
mongoDB.

Contents

1. Abstract:

The evolution of data on a type and size basis has led to a need to find a dynamic way to stand for and manipulate it. For that purpose, document-oriented is recognized as one of the best ways to handle big data.

In the last decade, the internet grows and especially IoT (internet of things), where plenty of sensors are spreads in the virtual world. Humanity faces an unusual number of data source types and increases in volume and speed. Nowadays, to interprets these three vector representations, a new term is published, which is bigdata. Big data refers to data with huge volume, high velocity, and variant source types. We can show what we are manipulating by explaining those three vectors:

- Volume: nowadays, collecting data from various sources is essential for companies and organizations to understand their needs and understand the market and raise their profit. Various sources like IoT, smart devices, business transactions, and social media produce data. Those data grow exponentially to reach more than a petabyte.
- Velocity: The amount of data is not the only problem. Non-stop internet growth needs to be processed on the fly and near real-time to be productive.
- Variety: finally, those diverse sources produce several types and formats of data: videos, emails, structured, semi-structured, or fully unstructured documents.

To handle big data applications modern design concept needs to be applied. Storing, processing, and analyzing to extract efficient knowledge from this data is a big data discussion trend. Big data is

one of the main attractive subjects in different business fields. Bigdata applications should deal with a vast volume and combination of data proficiently with its unpredictable nature. New querying strategies need to be defined no place to RDBMS systems when dealing with bigdata as we cannot formalize that data in a schema-based way or any specific predefined form. For that and to respect all three vectors, we need a new type of database. This database must be capable of handling the volume of data with a horizontal scale, give elasticity by taking generated data with no specific structure, and at the same time be able to model data with a de-normalized way to perfect analysis. This new database is known as the NoSQL databases. In our project, we will introduce one of the most known NoSQL databases, which is MongoDB.

1. Introduction:

MongoDB is an open-source, document-oriented, and distributed NoSQL database that stores many data and provides the end-user with high performance and scalability. The user can write with high throughput and retrieve data very efficiently, working in milliseconds and delivering quickly. Mainly the word MongoDB came from Humongous, which reflects its idea of storing and processing many data.

MongoDB is characterized by the absence of a design to be entities, as each entity in the entity set may have different confidential information from other entities in the same set.

Document-oriented databases store data like documents, and documents basically represent a record, which is a data structure composed of field and value pairs. Documents in MongoDB are remarkably like objects in JSON.

Another advantage is that using embedded documents and arrays reduces the need for expensive joins.

Documents are stored in collections, standing for a group of documents that resemble a table in the relational database. In contrast, it differs in the design scope, as documents are schema-less, which means that the user can have one collection (table) that where different data are stored, having flexibility in storing mixed data, a great advantage for growing businesses that do not know the kind of data need to gather in a timescale, but they need to start directly.

Figure 1: MongoDB document

Another fundamental aspect of MongoDB is its large features such as:

- High Performance data persistence
- Rich Query language
- High Availability
- Sharding distribution and scalability
- Support for multiple storage engines

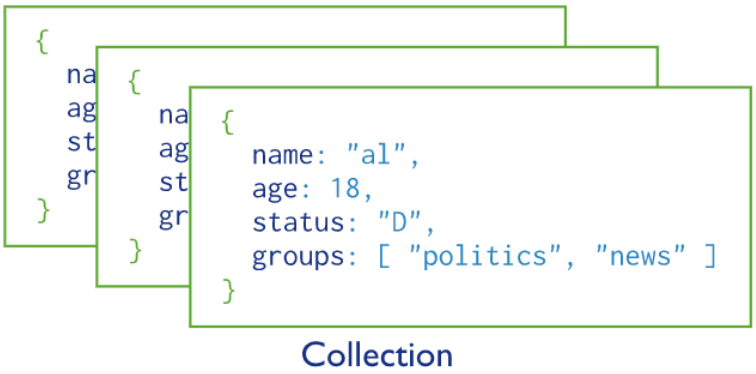

Figure 2: Mongo DB Collection

As a framework, MongoDB presents different cloud database services like Atlas, development deployers like Realm, Atlas Data Lakes, allowing users to natively query and analyze data across platforms like AWS S3 and MongoDB Atlas.

2. MongoDB and CAP Theorem

The CAP theorem is a mechanism applied to makes system designers informed of the trade-offs while creating networked shared-data systems. The Cap has inspired the design of many distributed data systems. The cap theorem concerns distributed systems that store state, a trade-off between consistency, availability, and partition tolerance. the theorem states can only guarantee strongly support two of the following three properties:

- Consistency: guarantee that each node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having an identical view of the data. There are several types of consistency models. Consistency in the cap (used to prove the theorem) refers to linearizability or sequential consistency, a powerful consistency class.

- Availability: every non-failing node delivers a reply for all read and writes requests in a fair amount of time. Availability is the Keyword in this scenario. Every node on the network partition reacts in a fair time window.

- partition tolerant: the system stays performing and supports its consistency guarantees despite network partitions. network partitions are a fact. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

MongoDB is built on the principles of the CAP Theorem. It is strongly consistent by default - if we make a write and then execute a read, considering the write was successful, we will always be able to read the result. MongoDB is a single-master system. Thus, all reads go to the primary by default. If we optionally allow reading from the secondaries, then

MongoDB becomes ultimately consistent where it is possible to read out-of-date results.

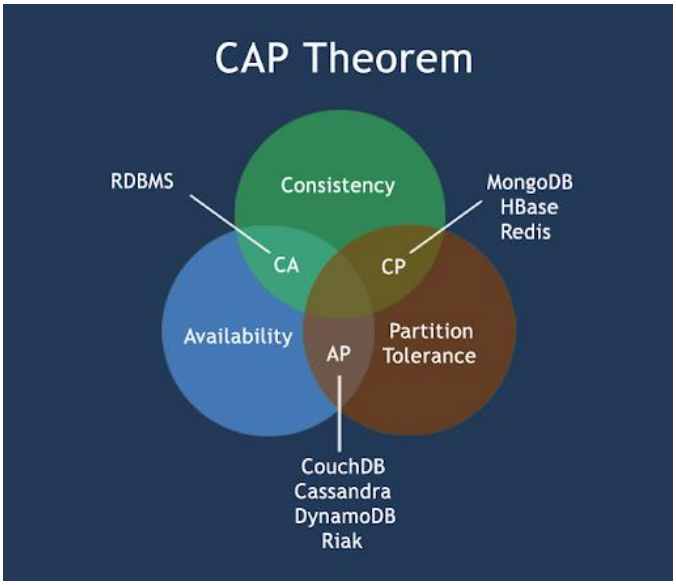

Figure 3: MongoDB CAP

On another scale, we can MongoDB can also be considered CP/AP/CA since it can be a trade-off between C, A, and P, depending on the database's configuration and practice type. To Sum it up, we can say:

Practice	Focus	Description
No partition	CA	The system is available and supplies strong consistency
partition, majority connected	AP	Not synchronized writes from the old primary are ignored
partition, majority not connected	CP	only read access is supplied to avoid separated and inconsistent systems

2.2 distribution of CAP focus for MongoDB

- Consistency:

Using a single connection and correct read-write process, MongoDB becomes consistent (Which will cost execution speed). As soon as we match those states (especially when we are reading from a secondary replica), MongoDB becomes Irresistible.

- Availability:

MongoDB gets high availability through Replica-Sets. As soon as the primary goes down or gets unavailable else, then the secondaries will decide a new primary to become available again. There is a disadvantage to this: “Every write that was performed by the old primary, but not synchronized to the secondaries will be rolled back and saved to a rollback-file, as soon as it

reconnects to the set (the old primary is a secondary now)". So, in this case some consistency is sacrificed for availability.

- Partition Tolerance:

Replica-Sets MongoDB also achieves the partition tolerance: If more than half of the Replica-Set servers are connected, we need to choose a new primary key. Why? To ensure two separated networks cannot both choose a new primary. When not enough secondaries are connected, we may lose some consistency level, but we still can read from them without writing. The set is almost unavailable for the sake of consistency.

3. Consistency

MongoDB is consistent by default reads and writes are issued to the primary member of a replica set. Applications can optionally read from secondary replicas, where data is eventually consistent by default. Reads from secondaries can be useful in scenarios where it is acceptable for data to be slightly out of date, such as some reporting applications. Applications can also read from the closest copy of the data (as measured by ping distance) when latency is more important than consistency.

4. Architecture

- Database:

In simple words, database is a physical box of data. Each database has its own set of files on the file system, with multiple databases being on a single MongoDB server.

- Collection:

A group of database documents can be called a collection. The RDBMS equivalent to a collection is a table. The complete collection exists within a single database. There are no schemas when it comes to collections. Inside the collection, various documents can have varied fields, but mostly the documents within a collection are meant for the same purpose or for serving the same end goal.

- Document:

A set of key-value pairs can be appointed as a document. Documents are associated with dynamic schemas. The benefit of having dynamic schemas is that a document in a single collection does not have to own the same structure or fields. Also, the common fields in a collection's document can have varied types of data.

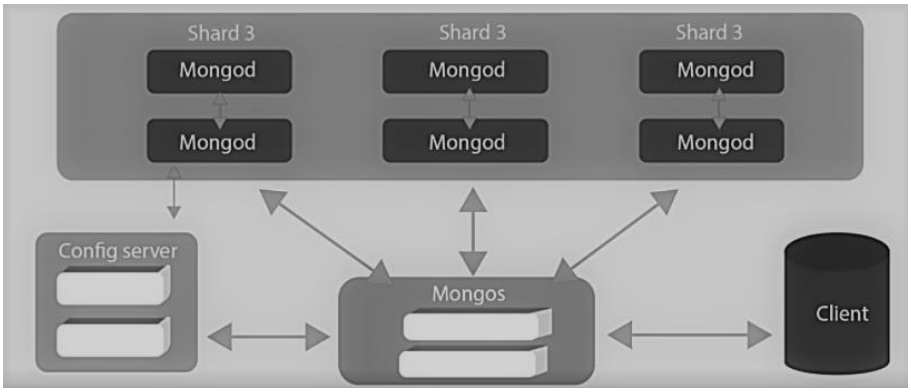

Figure 4: MongoDB basic architecture

5. Deployment

Deployment is the mechanism through which applications, modules, updates, and patches are delivered from developers to users. The methods used by developers to build, test, and deploy new code will affect how fast a product can respond to changes in customer preferences or requirements and the quality of each change.

Before deploying MongoDB, the user should pay attention to the system requirements, such as 10GB of free disk space plus a necessary space to hold the MongoDB data. Also, at least 4 GB of RAM, in addition to networking requirements and a backup support.

The user must understand the way his application is going to hold up in production, where it is good to develop a consistent, repeatable approach to managing the deployment environment to minimize errors while in production.

The best approach includes prototyping the setup, conducting testing a standalone, checking key metrics, and using the information to scale up the setup. This shall help to understand the hold of the production system before deployment, which also a user shall discuss some questions before trying with it, just as:

- What are the best practices of deployment?
- What are the key metrics that need to be checked to ensure the application is meeting its required service levels?
- How will you know when it’s time to add shards?
- What tools do you have to backup and restore the database?
- And what about securing access to all that new real-time big data?

When checking the deployment, a user shall know the working set, as MongoDB makes extensive use of RAM for low latency operations. In

MongoDB, all data is read and manipulated through memory-mapped files. Reading data from memory is measured in nanoseconds and reading data from disk is measured in milliseconds; and so, reading from memory is approximately 100,000 times faster than reading from disk.

- Deploy a Replica Set

A replica set is a group of MongoDB deployments that keep the same data set. Replica sets provide redundancy and high availability and are the basis for all production deployments.

- Deploy a Sharded Cluster

Sharded clusters provide horizontal scaling for large data sets and enable high throughput operations by distributing the data set across a group of servers.

- Deploy a Standalone MongoDB Instance

A standalone MongoDB instance can be deployed for Cloud Manager to manage. Standalone instances are used for testing and development and not for production systems as they lack replication and high availability.

6. Replication and Redundancy

Redundancy cannot be said, before mentioning the replication phase, which is the process of synchronizing data across multiple servers.

Replication provides redundancy and besides sharding, increases data availability with multiple copies of data on different database servers, it also protects a database from the loss of a single server, allows the user to recover from hardware failure and service interruptions and with added copies of the data, the user could dedicate one tier to disaster recovery, reporting, or backup.

Then summing up, we can refine that this mechanism will ensure that data is available in case of any node failure. We can configure the replication in two different scenarios from the perspective of the servers' roles:

- master-slave mode:
- in this case, we have a server that plays the role of master, and the other node in the cluster works as a slave. Slave node listens to the master to update their content, and all the communication with the application are the responsibility of the master node. In case of mater failure, one of the slave nodes is picked as master.

- Peer-to-Peer mode:

  in this mode, all nodes have equal responsibilities and help requests from the application and synchronize continuously.

Why do we need replication?

- To keep your data safe
- High 24*7 availability of data
- Disaster Recovery
- No downtime for maintenance
- Read scaling
- Replica set is transparent to the application

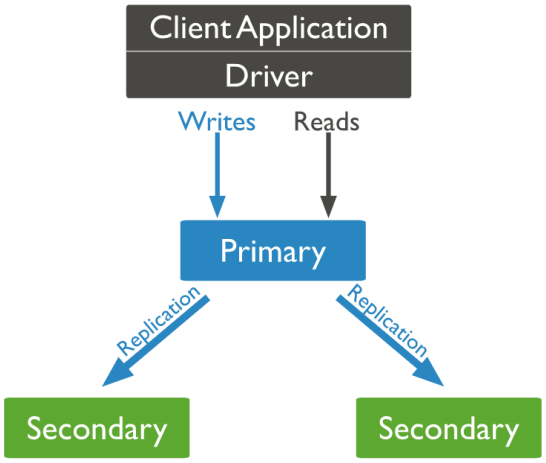

Figure 5: MongoDB Replication

Following the above, we can suggest that A replica set in MongoDB is a group of mongoDB (the primary process of MongoDB system which handles requests) processes that support the same data set. They provide redundancy and high availability and are the basis for all production deployments.

- ## Redundancy and Data Availability

Now we can say that replication supplies redundancy and increases data availability. With multiple copies of data on different database servers, replication supplies a level of fault tolerance against the loss of a single database server.

In some cases, replication can provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data in different data centers can increase data locality and availability for distributed applications.

"We can rely on the article of Anaconda- MongoDB failover and Redundancy, 2019:
Anaconda Enterprise Notebooks uses MongoDB as its database to stores information on users, projects, gateways, and compute nodes. In an Enterprise environment it may be beneficial to setup MongoDB for automatic failover and redundancy".

- ## Test Failover on MongoDB Atlas

Replica set elections are necessary every time Atlas makes configuration changes as well as during failure scenarios. Configuration changes may occur because of patch updates or scaling events. As a result, the user should write your applications to be capable of handling elections without any downtime.

- ## Rollbacks During Replica Set Failover

A rollback reverts write operations on a former primary when the member rejoins its replica set after a failover. It is necessary only if the primary had accepted write operations that the secondaries had not successfully replicated before the primary stepped down. When the primary rejoins the set as a secondary, it reverts, or “rolls back,” its write operations to keep database consistency with the other members.

MongoDB tries to avoid rollbacks, which should be rare. When a rollback does occur, it is often the result of a network partition. Secondaries that cannot keep up with the throughput of operations on the former primary, increase the size and impact of the rollback.

A rollback does not occur if the write operations replicate to another member of the replica set before the primary steps down and if that member still is available and accessible to much of the replica set.

7. Sharding

One of the important techniques to handle the data growth in databases is sharding. When you start a new application, you may not understand how popular it will be or how many requests you will receive. After a while, you will see that your database cannot manipulate either the data size either the number of connections. These days, it is easy to assume from the size of datasets on the internet that you may face some datasets that cannot fit into a single Pc, and you need a more powerful one. For this reason, sharding is set up. Sharding is the concept of splitting datasets beyond multiple machines. This concept gives us the ability to manage and store more data with multiple pc with lower resources, and at the same time, more answers for requests. With this strategies subset of data is distributed on each machine. Each node of these shards is a cluster.

Mongo DB has a better approach in terms of shardings than other database software as it has auto sharding. The cluster nodes are not independent in auto sharding, which differs from manual shardings where an application independently keeps different database servers. The auto sharding in MongoDB works better in front of any data distribution change or any failure in the system. We can clarify this concept as auto-sharding architecture is independent of application compared to manual sharding, where adding or removing any cluster is hard to keep. This way shows easy management and administration. MongoDB gives your application the ability to work with multiple clusters as they are one standalone Database server.

The configuration of sharding in MongoDb is not so simple. Before deploying sharding, you must have good knowledge of working with replica and standalone servers. Simultaneously, this will add complexity on the

operational level. in the design phase, you must choose the best for your data by considering the increase of your app requests and data. Implementing clustering on small data is a waste of time and resource because that in such a way, it is critical to choose between implement sharding or not.

Sharding will increase the RAM, disk space, which reduces load and gives higher throughput from a single node. Thus, you may need to check these criteria to choose the best plan.

8. Combining Sharding and Replication

Combining the sharding technique with replication will help reach maximum benefits (availability and more reliable resources). As we mentioned in the replication part, we have two methods, peer-to-peer and master-slave. This is applied in combination as we can use each mode with the replication, which means peer-to-peer replication with sharding and master-slave replication with sharding. In the first method, we may have multiple nodes in a cluster where data is sharded and replicated over them, which is beneficial for column-family databases. In the second scenario or master-slave, we may have multiple master nodes divided by data items, where the data item and the master node must be unique. Still, no problem to have a joint role where a master node for some data can be a slave node for others.

9. MongoDB in bigdata architecture

As we mentioned in our introduction in big data use cases, we face one of three vectors. After finding the architecture of MongoDB, we can clearly

prove that this type of database has a solution beyond those three vectors. we can see these three use cases:

- huge data need to be stored and processed.
- Large sets of multiple data types, structured, semi-structured, or unstructured, need to analyze and report by implement Aggregation and transformation.
- Stream of data that we need to analyze on the fly (near real-time or real-time).

The most critical and important case study nowadays is receiving data from multiple sources with diverse types. At in same time, we may have a union of input sources such as relational databases with applications such as ERP or CRM, data warehouses, mobile devices, social media, email, and real-time streaming data inputs such as IoT devices.

After receiving data, it is time to ingest it into the storage system. Here is the time for MongoDB, which manipulates unstructured data, and we may use a relational database to work with structured data. We may need to handle received data in two ways. Some need batch process after store and Hadoop will manipulate it, and the second needs to be work on real-time which manipulated by queuing and buffering it by apache Strom and apache spark. The prepared data can now be manifested in a structured format and we can by query it by analytical tools like BI (business intelligent) platform or mongodb. finally, by extracting useful business data to get knowledge

from it and get readable result for reporting and analysis.

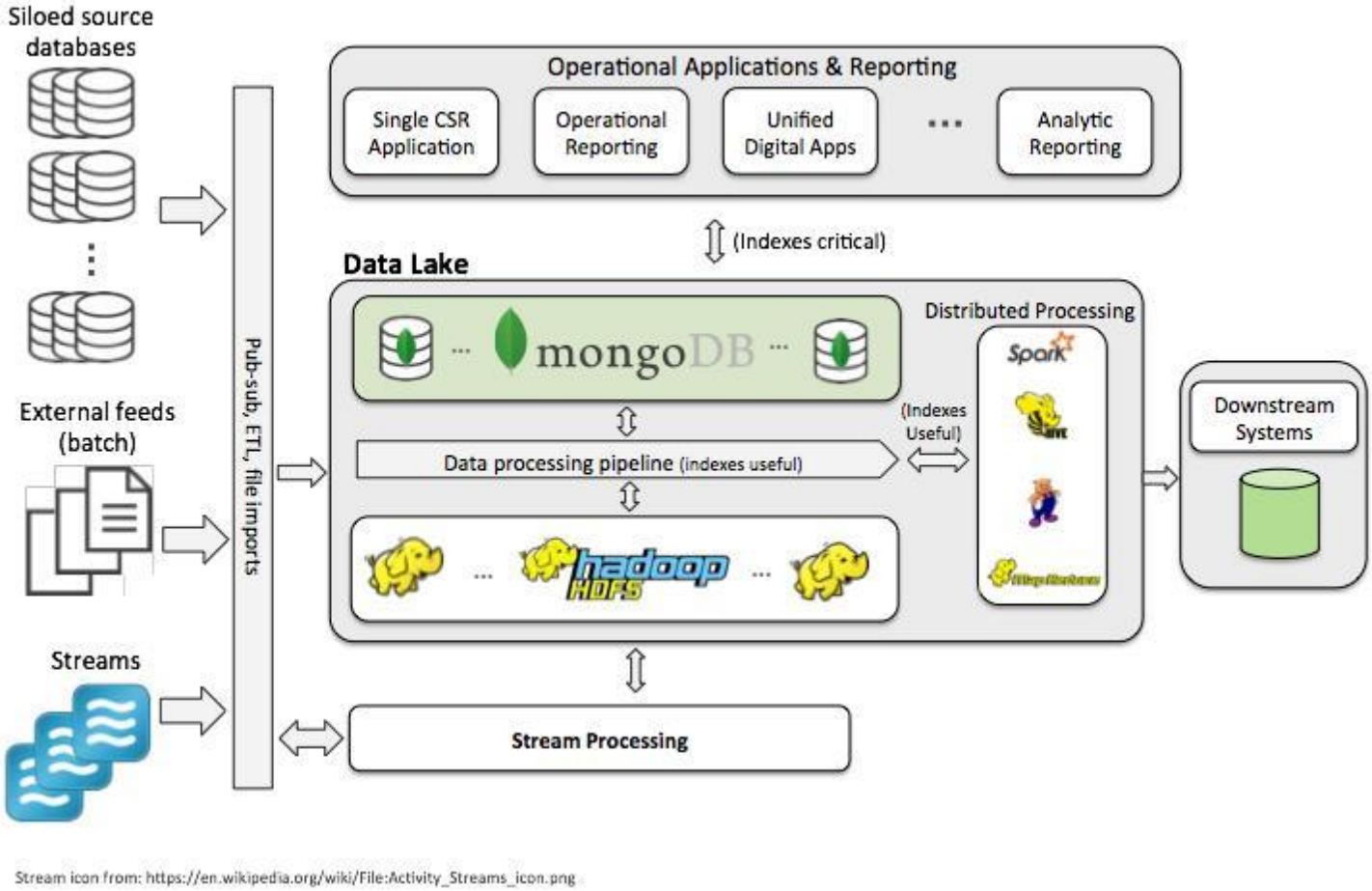

Figure 6: MongoDB in a Big Data Architecture

10. Security

- Data Security Components

The information system is the complete set of software, hardware, data, people, procedures, and networks necessary to use information as a resource for the organization. As a result, to secure an information system, we must secure all the components of this set. The protection of information is a process that cannot be done without protecting its entire elements, including software, hardware, and systems that use, store, and send it. To implement these securities, necessary tools must be applied to training, education, policies, awareness, and technology. Information security is defined as a CIA triangle, Confidentiality, Integrity, and Availability. Those are the important components, and we have some other critical characteristics that under those categories, like authentication and authorization.

As we have said before, MongoDB is a NoSQL and document architecture design, which makes it more flexible than relational databases. Hence, a huge data volume is stored in MongoDB than in traditional SQL databases. MongoDB databases exceed a petabyte of data. This large amount of data exposed in a single database makes data breaches more destructive and critical for a company. In that circumstance, MongoDB supplies various features and actions to avoid any breach and increase security measures. These security features must be implemented to secure the MongoDB environment. On that interest, in this section, we will be simply introducing each feature to get the best practice.

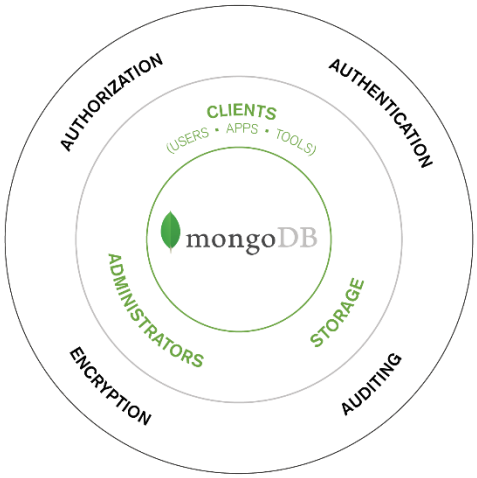

Figure 7: mongoDB security diagram

- ## Authentication in MongoDB

AS mentioned in the security part, part Authentication is one of the critical. Databases nowadays, especially mongo with its Atlas version, are exposed to the internet, making those databases a target for a hacker. On the other hand, the NoSql database's nature and its features in handling a massive volume of data make any breach or illegal access to those systems more critical for the companies and a great interest for any hacker. Hackers exploit vulnerabilities in such systems. Human is the weaker part of the system and the reason behind multiple attacks. We can find multiple configurations to secure such a database, but the problem is configuring it. Data scientists and database administrators have low knowledge of security, or the majority have no concern. That leads to a misconfiguration like not enabling authentication or encryption or simply using a common or simple password that can be brute forced.

MongoDB supports authentication through the database level. Authentication exists as you can create a user for every single logical database. However, it does not support items like password complexity,

age-based rotation, and centralization and identification of user roles versus service functions. Many of these features can be carried out using LDAP (Lightweight Directory Access Protocol), filling many of these gaps.

"However, LDAP support is available in MongoDB Enterprise, but not in MongoDB Community Edition."

- ## Authorization in MongoDB

We can enable access control in MongoDB to force authentication and oblige users to find themselves. In this way, each user has a privilege level that can reach a package of commands that can execute automatically. Replica sets and sharded clusters also require internal authentication between members. Permissions and privileges must provide concerning the job where at most limited required. The database administrator must use low privilege uses when no need to change. In other words, they must check what they need from accessing the database and go with that privilege. MongoDB has a built-in role that covers the most common authorization access level, which you can use when you do not have the needed knowledge, and at the same time, we can define our new user role.

```
db.createRole({
role : 'write_foo2_Collection',
privileges : [ {resource : {db : "percona", collection : "foo2"}, actions : ["insert","remove"]}
],
roles : ["read"]
})

db.updateUser('client_read', roles : ['write_foo2_Collection'])
```

- ## Transport Encryption in MongoDB

MongoDB has support for using transport encryption between the client and the nodes, and between the nodes in the cluster. Encrypting traffic ensures that no one can "sniff" sensitive data on the network. For example, packet analyzer tools like (Wireshark or TCPdump) can easily capture unencrypted sensitive data such as usernames and passwords.

MongoDB supports X.509 certificate authentication for use with a secure TLS/SSL connection. Members can use X.509 certificates to verify their membership in the replica set and shards.

It is necessary to create certificates on all nodes and have a certificate authority (CA) that signs them. As using a CA can be costly, it is also possible to use self-signed certificates. Using a public CA is not necessary inside a private infrastructure.

- ## Data Encryption in MongoDB

One of the most severe problems with MongoDB was that data files didn't have encryption at rest. Since version 3.6.8, Percona Server for MongoDB has offered at rest encryption for the MongoDB Community Edition. In upstream MongoDB software, data encryption at rest is available in MongoDB Enterprise version only.

- ## Auditing in MongoDB

Auditing is not produced to moderate a security threat but helps when investigating unauthorized access or tracking data access and modification. The general database auditing concept is about tracing the use of database records and authority. When we audit a database, we check each operation and access in a data audit trail. This covers information about:

1. which touch database object or data record,
2. which account acted,
3. when the activity occurred.

On another hand, a user can also take into consideration added tips:

1. Not using weak passwords
2. Authorize Users by Roles
3. Add a Replication Keyfile
4. Make Backups Regularly
5. Run MongoDB on a Non-Standard Port

6. Enable Firewall Rules or Security groups
7. Trying to hack his own database

References

[1] [Online]. Available: https://stackoverflow.com/questions/11292215/where-does-mongodb-stand-in-the-cap-theorem. [Accessed 1 March 2021].

[2] [Online]. Available: https://dev.to/fullstackcafe/25-nosql-interview-questions-answered-you-must-know-in-2020-1kh9. [Accessed 25 February 2021].

[3] [Online]. Available: https://www.mongodb.com/faq. [Accessed 26 February 2021].

[4] [Online]. Available: https://intellipaat.com/blog/what-is-mongodb/. [Accessed 28 February 2021].

[5] [Online]. Available: https://www.mongodb.com/blog/post/mongodb-atlas-best-practices-part-2. [Accessed 28 February 2021].

[6] [Online]. Available: https://docs.mongodb.com/kubernetes-operator/master/tutorial/deploy-replica-set/. [Accessed 2 March 2021].

[7] K. Chodorow, MongoDB: The Definitive Guide: Powerful and Scalable, O'Reilly Media, 2013.

[8] B. J. a. S. Abraham, "Exploring the merits of nosql: A study based," in *International Conference on Networks Advances*, 2017.

[9] [Online]. Available: https://paperzz.com/doc/4334187/mms-manual. [Accessed 03 March 2021].

[10] [Online]. Available: https://docs.mongodb.com/manual/replication/. [Accessed 03 March 2021].

[11] [Online]. Available: https://docs.atlas.mongodb.com/tutorial/test-failover/. [Accessed 04 March 2021].

[12] [Online]. Available: https://docs.mongodb.com/manual/core/replica-set-rollbacks/. [Accessed 05 March 2021].

[13] [Online]. Available: https://www.percona.com/blog/2020/08/10/securing-mongodb-top-five-security-concerns/. [Accessed 06 March 2021].