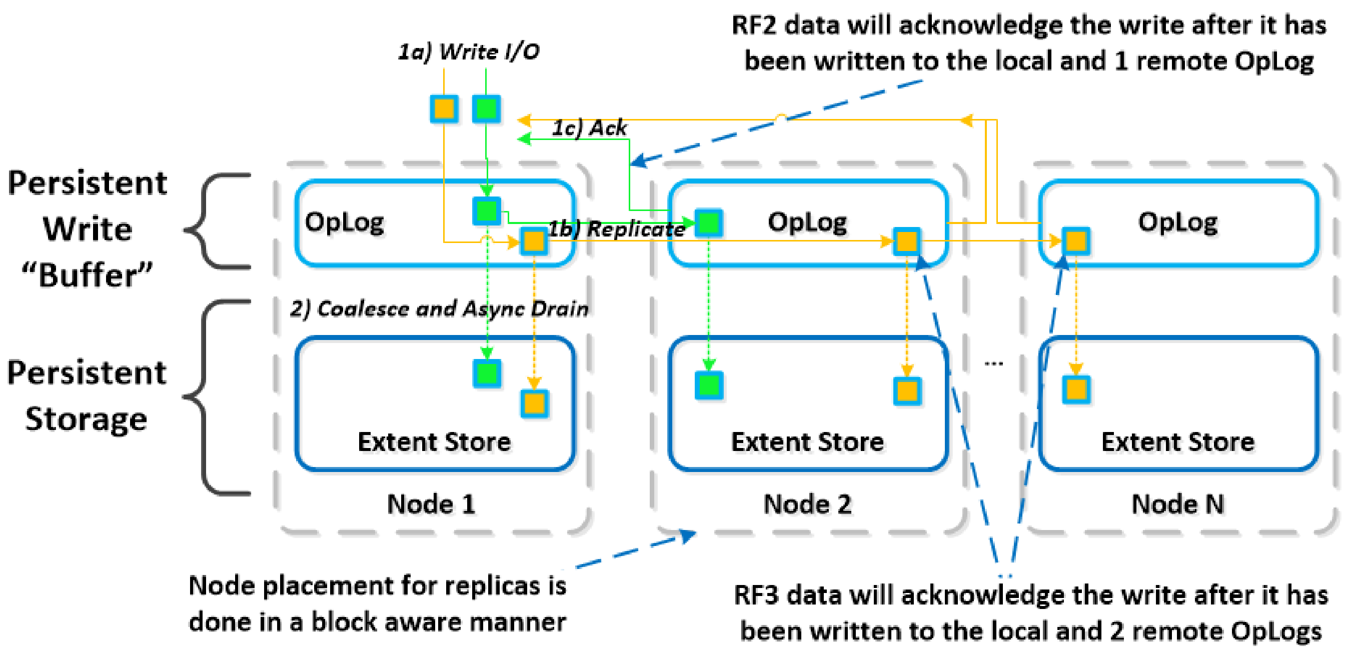
Data Protection

The Nutanix platform currently uses a resiliency factor aka replication factor (RF) and checksum to ensure data redundancy and availability in the case of a node or disk failure or corruption. As explained above the OpLog acts as a staging area to absorb incoming writes onto a low-latency SSD tier. Upon being written to the local OpLog the data is synchronously replicated to another one or two Nutanix CVM's OpLog (dependent on RF) before being acknowledged (Ack) as a successful write to the host. This ensures that the data exists in at least two or three independent locations and is fault tolerant.

NOTE: For RF3 a minimum of 5 nodes is required since metadata will be RF5. Data RF is configured via Prism and is done at the container level.

All nodes participate in OpLog replication to eliminate any "hot nodes" and ensuring linear performance at scale. While the data is being written a checksum is computed and stored as part of its metadata. Data is then asynchronously drained to the extent store where the RF is implicitly maintained. In the case of a node or disk failure the data is then re-replicated among all nodes in the cluster to maintain the RF. Any time the data is read the checksum is computed to ensure the data is valid. In the event where the checksum and data don't match the replica of the data will be read and will replace the non-valid copy.

Below we show an example of what this logically looks like:

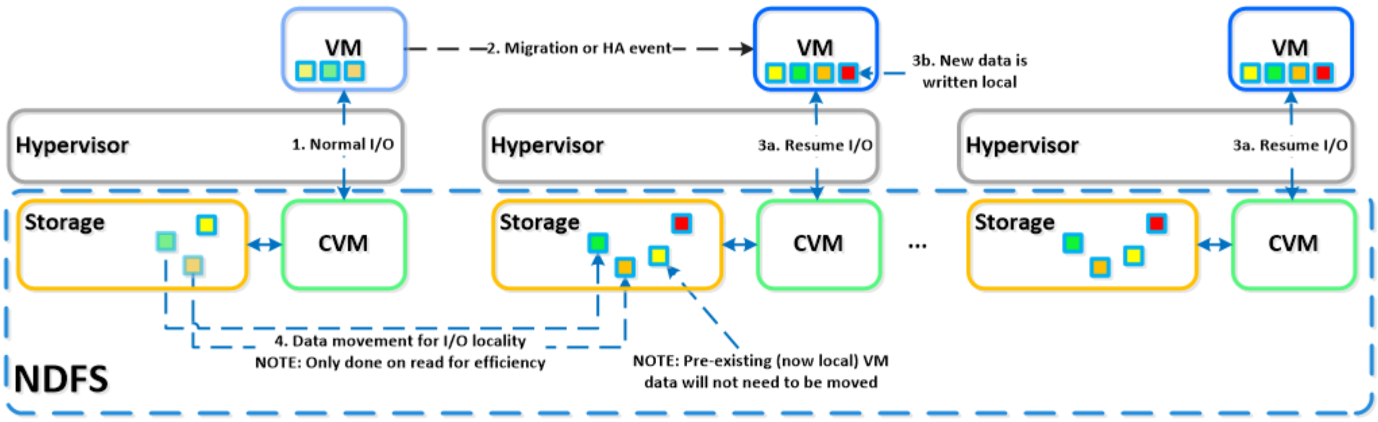

1a) Write I/O
RF2 data will acknowledge the write after it has been written to the local and 1 remote OpLog
1c) Ack
Persistent Write "Buffer"
OpLog
1b) Replicate
OpLog
OpLog
2) Coalesce and Async Drain
Persistent Storage
Extent Store
Extent Store
Extent Store
...
Node 1
Node 2
Node N
Node placement for replicas is done in a block aware manner
RF3 data will acknowledge the write after it has been written to the local and 2 remote OpLogs

Data Locality

Being a converged (compute+storage) platform, I/O and data locality is key to cluster and VM performance with Nutanix. As explained above in the I/O path, all read/write IOs are served by the local Controller VM (CVM) which is on each hypervisor adjacent to normal VMs. A VM's data is served locally from the CVM and sits on local disks under the CVM's control. When a VM is moved from one hypervisor node to another (or during a HA event) the newly migrated VM's data will be served by the now local CVM.

When reading old data (stored on the now remote node/CVM) the I/O will be forwarded by the local CVM to the remote CVM. All write I/Os will occur locally right away. NDFS will detect the I/Os are occurring from a different node and will migrate the data locally in the background allowing for all read I/Os to now be served locally. The data will only be migrated on a read as to not flood the network.

Below we show an example of how data will "follow" the VM as it moves between hypervisor nodes:

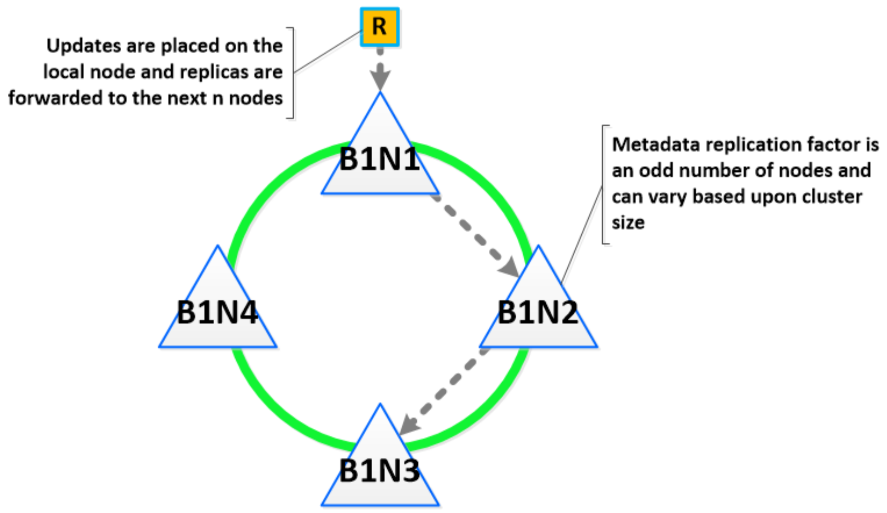

VM
2. Migration or HA event
VM
3b. New data is written local
VM
Hypervisor
1. Normal I/O
Hypervisor
3a. Resume I/O
Hypervisor
3a. Resume I/O
Storage
CVM
Storage
CVM
...
Storage
CVM
4. Data movement for I/O locality
NOTE: Only done on read for efficiency
NOTE: Pre-existing (now local) VM data will not need to be moved
NDFS

Scalable Metadata

Metadata is at the core of any intelligent system and is even more critical for any filesystem or storage array. In terms of NDFS there are a few key structs that are critical for its success: it has to be right 100% of the time (aka. "strictly consistent"), it has to be scalable, and it has to perform, at massive scale. As mentioned in the architecture section above, NDFS utilizes a "ring like" structure as a key-value store which stores essential metadata as well as other platform data (eg. stats, etc.).

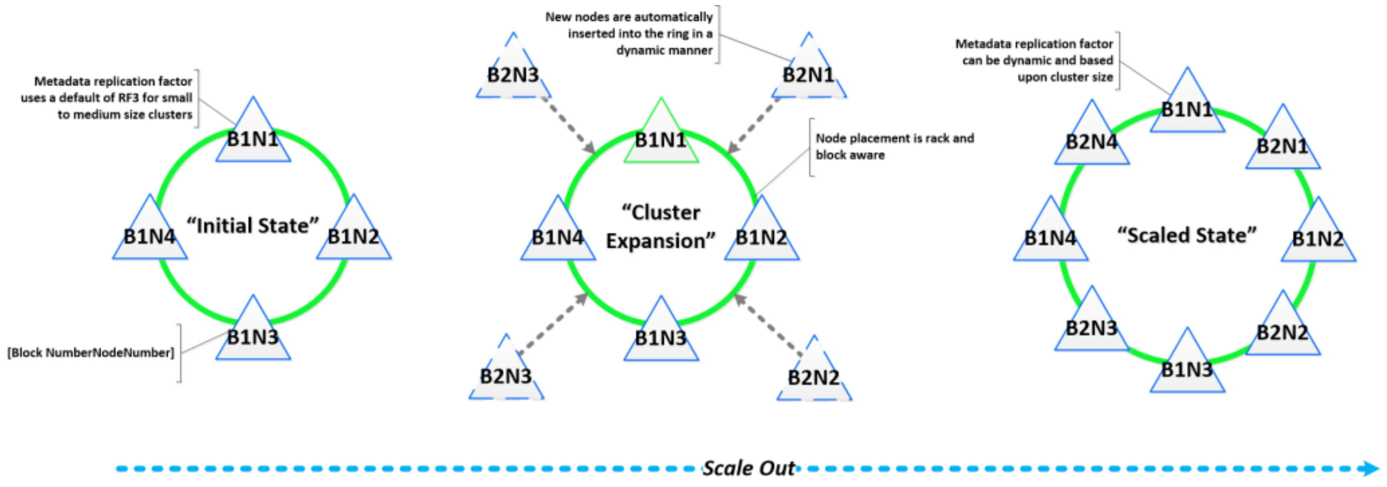
In order to ensure metadata availability and redundancy a RF is utilized among an odd amount of nodes (eg. 3, 5, etc.). Upon a metadata write or update the row is written to a node in the ring and then replicated to n number of peers (where n is dependent on cluster size). A majority of nodes must agree before anything is committed which is enforced using the paxos algorigthm. This ensures strict consistency for all data and metadata stored as part of the platform.

Below we show an example of a metadata insert/update for a 4 node cluster:


Updates are placed on the local node and replicas are forwarded to the next n nodes
R
B1N1
Metadata replication factor is an odd number of nodes and can vary based upon cluster size
B1N4
B1N2
B1N3

Performance at scale is also another important struct for NDFS metadata. Contrary to traditional dual-controller or "master" models, each Nutanix node is responsible for a subset of the overall platform's metadata. This eliminates the traditional bottlenecks by allowing metadata to be served and manipulated by all nodes in the cluster. A consistent hashing scheme is utilized to minimize the redistribution of keys during cluster size modifications (aka. "add/remove node") When the cluster scales (eg. from 4 to 8 nodes), the nodes are inserted throughout the ring between nodes for "block awareness" and reliability.

Below we show an example of the metadata "ring" and how it scales:


Metadata replication factor uses a default of RF3 for small to medium size clusters
B1N1
"Initial State"
B1N4
B1N2
B1N3
[Block NumberNodeNumber]
New nodes are automatically inserted into the ring in a dynamic manner
B2N3
B2N1
B1N1
Node placement is rack and block aware
"Cluster Expansion"
B1N4
B1N2
B1N3
B2N3
B2N2
Metadata replication factor can be dynamic and based upon cluster size
B1N1
B2N4
B2N1
"Scaled State"
B1N4
B1N2
B2N3
B2N2
B1N3
Scale Out

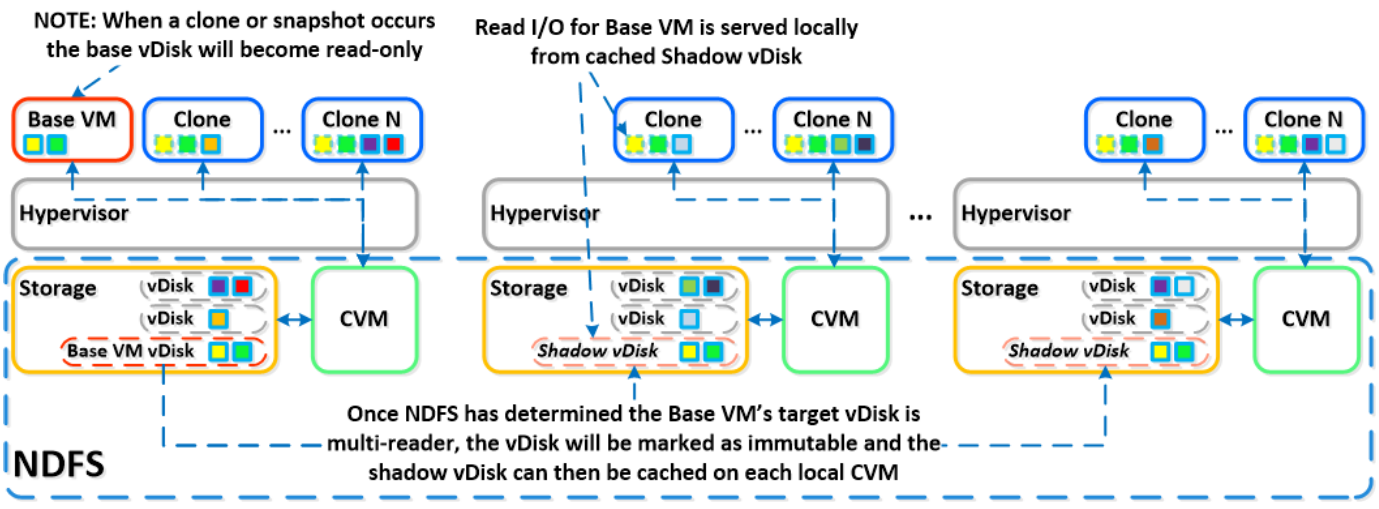
Shadow Clones

The Nutanix Distributed Filesystem has a feature called 'Shadow Clones' which allows for distributed caching of particular vDisks or VM data which is in a 'multi-reader' scenario. A great example of this is during a VDI deployment many 'linked clones' will be forwarding read requests to a central master or 'Base VM'. In the case of VMware View this is called the replica disk and is read by all linked clones and in XenDesktop this is called the MCS Master VM. This will also work in any scenario which may be a multi-reader scenario (eg. deployment servers, repositories, etc.).

Data or I/O locality is critical for the highest possible VM performance and a key struct of NDFS. With Shadow Clones, NDFS will monitor vDisk access trends similar to what it does for data locality. However in the case there are requests occurring from more than two remote CVMs (as well as the local CVM), and all of the requests are read I/O, the vDisk will be marked as immutable. Once the disk has been marked as immutable the vDisk can then be cached locally by each CVM making read requests to it (aka Shadow Clones of the base vDisk). This will allow VMs on each node to read the Base VM's vDisk locally.

In the case of VDI, this means the replica disk can be cached by each node and all read requests for the base will be served locally. NOTE: The data will only be migrated on a read as to not flood the network and allow for efficient cache utilization. In the case where the Base VM is modified the Shadow Clones will be dropped and the process will start over. Shadow clones are disabled by default (as of 3.5) and can be enabled/disabled using the following NCLI command: ncli cluster edit-params enable-shadow-clones=true.

Below we show an example of how Shadow Clones work and allow for distributed caching:

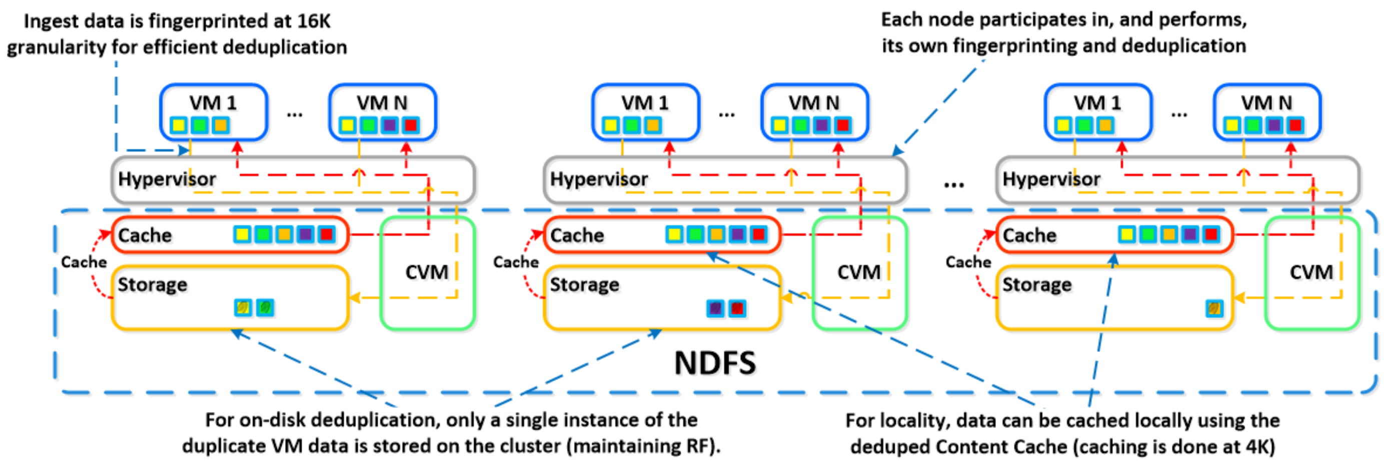

NOTE: When a clone or snapshot occurs the base vDisk will become read-only
Read I/O for Base VM is served locally from cached Shadow vDisk
Base VM
Clone
Clone N
Hypervisor
Storage
vDisk
vDisk
Base VM vDisk
Shadow vDisk
CVM
Once NDFS has determined the Base VM's target vDisk is multi-reader, the vDisk will be marked as immutable and the shadow vDisk can then be cached on each local CVM
NDFS

Elastic Dedupe Engine

The Elastic Dedupe Engine is a software based feature of NDFS which allows for data deduplication in the capacity (HDD) and performance (SSD/Memory) tiers. Sequential streams of data are fingerprinted during ingest using a SHA-1 hash at a 16K granularity. This fingerprint is only done on data ingest and is then stored persistently as part of the written block's metadata. NOTE: Initially a 4K granularity was used for fingerprinting, however after testing 16K offered the best blend of dedupability with reduced metadata overhead. When deduped data is pulled into the cache this is done at 4K.

Contrary to traditional approaches which utilize background scans, requiring the data to be re-read, Nutanix performs the fingerprint in-line on ingest. For duplicate data that can be deduplicated in the capacity tier the data does not need to be scanned or re-read, essentially duplicate copies can be removed.

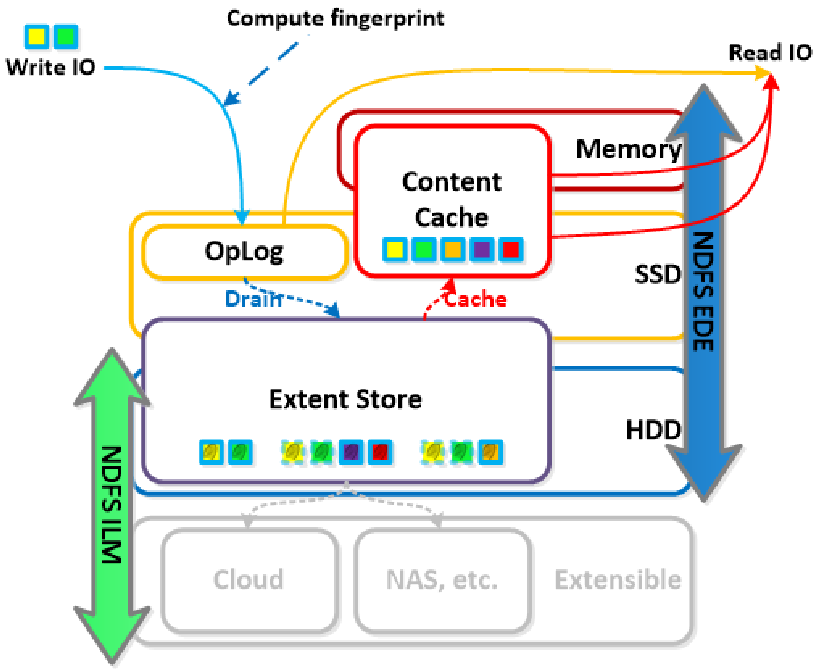
Below we show an example of how the Elastic Dedupe Engine scales and handles local VM I/O requests:


Ingest data is fingerprinted at 16K granularity for efficient deduplication
Each node participates in, and performs, its own fingerprinting and deduplication
VM 1
VM N
Hypervisor
Cache
Storage
CVM
NDFS
For on-disk deduplication, only a single instance of the duplicate VM data is stored on the cluster (maintaining RF).
For locality, data can be cached locally using the deduped Content Cache (caching is done at 4K)

Fingerprinting is done during data ingest of data with an I/O size of 64K or greater. Intel acceleration is leveraged for the SHA-1 computation which accounts for very minimal CPU overhead. In cases where fingerprinting is not done during ingest (eg. smaller I/O sizes), fingerprinting can be done as a background process. The Elastic Deduplication Engine spans both the capacity disk tier (HDD), but also the performance tier (SSD/Memory). As duplicate data is determined, based upon multiple copies of the same fingerprints, a background process will remove the duplicate data using the NDFS Map Reduce framework (curator).

For data that is being read, the data will be pulled into the NDFS Content Cache which is a multi-tier/pool cache. Any subsequent requests for data having the same fingerprint will be pulled directly from the cache. To learn more about the Content Cache and pool structure, please refer to the 'Content Cache' sub-section in the I/O path overview, or click HERE.

Below we show an example of how the Elastic Dedupe Engine interacts with the NDFS I/O path:

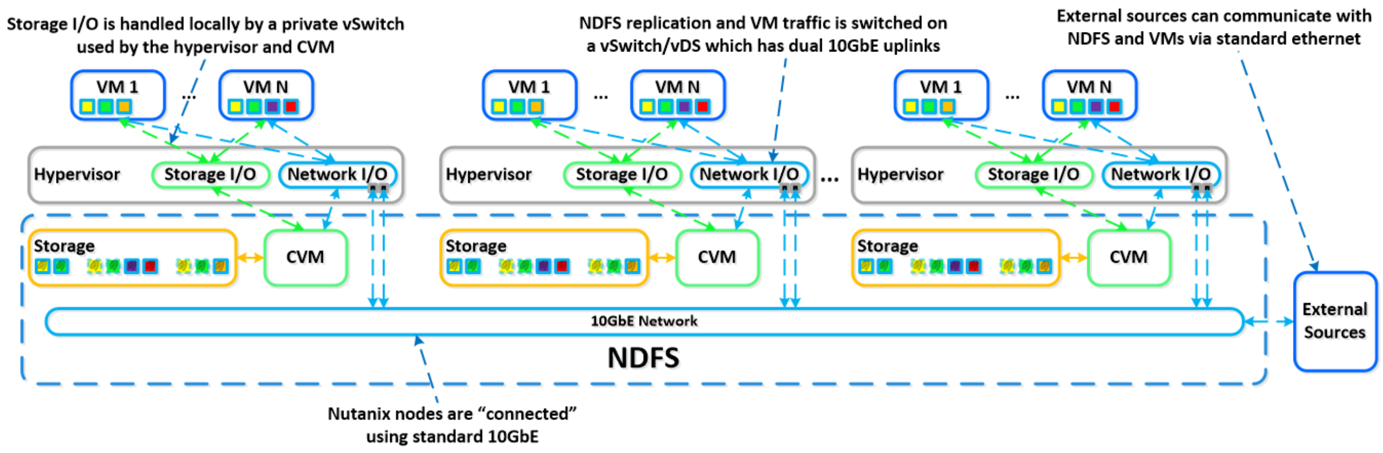


Networking and I/O

The Nutanix platform does not leverage any backplane for inter-node communication and only relies on a standard 10GbE network. All storage I/O for VMs running on a Nutanix node is handled by the hypervisor on a dedicated private network. The I/O request will be handled by the hypervisor which will then forward the request to the private IP on the local CVM. The CVM will then perform the remote replication with other Nutanix nodes using its external IP over the public 10GbE network.

For all read requests these will be served completely locally in most cases and never touch the 10GbE network. This means that the only traffic touching the public 10GbE network will be NDFS remote replication traffic and VM network I/O. There will however be cases where the CVM will forward requests to other CVMs in the cluster in the case of a CVM being down or data being remote. Also, cluster wide tasks such as disk balancing will temporarily generate I/O on the 10GbE network.

Below we show an example of how the VM's I/O path interacts with the private and public 10GbE network:

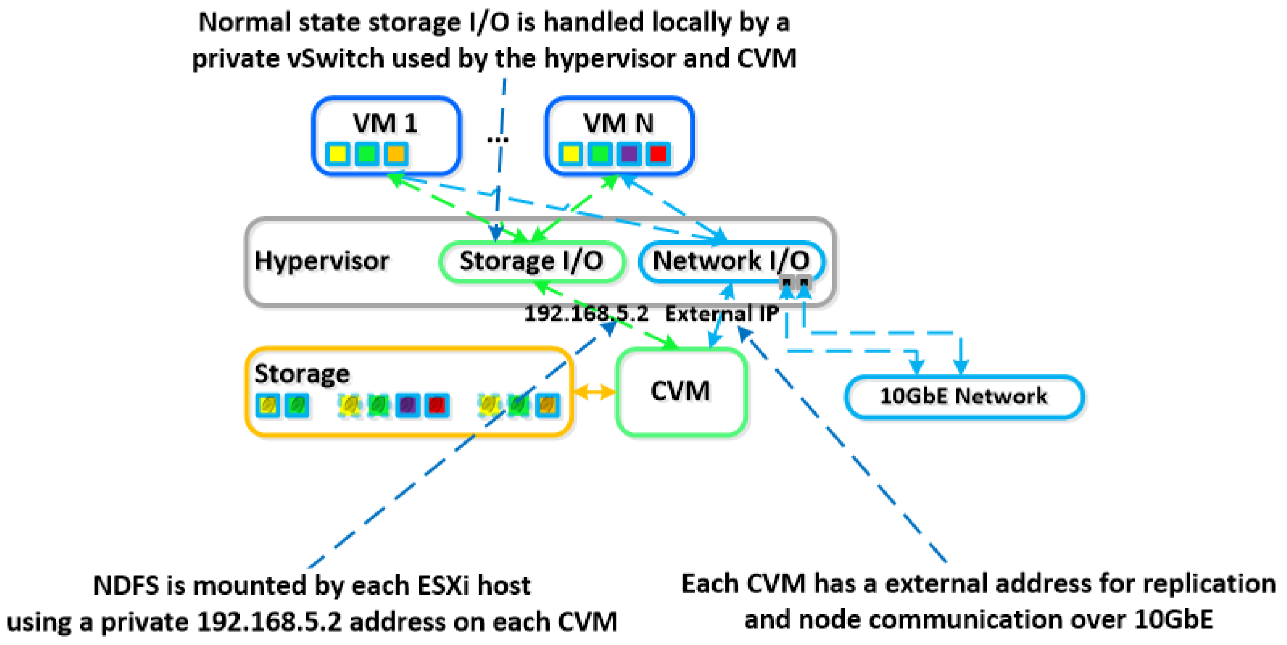

Storage I/O is handled locally by a private vSwitch used by the hypervisor and CVM
NDFS replication and VM traffic is switched on a vSwitch/vDS which has dual 10GbE uplinks
External sources can communicate with NDFS and VMs via standard ethernet
VM 1
VM N
Hypervisor
Storage I/O
Network I/O
Storage
CVM
10GbE Network
NDFS
External Sources
Nutanix nodes are "connected" using standard 10GbE

CVM Autopathing

Reliability and resiliency is a key, if not the most important, piece to NDFS. Being a distributed system NDFS is built to handle component, service and CVM failures. In this section I'll cover how CVM "failures" are handled (I'll cover how we handle component failures in future update). A CVM "failure" could include a user powering down the CVM, a CVM rolling upgrade, or any event which might bring down the CVM.

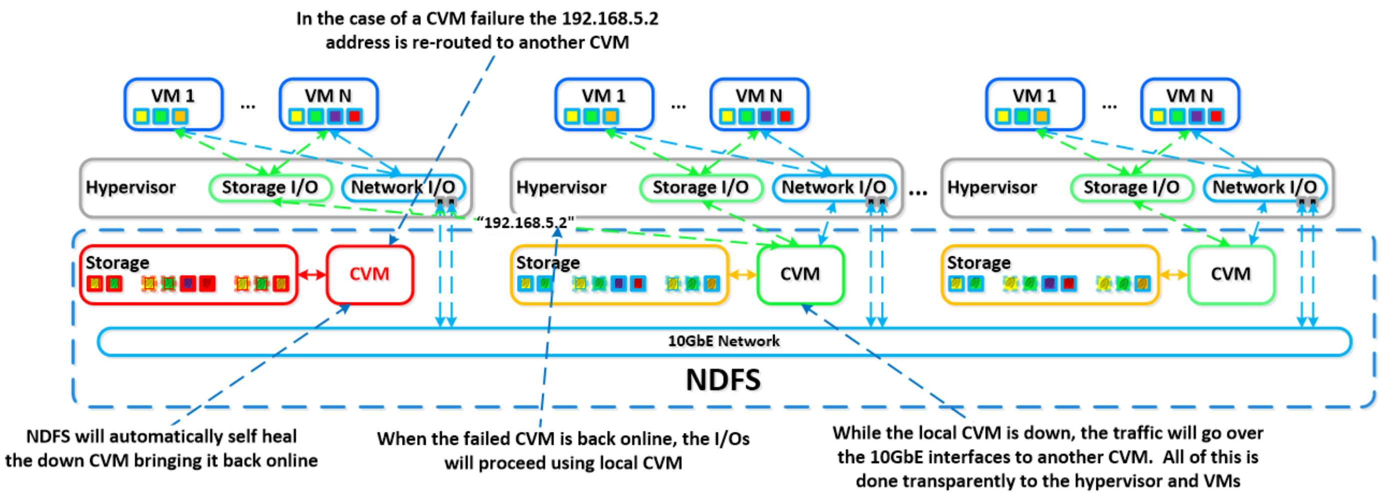
NDFS has a feature called autopathing where when a local CVM becomes unavailable the I/Os are then transparently handled by other CVMs in the cluster. The hypervisor and CVM communicate using a private 192.168.5.0 network on a dedicated vSwitch (more on this above). This means that for all storage I/Os these are happening to the internal IP addresses on the CVM (192.168.5.2). The external IP address of the CVM is used for remote replication and for CVM communication.

Below we show an example of what this looks like:


Normal state storage I/O is handled locally by a private vSwitch used by the hypervisor and CVM
VM 1
VM N
Hypervisor
Storage I/O
Network I/O
192.168.5.2
External IP
Storage
CVM
10GbE Network
NDFS is mounted by each ESXi host using a private 192.168.5.2 address on each CVM
Each CVM has a external address for replication and node communication over 10GbE

In the event of a local CVM failure the local 192.168.5.2 addresses previously hosted by the local CVM is unavailable. NDFS will automatically detect this outage and will redirect these I/Os to another CVM in the cluster over 10GbE. The re-routing is done transparently to the hypervisor and VMs running on the host. This means that even if a CVM is powered down the VMs will still continue to be able to perform I/Os to NDFS. NDFS is also self-healing meaning it will detect the CVM has been powered off and will automatically reboot or power-on the local CVM. Once the local CVM is back up and available, traffic will then seamlessly be transferred back and served by the local CVM.

Below we show a graphical representation of how this looks for a failed CVM:

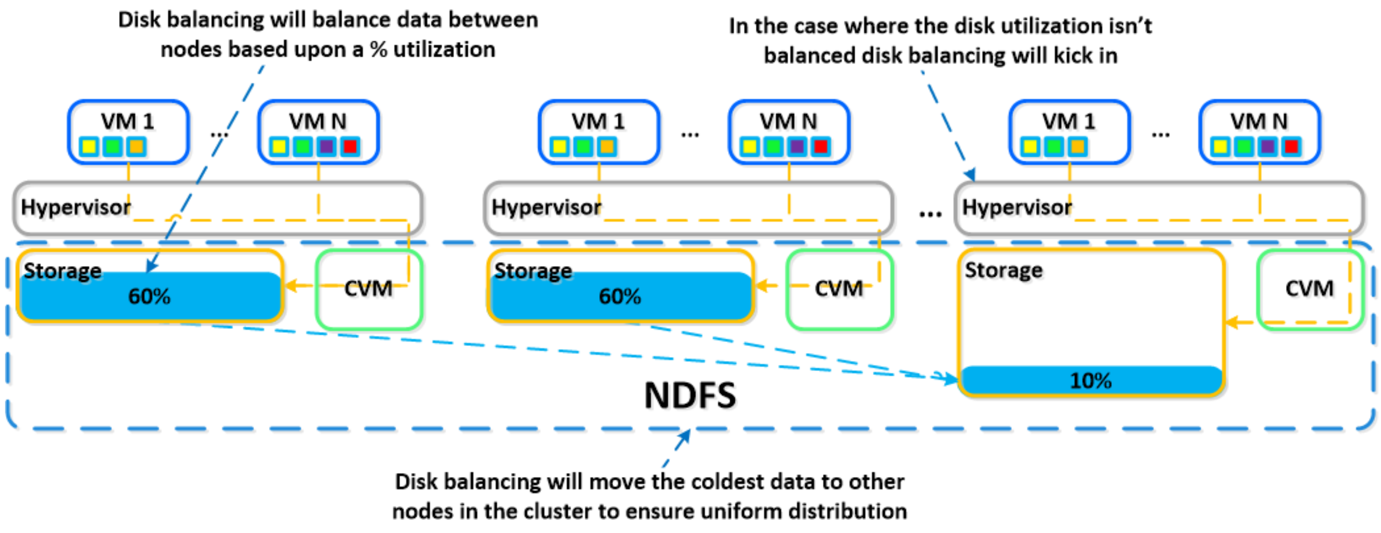

In the case of a CVM failure the 192.168.5.2 address is re-routed to another CVM
VM 1
VM N
Hypervisor
Storage I/O
Network I/O
"192.168.5.2"
Storage
CVM
10GbE Network
NDFS
NDFS will automatically self heal the down CVM bringing it back online
When the failed CVM is back online, the I/Os will proceed using local CVM
While the local CVM is down, the traffic will go over the 10GbE interfaces to another CVM. All of this is done transparently to the hypervisor and VMs

Disk Balancing

NDFS is designed to be a very dynamic platform which can react to various workloads as well as allow heterogeneous node types: compute heavy (3050, etc.) and storage heavy (60X0, etc.) to be mixed in a single cluster. Ensuring uniform distribution of data is an important item when mixing nodes with larger storage capacities.

NDFS has a native feature called disk balancing which is used to ensure uniform distribution of data throughout the cluster. Disk balancing works on a node's utilization of its local storage capacity and is integrated with NDFS ILM. Its goal is to keep utilization uniform among nodes once the utilization has breached a certain threshold.

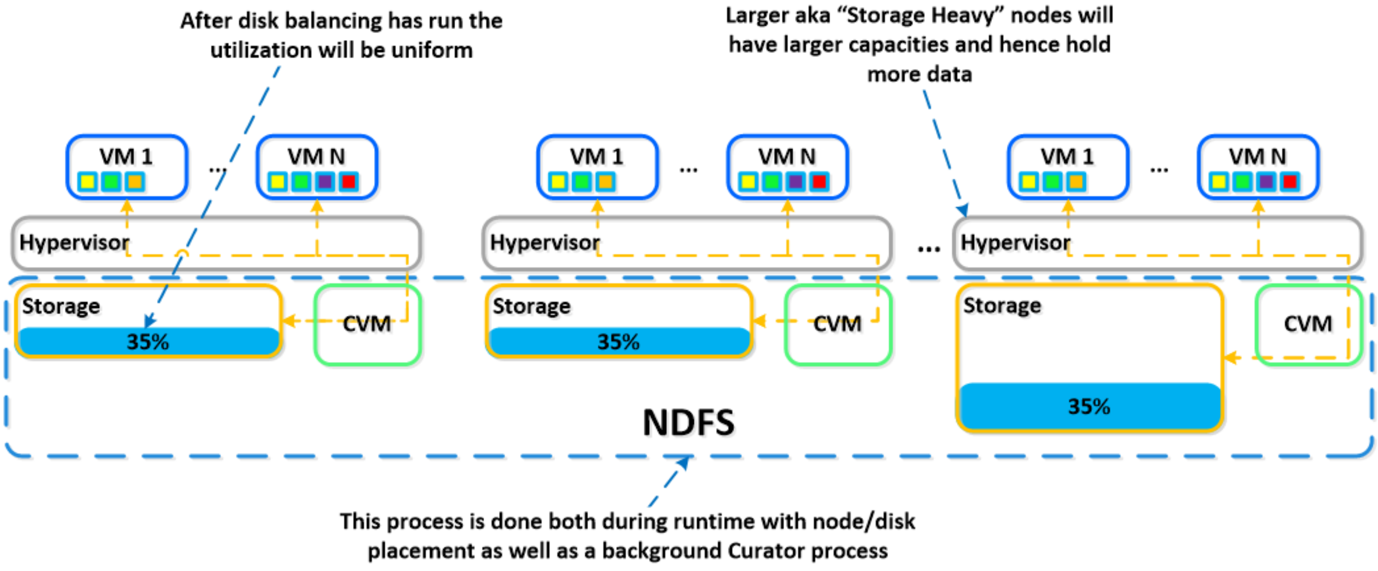
Below we show an example of a mixed cluster (3050 + 6050) in a "unbalanced" state:


Disk balancing will balance data between nodes based upon a % utilization
In the case where the disk utilization isn't balanced disk balancing will kick in
VM 1
VM N
Hypervisor
Storage
60%
CVM
10%
NDFS
Disk balancing will move the coldest data to other nodes in the cluster to ensure uniform distribution

Disk balancing leverages the NDFS Curator framework and is run as a scheduled process as well as when a threshold has been breached (eg. local node capacity utilization > n %). In the case where the data is not balanced Curator will determine which data needs to be moved and will distribute the tasks to nodes in the cluster. In the case where the node types are homogeneous (eg. 3050) utilization should be fairly uniform.

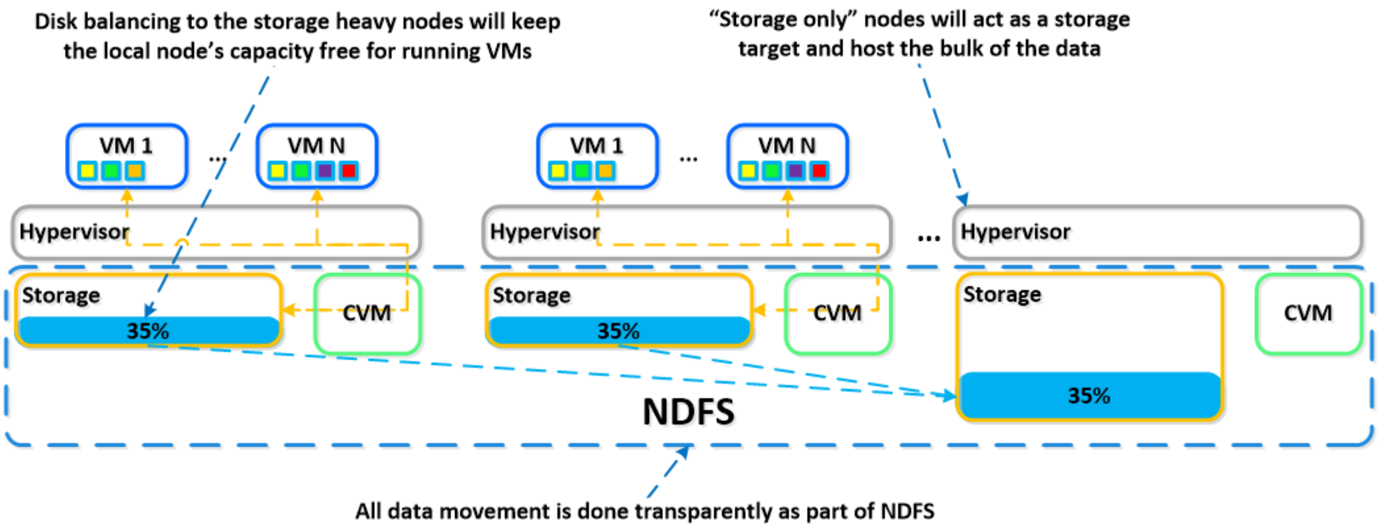
However, if there are certain VMs running on a node which are writing much more data than others there can become a skew in the per node capacity utilization. In this case disk balancing would run and move the coldest data on that node to other nodes in the cluster. In the case where the node types are heterogeneous (eg. 3050 + 6020/50/70), or where a node may be used in a "storage only" mode (not running any VMs), there will likely be a requirement to move data.

Below we show an example the mixed cluster after disk balancing has been run in a "balanced" state:


After disk balancing has run the utilization will be uniform
Larger aka "Storage Heavy" nodes will have larger capacities and hence hold more data
VM 1
VM N
Hypervisor
Storage
35%
CVM
NDFS
This process is done both during runtime with node/disk placement as well as a background Curator process

In some scenarios customers might run some nodes in a "storage only" state where only the CVM will run on the node whose primary purpose is bulk storage capacity. In this case the full nodes memory can be added to the CVM to provide a much larger read cache.

Below we show an example of how a storage only node would look in a mixed cluster with disk balancing moving data to it from the active VM nodes:


Disk balancing to the storage heavy nodes will keep the local node's capacity free for running VMs
"Storage only" nodes will act as a storage target and host the bulk of the data
VM 1
VM N
Hypervisor
Storage
35%
CVM
NDFS
All data movement is done transparently as part of NDFS

Software-Defined Controller Architecture

As mentioned above (likely numerous times), the Nutanix platform is a software based solution which ships as a bundled software + hardware appliance. The controller VM is where the vast majority of the Nutanix software and logic sits and was designed from the beginning to be an extensible and pluggable architecture.

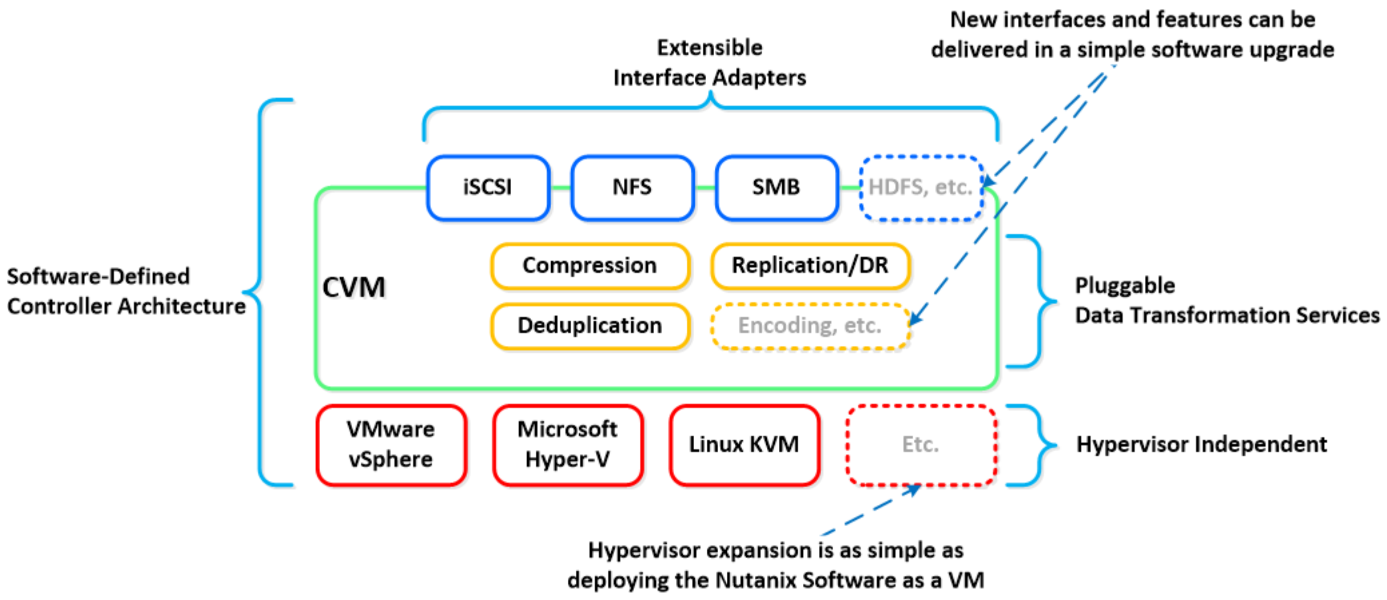
A key benefit to being software defined and not relying upon any hardware offloads or constructs is around extensibility. Like with any product life cycle there will always be advancements and new features which are introduced. By not relying on any custom ASIC/FPGA or hardware capabilities, Nutanix can develop and deploy these new features through a simple software update. This means that the deployment of a new feature (say deduplication) can be deployed by upgrading the current version of the Nutanix software. This also allows newer generation features to be deployed on legacy hardware models.

For example, say you're running a workload running an older version of Nutanix software on a prior generation hardware platform (eg. 2400). The running software version doesn't provide deduplication capabilities which your workload could benefit greatly from. To get these features you perform a rolling upgrade of the Nutanix software version while the workload is running, and whala you now have deduplication. It's really that easy.

Similar to features, the ability to create new "adapters" or interfaces into NDFS is another key capability. When the product first shipped it solely supported iSCSI for I/O from the hypervisor, this has now grown to include NFS and SMB. In the future there is the ability to create new adapters for various workloads and hypervisors (HDFS, etc.). And again, all deployed via a software update.

This is contrary to mostly all legacy infrastructures as a hardware upgrade or software purchase was normally required to get the "latest and greatest" features. With Nutanix it's different, since all features are deployed in software they can run on any hardware platform, any hypervisor and be deployed through simple software upgrades.

Below we show a logical representation of what this software-defined controller framework looks like:

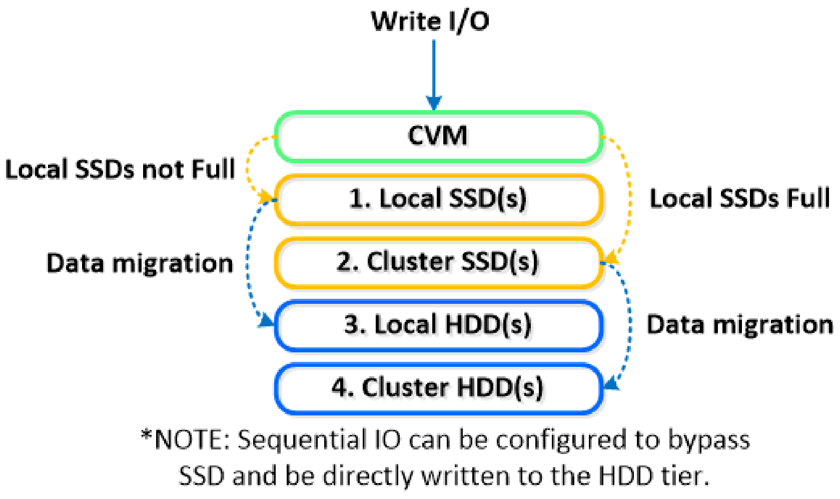

New interfaces and features can be delivered in a simple software upgrade
Extensible Interface Adapters
iSCSI
NFS
SMB
HDFS, etc.
Software-Defined Controller Architecture
CVM
Compression
Replication/DR
Deduplication
Encoding, etc.
Pluggable Data Transformation Services
VMware vSphere
Microsoft Hyper-V
Linux KVM
Etc.
Hypervisor Independent
Hypervisor expansion is as simple as deploying the Nutanix Software as a VM

Storage Tiering and Prioritization

The Disk Balancing section above talked about how storage capacity was pooled among all nodes in a Nutanix cluster and that ILM would be used to keep hot data local. A similar concept applies to disk tiering in which the cluster's SSD and HDD tiers are cluster wide and NDFS ILM is responsible for triggering data movement events.

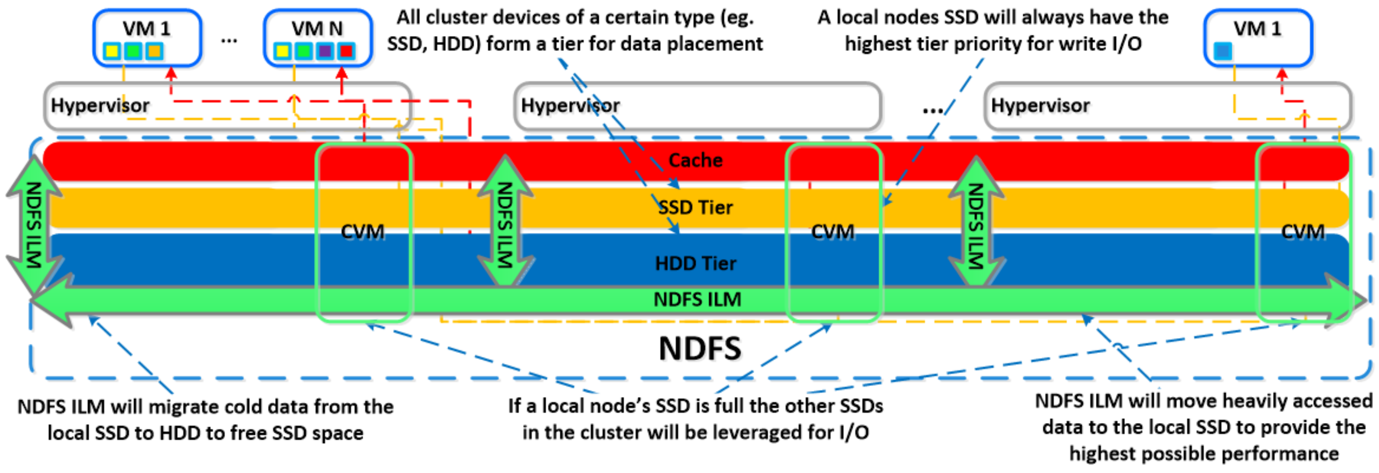
A local node's SSD tier is always the highest priority tier for all I/O generated by VMs running on that node, however all of the cluster's SSD resources are made available to all nodes within the cluster. The SSD tier will always offer the highest performance and is a very important thing to manage for hybrid arrays.

The tier prioritization can be classified at a high-level by the following:

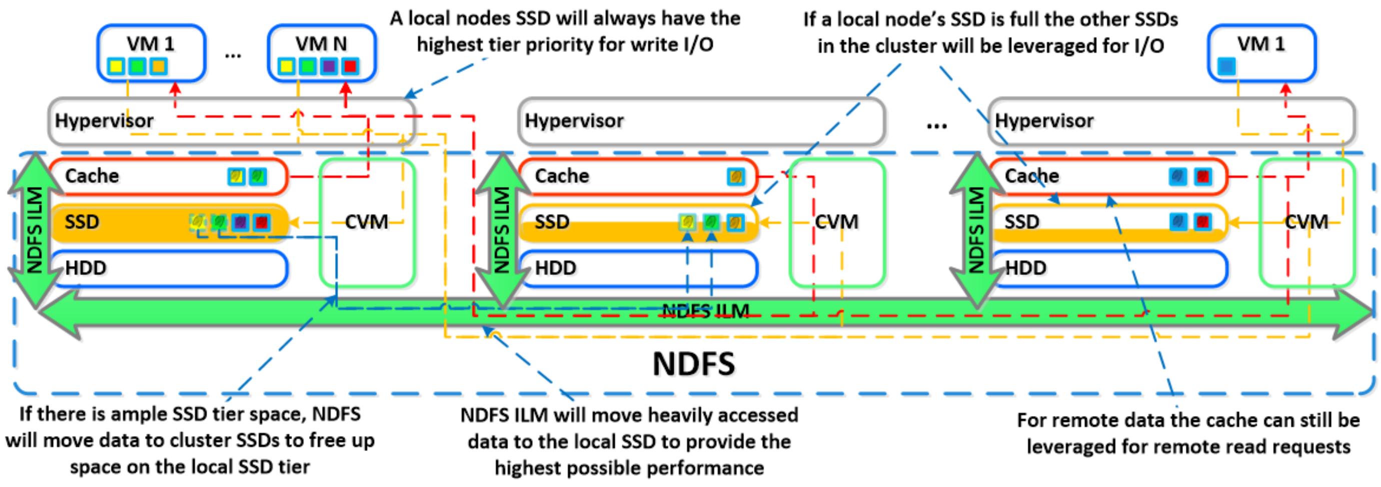


Specific types of resources (eg. SSD, HDD, etc.) are pooled together and form a cluster wide storage tier. This means that any node within the cluster can leverage the full tier capacity, regardless if it is local or not.

Below we show a high level example of how this pooled tiering looks:

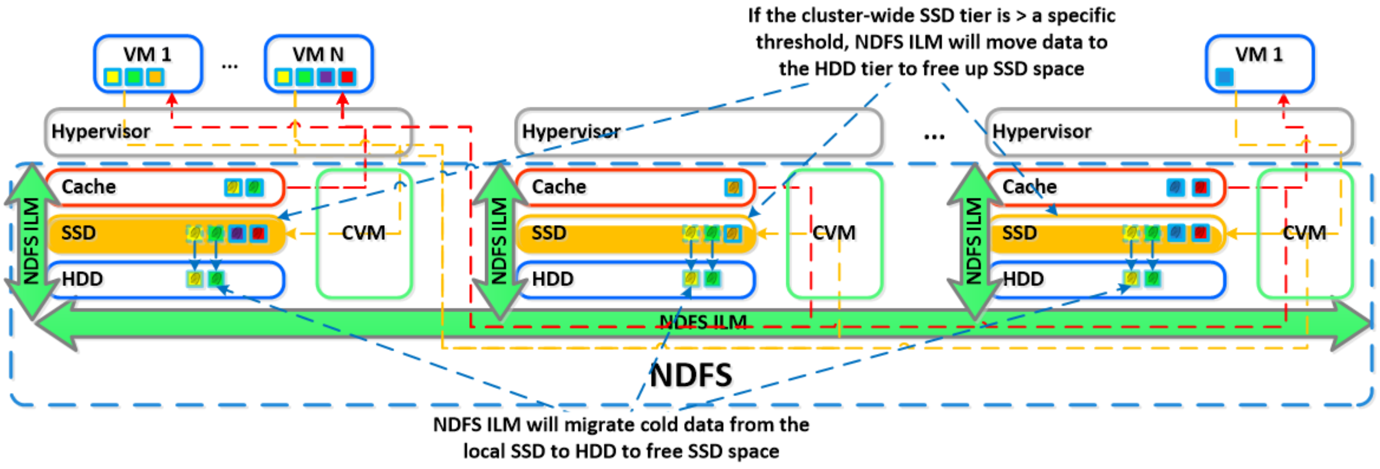

VM 1
...
VM N
All cluster devices of a certain type (eg. SSD, HDD) form a tier for data placement
A local nodes SSD will always have the highest tier priority for write I/O
VM 1
Hypervisor
Hypervisor
...
Hypervisor
Cache
SSD Tier
HDD Tier
CVM
CVM
CVM
NDFS ILM
NDFS ILM
NDFS ILM
NDFS ILM
NDFS
NDFS ILM will migrate cold data from the local SSD to HDD to free SSD space
If a local node's SSD is full the other SSDs in the cluster will be leveraged for I/O
NDFS ILM will move heavily accessed data to the local SSD to provide the highest possible performance

A common question is what happens when a local node's SSD becomes full? As mentioned in the Disk Balancing section a key concept is trying to keep uniform utilization of devices within disk tiers. In the case where a local node's SSD utilization is high, disk balancing will kick in to move the coldest data on the local SSDs to the other SSDs throughout the cluster. This will free up space on the local SSD to allow the local node to write to SSD locally instead of going over the network. A key point to mention is that all CVMs and SSDs are used for this remote I/O to eliminate any potential bottlenecks and remediate some of the hit by performing I/O over the network.


VM 1
...
VM N
A local nodes SSD will always have the highest tier priority for write I/O
If a local node's SSD is full the other SSDs in the cluster will be leveraged for I/O
VM 1
Hypervisor
Hypervisor
...
Hypervisor
Cache
SSD
HDD
CVM
Cache
SSD
HDD
CVM
Cache
SSD
HDD
CVM
NDFS ILM
NDFS ILM
NDFS ILM
NDFS ILM
NDFS
If there is ample SSD tier space, NDFS will move data to cluster SSDs to free up space on the local SSD tier
NDFS ILM will move heavily accessed data to the local SSD to provide the highest possible performance
For remote data the cache can still be leveraged for remote read requests

The other case is when the overall tier utilization breaches a specific threshold [curator_tier_usage_ilm_threshold_percent (Default=75)] where NDFS ILM will kick in and as part of a Curator job will down-migrate data from the SSD tier to the HDD tier. This will bring utilization within the threshold mentioned above or free up space by the following amount [curator_tier_free_up_percent_by_ilm (Default=15)], whichever is greater.

The data for down-migration is chosen using last access time. In the case where the SSD tier utilization is 95%, 20% of the data in the SSD tier will be moved to the HDD tier (95% –> 75%). However, if the utilization was 80% only 15% of the data would be moved to the HDD tier using the minimum tier free up amount.

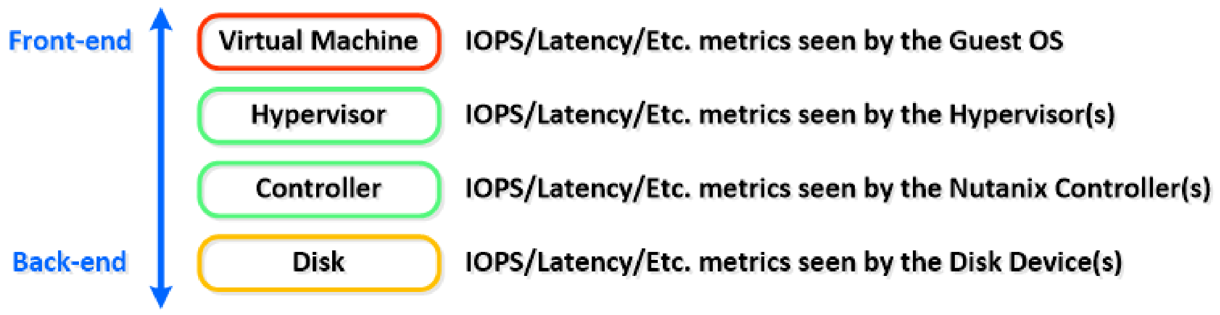

If the cluster-wide SSD tier is > a specific threshold, NDFS ILM will move data to the HDD tier to free up SSD space
VM 1
VM N
VM 1
Hypervisor
Cache
SSD
HDD
CVM
NDFS ILM
NDFS
NDFS ILM will migrate cold data from the local SSD to HDD to free SSD space

NDFS ILM will constantly monitor the I/O patterns and (down/up)-migrate data as necessary as well as bring the hottest data local regardless of tier.

Storage Layers and Monitoring

The Nutanix platform monitors storage at multiple layers throughout the stack ranging from the VM/Guest OS all the way down to the physical disk devices. Knowing the various tiers and how these relate is important whenever monitoring the solution and allows you to get full visibility of how the ops relate.

Below we show the various layers of where operations are monitored and the relative granularity which are explained below:


Front-end
Back-end
Virtual Machine
IOPS/Latency/Etc. metrics seen by the Guest OS
Hypervisor
IOPS/Latency/Etc. metrics seen by the Hypervisor(s)
Controller
IOPS/Latency/Etc. metrics seen by the Nutanix Controller(s)
Disk
IOPS/Latency/Etc. metrics seen by the Disk Device(s)

Virtual Machine Layer

- **Key Role:** Metrics reported by the Guest OS
- **Description:** Virtual Machine or Guest OS level metrics are pulled directly from the hypervisor and represent the performance the Guest OS is seeing and is indicative of the I/O performance the application is seeing.
- **When to use:** When troubleshooting or looking for OS or application level detail

Hypervisor Layer

- **Key Role:** Metrics reported by the Hypervisor(s)
- **Description:** Hypervisor level metrics are pulled directly from the hypervisor and represent the most accurate metrics the hypervisor(s) are seeing. This data can be viewed for one of more hypervisor node(s) or the aggregate cluster. This layer will provide the most accurate data in terms of what performance the platform is seeing and should be leveraged in most cases. In certain scenarios the hypervisor may combine or split operations coming

from VMs which can show the difference in metrics reported by the VM and hypervisor. These numbers will also include cache hits served by the Nutanix CVMs.

- **When to use:** Most common cases as this will provide the most detailed and valuable metrics

Controller Layer

- **Key Role:** Metrics reported by the Nutanix Controller(s)
- **Description:** Controller level metrics are pulled directly from the Nutanix Controller VMs (eg. Stargate 2009 page) and represent what the Nutanix front-end is seeing from NFS/SMB/iSCSI or any back-end operations (eg. ILM, disk balancing, etc.). This data can be viewed for one of more Controller VM(s) or the aggregate cluster. The metrics seen by the Controller Layer should match those seen by the hypervisor layer, however will include any backend operations (eg. ILM, disk balancing). These numbers will also include cache hits served by memory.
- **When to use:** Similar to the hypervisor layer, can be used to show how much backend operation is taking place

Disk Layer

- **Key Role:** Metrics reported by the Disk Device(s)
- **Description:** Disk level metrics are pulled directly from the physical disk devices (via the CVM) and represent what the back-end is seeing. This includes data hitting the OpLog or Extent Store where an I/O is performed on the disk. This data can be viewed for one of more disk(s), the disk(s) for a particular node or the aggregate disks in the cluster. In common cases it is expected that the disk ops should match the number of incoming writes as well as reads not served from the memory portion of the cache. Any reads being served by the memory portion of the cache will not be counted here as the op is not hitting the disk device.
- **When to use:** When looking to see how many ops are served from cache or hitting the disks

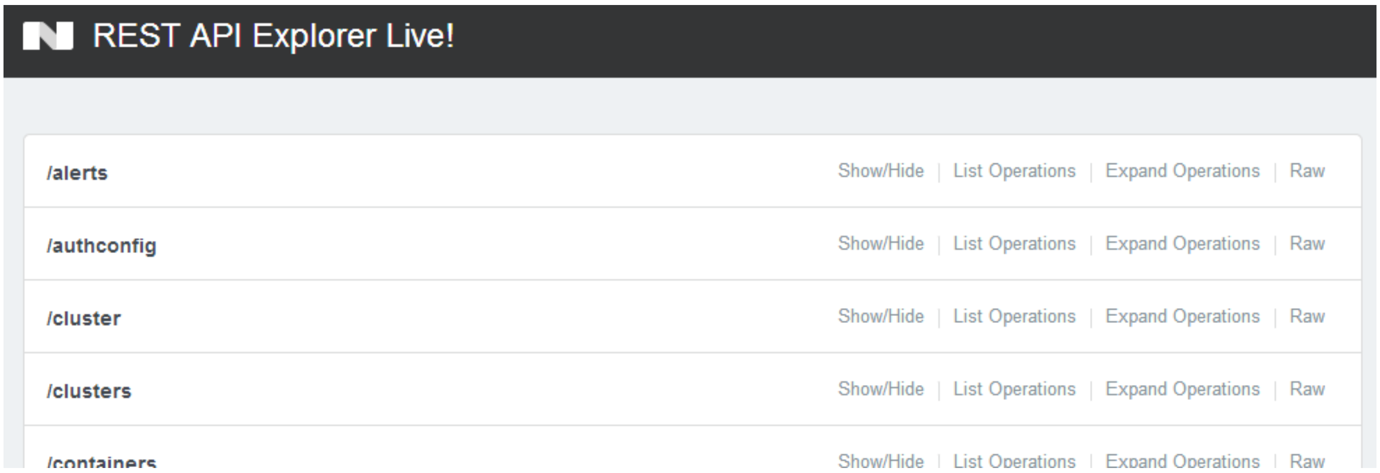
APIs and Interfaces

Core to any dynamic or "Software Defined" environment, Nutanix provides a vast array of interfaces allowing for simple programmability and interfacing. Here are the main interfaces:

- REST API
- NCLI
- Scripting interfaces – more coming here soon

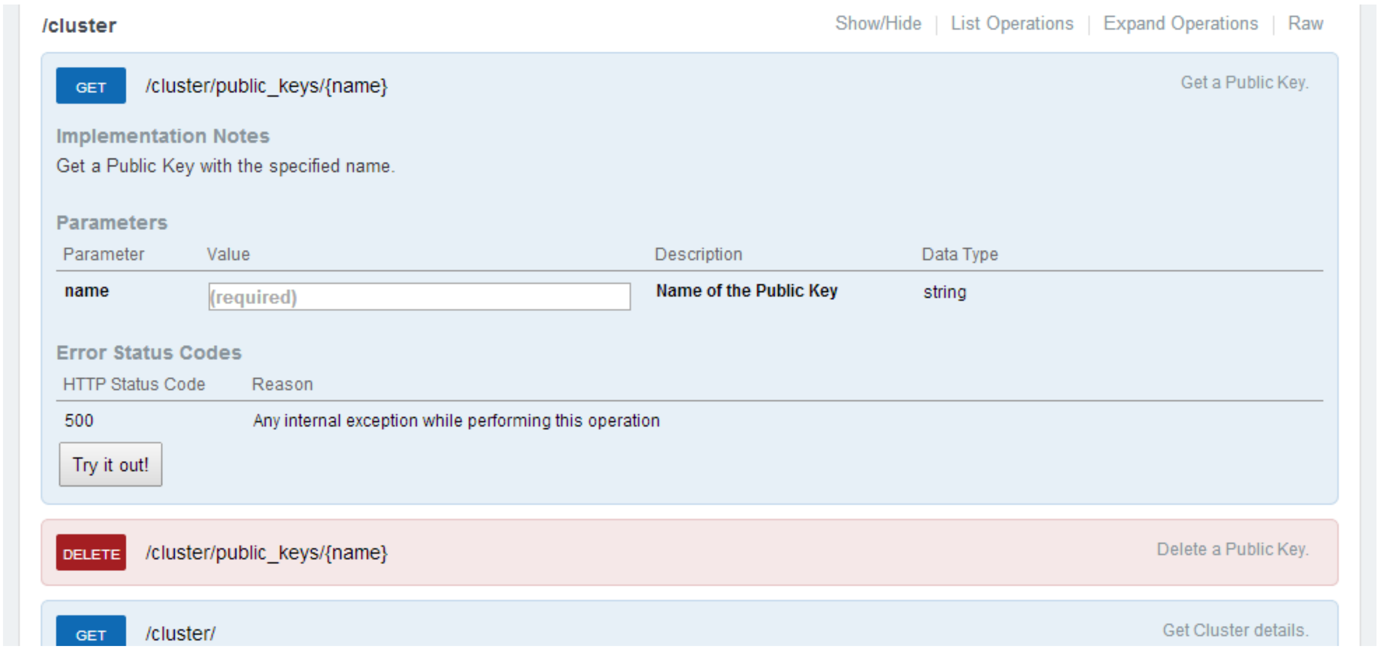
Core to this is the REST API which exposes every capability and data point of the Prism UI and allows for orchestration or automation tools to easily drive Nutanix action. This enables tools like VMware's vCAC or Microsoft's System Center Orchestrator to easily create custom workflows for Nutanix. Also, this means that any 3rd party developer could create their own custom UI and pull in Nutanix data via REST.

Below we show a small snippet of the Nutanix REST API explorer which allows developers to see the API and format:


REST API Explorer Live!

/alerts	Show/Hide | List Operations | Expand Operations | Raw
/authconfig	Show/Hide | List Operations | Expand Operations | Raw
/cluster	Show/Hide | List Operations | Expand Operations | Raw
/clusters	Show/Hide | List Operations | Expand Operations | Raw
/containers	Show/Hide | List Operations | Expand Operations | Raw

Operations can be expanded to display details and examples of the REST call:


/cluster — Show/Hide | List Operations | Expand Operations | Raw

GET /cluster/public_keys/{name} — Get a Public Key.

Implementation Notes

Get a Public Key with the specified name.

Parameters

Parameter	Value	Description	Data Type
name	(required)	Name of the Public Key	string

Error Status Codes

HTTP Status Code	Reason
500	Any internal exception while performing this operation

Try it out!

DELETE /cluster/public_keys/{name} — Delete a Public Key.

GET /cluster/ — Get Cluster details.

Availability Domains

Availability Domains aka node/block/rack awareness is a key struct for distributed systems to abide by for determining component and data placement. NDFS is currently node and block aware, however this will increase to rack aware as cluster sizes grow. Nutanix refers to a “block” as the chassis which contains either one, two or four server “nodes”.

NOTE: at minimum of 3 blocks must be utilized for block awareness to be activated, otherwise node awareness will be defaulted to. It is recommended to utilized uniformly populated blocks to ensure block awareness is enabled. Common scenarios and the awareness level utilized can be found at the bottom of this section. The 3 block requirement is due to ensure quorum.

For example a 3450 would be a block which holds 4 nodes. The reason for distributing roles or data across blocks to ensure if a block fails or needs maintenance the system can continue to run without interruption. NOTE: Within a block the redundant PSU and fans are the only shared components

Awareness can be broken into a few key focus areas:

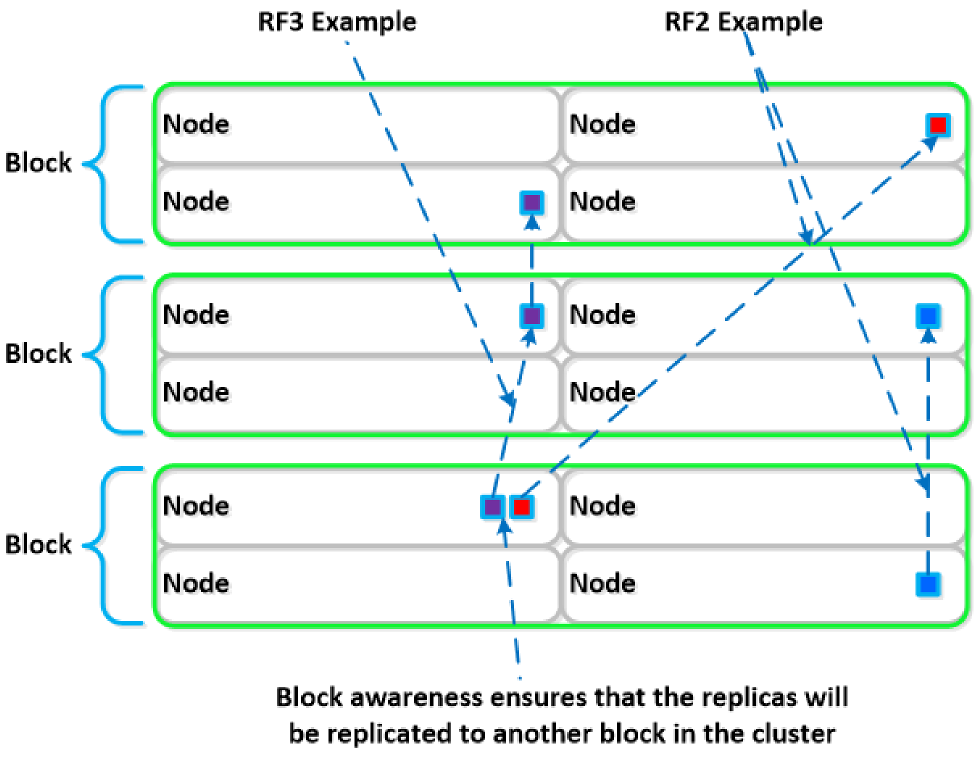
- Data (The VM data)
- Metadata (Cassandra)
- Configuration Data (Zookeeper)

Data

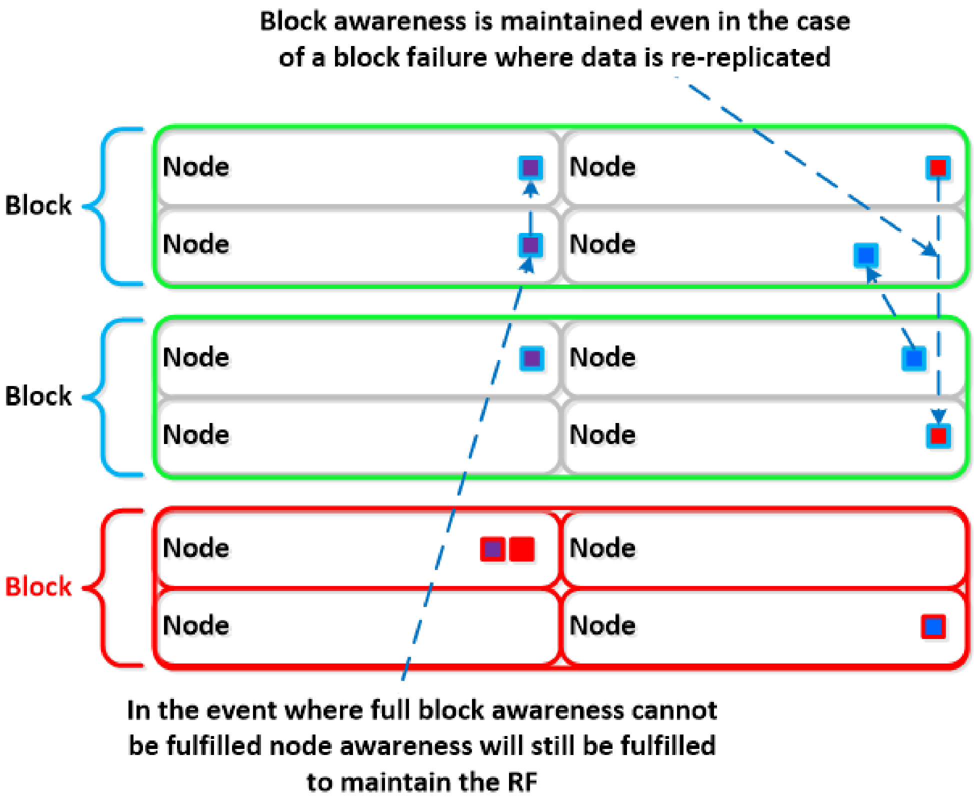
With NDFS data replicas will be written to other blocks in the cluster to ensure that in the case of a block failure or planned downtime, the data remains available. This is true for both RF2 and RF3 scenarios as well as in the case of a block failure.

An easy comparison would be "node awareness" where a replica would need to be replicated to another node which will provide protection in the case of a node failure. Block awareness further enhances this by providing data availability assurances in the case of block outages.

Below we show how the replica placement would work in a 3 block deployment:


RF3 Example
RF2 Example
Block
Node
Node
Node
Node
Block
Node
Node
Node
Node
Block
Node
Node
Node
Node
Block awareness ensures that the replicas will be replicated to another block in the cluster

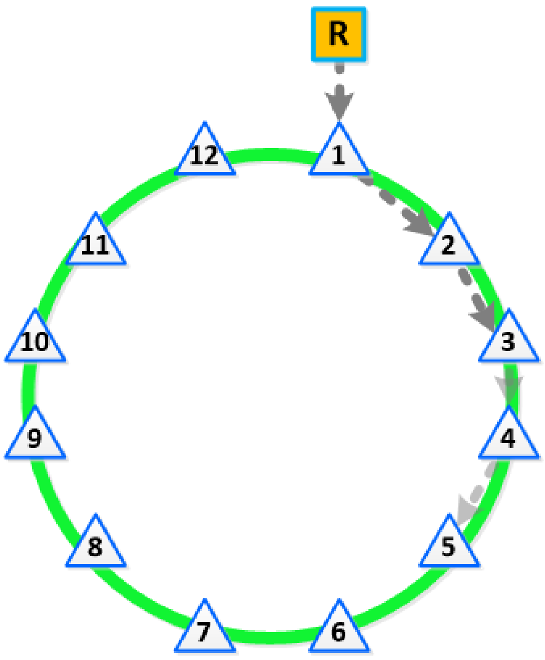
In the case of a block failure, block awareness will be maintained and the re-replicated blocks will be replicated to other blocks within the cluster:


Block awareness is maintained even in the case of a block failure where data is re-replicated
Block
Node
Node
Node
Node
Block
Node
Node
Node
Node
Block
Node
Node
Node
Node
In the event where full block awareness cannot be fulfilled node awareness will still be fulfilled to maintain the RF

Metadata

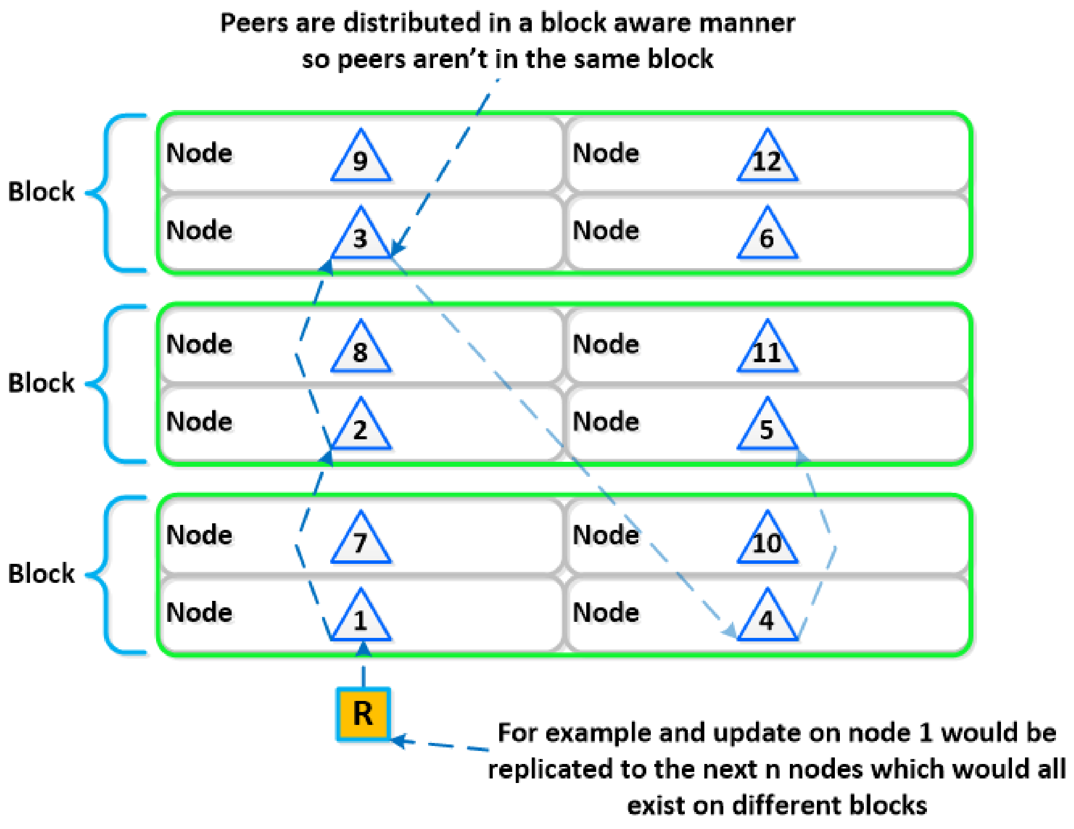
As mentioned in the Scalable Metadata section above, Nutanix leverages a heavily modified Cassandra platform to store metadata and other essential information. Cassandra leverages a ring-like structure and replicates to n number of peers within the ring to ensure data consistency and availability.

Below we show an example of the Cassandra ring for a 12 node cluster:



Cassandra peer replication iterates through nodes in a clockwise manner throughout the ring. With block awareness the peers are distributed among the blocks to ensure no two peers are on the same block.

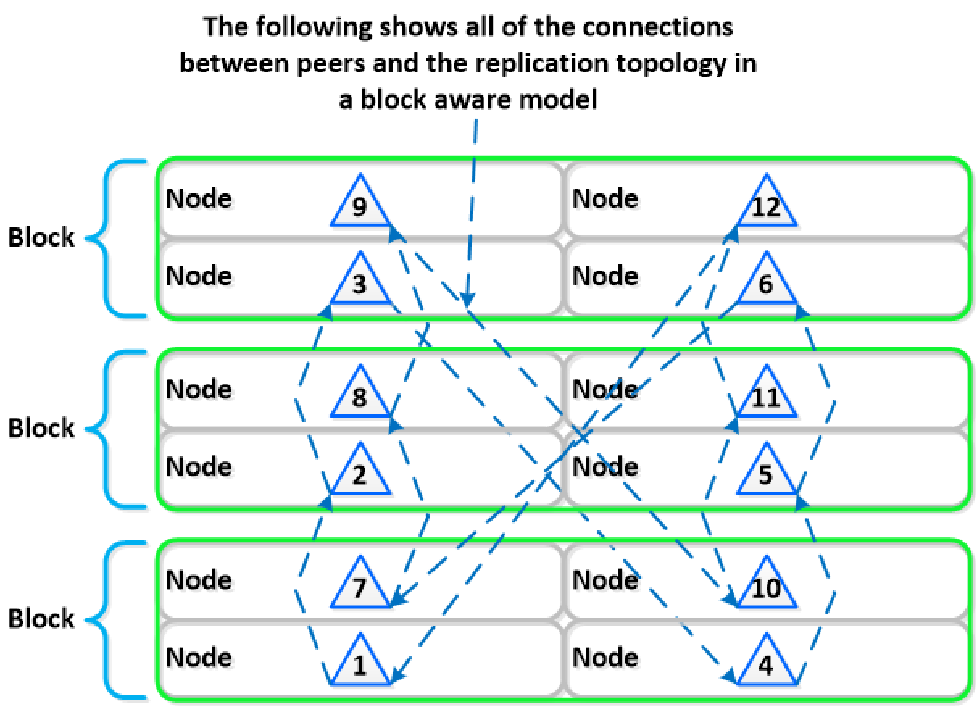
Below we show an example node layout translating the ring above into the block based layout:


Peers are distributed in a block aware manner so peers aren't in the same block
Block
Node 9
Node 12
Node 3
Node 6
Block
Node 8
Node 11
Node 2
Node 5
Block
Node 7
Node 10
Node 1
Node 4
R
For example and update on node 1 would be replicated to the next n nodes which would all exist on different blocks

With this block aware nature, in the event of a block failure there will still be at least two copies of the data (with

Metadata RF3 – In larger clusters RF5 can be leveraged).

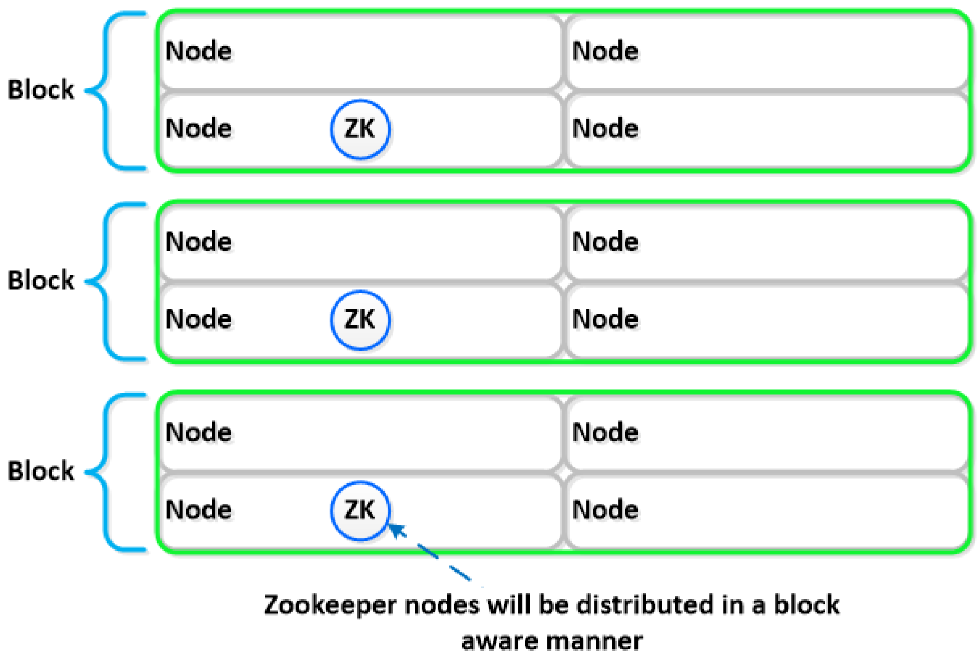
Below we show an example of all of the nodes replication topology to form the ring (yes – its a little busy):


The following shows all of the connections between peers and the replication topology in a block aware model
Block
Block
Block
Node
9
12
3
6
8
11
2
5
7
10
1
4

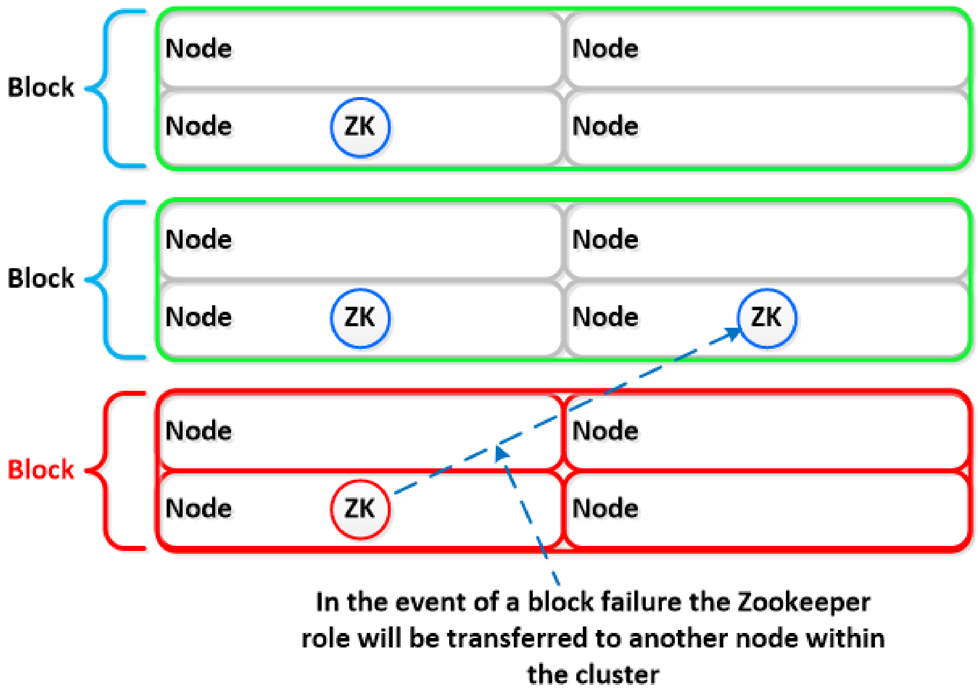
Configuration Data

Nutanix leverages Zookeeper to store essential configuration data for the cluster. This role is also distributed in a block aware manner to ensure availability in the case of a block failure.

Below we show an example layout showing 3 Zookeeper nodes distributed in a block aware manner:



In the event of a block outage, meaning on of the Zookeeper nodes will be gone, the Zookeeper role would be transferred to another node in the cluster as shown below:


Block
Node
Node
Node
ZK
Node
Block
Node
Node
Node
ZK
Node
ZK
Block
Node
Node
Node
ZK
Node
In the event of a block failure the Zookeeper role will be transferred to another node within the cluster

Below we breakdown some common scenarios and what level of awareness will be utilized:

- < 3 blocks –> **NODE** awareness
- 3+ blocks uniformly populated –> **BLOCK** + **NODE** awareness
- 3+ blocks not uniformly populated
  - If SSD tier variance between blocks is > max variance –> **NODE** awareness
    - Example: 2 x 3450 + 1 x 3150
  - If SSD tier variance between blocks is < max variance –> **BLOCK** + **NODE** awareness
    - Example: 2 x 3450 + 1 x 3350
  - NOTE: max tier variance is calculated as: 100 / (RF+1)
    - Eg. 33% for RF2 or 25% for RF3

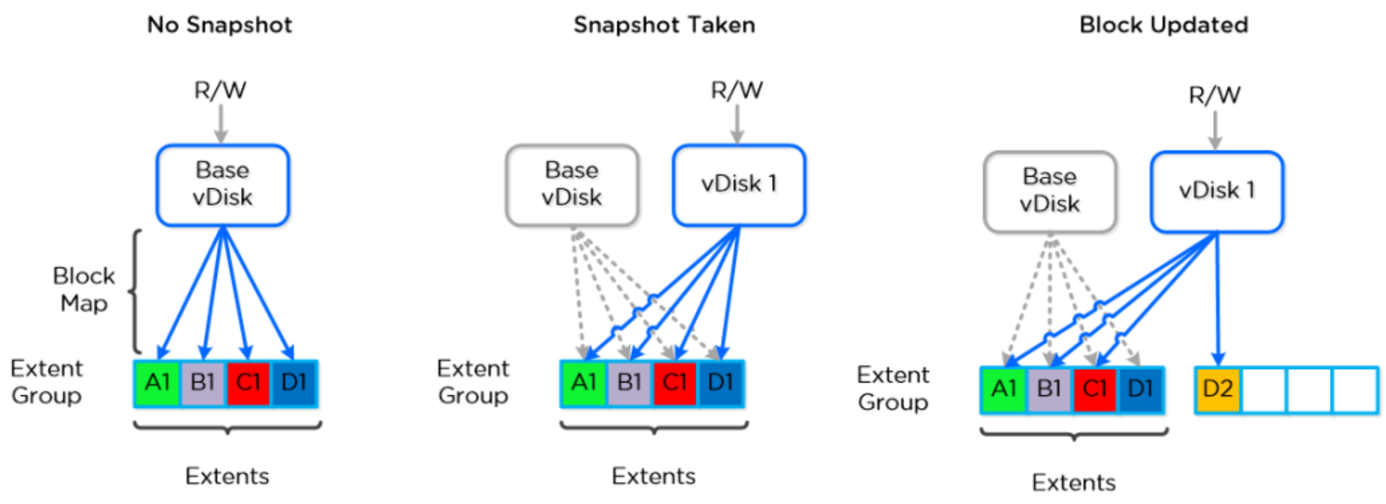
Snapshots & Clones

NDFS provides native support for offloaded snapshots and clones which can be leveraged via VAAI, ODX, ncli, REST, Prism, etc. Both the snapshots and clones leverage the redirect-on-write algorithm which is the most effective and efficient.

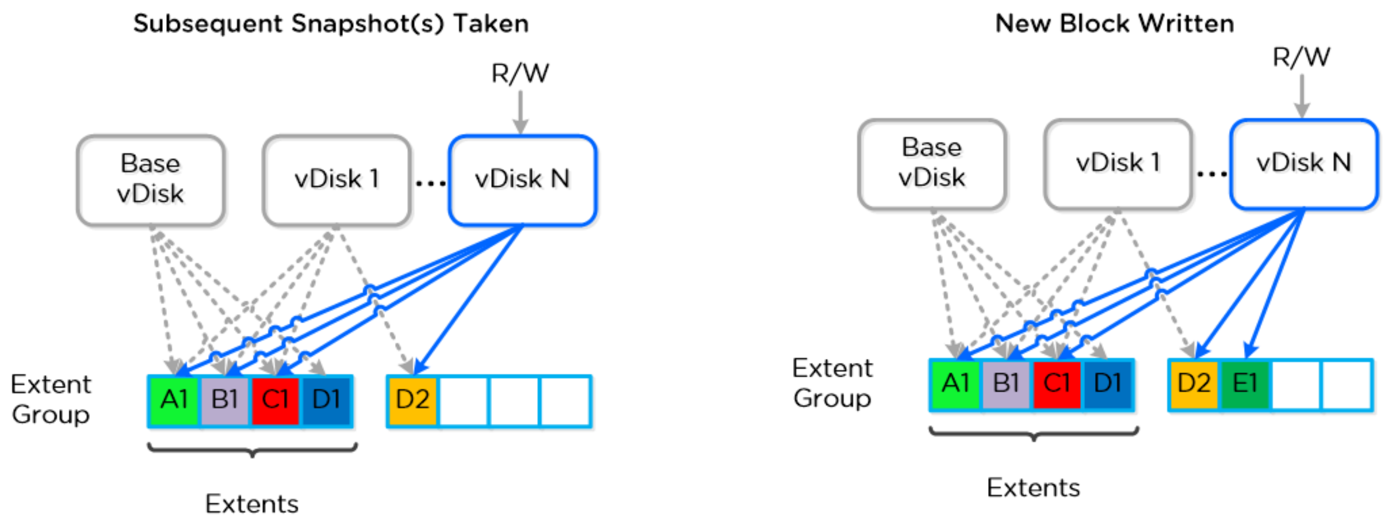
As explained in the Data Structure section above, a virtual machine consists of files (vmdk/vhdk) which are vDisks on the Nutanix platform. A vDisk is composed of extents which are logically contiguous chunks of data, which are stored within extent groups which are physically contiguous data stored as files on the storage devices.

When a snapshot or clone is taken the base vDisk is marked immutable and another vDisk is created as read/write. At this point both vDisks have the same block map, which is a metadata mapping of the vDisk to its corresponding extents. Contrary to traditional approaches which require traversal of the snapshot chain, which can add read latency, each vDisk has its own block map. This eliminates any of the overhead normally seen by large snapshot chain depths and allows you to take continuous snapshots without any performance impact.

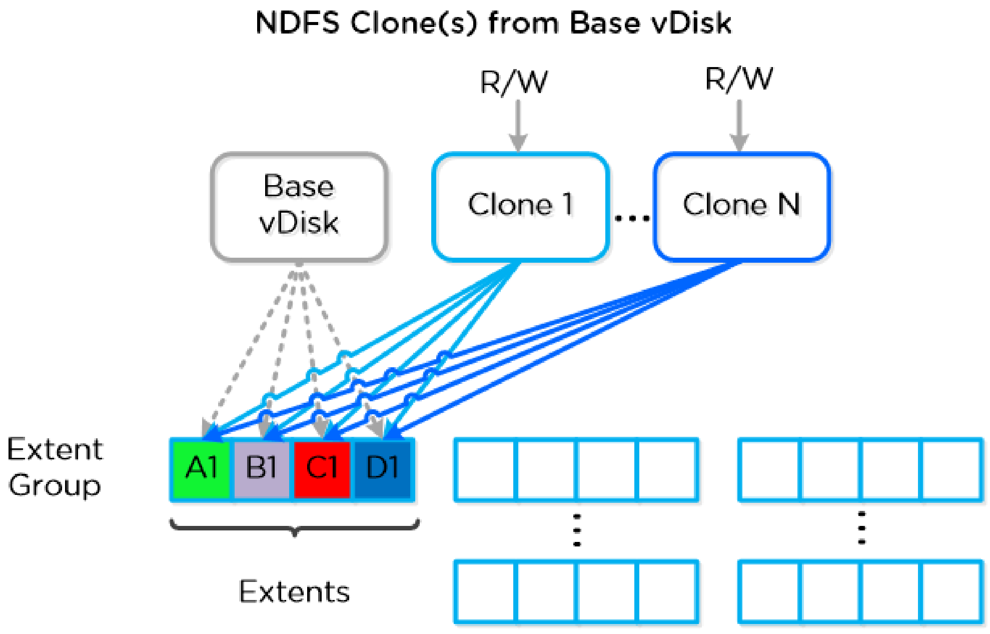
Below we show an example of how this works when a snapshot is taken (NOTE: I need to give some credit to NTAP as a base for these diagrams as I thought their representation was the clearest):



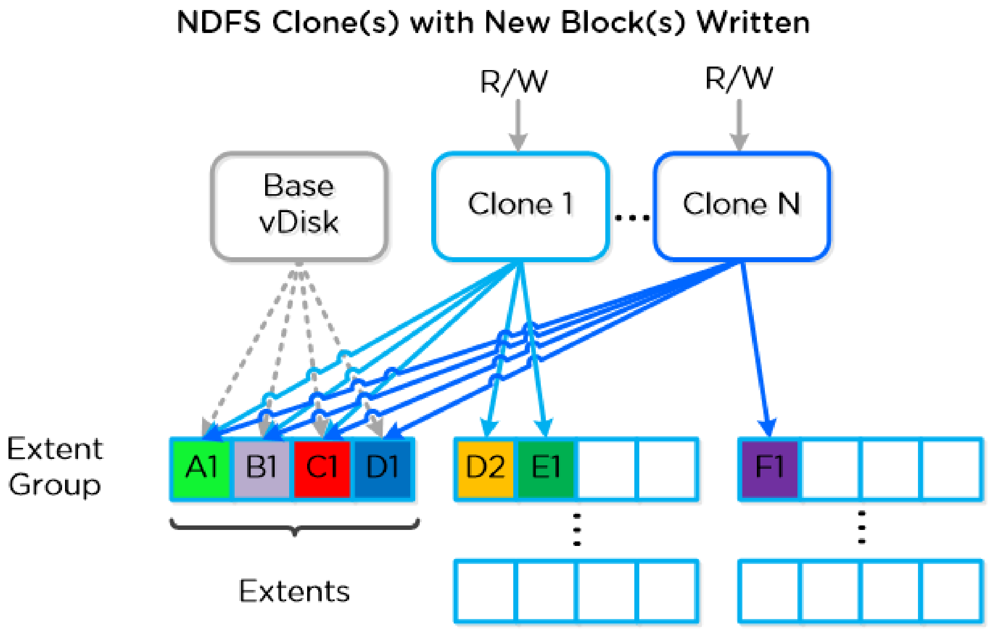
The same method applies when a snapshot or clone of a previously snapped or cloned vDisk is performed:



The same methods are used for both snapshots and/or clones of a VM or vDisk(s). When a VM or vDisk is cloned the current block map is locked and the clones are created. These updates are metadata only so no I/O actually takes place. The same method applies for clones of clones; essentially the previously cloned VM acts as the "Base vDisk" and upon cloning that block map is locked and two "clones" are created: one for the VM being cloned and another for the new clone. They both inherit the prior block map and any new writes/updates would take place on their individual block maps.



As mentioned prior each VM/vDisk has its own individual block map. So in the above example, all of the clones from the base VM would now own their block map and any write/update would occur there. Below we show an example of what this looks like:



Any subsequent clones or snapshots of a VM/vDisk would cause the original block map to be locked and would create a new one for R/W access.

Multi-Site Disaster Recovery

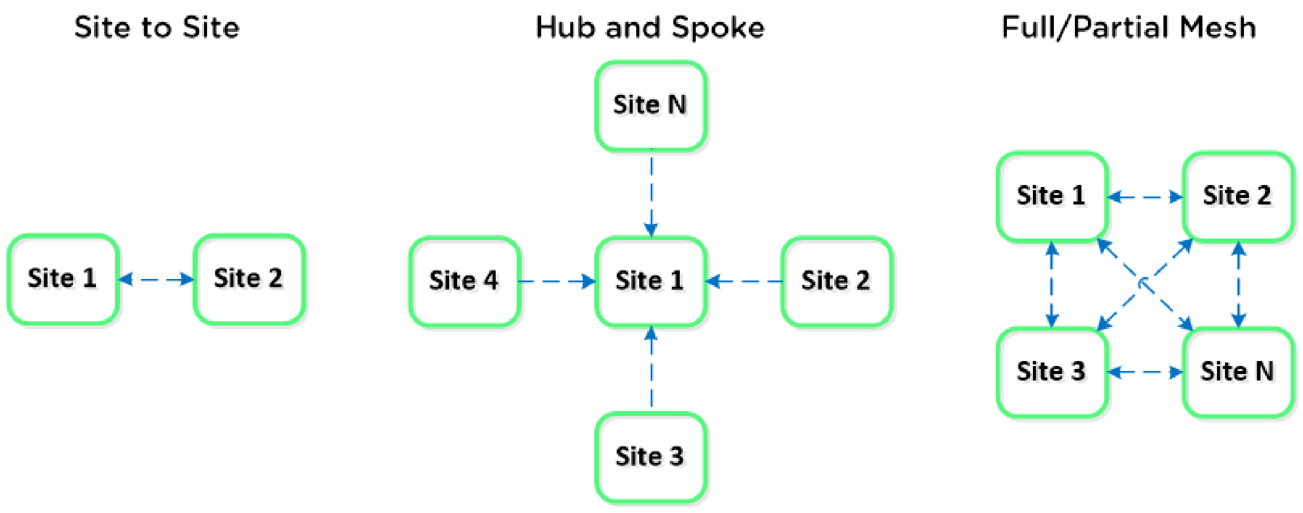
Nutanix provides native DR and replication capabilities which build upon the same features explained in the Snapshots & Clones section. Cerebro is the component responsible for managing the DR and replication in NDFS. Cerebro runs on every node and a Cerebro master is elected (similar to NFS master) and is responsible for managing replication tasks. In the event the CVM acting as Cerebro master fails, another is elected and assumes the role. The Cerebro page can be found on <CVM IP>:2020.

The DR function can be broken down into a few key focus areas:

- Replication Topology
- Implementation Constructs
- Global Deduplication

Replication Topologies

Traditionally there are a few key replication topologies: Site to site, Hub and spoke and Full and/or Partial mesh. Contrary to traditional solutions which only allow for Site to site or hub and spoke, Nutanix provides a fully mesh or flexible many-to-many model.



Essentially this allows the admin to determine a replication capability that meets their needs.

Implementation Constructs

Within Nutanix DR there are a few key constructs which are explained below:

Remote Site

- **Key Role:** A remote Nutanix cluster
- **Description:** A remote Nutanix cluster which can be leveraged as a target for backup or DR purposes.
- **Pro tip:** Ensure the target site has ample capacity (compute/storage) to handle a full site failure. In certain cases replication/DR between racks within a single site can also make sense.

Protection Domain (PD)

- **Key Role:** Macro group of VMs and/or files to protect
- **Description:** A group of VMs and/or files to be replicated together on a desired schedule. A PD can protect a full container or you can select individual VMs and/or files
- **Pro tip:** Create multiple PDs for various services tiers driven by a desired RPO/RTO. For file distribution (eg. golden images, ISOs, etc.) you can create a PD with the files to replication.

Consistency Group (CG)

- **Key Role:** Subset- of VMs/files in PD to be crash consistent
- **Description:** VMs and/or files which are part of a Protection Domain which need to be snapshotted in a crash consistent manner. This ensures that when VMs/files are recovered they come up in a consistent state. A protection domain can have multiple consistency groups.

- **Pro tip:** Group dependent application or service VMs in a consistency group to ensure they are recovered in a consistent state (eg. App and DB)

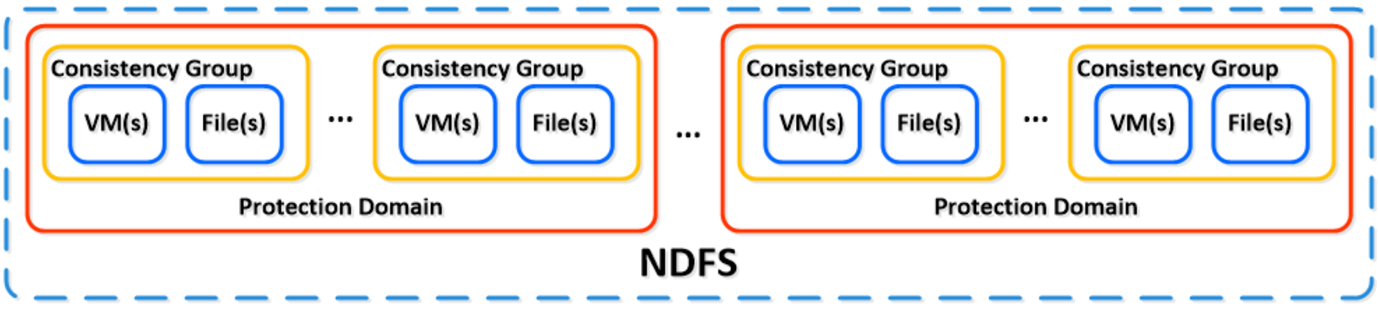
Replication Schedule

- **Key Role:** Snapshot and replication schedule
- **Description:** Snapshot and replication schedule for VMs in a particular PD and CG
- **Pro tip:** The snapshot schedule should be equal to your desired RPO

Retention Policy

- **Key Role:** Number of local and remote snapshots to keep
- **Description:** The retention policy defines the number of local and remote snapshots to retain. NOTE: A remote site must be configured for a remote retention/replication policy to be configured.
- **Pro tip:** The retention policy should equal the number of restore points required per VM/file

Below we show a logical representation of the relationship between a PD, CG and VM/Files for a single site:



It’s important to mention that a full container can be protected for simplicity, however the platform provides the ability to protect down to the granularity of a single VM and/or file level.

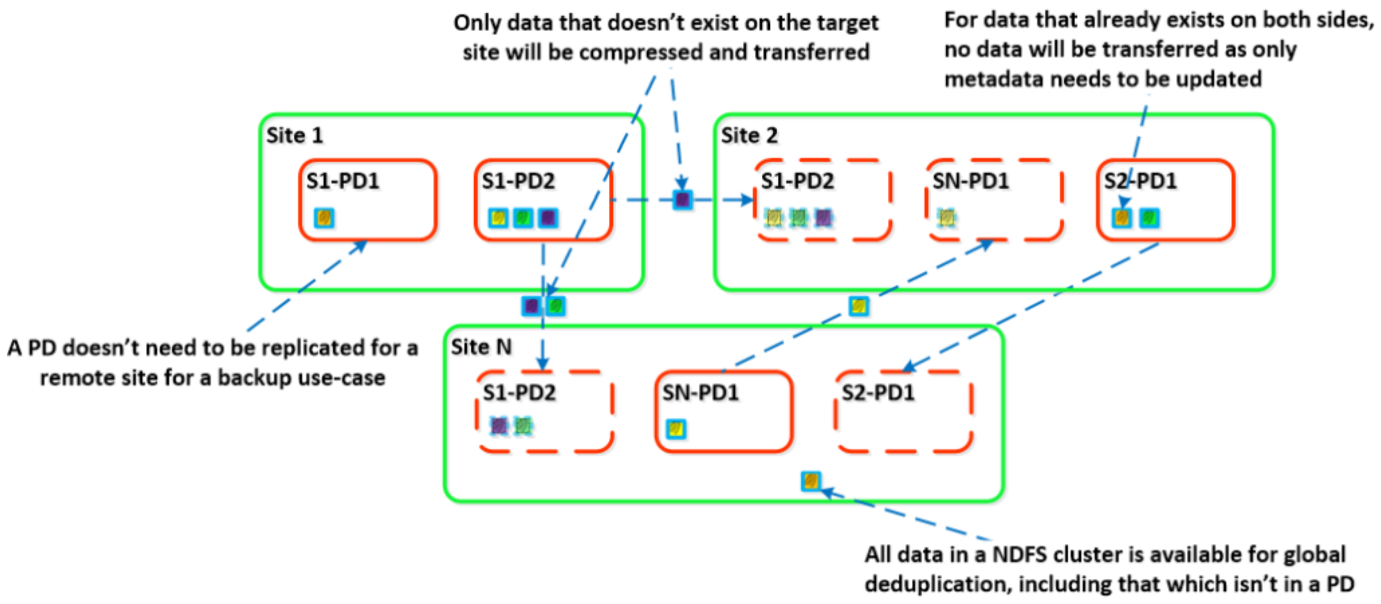
Global Deduplication

As explained in the Elastic Dedup Engine section above, NDFS has the ability to deduplicate data by just updating metadata pointers.

The same concept is applied to the DR and replication feature. Before sending data over the wire, NDFS will query remote site and check whether or not the fingerprint(s) already exist on the target (meaning the data already exists). If so, no data will be shipped over the wire and only a metadata update will occur.

For data which doesn’t exist on the target the data will be compressed and sent to the target site. At this point the data exists on both sites is usable for deduplication.

Below we show an example three site deployment where each site contains one of more protection domains (PD):


Only data that doesn't exist on the target site will be compressed and transferred
For data that already exists on both sides, no data will be transferred as only metadata needs to be updated
Site 1
S1-PD1
S1-PD2
Site 2
S1-PD2
SN-PD1
S2-PD1
A PD doesn't need to be replicated for a remote site for a backup use-case
Site N
S1-PD2
SN-PD1
S2-PD1
All data in a NDFS cluster is available for global deduplication, including that which isn't in a PD

I'll cover how replication works in a section to come!

top

Administration

Important Pages

These are advanced Nutanix pages besides the standard user interface that allow you to monitor detailed stats and metrics. The URLs are formatted in the following way: http://<Nutanix CVM IP/DNS>:<Port/path (mentioned below)> Example: http://MyCVM-A:2009 NOTE: if you're on a different subnet IPtables will need to be disabled on the CVM to access the pages.

# 2009 Page

- This is a Stargate page used to monitor the back end storage system and should only be used by advanced users. I'll have a post that explains the 2009 pages and things to look for.

# 2009/latency Page

- This is a Stargate page used to monitor the back end latency

# 2009/h/traces Page

- This is the Stargate page used to monitor activity traces for operations

# 2010 Page

- This is the Curator page which is used for monitoring curator runs

# 2010/master/control Page

- This is the Curator control page which is used to manually start Curator jobs

# 2011 Page

- This is the Chronos page which monitors jobs and tasks scheduled by curator

# 2020 Page

- This is the Cerebro page which monitors the protection domains, replication status and DR

# 7777 Page

- This is the Aegis Portal page which can be used to get good logs and statistics, useful commands, and modify Gflags

top

Cluster Commands

# Check cluster status

```
# Description: Check cluster status from the CLI

cluster status
```

# Check local CVM service status

```
# Description: Check a single CVM's service status from the CLI

genesis status
```

# Nutanix cluster upgrade

```
# Description: Perform rolling (aka "live") cluster upgrade from the CLI

# Upload upgrade package to ~/tmp/ on one CVM

# Untar package
tar xzvf ~/tmp/nutanix*

# Perform upgrade
~/tmp/install/bin/cluster -i ~/tmp/install upgrade

# Check status
upgrade_status
```

# Restart cluster service from CLI

```
# Description: Restart a single cluster service from the CLI

# Stop service
cluster stop <Service Name>

# Start stopped services
cluster start  #NOTE: This will start all stopped services
```

# Start cluster service from CLI

```
# Description: Start stopped cluster services from the CLI

# Start stopped services
cluster start  #NOTE: This will start all stopped services

# OR

# Start single service
Start single service: cluster start  <Service Name>
```

# Restart local service from CLI

```
# Description: Restart a single cluster service from the CLI

# Stop Service
genesis stop <Service Name>

# Start Service
cluster start
```

# Start local service from CLI

```
# Description: Start stopped cluster services from the CLI

cluster start #NOTE: This will start all stopped services
```

# Cluster add node from cmdline

```
# Description: Perform cluster add-node from CLI

ncli cluster discover-nodes | egrep "Uuid" | awk '{print $4}' | xargs -I UUID ncli cluster add-node node-uuid=UUID
```

# Find number of vDisks

```
# Description: Displays the number of vDisks

vdisk_config_printer | grep vdisk_id | wc -l
```

# Find cluster id

```
# Description: Find the cluster ID for the current cluster

zeus_config_printer | grep cluster_id
```

# Disable IPtables

```
# Description: Disables IPtables service on all cluster CVMs

for i in `svmips`;do ssh $i "sudo /etc/init.d/iptables save && sudo /etc/init.d/iptables stop && sudo chkconfig iptabl
```

# Check for Shadow Clones

```
# Description: Displays the shadow clones in the following format:  name#id@svm_id

vdisk_config_printer | grep '#'
```

# Reset Latency Page Stats

```
# Description: Reset the Latency Page (<CVM IP>:2009/latency) counters

for i in `svmips`;do wget $i:2009/latency/reset;done
```

# Find Number of vDisks

```
# Description: Find the current number of vDisks (files) on NDFS

vdisk_config_printer | grep vdisk_id | wc -l
```

# Start Curator scan from CLI

```
# Description: Starts a Curator full scan from the CLI

```

```
for i in `svmips`;do wget -O - "http://$i:2010/master/api/client/StartCuratorTasks?task_type=2"; done
```

# Compact ring

```
# Description: Compact the metadata ring

for i in `svmips`; do echo $i;ssh $i `nodetool -h localhost compact`;done
```

# Find NOS version

```
# Description: Find the NOS  version (NOTE: can also be done using NCLI)

for i in `svmips`;do echo $i;ssh $i 'cat /etc/nutanix/release_version';done
```

# Find CVM version

```
# Description: Find the CVM image version

for i in `svmips`; do echo $i;ssh $i `cat /etc/nutanix/svm-version`;done
```

# Manually fingerprint vDisk(s)

```
# Description: Create fingerprints for a particular vDisk (For dedupe)  NOTE: dedupe must be enabled on the container

vdisk_manipulator –vdisk_id=<vDisk ID> --operation=add_fingerprints
```

# Echo Factory_Config.json for all cluster nodes

```
# Description: Echos the factory_config.jscon for all nodes in the cluster

for i in `svmips`; do echo $i; ssh $i "cat /etc/nutanix/factory_config.json"; done
```

# Upgrade a single Nutanix node's NOS version

```
# Description: Upgrade a single node's NOS version to match that of the cluster

~/cluster/bin/cluster -u <NEW_NODE_IP> upgrade_node
```

# Install Nutanix Cluster Check (NCC)

```
# Description: Installs the Nutanix Cluster Check (NCC) health script to test for potential issues and cluster health

# Download NCC from the Nutanix Support Portal (portal.nutanix.com)

# SCP .tar.gz to the /home/nutanix directory

# Untar NCC .tar.gz
tar xzmf <ncc .tar.gz file name> --recursive-unlink

# Run install script
./ncc/bin/install.sh -f <ncc .tar.gz file name>

# Create links
source ~/ncc/ncc_completion.bash
echo "source ~/ncc/ncc_completion.bash" >> ~/.bashrc
```

# Run Nutanix Cluster Check (NCC)

```
# Description: Runs the Nutanix Cluster Check (NCC) health script to test for potential issues and cluster health.  Th

# Make sure NCC is installed (steps above)

# Run NCC health checks
```

```
ncc health_checks run_all
```

top

NCLI

NOTE: All of these actions can be performed via the HTML5 GUI. I just use these commands as part of my bash scripting to automate tasks.

# Add subnet to NFS whitelist

```
# Description: Adds a particular subnet to the NFS whitelist

ncli cluster add-to-nfs-whitelist ip-subnet-masks=10.2.0.0/255.255.0.0
```

# Display Nutanix Version

```
# Description: Displays the current version of the Nutanix software

ncli cluster version
```

# Display hidden NCLI options

```
# Description: Displays the hidden ncli commands/options

ncli helpsys listall hidden=true [detailed=false|true]
```

# List Storage Pools

```
# Description: Displays the existing storage pools

ncli sp ls
```

# List containers

```
# Description: Displays the existing containers

ncli ctr ls
```

# Create container

```
# Description: Creates a new container

ncli ctr create name=<NAME> sp-name=<SP NAME>
```

# List VMs

```
# Description: Displays the existing VMs

ncli vm ls
```

# List public keys

```
# Description: Displays the existing public keys

ncli cluster list-public-keys
```

# Add public key

```
# Description: Adds a public key for cluster access

# SCP private key to CVM

# Add private key to cluster
ncli cluster add-public-key name=myPK file-path=~/mykey.pub
```

# Remove public key

```
# Description: Removes a public key for cluster access

ncli cluster remove-public-keys name=myPK
```

# Create protection domain

```
# Description: Creates a protection domain

ncli pd create name=<NAME>
```

# Create remote site

```
# Description: Create a remote site for replication

ncli remote-site create name=<NAME> address-list=<Remote Cluster IP>
```

# Create protection domain for all VMs in container

```
# Description: Protect all VMs in the specified container

ncli pd protect name=<PD NAME> ctr-id=<Container ID> cg-name=<NAME>
```

# Create protection domain with specified VMs

```
# Description: Protect the VMs specified

ncli pd protect name=<PD NAME> vm-names=<VM Name(s)> cg-name=<NAME>
```

# Create protection domain for NDFS files (aka vDisk)

```
# Description: Protect the NDFS Files specified

ncli pd protect name=<PD NAME> files=<File Name(s)> cg-name=<NAME>
```

# Create snapshot of protection domain

```
# Description: Create a one-time snapshot of the protection domain

ncli pd add-one-time-snapshot name=<PD NAME> retention-time=<seconds>
```

# Create snapshot and replication schedule to remote site

```
# Description: Create a recurring snapshot schedule and replication to n remote sites

ncli pd set-schedule name=<PD NAME> interval=<seconds> retention-policy=<POLICY> remote-sites=<REMOTE SITE NAME>
```

# List replication status

```
# Description: Monitor replication status

ncli pd list-replication-status
```

# Migrate protection domain to remote site

```
# Description: Fail-over a protection domain to a remote site

ncli pd migrate name=<PD NAME> remote-site=<REMOTE SITE NAME>
```

# Activate protection domain

```
# Description: Activate a protection domain at a remote site

ncli pd activate name=<PD NAME>
```

# Enable NDFS Shadow Clones

```
# Description: Enables the NDFS Shadow Clone feature

ncli cluster edit-params enable-shadow-clones=true
```

# Enable Dedup for vDisk

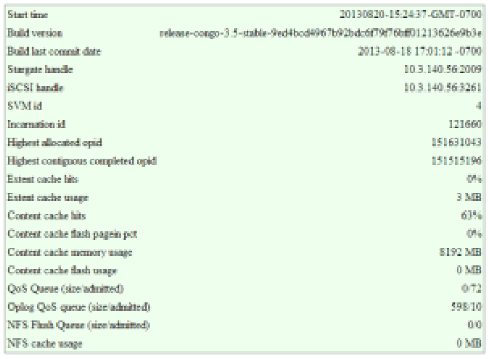
```
# Description: Enables fingerprinting and/or on disk dedup for a specific vDisk

ncli vdisk edit name=<VDISK NAME> fingerprint-on-write=<true/false> on-disk-dedup=<true/false>
```

top

Metrics & Thresholds

The below will cover specific metrics and thresholds on the Nutanix back end. More updates to these coming shortly!

2009 Stargate – Overview

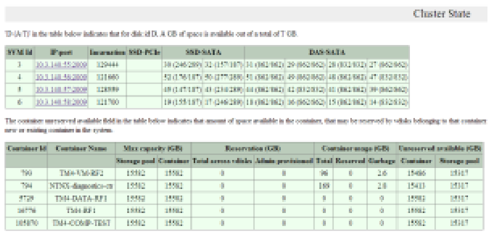


Start time	20130820-15:24:37-GMT-0700
Build version	release-congo-3.5-stable-9ed4bcd4967b92bdc6f79f76bff01213626e9b3e
Build last commit date	2013-08-18 17:01:12 -0700
Stargate handle	10.3.140.56:2009
iSCSI handle	10.3.140.56:3261
SVM id	4
Incarnation id	121660
Highest allocated opid	151631043
Highest contiguous completed opid	151515196
Extent cache hits	0%
Extent cache usage	3 MB
Content cache hits	63%
Content cache flash pagein pct	0%
Content cache memory usage	8192 MB
Content cache flash usage	0 MB
QoS Queue (size/admitted)	0/72
Oplog QoS queue (size/admitted)	598/10
NFS Flush Queue (size/admitted)	0/0
NFS cache usage	0 MB

Metric	Explanation	Threshold/Target
Start time	The start time of the Stargate service	
Build version	The build version currently running	
Build last commit date	The last commit date of the build	
Stargate handle	The Stargate handle	
iSCSI handle	The iSCSI handle	
SVM id	The SVM id of Stargate	
Incarnation id		
Highest allocated opid		
Highest contiguous completed opid		
Extent cache hits	The % of read requests served directly from the in-memory extent	

	cache
Extent cache usage	The MB size of the extent cache
Content cache hits	The % of read requests served directly from the content cache
Content cache flash pagein pct	
Content cache memory usage	The MB size of the in-memory content cache
Content cache flash usage	The MB size of the SSD content cache
QoS Queue (size/admitted)	The admission control queue size and number of admitted ops
Oplog QoS queue (size/admitted)	The oplog queue size and number of admitted ops
NFS Flush Queue (size/admitted)	
NFS cache usage	

2009 Stargate – Cluster State



Cluster State

'D (A/T)' in the table below indicates that for disk id D, A GB of space is available out of a total of T GB.

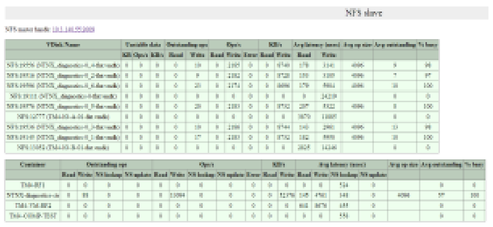
SVM Id	IP:port	Incarnation	SSD-PCIe	SSD-SATA		DAS-SATA			
3	10.3.140.55:2009	329444		30 (246/289)	32 (157/187)	31 (862/862)	29 (862/862)	28 (832/832)	27 (862/862)
4	10.3.140.56:2009	321660		52 (176/187)	50 (277/289)	51 (862/862)	49 (862/862)	48 (862/862)	47 (832/832)
5	10.3.140.57:2009	328599		45 (147/187)	43 (230/289)	44 (862/862)	42 (832/832)	41 (862/862)	39 (862/862)
6	10.3.140.58:2009	321700		19 (155/187)	17 (246/289)	18 (862/862)	16 (862/862)	15 (862/862)	14 (832/832)

The container unreserved available field in the table below indicates that amount of space available in the container, that may be reserved by vdisks belonging to that container new or existing containers in the system.

Container Id	Container Name	Max capacity (GB)		Reservation (GB)		Container usage (GB)			Unreserved available (GB)	
		Storage pool	Container	Total across vdisks	Admin provisioned	Total	Reserved	Garbage	Container	Storage pool
790	TM4-VM-RF2	15582	15582	0	0	96	0	2.6	15486	15317
794	NTNX-diagnostics-ctr	15582	15582	0	0	169	0	2.0	15413	15317
5729	TM4-DATA-RF1	15582	15582	0	0	0	0	0	15582	15317
16776	TM4-RF1	15582	15582	0	0	0	0	0	15582	15317
105870	TM4-COMP-TEST	15582	15582	0	0	0	0	0	15582	15317

Metric	**Explanation**	**Threshold/Target**
SVM Id	The Id of the Controller	
IP:port	The IP:port of the Stargate handle	
Incarnation		
SSD-PCIe	The SSD-PCIe devices and size/utilization	
SSD-SATA	The SSD-SATA devices and size/utilization	
DAS-SATA	The HDD-SATA devices and size/utilization	
Container Id	The Id of the container	
Container Name	The Name of the container	
Max capacity (GB) – Storage pool	The Max capacity of the storage pool	
Max capacity (GB) – Container	The Max capacity of the container (will normally match the storage pool size)	
Reservation (GB) – Total across vdisks	The reservation in GB across vdisks	
Reservation (GB) – Admin provisioned		
Container usage (GB) – Total	The total usage in GB per container	
Container usage (GB) – Reserved	The reservation used in GB per container	
Container usage (GB) – Garbage		
Unreserved available (GB) – Container	The available capacity in GB per container	
Unreserved available (GB) – Storage pool	The available capacity in GB for the storage pool	

2009 Stargate – NFS Slave



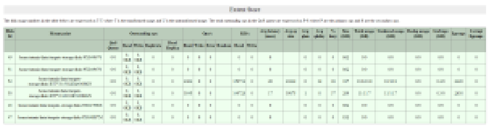
Metric	Explanation	Threshold/Target
Vdisk Name	The name of the Vdisk on NDFS	
Unstable data – KB		
Unstable data – Ops/s		
Unstable data – KB/s		
Outstanding Ops – Read	The number of outstanding read ops for the Vdisk	
Outstanding Ops – Write	The number of outstanding write ops for the Vdisk	
Ops/s – Read	The number of current read operations per second for the Vdisk	
Ops/s – Write	The number of current write operations per second for the Vdisk	
Ops/s – Error	The number of current error (failed) operations per second for the Vdisk	
KB/s – Read	The read throughput in KB/s for the Vdisk	
KB/s – Write	The write throughput in KB/s for the Vdisk	
Avg latency (usec) – Read	The average read op latecy in micro seconds for the Vdisk	
Avg latency (usec) – Write	The average write op latecy in micro seconds for the Vdisk	
Avg op size	The average op size in bytes for the Vdisk	
Avg outstanding	The average outstanding ops for the Vdisk	
% busy	The % busy of the Vdisk	
Container Name	The name of the container	
Outstanding Ops – Read	The number of outstanding read ops for the container	
Outstanding Ops – Write	The number of outstanding write ops for the container	
Outstanding Ops – NS lookup	The number of oustanding NFS lookup ops for the container	
Outstanding Ops – NS update	The number of outstanding NFS update ops for the container	
Ops/s – Read	The number of current read operations per second for the container	
Ops/s – Write	The number of current write operations per second for the container	
Ops/s – NS lookup	The number of current NFS lookup ops for the container	
Ops/s – NS update	The number of current NFS update ops for the container	
Ops/s – Error	The number of current error (failed) operations per second for the container	
KB/s – Read	The read throughput in KB/s for the container	
KB/s – Write	The write throughput in KB/s for the container	
Avg latency (usec) – Read	The average read op latecy in micro seconds for the container	
Avg latency (usec) – Write	The average write op latecy in micro seconds for the container	
Avg latency (usec) – NS lookup	The average NFS lookup latency in micro seconds for the container	
Avg latency (usec) – NS update	The average NFS lookup update in micro seconds for the container	
Avg op size	The average op size in bytes for the container	
Avg outstanding	The average outstanding ops for the container	

% busy	The % busy of the container

2009 Stargate – Hosted VDisks

Metric	Explanation	Threshold/Target
Vdisk Id	The Id of the Vdisk on NDFS	
Vdisk Name	The name of the Vdisk on NDFS	
Usage (GB)	The usage in GB per Vdisk	
Dedup (GB)		
Oplog – KB	The size of the Oplog for the Vdisk	
Oplog – Fragments	The number of fragments of the Oplog for the Vdisk	
Oplog – Ops/s	The number of current opeations per second for the Vdisk	
Oplog – KB/s	The throughput in KB/s for the Vdisk	
Outstanding Ops – Read	The number of outstanding read ops for the Vdisk	
Outstanding Ops – Write	The number of outstanding write ops for the Vdisk	
Outstanding Ops – Estore	The number of outstanding ops to the extent store for the Vdisk	
Ops/s – Read	The number of current read operations per second for the Vdisk	
Ops/s – Write	The number of current write operations per second for the Vdisk	
Ops/s – Error	The number of current error (failed) operations per second for the Vdisk	
Ops/s – Random		
KB/s – Read	The read throughput in KB/s for the Vdisk	
KB/s – Write	The write throughput in KB/s for the Vdisk	
Avg latency (usec)	The average op latency in micro seconds for the Vdisk	
Avg op size	The average op size in bytes for the Vdisk	
Avg qlen	The average queue length for the Vdisk	
% busy		

2009 Stargate – Extent Store



Metric	Explanation	Threshold/Target
Disk Id	The disk id of the physical device	
Mount point	The mount point of the physical device	
Outstanding Ops – QoS Queue	The number of (primary/secondary) ops for the device	
Outstanding Ops – Read	The number of outstanding read ops for the device	
Outstanding Ops – Write	The number of outstanding write ops for the device	
Outstanding Ops – Replicate		
Outstanding Ops – Read Replica		
Ops/s – Read	The number of current read operations per second for the device	
Ops/s – Write	The number of current write operations per second for the device	
Ops/s – Error	The number of current error (failed) operations per second for the device	
Ops/s – Random		

KB/s – Read	The read throughput in KB/s for the device
KB/s – Write	The write throughput in KB/s for the device
Avg latency (usec)	The average op latency in micro seconds for the device
Avg op size	The average op size in bytes for the device
Avg qlen	The average queue length for the device
Avg qdelay	The average queue delay for the device
% busy	
Size (GB)	
Total usage (GB)	The total usage in GB for the device
Unshared usage (GB)	
Dedup usage (GB)	
Garbage (GB)	
Egroups	The number of extent groups for the device
Corrupt Egroups	The number of corrupt (bad) extent groups for the device

top

Gflags

Coming soon 😃

top

Troubleshooting

# Find cluster error logs

```
# Description: Find ERROR logs for the cluster

for i in `svmips`;do ssh $i "cat ~/data/logs/<COMPONENT NAME or *>.ERROR";done

# Example for Stargate
for i in `svmips`;do ssh $i "cat ~/data/logs/Stargate.ERROR";done
```

# Find cluster fatal logs

```
# Description: Find FATAL logs for the cluster

for i in `svmips`;do ssh $i "cat ~/data/logs/<COMPONENT NAME or *>.FATAL";done

# Example for Stargate
for i in `svmips`;do ssh $i "cat ~/data/logs/Stargate.FATAL";done
```

top

Book of vSphere

Architecture

To be input

top

How It Works

Array Offloads – VAAI

The Nutanix platform supports the VMware APIs for Arry Integration (VAAI) which allows the hypervisor to offload certain tasks to the array. This is much more efficient as the hypervisor doesn't need to be the "man in the middle". Nutanix currently supports the VAAI primitives for NAS including the 'full file clone', 'fast file clone' and 'reserve space' primitives. Here's a good article explaining the various primitives: LINK. For both the full and fast file clones a NDFS "fast clone" is done meaning a writable snapshot (using re-direct on write) for each clone is created. Each of these clones has its own block map meaning that chain depth isn't anything to worry about. The following will determine whether or not VAAI will be used for specific scenarios:

- Clone VM with Snapshot –> VAAI will **NOT** be used
- Clone VM without Snapshot which is Powered Off –> VAAI **WILL** be used
- Clone VM to a different Datastore/Container –> VAAI will **NOT** be used
- Clone VM which is Powered On –> VAAI will **NOT** be used

These scenarios apply to VMware View:

- View Full Clone (Template with Snapshot) –> VAAI will **NOT** be used
- View Full Clone (Template w/o Snapshot) –> VAAI **WILL** be used
- View Linked Clone (VCAI) –> VAAI **WILL** be used

You can validate VAAI operations are taking place by using the 'NFS Adapter' Activity Traces page.

top

Administration

To be input

top

Important Pages

To be input

top

Command Reference

# ESXi cluster upgrade

```
# Description: Perform an automated upgrade of ESXi hosts using the CLI

# Upload upgrade offline bundle to a Nutanix NFS container

# Log in to Nutanix CVM

# Perform upgrade
for i in `hostips`;do echo $i && ssh root@$i "esxcli software vib install -d /vmfs/volumes/<Datastore Name>/<Offline 

# Example
for i in `hostips`;do echo $i && ssh root@$i "esxcli software vib install -d /vmfs/volumes/NTNX-upgrade/update-from-e
```

Performing a rolling reboot of ESXi hosts: For PowerCLI on automated hosts reboots, SEE HERE

# Restart ESXi host services

```
# Description: Restart each ESXi hosts services in a incremental manner

for i in `hostips`;do ssh root@$i "services.sh restart";done
```

# Display ESXi host nics in 'Up' state

```
# Description: Display the ESXi host's nics which are in a 'Up' state

for i in `hostips`;do echo $i && ssh root@$i esxcfg-nics -l | grep Up;done
```

# Display ESXi host 10GbE nics and status

```
# Description: Display the ESXi host's 10GbE nics and status

for i in `hostips`;do echo $i && ssh root@$i esxcfg-nics -l | grep ixgbe;done
```

# Display ESXi host active adapters

```
# Description: Display the ESXi host's active, standby and unused adapters

for i in `hostips`;do echo $i &&  ssh root@$i "esxcli network vswitch standard policy failover get --vswitch-name vSwi
```

# Display ESXi host routing tables

```
# Description: Display the ESXi host's routing tables

for i in `hostips`;do ssh root@$i 'esxcfg-route -l';done
```

# Check if VAAI is enabled on datastore

```
# Description: Check whether or not VAAI is enabled/supported for a datastore

vmkfstools -Ph /vmfs/volumes/<Datastore Name>
```

# Set VIB acceptance level to community supported

```
# Description: Set the vib acceptance level to CommunitySupported allowing for 3rd party vibs to be installed

esxcli software acceptance set --level CommunitySupported
```

# Install VIB

```
#Description: Install a vib without checking the signature

esxcli software vib install --viburl=/<VIB directory>/<VIB name> --no-sig-check

# OR

esxcli software vib install --depoturl=/<VIB directory>/<VIB name> --no-sig-check
```

# Check ESXi ramdisk space

```
# Description: Check free space of ESXi ramdisk

for i in `hostips`;do echo $i; ssh root@$i 'vdf -h';done
```

# Clear pynfs logs

```
# Description: Clears the pynfs logs on each ESXi host

for i in `hostips`;do echo $i; ssh root@$i '> /pynfs/pynfs.log';done
```

top

Metrics & Thresholds

To be input

top

Troubleshooting

To be input

top

Book of Hyper-V

Architecture

To be input

top

How It Works

Array Offloads – ODX

The Nutanix platform supports the Microsoft Offloaded Data Transfers (ODX) which allows the hypervisor to offload certain tasks to the array. This is much more efficient as the hypervisor doesn't need to be the "man in the middle". Nutanix currently supports the ODX primitives for SMB which include full copy and zeroing operations. However contrary to VAAI which has a "fast file" clone operation (using writable snapshots) the ODX primitives do not have an equivalent and perform a full copy. Given this, it is more efficient to rely on the native NDFS clones which can currently be invoked via nCLI, REST, or Powershell CMDlets

Currently ODX **IS** invoked for the following operations:

- In VM or VM to VM file copy on NDFS SMB share
- SMB share file copy
- Deploy template from SCVMM Library (NDFS SMB share) – NOTE: Shares must be added to the SCVMM cluster using short names (eg. not FQDN). An easy way to force this is to add an entry into the hosts file for the cluster (eg. 10.10.10.10 nutanix-130).

ODX is **NOT** invoked for the following operations:

- Clone VM through SCVMM
- Deploy template from SCVMM Library (non-NDFS SMB Share)
- XenDesktop Clone Deployment

You can validate ODX operations are taking place by using the 'NFS Adapter' Activity Traces page (yes, I said NFS, even though this is being performed via SMB). The operations activity show will be 'NfsSlaveVaaiCopyDataOp' when copying a vDisk and 'NfsSlaveVaaiWriteZerosOp' when zeroing out a disk.

top

Administration

To be input

top

Important Pages

To be input

top

Command Reference

# Execute command on multiple remote hosts

```
# Description: Execute a powershell on one or many remote hosts

$targetServers = "Host1","Host2","Etc"
Invoke-Command -ComputerName  $targetServers {
         <COMMAND or SCRIPT BLOCK>
}
```

# Check available VMQ Offloads

```
# Description: Display the available number of VMQ offloads for a particular host

gwmi –Namespace “root\virtualization\v2” –Class Msvm_VirtualEthernetSwitch | select elementname, MaxVMQOffloads
```

# Disable VMQ for VMs matching a specific prefix

```
# Description: Disable VMQ for specific VMs

$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix} | Get-VMNetworkAdapter | Set-VMNetworkAdapter -VmqWeight 0
```

# Enable VMQ for VMs matching a certain prefix

```
# Description: Enable VMQ for specific VMs

$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix} | Get-VMNetworkAdapter | Set-VMNetworkAdapter -VmqWeight 1
```

# Power-On VMs matching a certain prefix

```
# Description: Power-On VMs matchin a certain prefix

$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix -and $_.StatusString -eq "Stopped"} | Start-VM
```

# Shutdown VMs matching a certain prefix

```
# Description: Shutdown VMs matchin a certain prefix

$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix -and $_.StatusString -eq "Running"}} | Shutdown-VM -RunAsynchronously
```

# Stop VMs matching a certain prefix

```
# Description: Stop VMs matchin a certain prefix

$vmPrefix = "myVMs"
```

```
Get-VM | Where {$_.Name -match $vmPrefix} | Stop-VM
```

# Get Hyper-V host RSS settings

```
# Description: Get Hyper-V host RSS (recieve side scaling) settings

Get-NetAdapterRss
```

top

Metrics & Thresholds

To be input

top

Troubleshooting

To be input

top

Revisions

1. 09-04-2013 | Initial Version
2. 09-04-2013 | Updated with components section
3. 09-05-2013 | Updated with I/O path overview section
4. 09-09-2013 | Updated with converged architecture section
5. 09-11-2013 | Updated with data structure section
6. 09-24-2013 | Updated with data protection section
7. 09-30-2013 | Updated with data locality section
8. 10-01-2013 | Updated with shadow clones section
9. 10-07-2013 | Updated with scalable metadata section
10. 10-11-2013 | Updated with elastic dedupe engine
11. 11-01-2013 | Updated with networking and I/O section
12. 11-07-2013 | Updated with CVM autopathing section
13. 01-23-2014 | Updated with new content structure and layout
14. 02-10-2014 | Updated with storage layers and monitoring
15. 02-18-2014 | Updated with array offloads sections
16. 03-12-2014 | Updated with genesis
17. 03-17-2014 | Updated spelling and grammar
18. 03-19-2014 | Updated with apis and interfaces section
19. 03-25-2014 | Updated script block formatting
20. 03-26-2014 | Updated with dr & protection domain NCLI commands
21. 04-15-2014 | Updated with failure domains section and 4.0 updates

22. 04-23-2014 | Updated with command to echo factory_config.json
23. 05-28-2014 | Updated with snapshots and clones section
24. 06-09-2014 | Updated with multi-site disaster recovery section
25. 06-10-2014 | Updated cluster components graphic

top

Like 95 Tweet 473 +1 13

Categories

- AskSteve
- Benchmarking
- Nutanix
- Powershell
- Splunk
- SQL
- Uncategorized
- VMware

Archives

- March 2014 (1)
- February 2014 (2)
- November 2013 (2)
- September 2013 (3)
- August 2013 (1)
- July 2013 (2)
- June 2013 (6)
- May 2013 (3)

Legal Mumbo Jumbo