MongoDB Security Architecture

November 2015

Table of Contents

Introduction

The frequency and severity of data breaches continues to escalate year on year. Research from PWC identified over 117,000 attacks against information systems every day in 2014, representing an increase of 48% over the previous year. Companies reporting losses from security breaches totaling $20m or more doubled over the same period. Updated European Union General Data Protection regulations will increase penalties on those organizations proven to have not taken sufficient measures to protect their customers' data against unauthorized disclosure. With databases storing an organization's most important information assets, securing them is top of mind for administrators.

With the advanced security features available in MongoDB Enterprise Advanced, organizations have extensive capabilities to defend, detect, and control access to valuable online big data. These features, along with general security requirements, are discussed in this whitepaper and are then summarized as a security configuration checklist in the Appendix.

Requirements for Securing Online Big Data

In light of increasing threats over the past decade coupled with heightened concern for individual privacy, industries and governments around the world have embarked on a series of initiatives designed to increase security, reduce fraud and protect personally identifiable information (PII), including:

- PCI DSS for managing cardholder information
- HIPAA standards for managing healthcare information
- NIST 80-53 catalogs security controls for all U.S. federal information systems except those related to national security
- STIG for secure installation and maintenance of computer systems, defined for the US Department of Defense
- European Union Data Protection Directive

- Asia Pacific Economic Cooperation (APEC) data protection standardization

In addition to these initiatives, new regulations are being developed every year to cope with emerging threats and new demands for tighter controls governing data use.

Each set of regulations defines security and auditing requirements which are unique to a specific industry or application, and compliance is assessed on a per-project basis. Despite these differences, there are common foundational requirements across all of the directives, including:

- Restricting access to data, enforced via predefined privileges and security levels
- Measures to protect against the accidental or malicious disclosure, loss, destruction or damage of sensitive data
- The separation of duties when running applications and accessing data
- Recording the activities of users, administrative staff, and applications in accessing and processing data

These requirements inform the security architecture of MongoDB, with best practices for the implementation of a secure, compliant data management environment.

A holistic security architecture must cover the following:

- User access management to restrict access to sensitive data, implemented through authentication and authorization controls
- Logging operations against the database in an audit trail for forensic analysis
- Data protection via encryption of data in-motion over the network and at-rest in persistent storage
- Environmental and process controls

The requirements for each of these elements are discussed below.

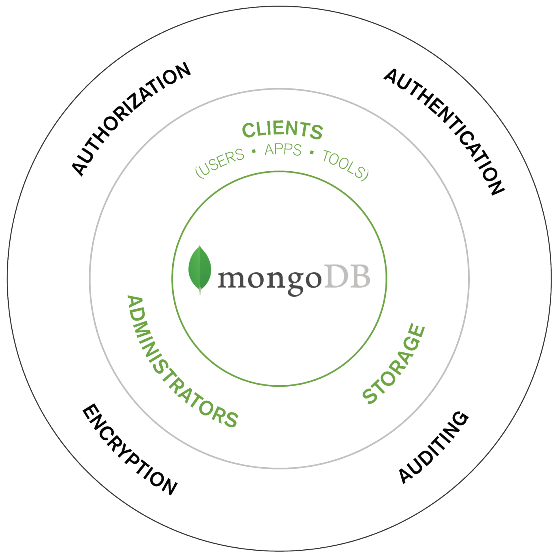


Figure 1: MongoDB End to End Security Architecture

User Access Management - Authentication

Authentication is designed to confirm the identity of entities accessing the database. In this context, entities are defined as:

- Users who need access to the database as part of their day-to-day business function
- Administrators (i.e. sysadmins, DBAs, QA staff) and developers
- Software systems including application servers, reporting tools, and management and backup suites
- Physical and logical nodes that the database runs on. Databases can be distributed across multiple nodes both for scaling operations and to ensure continuous operation in the event of systems failure or maintenance.

Best practices for authentication are as follows.

- **Create Security Credentials.** Create login credentials for each entity that will need access to the database, and avoid creating a single "admin" login that every user shares.

  By creating credentials it becomes easier to define, manage, and track system access for each user. Should a user's credentials become compromised, this approach makes it easier to revoke the user without disrupting other users who need access to the database.

Developers, Administrators, and DBAs should all have unique credentials to access the database. When logins are shared it can be impossible to identify who attempted different operations, and it eliminates the ability to assign fine-grained permissions. With unique logins, staff that move off of the project or leave the organization can have their access revoked without affecting other user accounts.

It is a best practice to create separate login credentials for each application that will be accessing the database. As new applications are introduced and old applications are retired, this approach helps manage access. Some MongoDB customers create multiple logins for different services within a single application, which are then recorded in audit trails and query logs.

Authentication should be enforced between nodes. This prevents unauthorized instances from joining a database cluster, preventing the illicit copying or movement of data to insecure nodes.

- **Supporting In-Database and Centralized User Access Management.** Databases should provide the ability to manage user authentication either within the database itself, typically via a Challenge/Response mechanism, or through integration with organization-wide identity management systems. Integrating MongoDB within the existing information security infrastructure enforces centralized and standardized control over user access. If, for example, a user's access must be revoked, the update can be made in a single repository and enforced instantly across all systems, including MongoDB.
- **Enforce Password Policies.** Passwords should adhere to minimum complexity requirements established by the organization, and they should be reset periodically. Low entropy passwords can be easy to break, even if they are encrypted. High entropy passwords can be compromised given sufficient time.

User Rights Management - Authorization

Once an entity has been authenticated, authorization governs what that entity is entitled to do in the database. Privileges are assigned to user roles that define a specific set of actions that can be performed against the database. Best practices include:

- **Grant Minimal Access to Entities.** Entities should be provided with the minimal database access they need to perform their function. If an application requires access to a logical database, it should be restricted to operations on that database alone, and prevented from accessing other logical databases. This helps protect against both malicious and accidental access or unauthorized modification of data.
- **Group Common Access Privileges into Roles.** Entities can often be grouped into "roles" such as "DBA", "Sysadmin", and "App server." Permissions for a role can be centrally managed and users can be added or removed from roles as needed. Using roles helps simplify management of access control by defining a single set of rules that apply to specific classes of entities, rather than having to define them individually for each user.
- **Control Which Actions an Entity Can Perform.** When granting access to a database, consideration should be made for which specific actions or commands each entity should have permission to run. For example, an application may need read/write permissions to the database, whereas a reporting tool may be restricted to read-only permission. Some users may be granted privileges that enable them to insert new data to the database, but not to update or delete existing data. Care should be taken to ensure that only the minimal set of privileges is provided. Credentials of the most privileged accounts could compromise the entire database if they are hacked internally or by an external intruder.
- **Control Access to Sensitive Data.** To prevent the emergence of data silos, it should be possible to restrict permissions to individual fields, based on security privileges. For example, some fields of a record may be accessible to all users of the database, while others containing sensitive information, such as PII, should be restricted to users with specific security clearance.

Auditing

By creating audit trails, changes to database configuration and data can be captured for each entity accessing the database, providing a log for compliance and forensic analysis. Auditing can also detect attempts to access unauthorized data.

- **Track Changes to Database Configuration.** Any time a database configuration is changed, the action should be recorded in an audit log which should include the change action, the identity of the user and a timestamp.
- **Track Changes to Data.** It should be possible to configure the audit trail to capture every query or write operation to the database. Care, however, should be exercised when configuring this rule for applications. For example, if the application is inserting tens of thousands of records per second, writing each operation to the audit log can impose a performance overhead to the database. The project team should determine any tradeoffs between performance and auditing requirements. It should be possible to filter events that are captured, for example only specific users, IP addresses or operations.

Encryption

Encryption is the encoding of critical data whenever it is in transit or at rest, enabling only authorized entities to read it. Data will be protected in the event that eavesdroppers or hackers gain access to the server, network or database.

- **Encrypt Connections to the Database.** All user or application access to the database should be via encrypted channels including connections established through the drivers, command line or shell, as well as remote access sessions to the database servers themselves. Internal communications between database nodes should also be encrypted, i.e. traffic replicated between nodes of a database cluster.
- **Enforce Strong Encryption.** The database should support FIPS (Federal Information Processing Standard) 140-2 to ensure the implementation of secure encryption algorithms.
- **Encrypt Data at Rest.** One of most common threats to security comes from attacks that bypass the database itself and target the underlying Operating System and physical storage of production servers or backup devices, in order to access raw data. On-disk encryption of the database's data files mitigates this threat.
- **Sign and Rotate Encryption Keys.** Encryption keys for network and disk encryption should be periodically rotated. SSL/TLS encryption channels should use signed certificates to ensure that clients can certify the credentials they receive from server components.

Environmental and Process Control

The environment in which the database and underlying infrastructure is running should be protected with both physical and logical controls. These are enforced in the underlying deployment environment, rather than in the database itself, and include:

- Installation of firewalls
- Network configurations
- Defining file system permissions
- Creation of physical access controls to the IT environment

As manual configuration errors are one of the largest causes of attackers bypassing security mechanisms, there are a series of operational processes that should be adopted to further promote and enforce secure operation, including:

- DBA and developer training
- Database monitoring and backup
- Database maintenance, i.e. applying the latest patches

MongoDB Security Features

With comprehensive controls for user rights management, auditing and encryption, coupled with best practices in environmental protection, MongoDB can meet the best practice requirements described earlier, enabling users to unlock the power of online big data.

The following section discusses MongoDB's security architecture before presenting a checklist of those

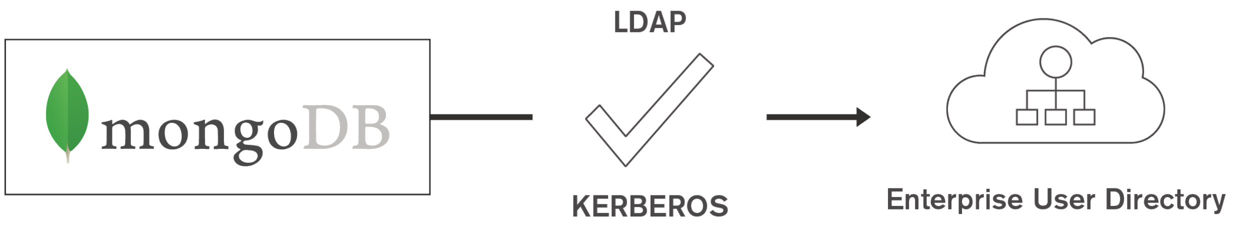


Figure 2: Integrating MongoDB with Centralized User Access Controls

capabilities needed to create a secure database environment.

MongoDB Enterprise Advanced

MongoDB Enterprise Advanced is the certified and supported production release of MongoDB, with advanced security features, including Kerberos and LDAP authentication, encryption of data at-rest, FIPS-compliance, and maintenance of audit logs. These capabilities extend MongoDB's already comprehensive security framework, which includes Role-Based Access Control, PKI certificates, Field-Level Redaction, and SSL/TLS data transport encryption. You can get access to all of the features of MongoDB Enterprise free of charge for evaluation and development environments.

You can learn more about MongoDB Enterprise Advanced and associated products and services later in the whitepaper.

MongoDB Authentication

The authentication of entities accessing MongoDB can be managed from within the database itself, or via integration with an external mechanism (i.e. LDAP, x.509 PKI certificates, or a Kerberos service). MongoDB Enterprise Advanced is required when using LDAP or Kerberos.

In Database Authentication

MongoDB authenticates entities on a per-database level using the SCRAM-SHA-1 IETF standard. Users are authenticated via the authentication command, while database nodes can be authenticated to the MongoDB cluster via keyfiles.

Review the Access Control documentation to learn more.

LDAP Authentication

LDAP is widely used by many organizations to standardize and simplify the way large numbers of users are managed across internal systems and applications. In many cases, LDAP is also used as the centralized authority for user access control to ensure that internal security policies are compliant with corporate and regulatory guidelines.

With LDAP integration, MongoDB can authenticate users directly against corporate LDAP infrastructure to enforce centralised access policies. Note that MongoDB currently supports LDAP authentication, and not authorization. See the following section of the whitepaper to learn more about the authorization controls available in MongoDB.

Administrators can configure MongoDB to authenticate users via Linux PAM or by proxying authentication requests to a specified LDAP service. LDAP integration is available in MongoDB Enterprise Advanced.

Kerberos Authentication

With MongoDB Enterprise Advanced, authentication using a Kerberos service is supported. Kerberos is an industry standard authentication protocol for large client/server systems, allowing both the client and server to verify each others' identity. With Kerberos support, MongoDB can take advantage of existing authentication infrastructure and processes, including Microsoft Windows Active Directory .

As with LDAP and x.509 certificates, before users can authenticate to MongoDB using Kerberos, they must first be created and granted privileges within MongoDB. The process for doing this, along with a full configuration checklist is described in the MongoDB and Kerberos tutorial.

x.509 Certificate Authentication

With support for x.509 certificates MongoDB can be integrated with existing information security infrastructure and certificate authorities, supporting both user and inter-node authentication.

Users can be authenticated to MongoDB using client certificates rather than self-maintained passwords.

Inter-cluster authentication and communication between MongoDB nodes can be secured with x.509 member certificates rather than keyfiles, ensuring stricter membership controls with less administrative overhead, i.e. by eliminating the shared password used by keyfiles. x.509 certificates can be used by nodes to verify their membership of MongoDB replica sets and sharded clusters. A single Certificate Authority (CA) should issue all the x.509 certificates for the members of a sharded cluster or a replica set.

Instructions for configuration are described in the MongoDB and x.509 certificates tutorial.

MongoDB and Red Hat Identity Management

Red Hat Enterprise Linux (RHEL) is a popular environment for MongoDB deployments. Providing ease of use to administrators and security professionals working in these environments, the MongoDB security features are integrated with the Identity Management (IdM) features of RHEL. This integration provides central management of individual entities and their authentication, authorization and privileges.

Review the Red Hat Linux Identity Management tutorial for instruction on configuration with MongoDB.

Red Hat IdM integration is available with MongoDB Enterprise Advanced and requires the database to be configured for Kerberos authentication.

MongoDB Authorization

MongoDB and Microsoft Active Directory

MongoDB Enterprise Advanced provides support for authentication using Microsoft Active Directory with both Kerberos and LDAP. The Active Directory domain controller authenticates the MongoDB users and servers running in a Windows network.

MongoDB allows administrators to define the specific permissions an application or user has, and what data they can see when querying the database.

User Defined Roles

MongoDB provides over ten built-in roles supporting different user and administrator privileges. These can be customised through User Defined Roles, enabling administrators to assign fine-grained privileges to users or applications. Authorization privileges can be based on the specific functionality a user needs in their role, or to reflect their organizational structure. MongoDB provides the ability to specify user privileges with both database and collection-level granularity.

Privileges are assigned to roles, and roles are in turn assigned to users. For example:

- Classes of users and applications can be assigned privileges to insert data, but not to update or delete data from the database
- DBAs may be assigned privileges that enable them to create collections and indexes on the database, while developers are restricted to CRUD operations
- Certain administrator roles may have cluster-wide privileges to build replica sets and configure sharding, while others are restricted to creating new users or inspecting logs
- Processes for monitoring MongoDB clusters can be restricted to run just those commands that retrieve server status, without having full administrative access to perform database operations
- Within a multi-tenant environment, “landlord” developers and administrators can be assigned permissions across physical databases, while “tenant” developers and administrators can be granted a more limited set of actions across logical databases or individual collections. This functionality enables a clear separation of duties and control, both between and within organizations.

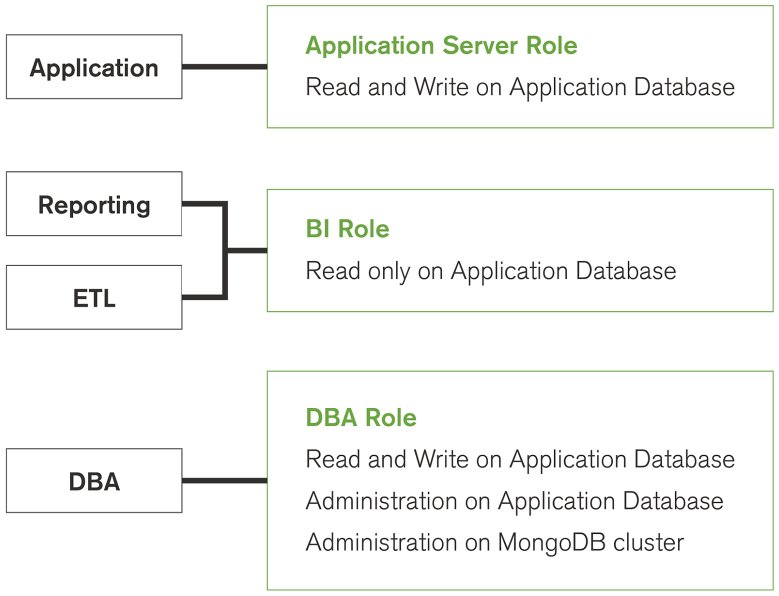

Application
Application Server Role
Read and Write on Application Database
Reporting
ETL
BI Role
Read only on Application Database
DBA
DBA Role
Read and Write on Application Database
Administration on Application Database
Administration on MongoDB cluster

Figure 3: MongoDB User Defined Roles Permit Separations of Duty

To ensure ease of account provisioning and maintenance, roles can be delegated across teams, ensuring the enforcement of consistent policies across specific functions within the organization.

Review the Authorization section of the documentation to learn more about roles in MongoDB.

When combined with the auditing capabilities available with MongoDB Enterprise Advanced, customers can define specific administrative actions per role, and then log all of those actions. As a result, the organization is able to enforce end-to-end operational control and maintain insight of actions for compliance and reporting.

MongoDB Field Level Redaction

MongoDB's field level redaction allows building access control to individual fields of document, working in conjunction with client-side code. Implemented via the redaction stage of MongoDB's Aggregation Pipeline, developers are provided with a method to restrict the content of a returned document on a per-field level. Permissions can be based on both the content of the document and on specific user privileges, based on security labels. Access control policies can be described using the MongoDB query language, making it simple for developers to implement the required controls.

Since data is redacted before it is returned to the application, exposure of sensitive information is reduced. Field level redaction is applicable to a wide range of sensitive data including personally identifiable information such as names, social security numbers, birthdates and bank account numbers. By co-locating data with different sensitivity levels within a single document, schema and query designs are simplified.

The redaction logic must be passed by the application to the database on each request. It therefore relies on trusted middleware running in the application to ensure the redaction pipeline stage is appended to any query that requires the redaction logic.

MongoDB Auditing

The MongoDB Enterprise Advanced auditing framework logs all access and actions executed against the database. The auditing framework captures administrative actions (DDL) such as schema operations as well as authentication and authorization activities, along with read and write (DML) operations to the database. Administrators can construct and filter audit trails for any operation against MongoDB, whether DML, DCL or DDL without having to rely on third party tools. For example, it is possible to log and audit the identities of users who

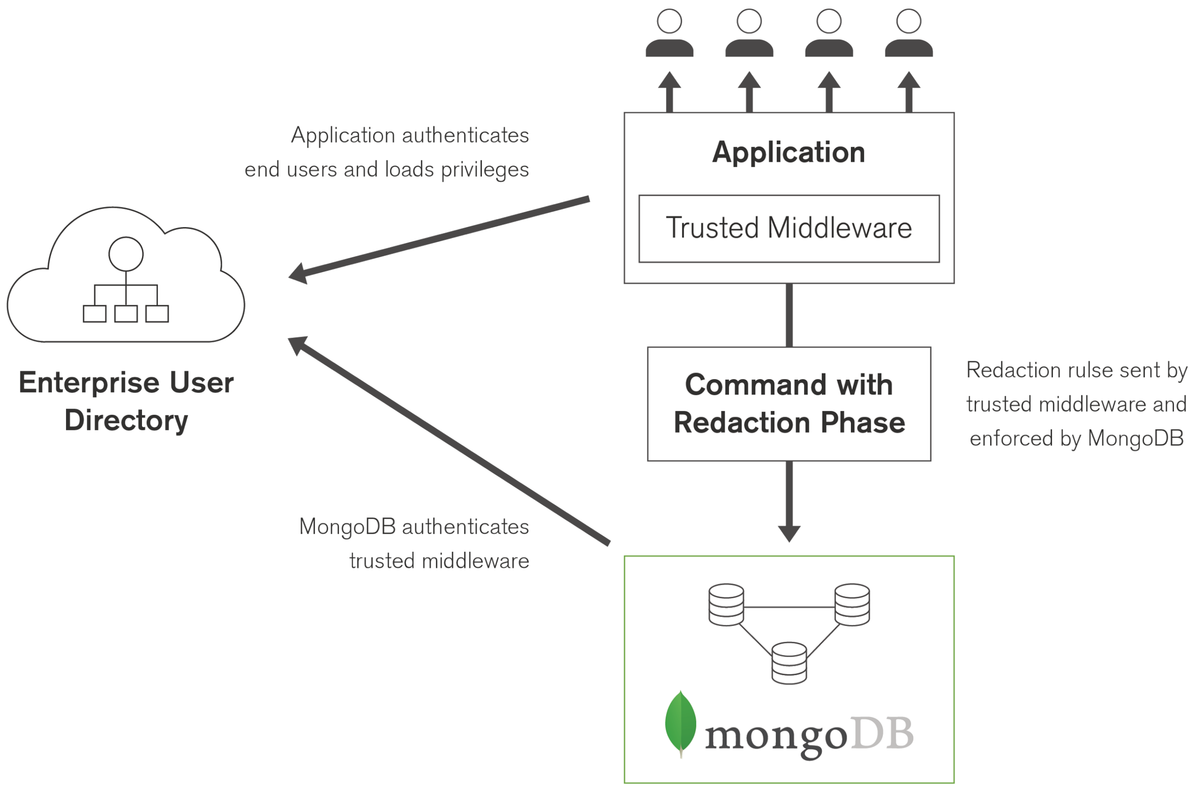

Application authenticates end users and loads privileges
Application
Trusted Middleware
Enterprise User Directory
Command with Redaction Phase
Redaction rulse sent by trusted middleware and enforced by MongoDB
MongoDB authenticates trusted middleware
mongoDB

Figure 4: MongoDB Field Level Redaction Restricts Access to Sensitive Data

retrieved specific documents, and any changes made to the database during their session.

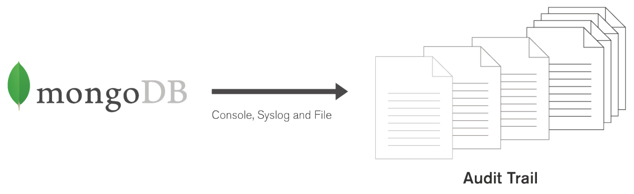


Figure 5: MongoDB Maintains an Audit Trail of Administrative Actions Against the Database

Administrators can configure MongoDB to log all actions or apply filters to capture only specific events, users or roles. The audit log can be written to multiple destinations in a variety of formats including to the console and syslog (in JSON format), and to a file (JSON or BSON), which can then be loaded to MongoDB and analyzed to identify relevant events. Each MongoDB server writes events to its attached storage. The DBA can then use their own tools to merge these events into a single audit log, enabling a cluster-wide view of operations that affected multiple nodes.

MongoDB Enterprise Advanced also supports role-based auditing. It is possible to log and report activities by specific role, such as userAdmin or dbAdmin – coupled with any inherited roles each user has – rather than having to extract activity for each individual administrator.

Auditing adds performance overhead to a MongoDB system. The amount is dependent on several factors including which events are logged and where the audit log is maintained, such as on an external storage device and the audit log format. Users should consider the specific needs of their application for auditing and their performance goals in order to determine their optimal configuration.

Learn more from the MongoDB auditing documentation.

MongoDB Encryption

Administrators can encrypt MongoDB data in motion over the network and at rest in permanent storage.

Network Encryption

Support for SSL/TLS allows clients to connect to MongoDB over an encrypted channel. Clients are defined as any entity capable of connecting to the MongoDB server, including:

- Users and administrators
- Applications
- MongoDB tools (e.g., mongodump, mongorestore, mongotop)
- Nodes that make up a MongoDB cluster, such as replica set members, query routers and config servers.

It is possible to mix SSL/TLS with non-SSL/TLS connections on the same port, which can be useful when applying finer grained encryption controls for internal and external traffic, as well as avoiding downtime when upgrading a MongoDB cluster to support SSL.

The TLS protocol is also supported with x.509 certificates.

MongoDB Enterprise Advanced supports FIPS 140-2 encryption if run in FIPS Mode with a FIPS validated Cryptographic module. The mongod and mongos processes should be configured with the "sslFIPSMode" setting. In addition, these processes should be deployed on systems with an OpenSSL library configured with the FIPS 140-2 module.

The MongoDB documentation includes a tutorial for configuring TLS/SSL connections.

Disk Encryption

There are multiple ways to encrypt data at rest with MongoDB. Encryption can implemented at the application level, or via external filesystem and disk encryption solutions. By introducing additional technology into the stack, both of these approaches can add cost and complexity.

With the introduction of the Encrypted storage engine in MongoDB 3.2, protection of data at-rest becomes an integral feature of the database. By natively encrypting database files on disk, administrators eliminate both the management and performance overhead of external encryption mechanisms. This new storage engine provides an additional level of defense, allowing only those staff with the appropriate database credentials access to encrypted data.

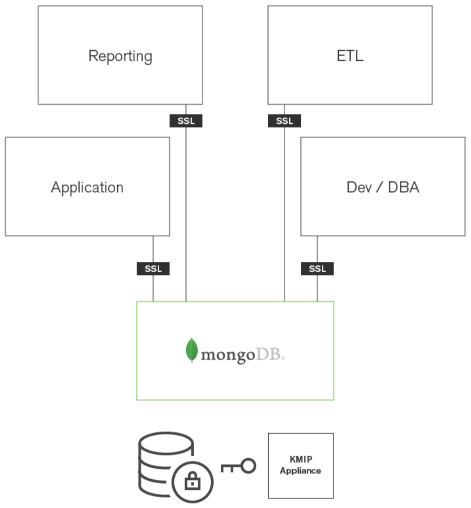


Figure 6: End to End Encryption – Data In-Flight and Data At-Rest

Using the Encrypted storage engine, the raw database content, referred to as plaintext, is encrypted using an algorithm that takes a random encryption key as input and generates ciphertext that can only be read if decrypted with the decryption key. The process is entirely transparent to the application. MongoDB supports a variety of encryption schema, with AES-256 (256 bit encryption) in CBC mode being the default. AES-256 in GCM mode is also supported. The encryption schema can be configured for FIPS 140-2 compliance.

The storage engine encrypts each database with a separate key. The key-wrapping scheme in MongoDB wraps all of the individual internal database keys with one external master key for each server. The Encrypted storage engine supports two key management options – in both cases, the only key being managed outside of MongoDB is the master key:

- Local key management via a keyfile.

- Integration with a third party key management appliance via the KMIP protocol (recommended).

Most regulatory requirements mandate that the encryption keys must be rotated and replaced with a new key at least once annually. MongoDB can achieve key rotation without incurring downtime by performing rolling restarts of the replica set. When using a KMIP appliance, the database files themselves do not need to be re-encrypted, thereby avoiding the significant performance overhead imposed by key rotation in other databases. Only the master key is rotated, and the internal database keystore is re-encrypted.

The Encrypted storage engine is designed for operational efficiency and performance:

- Compatible with WiredTiger's document level concurrency control and compression.
- Support for Intel's AES-NI equipped CPUs for acceleration of the encryption/decryption process.
- As documents are modified, only updated storage blocks need to be encrypted, rather than the entire database.

Based on user testing, the Encrypted storage engine minimizes performance overhead to around 15% (this can vary, based on data types being encrypted), which can be much less than the observed overhead imposed by some filesystem encryption solutions.

The Encrypted storage engine is based on WiredTiger and available as part of MongoDB Enterprise Advanced. Refer to the documentation to learn more, and see a tutorial on how to configure the storage engine.

Environment & Processes

Building on the database security controls discussed above, to further reduce the risk of exploitation, run MongoDB in a trusted environment, implement a secure development lifecycle, and enforce deployment best operational practices.

A "Defense in Depth" approach is recommended to complement secure MongoDB deployments, addressing a number of different methods for managing risk and reducing exposure.

The intention of a Defense in Depth approach is to layer your environment to ensure there are no exploitable single points of failure that could allow an intruder or untrusted party to access the data stored in the MongoDB database.

Secure environments use the following strategies to control access, with more detail available in the Network Exposure and Security section of the documentation.

- **Network Filter.** By using filters such as firewalls and router ACL rules, connections to MongoDB from unknown systems can be blocked.

  Firewalls should limit both incoming and outgoing traffic to/from a specific port to trusted and untrusted systems. For best results and to minimize overall exposure, ensure that only traffic from trusted sources can reach mongod and mongos instances and that the mongod and mongos instances can only connect to trusted outputs. In addition, unneeded system services should be deactivated.

- **Binding IP Addresses.** The bind_ip setting for mongod and mongos instances limits the network interfaces on which MongoDB programs will listen for incoming connections.
- **Running in VPNs.** Limit MongoDB programs to non-public local networks and virtual private networks. Virtual Private Networks (VPNs) make it possible to link two networks over an encrypted and limited-access trusted network. Typically MongoDB users configure SSL rather than IPSEC protocols for performance advantages.
- **Dedicated OS User Account.** A user account dedicated to MongoDB should be created and used to run MongoDB executables. MongoDB should not run as the "root" user.
- **File System Permissions.** The servers running MongoDB should employ filesystem permissions that prevent users from accessing the data files created by MongoDB. MongoDB configuration files and the cluster keyfile should be protected to disallow access by unauthorized users.
- **Query Injection.** As a client program assembles a query in MongoDB, it builds a BSON object, not a string. Thus traditional SQL injection attacks should not pose a

risk to the system for queries submitted as BSON objects.

However, several MongoDB operations permit the evaluation of arbitrary JavaScript expressions and care should be taken to avoid malicious expressions. Fortunately, most queries can be expressed in BSON and for cases where Javascript is required, it is possible to mix JavaScript and BSON so that user-specified values are evaluated as values and not as code.

MongoDB also allows the administrator to configure the MongoDB server to prevent the execution of Javascript scripts. This will prevent MapReduce jobs from running, but the aggregation framework can be used as an alternative in many use cases.

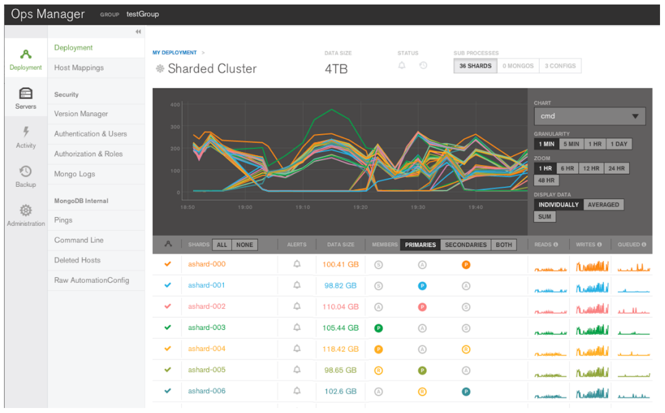
- **Physical Access Controls.** In addition to the logical controls discussed above, controlling physical access to servers, storage and backup media provides critical environmental protection.

Database Monitoring

Proactive monitoring of all components within an IT environment is always a best practice. System performance and availability depend on the timely detection and resolution of potential issues before they present problems to users.

From a database security perspective, monitoring is critical to identifying potential exploits in real time, thereby reducing the impact of any breach. For example, sudden peaks in the CPU and memory loads of host systems and high operations counters in the database can indicate a Denial of Service attack. MongoDB ships with a variety of tools including mongostat and mongotop that can be used to monitor your database.

The most comprehensive monitoring solution is provided by MongoDB Ops Manager, which is the simplest way to run MongoDB. Ops Manager makes it easy for operations teams to monitor, secure, back up, and scale MongoDB. Ops Manager is available with MongoDB Enterprise Advanced. MongoDB Cloud Manager is a hosted management platform for MongoDB providing many of the same capabilities as Ops Manager. MongoDB Enterprise Advanced customers can choose between Ops Manager and Cloud Manager for their deployments.



SHARDS	DATA SIZE
ashard-000	100.41 GB
ashard-001	98.82 GB
ashard-002	110.04 GB
ashard-003	105.44 GB
ashard-004	118.42 GB
ashard-005	98.65 GB
ashard-006	102.6 GB

Figure 7: Ops Manager Offers Charts, Custom Dashboards & Automated Alerting

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 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 MongoDB deployments of the organization.

Ops Manager allows 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.

Disaster Recovery: Backups & Point-in-Time Recovery

Ops Manager backups are maintained continuously, just a few seconds behind the operational system. If MongoDB 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 backup solutions that offers point-in-time recovery of replica sets and cluster-wide snapshots of sharded clusters. You can restore to precisely the moment you need, quickly and safely.

Training

Investing in skills can ensure a more secure environment. MongoDB University offers courses for both developers and DBAs:

- Free, web-based classes, delivered over 7 weeks, supported by lectures, homework and forums to interact with instructors and other students. Over 300,000 students have enrolled in these classes since they were first released.
- 2-day public training events held at MongoDB facilities
- Private training customized to an organization's specific requirements - including best practices in secure development and deployment - delivered at your site.

Database Maintenance

Always ensure you are running the latest production-certified release of both MongoDB and the drivers, and have applied the latest set of patches. While MongoDB Enterprise Advanced customers get access to emergency patches, fixes for security vulnerabilities are available to all MongoDB users as soon as they are released.

Conclusion

Data security and privacy is a critical concern. Data collected from social media, mobile devices and sensor networks has become as sensitive as traditional transaction data generated from back-office systems. For this reason, big data technologies must evolve to meet the regulatory compliance standards demanded by industry and government.

As demonstrated in this white paper, with MongoDB Enterprise Advanced organizations benefit from extensive capabilities to defend, detect and control access to valuable online big data. You can get started by reviewing the MongoDB Security Documentation, and downloading MongoDB Enterprise Advanced for evaluation today.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than a third 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 Cloud Manager is the easiest way to run MongoDB in the cloud. It makes MongoDB the system you worry about the least and like managing the most.

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.

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.org)
MongoDB Enterprise Download (mongodb.com/download)

MongoDB Security Checklist

The checklist defines the steps, along with key resources, to creating a secure MongoDB deployment.

Prepare Secure Operating Environment	
Download MongoDB Enterprise Advanced	Production-Certified release
Configure network (firewall, bind IP addresses, VPN, etc.)	Review platform-specific documentation MongoDB network documentation
Create MongoDB user account & permissions	Review platform-specific and filesystem documentation for creating OS logins and permissions
Configure the Encrypted storage engine	Storage Engine Documentation
Prepare MongoDB Deployment	
Configure preferred authentication	LDAP documentation Kerberos documentation Red Hat IdM documentation Challenge / Response documentation
Configure inter-cluster authentication	x.509 Certificate documentation
Setup SSL/TLS certificates	SSL/TLS documentation
Enable FIPS Mode	FIPS mode documentation
Configure auditing	Auditing of DDL & DML operations
Define Users and Roles	
Document roles that will access the system	Project team process
Create MongoDB admin account	Add User to MongoDB
Configure permissions for each role	Built-in (standard) MongoDB roles User defined roles
Optional Advanced Configuration: Implement field level redaction	Redaction documentation
Monitor the Deployment	
Configure MongoDB Management	Ops Manager documentation
Monitor and apply latest patches	Subscribe to the MongoDB Announcements Google Group for availability of the latest releases and patches Monitor patch alerts and updates for infrastructure (server, network and storage components, OS, middleware, etc.)