Nutanix Best Practices

Virtualizing MongoDB on Nutanix

NUTANIX

MongoDB Enterprise Certified

Version 1.0

July 2015

BP-2023

1 Executive Summary 4
2 Introduction 5
2.1 Audience 5
2.2 Purpose 5
3 Solution Overview 6
3.1 What is the Nutanix Architecture? 6
3.2 Acropolis Hypervisor 9
3.2.1 Acropolis Architecture 10
3.2.2 VM Networking 11
3.2.3 IP Address Management 12
3.3 MongoDB Architecture 12
3.3.1 MongoDB Architecture—High Availability 14
3.3.2 MongoDB Architecture—Scalability 15
3.3.3 MongoDB—Sharding 17
3.4 Running MongoDB on Nutanix 19
3.5 Nutanix—Data Transforms 20
3.6 Nutanix—Snapshots and Cloning 21
3.6.1 Nutanix—Snapshot for Backups and DR 23
3.6.2 Nutanix—Cloning Virtual Machines 24
4 Solution Design 26
5 Network 29
6 Site Planning and Solution Application 31
7 Best Practices 38
7.1 General 38
7.2 MongoDB 39
7.3 Nutanix 42
7.4 Network Infrastructure 42
8 Conclusion 44
9 Appendix 45
9.1 Appendix A: Replication Lag 45
9.2 Appendix B: References 45
9.3 Table of Figures 45
9.4 Table of Tables 46
9.5 Appendix C: Meet the Author 48

1 Executive Summary

In this document we make recommendations for configuring and deploying virtualized MongoDB implementations on Nutanix. We show how the underlying Nutanix Xtreme Computing Platform (XCP) easily scales to incorporate web-scale, big-data architectures. The Nutanix platform supports a range of hypervisors—including VMware vSphere, Microsoft Hyper-V, and Nutanix Acropolis—and is the ideal environment for running large-scale, virtualized MongoDB workloads.

New database technologies such as MongoDB can take advantage of specific Nutanix functionality and elastic scalability to deliver both sustained high performance and the fast, easy provisioning required to scale both vertically and horizontally. MongoDB is a general purpose, open-source document database, the kind of system described as OLTP-style "Systems of Engagement." Popular use cases include the Internet of Things (IOT), inventory management, shopping cart, ad serving, content management, messaging applications, and real-time analytics.

The Nutanix platform greatly simplifies the workflows for deploying and maintaining the large and often dispersed virtual server landscapes that distributed MongoDB application installs require. By means of a single, intuitive GUI, administrators can add nodes to the underlying Nutanix platform cluster with the click of button; this increases the cluster-wide access to the Nutanix Distributed File System (NDFS) and allows MongoDB virtual machines (VMs) to be deployed or relocated anywhere across the cluster estate.

Nutanix's automated storage tiering feature ensures that VMs always access data locally from an SSD-backed, cluster-wide, "hot" tier. This brings highly desirable storage I/O throughput benefits to MongoDB servers. Such Information Lifecycle Management (ILM) functionality also handles data balancing and leveling across compute and storage nodes.

Nutanix cluster software upgrades require no downtime. These one click operations apply to the Nutanix Operating System (NOS), storage software, firmware, and hypervisor. The workflows are simplified to the point where they are practically hands free after initiation.

Nutanix virtual machine snapshots and disaster recovery solutions can greatly simplify MongoDB backups. Regardless of how the database storage is laid out within the virtual machine, Nutanix enables simultaneous point-in-time snapshots of an entire group of virtual machines. Such backups can help create development and test environments. These can be local, remote, or in the public cloud using Nutanix Cloud Connect. This is an enablement feature that allows MongoDB virtual machine migration between Nutanix on-premise clusters and the public cloud.

Nutanix Prism provides rich, full-stack analytics for monitoring virtualized MongoDB implementations—this includes hardware, hypervisor, and virtual machines layers. For automation purposes, a REST API enables control of the entire hyperconverged infrastructure.

2 Introduction

2.1 Audience

This best practice document is part of the Nutanix Solutions Library and is intended for architecting, designing, managing, and supporting Nutanix infrastructures. Consumers of the document should be familiar with one of the supported hypervisors, such as ESXi, Hyper-V, and Nutanix Acropolis, as well as MongoDB and Nutanix.

We have organized this document to enable successful design, implementation, and transition to operation.

2.2 Purpose

This document will cover the following subject areas:

- Overview of the Nutanix solution.
- Overview of MongoDB and its use cases.
- The benefits of MongoDB on Nutanix.
- Design and configuration considerations when architecting a MongoDB solution on Nutanix.
- Best practices for configuring Nutanix when running MongoDB at scale in production.

3 Solution Overview

3.1 What is the Nutanix Architecture?

Nutanix delivers a hyper-converged infrastructure solution purpose-built for virtualization and cloud environments. This solution brings the performance and economic benefits of web-scale architecture to the enterprise through the Nutanix Distributed File System (NDFS).

Attributes of this solution include:

- Storage and compute resources hyper-converged on x86 servers.
- System intelligence located in software.
- Data, metadata, and operations fully distributed across entire cluster of x86 servers.
- Self-healing to tolerate and adjust to component failures.
- API-based automation and rich analytics.

The Nutanix platform does not rely on traditional SAN/NAS storage or expensive storage network interconnects. It combines highly dense storage and server compute (CPU and RAM) into a single platform building block. Each building block is based on industry-standard Intel processor technology and delivers a unified, scale-out, shared-nothing architecture with no single points of failure.

The Nutanix solution has no LUNs to manage, no RAID groups to configure, and no complicated storage multipathing to set up. All storage management is VM-centric and storage I/O is optimized by NDFS at the VM virtual disk level. There is one shared pool of storage that includes flash-based SSDs for high performance and low latency, and high capacity HDDs for affordable capacity. Data is automatically tiered by the file system across different types of storage devices using Nutanix's intelligent data placement algorithms. These algorithms make sure the most frequently used data is available in memory or in flash for the fastest possible performance.

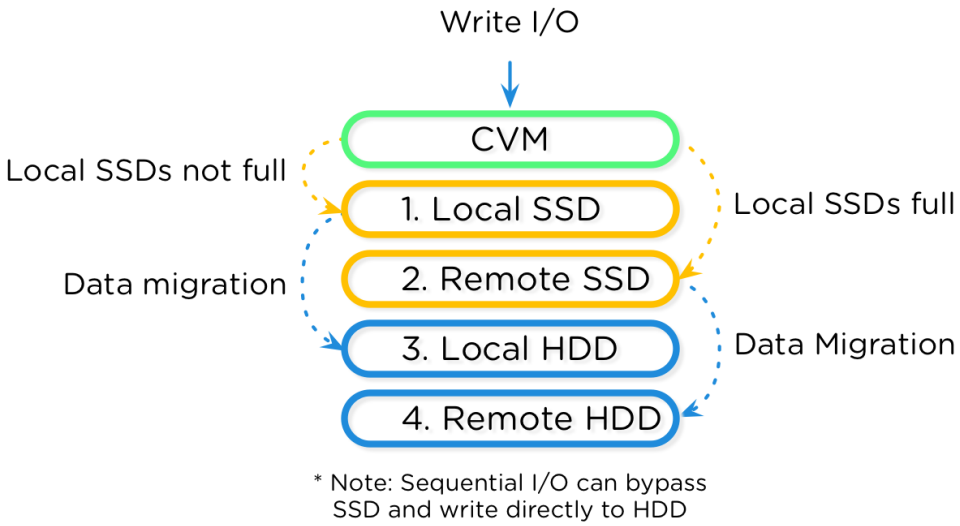


Figure 1: Information Lifecycle Management

With the Nutanix Distriibuted File System (NDFS), a Controller VM (CVM) writes data to local flash memory for fast acknowledgment; the CVM also handles read operations locally for reduced latency and fast data delivery.

Figure 2 shows an overview of the Nutanix architecture, including each hypervisor host (VMware ESXi, Microsoft Hyper-V, or Acropolis), user VMs, the Nutanix storage controller VM (Nutanix Controller VM), and its local disk devices. Each Controller VM is directly connected to the local storage controller and its associated disks. By using local storage controllers on each host, access to data through NDFS is localized, thereby reducing storage I/O latency. NDFS ensures that writes are replicated synchronously to at least one other node in the system, distributing data throughout the cluster for resiliency and availability. Having a local storage controller on each host ensures that storage performance as well as storage capacity increases when adding nodes.

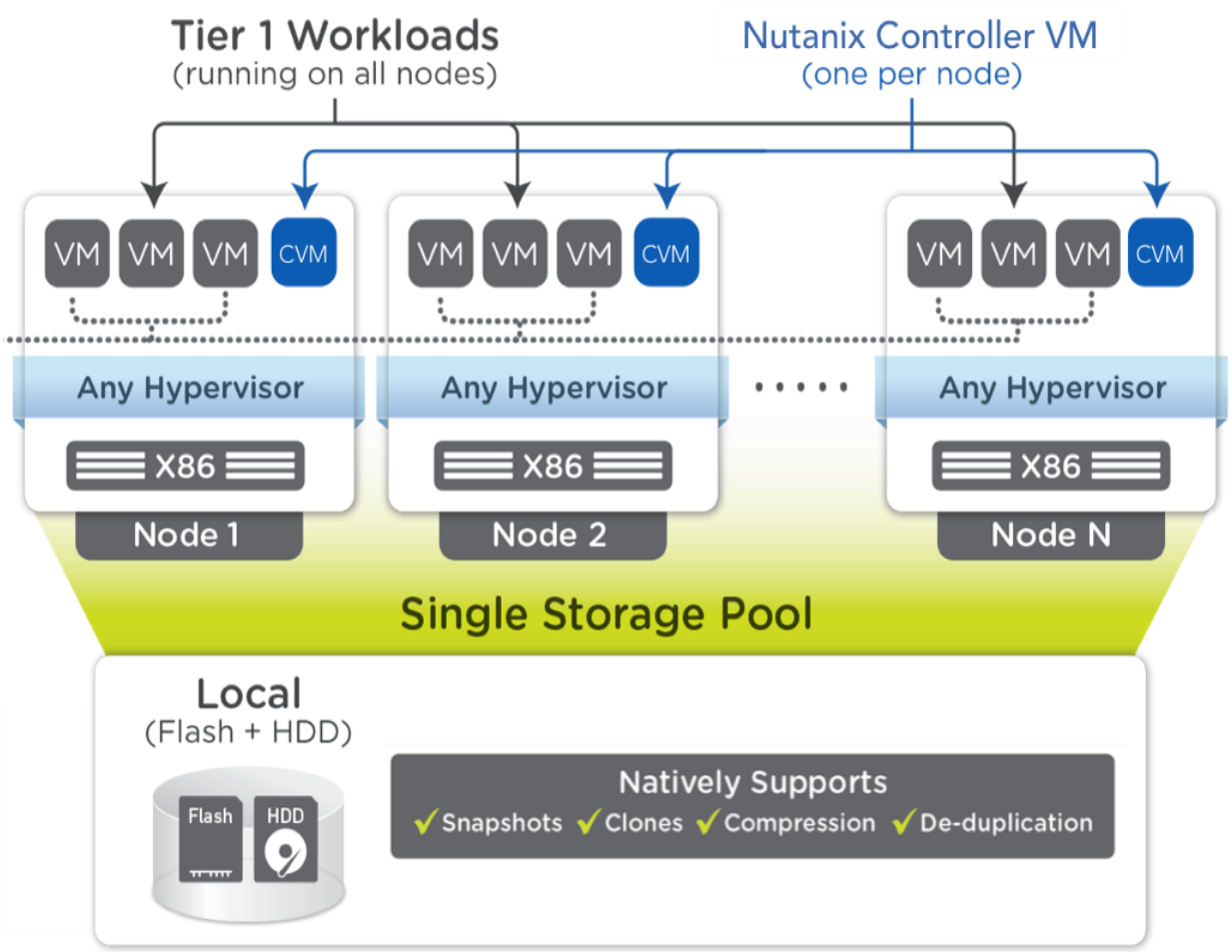


Figure 2: Overview of the Nutanix Architecture

Local storage for each node in the architecture appears to the hypervisor as one large pool of shared storage. This allows NDFS to support all key virtualization features. In the case of VMware vSphere, this includes support for VMware DRS, VMware High Availability, VMware Fault Tolerance, and many other features. Data localization also allows for performance and QoS to take place on each host, and thus noisy VMs do not impact their neighbors' performance. This functionality allows large, mixed-workload vSphere clusters that are more efficient and more resilient to failure when compared to traditional architectures with standalone, shared, and dual-controller storage arrays.

When VMs move from one hypervisor node to another, such as during vMotion or HA, 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 request will be forwarded by the local CVM to the remote CVM. All write I/O will occur locally. NDFS will detect that I/O is occurring from a different node and will migrate the data locally in the background, allowing for all read I/O to now be served locally. The data will be migrated on read only to minimize network utilization. Figure 3 shows how data will follow the VM as it moves between hypervisor nodes.

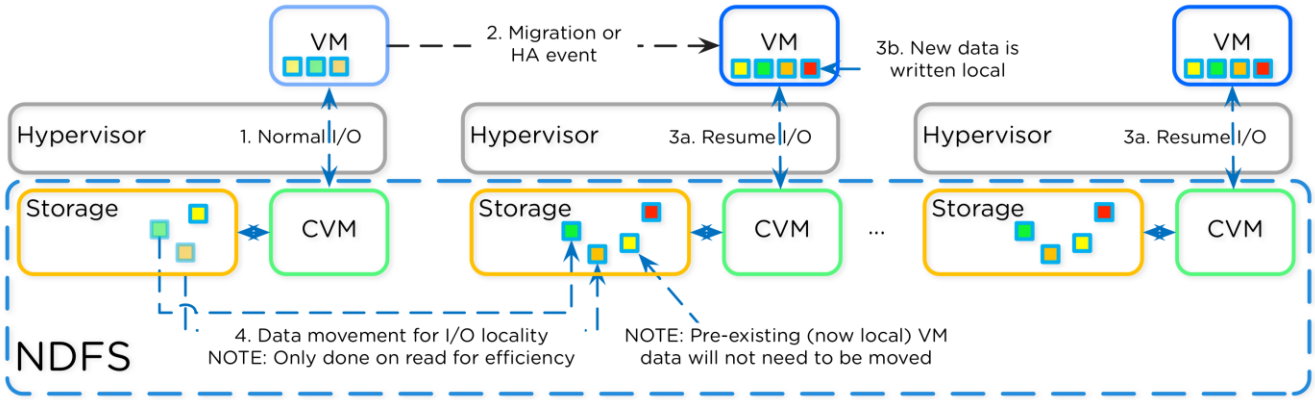


Figure 3: Data Locality and vMotion

Another key feature of NDFS is the Nutanix Elastic Deduplication Engine, a software-driven, massively scalable, and intelligent data reduction technology. It increases the effective capacity in the disk tier, as well as the RAM and flash cache tiers of the system, by eliminating duplicate data. This substantially increases storage efficiency, while also improving performance due to larger effective cache capacity in RAM and flash. Each node in the cluster performs deduplication individually, allowing for efficient and uniform deduplication at scale. This technology is increasingly effective with full persistent clones or P2V migrations.

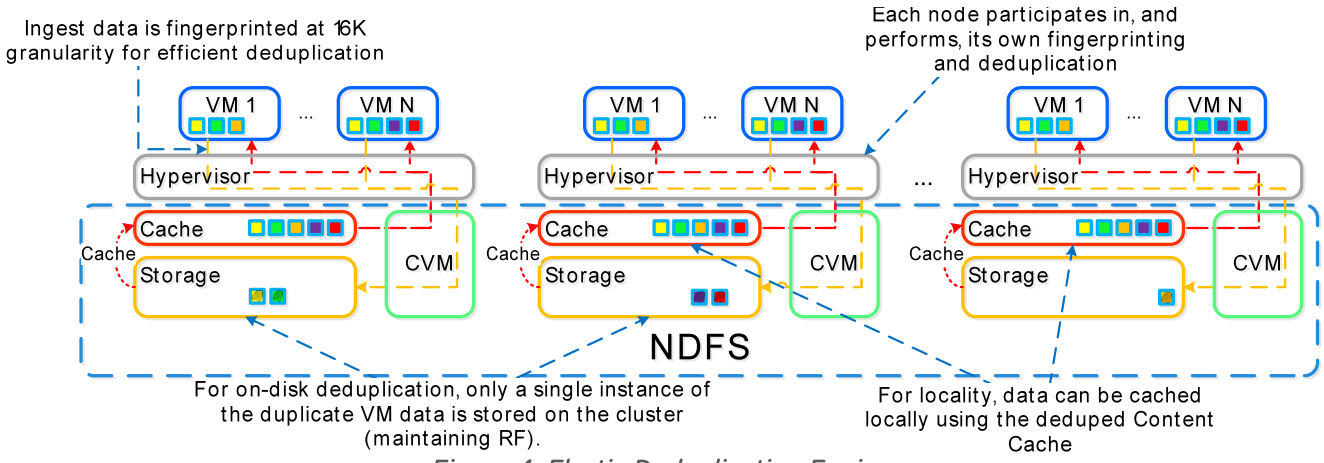


Figure 4: Elastic Deduplication Engine

3.2 Acropolis Hypervisor

The Acropolis Hypervisor integrates tightly with NDFS and provides cost-effective virtualization with consumer-grade management. Acropolis includes a distributed VM management service responsible for storing VM configuration, making scheduling decisions, and exposing a management interface. The Acropolis Hypervisor is based on Linux KVM (Kernel-based Virtual Machine), which is a full type-1 virtualization solution. The Prism interface, a robust REST API, and an interactive command line interface called aCLI (Acropolis CLI) combine to eliminate the complex management associated with KVM.

The Acropolis Hypervisor provides the following capabilities:

- **Virtual machine storage**. Storage devices for the virtual machine, such as SCSI and IDE devices.

- **Crash consistent snapshots.** Includes VM configuration and disk contents.
- **Virtual networks (L2).** Layer 2 network communication between virtual machines and to the external network, with support for multiple vSwitches and VLANS.
- **Managed networks (L3).** IP Address Management (IPAM) to provide Layer 3 addresses for virtual machines.
- **App Mobility Fabric.** Enables unprecedented workload flexibility, including moving workloads among Acropolis nodes, among different hypervisors (ESXi, Hyper-V), and even to the public cloud. The App Mobility Fabric also delivers a range of virtualization capabilities, such as live migration, HA, and disaster recovery.

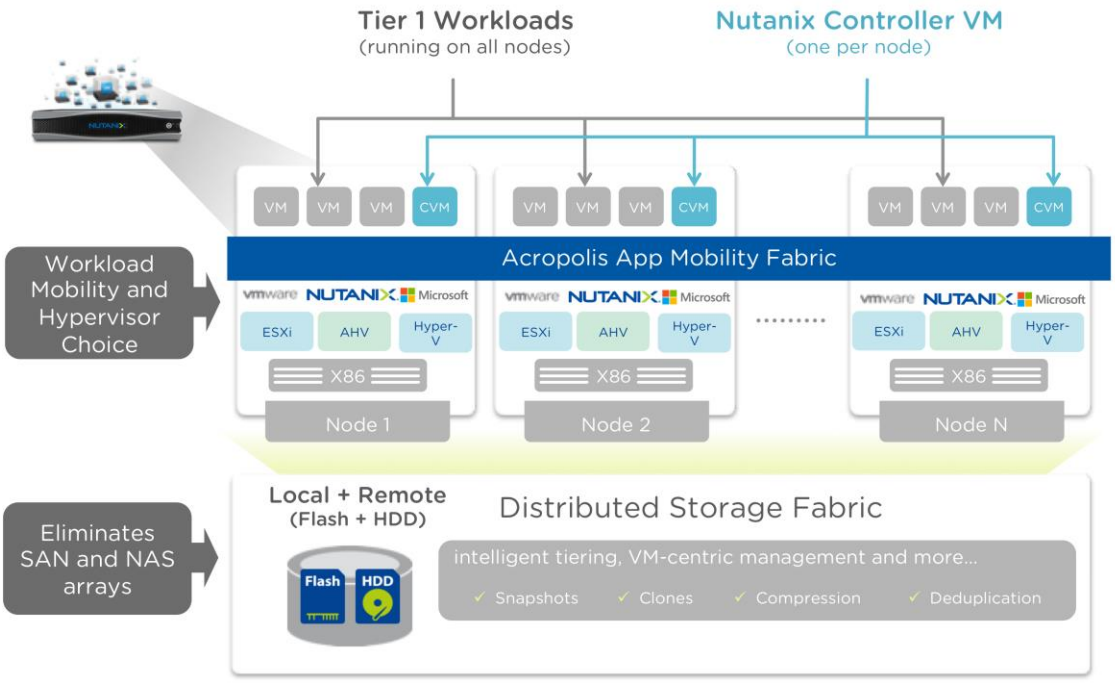


Figure 5: The Nutanix Architecture with Acropolis

3.2.1 Acropolis Architecture

The Acropolis Hypervisor extends KVM's base functionality. True to web-scale design principles, Acropolis is available on all nodes with a shared-nothing architecture. Command and control is maintained through a master-subordinate mechanism where the master is elected by the cluster nodes and replaced by a new election in the event of a failure. On a single Acropolis node, the Controller Virtual Machine accesses disks directly using PCI passthrough, as shown in the figure below.

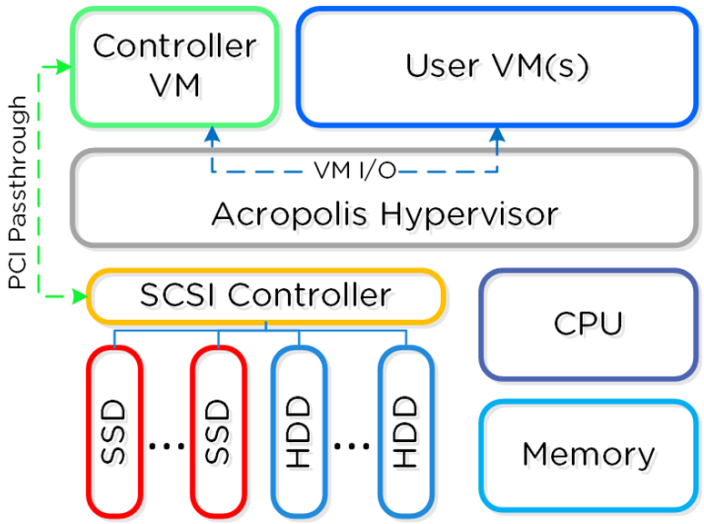


Figure 6: Acropolis and PCI Passthrough

3.2.2 VM Networking

Acropolis uses Open vSwitch (OvS) for all VM networking, as shown in Figure 7. Open vSwitch manages the bonded interfaces (bond0) that connect the hypervisor to the physical network. All VMs connect into one or more virtual switches for inter-VM communication. VMs can also communicate externally if the vSwitch has external interfaces. Each VM NIC connects to an Open vSwitch tap interface (vnet0, tap1). The Acropolis host connects to the CVM through a local Linux bridge (vnet1 to Local) for iSCSI storage traffic.

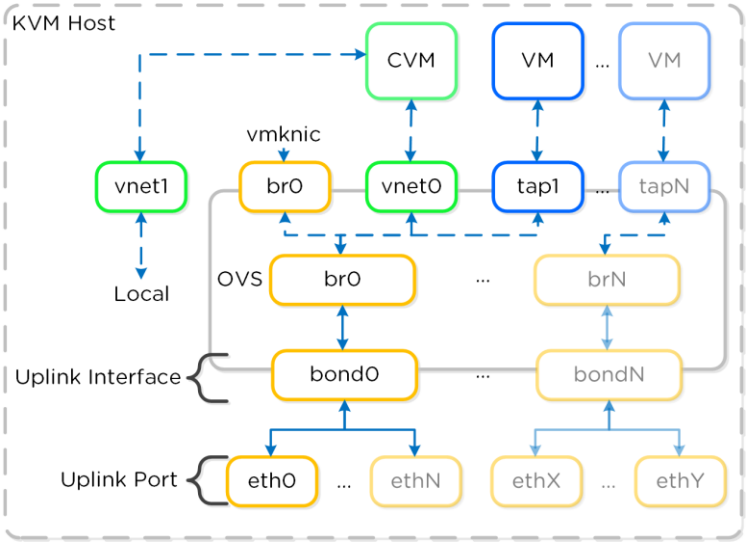


Figure 7: Acropolis and VM Networking

Note: Acropolis virtualizes CPUs using an HVM (Hardware Virtual Machine) device type. Other devices, such as network adapters, use paravirtualized devices that may require VirtIO drivers in your guest OS.

3.2.3 IP Address Management

An Acropolis network can be either managed or unmanaged. When unmanaged, the VM traffic is simply passed to the external network with proper VLAN tags. When managed, the Acropolis IP address management (IPAM) solution leverages VXLAN and OpenFlow rules to intercept DHCP requests and respond with addresses from a configured pool. In the figure below, we show a sample DHCP request using the Nutanix IPAM solution:

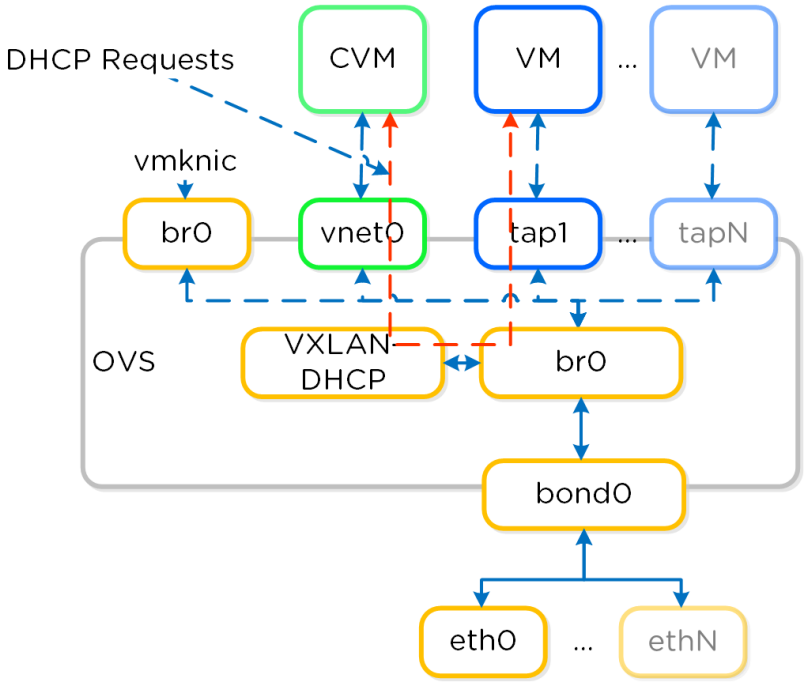


Figure 8: Acropolis IPAM

3.3 MongoDB Architecture

MongoDB is a document database that provides high performance, high availability, and easy scalability. Using in-memory computing, MongoDB ensures high-performance of both read and write capabilities. Native replication and failover enables enterprise-grade resilience. MongoDB scalability ranges from single-server deployments to larger, multi-location configurations.

The primary instance of a MongoDB database consists of a single mongod daemon (see Figure 9 below).

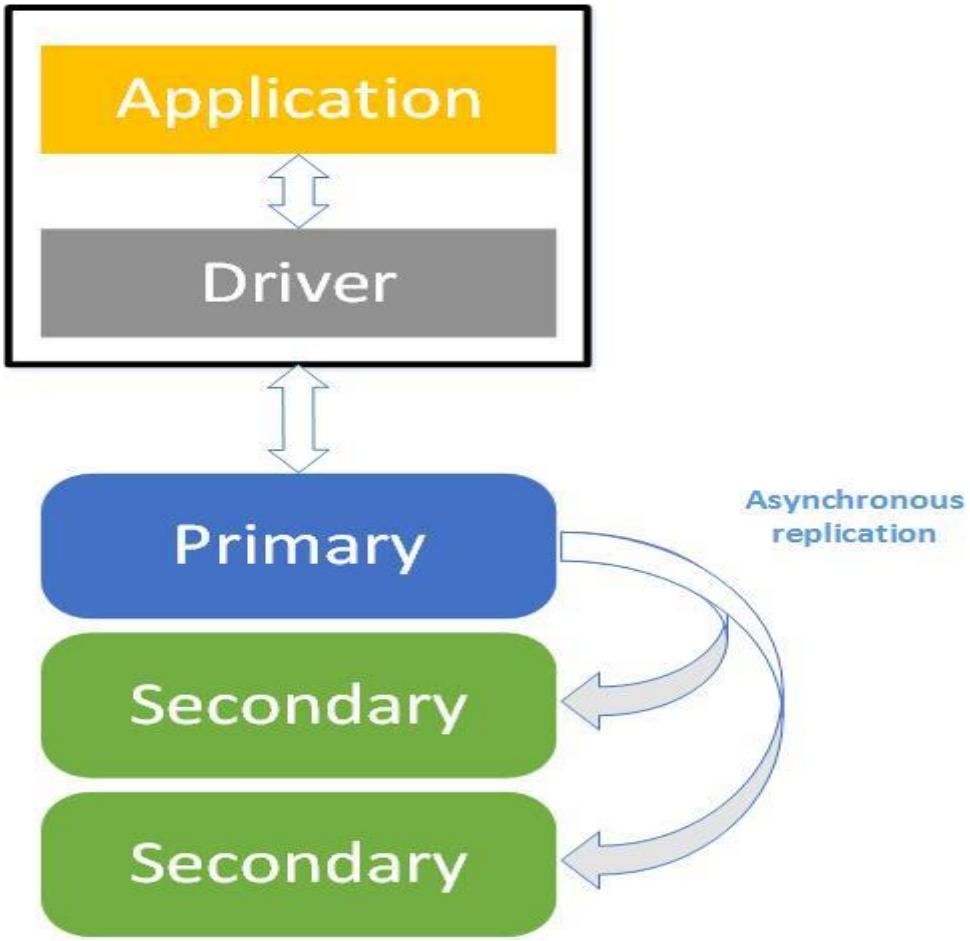


Figure 9: MongodDB Replica Set—The Basic Unit of Sharding

This daemon handles all read and writes by default. Both read and write operations are memory based. Data is written to memory first and then to a journal file on disk every journalCommitInterval milliseconds (default value of 100). Then the OS flushes the journal file contents, at 60 second intervals, to the permanent data files residing on disk. Write durability can be configured per transaction within the application using the following levels of acknowledgement:

- Data committed to memory.
- Data written to journal.
- Data written to data files.
- Data written to one or more replica or secondary nodes.

All read operations are read from memory. The working set of in-memory data includes both recently used pages of data and any associated indexes. When a required page of memory can't be found in RAM, a page fault occurs, and the required data must be read from a location on disk. If free space in memory is not available to hold the data, then a page of data currently held in RAM must be purged to disk. This will free up the necessary space and allow data to be read in to memory from disk.

Nutanix SSD-backed hot tier storage will lower the cost of any page-out activity to disk. Additionally, it will reduce any performance degradation due to periodic burst write workloads or when the working set size exceeds available memory.

3.3.1 MongoDB Architecture—High Availability

To increase redundancy and data availability, MongoDB employs replica sets. A replica set consists of three or more mongod instances, located on one primary and two or more secondary nodes. The primary node receives all writes to ensure strict consistency. By default, all read operations also go to the primary node, but the secondary nodes can be configured to receive read operations if required. Note that, due to what is termed replication lag, the secondary nodes may not always contain the most current data. Replication lag (see Appendix A: Replication Lag) can occur under any of the following underlying conditions:

- Under-specified secondary node.
- Large write bursts.
- Index build.
- Secondary locked for backup.
- Secondary offline.

In the event of a failure, such as a network partition, or a delay that mimics such a partition, then the primary node may be unable to communicate with other members of the replica set. Ordinarily, a heartbeat is sent between all of the replica set members (see Figure 10 below). If a heartbeat is not received within 10 seconds, the failing set member is marked inaccessible. If that member is the primary node then a replica set election occurs to find a new primary. A secondary node with sufficient votes gets elected as the new primary node. There needs to be an odd number of set members to ensure there are no tied elections. In the event of even-numbered replica set membership, a primary-node election arbiter process can be used.

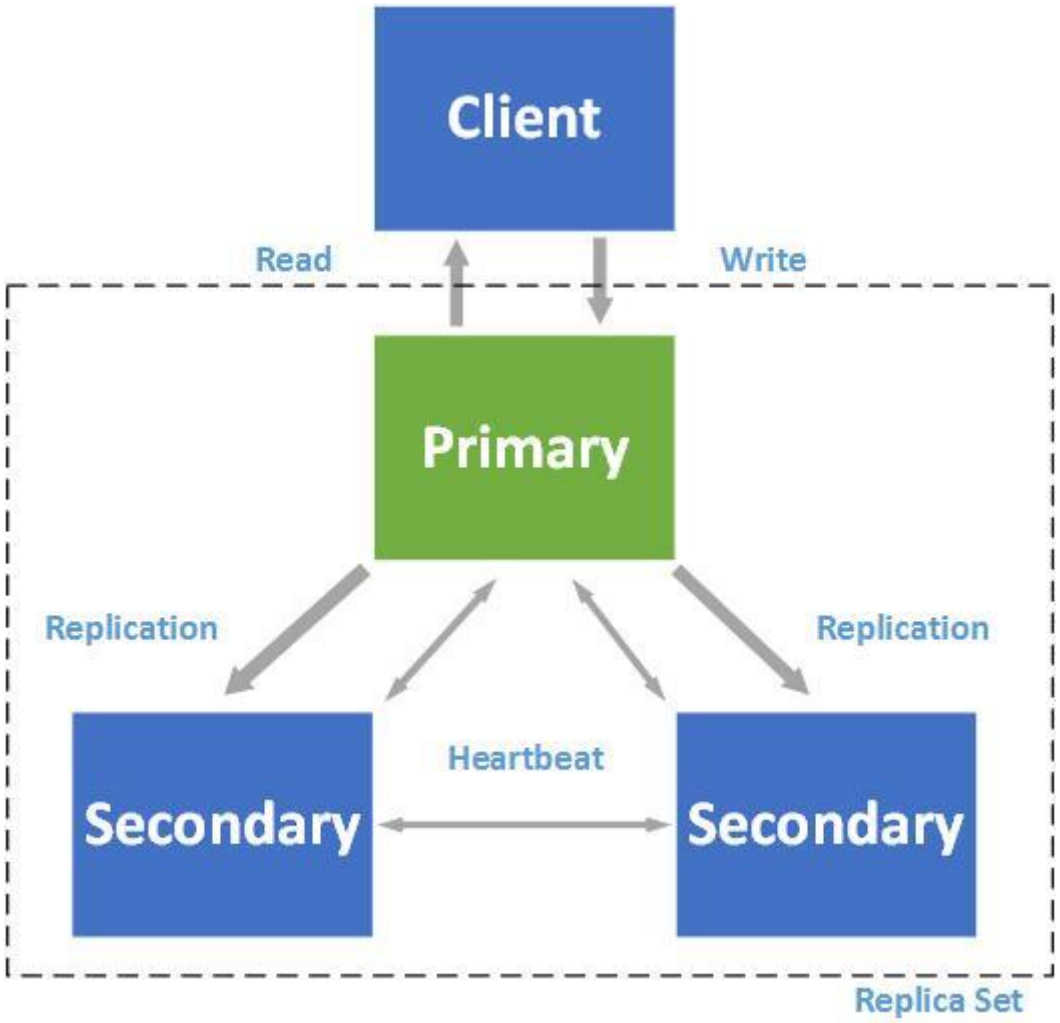


Figure 10: Replica Set Heartbeating

3.3.2 MongoDB Architecture—Scalability

We can look at two distinct methods for scaling in a highly distributed MongoDB on Nutanix environment:

Vertical scaling

Vertical scaling (often termed scale-up) increases performance by upgrading the hardware capabilities of the individual VMs—for example, increasing RAM, increasing the SSD-backed storage layer, additional CPU, and so on. MongoDB allows rolling upgrades for both primary and secondary VMs in a replica set. By far the most important criterion in any MongoDB architecture is having sufficient RAM. MongoDB performs optimally when the entire working set of required pages and associated indexes resides in RAM. Memory sizing should also account for a projected increase in memory usage over time.

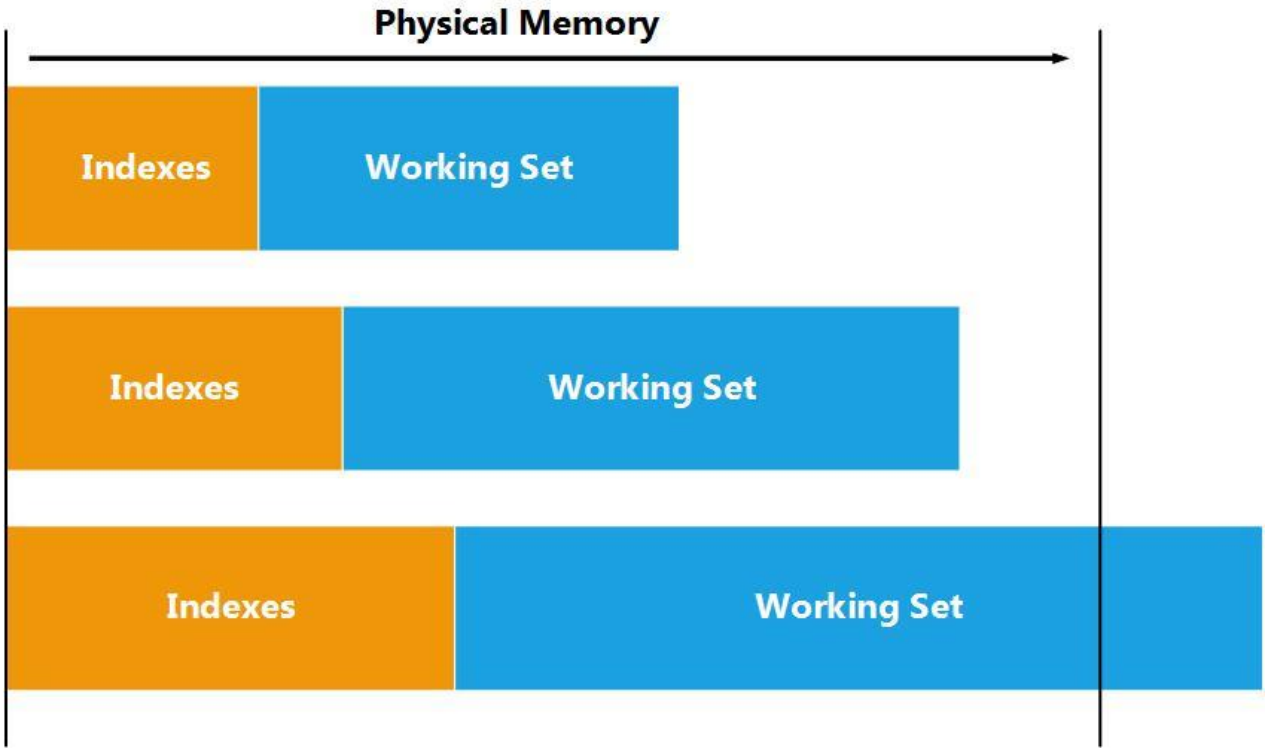


Figure 11: Working Set Size Growth Over Time Compared to Physical RAM

Horizontal scaling

Horizontal scaling (or scale out) involves distributing data or workload across all of the systems within a cluster. MongoDB achieves this through data sharding, which allows support for very large datasets and high-throughput operations. The entire data set is split into independent replica sets that together form the complete logical database entity. Data sharding allows you to scale your cluster linearly by adding more VMs. The Nutanix Xtreme Computing Platform lets you increase capacity without any downtime. This feature is particularly important in a web environment where load can increase suddenly and bringing down the website for maintenance can cost your business a great deal of revenue.

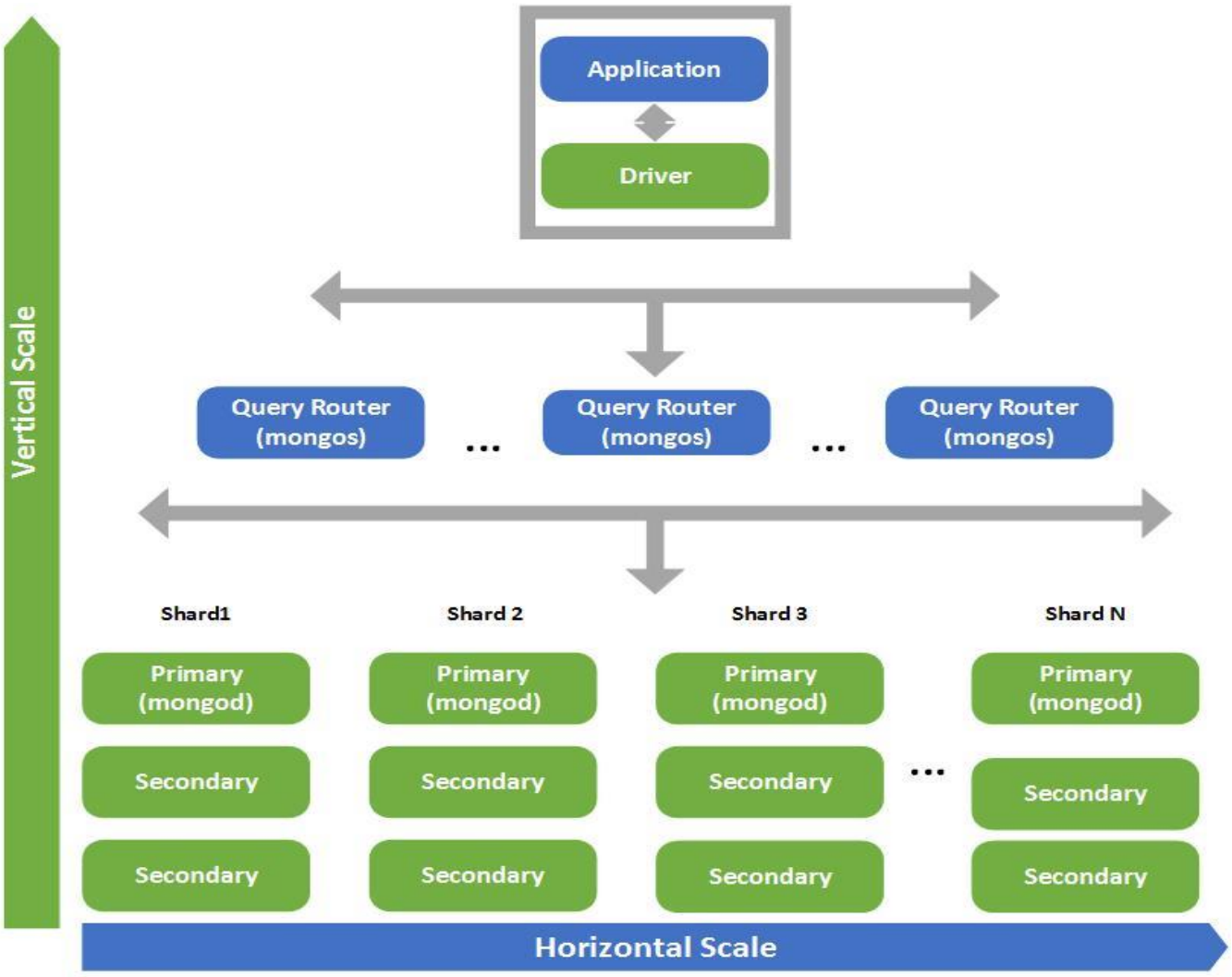


Figure 12: MongoDB Scaling—Logical Overview

3.3.3 MongoDB—Sharding

Administrators usually deploy sharding when all mechanisms for vertical scaling have been tried and no longer reduce contention. One of the primary reasons for sharding is that your system's active working set size is going to exceed the maximum RAM capacity (see Figure 13 below). Again, all reads and writes should happen in-memory to avoid paging-out to disk and, with this, additional latency. MongoDB provides the db.serverStatus command to view an estimate of the current working set size (see Best Practices section below).

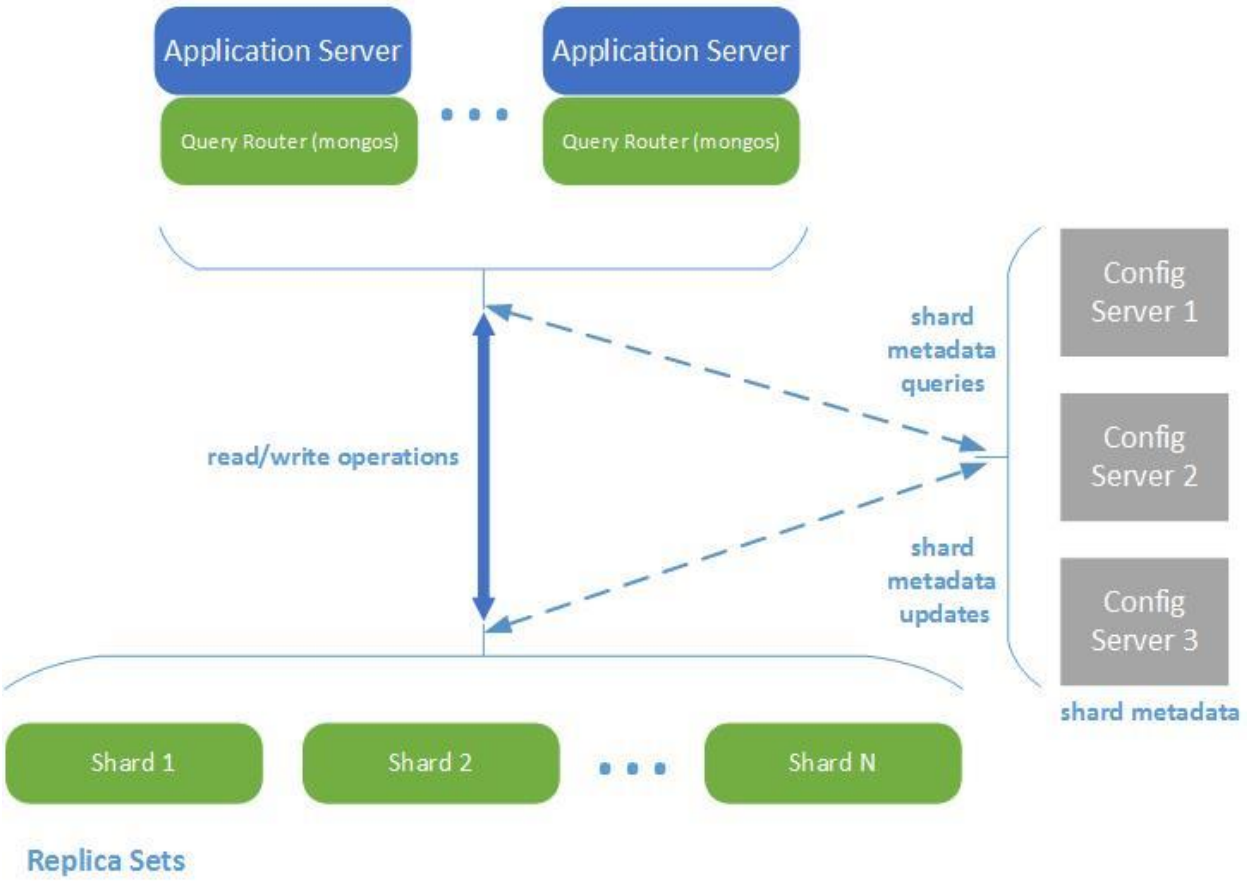


Figure 13: MongoDB Sharded Database Configuration

In a production environment, the following components are required for a sharded configuration:

- A shard, consisting of a single replica set, to provide high availability and resilience. We require a minimum of two such shards for horizontal scaling. Note that data sharding operates at the collection level. A `mongos` process, also termed a query router, enables data access. Accessing the shard directly means you will see only that shard's fraction of the cluster's data.
- Query routers, or `mongos` instances, which provide the sole interface to a sharded cluster for client applications. Such routers direct operations to the appropriate shard (or shards) and then return results to the clients. Sharded clusters in production environments should have a minimum of three query routers. However, they consume minimal system resources and have no persistent state. `mongos` instances are typically co-located with the application servers.
- Configuration servers are special `mongod` instances that store the sharded cluster's metadata. This data contains a mapping of the cluster's data set to the shards. They rely on a two-phase commit protocol to ensure strict consistency. The query router uses this metadata to target operations to specific shards. Production sharded clusters have exactly three configuration servers. These must be available at all times to deploy a sharded cluster or make updates to the cluster metadata.

3.4 Running MongoDB on Nutanix

Nutanix enables MongoDB to run elastic workloads on a highly scalable converged infrastructure.

- **Modular incremental scale**: With the Nutanix solution you can start small and scale both vertically and horizontally. A single Nutanix block provides up to 20 TB storage and 80+ cores in a compact 2U footprint. Given the modularity of the solution, you can scale per-node (up to ~5 TB/20+ cores), per block (up to ~20 TB/80+ cores), or with multiple blocks, giving you the ability to accurately match supply with demand and minimize upfront CapEx.
    - **Vertical scale**: increase RAM and CPU in a rolling fashion as workload increases; SSD-backed hot tier provides improved I/O response for all workload eventualities; easily upgrade to new hardware by adding nodes to a running cluster and then hot migrate VMs.
    - **Horizontal scale**: the ability to add a single block holding a replica set (or shard) creates several advantages. Nutanix supports MongoDB at elastic scale; can grow the database in a continuously highly available fashion; allows Operations to add capacity without the need for developer involvement or knowledge of application dependencies; distributes load and resource share across the cluster.
- **High performance**: Up to 100,000 plus random read IOPS and up to four GBs of sequential throughput in a compact 2U four-node cluster that scales linearly as new nodes are added.
- **Effective transparent data tiering:** Nutanix incorporates heat-optimized or activity-based tiering, which uses multiple storage tiers and places data on the tier that provides the best performance. The architecture supports local disks attached to the Controller Virtual Machine (SSD, HDD) as well as cloud-based source targets. The tiering logic is fully extensible, allowing new tiers to be added dynamically and extended. The Nutanix solution continuously monitors data access patterns to determine whether access is random, sequential, or mixed. An SSD tier maintains random I/O workloads to minimize latencies. Sequential workloads can be placed into HDD automatically to improve SSD endurance.
- **New compute and storage:** The Prism management interface makes it simple to increase capacity and the Prism Central tool allows you to easily manage multiple different cluster systems, eliminating the need for multiple sessions.
- **Enterprise-grade cluster management**: A simplified and intuitive approach to managing large clusters, including alert notifications, a converged GUI that serves as a single pane of glass for servers and storage, and a bonjour mechanism to auto-detect new nodes in the cluster. Spend more time enhancing your environment, not maintaining it.
- **High-density architecture**: The Nutanix advanced server architecture integrates eight Intel CPUs (potentially more than 80 cores) and up to 4 TB of memory into a single 2U appliance.
- **Business continuity and data protection**: MongoDB and its associated infrastructure VMs are mission critical and need enterprise-grade data management features, including backup and DR. With Nutanix, these are provided out-of-the-box

and can be managed the same as they would be for virtual environments. Nutanix also enables hybrid cloud architectures by integrating the public cloud (for example, Amazon Web Services) with Nutanix-powered private cloud environments.

- Data efficiency: Nutanix compression policies are truly VM-centric. Unlike traditional solutions that perform compression mainly at the LUN level, Nutanix provides all of these capabilities at the VM and file level, greatly increasing efficiency and simplicity. These capabilities ensure the highest possible compression and decompression performance on a sub-block level. TRIM support also provides data reclamation capabilities for deleted data.

3.5 Nutanix—Data Transforms

NDFS includes several mechanisms to optimize storage utilization and capacity savings. We discuss the most relevant ones below.

Compression

Nutanix provides both inline and post-process forms of compression—the best option depends on the data and workload type. Inline compression for large I/O size or sequential data streams will compress the data in-memory prior to it being written out to disk. Inline compression on random data is handled slightly differently. The data is written uncompressed to the Oplog. It is then coalesced and compressed in memory when subsequently written out to the extent store. Post-process compression sees the data written uncompressed to disk before the Nutanix Map Reduce framework compresses the data cluster wide. Both Nutanix and MongoDB (by way of the WiredTiger storage engine) implement the Google Snappy compression algorithm. This algorithm gives extremely good compression and decompression ratios with minimal computational overhead. Administrators should decide on what form of compression to use and only enable it once, either at the container level or within the application. We recommend Nutanix inline compression with read intensive workloads only when using MongoDB's MMAP storage engine. If enabling compression at the MongoDB layer, only the MongoDB database files will be compressed. Enabling compression at the Nutanix container level ensures all files resident on the container are compressed.

Erasure Coding (EC-X)

As cluster membership sizes increase, we can increase the replication factor from two to three to handle additional failures. This increase, of course, reduces useable disk space by creating additional redundant copies of the data. Nutanix addresses this with a feature called Erasure Coding-X (EC-X), which increases useable disk space while maintaining the same cluster resiliency. It does so by striping individual data blocks and associated parity blocks across nodes rather than disks (forming an erasure strip). In the event of a failure, the parity block along with the remaining blocks in the erasure strip are used to recalculate the missing data onto a new node. All blocks associated with erasure coding strips are stored on separate nodes. Each node can then take part in subsequent rebuilds, which reduces potential rebuild time. Erasure coding works best on cold data, archives, and backups. EC-X is suited to read-intensive MongoDB workloads (for example, performing database lookups or analytic queries). Containers with applications that incur numerous

overwrites, such as log file analysis or sensor data, require a longer delay than the one hour EC-X post-processing default. This ensures that only truly cold data is erasure coded. You can calculate the optimal `erasure-code-delay` value by running the following command from the CVM:

```
$ curator_cli get_egroup_access_info
```

The command generates output similar to what is below:

```
Container Id: 1025
+-----------------------------------------------------------------------------------------------------------------------------+
| Access \ Modify (secs) | 0-300 | 300-3600 | 3600-86400 | 86400-604800 | 604800-2592000 | 2592000-15552000 | 15552000-inf |
+-----------------------------------------------------------------------------------------------------------------------------+
| 0-300                  | 11    | 3        | 4          | 0            | 0              | 0                | 0            |
| 300-3600               | 12    | 62       | 3          | 1            | 2              | 0                | 0            |
| 3600-86400             | 0     | 35       | 97124      | 5390         | 40             | 0                | 0            |
| 86400-604800           | 0     | 9        | 55         | 189396       | 27226          | 0                | 0            |
| 604800-2592000         | 0     | 0        | 0          | 0            | 72306          | 0                | 0            |
| 2592000-15552000       | 0     | 0        | 0          | 0            | 0              | 0                | 0            |
| 15552000-inf           | 0     | 0        | 0          | 0            | 0              | 0                | 0            |
+-----------------------------------------------------------------------------------------------------------------------------+
```

In this sample output we can see that 72,306 egroups of data were last accessed or updated between 604,800 and 2,592,000 seconds ago—thus the data was last read or overwritten up to or greater than a week ago. Similarly, 189,396 extent groups of data were last read over a day (86,400s) ago. Accordingly, to set the EC delay to one week, either update the container parameters in Nutanix Prism or run the following from a CVM:

```
ncli ctr edit name=DEFAULT-CTR erasure-code-delay=604800
```

The table below shows the Nutanix storage pool and container configuration.

Table 1: Nutanix Storage Configuration

Name	Role	Details
SP01	Main storage pool for all data	All Disks
DEFAULT-CTR	Container for all MongoDB VMs	Acropolis Datastore Post Process Compression enabled (6 hour delay) Erasure Coding + 604,800s (1 week) Delay
ImageStore	Container for VM ISO images	Acropolis Datastore

3.6 Nutanix—Snapshots and Cloning

NDFS provides native support for offloaded snapshots and clones, which can be employed through a variety of means, including ncli, REST, and Prism. Both the snapshots and clones use the redirect-on-write algorithm, which is the most effective and efficient. Recall that for Nutanix, a virtual machine consists of files that—on the Nutanix platform—we call vDisks. A vDisk is composed of extents that are *logically* contiguous chunks of data stored within extent groups that 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. Traditional approaches require traversal of the snapshot chain, which can add read latency, and thus 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.

The same methods are used for both snapshots and clones of VMs and vDisks. 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. 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 or updates would take place on their individual block maps.

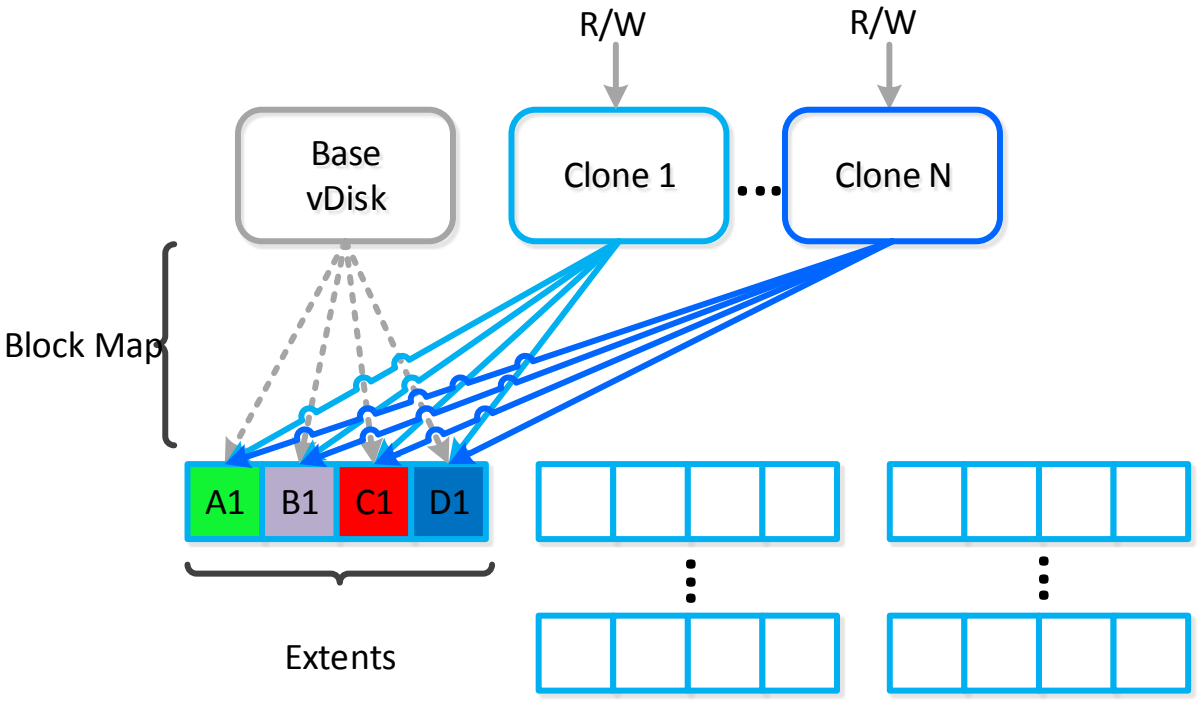


Figure 14: NDFS Clone(s) from Base vDisk

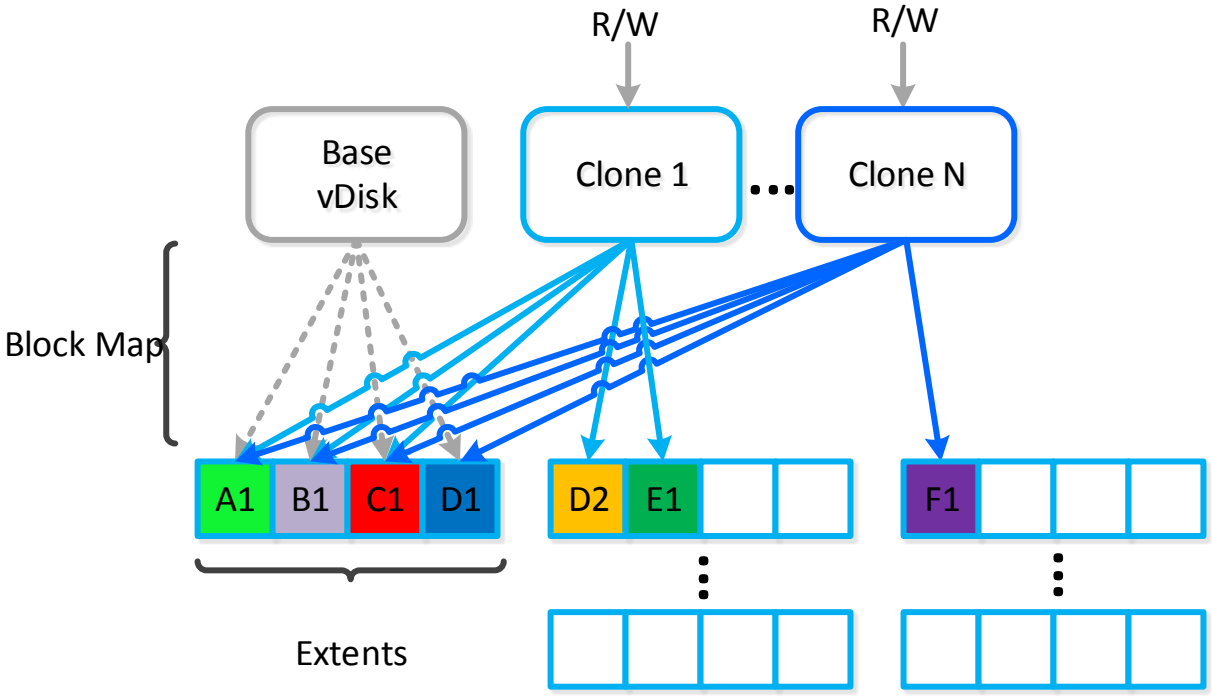


Figure 15: NDFS Clone(s) with Rewritten Base vDisk(s)

3.6.1 Nutanix—Snapshot for Backups and DR

MongoDB has a highly distributed architecture. Without Nutanix, true point-in-time snapshots can be difficult to achieve in a large, multi-shard configuration, particularly when

journal files are stored on a separate device. MongoDB outlines and supports a number of backup strategies. One strategy is to backup underlying files using snapshots from OS-specific volume management software. For smaller deployments, you can use the supplied command line utilities to invoke mongodump and mongorestore. These can, respectively, create or upload a BSON file representation of the documents held in the database.

The following applies to the default MongoDB storage engine (MMAPv1) only. For backup scenarios where journal and data files are stored on separate storage devices, writes to the database must be stopped to enforce a consistent snapshot of the database at that time. To flush all pending writes to disk and lock the database, we issue the db.fsyncLock() on a secondary replica. To unlock the database after the backup has been taken, we use the db.fsyncUnlock() command on the same replica. Bear in mind that these commands have to be completed on the same database connection.

The backup and DR functionality of the Nutanix platform is highly VM-centric. This means that snapshots are taken at a per-VM level. By following the MongoDB-defined pre-backup steps, a point-in-time backup can be taken of a VM group using the Nutanix VM snapshot feature. Such backups can then be used to roll out QA, test and development, or disaster recovery environments. For ease of migration to the public cloud, the Nutanix platform allows additional backup to remote sites located on AWS (Amazon Web Services) Elastic Compute Cloud servers.

Another best practice is to use a "hidden" replica to perform any dedicated task like backups or reporting. A hidden replica is not visible to the client application and is ineligible to become a set primary. It does, however, take part in set membership elections. Consequently, additional replica set members are required to maintain an odd number of members to avoid tied elections. The advantage of such a hidden replica in this instance is that the database lock and unlock functions are no longer required.

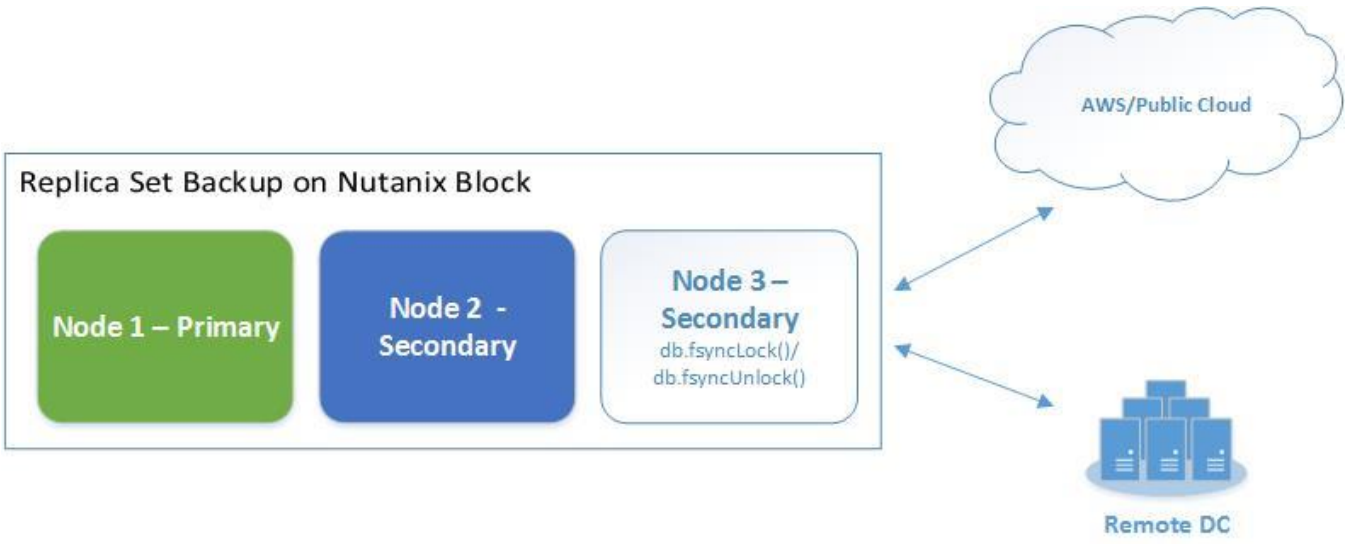


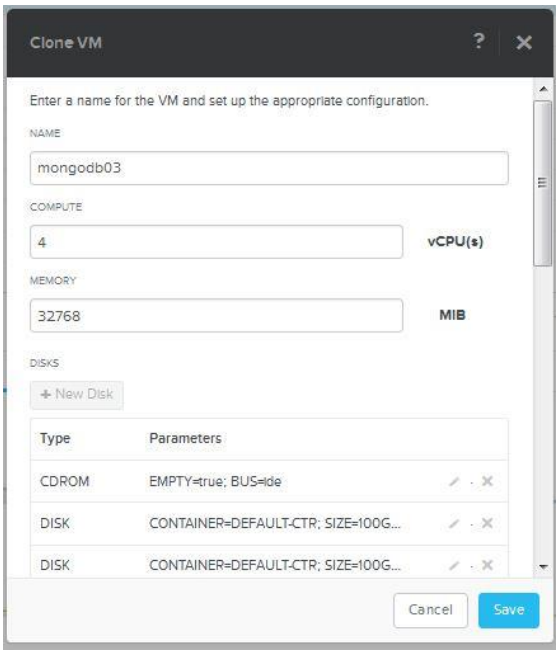
Figure 16: Nutanix Asynchronous DR—Backup to Cloud or Remote Site

3.6.2 Nutanix—Cloning Virtual Machines

Nutanix virtual machine cloning capabilities allow for faster MongoDB deployments. Consider a gold image or "template" VM that is preinstalled with the required revision of the

MongoDB software, configured using best practices. Such template VMs can then operate as an assembly line for production-ready deployments. Any member of a replica set has the potential to be a primary node. Ensuring that all set members are pre-built to the same standards means, for example, no unexpected difference in performance or behavior after failover events or set member redistribution across the NDFS datastore.

A MongoDB VM can be cloned via the GUI in a single operation. Such capabilities are key to supporting DevOps and Agile process workflows. Administrators can create an entire replica set in a matter of minutes using only the Nutanix CLI. The REST API can reduce this time to seconds. There is no need for such time-consuming hardware operations as SAN zoning or multi-pathing.



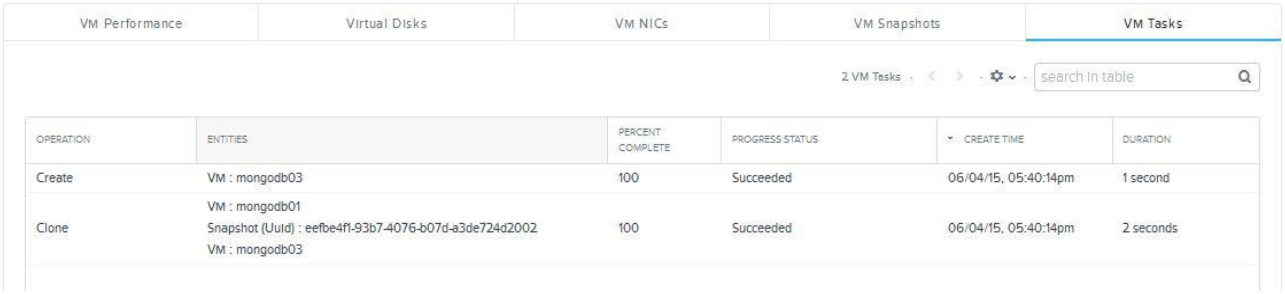


Figure 17: VM Clone Operation and Associated Tasks

4 Solution Design

Running MongoDB on Nutanix gives you the flexibility to start small—with a minimum of three nodes—and then either scale up or scale out a node, a block, or multiple blocks at a time. With Nutanix you can grow to massive scale without any impact on performance. Make sure, however, to apply the appropriate scaling. You may need to consider vertical scale before horizontal.

Table 2: Platform Design Decisions

Item	Detail	Rationale
General		
Minimum Size	3 x Nutanix nodes (3 Acropolis hosts)	• Minimum size requirement for (HA) replica set deployment
Scale Approach	Incremental modular scale	• Allow for growth from PoC to massive scale
Scale Unit	Nodes(s), Block(s)	• Granular scale to meet the precise capacity demands • Scale in n x node increments
Infrastructure Services	• Small deployments: Shared cluster • Large deployments: Dedicated cluster (Node A from 3 blocks or an NX-1350)	• Dedicated infrastructure cluster for larger deployments (best practice)
Nutanix		
Cluster Size	Up to 24-48 Acropolis hosts	Isolated fault domains
Storage Pool(s)	1 x Storage Pool per cluster	• Standard practice • ILM handles tiering
Container(s)	• 1 x Container for VMs • 1 x Container for ISO image store	Standard practice
Features/Enhancements	Increase CVM memory to 32 GB+	Best practice

Table 3: MongoDB and Infrastructure Design Decisions

Item	Detail	Rationale
MongoDB Servers		
mongod server VMs Primary or secondary	• Min: 3 (n+1) per replica set • Scale: 1 replica set per block /shard	HA for mongod servers
Server Sizing	• vCPU: 2 to 8 • Memory: 16 to 64 GB (application working set size dependent) • Disk: 80 GB (OS) + 6x nGB (Data) + 20GB (Journal) + 100GB (Logs)	• Standard sizing practice • Multiple disks/ raid sets for Data, Journal files and Logs
Data Protection	• Replica Sets • Geographically dispersed secondaries	Ensures availability of mongod servers
Infrastructure - Query Routers		
mongos VMs	Min : 3 per cluster Scale: Application dependent	HA for mongos servers
Server Sizing	• vCPU: 1-4 • Memory: 4 - 16GB	• Memory based on size of shard metadata cache • Allow ~1 MB per DB connection
Data Protection	Require additional VMs based on application growth/requirement	Ensures availability of mongos VMs
Infrastructure - Configuration Servers		
mongod (config server) VMs	3 per cluster	HA for mongod/config servers
Server Sizing	• vCPU: 1-4 • Memory: 4-16GB	Memory based on amount of metadata to be stored

	(application dependent)	
Data Protection	2-phase commit between 3 VM quorum	3 VM quorum per single MongoDB cluster only

Table 4: Network Design Decisions

Item	Detail	Rationale
Virtual Switches		
brNutanix	• Use: Acropolis host to CVM communication • Uplinks: N/A	Nutanix default
br0	• Use: All external Acropolis Management traffic • Uplinks: bond0	Standard practice
bond0	• NIC(s): 2 x 10GB	Utilize both 10 Gb adapters active/active
VLANS		

Production VLAN	• Network Site: Production
	• ID: Varies
	• Capability: Routable
	• Components:
	  ○ Acropolis Hosts
	  ○ Nutanix CVMs
	  ○ MongoDB Servers
	  ○ Query Routers
	  ○ Configuration Servers
		• Dedicated Infrastructure VLAN
		• Best Practice

5 Network

Designed for true linear scaling, Nutanix recommends a leaf-spine network architecture. A leaf-spine architecture consists of two network tiers: an L2 leaf and an L3 spine based on 40 GbE and non-blocking switches. This architecture maintains consistent performance without any throughput reduction due to a static maximum of three hops from any node in the network.

The figure below shows a design of a scale-out leaf-spine network architecture that provides 20 Gb active throughput from each node to its L2 leaf and scalable 80 Gb active throughput from each leaf-to-spine switch, providing scale from one Nutanix block to thousands without any impact to available bandwidth.

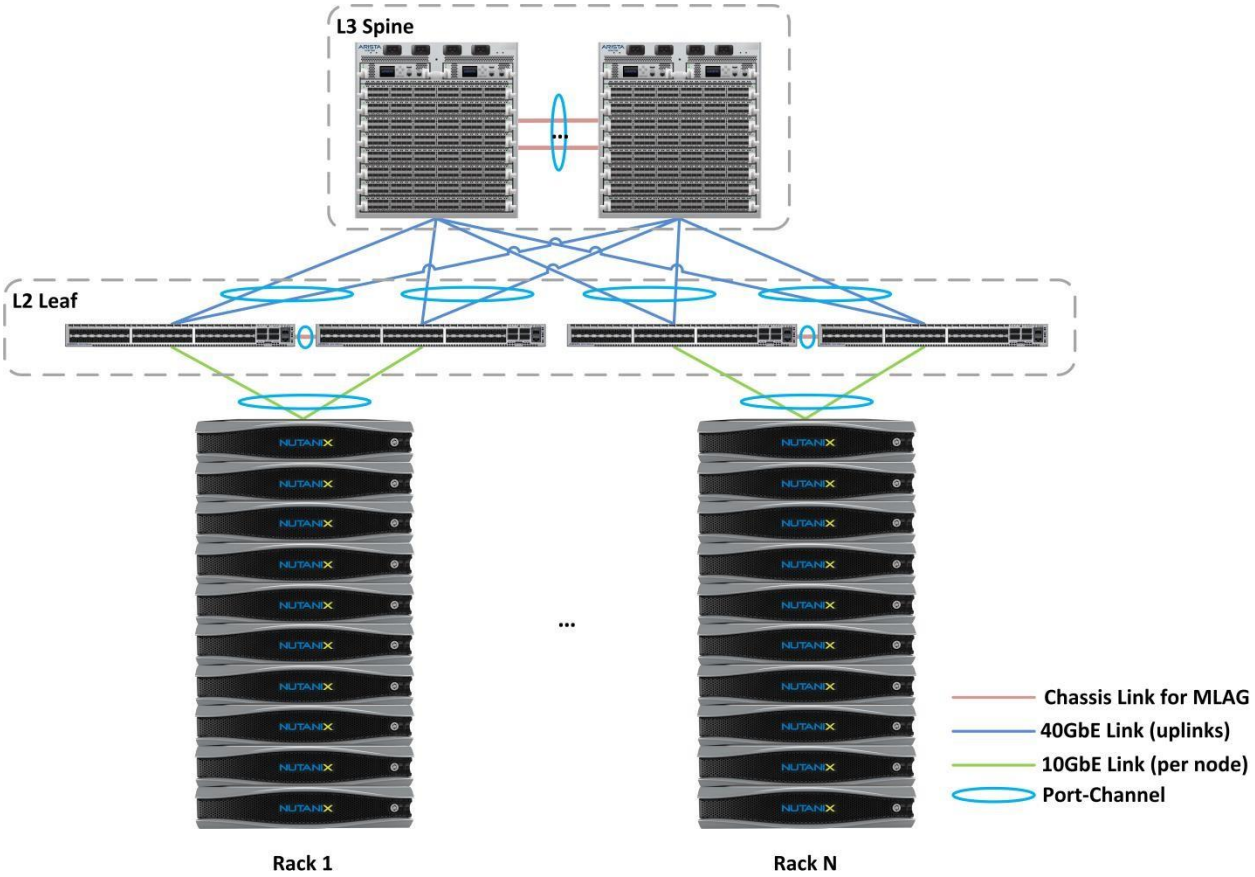


Figure 18: Leaf-Spine Network Architecture

Logical Network Design

Each Acropolis host has at least two default switches. One for internal communication and additional switches for external communication. The br0 vwitch is utilized for external node communication and VM traffic and has 10 GbE uplinks in a bond. The brNutanix switch is utilized for iSCSI and NFS I/O between the Acropolis host and the Nutanix CVM.

Following is a logical network representation of the network segments used in the solution with the corresponding components attached.

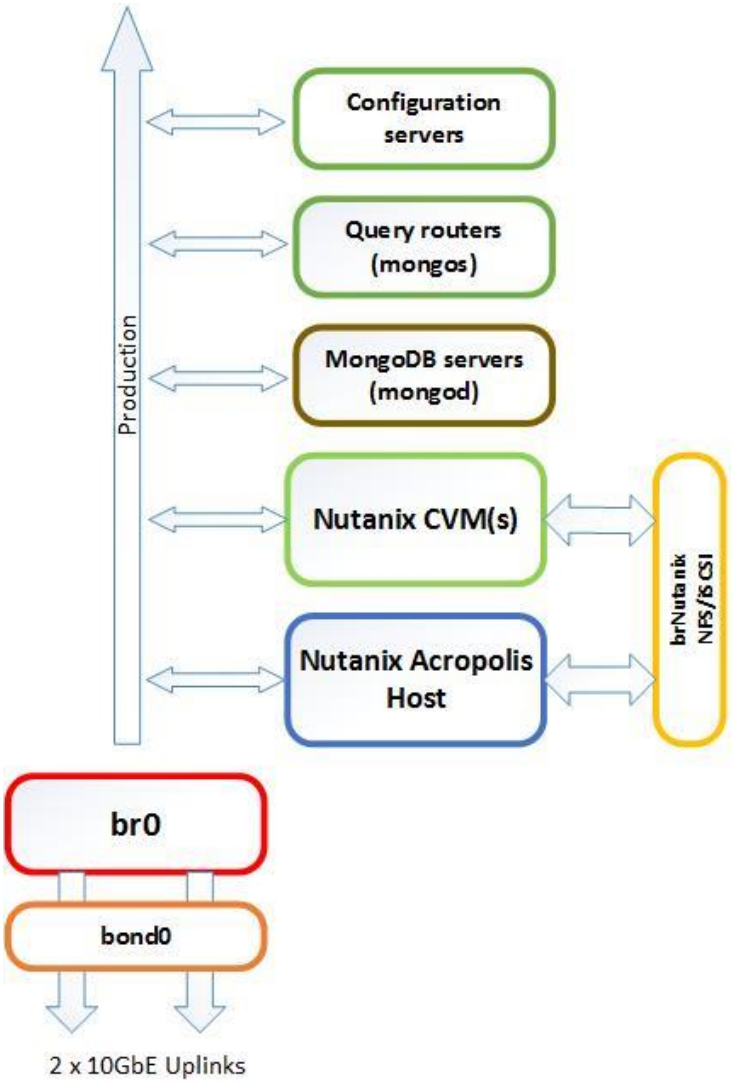


Figure 19: Network Logical Design

6 Site Planning and Solution Application

For production-scale deployments, it is important to use a topology that is distributed and meets end-user requirements, while also providing flexibility and locality.

To this end, we recommend that MongoDB on Nutanix employ a geographically distributed replica set. Such a set provides data recovery if one datacenter fails. These sets include at least one member in a secondary datacenter. The member is configured to prevent that member from ever becoming primary (by setting its priority to 0).

In many circumstances, these deployments consist of the following:

- One primary in the first (that is, primary) datacenter.
- A number of secondary members in the primary datacenter. These members can become the primary member at any time.
- One secondary member in a secondary datacenter. This member is ineligible to become primary, as its priority is set to 0.

If the primary is unavailable, the replica set will elect a new primary from the primary datacenter. In the instance that the connection between the primary and secondary datacenters fails, the member in the secondary center cannot independently become the primary. If the primary datacenter fails, you can manually recover the data set from the secondary datacenter.

While such configurations are recommended for larger production or more geographically distributed deployments, it's not required for smaller deployments (see Figure 20 below). Using the replica set as the unit of deployment, you can configure a single replica set on a Nutanix block. When you need to scale horizontally, add more hardware to configure more replica sets (shards) on the newly added nodes (we illustrate this in the following deployment examples).

It should be noted that, whether scaling vertically or horizontally, additional secondary members can be easily added to a replica set in different global locations to localize read access to the application end users. Such deployments are usually more than sufficient, especially if the database working set is never likely to outgrow available RAM. Indeed, for read-intensive applications, we recommend that you size VM RAM to hold the entire working set in memory, if possible, and then replicate for greater availability.

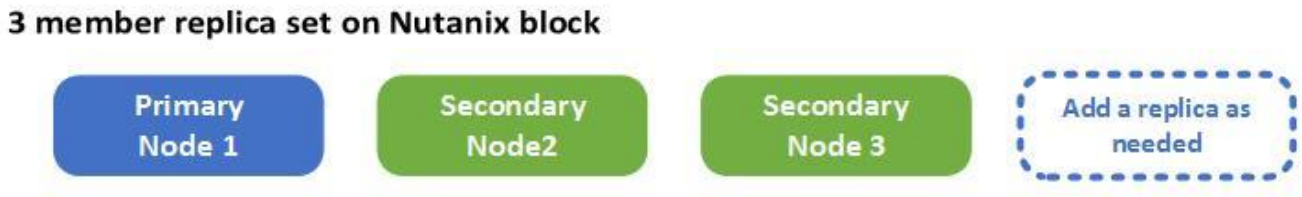


Figure 20: Small MongoDB Deployment—Additional Replicas for High Availability

For larger production deployments, we provide two examples:

- A sharded configuration with multiple replica sets.

- A sharded configuration with multiple, geographically distributed replica sets.

These sharded configuration examples require additional virtual machines (see Figures 21 and 22 below) as part of the MongoDB "infrastructure." Bear in mind the sizing of any mongos server will naturally depend on the amount of data, the number of shards, and the collections they contain.

In Tables 5 and 6 below, for each of the production examples we provide some initial recommendations for MongoDB replica and infrastructure VMs. These are based upon assumed workload characterizations. We break out the requirements for the MongoDB database processes: those that will handle application reads and writes, and the infrastructure type VMs, which are needed to allow the configuration to scale and maintain high availability. We make sure that all secondary nodes share the same specifications as a primary node in case they become primary in the event of a failover. We also want to reduce replication lag between the primary and secondary set members.

In our first sharded database example we show three member replica sets and the requisite number of query routers and configuration servers suitable for production. The number of shards can be increased as part of any application requirement. We can subsequently scale out further horizontally by adding an additional three node block for every replica set/shard.

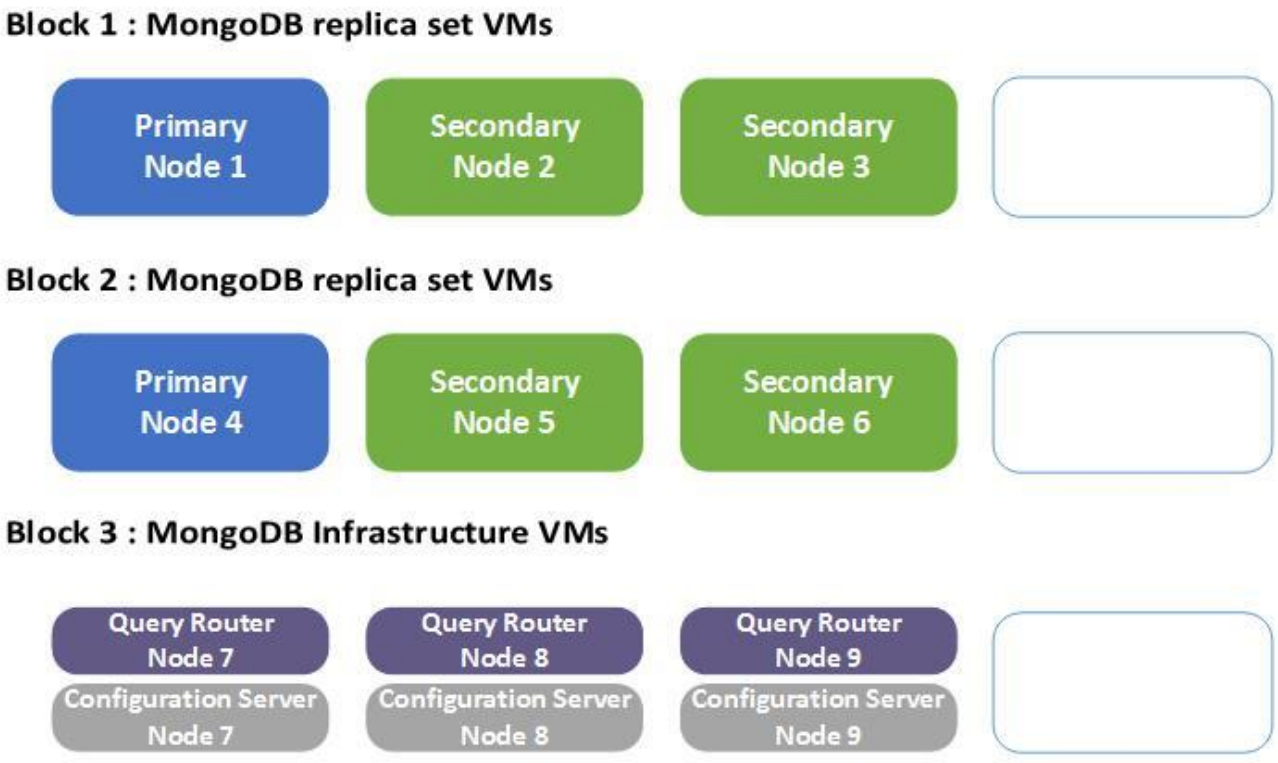


Figure 21: Sharded Production Deployment Using Three Member Replica Sets

In Figure 21, we have chosen to co-locate the query router and configuration servers on the same hypervisor hosts. While either of these VMs can reside on the same hypervisor host as any replica member, keep in mind the required recovery workflows if two components fail at the same time.

Table 5: MongoDB and Associated Infrastructure VMs—Initial Sizing of Production Horizontal Scale Deployments (3 Member Sets)

Small	Per VM			Total	
MongoDB Component	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
`mongod` primaries	2	2	16	4	32
`mongod` secondaries	4	2	16	8	64
Query Router (`mongos`)	3	1	4	3	12
Configuration Server (`mongod`)	3	1	4	3	12
			Totals	18 vCPUs	120 GB
Medium				**Total**	
MongoDB Component	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
`mongod` primaries	2	4	32	8	64
`mongod` secondaries	4	4	32	16	128
Query Router (`mongos`)	3	2	8	6	24
Configuration Servers (`mongod`)	3	2	8	6	24
			Totals	36 CPUs	240 GB
Large				**Total**	
MongoDB Conmponent	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
`mongod` primaries	2	8	64	16	128
`mongod` secondaries	4	8	64	32	256
Query Routers (`mongos`)	3	4	16	12	48
Configuration Servers (`mongod`)	3	4	16	12	48
			Totals	72 vCPUs	480 GB

Note: These are recommendations for sizing and should be modified after a current state analysis.

For our second sharded configuration example (see Figure 22), we have increased the membership of the replica sets to five. The first addition is that of a "hidden" replica specifically for backup and reporting. While this hidden replica can't become a primary, it can participate in membership elections. However, this causes set membership to be an even number of nodes. To avoid tied elections, we add a fifth replica to maintain an odd set membership. We added this fifth set member remotely and used it for replica set recovery after a primary site outage, as discussed at the start of this chapter.

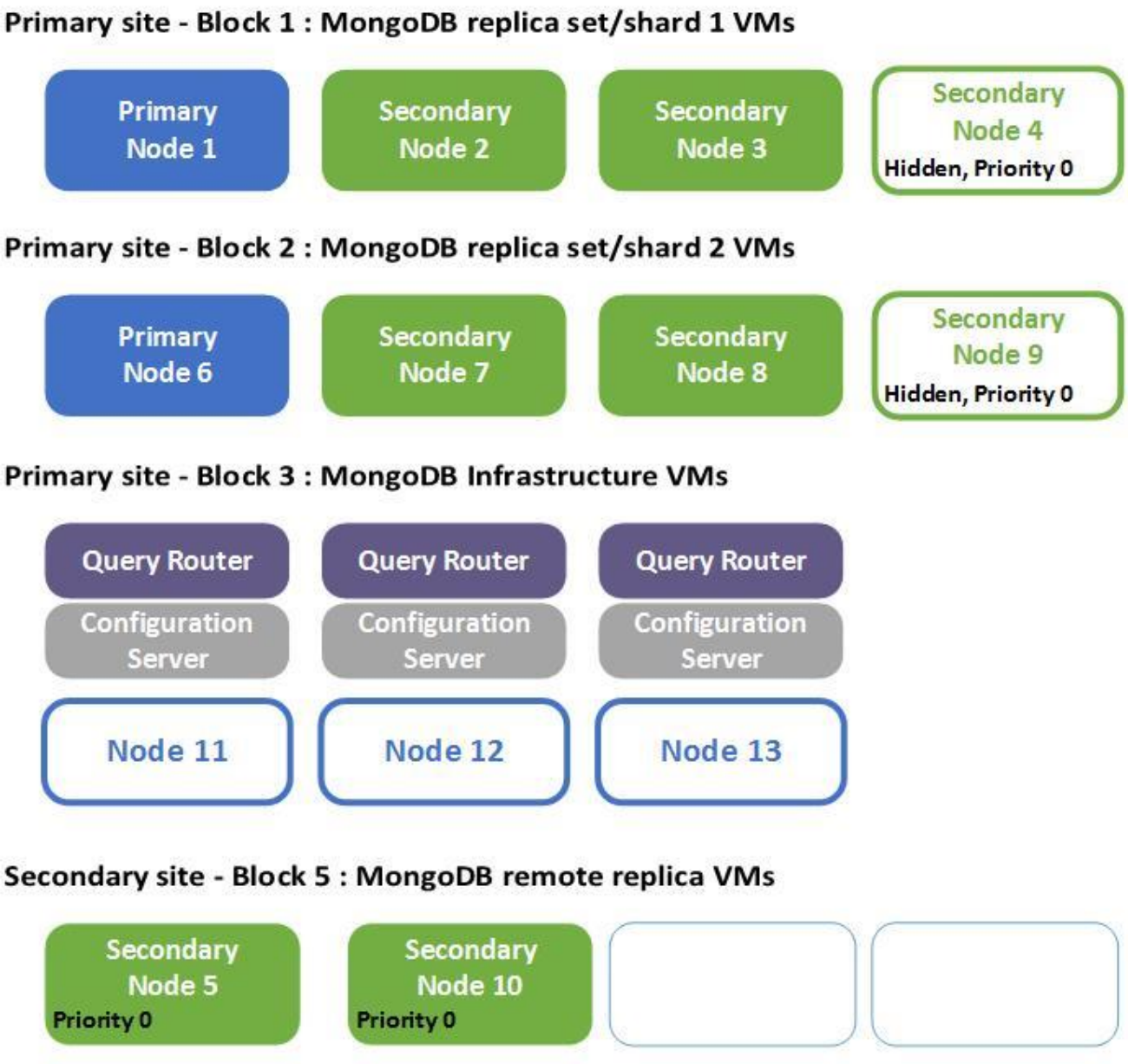


Figure 22: Production MongoDB Deployment Across Two Sites

Table 6: MongoDB and Associated Infrastructure VMs—Initial Sizing of Production Horizontal Scale Deployments (5 Member Sets)

Small	Per VM			Total	
MongoDB Component	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
`mongod` primaries	2	2	16	4	32
`mongod` secondaries	8	2	16	16	128
Query Router (`mongos`)	3	1	4	3	12

Configuration Server (mongod)	3	1	4	3	12
			Totals	26 vCPUs	184 GB
Medium				Total	
MongoDB Component	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
mongod primaries	2	4	32	8	64
mongod secondaries	8	4	32	32	256
Query Router (mongos)	3	2	8	6	24
Configuration Servers (mongod)	3	2	8	6	24
			Totals	52 vCPUs	368 GB
Large				Total	
MongoDB Conmponent	# of VMs	vCPUs	RAM (GB)	vCPUs	RAM (GB)
mongod primaries	2	8	64	16	128
mongod secondarioes	8	8	64	64	512
Query Routers (mongos)	3	4	16	12	48
Configuration Servers(mongod)	3	4	16	12	48
			Totals	104 vCPUs	736 GB

Note: These are recommendations for sizing and should be modified after a current state analysis.

As the production environment deployments grow we can add additional Nutanix compute and storage nodes incrementally. How we scale is entirely dependent on how the application grows. We can add Nutanix "storage heavy" nodes to provide additional storage only to the global NDFS, or we can add multiple node blocks to deploy additional shards (replica sets). Alternatively, if the database is already sufficiently sharded, then, depending on use case, we may only need to add more geographically dispersed replicas to the individual replica sets.

A sample use case for this might be an application where you have to manage separate databases around the world and their associated datasets that regularly have to batch update. Instead, you have a single MongoDB database (possibly sharded, if required) with replicas distributed globally.

A particularly useful workflow in all of the configurations described is that of hardware refresh. For example, we might want to take advantage of the improved vertical scaling capabilities of new hardware. By adding additional Nutanix blocks to the cluster, MongoDB VMs can be seamlessly migrated to the new hardware without experiencing any downtime.

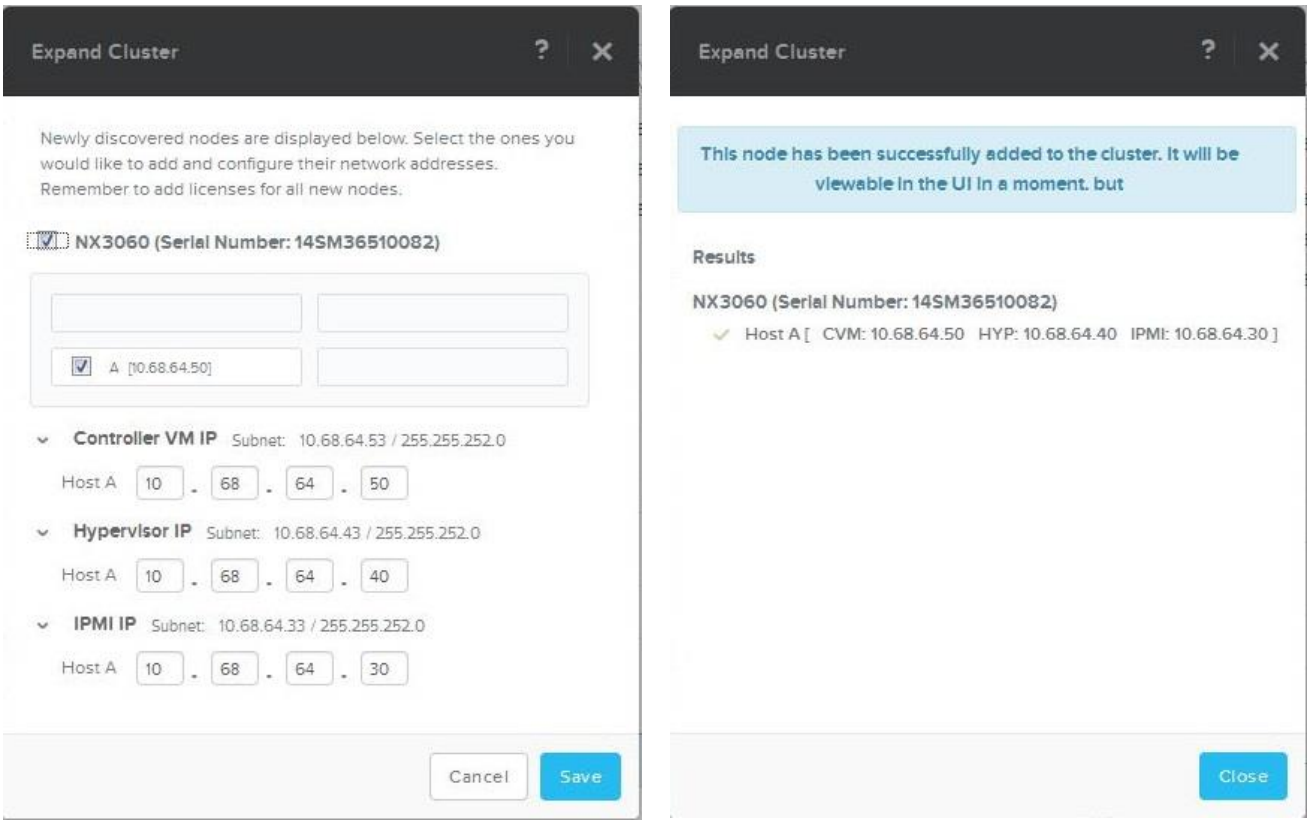


Figure 23: Screenshots Showing Auto-Discovery and Cluster Node Addition

One final example is that of a test and development environment whose setup (Figure 24) shows a sharded configuration contained entirely within a single Nutanix block. Note that there are no replica sets configured, just standalone mongod instances. Similarly, there is no additional availability configured for either the query router (mongos) or the configuration server. This configuration contains a single mongos and configuration server instance. In order to create additional data shards, a new node can be added to host an additional standalone mongod instance. However, in order to support high availability of the architecture and both read and write performance scalability, sharded systems should be built on top of replicated systems when it comes to production.

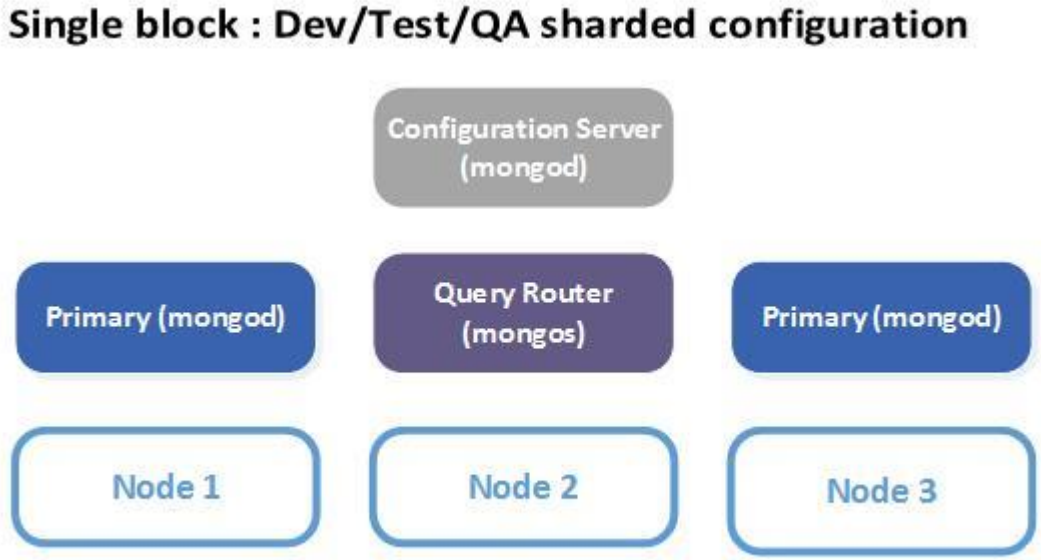


Figure 24: MongoDB Dev/Test/QA Configuration

7 Best Practices

The MongoDB on Nutanix best practices can be summarized into the following high-level items.

NOTE: the majority of best practice configuration and optimization occurs at the MongoDB and Linux level.

7.1 General

- Perform a current state analysis to identify workloads and sizing are correct. For example:
    - Define working set.
    - Establish database query efficiency.
    - Determine rate of data change.
    - Identify the peaks and troughs.
- Ensure that you architect a solution that meets both current and future needs in terms of available RAM and I/O throughput.
- Design to deliver consistent performance, reliability, and scale. Ensure additional process memory is available to the workload at all times and monitor for storage bottlenecks. For example, monitor Page Faults (PF) as a function of MongoDB operations/sec (OPS).

  If PF/OPS is

  - near 0—reads rarely require disk I/O.
  - near 1—reads regularly require disk I/O.
  - greater than 1—reads require heavy disk I/O.
- Scaling under load is not recommended. Make any decisions to the scale out long before it's necessary. MongoDB needs significant compute, memory, and networking resource to function, hence trying to perform additional sharding operations while the system is running at near peak load is non-optimal.
- Test, optimize, iterate, and scale your database prior to going into production. Starting points are:
    - Uptime requirements—HA and replication?
    - Throughput in terms of reads, writes, and users—include peak values.
    - Required operations/second.
    - Required responsiveness in terms of such conditions as latency, SLAs, TTFB trends, etc.

7.2 MongoDB

- Use 64 bit MongoDB builds for production. 32 bit builds are for test and development environments only.
- Ensure journaling enabled by default—this ensures MongoDB data files are recoverable after a crash.
- Ensure data file pre-allocation enabled by default—enables faster startup and prevents file system fragmentation.
- Ensure working set of pages and indexes fits in RAM—for current working set estimate prior to MongoDB release 3.0 use :

```
db.serverStatus({workingSet:1}).workingSet

"pagesInMemory" :  91521
```

Multiply working set pages by PAGESIZE to get size in bytes.

```
getconf PAGESIZE

4096
```

Get index size (in bytes).

```
db.stats().IndexSize

7131826688
```

In this example:

```
(915211 * 4096) + 7131826688 ~ 6GB
```

Post-3.0 release of MongoDB, the `workingSet` command option is no longer available. Use the mem section of the `serverStatus` output:

```
rs01:PRIMARY> db.serverStatus().mem

{

        "bits" : 64,

        "resident" : 20466,

        "virtual" : 148248,

        "supported" : true,

        "mapped" : 73725,

        "mappedWithJournal" : 147450

}
```

Where:

serverStatus.mem.virtual.

virtual displays the quantity, in megabytes (MB), of virtual memory used by the mongod process. With journaling enabled, the value of virtual is at least twice the value of mapped.

serverStatus.mem.mapped

The value of mapped provides the amount of mapped memory, in megabytes (MB), by the database. Because MongoDB uses memory-mapped files, this value is likely to be roughly equivalent to the total size of your database or databases.

serverStatus.mem.mappedWithJournal

mappedWithJournal provides the amount of mapped memory, in megabytes (MB), including the memory used for journaling. This value will always be twice the value of mapped. This field is only included if journaling is enabled.

- Ensure readahead size set to 32*512 sectors/16KB (as a good starting point) on block devices storing data files:
    - This avoids the database Resident Set Size (RSS) growing over time, caused by superfluous data being read into memory as part of any read. The eventual effect of this is that it pushes potentially required data out of memory to make room for potentially unwanted "read ahead" data.
    - To set `sudo blockdev --setra 32 /dev/sdx`.... or `/dev/dm-X` (if using LVM)
    - To verify `sudo blockdev --getra /dev/sdx` ...or `/dev/dm-x`

- Use EXT4 or XFS file systems
    - MongoDB preallocates its database files before using them and often creates large files. As such, you should use either the ext4 or XFS file system.
- Disable access time settings (noatime) on data files mount point:

```
/dev/mapper/mongojournal-mongojournal /mongodb/journal xfs
defaults,auto,noexec 0 0

/dev/mapper/mongodata-mongodata /mongodb/data xfs
defaults,auto,noatime,nodiratime,noexec 0 0

/dev/mapper/mongolog-mongolog /mongodb/log xfs defaults,auto,noexec 0 0
```

- Disable transparent hugepages in VMs that will host mongod database server processes. For example, edit startup scripts as follows:

```
#disable THP at boot time

if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then

        echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled

fi



if test -f /sys/kernel/mm/redhat_transparent_hugepage/defrag; then
```

```
        echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag

fi
```

Alternatively, add `transparent_hugepage=never` to the kernel command line (that is, in grub.conf).

- Set swappiness = 1: MongoDB is a memory-based database; if the nodes are sized correctly, then we won't need to swap. However, setting `swappiness=0` could cause unexpected invocations of the OOM (Out of Memory) killer in certain Linux distros.

```
sudo sysctl vm.swappiness=1 (for current runtime)

sudo echo 'vm.swappiness=1' >> /etc/sysctl.conf (make permanent)
```

- Disable NUMA
    - Either in VM BIOS or, invoke `mongod` with NUMA disabled. All versions of MongoDB ship with an init script that automates this as follows:

```
numactl –interleave=all /usr/bin/mongod –f /etc/mongod.conf
```

    - Also ensure:

```
$ sudo cat /proc/sys/vm/zone_reclaim_mode

0
```

- Use RAID10 for data volumes and place journal, log, and data files on separate storage devices.
    - All vDisks supplied by Nutanix are redundant by default; you only need to create stripe for data volumes.

```
/mongodb/data, /mongodb/journal, /mongodb/log
```

    - Create soft link to enable journal writes to separate device.

```
ln –s /mongodb/journal /mongodb/data/journal
```

    - Ensure correct ownership.

```
sudo chown –R mongod:mongod /mongodb/journal /mongodb/data /mongodb/log
```

- Ensure NTP time synchronization across all hosts.
    - Use NTP and a reliable time source to ensure consistent time between MongoDB nodes.
- Ensure ulimits set as follows:
    - –f (file size): unlimited
    - –t (cpu time): unlimited
    - –v (virtual memory): unlimited
    - –n (open files): above 20,000
    - –m (memory size): unlimited
    - –u (processes/threads): above 20,000
    - Set limit overrides in RHEL based distributions:

```
$ sudo vi /etc/security/limits.conf
* soft nofile 65536
* hard nofile 65536
* soft nproc 32768
* hard nproc 32768

$ sudo vi /etc/security/limits.d/90-nproc.conf
* soft nproc 32768
* hard nproc 32768
```

- Use a canonical name (CNAME) in DNS to identify configuration servers; name changes can mean a restart of all mongod and mongos servers.

7.3 Nutanix

- Use single RF=2 Container for MongoDB VMs.
- Enable compression on the container (if not already enabled in MongoDB storage engine).
- Enable Erasure Coding with a suitable delay to only process colder data
- Utilize appropriate model based upon compute and storage requirements.
- Ideally keep working set in SSD and keep database size for associated `mongod` process within node capacity.

NOTE: for larger databases that cannot fit on a node, ensure there is ample bandwidth between nodes.

- Utilize higher memory node models for MongoDB data workloads.
- Utilize a node that will be 2x memory size of largest single VM.
- Create a dedicated consistency group containing the MongoDB secondary for every shard or replica set and appropriate infrastructure VMs.

7.4 Network Infrastructure

To ensure prompt replication to secondary replica VMs:

- Use low-latency 10 GbE switches.
- Utilize active-active 10 GbE uplinks from each Nutanix node.

The following CVM convenience utilities exist to help manage Acropolis networking and are recommended:

- change_cvm_lan <tag>

Example:

```
$ change_cvm_vlan 64
```

- manage_ovs --help

  Example:

```
$ manage_ovs  --interfaces eth2,eth3 update_uplinks
```

8 Conclusion

The Nutanix Xtreme Computing Platform lets you virtualize MongoDB with the power and reliability of a highly performant underlying web-scale infrastructure. With one click, incremental addition of Nutanix compute and storage, you can easily scale as your database scales—from a three virtual machine replica set in a single block up to multi-node replica sets supporting global horizontal scale out MongoDB systems. You can also scale vertically by upgrading virtual or physical hardware (memory or CPU) with zero downtime.

Nutanix eliminates the need for complex NAS and SAN environments by delivering locally attached storage for MongoDB servers. A cluster-wide SSD-backed hot data tier maintains low I/O latency and response for running MongoDB, even in a demanding mixed workload environment. For maximum cold-tier efficiency, Nutanix provides inline compression coupled with post-process erasure coding.

Nutanix simplifies MongoDB management with Prism, streamlining database backups, rollout of test and QA environments, and seamless migration to the public cloud. Prism also provides cluster health overviews, full stack performance analytics, hardware and software alerting, storage utilization, and automated remote support.

The Nutanix Xtreme Computing Platform provides proven invisible infrastructure, allowing you to get the most out of critical applications like MongoDB while spending less time in the datacenter.

9 Appendix

9.1 Appendix A: Replication Lag

We define replication lag as follows:

```
rs01:PRIMARY> db.printSlaveReplicationInfo()
source: 192.168.1.32:27017
        syncedTo: Fri Feb 20 2015 17:24:06 GMT+0000 (GMT)
        0 secs (0 hrs) behind the primary
source: 192.168.1.27:27017
        syncedTo: Fri Feb 20 2015 17:24:06 GMT+0000 (GMT)
        0 secs (0 hrs) behind the primary
rs01:PRIMARY> db.printReplicationInfo()
configured oplog size:   30597.247314453125MB
log length start to end: 2982secs (0.83hrs)
oplog first event time:  Fri Feb 20 2015 16:34:24 GMT+0000 (GMT)
oplog last event time:   Fri Feb 20 2015 17:24:06 GMT+0000 (GMT)
now:                     Tue Feb 24 2015 16:34:02 GMT+0000 (GMT)
```

Replication lag is then: `Primary("oplog last event time") – Secondary("syncedTo")`

9.2 Appendix B: References

MongoDB - http://www.mongodb.com/

MongoDB Online Documentation - http://docs.mongodb.com/manual

MongoDB Whitepapers - http://docs.mongodb.com/white-papers

9.3 Table of Figures

Figure 1: Information Lifecycle Management 7

Figure 2: Overview of the Nutanix Architecture 8

Figure 3: Data Locality and vMotion 9

Figure 4: Elastic Deduplication Engine 9

Figure 5: The Nutanix Architecture with Acropolis 10

Figure 6: Acropolis and PCI Passthrough....11
Figure 7: Acropolis and VM Networking....11
Figure 8: Acropolis IPAM....12
Figure 9: MongodDB Replica Set—The Basic Unit of Sharding....13
Figure 10: Replica Set Heartbeating....15
Figure 11: Working Set Size Growth Over Time Compared to Physical RAM....16
Figure 12: MongoDB Scaling—Logical Overview....17
Figure 13: MongoDB Sharded Database Configuration....18
Figure 14: NDFS Clone(s) from Base vDisk....23
Figure 15: NDFS Clone(s) with Rewritten Base vDisk(s)....23
Figure 16: Nutanix Asynchronous DR—Backup to Cloud or Remote Site....24
Figure 17: VM Clone Operation and Associated Tasks....25
Figure 18: Leaf-Spine Network Architecture....29
Figure 19: Network Logical Design....30
Figure 20: Small MongoDB Deployment—Additional Replicas for High Availability....31
Figure 21: Sharded Production Deployment Using Three Member Replica Sets....32
Figure 22: Production MongoDB Deployment Across Two Sites....34
Figure 23: Screenshots Showing Auto-Discovery and Cluster Node Addition....36
Figure 24: MongoDB Dev/Test/QA Configuration....37

9.4 Table of Tables

Table 1: Nutanix Storage Configuration....21
Table 2: Platform Design Decisions....26
Table 3: MongoDB and Infrastructure Design Decisions....27
Table 4: Network Design Decisions....28
Table 5: MongoDB and Associated Infrastructure VMs—Initial Sizing of Production Horizontal Scale Deployments (3 Member Sets)....33

Table 6: MongoDB and Associated Infrastructure VMs—Initial Sizing of Production Horizontal Scale Deployments (5 Member Sets) 34

9.5 Appendix C: Meet the Author

Ray Hassan is a Solutions and Performance Engineer at Nutanix, Inc. His current focus is on emerging webscale technologies such as MongoDB. He designs, tests, and documents virtualized scale-out workloads on the Nutanix platform.

Ray has over 15 years of experience in clustering technology, from working escalations for global account customers, running both failover and geographically dispersed clusters on UNIX and Linux based platforms. He was the first internationally based Systems Reliability Engineer (SRE) and Nutanix employee #4 in EMEA.

Follow Ray on Twitter @cannybag


@nutanix