International Journal of Research Publication and Reviews, Vol 5, no 4, pp 3285-3287 April 2024

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Encryption Methods in NoSQL Databases: A Case Study with MongoDB

***Pradyuman Sharma*[1], *Dr.Vishalshrivastava*[2], *Dr. Akhilpandey*[3], *Dr. Vishal Shrivastava*[3]**

[1]B. Tech. Scholar, [2,3]Professor, [4]Professor

Computer Science & Engineering Arya College of Engineering & I.T. India, Jaipur

[1]pradyumansharma000@gmail.com, [2]vishalshrivastava.cs@aryacollege.in, [3]akhil@aryacollege.in, [4]vishalshrivastava.cs@aryacollege.in

ABSTRACT:

To ensure an excellent level of safety for the confidential data, suitable encryption techniques are required, and all parties involved must abide by them. This study presents an analysis of the various encryption methods and how well they work with safe configuration, authentication, access control, and data encryption to manage private data. It involves adding privacy keys for safety and surveillance purposes, as well as improving MongoDB's level-based access secured paradigm. This function is no longer provided by the data stores using NoSQL, also referred to as extremely compressed data on asynchronous database management systems, which facilitate the handling of data of internet user programmes. Among the terms that spring to mind are MongoDB, NoSQL, encryption methods, and data security.The correct

Introduction:

What MongoDB provides are document-based databases. Unlike conventional relational databases, MongoDB is a database without relationships that enables documents to house data of any form. Nevertheless, consumers' security and privacy are not well protected by the current MongoDB solutions. In this, we introduced a privacy accessibility policy that entails encrypting user credentials after they are obtained. This keeps MongoDB operating at a high speed while guaranteeing a high degree of security for critical user data. As a result, among the most important demands for every user that publishes their private information on any platform is to ensure its protection.

Techniques of Applying MongoDB:

The steps associated with MongoDB The syntax that MongoDB stores its data in is called BSON.

Step 1: Each of the numerous databases on the server has a sizable number of collections in it. They work similarly to tables in a relational database. We only need one collection to be used for modelling our data. If we used the shell to query the Post collection after entering some data, we would get BSON returned as the graphical representation of our data. The data flow The MongoDB server must first be started from the command line. After that, navigate to the bin directory-like where the mongo server launches via a port .The MongoDB.exe application will then launch the MongoDB server.

Step 2: What comes next, The principal security objectives, namely secrecy, integrity, authenticity, and non-repudiation, can be achieved through the utilisation of diverse real-time encryption techniques. Every technique has a unique set of uses, while some of them might even be suitable to get a higher level of safety in the particular situation at hand. Creating a general solution for the privacy-aware access control rule is a crucial objective to pursue.

Implementing Encryption in MongoDB:

Encryption must be used with MongoDB in order to protect sensitive information kept in your database. To safeguard your data both in transit and at rest, MongoDB provides a number of encryption capabilities. I'll walk you through the process of implementing encryption in MongoDB in this guide:

1. Select Encryption Techniques:

MongoDB provides two main encryption techniques:

Your data is protected when it is stored on disk thanks to encryption at rest.

Encrypted in Transit: This makes sure that any data that is sent between the database and the application is encrypted.

Depending on the level of protection you need, you can use either one of these approaches.

2. Enable Encrypt at Rest: In order to activate encryption at rest, you must set up MongoDB to use the encrypted WiredTiger storage engine.

Take these actions:

a. Set the --storageEngine option to wiredTiger when you launch the MongoDB server.

b. You should include the key encryption file and encrypt settings for your MongoDB config file, which is usually called mongod.conf. For instance:

yamlCopy code

```
storage:
wiredTiger:
engineConfig:
encryption:
keyFile: /path/to/keyfile
```

c. Utilizing openssl random -base64 command, generate an encryption key file. This key file must be protected in order to decode data.

3. Turn on Encryption While in Transit:

You may use SSL/TLS to encrypt data that is sent among your application and MongoDB. An SSL/TLS certificate is required for this. To activate encryption in transport, follow these steps

yamlCopy code

```
net:

ssl:
mode: requireSSL

PEMKeyFile: /path/to/your.pem

CAFile: /path/to/ca.pem
```

The paths to the certificates and CA files should be used in lieu of /path/to/your.pem and /path/to/ca.pem.

4. User authorisation and Authentication: User authorisation and authentication are integrated into MongoDB. Make sure you grant people the proper access, limiting their ability to access your data according to their responsibilities and permissions. If centralized authentication is required, use systems such as Kerberos or LDAP in addition to using strong passwords.

5. Audit Logging: To keep an eye on user activity and spot any questionable activity, turn on MongoDB's audit logging. Set up an inspection log to record pertinent security incidents.

6. Frequent Updates and Monitoring: Apply security updates to the MongoDB server and all of its components. Keep a regular eye out for irregularities and security problems in the database.

7. Backups and Recovery: To safeguard your data in the event of a security breach or data loss, put in place a strong data backup and recovery plan.

8. Protect Your ecosystem: Don't only focus on MongoDB security; make sure the whole ecosystem is secure. This covers protecting the network infrastructure, the OS, and the server.

9. frequent Security Audits: To find gaps and holes in your MongoDB system, do security checks and penetration tests on a frequent basis.

10. Safety and Regulations: Verify that, if appropriate, your MongoDB configuration conforms with all applicable data protection laws, including GDPR, HIPAA, and others.

Keep in mind that protecting your MongoDB environment involves more than just installing encryption. It ought to be combined with a thorough security plan that incorporates user authentication, access control, and best practices for security at Applications.

Obstacles and Things to Think About:

Although MongoDB's encryption techniques greatly improve the safety of data, there are a few issues and things to be aware of:
Key management: For field-level encryption in particular, proper key management is essential. Encryption key availability and security must be guaranteed by organizations.

Performance cost: When encryption is used at the field level, it might result in a performance cost. Optimizing and planning carefully are required to reduce the impact on performance.

Compatibility: When implementing encryption, compatibility issues with various client apps and setups may arise, necessitating extensive testing and configuration modifications.

Conclusion:

In the digital era, security of information is a top priority. NoSQL databases, such as MongoDB, are rising to the occasion by offering encryption techniques for data that is in transit, at rest, and on the field. Sensitive data may be protected with many layers of protection using these techniques. The advantages of improved data security outweigh the drawbacks, even with issues like handling keys and performance overhead.

Organizations should be careful to implement and optimize encryption techniques for their NoSQL database systems as the digital ecosystem changes. MongoDB is a strong option for protecting data in NoSQL databases because of its extensive encryption features, which are essential for guaranteeing data confidentiality and integrity.

HYPERLINKS

- Hippocratic databases, R. Agrawal & Y. Xu. In Very Large Data Bases (VLDB): The 28th International Conference, 2002.
- M. A. Davidson and K. Browder. Technical paper, 2002; The Virtual Privat Database in Oracle9iR2. Technical White Paper about Oracle.
- Purpose-based access control in systems with relational databases for privacy protection, J. Byun and N. Li. 2008; The VLDB Journal, 17(4).
- Cattell, R. NoSQL and Scalable SQL Data Stores. 39(4):12–27, SIGMOD Rec., May 2011.
- Cavoukian Privacy by Design: principles, practices, and outcomes. P. de Hert, Y. Poullet, R. Leenes, S. Gutwirth, and editors, European Data Safety: Coming of Age. 2013
- E. Ferrari and P. Colombo. Implementing access control in systems that manage relational databases based on purpose. Knowledge and data technology (TKDE) Transactions, IEEE, 26(11), 2014.
- Burrows, M., T. Chandra, A. Fikes, R. E. Gruber and F. Chang. Bigtable is a distributed structured data storage system. 26(2):4 of ACM Transactions on Computing Systems (TOCS), 2008.