MongoDB on AWS

Guidelines and Best Practices

Rahul Bhartia

May 2015

© 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract 4

Introduction 4

NoSQL on AWS 4

MongoDB: A Primer 5

Storage and Access Patterns 6

Availability and Scaling 10

Designs for Deploying MongoDB on AWS 12

High-Performance Storage 12

High Availability and Scale 15

MongoDB: Operations 20

Using MMS or Ops Manager 21

Do It Yourself 22

Network Security 26

Conclusion 29

Further Reading 29

Notes 30

Abstract

Amazon Web Services (AWS) is a flexible, cost-effective, easy-to-use cloud computing platform.[1] MongoDB is a popular NoSQL database that is widely deployed in the AWS cloud.[2] Running your own MongoDB deployment on Amazon Elastic Cloud Compute (Amazon EC2) is a great solution for users whose applications require high-performance operations on large datasets.

This whitepaper provides an overview of MongoDB and its implementation on the AWS cloud platform. It also discusses best practices and implementation characteristics such as performance, durability, and security, and focuses on AWS features relevant to MongoDB that help ensure scalability, high availability, and disaster recovery.

Introduction

NoSQL refers to a subset of structured storage software that is optimized for high-performance operations on large datasets. As the name implies, querying of these systems is not based on the SQL language—instead, each product provides its own interface for accessing the system and its features.

One way to organize the different NoSQL products is by looking at the underlying data model:

- Key-value stores – Data is organized as key-value relationships and accessed by primary key.
- Graph databases – Data is organized as graph data structures and accessed through semantic queries.
- Document databases – Data is organized as documents (e.g., JSON) and accessed by fields within the document.

NoSQL on AWS

AWS provides an excellent platform for running many advanced data systems in the cloud. Some of the unique characteristics of the AWS cloud provide strong benefits for running NoSQL systems. A general understanding of these characteristics can help you make good architecture decisions for your system.

In addition, AWS provides the following services for NoSQL and storage that do not require direct administration, and offers usage-based pricing. Consider these options as possible alternatives to building your own system with open source software (OSS) or a commercial NoSQL product.

- Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.[3] All data items are stored on solid-state drives (SSDs) and are automatically replicated across three Availability Zones in an AWS region to provide built-in high availability and data durability. With Amazon DynamoDB, you can offload the administrative burden of operating and scaling a highly available distributed database cluster while paying a low variable price for only the resources you consume.
- Amazon Simple Storage Service (Amazon S3) provides a simple web services interface that can store and retrieve any amount of data anytime from anywhere on the web.[4] Amazon S3 gives developers access to the same highly scalable, reliable, secure, fast, and inexpensive infrastructure that Amazon uses to run its own global network of websites. Amazon S3 maximizes benefits of scale, and passes those benefits on to you.

MongoDB: A Primer

MongoDB is a popular NoSQL document database that provides rich features, fast time-to-market, global scalability, and high availability, and is inexpensive to operate.

MongoDB can be scaled within and across multiple distributed locations. As your deployments grow in terms of data volume and throughput, MongoDB scales easily with no downtime, and without changing your application. And as your availability and recovery goals evolve, MongoDB lets you adapt flexibly across data centers.

After reviewing the general features of MongoDB, we will take a look at some of the key considerations for performance and high availability when using MongoDB on AWS.

Storage and Access Patterns

MongoDB version 3.0 exposes a new storage engine API, which enables the integration of pluggable storage engines that extend MongoDB with new capabilities and enables optimal use of specific architectures. MongoDB 3.0 includes two storage engines:

- The default MMAPv1 engine. This is an improved version of the engine used in previous MongoDB releases, and includes collection-level concurrency control.
- The new WiredTiger storage engine. This engine provides document-level concurrency control and native compression. For many applications, it will result in lower storage costs, better hardware utilization, and more predictable performance.

To enable the WiredTiger storage engine, use the `storageEngine` option on the **mongod** command line; for example:

```
mongod --storageEngine WiredTiger
```

Although each storage engine is optimized for different workloads, users still leverage the same MongoDB query language, data model, scaling, security, and operational tooling, independent of the engine they use. As a result, most of the best practices discussed in this guide apply to both storage engines. Any differences in recommendations between the two storage engines are noted.

Now let's take a look at the MongoDB processes that require access to disk and their access patterns.

Data Access

In order for MongoDB to process an operation on an object, that object must reside in memory. The MMAPv1 storage engine uses memory-mapped files, whereas WiredTiger manages objects through its in-memory cache. When you perform a read or write operation on an object that is not currently in memory, it leads to a page fault (MMAPv1) or cache miss (WiredTiger) so that the object can be read from disk and loaded into memory.

If your application's working set is much larger than the available memory, access requests to some objects will cause reads from disk before the operation can complete. Such requests are often the largest driver of random I/O, especially for databases that are larger in size than available memory. If your working set exceeds available memory on a single server, you should consider sharding your system across multiple servers.

You should pay attention to the read-ahead settings on your block device to see how much data is read in such situations. Having a large setting for read-ahead is discouraged, because it will cause the system to read more data into memory than is necessary, and it might possibly evict other data that may be used by your application. This is particularly true for services that limit block size, such as Amazon Elastic Block Store (EBS) volumes, which are described later in this paper.

Write Operations

At a high level, both storage engines write data to memory, and then periodically synchronize the data to disk. However, the two MongoDB 3.0 storage engines differ in their approach:

- MMAPv1 implements collection-level concurrency control with atomic in-place updates of document values. To ensure that all modifications to a MongoDB dataset are durably written to disk, MongoDB records all modifications in a journal that it writes to disk more frequently than it writes the data files. By default, data files are flushed to disk every 60 seconds. You can change this interval by using the **mongod** `syncDelay` option.
- WiredTiger implements document-level concurrency control with support for multiple concurrent writers and native compression. WiredTiger rewrites the document instead of implementing in-place updates. WiredTiger uses a write-ahead transaction log in combination with checkpoints to ensure data persistence. By default, data is flushed to disk every 60 seconds after the last checkpoint, or after 2 GB of data has been written. You can change this interval by using the **mongod** `wiredTigerCheckpointDelaySecs` option.

Journal

MongoDB uses write-ahead logging to an on-disk journal to guarantee write operation durability. Before applying a change to the data files, MongoDB writes the idempotent change operation to the journal. If MongoDB should terminate or encounter an error before it can write the changes from the journal to the data files, MongoDB can reapply the write operation and maintain a consistent state.

The journal is periodically flushed to disk, and its behavior is slightly different in each storage engine:

- MMAPv1. The journal is flushed to disk every 100 ms by default. If MongoDB is waiting for the journal before acknowledging the write operation, the journal is flushed to disk every 30 ms.
- WiredTiger. Journal changes are written to disk periodically or immediately if an operation is waiting for the journal before acknowledging the write operation.

With MMAPv1, the journal should always be enabled because it allows the database to recover in case of an unclean shutdown. With WiredTiger, the append-only file system means that the transaction log is not necessary for recovery in case of an unclean shutdown, because the data files are always valid.

Locating MongoDB's journal files and data files on separate storage arrays may improve performance, as write operations to each file will not compete for the same resources. Depending on the frequency of write operations, journal files can also be stored on conventional disks due to their sequential write profile.

Basic Tips

MongoDB provides recommendations for read-ahead and other environment settings in their production notes.[5] Here's a summary of their recommendations:

Packages

- Always use 64-bit builds for production. 32-bit builds support systems with only 2 GB of memory.
- Use MongoDB 3.0 or later. Significant feature enhancements in the 3.0 release include support for two storage engines, MMAPv1 and WiredTiger, as discussed earlier in this paper.

Concurrency

- With MongoDB 3.0, WiredTiger enforces control at the document level while the MMAPv1 storage engine implements collection-level concurrency control. For many applications, WiredTiger will provide benefits in greater hardware utilization by supporting simultaneous write access to multiple documents in a collection.

Networking

- Always run MongoDB in a trusted environment, and limit exposure by using network rules that prevent access from all unknown machines, systems, and networks. See the Security section of the MongoDB 3.0 manual for additional information.[6]

Storage

- Use XFS or Ext4. These file systems support I/O suspend and write-cache flushing, which is critical for multi-disk consistent snapshots. XFS and Ext4 also support important tuning options to improve MongoDB performance.

> For improved performance, consider separating your data, journal, and logs onto different storage devices, based on your application's access and write pattern.

- Turn off atime and diratime when you mount the data volume. Doing so reduces I/O overhead by disabling features that aren't useful to MongoDB.
- Assign swap space for your system. Allocating swap space can avoid issues with memory contention and can prevent out-of-memory conditions.
- Use a NOOP scheduler for best performance. The NOOP scheduler allows the operating system to defer I/O scheduling to the underlying hypervisor.

Operating system

- Raise file descriptor limits. The default limit of 1024 open files on most systems won't work for production-scale workloads. For more information, refer to http://www.mongodb.org/display/DOCS/Too+Many+Open+Files.
- Disable transparent huge pages. MongoDB performs better with standard (4096-byte) virtual memory pages.

- Ensure that read-ahead settings for the block devices that store the database files are appropriate. For random-access use patterns, set low read-ahead values. A read-ahead setting of 32 (16 KB) often works well.

Availability and Scaling

The design of your MongoDB installation depends on the scale at which you want to operate. This section provides general descriptions of various MongoDB deployment topologies.

Standalone Instances

Mongod is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations. Standalone deployments are useful for development, but should not be used in production because they provide no automatic failover.

Replica Sets

A MongoDB replica set is a group of mongod processes that maintain multiple copies of the data and perform automatic failover for high availability.

A replica set consists of multiple replicas. At any given time, one member acts as the primary member and the others act as secondary members. MongoDB is strongly consistent by default: read and write operations are issued to a primary copy of the data. If the primary member fails for any reason (e.g., hardware failure, network partition) one of the secondary members is automatically elected to the primary role and begins to process all write operations.

Applications can optionally specify a read preference to read from the nearest secondary members, as measured by ping distance.[7] Reading from secondaries will require additional considerations to account for eventual consistency, but can be utilized when low latency is more important than consistency. Applications can also specify the desired consistency of any single write operation with a write concern.[8]

All secondary members within a replica set do not have to use the same storage engine options, but it is important to ensure that the secondary member can keep up with the primary member so that it doesn't drift too far out of sync.

Replica sets also support operational flexibility by providing a way to upgrade hardware and software without requiring the database to go offline. This is an important feature, as these types of operations can account for as much as one third of all downtime in traditional systems.

Sharded Clusters

MongoDB provides horizontal scale-out for databases on low cost, commodity hardware by using a technique called *sharding*. Sharding distributes data across multiple partitions called *shards*.

Sharding allows MongoDB deployments to address the hardware limitations of a single server, such as bottlenecks in RAM or disk I/O, without adding complexity to the application. MongoDB automatically balances the data in the sharded cluster as the data grows or the size of the cluster increases or decreases.

A sharded cluster consists of the following:

- A shard, which is a standalone instance or replica set that holds a subset of a collection's data. For production deployments, all shards should be deployed as replica sets.
- A config server, which is a mongod process that maintains metadata about the state of the sharded cluster. Production deployments should use three config servers.
- A query router called *mongos*, which uses the shard key contained in the query to route the query efficiently only to the shards, which have documents matching the shard key. Applications send all queries to a query router. Typically, in a deployment with a large number of application servers, you would load balance across a pool of query routers.

Your cluster should manage a large quantity of data for sharding to have an effect. Most of the time, sharding a small collection is not worth the added complexity and overhead unless you need additional write capacity. If you have a small dataset, a properly configured single MongoDB instance or a replica set will usually be enough for your persistence layer needs.

Designs for Deploying MongoDB on AWS

This section discusses how you can apply MongoDB features to AWS features and services to deploy MongoDB in the most optimal and efficient way.

High-Performance Storage

Understanding the total I/O required is key to selecting an appropriate storage configuration on AWS. Most of the I/O driven by MongoDB is random. If your working set is much larger than memory, with random access patterns you may need many thousands of IOPS from your storage layer to satisfy demand.

> Note that WiredTiger provides compression, which will affect the sizing of your deployment, allowing you to scale your storage resources more efficiently.

AWS offers two broad choices to construct the storage layer of your MongoDB infrastructure: Amazon Elastic Block Store and Amazon EC2 instance store.

Amazon Elastic Block Store (Amazon EBS)

Amazon EBS provides persistent block-level storage volumes for use with Amazon EC2 instances in the AWS cloud. Each Amazon EBS volume is automatically replicated within its Availability Zone to protect you from component failure, offering high availability and durability. Amazon EBS volumes offer the consistent and low-latency performance needed to run your workloads. Amazon EBS volumes provide a great design for systems that require storage performance variability.

There are two types of Amazon EBS volumes you should consider for MongoDB deployments:

- General Purpose (SSD) volumes offer single-digit millisecond latencies, deliver a consistent baseline performance of 3 IOPS/GB to a maximum of 10,000 IOPS, and provide up to 160 MB/s of throughput per volume.
- Provisioned IOPS (SSD) volumes offer single-digit millisecond latencies, deliver a consistent baseline performance of up to 30 IOPS/GB to a maximum of 20,000 IOPS, and provide up to 320 MB/s of throughput per

volume, making it much easier to predict the expected performance of a system configuration.

At minimum, using a single EBS volume on an Amazon EC2 instance can achieve 10,000 IOPS or 20,000 IOPS from the underlying storage, depending upon the volume type. For best performance, use EBS-optimized instances.[9] EBS-optimized instances deliver dedicated throughput between Amazon EC2 and Amazon EBS, with options between 500 and 4,000 megabits per second (Mbps) depending on the instance type used.

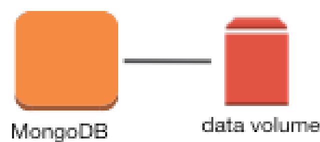

Figure 1: Using a Single Amazon EBS Volume

To scale IOPS further beyond that offered by a single volume, you could use multiple EBS volumes. You can choose from multiple combinations of volume size and IOPS, but remember to optimize based on the maximum IOPS supported by the instance.

Figure 2: Using Multiple Amazon EBS Volumes

In this configuration, you may want to attach enough volumes with combined IOPS beyond the IOPS offered by the EBS-optimized EC2 instance. For example, one Provisioned IOPS (SSD) volume with 16,000 IOPS or two General Purpose (SSD) volumes with 8,000 IOPS striped together would match the 16,000 IOPS offered by c2.4xlarge instances.

Instances with a 10 Gbps network and enhanced networking can provide up to 48,000 IOPS and 800 MB/s of throughput to Amazon EBS volumes.[10] For example, with these instances, five General Purpose (SSD) volumes of 10,000 IOPS each can saturate the link to Amazon EBS.

Amazon EBS also provides a feature for backing up the data on your EBS volumes to Amazon S3 by taking point-in-time snapshots. For information about using the snapshots for MongoDB backups, see the Backup Using Amazon EBS snapshots section of this paper.

MongoDB Management Service (MMS) provides continuous incremental backups and point-in-time recovery. For more information, refer to the MMS backup section.

Amazon EC2 Instance Store

Many Amazon EC2 instance types can access disk storage located on disks that are physically attached to the host computer. This disk storage is referred to as an *instance store*. If you're using an instance store on instances that expose more than a single volume, you can mirror the instance stores (using RAID 10) to enhance operational durability. Remember, even though the instance stores are mirrored, if the instance is stopped, fails, or is terminated, you'll lose all your data. Therefore, we strongly recommend operating MongoDB with replica sets when using instance stores.

When using a logical volume manager (e.g., mdadm or LVM), make sure that all metadata and data are consistent when you perform the backup (see the section Backup - Amazon EBS Snapshots). For simplified backups to Amazon S3, you should consider adding another secondary member that uses Amazon EBS volumes, and this member should be configured to ensure that it never becomes a primary member.

There are two main instance types you should consider for your MongoDB deployments:

- I2 instances – High I/O (I2) instances are optimized to deliver tens of thousands of low-latency, random IOPS to applications. With the i2.8xlarge instance, you can get 365,000 read IOPS and 315,000 first-write IOPS (4,096 byte block size) when running the Linux AMI with

kernel version 3.8 or later, and you can utilize all the SSD-based instance store volumes available to the instance.

- D2 – Dense-storage (D2) instances provide an array of 24 internal SATA drives of 2 TB each, which can be configured in a single RAID 0 array of 48 TB and provide 3.5 Gbps read and 3.1 Gbps write disk throughput with a 2 MB block size. For example, 30 d2.8xlarge instances in a 10-shard configuration with 3 replica sets each can give you the ability to store up to half a PB of data.

You can also cluster these instance types in a placement group. Placement groups provide low latency and high-bandwidth connectivity between the instances within a single Availability Zone. For more information, see Placement Groups in the AWS documentation.[11] Instances with enhanced networking inside a placement group will provide the minimum latencies for replication due to a low-latency, 10 Gbps network with higher performance (packets per second), and lower jitter.

High Availability and Scale

MongoDB provides native replication capabilities for high availability and uses automatic sharding to provide horizontal scalability.

Although you can scale vertically by using high-performance instances instead of a replicated and sharded topology, vertically scaled instances don't provide the significant fault tolerance benefits that come with a replicated topology. Because AWS has a virtually unlimited pool of resources, it is often better to scale horizontally.

You can scale all your instances in a single location, but doing so can make your entire cluster unavailable in the event of failure. To help deploy highly available applications, Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of regions and Availability Zones. Each region is a separate geographic area. Each region has multiple, isolated locations known as *Availability Zones*. Each region is completely independent. Each Availability Zone is isolated, but the Availability Zones in a region are connected through low-latency links. The following diagram illustrates the relationship between regions and Availability Zones.

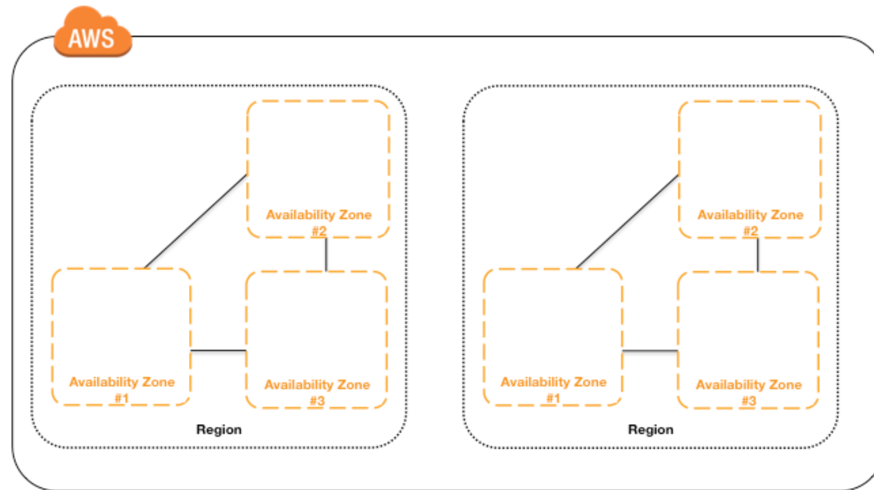

Figure 3: AWS Regions and Availability Zones

The following steps and diagrams will guide you through some design choices that can scale well to meet different workloads.

As a first step, you can separate the mongod server from the application tier. In this setup, there's no need for config servers or the mongos query router, because there's no sharding structure for your applications to navigate.

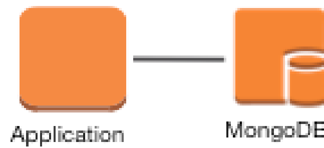

Figure 4: A Single MongoDB Instance

While the above setup will help you scale a bit, for production deployments, you should use sharding, replica sets, or both to provide higher write throughput and fault tolerance.

High Availability

As a next step to achieve high availability with MongoDB, you can add replica sets and run them across separate Availability Zones (or regions). When MongoDB detects that the primary node in the cluster has failed, it automatically performs an election to determine which node will be the new primary.

Figure 5 shows three instances for high availability within an AWS region, but you can use additional instances if you need greater availability guarantees or if you want to keep copies of data near users for low-latency access.

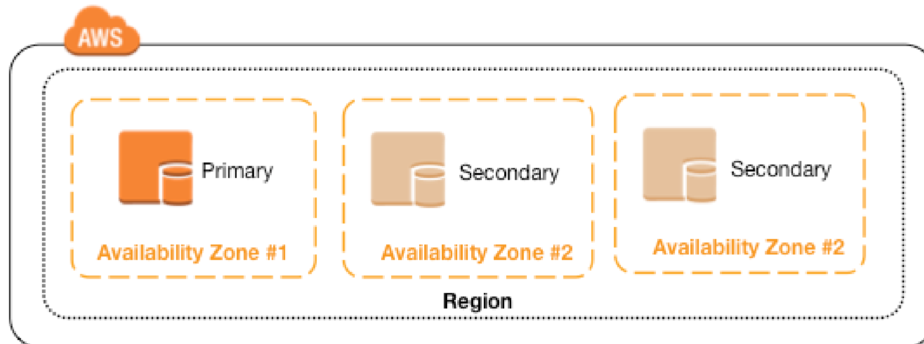

Figure 5: High Availability Within an AWS Region

Use an odd number of replicas in a MongoDB replica set so that a quorum can be reached if required. When using an even number of instances in a replica set, you should use an arbiter to ensure that a quorum is reached. Arbiters act as voting members of the replica set but do not store any data. A micro instance is a great candidate to host an arbiter node.

You can also set the priority for elections to affect the selection of primaries, for fine-grained control of where primaries will be located. For example, when you deploy a replica member to another region, consider configuring that replica with a low priority as illustrated in Figure 6. To prevent the member in another region from being elected primary, set the member's priority to 0 (zero).

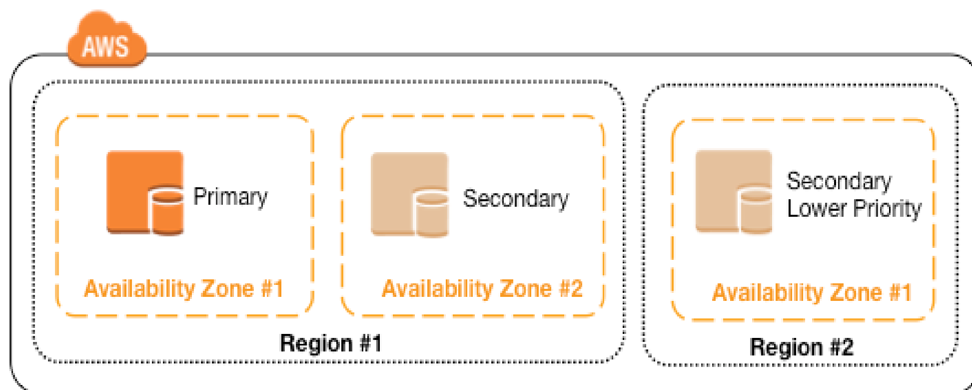

Figure 6: High Availability Across AWS Regions

You can use this pattern to replicate data in multiple regions, so the data can be accessed with low latency in each region. Read operations can be issued with a read preference mode of `nearest`, ensuring that the data is served from the closest region to the user, based on ping distance. All read preference modes except `primary` might return stale data, because secondary nodes replicate operations from the primary node asynchronously. Ensure that your application can tolerate stale data if you choose to use a non-primary mode.

Scaling

For scaling, you can create a sharded cluster of replica sets. In Figure 7, a cluster with two shards is deployed across multiple Availability Zones in the same AWS region. When deploying the members of shards across regions, configure the members with a lower priority to ensure that members in the secondary region become primary members only in the case of failure across the primary region.

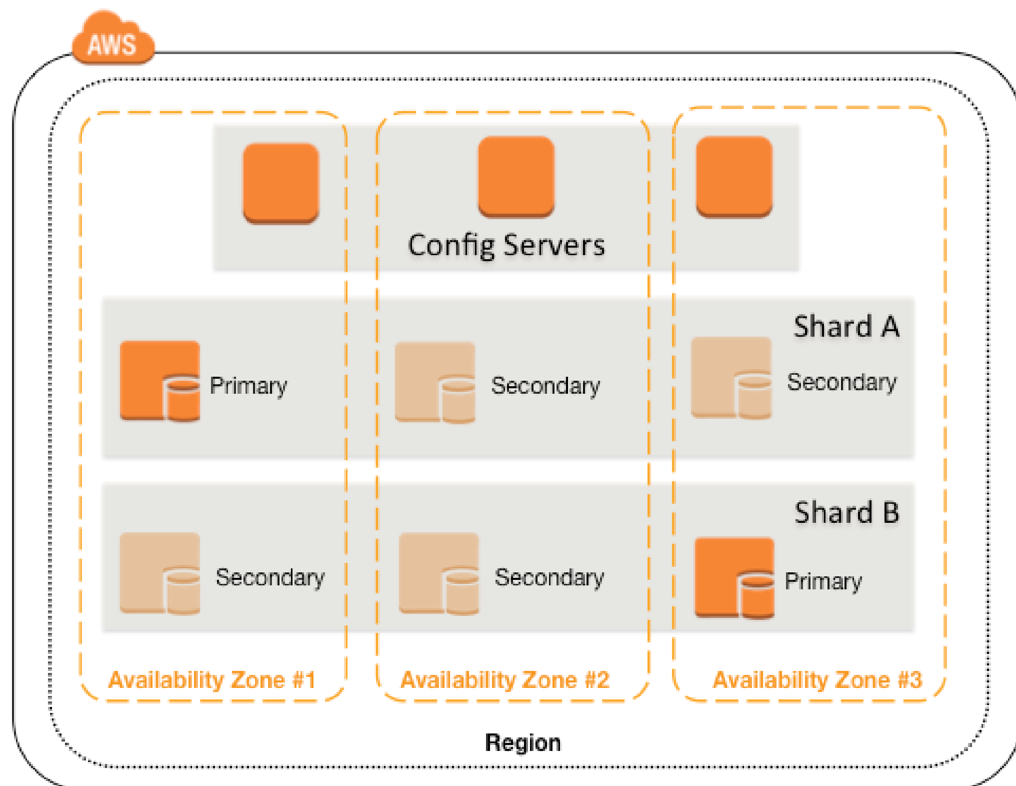

Figure 7: Scaling Within an AWS Region with High Availability

By using multiple regions, you can also deploy MongoDB in a configuration that provides fault tolerance and tolerates network partitions without data loss when using the `Majority` write concern. In this deployment, two regions are configured with equal numbers of replica set members, and a third region contains arbiters or an additional hidden replica member.

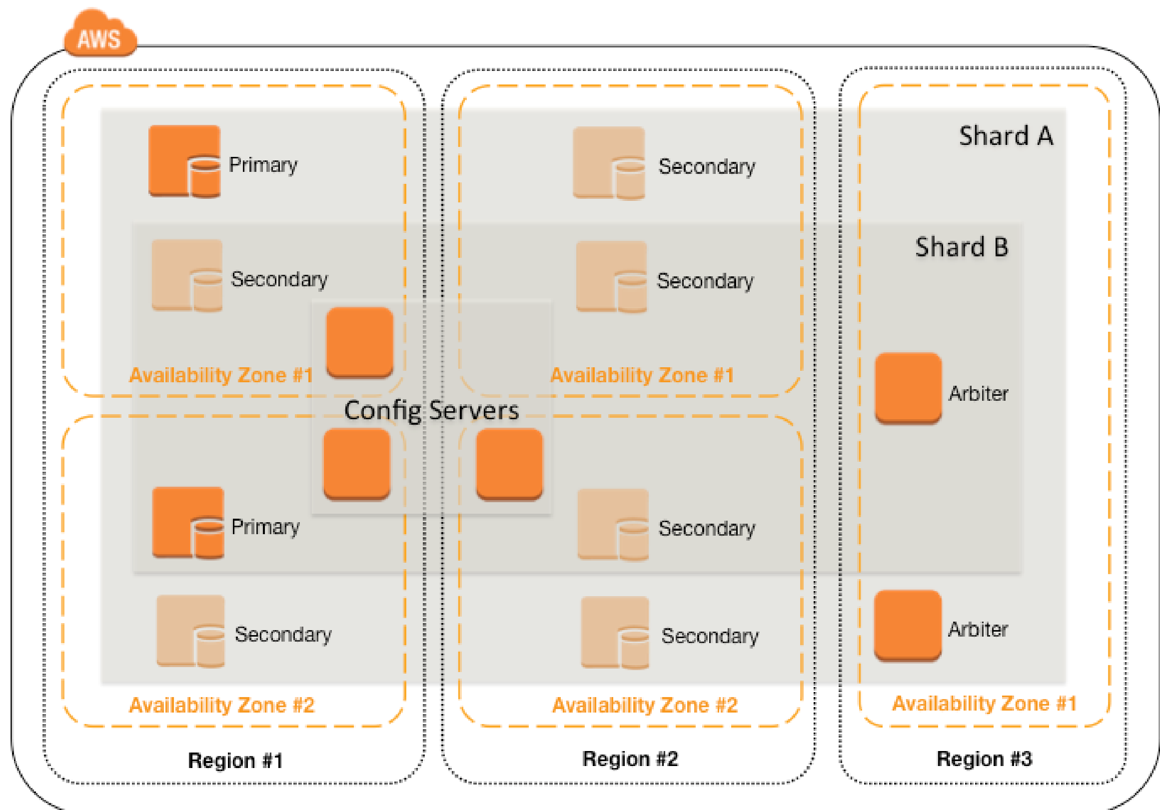

Figure 8: Scaling Across AWS Regions

MongoDB includes a number of features that allow segregation of operations by location or geographical groupings. MongoDB's tag aware sharding supports tagging a range of shard key values to associate that range with a shard or group of shards.[12] Those shards receive all inserts within the tagged range, which ensures that the most relevant data resides on shards that are geographically closest to the application servers, allowing for segmentation of operations across regions.

In a location-aware deployment, each MongoDB shard is localized to a specific region. As illustrated in Figure 9, in this deployment each region has a primary

replica member for its shard and also maintains secondary replica members in another Availability Zone in the same region and in another region. Applications can perform local write operations for their data, and they can also perform local read operations for the data replicated from the other region.

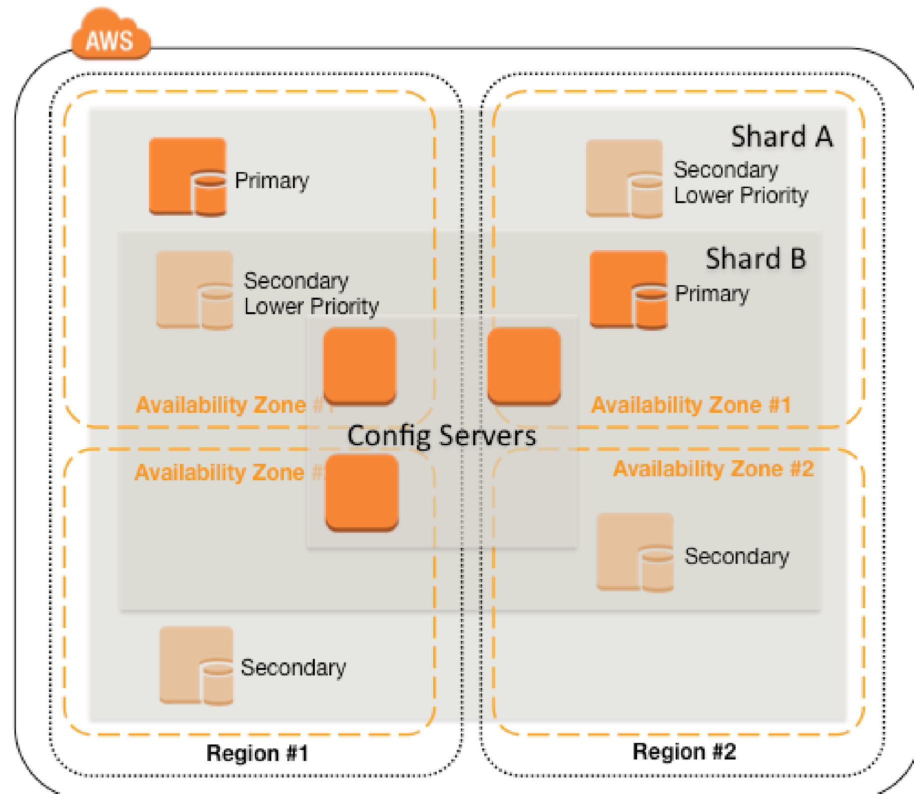

Figure 9: Scaling Across Regions with Location Awareness

MongoDB: Operations

Now that we have looked into the availability and performance aspects, let's take a look at the operational aspects (deployment, monitoring and maintenance, and network security) of your MongoDB cluster on AWS.

There are two popular approaches to managing MongoDB deployments:

- Using MongoDB Management Service (MMS), which is a cloud service, or Ops Manager, which is management software you can deploy, to provide monitoring, backup, and automation of MongoDB instances.

- Using a combination of tools and services such as AWS CloudFormation and Amazon CloudWatch along with other automation and monitoring tools.

The following sections discuss these two options as they relate to the scenarios outlined in previous sections.

Using MMS or Ops Manager

MMS is a cloud service for managing MongoDB. By using MMS, you can deploy, monitor, back up, and scale MongoDB through the MMS interface or via an API call. MMS communicates with your infrastructure through agents installed on each of your servers, and coordinates critical operations across the servers in your MongoDB deployment.

MongoDB subscribers can also install and run Ops Manager on AWS to manage, monitor, and back up their MongoDB deployments. Ops Manager is similar to MMS. It provides monitoring and backup agents, which assist with these operations.

You can deploy, monitor, back up, and scale MongoDB via the MMS or Ops Manager user interface directly, or invoke the RESTful APIs from existing enterprise tools, including popular monitoring and orchestration frameworks.

The next sections introduce the three key features of MMS—deployment, monitoring, and backup—very briefly. For more information, see the MMS documentation[13] and Ops Manager documentation[14] on the MongoDB website.

Deployment

With MMS or Ops Manager, you can deploy MongoDB replica sets, sharded clusters, and standalone instances quickly. They work by communicating with an automation agent installed on each server. The automation agent contacts MMS or Ops Manager and gets instructions on the goal state of your MongoDB deployment.

Users can authorize MMS via cross-account AWS Identity and Access Management (IAM) roles to allow MMS to provision Amazon EC2 instances on

AWS and start the MMS automation agent. Hence, deploying MongoDB on AWS is even simpler when you use MMS.

In addition to getting support for initial deployment, you can also dynamically resize capacity by adding shards and replica set members.

Monitoring

MMS and Ops Manager monitoring provides real-time reporting, visualization, and alerting on key database and hardware indicators. A lightweight Monitoring Agent runs within your MongoDB deployment and collects statistics from the nodes in your MongoDB deployment. The agent transmits database statistics back to MMS or Ops Manager to provide real-time reporting. You can set alerts on the indicators you choose.

Backup

MMS and Ops Manager are the only solutions that offer point-in-time backups of replica sets and cluster-wide snapshots of sharded clusters. A lightweight Backup Agent runs within your infrastructure and backs up data from the MongoDB processes you have specified.

The Backup Agent conducts an initial sync and then tails the operation log (oplog) to provide a continuous backup. Because the Backup Agent only reads the oplog, it minimizes the performance impact on the cluster and is similar to adding an additional replica to a replica set.

These 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.

Do It Yourself

Deployment – AWS MongoDB Quick Start

AWS Quick Start reference deployments help you deploy fully functional enterprise software on the AWS cloud, following AWS best practices for security and availability.[15]

The MongoDB Quick Start automatically launches and runs a MongoDB cluster of up to 13 nodes on AWS. It automates the deployment through an AWS

CloudFormation template, and enables you to launch the MongoDB cluster either into your own Amazon Virtual Private Cloud (Amazon VPC) or into a newly created Amazon VPC. Customization options include the MongoDB version you want to deploy (version 2.6 or 3.0), the number of replicas you want to launch to ensure high availability (1-3 replicas), and the number of shards you want to use to improve throughput and performance (0-3 shards). The Quick Start also provides micro-sharding options and lets you customize storage types and sizes.

The MongoDB Quick Start takes approximately 15 minutes to deploy. You pay only for the AWS compute and storage resources you use—there is no additional cost for running the Quick Start.

For more information about the MongoDB architecture and implementation, and to launch the Quick Start, see the Quick Start deployment guide.[16]

Monitoring – Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, to collect and monitor log files, and to set alarms. Amazon CloudWatch can send an alarm by SMS or email when user-defined thresholds are reached on individual AWS services. For example, you can set an alarm to warn of excessive storage throughput.

CloudWatch metrics can also be used to drive policies for Auto Scaling groups to automatically scale your compute resources up or down based on your custom metrics, as shown in Figure 10.[17]

amazon webservices

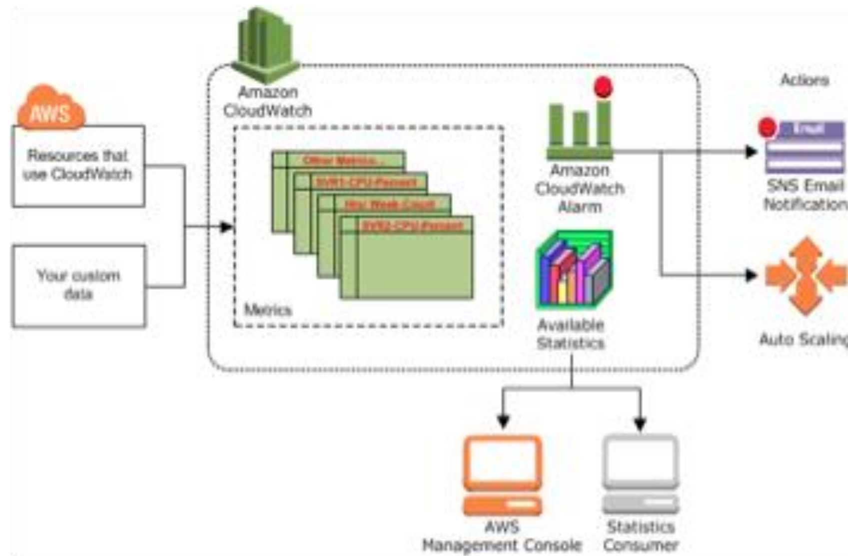

Figure 10: Monitoring Using Amazon CloudWatch

Alternatively, you can write a custom metric and submit it to Amazon CloudWatch for monitoring. For example, you can write a custom metric to check for current free memory on your instances, and to set alarms or trigger automatic responses when those measures exceed a threshold that you specify.

To publish metrics regarding MongoDB into CloudWatch you should use IAM roles to grant permissions to your instances. Here's an example IAM policy you can use:

```
{
  "Version": "2012-10-17",
    "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricData",
        "ec2:DescribeTags"
      ],
      "Resource":  "*"
    }
```

```
  ]
}
```

You can then use the AWS command line interface (AWS CLI) to publish any metrics directly into Cloudwatch. The following example demonstrates how to publish a simple metric named `ResidentMemory` into Cloudwatch with a value of 32 GB.

```
aws cloudwatch put-metric-data --metric-name ResidentMemory
--namespace MongoDB --timestamp 2014-01-01T00:00:00Z --
value 32 --unit Gigabytes
```

The monitoring section of the MongoDB manual also provides a good overview of the monitoring and reporting utilities available.[18]

Backup – Amazon EBS Snapshots

You can back up the data on your EBS volumes to Amazon S3 by taking point-in-time snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved. When you delete a snapshot, only the data exclusive to that snapshot is removed. Active snapshots contain all the information needed to restore your data (from the time the snapshot was taken) to a new EBS volume.

Taking a consistent Amazon EBS snapshot also depends upon the capabilities of a specific operating system or file system. An example of a file system that can flush its data for a consistent backup is XFS (see the xfs_freeze command[19]). When using a logical volume manager such as mdadm or LVM, you should perform the backup from the volume manager layer rather than the underlying devices. This ensures that all metadata remains consistent and that the various subcomponent volumes are coherent. You can take a number of approaches to accomplish consistency. For example, you can use the script made available by Alestic.com at https://github.com/alestic/ec2-consistent-snapshot.

To get a correct snapshot of a running mongod process:

- You must have journaling enabled. Otherwise, there is no guarantee that the snapshot will be consistent or valid.
- The journal should reside on the same logical volume as the data files, or you must use the MongoDB `db.fsyncLock()` method to capture a valid snapshot of your data.

To get a consistent snapshot of a sharded system, you must disable the balancer and capture a snapshot from every shard and config server at approximately the same moment in time, using the method outlined previously in this section.

For more information about backup and recovery of MongoDB deployments, see the Backup and Recovery section of the MongoDB manual.[20]

Network Security

The following sections provide a brief overview of using network security with MongoDB on AWS.

Amazon Virtual Private Cloud (Amazon VPC)

Amazon VPC enables you to create an isolated portion of the AWS cloud and launch Amazon EC2 instances that have private (RFC 1918) addresses in the range of your choice (e.g., 10.0.0.0/16). You can define subnets within your Amazon VPC, grouping similar kinds of instances based on IP address range, and then set up routing and security to control the flow of traffic in and out of the instances and subnets. If a subnet's traffic is routed to an Internet gateway, the subnet is known as a *public subnet*; otherwise, it's called a *private subnet*.

Deploying your MongoDB cluster into an Amazon VPC with a private subnet and configuring your security group to permit ingress over the appropriate TCP ports builds another layer of network security. Like any network service, these ports should be opened conservatively; for example, open them only to your corporate office (over an IPsec VPN tunnel) or to other authenticated and authorized machines. The following table shows default TCP port numbers for MongoDB processes.

Process	Port
27017	The default port for mongod and mongos instances. You can change this port with the `port` option in a configuration file or `--port` runtime operation.
27018	The default port when running with the `--shardsvr` runtime operation or the `shardsvr` value for the `clusterRole` setting in a configuration file.
27019	The default port when running with the `--configsvr` runtime operation or the `configsvr` value for the `clusterRole` setting in a configuration file.
28017	The default port for the web status page. This page is always accessible at a port number that is 1000 greater than the port used by mongod.

A common approach is to create a MongoDB security group that contains the nodes of your cluster. To ensure that only your app servers can connect to your MongoDB instances, add a rule in your MongoDB security group with the source field set to the security group name that contains your app servers, and the port set to 27017.

Ensure that the HTTP status interface, the REST API, and the JSON API are all disabled in production environments to prevent potential data exposure and vulnerability to attackers.

Deploying Across Availability Zones

You can create an Amazon VPC that spans multiple Availability Zones, but each subnet must reside entirely within one Availability Zone and cannot span zones. To deploy MongoDB across separate Availability Zones, you should create one Amazon VPC with private subnets for each zone. A common security group can be used to allow the instances across the subnet to communicate.

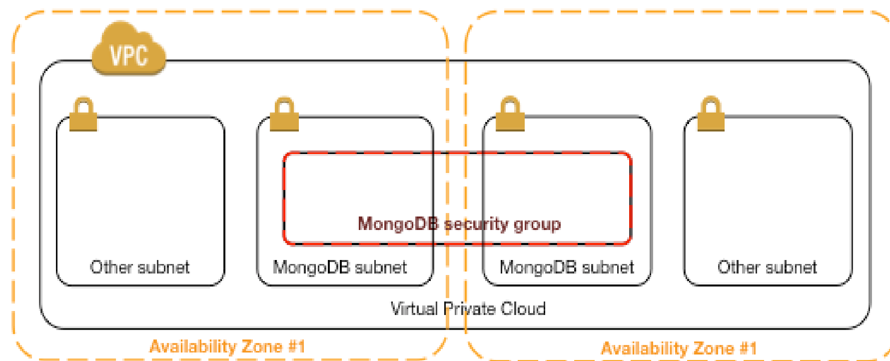

Figure 11: Deploying MongoDB in an Amazon VPC

Deploying Across Regions

For MongoDB deployments that span regions, you would need to create an Amazon VPC in each region and allow communication on ports listed via security groups. Security groups are defined within the scope of an Amazon VPC, so you would need to use multiple security groups to secure access within and across regions.

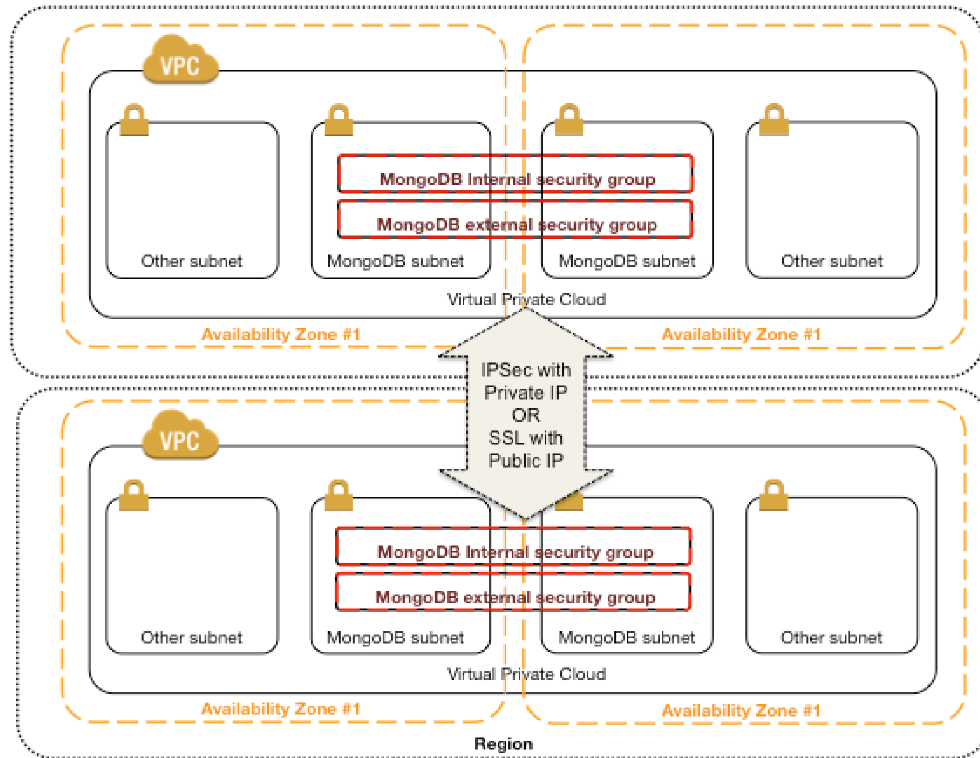

Figure 12: Deploying MongoDB Across Regions

In this deployment, you should also secure the network communications between regions by using one of the following methods:

- Secure IPsec tunnel to connect multiple Amazon VPCs into a larger virtual private network that allows instances in each Amazon VPC to seamlessly connect to each other using private IP addresses. For more information, see the article Connecting Multiple VPCs with EC2 Instances (IPSec) on the AWS website.[21]
- SSL to encrypt MongoDB's entire network traffic and allow instances to connect to each other using public IP addresses. For more information, see Configure mongod and mongos for SSL in the MongoDB documentation.[22]

Conclusion

The AWS cloud provides a unique platform for running NoSQL applications, including MongoDB. With capacities that can meet dynamic needs, costs based on use, and easy integration with other AWS products such as Amazon CloudWatch, AWS CloudFormation, and Amazon EBS, the AWS cloud enables you to run a variety of NoSQL applications without having to manage the hardware yourself. MongoDB, in combination with AWS, provides a robust platform for developing scalable, high-performance applications.

Further Reading

For additional help, please consult the following sources:

- Amazon EC2 FAQ
- Amazon EBS Overview
- Amazon VPC Documentation
- AWS Security Center
- Security section of the MongoDB manual
- Production Notes section of the MongoDB manual
- MongoDB MMS documentation
- MongoDB Architecture Guide
- Performance Best Practices for MongoDB
- MongoDB Multi-Data Center Deployments

Notes

[1] http://aws.amazon.com

[2] https://www.mongodb.org/

[3] http://aws.amazon.com/dynamodb/

[4] http://aws.amazon.com/s3/

[5] http://docs.mongodb.org/master/administration/production-notes/

[6] http://docs.mongodb.org/master/security/

[7] http://docs.mongodb.org/manual/core/read-preference/

[8] http://docs.mongodb.org/manual/core/write-concern/

[9] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSOptimized.html

[10] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html

[11] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html

[12] http://docs.mongodb.org/manual/core/tag-aware-sharding/

[13] https://docs.mms.mongodb.com/

[14] https://docs.opsmanager.mongodb.com/

[15] http://aws.amazon.com/quickstart/

[16] https://s3.amazonaws.com/quickstart-reference/mongodb/latest/doc/MongoDB_on_the_AWS_Cloud.pdf

[17] http://aws.amazon.com/documentation/autoscaling/

[18] http://docs.mongodb.org/manual/administration/monitoring

[19] http://linux.die.net/man/8/xfs_freeze

[20] http://docs.mongodb.org/manual/administration/backup/

[21] https://aws.amazon.com/articles/5472675506466066

[22] http://docs.mongodb.org/manual/tutorial/configure-ssl/