Delivering MongoDB-as-a-Service: Top 10 Considerations

A MongoDB Whitepaper
November 2017

mongoDB

Table of Contents

Introduction

With several hundred thousand production deployments and customers in more than 50% of Fortune 100 companies, MongoDB is the industry's fastest growing database. An increasing number of organizations are using MongoDB Enterprise Advanced to deliver a Database-as-a-Service (DBaaS), standardizing the way in which internal business units and project teams consume MongoDB, thereby improving:

- **Business Agility.** Making it simple to rapidly spin up new development environments that can be quickly migrated to production deployments when the project goes live
- **Operational Efficiency.** Re-using standard infrastructure, processes, tools, and best practices across multiple projects
- **Business Unit Accountability.** Billing project teams for the resources they consume
- **Corporate Governance.** Enforcing centralized controls for Quality of Service (QoS), security, disaster recovery, and more

Organizations such as a top investment bank, The Royal Bank of Scotland, and the US Department of Veteran Affairs use MongoDB as their Database-as-a-Service (DBaaS) platform. Building upon the success these and others have had, this whitepaper provides the top 10 considerations IT groups need to make in building their own MongoDB-as-a-Service, whether delivered from private clouds running in internal data centers or from any of the leading public cloud platforms.

This paper is focused on those organizations building their own private MongoDB-as-a-Service, but MongoDB also offers MongoDB Atlas as a ready built service. Some of the considerations covered in the paper, such as identifying common workloads for capacity planning purposes, and scaling with sharding also apply to MongoDB Atlas.

Step 1: Identify Common Workload Requirements

By engaging with project teams, both those running live applications and those planning for release within the next

	Current	Projected (12 months)
Database Size (GB, TB, PB)		
Average Document Size (KB, MB)		
Data Retention Period (Days, Months, Years)		
Write Operations per Day		
Query Operations per Day		
Query Profile (% of operations using primary key, secondary indexes, aggregations, MapReduce jobs)		
Average Number of Documents Returned per Query		

Table 1: Sizing Database Load

six months, the IT group can capture current and anticipated database usage, architecture design, and operational policies. This will ensure the IT group designs a shared service delivery infrastructure that will meet both the short and medium term needs of its internal customers. The process will also identify candidates for an initial pilot of the service before it is made generally available to project teams across the organization.

Key stakeholders for consultation in this stage include the following for each project:

- Business owners
- Architects
- Developers
- DBAs
- Operations staff
- Network and storage engineers
- Corporate security and compliance representatives

Database Usage

The first stage is to document current and projected MongoDB usage for each project. Key statistics to capture are shown in Table 1.

Architecture Design

A profile of the existing or planned infrastructure will help size platform requirements and cluster configurations. Key data to capture is shown in Table 2.

Operational Policies

The final stage of the discovery process is to capture requirements that dictate how the application is run in production, including:

- Performance and availability SLAs (Service Level Agreements)
- Provisioning, upgrade, and change control processes
- Data archive, backup, and restore policies
- Database management, and monitoring
- Security requirements (e.g., access control, encryption, and auditing)

Key Takeaways

While not exhaustive, the checklists above will help to profile MongoDB usage and inform a design that meets the immediate needs of internal customers. It is also important to remember that with its loosely-coupled, flexible architecture the IT group is not locked in to a rigid MongoDB design. It can be rapidly adapted and re-provisioned to meet new application requirements as they evolve in the future.

	Current	Projected (12 months)
MongoDB Version and Drivers Used		
Operating System & Version		
Physical Host or Instance Specification (Number of processors and cores, RAM)		
Internal Storage Specification (Number and Capacity of SSDs & HDDs, RAID Level)		
External Storage Specification (SAN, NAS, Provisioned Bandwidth). (Note, local storage is preferred. See "Hardware Selection" for more information)		
Number of MongoDB Instances per Physical Host		
Number of Replica Set Members		
Number of Shards		
Network Specification (MB/s, GB/s)		

Table 2: Infrastructure and Design

Step 2: Hardware & OS Selection

While the analysis from Step 1 can provide guidance based on the current hardware platforms in use, it is important to recognize that different applications can drive the selection of different hardware configurations. To achieve the efficiency benefits promised by a shared MongoDB-as-a-Service infrastructure, the IT group needs to define a standard set of reusable hardware building blocks that will satisfy the broadest set of performance and availability requirements across a range of applications.

Making hardware selection much simpler, MongoDB is specifically designed for commodity hardware and has few hardware requirements or limitations. MongoDB will generally take advantage of more RAM, faster CPU clock speeds, and local storage. MongoDB has extensive experience helping customers to select the appropriate hardware and tune their configurations. By building your Database-as-a-Service on MongoDB Enterprise Advanced, our consultants can work with your IT group to validate and optimize MongoDB systems.

RAM & CPU

MongoDB makes extensive use of RAM to increase performance. Ideally the database's working set (i.e. the "hot" subset of data and indexes that are accessed most frequently by the application) fit into RAM. As a general rule of thumb, the more RAM, the better the performance. Therefore, hardware budget should be prioritized towards memory-rich systems. RAM footprints of 128GB to 512GB will typically provide the best general purpose platform. If the working set will exceed available memory, then MongoDB can be automatically distributed (sharded) across multiple nodes. Sharding is discussed later in the Guide.

MongoDB will deliver better performance on faster CPUs. The MongoDB WiredTiger storage engine is better able to saturate multi-core processor resources than the MMAPv1 storage engine, as are the Encrypted and In-Memory storage engines (based on WiredTiger). Dual socket servers equipped with modern 64-bit Intel or AMD processors make great general purpose platforms.

Storage

MongoDB does not require shared storage (e.g. Storage Area Networks), and is instead optimized for locally attached storage. Data access patterns in MongoDB do not have sequential properties, and as a result applications may experience substantial performance gains by using SSDs, especially where workloads require random updates to very large working sets. While data files benefit from SSDs, MongoDB's journal files do exhibit high sequential write profiles and are therefore good candidates for fast local hard disk drives.

When planning storage provision, it is important to consider storage engine options. The WiredTiger and Encrypted storage engines provide several compression options, making them up to 80% more storage-efficient than the MMAPv1 storage engine.

Most MongoDB deployments should use RAID-10 storage. RAID-5 and RAID-6 do not provide sufficient performance. RAID-0 provides good write performance, but limited read performance and insufficient fault tolerance.

If shared storage is the only option available, it is recommended to use explicitly provisioned block storage, such as Amazon Web Services (AWS) Provisioned IOPS (PIOPS) or equivalent. This type of implementation provides a balance between decoupled, re-assignable storage and guaranteed throughput. Block storage shared by multiple applications lacks the assured Quality of Service (QoS) guarantees, which can impact performance. Given generally low random-access performance, shared NAS filesystem storage is not recommended for MongoDB deployments.

When evaluating deployment on a SAN, it is important to conduct thorough stress testing to characterize the IOPS needed to sustain required performance levels both now and in the future. In addition to provisioning dedicated IOPS, there are some other best practices that should be considered:

- Locate the MongoDB journal on a separate fast local drive.
- MongoDB data files should be provisioned to separate SAN spindles.
- Avoid over-subscription by isolating the MongoDB workload from others that share the same physical SAN and networking infrastructure.
- Without proper redundancy, SANs can present a single point of failure. If all members of a MongoDB replica set are co-located on the same SAN, ensure mechanisms exist for fast SAN recovery.

Operating System

MongoDB Enterprise Advanced is certified for multiple operating systems:

- Four Linux distributions: Red Hat Enterprise Linux, CentOS, Ubuntu, SuSE, and Amazon Linux
- Windows 7/Windows Server 2008 R2 or later
- macOS

In choosing an operating system, enterprise mandates must be considered first. Where the enterprise supports multiple options, Linux is preferred.

Key Takeaways

When looking to define standard hardware building blocks for MongoDB, start with these general recommendations:

- The more RAM the better
- Select fast CPUs
- Use local storage, preferably SSDs, or explicitly provisioned shared storage such as AWS PIOPS or equivalent (use SSDs in the shared storage, if available)
- Use an operating system certified with MongoDB Enterprise Advanced

A good choice of server platform would be a dual socket Intel or AMD-based server platform with local SSDs. If deploying on a public cloud, AWS r3.4xlarge, or AWS r3.8xlarge with EBS Provisioned IOPS (or equivalent from other vendors) is preferred.

You can learn more about hardware and OS selection by downloading the MongoDB Operations Best Practices guide.

Step 3: Virtualization Strategy

While not a prerequisite, building an infrastructure to deliver MongoDB-as-a-Service enables the IT group to utilize virtualization technologies. In efforts to drive up system utilization and enhance operational efficiency by eliminating “one application per server", most enterprises have already standardized on a certified set of virtualization technologies. MongoDB Enterprise Advanced is supported on all mainstream virtualized public and private cloud infrastructure, including:

- Hypervisor virtualization such as Xen, KVM, VMware vCloud Suite and vSphere platform, Oracle VirtualBox, OpenVZ and Microsoft Hyper-V
- Container virtualization, such as Linux Containers (LXC) and Docker
- Private and public cloud platforms based on the virtualization technologies described above, including OpenStack, Cloud Foundry and OpenShift, and public cloud offerings such as AWS, Google Compute Engine, Rackspace, and Microsoft Azure
- Non-virtualized public cloud offerings such as IBM's SoftLayer

With multiple VM (Virtual Machine) images or containers running MongoDB on a single physical host, consideration should be given to ensuring adequate resources are allocated to each instance. Avoid over-provisioning resources such as RAM. Most importantly, ensure that multiple members of a replica set are not sharing the same underlying hardware, as this will create a single point of failure.

Key Takeaways

MongoDB supports all mainstream virtualization platforms. As we will see below, the choice of virtualization technology can impact the strategy for database multi-tenancy within a single physical MongoDB cluster.

Step 4: Enabling Multi-Tenant Services

There are multiple approaches to building a multi-tenant MongoDB service on top of the virtualization technologies discussed in Step 3. The appropriate choice will depend on specific requirements for security, workload isolation, and performance. The following section focuses on the two latter criteria, while security is discussed in Step 5.

Hypervisor-Based Virtual Machines

Each physical server is partitioned into multiple VMs running a full operating system image and MongoDB (`mongod`) process. System resources such as CPU, RAM, and disk IO can be dedicated to each VM, preventing one VM from impacting the performance of others.

While this approach does not allow the density of instances seen with lighter-weight container-based virtualization, it does enable stronger isolation between each instance. It is also a well tested, mature approach used by technologies such as VMware vSphere and services such as AWS EC2.

A key consideration in deploying enterprise hypervisor technologies is to avoid over-provisioning at any level of CPU cores, RAM, network, or storage. These technologies assume that most hypervisor client systems will rarely use their allotted resources. That assumption is invalid for an operational database such as MongoDB. In particular, memory ballooning should be avoided or disabled, as it will conflict directly with MongoDB's approach to using RAM.

Containers

Using Linux's LXC containers and cgroups, a single physical host and Linux kernel can be partitioned into multiple isolated user-level containers, each running a single MongoDB process, assigned with unique user credentials for access control. As with VMs, system resources can be dedicated to each container to prevent oversubscription by competing workloads.

There are several advantages to using containers versus VMs:

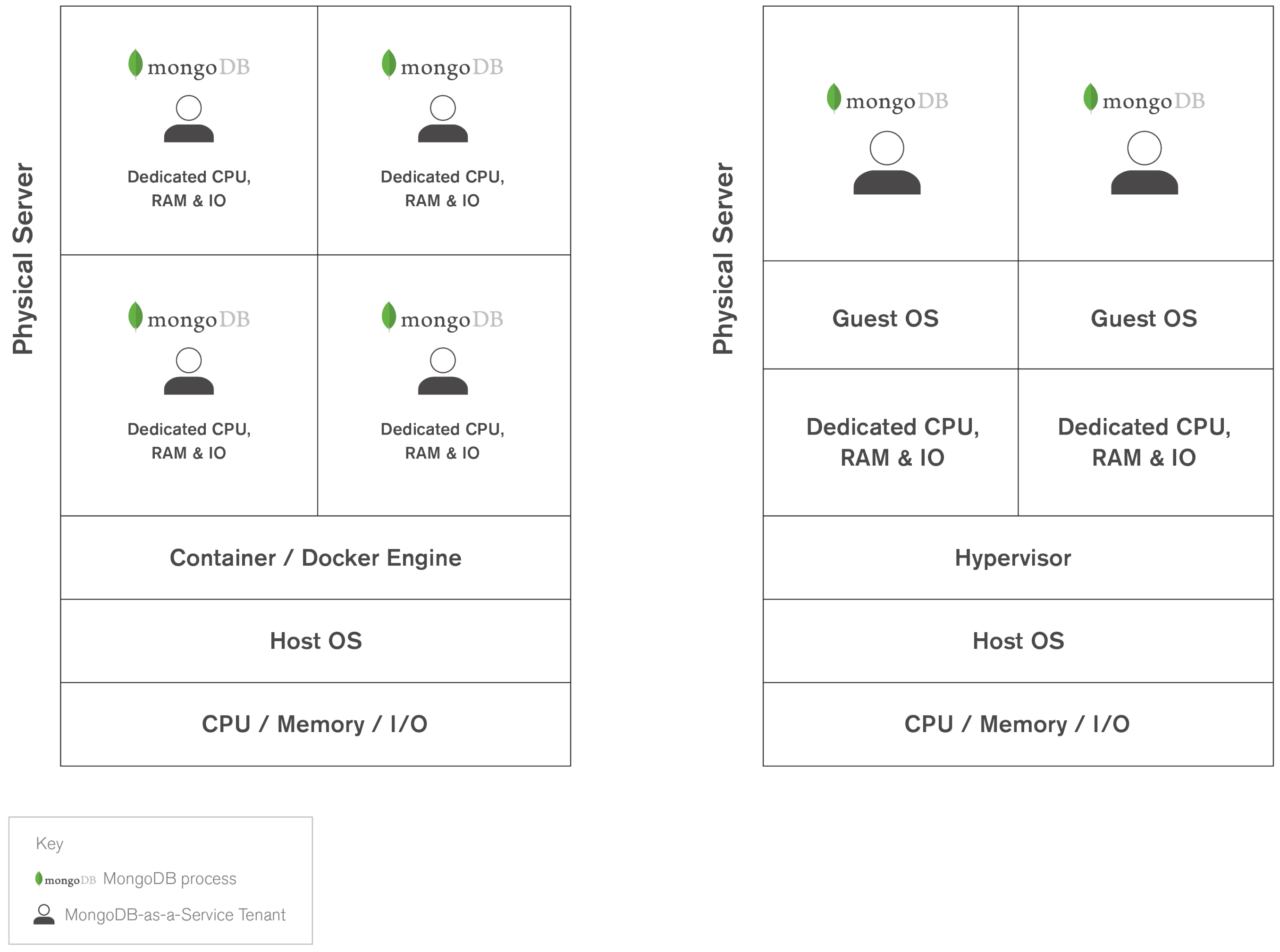


Figure 1: MongoDB Multi-Tenancy with Virtualization – Containers vs. VMs

- **Pack more instances per physical host** as there is less system overhead. Containers use one operating system image shared between all VMs rather than each VM carrying its own operating system
- **Faster to instantiate** a LXC or Docker container than it is to boot a guest operating system in a VM

It is common to run containers within VMs (e.g., when using Docker on Amazon EC2), providing a double level of virtualization.

The disadvantage to container-based virtualization is that there is less isolation between each container. A failure of the underlying operating system will result in failures of all the containers running on it.

More information on using MongoDB with Docker containers can be found in Enabling Microservices: Containers & Orchestration Explained.

Process Separation

An alternative approach is to run a MongoDB process for each tenant in a single operating system image. This allows for a high density of tenants, but with limited isolation there can be contention for system resources between processes. Linux cgroups can be used to constrain the use of RAM, CPU cores, and disk and network IO by each `mongod` process.

Logical DB Separation

Each tenant can be provisioned with a logical database (i.e., schema) in a shared MongoDB instance. While each tenant can be configured with their own access credentials, this approach affords the weakest level of isolation. Each tenant will be sharing not only the same hardware, but also

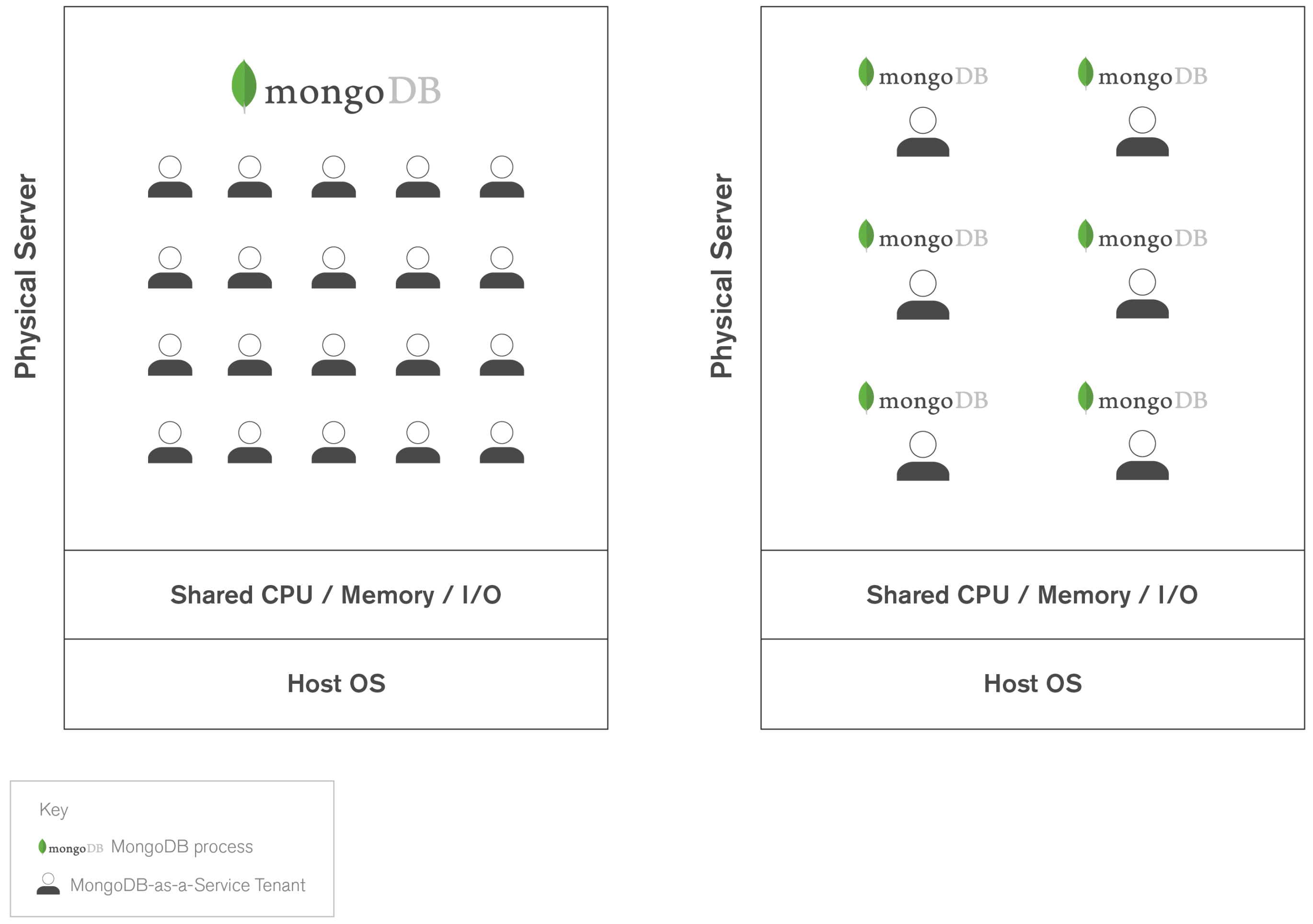

Physical Server
mongoDB
Shared CPU / Memory / I/O
Host OS
Physical Server
mongoDB
Shared CPU / Memory / I/O
Host OS
Key
MongoDB process
MongoDB-as-a-Service Tenant

Figure 2: MongoDB Multi-Tenancy with Logical vs. Process Separation

the same database resources such as address space, journal, and oplogs (used for replication).

As a result of this loosely coupled separation, one tenant could completely saturate the system, starving other users of resources. In addition, every tenant will be forced into using the same cluster topology, as replication is configured per MongoDB process, not per database.

MongoDB 3.6 enables operations teams to more easily inspect, monitor, and control each user session running in the database (across all logical databases). They can view, group, and search user sessions across every node in the cluster, and respond to performance issues in real time. For example, if a user or developer error is causing runaway queries, administrators now have the fine-grained operational oversight to view and terminate that session by removing all associated session state across a sharded cluster in a single operation.

Key Takeaways

When designing for multi-tenancy on a shared resource pool, the IT team must balance isolation, performance, and security. Users have a range of options from dedicated hardware -> VM -> container -> process -> logical database that provide decreasing levels of isolation between instances, but increasing density.

There is not a one-size-fits-all; and differing technologies can be combined to manage applications at different stages of their lifecycle and to accommodate specific SLAs and usage patterns. MongoDB is sufficiently flexible to support all of the approaches discussed above.

Step 5: Enforcing Security Isolation between Multiple Tenants

MongoDB Enterprise Advanced features extensive capabilities to enforce security isolation between tenants. Security is a dimension of service design that should be defined early, though it may be implemented progressively as the enterprise services mature. Details vary by organization and must go hand-in-hand with multi-tenant access to the cluster.

Authentication

MongoDB provides a variety of security management capabilities and integrates with typical enterprise security infrastructure, such as LDAP, Kerberos, and x509 certificates for the authentication of users, applications, and other nodes within the cluster (i.e. shards and replica set members). These capabilities may be applied at various levels of granularity, from the entire shared infrastructure, to individual clusters, databases, or collections, all the way down to the level of individual, labelled fields within documents (using field level redaction).

Authorization

A key enabler for multi-tenancy within a single cluster is MongoDB's user-defined roles, enabling administrators to assign fine-grained privileges to users, or applications. User privileges can be defined at both database and collection-level granularity. Authorization privileges can be based on the specific functionality users need in their roles, or to reflect departmental structures. For example:

- Administrators may be assigned privileges that enable them to create collections and indexes on a database, while business unit developers are restricted to document-level CRUD (Create, Read, Update, Delete) operations on a single collection.
- Specific administrator roles may have service-wide privileges to build replica sets and configure sharding, while others are restricted to creating new users, or inspecting logs.
- Within a multi-tenant environment, *landlord* developers and administrators in the IT team can be assigned permissions across multiple physical clusters and databases, while *tenant* developers and administrators in individual project teams can be granted a more limited set of actions across the logical databases or individual collections used by their application. This functionality enables a clear separation of duties and control.

For simplicity in account provisioning and maintenance, predefined roles can be delegated across entire teams, ensuring the enforcement of consistent policies across specific functions within the organization.

MongoDB Enterprise Advanced offers authorization integration with LDAP. This enables existing user privileges stored in the LDAP server to be mapped to MongoDB roles, without users having to be recreated in MongoDB itself. When configured with an LDAP server for authorization, MongoDB will allow user authentication via LDAP, Active Directory, Kerberos, or X.509 without requiring local user documents in the $external database. When a user successfully authenticates, MongoDB will perform a query against the LDAP server to retrieve all groups the LDAP user is a member of, and will transform those groups into their equivalent MongoDB roles.

Additionally, MongoDB offers read-only views for field-level security as a critical building block for trusted systems. MongoDB allows administrators to define non-materialized views that expose only a subset of data from an underlying collection. Permissions granted against the view are specified separately from permissions granted to the underlying collection(s). With redaction of data at the document or field level, a single record can contain data with multiple security levels accessible only to users with explicit privileges. This avoids the complexity of separating data across multiple databases, each with their own access policies.

Auditing

For compliance reporting, security administrators can use the MongoDB Enterprise Advanced's native audit log to track track any operation taken against the database – whether DML, DCL, or DDL.

Encryption

MongoDB data can be encrypted on the network and on disk. Support for SSL allows clients to connect to MongoDB over an encrypted channel. MongoDB supports FIPS 140-2 encryption when run in FIPS Mode with a FIPS validated Cryptographic module.

Data at rest can be protected using the Encrypted Storage Engine, which provides native encryption, avoiding much of the performance overhead of external encryption mechanisms. This storage engine can optionally integrate with a third party key management appliance via KMIP.

Key Takeaways

Definition of security policies should start at the outset of the project, based on corporate compliance and privacy directives.

Learn more about the security controls in MongoDB by downloading the MongoDB Security Reference Architecture.

Step 6: Meeting Service Level Agreement (SLA) Requirements

A critical factor in adoption of the MongoDB service is the platform's ability to meet the SLA requirements of each application. SLAs are most commonly defined in two dimensions:

- Application uptime – often expressed as a percentage of availability over time, for example 99.9% (system is unavailable for no more than 8.76 hours per year), 99.99% (unavailability of 52.56 minutes per year), or 99.999% (5.26 minutes per year). The availability percentage would typically include Mean Time to Recover (MTTR) after a failure;
- Delivered performance in the 95th percentile, expressed in operations per second and / or latency to the client.

SLAs are also sometimes defined for speed of issue resolution and the time to deliver new applications, though both of these are beyond the scope of this document.

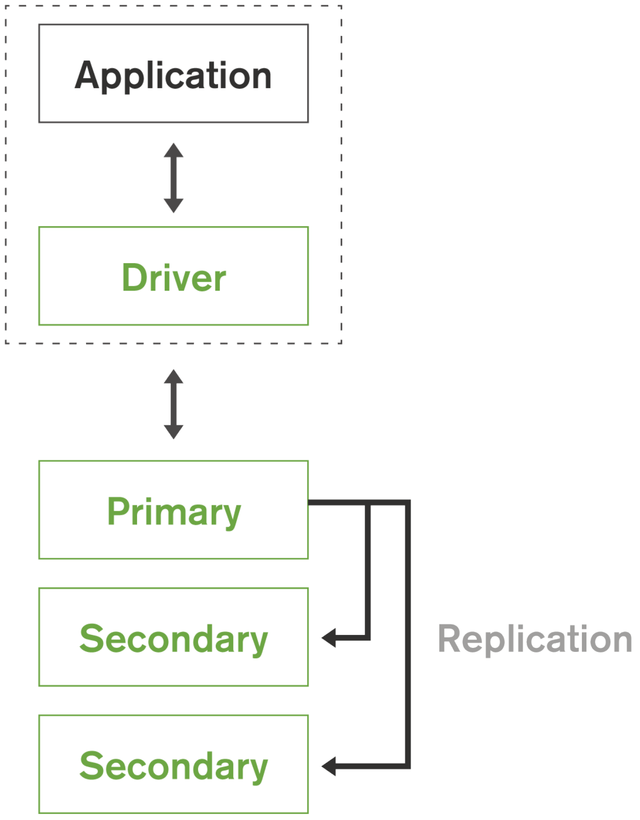


Figure 3: Self-Healing MongoDB Replica Sets for High Availability

Maintaining Service Continuity with MongoDB Replica Sets

While development environments can be run on a single instance of MongoDB, production applications should always use MongoDB's native replication to provide resilience in the event of platform outages.

MongoDB maintains multiple copies of data in replica sets. With fully automated failover and recovery, replica sets are self-healing so it is unnecessary to manually intervene to restore a system in the event of a failure. Replica sets also enable operational flexibility by providing a way to perform system maintenance (i.e., upgrading hardware and software) while preserving service continuity.

A replica set consists of multiple database replicas. At any given time, one member acts as the primary replica set member and the other members act as secondary replica set members. If the primary member suffers an outage (e.g. as a result of power failure, hardware fault, network partition) one of the secondary members is automatically

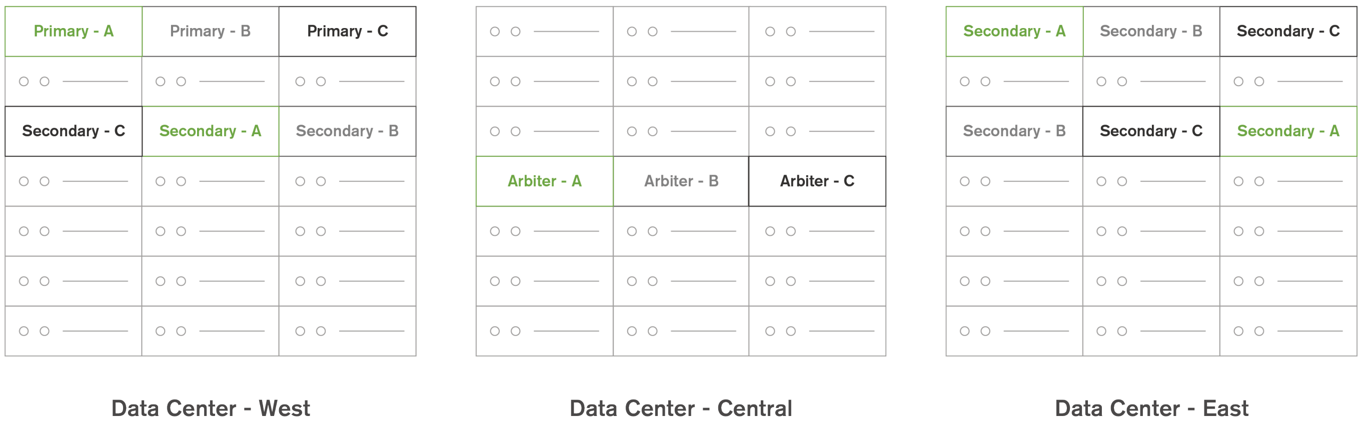


Figure 4: Active/Active Data Centers - Tolerates Failures of Servers, Racks & Data Center, plus Network Partitions

elected to primary and the client connections failover to that new primary.

The number of replicas in a MongoDB replica set is configurable, with a larger number of replica members providing increased data durability and protection against database downtime (e.g. in case of multiple machine failures, rack failures, data center failures, or network partitions). In MongodDB 3.0 and higher, replica sets can contain up to 50 members. Replica set members can be deployed in a single data center or across multiple data centers in active-standby or active-active modes, providing geographic resilience in the event of regional disasters. In addition, MongoDB provides advanced options to control data center awareness.

Read the MongoDB and Multi-Data Center Deployments whitepaper to learn more about replication and geographic awareness.

Deploying Replica Sets in a Shared MongoDB Service

Depending on the SLAs, multiple applications can be hosted on a single replica set, with workload isolation enforced by the appropriate multi-tenancy strategy discussed in Step 5.

The IT team then has the flexibility to separate the most performance or availability-sensitive applications to their own dedicated replica sets within the resource pool, while still maintaining centralized control and management of the service.

As a best practice replica set members should at the very least run on separate physical servers, preferably in separate racks and for highest resilience, across regionally-separated data centers.

The number of replica set members should also be carefully considered, ideally using a quantitative model of empirically-based probabilities of the various failure levels of different infrastructure components (i.e. VM, physical server, rack, data center, and region). At a minimum, three members should be deployed in each replica set, though in less critical applications it is possible to use two replica set members and an arbiter (note that in this model, the replica set would be unable to serve writes if configured with a majority write concern in the event of a failure of either of the replica set members).

Database Scaling with MongoDB Automatic Sharding

While performance-intensive applications can be moved to their own dedicated replica sets, as the workload continues to grow users should consider scaling out (sharding) MongoDB if any of the following conditions are anticipated:

- **RAM Limitation.** The size of the system's active working set plus indexes is expected to exceed the capacity of the maximum amount of RAM in the system.
- **Disk I/O Limitation.** The system will have a large amount of write activity, and the operating system will not be able to write data fast enough to meet demand, or I/O bandwidth will limit how fast the writes can be flushed to disk.

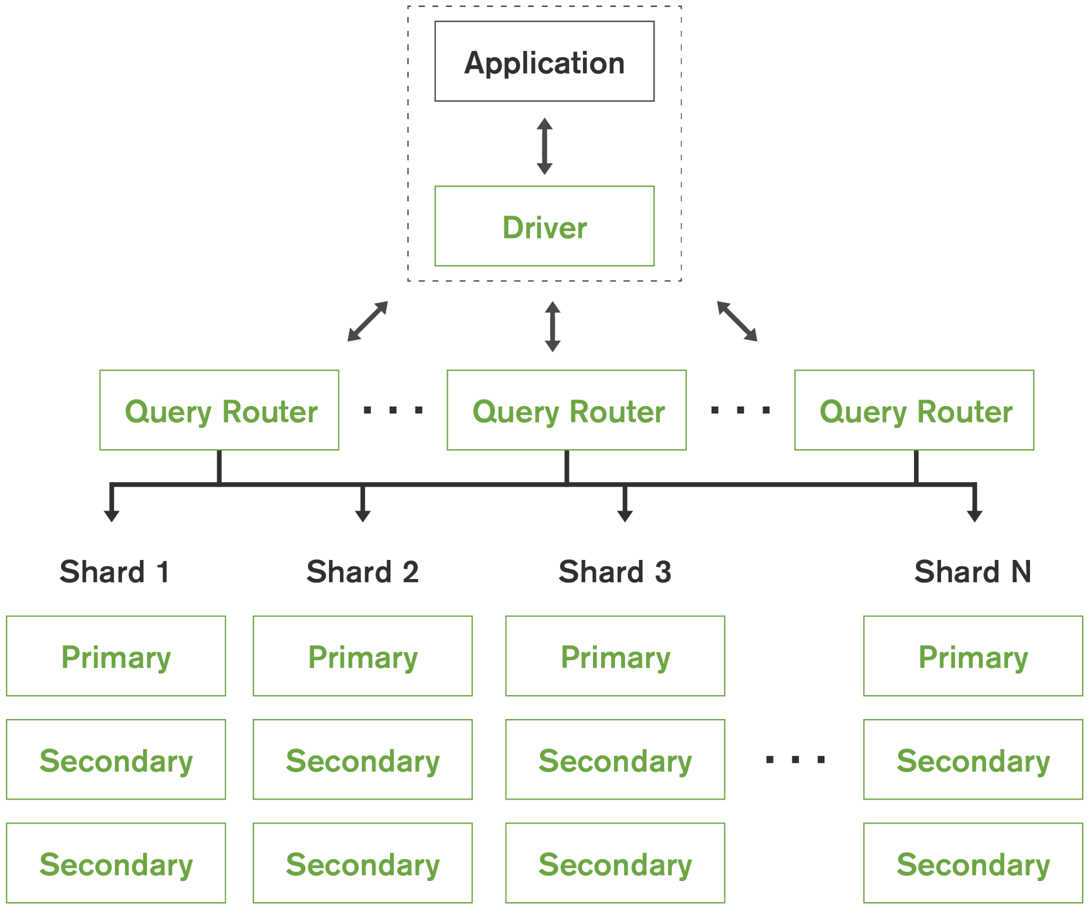


Figure 5: Sharding and replica sets – automatic sharding provides horizontal scalability; replica sets prevent downtime

- **Storage Limitation.** The data set will grow to exceed the storage capacity of a single node in the system.

Applications that meet these criteria, or that are likely to do so in the future, should be designed for scaling out in advance rather than waiting until they run out of capacity.

MongoDB provides horizontal scale out using a technique called sharding, allowing MongoDB deployments to scale beyond the hardware limitations of a single server. Sharding distributes data across multiple physical partitions called shards, and is transparent to applications. Shards can be located within a single data center or distributed across multiple data centers. As illustrated in Figure 5, each shard is deployed in a replica set, to provide both scalability and high availability to the MongoDB service.

MongoDB automatically balances the data in the cluster as the data grows or the size of the cluster increases or decreases. For more on sharding see the Sharding Introduction.

Deploying Shards in a Shared MongoDB Service

While sharding is automatic and transparent to the application, careful consideration needs to be given to selecting a shard key as this controls how the database is partitioned and distributed across the hardware cluster. Shard key selection can have a significant impact on the performance of the database. The choice of shard key is application-dependent, based on the database schema and the way in which the application queries and writes data.

Unless MongoDB is servicing a single application accessed by multiple tenants (i.e. Software-as-a-Service, or SaaS) it is not appropriate to provision all applications to a single sharded cluster. Instead, each application requiring the additional scaling that sharding brings should be deployed to its own sharded cluster within the shared MongoDB resource pool. This approach ensures that each application is scaled according to its workload patterns.

Review the documentation to learn more about shard key selection.

Key Takeaways

Failure to meet SLAs will not only result in the MongoDB service failing to gain traction within the organization, it can also result in damage to the corporate brand, lost customers, and even regulatory penalties.

- All production applications should use MongoDB's replica sets to avoid downtime that can result from system failures.
- Busier or more critical apps can be provisioned to their own dedicated replica sets to achieve higher performance.
- When an application needs to scale beyond the capacity of a single replica set master, the database can be re-provisioned onto a sharded cluster.

Even though you may have some application databases co-located on the same physical hardware and others distributed to dedicated replica sets and sharded clusters, you can still manage the overall MongoDB resource pool as a single, shared service. This is discussed in the following section.

Step 7: Managing the Service: Provisioning, Monitoring, and Disaster Recovery

Ops Manager is the simplest way to run MongoDB, making it easy for operations teams to deploy, monitor, backup, and scale MongoDB. Ops Manager was created by the engineers who develop the database and is available as part of MongoDB Enterprise Advanced. Many of the capabilities of Ops Manager are also available with MongoDB Cloud Manager, hosted in the cloud. Today, Cloud Manager supports thousands of deployments, including systems from one to hundreds of servers.

Ops Manager and Cloud Manager incorporate best practices to help keep managed databases healthy and optimized. They ensures operational continuity by converting complex manual tasks into reliable, automated procedures with the click of a button or via an API call:

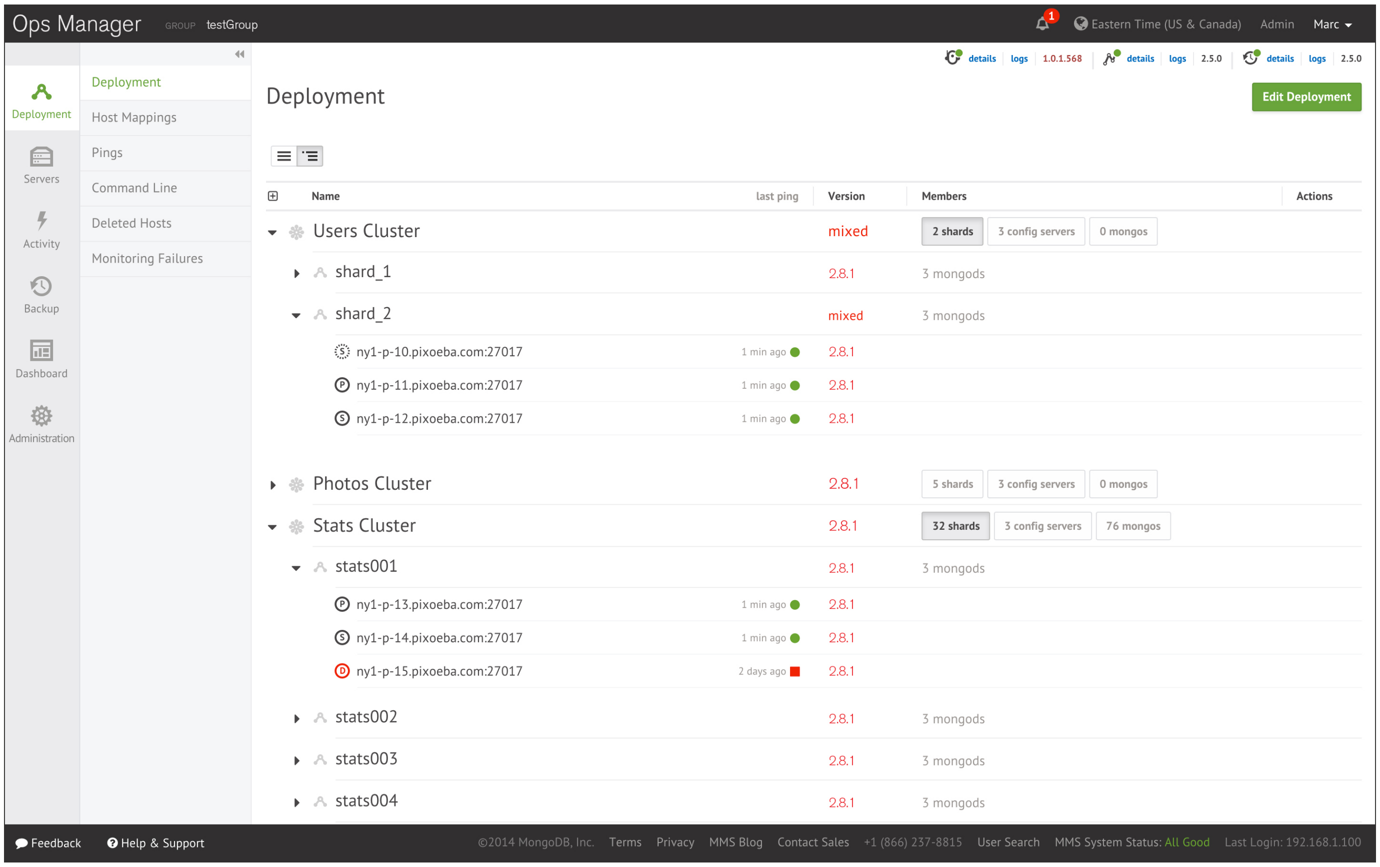
- **Deploy.** Any topology, at any scale
- **Upgrade.** In minutes, with no downtime
- **Scale.** Add capacity, without taking the application offline;
- **Scheduled Backups.** Customize to meet recovery goals
- **Point-in-time Recovery.** Restore to any point in time, because disasters aren't scheduled
- **Performance Alerts.** Monitor 100+ system metrics and get custom alerts before the system degrades.

Ops Manager roles can be defined to IT group administrators across the entire shared environment, and delegated to individual project teams to provide access to just the resources they have provisioned. From MongoDB 3.6, multiple Projects (each managing multiple MongoDB clusters) can be placed under a single organization, allowing operations teams to centrally view and administer all Projects under the organization hierarchy.

Deployments and Upgrades

It must be simple for project teams to request allocation of resources from the MongoDB resource pool, and for those resources to then be provisioned and managed. Ops Manager reliably orchestrates the tasks that administrators have traditionally performed manually – provisioning a new cluster, upgrades, restoring systems to a point in time, and many other operational tasks.

Ops Manager provides the ability to create pre-provisioned server pools. The Ops Manager agent can be installed across a fleet of servers (physical hardware, VMs, AWS instances, etc.) by a configuration management tool such as Chef, Puppet, or Ansible. The server pool can then be exposed to internal teams, ready for provisioning servers into their local groups, either by the programmatic Ops Manager API or the Ops Manager GUI. When users request an instance, Ops Manager will remove the server from the pool, and then provision and configure it into the local group. It can return the server to the pool when it is no longer required, all without sysadmin intervention. Administrators can track when servers are provisioned



Deployment

Name	last ping	Version	Members	Actions
Users Cluster		mixed	2 shards · 3 config servers · 0 mongos	
shard_1		2.8.1	3 mongods	
shard_2		mixed	3 mongods	
ny1-p-10.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-11.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-12.pixoeba.com:27017	1 min ago ●	2.8.1		
Photos Cluster		2.8.1	5 shards · 3 config servers · 0 mongos	
Stats Cluster		2.8.1	32 shards · 3 config servers · 76 mongos	
stats001		2.8.1	3 mongods	
ny1-p-13.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-14.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-15.pixoeba.com:27017	2 days ago 🟥	2.8.1		
stats002		2.8.1	3 mongods	
stats003		2.8.1	3 mongods	
stats004		2.8.1	3 mongods	

Figure 6: Ops Manager self-service portal: simple, intuitive, and powerful. Deploy and upgrade clusters with a single click.

from the pool, and receive alerts when available server resources are running low. Pre-provisioned server pools allow administrators to create true, on-demand database resources for private cloud environments.

Building upon server pools, Ops Manager offers certified integration with Cloud Foundry. BOSH, the Cloud Foundry configuration management tool, can install the Ops Manager agent onto the server configuration requested by the user, and then use the Ops Manager API to build the desired MongoDB configuration. Once the deployment has reached goal state, Cloud Foundry will notify the user of the URL of their MongoDB deployment. From this point, users can log in to Ops Manager to monitor, back-up, and automate upgrades of their deployment.

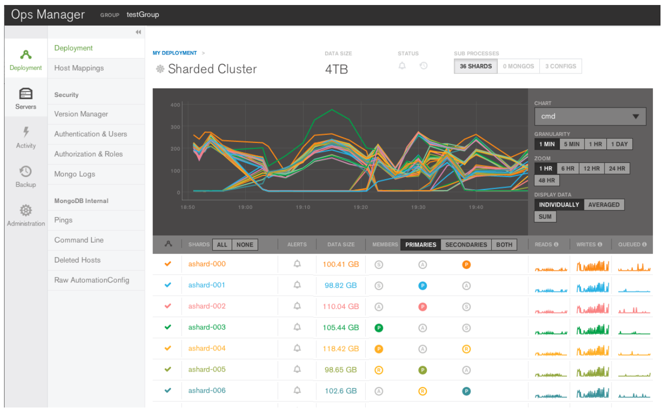
Ops Manager is designed to adapt to problems as they arise by continuously assessing state and making adjustments as needed. Here's how:

- Ops Manager agents are installed on servers (where MongoDB will be deployed), either through configuration management tools, or manually by an administrator.
- The administrator creates a new design goal for the system, either as a modification to an existing deployment (e.g., upgrade, oplog resize, new shard), or as a new system.
- The agents periodically check in with the Ops Manager central server and receive the new design instructions.
- Agents create and follow a plan for implementing the design. Using a sophisticated rules engine, agents continuously adjust their individual plans as conditions change. In the face of many failure scenarios – such as server failures and network partitions – agents will revise their plans to reach a safe state.
- Minutes later, the system is deployed, safely and reliably.

If the instance is a short-lived development environment, a single click will terminate the instances and return the servers to the resource pool, ready for consumption by another team.

In addition to initial deployment, Ops Manager and Cloud Manager make it possible to dynamically resize capacity by adding shards and replica set members. Other maintenance tasks such as upgrading MongoDB or resizing the oplog can be reduced from dozens or hundreds of manual steps to the click of a button, all with zero downtime.

Monitoring



SHARDS	ALERTS	DATA SIZE	MEMBERS			READS	WRITES	QUEUED
✓ ashard-000		100.41 GB	S	A	P			
✓ ashard-001		98.82 GB	S	P	A			
✓ ashard-002		110.04 GB	A	P	S			
✓ ashard-003		105.44 GB	P	A	S			
✓ ashard-004		118.42 GB	P	A	R			
✓ ashard-005		98.65 GB	R	P	A			
✓ ashard-006		102.6 GB	A	R	P			

Figure 7: Ops Manager provides real time & historic visibility into the MongoDB deployment.

Ops Manager provides administrators and project owners with visibility into the MongoDB service. Featuring charts, custom dashboards, and automated alerting, Ops Manager tracks 100+ key database and systems health metrics including operations counters, memory, and CPU utilization, replication status, open connections, queues, and any node status.

The metrics are securely reported to Ops Manager and Cloud Manager where they are processed, aggregated, alerted and visualized in a browser, letting administrators easily determine the health of MongoDB in real time. Views can be based on explicit permissions, so project team visibility can be restricted to their own applications, while systems administrators can monitor all the MongoDB deployments in the organization.

Historic performance can be reviewed in order to create operational baselines and to support capacity planning. Integration with existing monitoring tools is also straightforward via the Ops Manager RESTful API, making the deep insights from Ops Manager part of a consolidated view across your operations.

Ops Manager and Cloud Manager allow administrators to set custom alerts when key metrics are out of range. Alerts can be configured for a range of parameters affecting individual hosts, replica sets, agents and backup. Alerts can be sent via SMS and email or integrated into existing incident management systems such as PagerDuty and HipChat to proactively warn of potential issues, before they escalate to costly outages.

If using Cloud Manager, access to monitoring data can also be shared with MongoDB support engineers, providing fast issue resolution by eliminating the need to ship logs between different teams.

Disaster Recovery: Backups & Point-in-Time Recovery

A backup and recovery strategy is necessary to protect your mission-critical data against catastrophic failure, such as a fire or flood in a data center, or human error such as code errors or accidentally dropping collections. With a backup and recovery strategy in place, administrators can restore business operations without data loss, and the organization can meet regulatory and compliance requirements. Taking regular backups offers other advantages, as well. The backups can be used to seed new environments for development, staging, or QA without impacting production systems.

Ops Manager and Cloud Manager backups are maintained continuously, just a few seconds behind the operational system. If the MongoDB cluster experiences a failure, the most recent backup is only moments behind, minimizing exposure to data loss. Ops Manager and Cloud Manager are the only MongoDB solutions that offer point-in-time backup of replica sets and cluster-wide snapshots of sharded clusters. You can restore to precisely the moment you need, quickly and safely.

Because Ops Manager and Cloud Manager only read the oplog, the ongoing performance impact is minimal – similar to that of adding an additional replica to a replica set.

By using MongoDB Enterprise Advanced you can deploy Ops Manager to control backups in your local data center, or use Cloud Manager, which includes a fully managed backup solution with a pay-as-you-go model. Dedicated

MongoDB engineers monitor user backups on a 24x365 basis, alerting operations teams if problems arise.

Integrating MongoDB with External Monitoring Solutions

The Ops Manager and Cloud Manager API provides integration with external management frameworks through programmatic access to automation features and monitoring data.

In addition to Ops Manager and Cloud Manager, MongoDB Enterprise Advanced can report system information to SNMP traps, supporting centralized data collection and aggregation via external monitoring solutions. Review the documentation to learn more about SNMP integration.

Key Takeaway

Ops Manager provides the management platform to provision, monitor and backup the MongoDB service. Using Ops Manager, the IT team can manage the MongoDB resource pool as a central asset, shared by multiple project teams.

Step 8: Cost Accounting & Chargeback

How cost accounting and chargeback is managed is largely dependent on specific organizational policies. There are, however, best practices to observe:

- If those project teams consuming the service do not bear proportionate costs, there is a risk of overuse and depletion of available resources. Provisioned capacity can be left idle by teams who have no motivation to return it to the service's resource pool.
- Conversely, if the resources are overpriced, the consumers will make little if any use of them, instead favoring less expensive options, including local business unit resources or public cloud providers.

Accounting processes will typically begin with the underlying infrastructure layer (i.e., servers and storage) whose resources are consumed first. As services are built on the underlying infrastructure, the chosen virtualization technologies must supplement this with appropriate charges for software, support, and administration costs.

Accounting Example: AWS Tag-Based Cost Allocation

AWS is used to provide an example of cost accounting within a shared resource pool. Each provisioned instance includes the following tags, which are then used to identify billable resource usage:

Tag Name	Significance
user:Owner	Username of the resource requestor
user:Stack	Development / Test / Production
user:CostCenter	Business unit or project team
user:Application	Formal name of the application consuming the resource

Table 3: Using Tags for Cost Accounting

AWS monthly Custom Billing Reports can be generated based on these tags, with expenses charged back to the applicable cost center.

Key Takeaways

Cost accounting and chargeback policies are specific to each organization. Many public and private cloud infrastructures provide mechanisms to tracking and billing the use of underlying infrastructure resources.

Step 9: Define the Implementation Plan

With the variety of enterprise requirements for delivering MongoDB as a Service, there is no single “out of the box" template for an implementation plan. Using the considerations presented in this whitepaper, MongoDB consultants can apply best practices to collaborate with the IT group in defining a plan that accelerates implementation, while at the same time reducing risk.

Personnel Requirements

The IT group implementing the MongoDB service should seek participation from representatives drawn from all internal stakeholders. The primary service implementation work may be performed by operations-capable developers from within the organization's own staff, or by a trusted Systems Integrator (SI). However, active participation and review throughout the development process should be provided by:

- MongoDB-as-a-Service project management
- Business unit architects
- Operations staff who will assume responsibility of the service
- Network and storage administrators
- Application developers who are the internal customers for the first phase of the service
- Corporate security and compliance representatives

Augmenting the Team: MongoDB Consulting Services

MongoDB Consulting Engineers should also be used as extensions to the project team, bringing expertise and best practices from other MongoDB-as-a-Service engagements. A range of fixed-term engagements are available to support you through design, testing, launch, and ongoing management of the service:

- The MongoDB Private Cloud Accelerator consulting package provides support from the experts to get your MongoDB private cloud up and running.
- The MongoDB Health Check provides an assessment of the service's architecture design readiness and operational policies.
- The Operations Rapid Start package gives your operations and devops teams the skills and tools to run and manage MongoDB with confidence.
- Once launched, a MongoDB Dedicated Consulting Engineer provides ongoing advisory services to the IT team from a named, experienced engineer.

These consulting packages complement a range of services that can be provided for individual project teams during the development phase of their applications, including MongoDB schema design, sharding, and performance tuning.

Learn more about the full range of MongoDB consulting services.

Key Takeaways

Create a service implementation team with 360-degree involvement of MongoDB and enterprise stakeholders.

Step 10: Production-Grade DBaaS - Supported, Secure, and Automated

We are the MongoDB experts. Over 4,300 organizations rely on our commercial products, including startups and more than half of the Fortune 100. We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can scale up and down when you need to, with no downtime, full security, and high performance.

MongoDB Stitch is a backend as a service (BaaS), giving developers full access to MongoDB, declarative read/write controls, and integration with their choice of services.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Professional helps you manage your deployment and keep it running smoothly. It includes support from MongoDB engineers, as well as access to MongoDB Cloud Manager.

Development Support helps you get up and running quickly. It gives you a complete package of software and services for the early stages of your project.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

MongoDB Atlas: Database as a Service For MongoDB

MongoDB Atlas is a cloud database service that makes it easy to deploy, operate, and scale MongoDB in the cloud by automating time-consuming administration tasks such as database setup, security implementation, scaling, patching, and more.

MongoDB Atlas is available on-demand through a pay-as-you-go model and billed on an hourly basis.

It's easy to get started – use a simple GUI to select the public cloud provider, region, instance size, and features you need. MongoDB Atlas provides:

- Security features to protect your data, with fine-grained access control and end-to-end encryption
- Built in replication for always-on availability. Cross-region replication within a public cloud can be enabled to help tolerate the failure of an entire cloud region.
- Fully managed, continuous and consistent backups with point in time recovery to protect against data corruption, and the ability to query backups in-place without full restores
- Fine-grained monitoring and customizable alerts for comprehensive performance visibility
- One-click scale up, out, or down on demand. MongoDB Atlas can provision additional storage capacity as needed without manual intervention.
- Automated patching and single-click upgrades for new major versions of the database, enabling you to take advantage of the latest and greatest MongoDB features
- Live migration to move your self-managed MongoDB clusters into the Atlas service with minimal downtime

MongoDB Atlas can be used for everything from a quick Proof of Concept, to test/QA environments, to powering production applications. The user experience across MongoDB Atlas, Cloud Manager, and Ops Manager is consistent, ensuring that disruption is minimal if you decide to manage MongoDB yourself and migrate to your own infrastructure.

MongoDB Stitch: Backend as a Service

MongoDB Stitch is a backend as a service (BaaS), giving developers a REST-like API to MongoDB, and composability with other services, backed by a robust system for configuring fine-grained data access controls. Stitch provides native SDKs for JavaScript, iOS, and Android.

Built-in integrations give your application frontend access to your favorite third party services: Twilio, AWS S3, Slack, Mailgun, PubNub, Google, and more. For ultimate flexibility, you can add custom integrations using MongoDB Stitch's HTTP service.

MongoDB Stitch allows you to compose multi-stage pipelines that orchestrate data across multiple services; where each stage acts on the data before passing its results on to the next.

Unlike other BaaS offerings, MongoDB Stitch works with your existing as well as new MongoDB clusters, giving you access to the full power and scalability of the database. By defining appropriate data access rules, you can selectively expose your existing MongoDB data to other applications through MongoDB Stitch's API.

Take advantage of the free tier to get started; when you need more bandwidth, the usage-based pricing model ensures you only pay for what you consume. Learn more and try it out for yourself.

Conclusion

As more internal business units and project teams build modern applications on MongoDB, IT groups can improve agility, efficiency, accountability and governance by offering MongoDB-as-a-Service. This white paper has been designed to provide the top 10 considerations you make as you embark on the next phase of industrializing MongoDB consumption in your organization.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)
MongoDB Stitch backend as a service (mongodb.com/cloud/stitch)

mongoDB