
MongoDB Security Guide

Release 3.2.3

MongoDB, Inc.

February 17, 2016

© MongoDB, Inc. 2008 - 2016 This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License](#)

1	Security Checklist	3
1.1	Enable Access Control and Enforce Authentication	3
1.2	Configure Role-Based Access Control	3
1.3	Encrypt Communication	4
1.4	Limit Network Exposure	4
1.5	Audit System Activity	4
1.6	Encrypt and Protect Data	4
1.7	Run MongoDB with a Dedicated User	4
1.8	Run MongoDB with Secure Configuration Options	4
1.9	Request a Security Technical Implementation Guide (where applicable)	5
1.10	Consider Security Standards Compliance	5
2	Authentication	7
2.1	Authentication Methods	7
2.2	Authentication Mechanisms	7
2.3	Internal Authentication	8
2.4	Authentication on Sharded Clusters	8
3	Role-Based Access Control	23
3.1	Enable Access Control	23
3.2	Roles	23
3.3	Users and Roles	24
3.4	Built-In Roles and User-Defined Roles	24
4	Encryption	29
4.1	Transport Encryption	29
4.2	Encryption at Rest	29
5	Auditing	33
5.1	Enable and Configure Audit Output	33
5.2	Audit Events and Filter	33
5.3	Audit Guarantee	34
6	Security Hardening	35
6.1	MongoDB Configuration Hardening	35
6.2	Network Hardening	35
7	Security Tutorials	39
7.1	Enable Access Control	39

7.2	Authentication Mechanisms	47
7.3	Configure Users and Roles	70
7.4	Network	79
7.5	Encryption	97
7.6	Auditing	102
7.7	Miscellaneous	107
8	Security Reference	113
8.1	Security Methods in the <code>mongo</code> Shell	113
8.2	Security Reference Documentation	114
9	Additional Resources	143

On this page

- [Additional Resources](#) (page 143)

Maintaining a secure MongoDB deployment requires administrators to implement controls to ensure that users and applications have access to only the data that they require. MongoDB provides features that allow administrators to implement these controls and restrictions for any MongoDB deployment.

If you are already familiar with security and MongoDB security practices, consider the *Security Checklist* (page 3) for a collection of recommended actions to protect a MongoDB deployment.

Security Checklist

On this page

- [Enable Access Control and Enforce Authentication](#) (page 3)
- [Configure Role-Based Access Control](#) (page 3)
- [Encrypt Communication](#) (page 4)
- [Limit Network Exposure](#) (page 4)
- [Audit System Activity](#) (page 4)
- [Encrypt and Protect Data](#) (page 4)
- [Run MongoDB with a Dedicated User](#) (page 4)
- [Run MongoDB with Secure Configuration Options](#) (page 4)
- [Request a Security Technical Implementation Guide \(where applicable\)](#) (page 5)
- [Consider Security Standards Compliance](#) (page 5)

This document provides a list of security measures that you should implement to protect your MongoDB installation.

1.1 Enable Access Control and Enforce Authentication

Enable access control and specify the authentication mechanism. You can use the default MongoDB authentication mechanism or an existing external framework. Authentication requires that all clients and servers provide valid credentials before they can connect to the system. In clustered deployments, enable authentication for each MongoDB server.

See [Authentication](#) (page 7) and [Enable Client Access Control](#) (page 40).

1.2 Configure Role-Based Access Control

Create a user administrator **first**, then create additional users. Create a unique MongoDB user for each person and application that accesses the system.

Create roles that define the exact access a set of users needs. Follow a principle of least privilege. Then create users and assign them only the roles they need to perform their operations. A user can be a person or a client application.

See [Role-Based Access Control](#) (page 23) and [Manage User and Roles](#) (page 70), .

1.3 Encrypt Communication

Configure MongoDB to use TLS/SSL for all incoming and outgoing connections. Use TLS/SSL to encrypt communication between `mongod` and `mongos` components of a MongoDB client as well as between all applications and MongoDB.

See *Configure mongod and mongos for TLS/SSL* (page 79).

1.4 Limit Network Exposure

Ensure that MongoDB runs in a trusted network environment and limit the interfaces on which MongoDB instances listen for incoming connections. Allow only trusted clients to access the network interfaces and ports on which MongoDB instances are available.

See *Security Hardening* (page 35) and the `bindIp` setting.

1.5 Audit System Activity

Track access and changes to database configurations and data. [MongoDB Enterprise¹](#) includes a system auditing facility that can record system events (e.g. user operations, connection events) on a MongoDB instance. These audit records permit forensic analysis and allow administrators to verify proper controls.

See *Auditing* (page 33) and *Configure Auditing* (page 102).

1.6 Encrypt and Protect Data

Encrypt MongoDB data on each host using file-system, device, or physical encryption. Protect MongoDB data using file-system permissions. MongoDB data includes data files, configuration files, auditing logs, and key files.

1.7 Run MongoDB with a Dedicated User

Run MongoDB processes with a dedicated operating system user account. Ensure that the account has permissions to access data but no unnecessary permissions.

See <https://docs.mongodb.org/manual/installation> for more information on running MongoDB.

1.8 Run MongoDB with Secure Configuration Options

MongoDB supports the execution of JavaScript code for certain server-side operations: `mapReduce`, `group`, and `$where`. If you do not use these operations, disable server-side scripting by using the `--noscripting` option on the command line.

Use only the MongoDB wire protocol on production deployments. Do **not** enable the following, all of which enable the web server interface: `net.http.enabled`, `net.http.JSONPEnabled`, and `net.http.RESTInterfaceEnabled`. Leave these *disabled*, unless required for backwards compatibility.

¹<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

Deprecated since version 3.2: HTTP interface for MongoDB

Keep input validation enabled. MongoDB enables input validation by default through the `wireObjectCheck` setting. This ensures that all documents stored by the `mongod` instance are valid *BSON*.

See *Security Hardening* (page 35) for more information on hardening MongoDB configuration.

1.9 Request a Security Technical Implementation Guide (where applicable)

The Security Technical Implementation Guide (STIG) contains security guidelines for deployments within the United States Department of Defense. MongoDB Inc. provides its STIG, upon request, for situations where it is required. Please [request a copy](#)² for more information.

1.10 Consider Security Standards Compliance

For applications requiring HIPAA or PCI-DSS compliance, please refer to the [MongoDB Security Reference Architecture](#)³ to learn more about how you can use the key security capabilities to build compliant application infrastructure.

²<http://www.mongodb.com/lp/contact/stig-requests>

³http://info.mongodb.com/rs/mongodb/images/MongoDB_Security_Architecture_WP.pdf

Authentication

On this page

- [Authentication Methods](#) (page 7)
- [Authentication Mechanisms](#) (page 7)
- [Internal Authentication](#) (page 8)
- [Authentication on Sharded Clusters](#) (page 8)

Authentication is the process of verifying the identity of a client. When access control, i.e. [authorization](#) (page 23), is enabled, MongoDB requires all clients to authenticate themselves in order to determine their access.

Although authentication and [authorization](#) (page 23) are closely connected, authentication is distinct from authorization. Authentication verifies the identity of a user; authorization determines the verified user's access to resources and operations.

2.1 Authentication Methods

To authenticate a [user](#) (page 8), MongoDB provides the `db.auth()` method.

For the `mongo` shell and the MongoDB tools, you can also authenticate a user by passing in the user authentication information from the command line.

2.2 Authentication Mechanisms

MongoDB supports a number of [authentication mechanisms](#) (page 10) that clients can use to verify their identity. These mechanisms allow MongoDB to integrate into your existing authentication system.

MongoDB supports multiple authentication mechanisms:

- [SCRAM-SHA-1](#) (page 11)
- [MongoDB Challenge and Response \(MONGODB-CR\)](#) (page 12)

Changed in version 3.0: New challenge-response users created in 3.0 will use `SCRAM-SHA-1`. If using 2.6 user data, MongoDB 3.0 will continue to use the `MONGODB-CR`.

- [x.509 Certificate Authentication](#) (page 12).

In addition to supporting the aforementioned mechanisms, MongoDB Enterprise also supports the following mechanisms:

- [LDAP proxy authentication](#) (page 15), and
- [Kerberos authentication](#) (page 15).

2.3 Internal Authentication

In addition to verifying the identity of a client, MongoDB can require members of replica sets and sharded clusters to *authenticate their membership* (page 19) to their respective replica set or sharded cluster. See *Internal Authentication* (page 19) for more information.

2.4 Authentication on Sharded Clusters

In sharded clusters, clients generally authenticate directly to the `mongos` instances. However, some maintenance operations may require authenticating directly to a specific shard. For more information on authentication and sharded clusters, see *Sharded Cluster Users* (page 9).

2.4.1 Users

On this page

- [User Management Interface](#) (page 8)
- [Authentication Database](#) (page 8)
- [Authenticate a User](#) (page 9)
- [Centralized User Data](#) (page 9)
- [Sharded Cluster Users](#) (page 9)
- [Localhost Exception](#) (page 10)

To authenticate a client in MongoDB, you must add a corresponding user to MongoDB.

User Management Interface

To add a user, MongoDB provides the `db.createUser()` method. When adding a user, you can assign *roles* (page 23) to the user in order to grant privileges.

Note: The first user created in the database should be a user administrator who has the privileges to manage other users. See *Enable Client Access Control* (page 40).

You can also update existing users, such as to change password and grant or revoke roles. For a full list of user management methods, see *user-management-methods*.

Authentication Database

When adding a user, you create the user in a specific database. This database is the authentication database for the user.

A user can have privileges across different databases; i.e. a user's privileges are not limited to the authentication database. By assigning to the user roles in other databases, a user created in one database can have permissions to act on other databases. For more information on roles, see *Role-Based Access Control* (page 23).

The user's name and authentication database serve as a unique identifier for that user. That is, if two users have the same name but are created in different databases, they are two separate users. If you intend to have a single user with permissions on multiple databases, create a single user with roles in the applicable databases instead of creating the user multiple times in different databases.

Authenticate a User

To authenticate a user, either

- Use the command line authentication options (e.g. `-u`, `-p`, `--authenticationDatabase`) when connecting to the `mongod` or `mongos` instance, or
- Connect first to the `mongod` or `mongos` instance, and then run the `authenticate` command or the `db.auth()` method against the authentication database.

To authenticate, the client must authenticate the user against the user's *authentication database*.

For instance, if using the `mongo` shell as a client, you can specify the authentication database for the user with the `--authenticationDatabase` option.

Centralized User Data

Changed in version 2.6.

MongoDB stores all user information, including `name` (page 126), `password` (page 126), and the *user's authentication database* (page 126), in the `system.users` (page 125) collection in the `admin` database.

Do not access this collection directly but instead use the *user management commands*.

Sharded Cluster Users

To create users for a sharded cluster, connect to the `mongos` instance and add the users. Clients then authenticate these users through the `mongos` instances.

Changed in version 2.6: MongoDB stores these sharded cluster user data in the `admin` database of the *config servers*. Previously, the credentials for authenticating to a database on a sharded cluster resided on the *primary shard* for that database.

Shard Local Users

However, some maintenance operations, such as `cleanupOrphaned`, `compact`, `rs.reconfig()`, require direct connections to specific shards in a sharded cluster. To perform these operations, you must connect directly to the shard and authenticate as a *shard local* administrative user.

To create a *shard local* administrative user, connect directly to the shard and create the user. MongoDB stores *shard local* users in the `admin` database of the shard itself.

These *shard local* users are completely independent from the users added to the sharded cluster via `mongos`. *Shard local* users are local to the shard and are inaccessible by `mongos`.

Direct connections to a shard should only be for shard-specific maintenance and configuration. In general, clients should connect to the sharded cluster through the `mongos`.

Localhost Exception

The localhost exception allows you to enable access control and then create the first user in the system. With the localhost exception, after you enable access control, connect to the localhost interface and create the first user in the `admin` database. The first user must have privileges to create other users, such as a user with the `userAdmin` (page 117) or `userAdminAnyDatabase` (page 121) role.

Changed in version 3.0: The localhost exception changed so that these connections *only* have access to create the first user on the `admin` database. In previous versions, connections that gained access using the localhost exception had unrestricted access to the MongoDB instance.

The localhost exception applies only when there are no users created in the MongoDB instance.

In the case of a sharded cluster, the localhost exception applies to each shard individually as well as to the cluster as a whole. Once you create a sharded cluster and add a user administrator through the `mongos` instance, you must still prevent unauthorized access to the individual shards. Follow one of the following steps for each shard in your cluster:

- Create an administrative user, or
- Disable the localhost exception at startup. To disable the localhost exception, set the `enableLocalhostAuthBypass` parameter to 0.

2.4.2 Authentication Mechanisms

On this page

- [Default Authentication Mechanism](#) (page 10)
- [Specify Authentication Mechanism](#) (page 10)

MongoDB supports the following authentication mechanisms:

- *SCRAM-SHA-1* (page 11)
- *MongoDB Challenge and Response (MONGODB-CR)* (page 12)

Changed in version 3.0: New challenge-response users created in 3.0 will use `SCRAM-SHA-1`. If using 2.6 user data, MongoDB 3.0 will continue to use the `MONGODB-CR`.

- *x.509 Certificate Authentication* (page 12).

In addition, MongoDB Enterprise also provides supports for additional mechanisms. See [Enterprise Authentication Mechanisms](#) (page 14) for additional mechanisms available in MongoDB Enterprise.

Default Authentication Mechanism

Changed in version 3.0.

MongoDB uses the *SCRAM-SHA-1* (page 11) as the default challenge and response authentication mechanism. Previous versions used *MONGODB-CR* (page 12) as the default.

Specify Authentication Mechanism

To specify the authentication mechanism to use, set the `authenticationMechanisms` parameter for `mongod` and `mongos`.

Clients specify the authentication mechanism in the `db.auth()` method. For the `mongo` shell and the MongoDB tools, you can also specify the authentication mechanism from the command line.

SCRAM-SHA-1

On this page

- [SCRAM-SHA-1 Advantages](#) (page 11)
- [SCRAM-SHA-1 and MongoDB-CR User Credentials](#) (page 11)
- [Additional Information](#) (page 11)

New in version 3.0.

SCRAM-SHA-1 is the default authentication mechanism for MongoDB. SCRAM-SHA-1 is an IETF standard, [RFC 5802](#)¹, that defines best practice methods for implementation of challenge-response mechanisms for authenticating users with passwords.

SCRAM-SHA-1 verifies the supplied user credentials against the user's `name` (page 126), `password` (page 126) and `authentication database` (page 126). The authentication database is the database where the user was created, and together with the user's name, serves to identify the user.

Note: A driver upgrade is **necessary** to use the SCRAM-SHA-1 authentication mechanism if your current driver version does not support SCRAM-SHA-1. See *required driver versions* for details.

SCRAM-SHA-1 Advantages MongoDB's implementation of SCRAM-SHA-1 represents an improvement in security over the previously-used MONGODB-CR, providing:

- A tunable work factor (`iterationCount`),
- Per-user random salts rather than server-wide salts,
- A cryptographically stronger hash function (SHA-1 rather than MD5), and
- Authentication of the server to the client as well as the client to the server.

SCRAM-SHA-1 and MongoDB-CR User Credentials SCRAM-SHA-1 is the default mechanism for MongoDB versions beginning with the 3.0 series. However, if you are upgrading a MongoDB 2.6 instances that already have users credentials, MongoDB will continue to use MONGODB-CR for challenge-response authentication until you upgrade the authentication schema.

Even when using the MONGODB-CR authentication mechanism, clients and drivers that support MongoDB 3.0 features (see *compatibility-driver-versions*) will use the SCRAM communication protocol. That is, MONGODB-CR authentication mechanism also implies *SCRAM-SHA-1* (page 11).

For details on upgrading the authentication schema model to SCRAM-SHA-1, see <https://docs.mongodb.org/manual/release-notes/3.0-scram>.

Warning: The procedure to upgrade to SCRAM-SHA-1 **discards** the MONGODB-CR credentials used by 2.6. As such, the procedure is **irreversible**, short of restoring from backups. The procedure also disables MONGODB-CR as an authentication mechanism.

Additional Information

- [Blog Post: Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained \(Part 1\)](#)²

¹<https://tools.ietf.org/html/rfc5802>

²<https://www.mongodb.com/blog/post/improved-password-based-authentication-mongodb-30-scram-explained-part-1?jmp=docs>

- [Blog Post: Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained \(Part 2\)](#)³

MONGODB-CR

On this page

- [MONGODB-CR and SCRAM-SHA-1](#) (page 12)

MONGODB-CR is a challenge-response mechanism that authenticates users through passwords. MONGODB-CR verifies supplied user credentials against the user's `name` (page 126), `password` (page 126) and `authentication database` (page 126). The authentication database is the database where the user was created, and the user's database and the user's name together serve to identify the user.

MONGODB-CR and SCRAM-SHA-1 Changed in version 3.0.

MongoDB no longer defaults to MONGODB-CR and instead uses SCRAM-SHA-1 as the default authentication mechanism.

Even when using the MONGODB-CR authentication mechanism, clients and drivers that support MongoDB 3.0 features (see *compatibility-driver-versions*) will use the SCRAM communication protocol. That is, MONGODB-CR authentication mechanism also implies *SCRAM-SHA-1* (page 11).

x.509

On this page

- [Certificate Authority](#) (page 12)
- [Client x.509 Certificates](#) (page 12)
- [Member x.509 Certificates](#) (page 13)

New in version 2.6.

MongoDB supports x.509 certificate authentication for client authentication and internal authentication of the members of replica sets and sharded clusters.

x.509 certificate authentication requires a secure *TLS/SSL connection* (page 79).

Certificate Authority For production use, your MongoDB deployment should use valid certificates generated and signed by a single certificate authority. You or your organization can generate and maintain an independent certificate authority, or use certificates generated by a third-party SSL vendor. Obtaining and managing certificates is beyond the scope of this documentation.

Client x.509 Certificates To authenticate to servers, clients can use x.509 certificates instead of usernames and passwords.

³<https://www.mongodb.com/blog/post/improved-password-based-authentication-mongodb-30-scr-am-explained-part-2?jmp=docs>

Client Certificate Requirements The client certificate must have the following properties:

- A single Certificate Authority (CA) must issue the certificates for both the client and the server.
- Client certificates must contain the following fields:

```
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
```

- Each unique MongoDB user must have a unique certificate.
- A client x.509 certificate's subject, which contains the Distinguished Name (DN), must **differ** from that of a *Member x.509 Certificate* (page 51). Specifically, the subjects must differ with regards to at least one of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).

Warning: If a client x.509 certificate's subject has the same O, OU, and DC combination as the *Member x.509 Certificate* (page 51), the client will be identified as a cluster member and granted full permission on the system.

MongoDB User and \$external Database To authenticate with a client certificate, you must first add the value of the `subject` from the client certificate as a MongoDB user. Each unique x.509 client certificate corresponds to a single MongoDB user; i.e. you cannot use a single client certificate to authenticate more than one MongoDB user.

Add the user in the `$external` database; i.e. the *Authentication Database* (page 8) is the `$external` database

Authenticate To authenticate using x.509 client certificate, connect to MongoDB over TLS/SSL connection; i.e. include the `--ssl` and `--sslPEMKeyFile` command line options.

Then in the `$external` database, use `db.auth()` to authenticate the *user corresponding to the client certificate* (page 13).

For an example, see *Use x.509 Certificates to Authenticate Clients* (page 48)

Member x.509 Certificates For internal authentication, members of sharded clusters and replica sets can use x.509 certificates instead of keyfiles, which use *MONGODB-CR* (page 12) authentication mechanism.

Member Certificate Requirements The member certificate, used for internal authentication to verify membership to the sharded cluster or a replica set, must have the following properties:

- A single Certificate Authority (CA) must issue all the x.509 certificates for the members of a sharded cluster or a replica set.
- The Distinguished Name (DN), found in the member certificate's `subject`, must specify a non-empty value for *at least one* of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).
- The Organization attributes (O's), the Organizational Unit attributes (OU's), and the Domain Components (DC's) must match those from the certificates for the other cluster members. To match, the certificate must match all specifications of these attributes, or even the non-specification of these attributes. The order of the attributes does not matter.

In the following example, the two DN's contain matching specifications for O, OU as well as the non-specification of the DC attribute.

```
CN=host1,OU=Dept1,O=MongoDB,ST=NY,C=US
C=US,ST=CA,O=MongoDB,OU=Dept1,CN=host2
```

However, the following two DN's contain a mismatch for the OU attribute since one contains two OU specifications and the other, only one specification.

```
CN=host1,OU=Dept1,OU=Sales,O=MongoDB
CN=host2,OU=Dept1,O=MongoDB
```

- Either the Common Name (CN) or one of the Subject Alternative Name (SAN) entries must match the hostname of the server, used by the other members of the cluster.

For example, the certificates for a cluster could have the following subjects:

```
subject= CN=<myhostname1>,OU=Dept1,O=MongoDB,ST=NY,C=US
subject= CN=<myhostname2>,OU=Dept1,O=MongoDB,ST=NY,C=US
subject= CN=<myhostname3>,OU=Dept1,O=MongoDB,ST=NY,C=US
```

- If the certificate includes the Extended Key Usage (extendedKeyUsage) setting, the value must include clientAuth (“TLS Web Client Authentication”).

```
extendedKeyUsage = clientAuth
```

You can also use a certificate that does not include the Extended Key Usage (EKU).

MongoDB Configuration To specify x.509 for internal authentication, in addition to the other SSL configurations appropriate for your deployment, for each member of the replica set or sharded cluster, include either:

- `security.clusterAuthMode` and `net.ssl.clusterFile` if using a configuration file, or
- `--clusterAuthMode` and `--sslClusterFile` command line options.

Member Certificate and PEMKeyFile To configure MongoDB for client certificate authentication, the `mongod` and `mongos` specify a `PEMKeyFile` to prove its identity to clients, either through `net.ssl.PEMKeyFile` setting in the configuration file or `--sslPEMKeyFile` command line option.

If no `clusterFile` certificate is specified for internal member authentication, MongoDB will attempt to use the `PEMKeyFile` certificate for member authentication. In order to use `PEMKeyFile` certificate for internal authentication as well as for client authentication, then the `PEMKeyFile` certificate must either:

- Omit `extendedKeyUsage` or
- Specify `extendedKeyUsage` values that include `clientAuth` in addition to `serverAuth`.

For an example of x.509 internal authentication, see *Use x.509 Certificate for Membership Authentication* (page 51).

2.4.3 Enterprise Authentication Mechanisms

On this page

- [Kerberos Authentication](#) (page 15)
- [LDAP Proxy Authority Authentication](#) (page 15)

In addition to the authentication mechanisms offered, MongoDB Enterprise provides integration with the following authentication mechanisms.

Kerberos Authentication

MongoDB Enterprise⁴ supports authentication using a Kerberos service. Kerberos is an industry standard authentication protocol for large client/server systems.

To use MongoDB with Kerberos, you must have a properly configured Kerberos deployment, configured *Kerberos service principals* (page 16) for MongoDB, and added *Kerberos user principal* (page 16) to MongoDB.

For more information on Kerberos and MongoDB, see:

- *Kerberos Authentication* (page 15),
- *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and
- *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

LDAP Proxy Authority Authentication

MongoDB Enterprise (excluding Windows version)⁵ supports proxy authentication through a Lightweight Directory Access Protocol (LDAP) service.

LDAP support for user authentication requires proper configuration of the `saslauthd` daemon process as well as the MongoDB server.

For more information on LDAP and MongoDB, see

- *LDAP Proxy Authority Authentication* (page 18),
- *Authenticate Using SASL and LDAP with OpenLDAP* (page 66) and
- *Authenticate Using SASL and LDAP with ActiveDirectory* (page 63).

Kerberos Authentication

On this page

- *Overview* (page 15)
- *Kerberos Components and MongoDB* (page 15)
- *Operational Considerations* (page 17)
- *Kerberized MongoDB Environments* (page 17)
- *Additional Resources* (page 18)

New in version 2.4.

Overview MongoDB Enterprise provides support for Kerberos authentication of MongoDB clients to `mongod` and `mongos`. Kerberos is an industry standard authentication protocol for large client/server systems. Kerberos allows MongoDB and applications to take advantage of existing authentication infrastructure and processes.

Kerberos Components and MongoDB

⁴<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

⁵<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

Principals In a Kerberos-based system, every participant in the authenticated communication is known as a “principal”, and every principal must have a unique name.

Principals belong to administrative units called *realms*. For each realm, the Kerberos Key Distribution Center (KDC) maintains a database of the realm’s principal and the principals’ associated “secret keys”.

For a client-server authentication, the client requests from the KDC a “ticket” for access to a specific asset. KDC uses the client’s secret and the server’s secret to construct the ticket which allows the client and server to mutually authenticate each other, while keeping the secrets hidden.

For the configuration of MongoDB for Kerberos support, two kinds of principal names are of interest: *user principals* (page 16) and *service principals* (page 16).

User Principal To authenticate using Kerberos, you must add the Kerberos user principals to MongoDB to the `$external` database. User principal names have the form:

```
<username>@<KERBEROS REALM>
```

For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB in the `$external` database.

For examples of adding a user to MongoDB as well as authenticating as that user, see *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

See also:

Configure Users and Roles (page 70) for general information regarding creating and managing users in MongoDB.

Service Principal Every MongoDB `mongod` and `mongos` instance (or `mongod.exe` or `mongos.exe` on Windows) must have an associated service principal. Service principal names have the form:

```
<service>/<fully qualified domain name>@<KERBEROS REALM>
```

For MongoDB, the `<service>` defaults to `mongodb`. For example, if `m1.example.com` is a MongoDB server, and `example.com` maintains the `EXAMPLE.COM` Kerberos realm, then `m1` should have the service principal name `mongodb/m1.example.com@EXAMPLE.COM`.

To specify a different value for `<service>`, use `serviceName` during the start up of `mongod` or `mongos` (or `mongod.exe` or `mongos.exe`). `mongo shell` or other clients may also specify a different service principal name using `serviceName`.

Service principal names must be reachable over the network using the fully qualified domain name (FQDN) part of its service principal name.

By default, Kerberos attempts to identify hosts using the `/etc/kerb5.conf` file before using DNS to resolve hosts.

On Windows, if running MongoDB as a service, see *Assign Service Principal Name to MongoDB Windows Service* (page 61).

Linux Keytab Files Linux systems can store Kerberos authentication keys for a *service principal* (page 16) in *keytab* files. Each Kerberized `mongod` and `mongos` instance running on Linux must have access to a keytab file containing keys for its *service principal* (page 16).

To keep keytab files secure, use file permissions that restrict access to only the user that runs the `mongod` or `mongos` process.

Tickets On Linux, MongoDB clients can use Kerberos’s `kinit` program to initialize a credential cache for authenticating the user principal to servers.

Windows Active Directory Unlike on Linux systems, `mongod` and `mongos` instances running on Windows do not require access to keytab files. Instead, the `mongod` and `mongos` instances read their server credentials from a credential store specific to the operating system.

However, from the Windows Active Directory, you can export a keytab file for use on Linux systems. See [Ktpass⁶](#) for more information.

Authenticate With Kerberos To configure MongoDB for Kerberos support and authenticate, see *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

Operational Considerations

The HTTP Console The MongoDB [HTTP Console⁷](#) interface does not support Kerberos authentication.

Deprecated since version 3.2: HTTP interface for MongoDB

DNS Each host that runs a `mongod` or `mongos` instance must have both `A` and `PTR` DNS records to provide forward and reverse lookup.

Without `A` and `PTR` DNS records, the host cannot resolve the components of the Kerberos domain or the Key Distribution Center (KDC).

System Time Synchronization To successfully authenticate, the system time for each `mongod` and `mongos` instance must be within 5 minutes of the system time of the other hosts in the Kerberos infrastructure.

Kerberized MongoDB Environments

Driver Support The following MongoDB drivers support Kerberos authentication:

- [C⁸](#)
- [C++⁹](#)
- [Java¹⁰](#)
- [C#¹¹](#)
- [Node.js¹²](#)
- [PHP¹³](#)
- [Python¹⁴](#)
- [Ruby¹⁵](#)

⁶<http://technet.microsoft.com/en-us/library/cc753771.aspx>

⁷<https://docs.mongodb.org/ecosystem/tools/http-interfaces/#http-console>

⁸<https://api.mongodb.org/c/current/authentication.html#kerberos>

⁹<https://docs.mongodb.org/ecosystem/tutorial/authenticate-with-cpp-driver/>

¹⁰<https://docs.mongodb.org/ecosystem/tutorial/authenticate-with-java-driver/>

¹¹<http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/authentication/#gssapi-kerberos>

¹²http://mongodb.github.io/node-mongodb-native/2.0/tutorials/enterprise_features/

¹³<http://php.net/manual/en/mongoclient.construct.php>

¹⁴<http://api.mongodb.org/python/current/examples/authentication.html>

¹⁵<https://docs.mongodb.org/ecosystem/tutorial/ruby-driver-tutorial/#gssapi-kerberos-mechanism>

Use with Additional MongoDB Authentication Mechanism Although MongoDB supports the use of Kerberos authentication with other authentication mechanisms, only add the other mechanisms as necessary. See the [Incorporate Additional Authentication Mechanisms](#) section in *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59) for details.

Additional Resources

- [MongoDB LDAP and Kerberos Authentication with Dell \(Quest\) Authentication Services](#)¹⁶
- [MongoDB with Red Hat Enterprise Linux Identity Management and Kerberos](#)¹⁷

LDAP Proxy Authority Authentication

On this page

- [Considerations](#) (page 18)
- [MongoDB Configuration](#) (page 18)
- [LDAP User](#) (page 19)
- [Additional Information](#) (page 19)

MongoDB Enterprise¹⁸ supports proxy authentication through a Lightweight Directory Access Protocol (LDAP) service.

Considerations MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards. See <https://docs.mongodb.org/manual/release-notes/2.6-upgrade> for upgrade instructions.

Use secure encrypted or trusted connections between clients and the server, as well as between `saslauthd` and the LDAP server. The LDAP server uses the SASL PLAIN mechanism, sending and receiving data in **plain text**. You should use only a trusted channel such as a VPN, a connection encrypted with TLS/SSL, or a trusted wired network.

MongoDB Configuration To configure the MongoDB server to use LDAP authentication mechanism, use the following command line options:

- `--auth` to enable access control,
- `--authenticationMechanisms` set to `PLAIN`, and
- `--saslauthdPath` parameter set to the path to the Unix-domain Socket of the `saslauthd` instance.

Or, if using the YAML configuration file, use the following settings:

- `security.authorization` set to `enabled`,
- `setParameter.authenticationMechanisms` set to `PLAIN`, and
- `setParameter.saslauthdPath` set to the path to the Unix-domain Socket of the `saslauthd` instance.

¹⁶<https://www.mongodb.com/blog/post/mongodb-ldap-and-kerberos-authentication-dell-quest-authentication-services?jmp=docs>

¹⁷<http://docs.mongodb.org/ecosystem/tutorial/manage-red-hat-enterprise-linux-identity-management?jmp=docs>

¹⁸<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

LDAP User In order to authenticate a user with the LDAP authentication mechanism, add a corresponding *user* (page 8) to the `$external` database. You do not need to save the user's password in MongoDB.

The `$external` database is the *authentication database* (page 8) for the LDAP user. To authenticate the LDAP user, you must authenticate against the `$external` database. When authenticating, specify `PLAIN` for the authentication mechanism.

LDAP authentication requires that MongoDB forward the user's password in plan text. As such, you must specify `digestPassword` set to `false` during authentication.

Additional Information For information on configuring MongoDB to use LDAP and authenticating users using LDAP, see:

- *Authenticate Using SASL and LDAP with OpenLDAP* (page 66) and
- *Authenticate Using SASL and LDAP with ActiveDirectory* (page 63).

2.4.4 Internal Authentication

On this page

- *Keyfiles* (page 19)
- *x.509* (page 20)

You can authenticate members of *replica sets* and *sharded clusters*. For the internal authentication of the members, MongoDB can use either keyfiles or *x.509* (page 12) certificates.

Note: Enabling internal authentication also enables *client authorization* (page 23).

Keyfiles

Keyfiles use *SCRAM-SHA-1* (page 11) challenge and response authentication mechanism. The contents of the keyfiles serve as the shared password for the members. A key's length must be between 6 and 1024 characters and may only contain characters in the base64 set.

MongoDB strips whitespace characters (e.g. `x0d`, `x09`, and `x20`) for cross-platform convenience. As a result, the following operations produce identical keys:

```
echo -e "my secret key" > key1
echo -e "my secret key\n" > key2
echo -e "my  secret  key" > key3
echo -e "my\r\nsecret\r\nkey\r\n" > key4
```

On UNIX systems, the keyfile must not have group or world permissions. On Windows systems, keyfile permissions are not checked.

The content of the keyfile must be the same on all `mongod` and `mongos` instances that connect to each other. You must store the keyfile on each member of the replica set or sharded clusters.

To specify the keyfile, use the `security.keyFile` setting or `--keyFile` command line option.

For an example of keyfile internal authentication, see *Enable Internal Authentication* (page 42).

x.509

Members of a replica set or sharded cluster can use x.509 certificates for internal authentication instead of using keyfiles. MongoDB supports x.509 certificate authentication for use with a secure TLS/SSL connection.

Member Certificate Requirements

The member certificate, used for internal authentication to verify membership to the sharded cluster or a replica set, must have the following properties:

- A single Certificate Authority (CA) must issue all the x.509 certificates for the members of a sharded cluster or a replica set.
- The Distinguished Name (DN), found in the member certificate's `subject`, must specify a non-empty value for *at least one* of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).
- The Organization attributes (O's), the Organizational Unit attributes (OU's), and the Domain Components (DC's) must match those from the certificates for the other cluster members. To match, the certificate must match all specifications of these attributes, or even the non-specification of these attributes. The order of the attributes does not matter.

In the following example, the two DN's contain matching specifications for O, OU as well as the non-specification of the DC attribute.

```
CN=host1,OU=Dept1,O=MongoDB,ST=NY,C=US
C=US,ST=CA,O=MongoDB,OU=Dept1,CN=host2
```

However, the following two DN's contain a mismatch for the OU attribute since one contains two OU specifications and the other, only one specification.

```
CN=host1,OU=Dept1,OU=Sales,O=MongoDB
CN=host2,OU=Dept1,O=MongoDB
```

- Either the Common Name (CN) or one of the Subject Alternative Name (SAN) entries must match the hostname of the server, used by the other members of the cluster.

For example, the certificates for a cluster could have the following subjects:

```
subject= CN=<myhostname1>,OU=Dept1,O=MongoDB,ST=NY,C=US
subject= CN=<myhostname2>,OU=Dept1,O=MongoDB,ST=NY,C=US
subject= CN=<myhostname3>,OU=Dept1,O=MongoDB,ST=NY,C=US
```

- *If* the certificate includes the Extended Key Usage (`extendedKeyUsage`) setting, the value must include `clientAuth` ("TLS Web Client Authentication").

```
extendedKeyUsage = clientAuth
```

You can also use a certificate that does not include the Extended Key Usage (EKU).

MongoDB Configuration

To specify x.509 for internal authentication, in addition to the other SSL configurations appropriate for your deployment, for each member of the replica set or sharded cluster, include either:

- `security.clusterAuthMode` and `net.ssl.clusterFile` if using a configuration file, or
- `--clusterAuthMode` and `--sslClusterFile` command line options.

Member Certificate and PEMKeyFile

To configure MongoDB for client certificate authentication, the `mongod` and `mongos` specify a `PEMKeyFile` to prove its identity to clients, either through `net.ssl.PEMKeyFile` setting in the configuration file or `--sslPEMKeyFile` command line option.

If no `clusterFile` certificate is specified for internal member authentication, MongoDB will attempt to use the `PEMKeyFile` certificate for member authentication. In order to use `PEMKeyFile` certificate for internal authentication as well as for client authentication, then the `PEMKeyFile` certificate must either:

- Omit `extendedKeyUsage` or
- Specify `extendedKeyUsage` values that include `clientAuth` in addition to `serverAuth`.

For an example of x.509 internal authentication, see *Use x.509 Certificate for Membership Authentication* (page 51).

To upgrade from keyfile internal authentication to x.509 internal authentication, see *Upgrade from Keyfile Authentication to x.509 Authentication* (page 53).

Role-Based Access Control

On this page

- [Enable Access Control](#) (page 23)
- [Roles](#) (page 23)
- [Users and Roles](#) (page 24)
- [Built-In Roles and User-Defined Roles](#) (page 24)

MongoDB employs Role-Based Access Control (RBAC) to govern access to a MongoDB system. A user is granted one or more *roles* (page 23) that determine the user's access to database resources and operations. Outside of role assignments, the user has no access to the system.

3.1 Enable Access Control

MongoDB does not enable access control by default. You can enable authorization using the `--auth` or the `security.authorization` setting. Enabling *internal authentication* (page 19) also enables client authorization.

Once access control is enabled, users must *authenticate* (page 7) themselves.

3.2 Roles

A role grants privileges to perform the specified *actions* (page 129) on *resource* (page 127). Each privilege is either specified explicitly in the role or inherited from another role or both.

3.2.1 Privileges

A privilege consists of a specified resource and the actions permitted on the resource.

A *resource* (page 127) is either a database, collection, set of collections, or the cluster. If the resource is the cluster, the affiliated actions affect the state of the system rather than a specific database or collection. For information on the resource documents, see *Resource Document* (page 127).

An *action* (page 129) specifies the operation allowed on the resource. For available actions see *Privilege Actions* (page 129).

3.2.2 Inherited Privileges

A role can include one or more existing roles in its definition, in which case the role inherits all the privileges of the included roles.

A role can inherit privileges from other roles in its database. A role created on the `admin` database can inherit privileges from roles in any database.

3.2.3 View Role's Privileges

You can view the privileges for a role by issuing the `rolesInfo` command with the `showPrivileges` and `showBuiltinRoles` fields both set to `true`.

3.3 Users and Roles

You can assign roles to users during the user creation. You can also update existing users to grant or revoke roles. For a full list of user management methods, see *user-management-methods*

A user assigned a role receives all the privileges of that role. A user can have multiple roles. By assigning to the user roles in various databases, a user created in one database can have permissions to act on other databases.

Note: The first user created in the database should be a user administrator who has the privileges to manage other users. See *Enable Client Access Control* (page 40).

3.4 Built-In Roles and User-Defined Roles

MongoDB provides *built-in roles* (page 24) that provide set of privileges commonly needed in a database system.

If these built-in-roles cannot provide the desired set of privileges, MongoDB provides methods to create and modify *user-defined roles* (page 27).

3.4.1 Built-In Roles

On this page

- Database User Roles (page 25)
- Database Administration Roles (page 25)
- Cluster Administration Roles (page 25)
- Backup and Restoration Roles (page 25)
- All-Database Roles (page 26)
- Superuser Roles (page 26)
- Internal Role (page 27)

MongoDB provides built-in roles that provide the different levels of access commonly needed in a database system. Built-in *database user roles* (page 115) and *database administration roles* (page 116) roles exist in *each* database. The `admin` database contains additional roles.

This page provides a brief description of the built-in roles. For the specific privileges granted by each role, see the *Built-In Roles* (page 114) reference page.

Database User Roles

Every database includes the following roles:

Role	Short Description
<code>read</code> (page 115)	Provides the ability to read data on all <i>non</i> -system collections and on the following system collections: <code>system.indexes</code> , <code>system.js</code> , and <code>system.namespaces</code> collections. For the specific privileges granted by the role, see <code>read</code> (page 115).
<code>readWrite</code> (page 115)	Provides all the privileges of the <code>read</code> (page 115) role and the ability to modify data on all <i>non</i> -system collections and the <code>system.js</code> collection. For the specific privileges granted by the role, see <code>readWrite</code> (page 115).

Database Administration Roles

Every database includes the following database administration roles:

Role	Short Description
<code>dbAdmin</code> (page 116)	Provides the ability to perform administrative tasks such as schema-related tasks, indexing, gathering statistics. This role does not grant privileges for user and role management. For the specific privileges granted by the role, see <code>dbAdmin</code> (page 116).
<code>dbOwner</code> (page 116)	Provides the ability to perform any administrative action on the database. This role combines the privileges granted by the <code>readWrite</code> (page 115), <code>dbAdmin</code> (page 116) and <code>userAdmin</code> (page 117) roles.
<code>userAdmin</code> (page 117)	Provides the ability to create and modify roles and users on the current database. Since the <code>userAdmin</code> (page 117) role allows users to grant any privilege to any user, including themselves, the role also indirectly provides <i>superuser</i> (page 122) access to either the database or, if scoped to the <code>admin</code> database, the cluster. For the specific privileges granted by the role, see <code>userAdmin</code> (page 117).

Cluster Administration Roles

The `admin` database includes the following roles for administering the whole system rather than a specific database. These roles include but are not limited to *replica set* and *sharded cluster* administrative functions.

Role	Short Description
<code>clusterAdmin</code> (page 117)	Provides the greatest cluster-management access. This role combines the privileges granted by the <code>clusterManager</code> (page 117), <code>clusterMonitor</code> (page 118), and <code>hostManager</code> (page 119) roles. Additionally, the role provides the <code>dropDatabase</code> (page 132) action.
<code>clusterManager</code> (page 117)	Provides management and monitoring actions on the cluster. A user with this role can access the <code>config</code> and <code>local</code> databases, which are used in sharding and replication, respectively. For the specific privileges granted by the role, see <code>clusterManager</code> (page 117).
<code>clusterMonitor</code> (page 118)	Provides read-only access to monitoring tools, such as the MongoDB Cloud Manager ¹ and Ops Manager ² monitoring agent. For the specific privileges granted by the role, see <code>clusterMonitor</code> (page 118).
<code>hostManager</code> (page 119)	Provides the ability to monitor and manage servers. For the specific privileges granted by the role, see <code>hostManager</code> (page 119).

Backup and Restoration Roles

The `admin` database includes the following roles for backing up and restoring data:

¹<https://cloud.mongodb.com/?jmp=docs>

²<https://docs.opsmanager.mongodb.com/current/>

Role	Short Description
<code>backup</code> (page 120)	Provides privileges needed to back up data. This role provides sufficient privileges to use the MongoDB Cloud Manager³ backup agent, Ops Manager⁴ backup agent, or to use <code>mongodump</code> . For the specific privileges granted by the role, see <code>backup</code> (page 120).
<code>restore</code> (page 120)	Provides privileges needed to restore data with <code>mongorestore</code> without the <code>--oplogReplay</code> option or without <code>system.profile</code> collection data. For the specific privileges granted by the role, see <code>restore</code> (page 120).

All-Database Roles

The admin database provides the following roles that apply to all databases in a `mongod` instance and are roughly equivalent to their single-database equivalents:

Role	Short Description
<code>readAnyDatabase</code> (page 121)	Provides the same read-only permissions as <code>read</code> (page 115), except it applies to <i>all</i> databases in the cluster. The role also provides the <code>listDatabases</code> (page 134) action on the cluster as a whole. For the specific privileges granted by the role, see <code>readAnyDatabase</code> (page 121).
<code>readWriteAnyDatabase</code> (page 121)	Provides the same read and write permissions as <code>readWrite</code> (page 115), except it applies to <i>all</i> databases in the cluster. The role also provides the <code>listDatabases</code> (page 134) action on the cluster as a whole. For the specific privileges granted by the role, see <code>readWriteAnyDatabase</code> (page 121).
<code>userAdminAnyDatabase</code> (page 121)	Provides the same access to user administration operations as <code>userAdmin</code> (page 117), except it applies to <i>all</i> databases in the cluster. Since the <code>userAdminAnyDatabase</code> (page 121) role allows users to grant any privilege to any user, including themselves, the role also indirectly provides <i>superuser</i> (page 122) access. For the specific privileges granted by the role, see <code>userAdminAnyDatabase</code> (page 121).
<code>dbAdminAnyDatabase</code> (page 122)	Provides the same access to database administration operations as <code>dbAdmin</code> (page 116), except it applies to <i>all</i> databases in the cluster. The role also provides the <code>listDatabases</code> (page 134) action on the cluster as a whole. For the specific privileges granted by the role, see <code>dbAdminAnyDatabase</code> (page 122).

Superuser Roles

The following role provides full privileges on all resources:

Role	Short Description
<code>root</code> (page 122)	Provides access to the operations and all the resources of the <code>readWriteAnyDatabase</code> (page 121), <code>dbAdminAnyDatabase</code> (page 122), <code>userAdminAnyDatabase</code> (page 121) and <code>clusterAdmin</code> (page 117) roles <i>combined</i> . For the specific privileges granted by the role, see <code>root</code> (page 122).

³<https://cloud.mongodb.com/?jmp=docs>

⁴<https://docs.opsmanager.mongodb.com/current/>

Internal Role

Role	Short Description
<code>__system</code> (page 122)	Provides privileges to take any action against any object in the database. Do not assign this role to user objects representing applications or human administrators, other than in exceptional circumstances. For more information, see <code>root</code> (page 122).

See also:

Built-In Roles (page 114)

3.4.2 User-Defined Roles

On this page

- [Role Management Interface](#) (page 27)
- [Scope](#) (page 27)
- [Centralized Role Data](#) (page 27)

New in version 2.6.

MongoDB provides a number of *built-in roles* (page 114). However, if these roles cannot describe the desired set of privileges, you can create new roles.

Role Management Interface

To add a role, MongoDB provides the `db.createRole()` method. MongoDB also provides methods to update existing user-defined roles. For a full list of role management methods, see *role-management-methods*.

Scope

When adding a role, you create the role in a specific database. MongoDB uses the combination of the database and the role name to uniquely define a role.

Except for roles created in the `admin` database, a role can only include privileges that apply to its database and can only inherit from other roles in its database.

A role created in the `admin` database can include privileges that apply to the `admin` database, other databases or to the *cluster* (page 128) resource, and can inherit from roles in other databases as well as the `admin` database.

Centralized Role Data

MongoDB stores all role information in the *system.roles* (page 123) collection in the `admin` database

Do not access this collection directly but instead use the *role management commands* to view and edit custom roles.

3.4.3 Collection-Level Access Control

On this page

- [Privileges and Scope](#) (page 28)
- [Additional Information](#) (page 28)

Collection-level access control allows administrators to grant users privileges that are scoped to specific collections.

Administrators can implement collection-level access control through *user-defined roles* (page 27). By creating a role with *privileges* (page 23) that are scoped to a specific collection in a particular database, administrators can provision users with roles that grant privileges on a collection level.

Privileges and Scope

A privilege consists of *actions* (page 129) and the *resources* (page 127) upon which the actions are permissible; i.e. the resources define the scope of the actions for that privilege.

By specifying both the database and the collection in the *resource document* (page 127) for a privilege, administrator can limit the privilege actions just to a specific collection in a specific database. Each privilege action in a role can be scoped to a different collection.

For example, a user defined role can contain the following privileges:

```
privileges: [
  { resource: { db: "products", collection: "inventory" }, actions: [ "find", "update", "insert" ] },
  { resource: { db: "products", collection: "orders" }, actions: [ "find" ] }
]
```

The first privilege scopes its actions to the `inventory` collection of the `products` database. The second privilege scopes its actions to the `orders` collection of the `products` database.

Additional Information

For more information on user-defined roles and MongoDB authorization model, see *Role-Based Access Control* (page 23). For a tutorial on creating user-defined roles, see *Manage User and Roles* (page 70).

Encryption

On this page

- [Transport Encryption](#) (page 29)
- [Encryption at Rest](#) (page 29)

4.1 Transport Encryption

You can use TLS/SSL (Transport Layer Security/Secure Sockets Layer) to encrypt all of MongoDB's network traffic. TLS/SSL ensures that MongoDB network traffic is only readable by the intended client.

See *Transport Encryption* (page 29) for more information.

4.2 Encryption at Rest

There are two broad classes of approaches to encrypting data at rest with MongoDB: Application Level Encryption and Storage Encryption. You can use these solutions together or independently.

New in version 3.2: MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine. This feature allows MongoDB to encrypt data files such that only parties with the decryption key can decode and read the data.

See *Encryption At Rest* (page 30) for more information.

4.2.1 Transport Encryption

On this page

- [TLS/SSL](#) (page 30)
- [Certificates](#) (page 30)
- [Identity Verification](#) (page 30)
- [FIPS Mode](#) (page 30)

TLS/SSL

MongoDB supports TLS/SSL (Transport Layer Security/Secure Sockets Layer) to encrypt all of MongoDB's network traffic. TLS/SSL ensures that MongoDB network traffic is only readable by the intended client.

MongoDB TLS/SSL implementation uses OpenSSL libraries. MongoDB's SSL encryption only allows use of strong SSL ciphers with a minimum of 128-bit key length for all connections.

Certificates

Before you can use SSL, you must have a `.pem` file containing a public key certificate and its associated private key.

MongoDB can use any valid SSL certificate issued by a certificate authority or a self-signed certificate. If you use a self-signed certificate, although the communications channel will be encrypted, there will be *no* validation of server identity. Although such a situation will prevent eavesdropping on the connection, it leaves you vulnerable to a man-in-the-middle attack. Using a certificate signed by a trusted certificate authority will permit MongoDB drivers to verify the server's identity.

For example, see *TLS/SSL Configuration for Clients* (page 84).

Identity Verification

In addition to encrypting connections, SSL allows for authentication using certificates, both for *client authentication* (page 7) and for *internal authentication* (page 19) of members of replica sets and sharded clusters.

For more information, see:

- *Configure mongod and mongos for TLS/SSL* (page 79)
- *TLS/SSL Configuration for Clients* (page 84)
- *Use x.509 Certificates to Authenticate Clients* (page 48)
- *Use x.509 Certificate for Membership Authentication* (page 51)

FIPS Mode

Enterprise Feature

Available in MongoDB Enterprise only.

The Federal Information Processing Standard (FIPS) is a U.S. government computer security standard used to certify software modules and libraries that encrypt and decrypt data securely. You can configure MongoDB to run with a FIPS 140-2 certified library for OpenSSL. Configure FIPS to run by default or as needed from the command line.

For an example, see *Configure MongoDB for FIPS* (page 88).

4.2.2 Encryption At Rest

On this page

- *Encrypted Storage Engine* (page 31)
- *Application Level Encryption* (page 32)
- *Third Party Storage Encryption* (page 32)

Encryption at rest, when used in conjunction with transport encryption and good security policies that protect relevant accounts, passwords, and encryption keys, can help ensure compliance with security and privacy standards, including HIPAA, PCI-DSS, and FERPA.

Encrypted Storage Engine

New in version 3.2.

Enterprise Feature

Available in MongoDB Enterprise only.

Important: Available for the WiredTiger Storage Engine only.

MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine. This feature allows MongoDB to encrypt data files such that only parties with the decryption key can decode and read the data.

Encryption Process

If encryption is enabled, the default encryption mode that MongoDB Enterprise uses is the AES256-CBC (or 256-bit Advanced Encryption Standard in Cipher Block Chaining mode) via OpenSSL. AES-256 uses a symmetric key; i.e. the same key to encrypt and decrypt text. MongoDB Enterprise also supports authenticated encryption AES256-GCM (or 256-bit Advanced Encryption Standard in Galois/Counter Mode). FIPS mode encryption is also available.

The data encryption includes:

- Generating an system key.
- Generating keys for each database.
- Encrypting data with the database keys.
- Encrypting the database keys with the system key.

The encryption occur transparently in the storage layer; i.e. all data files are fully encrypted from a filesystem perspective, and data only exists in an unencrypted state in memory and during transmission.

To encrypt all of MongoDB's network traffic, you can use TLS/SSL (Transport Layer Security/Secure Sockets Layer). See *Configure mongod and mongos for TLS/SSL* (page 79) and *TLS/SSL Configuration for Clients* (page 84).

Key Management

Important: Secure management of the encryption keys is critical.

The database keys are internal to the server and are only paged to disk in an encrypted format. MongoDB never pages the system key to disk under any circumstances.

Only the system key is external to the server (i.e. kept separate from the data and the database keys), and requires external management. To manage the system key, MongoDB's encrypted storage engine supports two key management options:

- Integration with a third party key management appliance via the Key Management Interoperability Protocol (KMIP). **Recommended**
- Local key management via a keyfile.

To configure MongoDB for encryption and use one of the two key management options, see *Configure Encryption* (page 97).

Encryption and Replication

Encryption is not a part of replication:

- System keys and database keys are not replicated, and
- Data is not natively encrypted over the wire.

Although you could reuse the same key for the nodes, MongoDB recommends the use of individual keys for each node as well as the use of transport encryption.

For details, see *Rotate Encryption Keys* (page 100).

Application Level Encryption

Application Level Encryption provides encryption on a per-field or per-document basis within the application layer. To encrypt document or field level data, write custom encryption and decryption routines or use a commercial solution such as the *Vormetric Data Security Platform*¹.

Third Party Storage Encryption

A number of third-party libraries can integrate with the operating system to provide transparent disk-level encryption. For example:

- **Linux Unified Key Setup (LUKS)** LUKS is available for most Linux distributions. For configuration explanation, see the *LUKS documentation from Red Hat*².
- **IBM Guardium Data Encryption** *IBM Guardium Data Encryption*³ provides support for disk-level encryption for Linux and Windows operating systems.
- **Vormetric Data Security Platform** The *Vormetric Data Security Platform*⁴ provides disk and file-level encryption in addition to application level encryption.
- **Bitlocker Drive Encryption** *Bitlocker Drive Encryption*⁵ is a feature available on Windows Server 2008 and 2012 that provides disk encryption.

¹<http://www.vormetric.com/sites/default/files/sb-MongoDB-Letter-2014-0611.pdf>

²https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/sec-Encryption.html

³<http://www-03.ibm.com/software/products/en/infosphere-guardium-data-encryption>

⁴<http://www.vormetric.com/sites/default/files/sb-MongoDB-Letter-2014-0611.pdf>

⁵<http://technet.microsoft.com/en-us/library/hh831713.aspx>

On this page

- [Enable and Configure Audit Output](#) (page 33)
- [Audit Events and Filter](#) (page 33)
- [Audit Guarantee](#) (page 34)

New in version 2.6.

MongoDB Enterprise includes an auditing capability for `mongod` and `mongos` instances. The auditing facility allows administrators and users to track system activity for deployments with multiple users and applications.

5.1 Enable and Configure Audit Output

The auditing facility can write audit events to the console, the *syslog*, a JSON file, or a BSON file. To enable auditing for MongoDB Enterprise, see [Configure Auditing](#) (page 102).

For information on the audit log messages, see [System Event Audit Messages](#) (page 134).

5.2 Audit Events and Filter

Once enabled, the auditing system can record the following operations:

- schema (DDL),
- replica set and sharded cluster,
- authentication and authorization, and
- CRUD operations (requires `auditAuthorizationSuccess` set to `true`).

For details on audited actions, see [Audit Event Actions, Details, and Results](#) (page 135).

With the auditing system, you can [set up filters](#) (page 104) to restrict the events captured. To set up filters, see [Configure Audit Filters](#) (page 104).

5.3 Audit Guarantee

The auditing system writes every audit event ¹ to an in-memory buffer of audit events. MongoDB writes this buffer to disk periodically. For events collected from any single connection, the events have a total order: if MongoDB writes one event to disk, the system guarantees that it has written all prior events for that connection to disk.

If an audit event entry corresponds to an operation that affects the durable state of the database, such as a modification to data, MongoDB will always write the audit event to disk *before* writing to the *journal* for that entry.

That is, before adding an operation to the journal, MongoDB writes all audit events on the connection that triggered the operation, up to and including the entry for the operation.

These auditing guarantees require that MongoDB run with `journaling` enabled.

Warning: MongoDB may lose events **if** the server terminates before it commits the events to the audit log. The client may receive confirmation of the event before MongoDB commits to the audit log. For example, while auditing an aggregation operation, the server might crash after returning the result but before the audit log flushes.

¹ Audit configuration can include a *filter* (page 104) to limit events to audit.

Security Hardening

On this page

- [MongoDB Configuration Hardening](#) (page 35)
- [Network Hardening](#) (page 35)

To reduce the risk exposure of the entire MongoDB system, ensure that only trusted hosts have access to MongoDB.

6.1 MongoDB Configuration Hardening

For MongoDB, ensure that HTTP status interface and the REST API are disabled in production to prevent potential data exposure to attackers.

Deprecated since version 3.2: HTTP interface for MongoDB

For more information, see *MongoDB Configuration Hardening* (page 35).

6.2 Network Hardening

To restrict exposure to MongoDB, configure firewalls to control access to MongoDB systems. Use of VPNs can also provide a secure tunnel.

For more information, see *Hardening Network Infrastructure* (page 37).

6.2.1 MongoDB Configuration Hardening

On this page

- [HTTP Status Interface](#) (page 36)
- [REST API](#) (page 36)
- [bind_ip](#) (page 37)

HTTP Status Interface

Warning: 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.

Deprecated since version 3.2: HTTP interface for MongoDB

Changed in version 2.6: The `mongod` and `mongos` instances run with the HTTP interface *disabled* by default. See `net.http.enabled` setting.

The HTTP status interface provides a web-based interface that includes a variety of operational data, logs, and status reports regarding the `mongod` or `mongos` instance. The HTTP status interface is *disabled* by default and is not recommended for production use.

The `net.http.enabled` setting enables HTTP status interface. When enabled without the `net.http.RESTInterfaceEnabled` setting, the HTTP interface is entirely read-only and limited in scope.

The HTTP interface uses the port that is 1000 greater than the primary `mongod` port. By default, the HTTP interface port is 28017, but is indirectly set using the `net.port` option which allows you to configure the primary `mongod` port.

The HTTP status interface does not include support for authentication other than `MONGODB-CR`.

While MongoDB Enterprise does support Kerberos authentication, Kerberos is not supported in HTTP status interface in any version of MongoDB.

Changed in version 3.0: Neither the HTTP status interface nor the REST API support the *SCRAM-SHA-1* (page 11) challenge-response user authentication mechanism introduced in version 3.0.

Warning: If you enable the interface, you should only allow trusted clients to access this port. See *Firewalls* (page 37).

REST API

Warning: 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.

The REST API to MongoDB provides additional information and write access on top of the HTTP status interface. While the REST API does not provide any support for insert, update, or remove operations, it does provide administrative access, and its accessibility represents a vulnerability in a secure environment.

Deprecated since version 3.2: HTTP interface for MongoDB

The REST interface is *disabled* by default and is not recommended for production use.

The `net.http.RESTInterfaceEnabled` setting for `mongod` enables a fully interactive administrative *REST* interface, which is *disabled* by default. Enabling the REST API enables the HTTP interface, even if the HTTP interface option is disabled, and makes the HTTP interface fully interactive.

The REST API does not include support for authentication other than `MONGODB-CR`.

Warning: If you enable the interface, you should only allow trusted clients to access this port. See *Firewalls* (page 37).

Changed in version 3.0: Neither the HTTP status interface nor the REST API support the *SCRAM-SHA-1* (page 11) challenge-response user authentication mechanism introduced in version 3.0.

`bind_ip`

The `net.bindIp` setting (or the `--bind_ip` command line option) for `mongod` and `mongos` instances limits the network interfaces on which MongoDB programs will listen for incoming connections.

Warning: Make sure that your `mongod` and `mongos` instances are only accessible on trusted networks. If your system has more than one network interface, bind MongoDB programs to the private or internal network interface.

See also:

Firewalls (page 37), *configuration-security*

6.2.2 Hardening Network Infrastructure

On this page

- [Firewalls](#) (page 37)
- [Virtual Private Networks](#) (page 37)

Firewalls

Firewalls allow administrators to filter and control access to a system by providing granular control over network communications. For administrators of MongoDB, the following capabilities are important: limiting incoming traffic on a specific port to specific systems and limiting incoming traffic from untrusted hosts.

On Linux systems, the `iptables` interface provides access to the underlying `netfilter` firewall. On Windows systems, `netsh` command line interface provides access to the underlying Windows Firewall. For additional information about firewall configuration, see:

- [Configure Linux iptables Firewall for MongoDB](#) (page 90) and
- [Configure Windows netsh Firewall for MongoDB](#) (page 94).

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.

See also:

For MongoDB deployments on Amazon's web services, see the [Amazon EC2¹](#) page, which addresses Amazon's Security Groups and other EC2-specific security features.

Virtual Private Networks

Virtual private networks, or VPNs, make it possible to link two networks over an encrypted and limited-access trusted network. Typically, MongoDB users who use VPNs use TLS/SSL rather than IPSEC VPNs for performance issues.

Depending on configuration and implementation, VPNs provide for certificate validation and a choice of encryption protocols, which requires a rigorous level of authentication and identification of all clients. Furthermore, because

¹<https://docs.mongodb.org/ecosystem/platforms/amazon-ec2>

VPNs provide a secure tunnel, by using a VPN connection to control access to your MongoDB instance, you can prevent tampering and “man-in-the-middle” attacks.

Security Tutorials

The following tutorials provide instructions for enabling and using the security features available in MongoDB.

Before enabling role based access control, you should first consider the users of the system. Once the users have been identified, determine the roles required by the users. Roles may inherit from other roles to provide a hierarchy.

Enable Access Control (page 39) Tutorials for enabling access control.

Authentication Mechanisms (page 47) Tutorials for specifying various authentication mechanisms supported by MongoDB.

Configure Users and Roles (page 70) Tutorials for managing users and roles.

Network (page 79) Tutorials for securing your network via TLS/SSL and firewall configuration.

Encryption (page 97) Tutorials for storage encryption.

Auditing (page 102) Tutorials for configuring auditing.

Miscellaneous (page 107) Tutorial illustrating field-level redaction or instructions for reporting a security vulnerability to MongoDB.

7.1 Enable Access Control

The tutorials in this section enable access control. Once access control is enabled, users must *authenticate* (page 7) themselves. The following tutorials use the *default authentication mechanism* (page 10) .

Important: Before enabling role based access control, you should first consider the users of the system. Once the users have been identified, determine the roles required by the users. Roles may inherit from other roles to provide a hierarchy.

A user should have only the minimal set of privileges required to ensure a system of *least privilege*.

Each application and user of a MongoDB system should map to a distinct user in MongoDB; i.e. do not create a group user that is shared among multiple individuals. This *access isolation* facilitates access revocation and ongoing user maintenance.

Enable Client Access Control (page 40) Describes the process for enabling client access control for MongoDB deployments.

Enable Internal Authentication (page 42) Describes the process for enabling internal authentication members of replica sets and sharded clusters. Enabling internal authentication implicitly enables client access control.

7.1.1 Enable Client Access Control

On this page

- [Overview](#) (page 40)
- [Considerations](#) (page 40)
- [Procedures](#) (page 40)
- [Additional Information](#) (page 42)

Overview

Enabling access control requires authentication of every user. Once authenticated, users only have the privileges as defined in the roles granted to the users.

To enable access control, use either the command line option `--auth` or `security.authorization` configuration file setting.

Note: The tutorial enables access control and uses the *default authentication mechanism* (page 10). To specify a different authentication mechanism, see *Authentication Mechanisms* (page 47).

You can also enable client access control by *enabling internal authentication* (page 42) of replica sets or sharded clusters. For instructions on enabling internal authentication, see *Enable Internal Authentication* (page 42).

Considerations

With access control enabled, ensure you have a user with `userAdmin` (page 117) or `userAdminAnyDatabase` (page 121) role in the `admin` database.

This tutorial assumes a *standalone* environment.

The *Enable Internal Authentication* (page 42) tutorial has steps specific to enabling access control on replica sets and sharded clusters.

You can create users before enabling access control or you can create users after enabling access control. If you enable access control before creating any user, MongoDB provides a *localhost exception* (page 10) which allows you to create a user administrator in the `admin` database. Once created, authenticate as the user administrator to create additional users as needed.

Procedures

Add Users Before Enabling Access Control

The following procedure first adds a user administrator to a MongoDB instance running without access control and then enables access control.

Step 1: Start MongoDB without access control. For example, the following starts a standalone `mongod` instance without access control.

```
mongod --port 27017 --dbpath /data/db1
```

For details on starting a `mongod` or `mongos`, see <https://docs.mongodb.org/manual/tutorial/manage-mongodb-p/> or <https://docs.mongodb.org/manual/tutorial/deploy-shard-cluster>.

Step 2: Connect to the instance. For example, connect a mongo shell to the instance.

```
mongo --port 27017
```

Specify additional command line options as appropriate to connect the mongo shell to your deployment, such as `--host`.

Step 3: Create the user administrator. Add a user with the `userAdminAnyDatabase` (page 121) role. For example, the following creates the user `myUserAdmin` on the `admin` database:

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Step 4: Re-start the MongoDB instance with access control. Re-start the `mongod` instance with the `--auth` command line option or, if using a configuration file, the `security.authorization` setting.

```
mongod --auth --port 27017 --dbpath /data/db1
```

Step 5: Authenticate as the user administrator. Either connect a new mongo shell to the MongoDB instance with the `-u <username>`, `-p <password>`, and the `--authenticationDatabase <database>`:

```
mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"
```

The mongo shell executes a number of commands at start up. As a result, when you log in as the user administrator, you may see authentication errors from one or more commands. You may ignore these errors, which are expected, because the `userAdminAnyDatabase` (page 121) role does not have permissions to run some of the start up commands.

Or, in the mongo shell connected without authentication, switch to the authentication database, and use `db.auth()` method to authenticate:

```
use admin
db.auth("myUserAdmin", "abc123" )
```

Step 5: Create additional users as needed for your deployment. If you need to disable access control for any reason, restart the MongoDB instance without the `--auth` command line option, or if using a configuration file, the `security.authorization` setting.

Add Users After Enabling Access Control

The following procedure first enables access control, and then uses *localhost exception* (page 10) to add a user administrator.

Step 1: Start the MongoDB instance with access control. Start the `mongod` instance with the `--auth` command line option or, if using a configuration file, the `security.authorization` setting.

```
mongod --auth --port 27017 --dbpath /data/db1
```

Step 2: Connect to the MongoDB instance via the localhost exception. To add the first user using *Localhost Exception* (page 10), connect a mongo shell to the mongod instance. Run the mongo shell from the same host as the mongod instance.

Step 3: Create the system user administrator. Add the user with the `userAdminAnyDatabase` (page 121) role, and only that role.

The following example creates the user `myUserAdmin` user on the `admin` database:

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

After you create the user administrator, the *localhost exception* (page 10) is no longer available.

Step 4: Authenticate as the user administrator. Either connect a new mongo shell to the MongoDB instance with the `-u <username>`, `-p <password>`, and the `--authenticationDatabase <database>`:

```
mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"
```

The mongo shell executes a number of commands at start up. As a result, when you log in as the user administrator, you may see authentication errors from one or more commands. You may ignore these errors, which are expected, because the `userAdminAnyDatabase` (page 121) role does not have permissions to run some of the start up commands.

Or, in the mongo shell connected without authentication, switch to the authentication database, and use `db.auth()` method to authenticate:

```
use admin
db.auth("myUserAdmin", "abc123" )
```

Step 5: Create additional users as needed for your deployment.

Additional Information

See also *Manage User and Roles* (page 70).

7.1.2 Enable Internal Authentication

On this page

- [Overview](#) (page 43)
- [Considerations](#) (page 43)
- [Procedures](#) (page 43)
- [x.509 Internal Authentication](#) (page 47)

Overview

When authentication is enabled on a replica set or a sharded cluster, members of the replica set or the sharded clusters must provide credentials to authenticate.

To enable authentication on a replica set or a sharded cluster, you must enable authentication individually for each member. For a sharded cluster, this means enabling authentication on each `mongos` and each `mongod`, including the config servers and each member of a shard's replica set.

The following tutorial uses a [keyfile](#) (page 19) to enable internal authentication. You can also use x.509 certificate for internal authentication. For details on using x.509, see [Use x.509 Certificate for Membership Authentication](#) (page 51).

Considerations

Access Control

Enabling internal authentication enables [access control](#) (page 23). The following tutorial assumes *no* users have been created in the system before enabling internal authentication, and uses [Localhost Exception](#) (page 10) to add a user administrator after access control has been enabled.

If you prefer, you can create the users before enabling internal authentication.

Sharded Cluster

It is not possible to convert an existing sharded cluster that does not enforce access control to require authentication without taking all components of the cluster offline for a short period of time.

For sharded clusters, the [Localhost Exception](#) (page 10) will apply to the individual shards unless you either create an administrative user or disable the localhost exception on each shard.

Procedures

Update Existing Deployment

Step 1: Create a keyfile. Create the [keyfile](#) (page 19) your deployment will use to authenticate to members to each other. You can generate a keyfile using any method you choose. Ensure that the password stored in the keyfile is both long and contains a high amount of randomness.

For example, the following operation uses `openssl` command to generate pseudo-random data to use for a keyfile:

```
openssl rand -base64 741 > /srv/mongodb/mongodb-keyfile
chmod 600 mongodb-keyfile
```


Step 2: Enable authentication for each member of the sharded cluster or replica set. For *each* `mongod` in the replica set or for *each* `mongos` and `mongod` in the sharded cluster, including all config servers and shards, specify the keyfile using either a configuration file or a command line option.

In a configuration file, set the `security.keyFile` option to the keyfile's path and then start the component, as in the following example:

```
security:
  keyFile: /srv/mongodb/keyfile
```

Include any other settings as appropriate for your deployment.

Or, when starting the component, specify the `--keyFile` option. For example, for a `mongod`

```
mongod --keyfile /srv/mongodb/mongodb-keyfile --dbpath <path to data>
```

Include any other options as appropriate for your deployment.

Enabling internal authentication enables *access control* (page 23).

Step 3: Connect to the MongoDB instance via the localhost exception. To add the first user using *Localhost Exception* (page 10):

- For a replica set, connect a mongo shell to the primary. Run the mongo shell from the same host as the primary.
- For a sharded cluster, connect a mongo shell to the mongos. Run the mongo shell from same host as the mongos.

Step 4: Add first user. Add a user with the `userAdminAnyDatabase` (page 121) role. For example, the following creates the user `myUserAdmin` on the `admin` database:

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

After you create the user administrator, for a replica set, the *localhost exception* (page 10) is no longer available.

For sharded clusters, you must still prevent unauthorized access to the individual shards. Follow one of the following steps for each shard in your cluster:

- Create an administrative user, or
- Disable the *Localhost Exception* (page 10) at startup. To disable the localhost exception, set the `enableLocalhostAuthBypass` to 0.

Step 5: Authenticate as the user administrator. Either connect a new mongo shell to the MongoDB instance with the `-u <username>`, `-p <password>`, and the `--authenticationDatabase <database>`:

```
mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"
```

The mongo shell executes a number of commands at start up. As a result, when you log in as the user administrator, you may see authentication errors from one or more commands. You may ignore these errors, which are expected, because the `userAdminAnyDatabase` (page 121) role does not have permissions to run some of the start up commands.

Or, in the `mongo` shell connected without authentication, switch to the authentication database, and use `db.auth()` method to authenticate:

```
use admin
db.auth("myUserAdmin", "abc123" )
```

Step 6: Create additional users as needed for your deployment.

Deploy New Replica Set with Access Control

Step 1: Start one member of the replica set. This `mongod` should *not* enable `auth`.

Step 2: Create administrative users. The following operations will create two users: a user administrator that will be able to create and modify users (`myUserAdmin`), and a `root` (page 122) user (`siteRootAdmin`) that you will use to complete the remainder of the tutorial:

```
use admin
db.createUser( {
  user: "myUserAdmin",
  pwd: "<password>",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
});
db.createUser( {
  user: "siteRootAdmin",
  pwd: "<password>",
  roles: [ { role: "root", db: "admin" } ]
});
```

Step 3: Stop the mongod instance.

Step 4: Create the key file to be used by each member of the replica set. Create the key file your deployment will use to authenticate servers to each other.

To generate pseudo-random data to use for a `keyfile`, issue the following `openssl` command:

```
openssl rand -base64 741 > mongodb-keyfile
chmod 600 mongodb-keyfile
```

You may generate a key file using any method you choose. Always ensure that the password stored in the key file is both long and contains a high amount of entropy. Using `openssl` in this manner helps generate such a key.

Step 5: Copy the key file to each member of the replica set. Copy the `mongodb-keyfile` to all hosts where components of a MongoDB deployment run. Set the permissions of these files to `600` so that only the *owner* of the file can read or write this file to prevent other users on the system from accessing the shared secret.

Step 6: Start each member of the replica set with the appropriate options. For each member, start a `mongod` and specify the key file and the name of the replica set. Also specify other parameters as needed for your deployment. For replication-specific parameters, see `cli-mongod-replica-set` required by your deployment.

If your application connects to more than one replica set, each set should have a distinct name. Some drivers group replica set connections by replica set name.

The following example specifies parameters through the `--keyFile` and `--replSet` command-line options:

```
mongod --keyFile /mysecretdirectory/mongodb-keyfile --replSet "rs0"
```

The following example specifies parameters through a configuration file:

```
mongod --config $HOME/.mongodb/config
```

In production deployments, you can configure a *init script* to manage this process. Init scripts are beyond the scope of this document.

Step 7: Connect to the member of the replica set where you created the administrative users. Connect to the replica set member you started and authenticate as the `siteRootAdmin` user. From the `mongo` shell, use the following operation to authenticate:

```
use admin
db.auth("siteRootAdmin", "<password>");
```

Step 8: Initiate the replica set. Use `rs.initiate()` on *one and only one* member of the replica set:

```
rs.initiate()
```

MongoDB initiates a set that consists of the current member and that uses the default replica set configuration.

Step 9: Verify the initial replica set configuration. Use `rs.conf()` to display the replica set configuration object:

```
rs.conf()
```

The replica set configuration object resembles the following:

```
{
  "_id" : "rs0",
  "version" : 1,
  "members" : [
    {
      "_id" : 1,
      "host" : "mongodb0.example.net:27017"
    }
  ]
}
```

Step 10: Add the remaining members to the replica set. Add the remaining members with the `rs.add()` method. You must be connected to the *primary* to add members to a replica set.

`rs.add()` can, in some cases, trigger an election. If the *mongod* you are connected to becomes a *secondary*, you need to connect the *mongo* shell to the new primary to continue adding new replica set members. Use `rs.status()` to identify the primary in the replica set.

The following example adds two members:

```
rs.add("mongodb1.example.net")
rs.add("mongodb2.example.net")
```

When complete, you have a fully functional replica set. The new replica set will elect a *primary*.

Step 11: Check the status of the replica set. Use the `rs.status()` operation:

```
rs.status()
```

Step 12: Create additional users to address operational requirements. You can use *built-in roles* (page 114) to create common types of database users, such as the `dbOwner` (page 116) role to create a database administrator, the `readWrite` (page 115) role to create a user who can update data, or the `read` (page 115) role to create user who can search data but no more. You also can define *custom roles* (page 27).

For example, the following creates a database administrator for the `products` database:

```
use products
db.createUser(
  {
    user: "productsDBAdmin",
    pwd: "password",
    roles:
    [
      {
        role: "dbOwner",
        db: "products"
      }
    ]
  }
)
```

For an overview of roles and privileges, see *Role-Based Access Control* (page 23). For more information on adding users, see *Manage User and Roles* (page 70).

x.509 Internal Authentication

For details on using x.509 for internal authentication, see *Use x.509 Certificate for Membership Authentication* (page 51).

To upgrade from keyfile internal authentication to x.509 internal authentication, see *Upgrade from Keyfile Authentication to x.509 Authentication* (page 53).

7.2 Authentication Mechanisms

The following tutorials provide information on configuring MongoDB to use authentication mechanisms other than the *default authentication mechanism* (page 10). For tutorials on using *default authentication mechanism* (page 10), see *Enable Access Control* (page 39).

***Use x.509 Certificates to Authenticate Clients* (page 48)** Use x.509 for client authentication.

***Use x.509 Certificate for Membership Authentication* (page 51)** Use x.509 for internal member authentication for replica sets and sharded clusters.

***Upgrade from Keyfile Authentication to x.509 Authentication* (page 53)** Upgrade from keyfile internal authentication to x.509 internal authentication.

***Configure MongoDB with Kerberos Authentication on Linux* (page 55)** For MongoDB Enterprise Linux, describes the process to enable Kerberos-based authentication for MongoDB deployments.

***Configure MongoDB with Kerberos Authentication on Windows* (page 59)** For MongoDB Enterprise for Windows, describes the process to enable Kerberos-based authentication for MongoDB deployments.

Troubleshoot Kerberos Authentication (page 61) Steps to troubleshoot Kerberos-based authentication for MongoDB deployments.

Authenticate Using SASL and LDAP with ActiveDirectory (page 63) Describes the process for authentication using SASL/LDAP with ActiveDirectory.

Authenticate Using SASL and LDAP with OpenLDAP (page 66) Describes the process for authentication using SASL/LDAP with OpenLDAP.

7.2.1 Use x.509 Certificates to Authenticate Clients

On this page

- [Prerequisites \(page 48\)](#)
- [Procedures \(page 49\)](#)

New in version 2.6.

MongoDB supports x.509 certificate authentication for use with a secure *TLS/SSL connection* (page 79). The x.509 client authentication allows *clients to authenticate to servers with certificates* (page 48) rather than with a username and password.

To use x.509 authentication for the internal authentication of replica set/sharded cluster members, see *Use x.509 Certificate for Membership Authentication* (page 51).

Prerequisites

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, in particular x.509 certificates, and Certificate Authority is beyond the scope of this document. This tutorial assumes prior knowledge of TLS/SSL as well as access to valid x.509 certificates.

Certificate Authority

For production use, your MongoDB deployment should use valid certificates generated and signed by a single certificate authority. You or your organization can generate and maintain an independent certificate authority, or use certificates generated by a third-party SSL vendor. Obtaining and managing certificates is beyond the scope of this documentation.

Client x.509 Certificate

The client certificate must have the following properties:

- A single Certificate Authority (CA) must issue the certificates for both the client and the server.
- Client certificates must contain the following fields:

```
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
```

- Each unique MongoDB user must have a unique certificate.

- A client x.509 certificate's subject, which contains the Distinguished Name (DN), must **differ** from that of a *Member x.509 Certificate* (page 51). Specifically, the subjects must differ with regards to at least one of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).

Warning: If a client x.509 certificate's subject has the same O, OU, and DC combination as the *Member x.509 Certificate* (page 51), the client will be identified as a cluster member and granted full permission on the system.

Procedures

Configure MongoDB Server

Use Command-line Options You can configure the MongoDB server from the command line, e.g.:

```
mongod --clusterAuthMode x509 --sslMode requireSSL --sslPEMKeyFile <path to SSL certificate and key file>
```

Warning: If the `--sslCAFile` option and its target file are not specified, x.509 client and member authentication will not function. `mongod`, and `mongos` in sharded systems, will not be able to verify the certificates of processes connecting to it against the trusted certificate authority (CA) that issued them, breaking the certificate chain.

As of version 2.6.4, `mongod` will not start with x.509 authentication enabled if the CA file is not specified.

Use Configuration File You may also specify these options in the configuration file.

Starting in MongoDB 2.6, you can specify the configuration for MongoDB in YAML format, e.g.:

```
security:
  clusterAuthMode: x509
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
    CAFile: <path to root CA PEM file>
```

For backwards compatibility, you can also specify the configuration using the [older configuration file format](#)¹, e.g.:

```
clusterAuthMode = x509
sslMode = requireSSL
sslPEMKeyFile = <path to TLS/SSL certificate and key PEM file>
sslCAFile = <path to the root CA PEM file>
```

Include any additional options, TLS/SSL or otherwise, that are required for your specific configuration.

Add x.509 Certificate subject as a User

To authenticate with a client certificate, you must first add the value of the `subject` from the client certificate as a MongoDB user. Each unique x.509 client certificate corresponds to a single MongoDB user; i.e. you cannot use a single client certificate to authenticate more than one MongoDB user.

Note: The RDNs in the `subject` string must be compatible with the [RFC2253](#)² standard.

¹<https://docs.mongodb.org/v2.4/reference/configuration-options>

²<https://www.ietf.org/rfc/rfc2253.txt>

1. You can retrieve the RFC2253 formatted subject from the client certificate with the following command:

```
openssl x509 -in <pathToClient PEM> -inform PEM -subject -nameopt RFC2253
```

The command returns the subject string as well as certificate:

```
subject= CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry
-----BEGIN CERTIFICATE-----
# ...
-----END CERTIFICATE-----
```

2. Add the RFC2253 compliant value of the subject as a user. Omit spaces as needed.

For example, in the mongo shell, to add the user with both the `readWrite` role in the `test` database and the `userAdminAnyDatabase` role which is defined only in the `admin` database:

```
db.getSiblingDB("$external").runCommand(
  {
    createUser: "CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry",
    roles: [
      { role: 'readWrite', db: 'test' },
      { role: 'userAdminAnyDatabase', db: 'admin' }
    ],
    writeConcern: { w: "majority" , wtimeout: 5000 }
  }
)
```

In the above example, to add the user with the `readWrite` role in the `test` database, the role specification document specified `'test'` in the `db` field. To add `userAdminAnyDatabase` role for the user, the above example specified `'admin'` in the `db` field.

Note: Some roles are defined only in the `admin` database, including: `clusterAdmin`, `readAnyDatabase`, `readWriteAnyDatabase`, `dbAdminAnyDatabase`, and `userAdminAnyDatabase`. To add a user with these roles, specify `'admin'` in the `db`.

See *Manage User and Roles* (page 70) for details on adding a user with roles.

Authenticate with a x.509 Certificate

To authenticate with a client certificate, you must first add a MongoDB user that corresponds to the client certificate. See *Add x.509 Certificate subject as a User* (page 49).

To authenticate, use the `db.auth()` method in the `$external` database, specifying `"MONGODB-X509"` for the `mechanism` field, and the *user that corresponds to the client certificate* (page 49) for the `user` field.

For example, if using the mongo shell,

1. Connect mongo shell to the mongod set up for SSL:

```
mongo --ssl --sslPEMKeyFile <path to CA signed client PEM file> --sslCAFile <path to root CA PEM file>
```

2. To perform the authentication, use the `db.auth()` method in the `$external` database. For the `mechanism` field, specify `"MONGODB-X509"`, and for the `user` field, specify the user, or the subject, that corresponds to the client certificate.

```
db.getSiblingDB("$external").auth(
  {
    mechanism: "MONGODB-X509",
    user: "CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry"
```

```
}
)
```

7.2.2 Use x.509 Certificate for Membership Authentication

On this page

- [Member x.509 Certificate](#) (page 51)
- [Configure Replica Set/Sharded Cluster](#) (page 52)
- [Additional Information](#) (page 53)

New in version 2.6.

MongoDB supports x.509 certificate authentication for use with a secure *TLS/SSL connection* (page 79). Sharded cluster members and replica set members can use x.509 certificates to verify their membership to the cluster or the replica set instead of using *keyfiles* (page 19). The membership authentication is an internal process.

For client authentication with x.509, see *Use x.509 Certificates to Authenticate Clients* (page 48).

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, in particular x.509 certificates, and Certificate Authority is beyond the scope of this document. This tutorial assumes prior knowledge of TLS/SSL as well as access to valid x.509 certificates.

Member x.509 Certificate

Certificate Requirements

The member certificate, used for internal authentication to verify membership to the sharded cluster or a replica set, must have the following properties:

- A single Certificate Authority (CA) must issue all the x.509 certificates for the members of a sharded cluster or a replica set.
- The Distinguished Name (DN), found in the member certificate's *subject*, must specify a non-empty value for *at least one* of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).
- The Organization attributes (O's), the Organizational Unit attributes (OU's), and the Domain Components (DC's) must match those from the certificates for the other cluster members. To match, the certificate must match all specifications of these attributes, or even the non-specification of these attributes. The order of the attributes does not matter.

In the following example, the two DN's contain matching specifications for O, OU as well as the non-specification of the DC attribute.

```
CN=host1,OU=Dept1,O=MongoDB,ST=NY,C=US
C=US, ST=CA, O=MongoDB, OU=Dept1, CN=host2
```

However, the following two DN's contain a mismatch for the OU attribute since one contains two OU specifications and the other, only one specification.

```
CN=host1,OU=Dept1,OU=Sales,O=MongoDB
CN=host2,OU=Dept1,O=MongoDB
```


- Either the Common Name (CN) or one of the Subject Alternative Name (SAN) entries must match the hostname of the server, used by the other members of the cluster.

For example, the certificates for a cluster could have the following subjects:

```
subject= CN=<myhostname1>, OU=Dept1, O=MongoDB, ST=NY, C=US
subject= CN=<myhostname2>, OU=Dept1, O=MongoDB, ST=NY, C=US
subject= CN=<myhostname3>, OU=Dept1, O=MongoDB, ST=NY, C=US
```

- *If* the certificate includes the Extended Key Usage (extendedKeyUsage) setting, the value must include clientAuth (“TLS Web Client Authentication”).

```
extendedKeyUsage = clientAuth
```

You can also use a certificate that does not include the Extended Key Usage (EKU).

Member Certificate and PEMKeyFile

To configure MongoDB for client certificate authentication, the `mongod` and `mongos` specify a `PEMKeyFile` to prove its identity to clients, either through `net.ssl.PEMKeyFile` setting in the configuration file or `--sslPEMKeyFile` command line option.

If no `clusterFile` certificate is specified for internal member authentication, MongoDB will attempt to use the `PEMKeyFile` certificate for member authentication. In order to use `PEMKeyFile` certificate for internal authentication as well as for client authentication, then the `PEMKeyFile` certificate must either:

- Omit `extendedKeyUsage` or
- Specify `extendedKeyUsage` values that include `clientAuth` in addition to `serverAuth`.

Configure Replica Set/Sharded Cluster

Use Command-line Options

To specify the x.509 certificate for internal cluster member authentication, append the additional TLS/SSL options `--clusterAuthMode` and `--sslClusterFile`, as in the following example for a member of a replica set:

```
mongod --replSet <name> --sslMode requireSSL --clusterAuthMode x509 --sslClusterFile <path to member>
```

Include any additional options, TLS/SSL or otherwise, that are required for your specific configuration. For instance, if the membership key is encrypted, set the `--sslClusterPassword` to the passphrase to decrypt the key or have MongoDB prompt for the passphrase. See *SSL Certificate Passphrase* (page 83) for details.

Warning: If the `--sslCAFile` option and its target file are not specified, x.509 client and member authentication will not function. `mongod`, and `mongos` in sharded systems, will not be able to verify the certificates of processes connecting to it against the trusted certificate authority (CA) that issued them, breaking the certificate chain.

As of version 2.6.4, `mongod` will not start with x.509 authentication enabled if the CA file is not specified.

Use Configuration File

You can specify the configuration for MongoDB in a YAML formatted configuration file, as in the following example:

```

security:
  clusterAuthMode: x509
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
    CAFile: <path to root CA PEM file>
    clusterFile: <path to x.509 membership certificate and key PEM file>

```

See `security.clusterAuthMode`, `net.ssl.mode`, `net.ssl.PEMKeyFile`, `net.ssl.CAFile`, and `net.ssl.clusterFile` for more information on the settings.

Additional Information

To upgrade from keyfile internal authentication to x.509 internal authentication, see [Upgrade from Keyfile Authentication to x.509 Authentication](#) (page 53).

7.2.3 Upgrade from Keyfile Authentication to x.509 Authentication

On this page

- [Clusters Currently Using TLS/SSL](#) (page 53)
- [Clusters Currently Not Using TLS/SSL](#) (page 54)

To upgrade clusters that are currently using *keyfile authentication* (page 19) to x.509 authentication, use the following rolling upgrade processes.

Clusters Currently Using TLS/SSL

For clusters using TLS/SSL and keyfile authentication, to upgrade to x.509 cluster authentication, use the following rolling upgrade process:

1. For each node of a cluster, start the node with the option `--clusterAuthMode` set to `sendKeyFile` and the option `--sslClusterFile` set to the appropriate path of the node's certificate. Include other *TLS/SSL options* (page 79) as well as any other options that are required for your specific configuration. For example:

```
mongod --replSet <name> --sslMode requireSSL --clusterAuthMode sendKeyFile --sslClusterFile <pat
```

With this setting, each node continues to use its keyfile to authenticate itself as a member. However, each node can now accept either a keyfile or an x.509 certificate from other members to authenticate those members. Upgrade all nodes of the cluster to this setting.

2. Then, for each node of a cluster, connect to the node and use the `setParameter` command to update the `clusterAuthMode` to `sendX509`.³ For example,

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, clusterAuthMode: "sendX509" } )
```

With this setting, each node uses its x.509 certificate, specified with the `--sslClusterFile` option in the previous step, to authenticate itself as a member. However, each node continues to accept either a keyfile or an x.509 certificate from other members to authenticate those members. Upgrade all nodes of the cluster to this setting.

³ As an alternative to using the `setParameter` command, you can also restart the nodes with the appropriate TLS/SSL and x.509 options and values.

- Optional but recommended. Finally, for each node of the cluster, connect to the node and use the `setParameter` command to update the `clusterAuthMode` to `x509` to only use the x.509 certificate for authentication. ¹ For example:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, clusterAuthMode: "x509" } )
```

- After the upgrade of all nodes, edit the `configuration` file with the appropriate x.509 settings to ensure that upon subsequent restarts, the cluster uses x.509 authentication.

See `--clusterAuthMode` for the various modes and their descriptions.

Clusters Currently Not Using TLS/SSL

For clusters using keyfile authentication but not TLS/SSL, to upgrade to x.509 authentication, use the following rolling upgrade process:

- For each node of a cluster, start the node with the option `--sslMode` set to `allowSSL`, the option `--clusterAuthMode` set to `sendKeyFile` and the option `--sslClusterFile` set to the appropriate path of the node's certificate. Include other *TLS/SSL options* (page 79) as well as any other options that are required for your specific configuration. For example:

```
mongod --replSet <name> --sslMode allowSSL --clusterAuthMode sendKeyFile --sslClusterFile <path>
```

The `--sslMode allowSSL` setting allows the node to accept both TLS/SSL and non-TLS/non-SSL incoming connections. Its outgoing connections do not use TLS/SSL.

The `--clusterAuthMode sendKeyFile` setting allows each node continues to use its keyfile to authenticate itself as a member. However, each node can now accept either a keyfile or an x.509 certificate from other members to authenticate those members.

Upgrade all nodes of the cluster to these settings.

- Then, for each node of a cluster, connect to the node and use the `setParameter` command to update the `sslMode` to `preferSSL` and the `clusterAuthMode` to `sendX509`. ¹ For example:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "preferSSL", clusterAuthMode: "
```

With the `sslMode` set to `preferSSL`, the node accepts both TLS/SSL and non-TLS/non-SSL incoming connections, and its outgoing connections use TLS/SSL.

With the `clusterAuthMode` set to `sendX509`, each node uses its x.509 certificate, specified with the `--sslClusterFile` option in the previous step, to authenticate itself as a member. However, each node continues to accept either a keyfile or an x.509 certificate from other members to authenticate those members.

Upgrade all nodes of the cluster to these settings.

- Optional but recommended. Finally, for each node of the cluster, connect to the node and use the `setParameter` command to update the `sslMode` to `requireSSL` and the `clusterAuthMode` to `x509`. ¹ For example:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "requireSSL", clusterAuthMode: "
```

With the `sslMode` set to `requireSSL`, the node only uses TLS/SSLs connections.

With the `clusterAuthMode` set to `x509`, the node only uses the x.509 certificate for authentication.

- After the upgrade of all nodes, edit the `configuration` file with the appropriate TLS/SSL and x.509 settings to ensure that upon subsequent restarts, the cluster uses x.509 authentication.

See `--clusterAuthMode` for the various modes and their descriptions.

7.2.4 Configure MongoDB with Kerberos Authentication on Linux

On this page

- [Overview](#) (page 55)
- [Prerequisites](#) (page 55)
- [Procedure](#) (page 55)
- [Additional Considerations](#) (page 57)
- [Additional Resources](#) (page 58)

New in version 2.4.

Overview

MongoDB Enterprise supports authentication using a *Kerberos service* (page 15). Kerberos is an industry standard authentication protocol for large client/server system.

Prerequisites

Setting up and configuring a Kerberos deployment is beyond the scope of this document. This tutorial assumes you have configured a *Kerberos service principal* (page 16) for each `mongod` and `mongos` instance in your MongoDB deployment, and you have a valid *keytab file* (page 16) for for each `mongod` and `mongos` instance.

To verify MongoDB Enterprise binaries:

```
mongod --version
```

In the output from this command, look for the string `modules: subscription` or `modules: enterprise` to confirm your system has MongoDB Enterprise.

Procedure

The following procedure outlines the steps to add a Kerberos user principal to MongoDB, configure a standalone `mongod` instance for Kerberos support, and connect using the `mongo` shell and authenticate the user principal.

Step 1: Start `mongod` without Kerberos.

For the initial addition of Kerberos users, start `mongod` without Kerberos support.

If a Kerberos user is already in MongoDB and has the *privileges required to create a user*, you can start `mongod` with Kerberos support.

Step 2: Connect to `mongod`.

Connect via the `mongo` shell to the `mongod` instance. If `mongod` has `--auth` enabled, ensure you connect with the *privileges required to create a user*.

Step 3: Add Kerberos Principal(s) to MongoDB.

Add a Kerberos principal, <username>@<KERBEROS REALM> or <username>/<instance>@<KERBEROS REALM>, to MongoDB in the `$external` database. Specify the Kerberos realm in all uppercase. The `$external` database allows `mongod` to consult an external source (e.g. Kerberos) to authenticate. To specify the user's privileges, assign *roles* (page 23) to the user.

The following example adds the Kerberos principal `application/reporting@EXAMPLE.NET` with read-only access to the `records` database:

```
use $external
db.createUser(
  {
    user: "application/reporting@EXAMPLE.NET",
    roles: [ { role: "read", db: "records" } ]
  }
)
```

Add additional principals as needed. For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB. For more information about creating and managing users, see <https://docs.mongodb.org/manual/reference/command/nav-user-management>.

Step 4: Start mongod with Kerberos support.

To start `mongod` with Kerberos support, set the environmental variable `KRB5_KTNAME` to the path of the keytab file and the `mongod` parameter `authenticationMechanisms` to `GSSAPI` in the following form:

```
env KRB5_KTNAME=<path to keytab file> \
mongod \
--setParameter authenticationMechanisms=GSSAPI
<additional mongod options>
```

For example, the following starts a standalone `mongod` instance with Kerberos support:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \
/opt/mongodb/bin/mongod --auth \
--setParameter authenticationMechanisms=GSSAPI \
--dbpath /opt/mongodb/data
```

The path to your `mongod` as well as your *keytab file* (page 16) may differ. Modify or include additional `mongod` options as required for your configuration. The *keytab file* (page 16) must be only accessible to the owner of the `mongod` process.

With the official `.deb` or `.rpm` packages, you can set the `KRB5_KTNAME` in an environment settings file. See [KRB5_KTNAME](#) (page 57) for details.

Step 5: Connect mongo shell to mongod and authenticate.

Connect the `mongo` shell client as the Kerberos principal `application/reporting@EXAMPLE.NET`. Before connecting, you must have used Kerberos's `kinit` program to get credentials for `application/reporting@EXAMPLE.NET`.

You can connect and authenticate from the command line.

```
mongo --authenticationMechanism=GSSAPI --authenticationDatabase='$external' \
--username application/reporting@EXAMPLE.NET
```

Or, alternatively, you can first connect `mongo` to the `mongod`, and then from the `mongo` shell, use the `db.auth()` method to authenticate in the `$external` database.

```
use $external
db.auth( { mechanism: "GSSAPI", user: "application/reporting@EXAMPLE.NET" } )
```

Additional Considerations

KRB5_KTNAME

If you installed MongoDB Enterprise using one of the official `.deb` or `.rpm` packages, and you use the included `init/upstart` scripts to control the `mongod` instance, you can set the `KRB5_KTNAME` variable in the default environment settings file instead of setting the variable each time.

For `.rpm` packages, the default environment settings file is `/etc/sysconfig/mongod`.

For `.deb` packages, the file is `/etc/default/mongod`.

Set the `KRB5_KTNAME` value in a line that resembles the following:

```
export KRB5_KTNAME="<path to keytab>"
```

Configure mongos for Kerberos

To start `mongos` with Kerberos support, set the environmental variable `KRB5_KTNAME` to the path of its *keytab file* (page 16) and the `mongos` parameter `authenticationMechanisms` to `GSSAPI` in the following form:

```
env KRB5_KTNAME=<path to keytab file> \
mongos \
--setParameter authenticationMechanisms=GSSAPI \
<additional mongos options>
```

For example, the following starts a `mongos` instance with Kerberos support:

```
env KRB5_KTNAME=/opt/mongodb/mongos.keytab \
mongos \
--setParameter authenticationMechanisms=GSSAPI \
--configdb shard0.example.net, shard1.example.net, shard2.example.net \
--keyFile /opt/mongodb/mongos.keyfile
```

The path to your `mongos` as well as your *keytab file* (page 16) may differ. The *keytab file* (page 16) must be only accessible to the owner of the `mongos` process.

Modify or include any additional `mongos` options as required for your configuration. For example, instead of using `--keyFile` for internal authentication of sharded cluster members, you can use *x.509 member authentication* (page 51) instead.

Use a Config File

To configure `mongod` or `mongos` for Kerberos support using a configuration file, specify the `authenticationMechanisms` setting in the configuration file:

If using the YAML configuration file format:

```
setParameter:
  authenticationMechanisms: GSSAPI
```

Or, if using the older `.ini` configuration file format:

```
setParameter=authenticationMechanisms=GSSAPI
```

Modify or include any additional `mongod` options as required for your configuration. For example, if `/opt/mongodb/mongod.conf` contains the following configuration settings for a standalone `mongod`:

```
security:
  authorization: enabled
setParameter:
  authenticationMechanisms: GSSAPI
storage:
  dbPath: /opt/mongodb/data
```

Or, if using the older configuration file format⁴:

```
auth = true
setParameter=authenticationMechanisms=GSSAPI
dbpath=/opt/mongodb/data
```

To start `mongod` with Kerberos support, use the following form:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \
/opt/mongodb/bin/mongod --config /opt/mongodb/mongod.conf
```

The path to your `mongod`, *keytab file* (page 16), and configuration file may differ. The *keytab file* (page 16) must be only accessible to the owner of the `mongod` process.

Troubleshoot Kerberos Setup for MongoDB

If you encounter problems when starting `mongod` or `mongos` with Kerberos authentication, see *Troubleshoot Kerberos Authentication* (page 61).

Incorporate Additional Authentication Mechanisms

Kerberos authentication (*GSSAPI* (page 15) (Kerberos)) can work alongside MongoDB's challenge/response authentication mechanisms (*SCRAM-SHA-1* (page 11) and *MONGODB-CR* (page 12)), MongoDB's authentication mechanism for LDAP (*PLAIN* (page 15) (LDAP SASL)), and MongoDB's authentication mechanism for x.509 (*MONGODB-X509* (page 12)). Specify the mechanisms as follows:

```
--setParameter authenticationMechanisms=GSSAPI, SCRAM-SHA-1
```

Only add the other mechanisms if in use. This parameter setting does not affect MongoDB's internal authentication of cluster members.

Additional Resources

- MongoDB LDAP and Kerberos Authentication with Dell (Quest) Authentication Services⁵
- MongoDB with Red Hat Enterprise Linux Identity Management and Kerberos⁶

⁴<https://docs.mongodb.org/v2.4/reference/configuration-options>

⁵<https://www.mongodb.com/blog/post/mongodb-ldap-and-kerberos-authentication-dell-quest-authentication-services?jmp=docs>

⁶<http://docs.mongodb.org/ecosystem/tutorial/manage-red-hat-enterprise-linux-identity-management?jmp=docs>

7.2.5 Configure MongoDB with Kerberos Authentication on Windows

On this page

- [Overview](#) (page 59)
- [Prerequisites](#) (page 59)
- [Procedures](#) (page 59)
- [Additional Considerations](#) (page 60)

New in version 2.6.

Overview

MongoDB Enterprise supports authentication using a *Kerberos service* (page 15). Kerberos is an industry standard authentication protocol for large client/server system. Kerberos allows MongoDB and applications to take advantage of existing authentication infrastructure and processes.

Prerequisites

Setting up and configuring a Kerberos deployment is beyond the scope of this document. This tutorial assumes have configured a *Kerberos service principal* (page 16) for each `mongod.exe` and `mongos.exe` instance.

Procedures

Step 1: Start `mongod.exe` without Kerberos.

For the initial addition of Kerberos users, start `mongod.exe` without Kerberos support.

If a Kerberos user is already in MongoDB and has the *privileges required to create a user*, you can start `mongod.exe` with Kerberos support.

Step 2: Connect to `mongod`.

Connect via the `mongo.exe` shell to the `mongod.exe` instance. If `mongod.exe` has `--auth` enabled, ensure you connect with the *privileges required to create a user*.

Step 3: Add Kerberos Principal(s) to MongoDB.

Add a Kerberos principal, `<username>@<KERBEROS_REALM>`, to MongoDB in the `$external` database. Specify the Kerberos realm in **ALL UPPERCASE**. The `$external` database allows `mongod.exe` to consult an external source (e.g. Kerberos) to authenticate. To specify the user's privileges, assign *roles* (page 23) to the user.

The following example adds the Kerberos principal `reportingapp@EXAMPLE.NET` with read-only access to the `records` database:

```
use $external
db.createUser(
  {
    user: "reportingapp@EXAMPLE.NET",
    roles: [ { role: "read", db: "records" } ]
  }
)
```



```
    }  
  )
```

Add additional principals as needed. For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB. For more information about creating and managing users, see <https://docs.mongodb.org/manual/reference/command/nav-user-management>.

Step 4: Start `mongod.exe` with Kerberos support.

You must start `mongod.exe` as the *service principal account* (page 61).

To start `mongod.exe` with Kerberos support, set the `mongod.exe` parameter `authenticationMechanisms` to GSSAPI:

```
mongod.exe --setParameter authenticationMechanisms=GSSAPI <additional mongod.exe options>
```

For example, the following starts a standalone `mongod.exe` instance with Kerberos support:

```
mongod.exe --auth --setParameter authenticationMechanisms=GSSAPI
```

Modify or include additional `mongod.exe` options as required for your configuration.

Step 5: Connect `mongo.exe` shell to `mongod.exe` and authenticate.

Connect the `mongo.exe` shell client as the Kerberos principal `application@EXAMPLE.NET`.

You can connect and authenticate from the command line.

```
mongo.exe --authenticationMechanism=GSSAPI --authenticationDatabase='$external' \  
--username reportingapp@EXAMPLE.NET
```

Or, alternatively, you can first connect `mongo.exe` to the `mongod.exe`, and then from the `mongo.exe` shell, use the `db.auth()` method to authenticate in the `$external` database.

```
use $external  
db.auth( { mechanism: "GSSAPI", user: "reportingapp@EXAMPLE.NET" } )
```

Additional Considerations

Configure `mongos.exe` for Kerberos

To start `mongos.exe` with Kerberos support, set the `mongos.exe` parameter `authenticationMechanisms` to GSSAPI. You must start `mongos.exe` as the *service principal account* (page 61):

```
mongos.exe --setParameter authenticationMechanisms=GSSAPI <additional mongos options>
```

For example, the following starts a `mongos` instance with Kerberos support:

```
mongos.exe --setParameter authenticationMechanisms=GSSAPI --configdb shard0.example.net, shard1.example.net
```

Modify or include any additional `mongos.exe` options as required for your configuration. For example, instead of using `--keyFile` for internal authentication of sharded cluster members, you can use *x.509 member authentication* (page 51) instead.

Assign Service Principal Name to MongoDB Windows Service

Use `setspn.exe` to assign the service principal name (SPN) to the account running the `mongod.exe` and the `mongos.exe` service:

```
setspn.exe -A <service>/<fully qualified domain name> <service account name>
```

For example, if `mongod.exe` runs as a service named `mongodb` on `testserver.mongodb.com` with the service account name `mongodtest`, assign the SPN as follows:

```
setspn.exe -A mongodb/testserver.mongodb.com mongodtest
```

Incorporate Additional Authentication Mechanisms

Kerberos authentication (*GSSAPI* (page 15) (Kerberos)) can work alongside MongoDB's challenge/response authentication mechanisms (*SCRAM-SHA-1* (page 11) and *MONGODB-CR* (page 12)), MongoDB's authentication mechanism for LDAP (*PLAIN* (page 15) (LDAP SASL)), and MongoDB's authentication mechanism for x.509 (*MONGODB-X509* (page 12)). Specify the mechanisms as follows:

```
--setParameter authenticationMechanisms=GSSAPI,SCRAM-SHA-1
```

Only add the other mechanisms if in use. This parameter setting does not affect MongoDB's internal authentication of cluster members.

7.2.6 Troubleshoot Kerberos Authentication

On this page

- [Kerberos Configuration Checklist](#) (page 61)
- [Debug with More Verbose Logs on Linux](#) (page 62)
- [Common Error Messages](#) (page 62)

New in version 2.4.

Kerberos Configuration Checklist

If you have difficulty starting `mongod` or `mongos` with *Kerberos* (page 15), ensure that:

- The `mongod` and the `mongos` binaries are from MongoDB Enterprise.

To verify MongoDB Enterprise binaries:

```
mongod --version
```

In the output from this command, look for the string `modules: subscription` or `modules: enterprise` to confirm your system has MongoDB Enterprise.

- You are not using the [HTTP Console](#)⁷. MongoDB Enterprise does not support Kerberos authentication over the HTTP Console interface.
- On Linux, either the service principal name (SPN) in the *keytab file* (page 16) matches the SPN for the `mongod` or `mongos` instance, or the `mongod` or the `mongos` instance use the `--setParameter saslHostName=<host name>` to match the name in the keytab file.

⁷<https://docs.mongodb.org/ecosystem/tools/http-interface/#http-console>

- The canonical system hostname of the system that runs the `mongod` or `mongos` instance is a resolvable, fully qualified domain for this host. You can test the system hostname resolution with the `hostname -f` command at the system prompt.
- Each host that runs a `mongod` or `mongos` instance has both the `A` and `PTR` DNS records to provide forward and reverse lookup. The records allow the host to resolve the components of the Kerberos infrastructure.
- Both the Kerberos Key Distribution Center (KDC) and the system running `mongod` instance or `mongos` must be able to resolve each other using DNS. By default, Kerberos attempts to resolve hosts using the content of the `/etc/kerb5.conf` before using DNS to resolve hosts.
- The time synchronization of the systems running `mongod` or the `mongos` instances and the Kerberos infrastructure are within the maximum time skew (default is 5 minutes) of each other. Time differences greater than the maximum time skew will prevent successful authentication.

Debug with More Verbose Logs on Linux

If you still encounter problems with Kerberos on Linux, you can start both `mongod` and `mongo` (or another client) with the environment variable `KRB5_TRACE` set to different files to produce more verbose logging of the Kerberos process to help further troubleshooting. For example, the following starts a standalone `mongod` with `KRB5_TRACE` set:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \  
KRB5_TRACE=/opt/mongodb/log/mongodb-kerberos.log \  
/opt/mongodb/bin/mongod --dbpath /opt/mongodb/data \  
--fork --logpath /opt/mongodb/log/mongod.log \  
--auth --setParameter authenticationMechanisms=GSSAPI
```

Common Error Messages

In some situations, MongoDB will return error messages from the GSSAPI interface if there is a problem with the Kerberos service. Some common error messages are:

GSSAPI error in client while negotiating security context. This error occurs on the client and reflects insufficient credentials or a malicious attempt to authenticate.

If you receive this error, ensure that you are using the correct credentials and the correct fully qualified domain name when connecting to the host.

GSSAPI error acquiring credentials. This error occurs during the start of the `mongod` or `mongos` and reflects improper configuration of the system hostname or a missing or incorrectly configured keytab file.

If you encounter this problem, consider the items in the *Kerberos Configuration Checklist* (page 61), in particular, whether the SPN in the *keytab file* (page 16) matches the SPN for the `mongod` or `mongos` instance.

To determine whether the SPNs match:

1. Examine the keytab file, with the following command:

```
klist -k <keytab>
```

Replace `<keytab>` with the path to your keytab file.

2. Check the configured hostname for your system, with the following command:

```
hostname -f
```

Ensure that this name matches the name in the keytab file, or start `mongod` or `mongos` with the `--setParameter saslHostName=<hostname>`.

See also:

- *Kerberos Authentication* (page 15)
- *Configure MongoDB with Kerberos Authentication on Linux* (page 55)
- *Configure MongoDB with Kerberos Authentication on Windows* (page 59)

7.2.7 Authenticate Using SASL and LDAP with ActiveDirectory

On this page

- *Considerations* (page 63)
- *Configure saslauthd* (page 63)
- *Configure MongoDB* (page 64)

MongoDB Enterprise provides support for proxy authentication of users. This allows administrators to configure a MongoDB cluster to authenticate users by proxying authentication requests to a specified Lightweight Directory Access Protocol (LDAP) service.

Considerations

MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards. See <https://docs.mongodb.org/manual/release-notes/2.6-upgrade> for upgrade instructions.

Use secure encrypted or trusted connections between clients and the server, as well as between `saslauthd` and the LDAP server. The LDAP server uses the SASL PLAIN mechanism, sending and receiving data in **plain text**. You should use only a trusted channel such as a VPN, a connection encrypted with TLS/SSL, or a trusted wired network.

Configure saslauthd

LDAP support for user authentication requires proper configuration of the `saslauthd` daemon process as well as the MongoDB server.

Step 1: Specify the mechanism.

On systems that configure `saslauthd` with the `/etc/sysconfig/saslauthd` file, such as Red Hat Enterprise Linux, Fedora, CentOS, and Amazon Linux AMI, set the mechanism `MECH` to `ldap`:

```
MECH=ldap
```

On systems that configure `saslauthd` with the `/etc/default/saslauthd` file, such as Ubuntu, set the `MECHANISMS` option to `ldap`:

```
MECHANISMS="ldap"
```

Step 2: Adjust caching behavior.

On certain Linux distributions, `saslauthd` starts with the caching of authentication credentials *enabled*. Until restarted or until the cache expires, `saslauthd` will not contact the LDAP server to re-authenticate users in its authentication cache. This allows `saslauthd` to successfully authenticate users in its cache, even in the LDAP server is down or if the cached users' credentials are revoked.

To set the expiration time (in seconds) for the authentication cache, see the `-t option`⁸ of `saslauthd`.

Step 3: Configure LDAP Options with ActiveDirectory.

If the `saslauthd.conf` file does not exist, create it. The `saslauthd.conf` file usually resides in the `/etc` folder. If specifying a different file path, see the `-O option`⁹ of `saslauthd`.

To use with ActiveDirectory, start `saslauthd` with the following configuration options set in the `saslauthd.conf` file:

```
ldap_servers: <ldap uri>
ldap_use_sasl: yes
ldap_mech: DIGEST-MD5
ldap_auth_method: fastbind
```

For the `<ldap uri>`, specify the uri of the ldap server. For example, `ldap_servers: ldaps://ad.example.net`.

For more information on `saslauthd` configuration, see <http://www.openldap.org/doc/admin24/guide.html#Configuringsaslauthd>.

Step 4: Test the `saslauthd` configuration.

Use `testsaslauthd` utility to test the `saslauthd` configuration. For example:

```
testsaslauthd -u testuser -p testpassword -f /var/run/saslauthd/mux
```

Note: `/var/run/saslauthd` directory must have permissions set to 755 for MongoDB to successfully authenticate.

Configure MongoDB

Step 1: Add user to MongoDB for authentication.

Add the user to the `$external` database in MongoDB. To specify the user's privileges, assign *roles* (page 23) to the user.

For example, the following adds a user with read-only access to the `records` database.

```
db.getSiblingDB("$external").createUser(
  {
    user : <username>,
    roles: [ { role: "read", db: "records" } ]
  }
)
```

⁸http://www.linuxcommand.org/man_pages/saslauthd8.html

⁹http://www.linuxcommand.org/man_pages/saslauthd8.html

Add additional principals as needed. For more information about creating and managing users, see <https://docs.mongodb.org/manual/reference/command/nav-user-management>.

Step 2: Configure MongoDB server.

To configure the MongoDB server to use the `saslauthd` instance for proxy authentication, start the `mongod` with the following options:

- `--auth`,
- `authenticationMechanisms` parameter set to `PLAIN`, and
- `saslauthdPath` parameter set to the path to the Unix-domain Socket of the `saslauthd` instance.

Configure the MongoDB server using either the command line option `--setParameter` or the configuration file. Specify additional configurations as appropriate for your configuration.

If you use the `authorization` option to enforce authentication, you will need privileges to create a user.

Use specific `saslauthd` socket path. For socket path of `/<some>/<path>/saslauthd`, set the `saslauthdPath` to `/<some>/<path>/saslauthd/mux`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath=/<some>/<path>/saslauthd/mux --setParameter authenticationMechanisms=PLAIN
```

Or if using a YAML format configuration file, specify the following settings in the file:

```
security:
  authorization: enabled

setParameter:
  saslauthdPath: /<some>/<path>/saslauthd/mux
  authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format¹⁰:

```
auth=true
setParameter=saslauthdPath=/<some>/<path>/saslauthd/mux
setParameter=authenticationMechanisms=PLAIN
```

Use default Unix-domain socket path. To use the default Unix-domain socket path, set the `saslauthdPath` to the empty string `" "`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath="" --setParameter authenticationMechanisms=PLAIN
```

Or if using a YAML format configuration file, specify the following settings in the file:

```
security:
  authorization: enabled

setParameter:
  saslauthdPath: ""
  authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format¹¹:

¹⁰<https://docs.mongodb.org/v2.4/reference/configuration-options>

¹¹<https://docs.mongodb.org/v2.4/reference/configuration-options>

```
auth=true
setParameter=saslauthdPath=""
setParameter=authenticationMechanisms=PLAIN
```

Step 3: Authenticate the user in the mongo shell.

To perform the authentication in the mongo shell, use the `db.auth()` method in the `$external` database.

Specify the value "PLAIN" in the `mechanism` field, the user and password in the `user` and `pwd` fields respectively, and the value `false` in the `digestPassword` field. You **must** specify `false` for `digestPassword` since the server must receive an undigested password to forward on to `saslauthd`, as in the following example:

```
db.getSiblingDB("$external").auth(
  {
    mechanism: "PLAIN",
    user: <username>,
    pwd: <cleartext password>,
    digestPassword: false
  }
)
```

The server forwards the password in plain text. In general, use only on a trusted channel (VPN, TLS/SSL, trusted wired network). See Considerations.

7.2.8 Authenticate Using SASL and LDAP with OpenLDAP

On this page

- [Considerations](#) (page 66)
- [Configure saslauthd](#) (page 67)
- [Configure MongoDB](#) (page 68)

MongoDB Enterprise provides support for proxy authentication of users. This allows administrators to configure a MongoDB cluster to authenticate users by proxying authentication requests to a specified Lightweight Directory Access Protocol (LDAP) service.

Considerations

MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards. See <https://docs.mongodb.org/manual/release-notes/2.6-upgrade> for upgrade instructions.

Use secure encrypted or trusted connections between clients and the server, as well as between `saslauthd` and the LDAP server. The LDAP server uses the SASL PLAIN mechanism, sending and receiving data in **plain text**. You should use only a trusted channel such as a VPN, a connection encrypted with TLS/SSL, or a trusted wired network.

Configure `saslauthd`

LDAP support for user authentication requires proper configuration of the `saslauthd` daemon process as well as the MongoDB server.

Step 1: Specify the mechanism.

On systems that configure `saslauthd` with the `/etc/sysconfig/saslauthd` file, such as Red Hat Enterprise Linux, Fedora, CentOS, and Amazon Linux AMI, set the mechanism `MECH` to `ldap`:

```
MECH=ldap
```

On systems that configure `saslauthd` with the `/etc/default/saslauthd` file, such as Ubuntu, set the `MECHANISMS` option to `ldap`:

```
MECHANISMS="ldap"
```

Step 2: Adjust caching behavior.

On certain Linux distributions, `saslauthd` starts with the caching of authentication credentials *enabled*. Until restarted or until the cache expires, `saslauthd` will not contact the LDAP server to re-authenticate users in its authentication cache. This allows `saslauthd` to successfully authenticate users in its cache, even in the LDAP server is down or if the cached users' credentials are revoked.

To set the expiration time (in seconds) for the authentication cache, see the `-t` option¹² of `saslauthd`.

Step 3: Configure LDAP Options with OpenLDAP.

If the `saslauthd.conf` file does not exist, create it. The `saslauthd.conf` file usually resides in the `/etc` folder. If specifying a different file path, see the `-O` option¹³ of `saslauthd`.

To connect to an OpenLDAP server, update the `saslauthd.conf` file with the following configuration options:

```
ldap_servers: <ldap uri>
ldap_search_base: <search base>
ldap_filter: <filter>
```

The `ldap_servers` specifies the `uri` of the LDAP server used for authentication. In general, for OpenLDAP installed on the local machine, you can specify the value `ldap://localhost:389` or if using LDAP over TLS/SSL, you can specify the value `ldaps://localhost:636`.

The `ldap_search_base` specifies distinguished name to which the search is relative. The search includes the base or objects below.

The `ldap_filter` specifies the search filter.

The values for these configuration options should correspond to the values specific for your test. For example, to filter on email, specify `ldap_filter: (mail=%n)` instead.

OpenLDAP Example A sample `saslauthd.conf` file for OpenLDAP includes the following content:

¹²http://www.linuxcommand.org/man_pages/saslauthd8.html

¹³http://www.linuxcommand.org/man_pages/saslauthd8.html


```
ldap_servers: ldaps://ad.example.net
ldap_search_base: ou=Users,dc=example,dc=com
ldap_filter: (uid=%u)
```

To use this sample OpenLDAP configuration, create users with a `uid` attribute (login name) and place under the `Users` organizational unit (`ou`) under the domain components (`dc`) `example` and `com`.

For more information on `saslauthd` configuration, see <http://www.openldap.org/doc/admin24/guide.html#Configuringsaslauthd>.

Step 4: Test the `saslauthd` configuration.

Use `testsaslauthd` utility to test the `saslauthd` configuration. For example:

```
testsaslauthd -u testuser -p testpassword -f /var/run/saslauthd/mux
```

Note: `/var/run/saslauthd` directory must have permissions set to `755` for MongoDB to successfully authenticate.

Configure MongoDB

Step 1: Add user to MongoDB for authentication.

Add the user to the `$external` database in MongoDB. To specify the user's privileges, assign *roles* (page 23) to the user.

For example, the following adds a user with read-only access to the `records` database.

```
db.getSiblingDB("$external").createUser(
  {
    user : <username>,
    roles: [ { role: "read", db: "records" } ]
  }
)
```

Add additional principals as needed. For more information about creating and managing users, see <https://docs.mongodb.org/manual/reference/command/nav-user-management>.

Step 2: Configure MongoDB server.

To configure the MongoDB server to use the `saslauthd` instance for proxy authentication, start the `mongod` with the following options:

- `--auth`,
- `authenticationMechanisms` parameter set to `PLAIN`, and
- `saslauthdPath` parameter set to the path to the Unix-domain Socket of the `saslauthd` instance.

Configure the MongoDB server using either the command line option `--setParameter` or the configuration file. Specify additional configurations as appropriate for your configuration.

If you use the `authorization` option to enforce authentication, you will need privileges to create a user.

Use specific saslauthd socket path. For socket path of `/<some>/<path>/saslauthd`, set the `saslauthdPath` to `/<some>/<path>/saslauthd/mux`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath=/<some>/<path>/saslauthd/mux --setParameter authenticationMechanisms=PLAIN
```

Or if using a YAML format configuration file, specify the following settings in the file:

```
security:
  authorization: enabled

setParameter:
  saslauthdPath: /<some>/<path>/saslauthd/mux
  authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format¹⁴:

```
auth=true
setParameter=saslauthdPath=/<some>/<path>/saslauthd/mux
setParameter=authenticationMechanisms=PLAIN
```

Use default Unix-domain socket path. To use the default Unix-domain socket path, set the `saslauthdPath` to the empty string `" "`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath="" --setParameter authenticationMechanisms=PLAIN
```

Or if using a YAML format configuration file, specify the following settings in the file:

```
security:
  authorization: enabled

setParameter:
  saslauthdPath: ""
  authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format¹⁵:

```
auth=true
setParameter=saslauthdPath=""
setParameter=authenticationMechanisms=PLAIN
```

Step 3: Authenticate the user in the mongo shell.

To perform the authentication in the mongo shell, use the `db.auth()` method in the `$external` database.

Specify the value `"PLAIN"` in the `mechanism` field, the user and password in the `user` and `pwd` fields respectively, and the value `false` in the `digestPassword` field. You **must** specify `false` for `digestPassword` since the server must receive an undigested password to forward on to `saslauthd`, as in the following example:

```
db.getSiblingDB("$external").auth(
  {
    mechanism: "PLAIN",
    user: <username>,
    pwd: <cleartext password>,
    digestPassword: false
  }
)
```

¹⁴<https://docs.mongodb.org/v2.4/reference/configuration-options>

¹⁵<https://docs.mongodb.org/v2.4/reference/configuration-options>

The server forwards the password in plain text. In general, use only on a trusted channel (VPN, TLS/SSL, trusted wired network). See Considerations.

7.3 Configure Users and Roles

The following tutorials provide instructions on creating and managing users and roles.

Manage User and Roles (page 70) Manage users by creating new users, creating new roles, and modifying existing users.

Change Your Password and Custom Data (page 77) Create role with sufficient privileges to allow users to change their own passwords and modify the optional *custom data* associated with their user credential.

7.3.1 Manage User and Roles

On this page

- [Overview \(page 70\)](#)
- [Prerequisites \(page 70\)](#)
- [Add a User \(page 71\)](#)
- [Create a User-Defined Role \(page 72\)](#)
- [Modify Access for Existing User \(page 73\)](#)
- [Modify Password for Existing User \(page 75\)](#)
- [View a User's Role \(page 75\)](#)
- [View Role's Privileges \(page 76\)](#)

Overview

Changed in version 2.6: MongoDB 2.6 introduces a new *authorization model* (page 23).

MongoDB employs Role-Based Access Control (RBAC) to determine access for users. A user is granted one or more *roles* (page 23) that determine the user's access or privileges to MongoDB *resources* (page 127) and the *actions* (page 129) that user can perform. A user should have only the minimal set of privileges required to ensure a system of *least privilege*.

Each application and user of a MongoDB system should map to a distinct application or user. This *access isolation* facilitates access revocation and ongoing user maintenance.

This tutorial provides examples for user and role management under the MongoDB's authorization model.

Prerequisites

Important: If you have *enabled access control* (page 40) for your deployment, you must authenticate as a user with the required privileges specified in each section. A user administrator with the `userAdminAnyDatabase` (page 121) role, or `userAdmin` (page 117) role in the specific databases, provides the required privileges to perform the operations listed in this tutorial. See *Enable Client Access Control* (page 40) for details on adding user administrator as the first user.

Add a User

To create a user, specify the user name, password, and *roles* (page 23). For users that authenticate using external mechanisms ¹⁶, you do not need to provide the password when creating users.

When assigning roles, select the roles that have the exact required *privileges* (page 23). If the correct roles does not exist, you can *create new roles* (page 72).

Prerequisites

- To create a new user in a database, you must have `createUser` (page 130) *action* (page 129) on that *database resource* (page 128).
- To grant roles to a user, you must have the `grantRole` (page 130) *action* (page 129) on the role's database.

Built-in roles `userAdmin` (page 117) and `userAdminAnyDatabase` (page 121) provide `createUser` (page 130) and `grantRole` (page 130) actions on their respective *resources* (page 127).

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

Step 2: Create the new user. Create the user in the database to which the user will belong. Pass a well formed user document to the `db.createUser()` method.

The following operation creates a user in the `reporting` database with the specified name, password, and roles.

```
use reporting
db.createUser(
  {
    user: "reportsUser",
    pwd: "12345678",
    roles: [
      { role: "read", db: "reporting" },
      { role: "read", db: "products" },
      { role: "read", db: "sales" },
      { role: "readWrite", db: "accounts" }
    ]
  }
)
```

To authenticate the `reportsUser`, you must authenticate the user in the `reporting` database; i.e. specify `--authenticationDatabase reporting`.

You can create a user without assigning roles, choosing instead to assign the roles later. To do so, create the user with an empty *roles* (page 126) array.

¹⁶ See *x.509* (page 12), *Kerberos Authentication* (page 15), and *LDAP Proxy Authority Authentication* (page 15)

Create a User-Defined Role

Roles grant users access to MongoDB resources. MongoDB provides a number of *built-in roles* (page 114) that administrators can use to control access to a MongoDB system. However, if these roles cannot describe the desired set of privileges, you can create new roles in a particular database.

Except for roles created in the `admin` database, a role can only include privileges that apply to its database and can only inherit from other roles in its database.

A role created in the `admin` database can include privileges that apply to the `admin` database, other databases or to the *cluster* (page 128) resource, and can inherit from roles in other databases as well as the `admin` database.

To create a new role, use the `db.createRole()` method, specifying the privileges in the `privileges` array and the inherited roles in the `roles` array.

MongoDB uses the combination of the database name and the role name to uniquely define a role. Each role is scoped to the database in which you create the role, but MongoDB stores all role information in the `admin.system.roles` collection in the `admin` database.

Prerequisites

To create a role in a database, you must have:

- the `createRole` (page 130) *action* (page 129) on that *database resource* (page 128).
- the `grantRole` (page 130) *action* (page 129) on that database to specify privileges for the new role as well as to specify roles to inherit from.

Built-in roles `userAdmin` (page 117) and `userAdminAnyDatabase` (page 121) provide `createRole` (page 130) and `grantRole` (page 130) actions on their respective *resources* (page 127).

Create a Role to Manage Current Operations

The following example creates a role named `manageOpRole` which provides only the privileges to run both `db.currentOp()` and `db.killOp()`.¹⁷

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 72) section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

The `myUserAdmin` has privileges to create roles in the `admin` as well as other databases.

Step 2: Create a new role to manage current operations. `manageOpRole` has privileges that act on multiple databases as well as the *cluster resource* (page 128). As such, you must create the role in the `admin` database.

```
use admin
db.createRole(
  {
    role: "manageOpRole",
    privileges: [
      { resource: { cluster: true }, actions: [ "killop", "inprog" ] },
```

¹⁷ The built-in role `clusterMonitor` (page 118) also provides the privilege to run `db.currentOp()` along with other privileges, and the built-in role `hostManager` (page 119) provides the privilege to run `db.killOp()` along with other privileges.

```

    { resource: { db: "", collection: "" }, actions: [ "killCursors" ] }
  ],
  roles: []
}
)

```

The new role grants permissions to kill any operations.

Warning: Terminate running operations with extreme caution. Only use `db.killOp()` to terminate operations initiated by clients and *do not* terminate internal database operations.

Create a Role to Run `mongostat`

The following example creates a role named `mongostatRole` that provides only the privileges to run `mongostat`.¹⁸

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 72) section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

The `myUserAdmin` has privileges to create roles in the `admin` as well as other databases.

Step 2: Create a new role to manage current operations. `mongostatRole` has privileges that act on the *cluster resource* (page 128). As such, you must create the role in the `admin` database.

```

use admin
db.createRole(
  {
    role: "mongostatRole",
    privileges: [
      { resource: { cluster: true }, actions: [ "serverStatus" ] }
    ],
    roles: []
  }
)

```

Modify Access for Existing User

Prerequisites

- You must have the `grantRole` (page 130) *action* (page 129) on a database to grant a role on that database.
- You must have the `revokeRole` (page 130) *action* (page 129) on a database to revoke a role on that database.
- To view a role's information, you must be either explicitly granted the role or must have the `viewRole` (page 131) *action* (page 129) on the role's database.

¹⁸ The built-in role `clusterMonitor` (page 118) also provides the privilege to run `mongostat` along with other privileges.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

Step 2: Identify the user's roles and privileges. To display the roles and privileges of the user to be modified, use the `db.getUser()` and `db.getRole()` methods.

For example, to view roles for `reportsUser` created in *Add a User* (page 71), issue:

```
use reporting
db.getUser("reportsUser")
```

To display the privileges granted to the user by the `readWrite` role on the "accounts" database, issue:

```
use accounts
db.getRole("readWrite", { showPrivileges: true })
```

Step 3: Identify the privileges to grant or revoke. If the user requires additional privileges, grant to the user the role, or roles, with the required set of privileges. If such a role does not exist, *create a new role* (page 72) with the appropriate set of privileges.

To revoke a subset of privileges provided by an existing role: revoke the original role and grant a role that contains only the required privileges. You may need to *create a new role* (page 72) if a role does not exist.

Step 4: Modify the user's access.

Revoke a Role Revoke a role with the `db.revokeRolesFromUser()` method. The following example operation removes the `readWrite` (page 115) role on the `accounts` database from the `reportsUser`:

```
use reporting
db.revokeRolesFromUser(
  "reportsUser",
  [
    { role: "readWrite", db: "accounts" }
  ]
)
```

Grant a Role Grant a role using the `db.grantRolesToUser()` method. For example, the following operation grants the `reportsUser` user the `read` (page 115) role on the `accounts` database:

```
use reporting
db.grantRolesToUser(
  "reportsUser",
  [
    { role: "read", db: "accounts" }
  ]
)
```

For sharded clusters, the changes to the user are instant on the `mongos` on which the command runs. However, for other `mongos` instances in the cluster, the user cache may wait up to 10 minutes to refresh. See `userCacheInvalidationIntervalSecs`.

Modify Password for Existing User

Prerequisites

To modify the password of another user on a database, you must have the `changeAnyPassword` *action* (page 129) on that database.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to the `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 75) section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

Step 2: Change the password. Pass the user's username and the new password to the `db.changeUserPassword()` method.

The following operation changes the `reporting` user's password to `SOh3TbYhxuLiW8ypJPxmt1oOfL`:

```
db.changeUserPassword("reporting", "SOh3TbYhxuLiW8ypJPxmt1oOfL")
```

See also:

Change Your Password and Custom Data (page 77)

View a User's Role

Prerequisites

To view another user's information, you must have the `viewUser` (page 131) *action* (page 129) on the other user's database.

Users can view their own information.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```


Step 2: Identify the user's roles. Use the `usersInfo` command or `db.getUser()` method to display user information.

For example, to view roles for `reportsUser` created in *Add a User* (page 71), issue:

```
use reporting
db.getUser("reportsUser")
```

In the returned document, the `roles` (page 126) field displays all roles for `reportsUser`:

```
...
"roles" : [
  { "role" : "readWrite", "db" : "accounts" },
  { "role" : "read", "db" : "reporting" },
  { "role" : "read", "db" : "products" },
  { "role" : "read", "db" : "sales" }
]
```

View Role's Privileges

Prerequisites

To view a role's information, you must be either explicitly granted the role or must have the `viewRole` (page 131) *action* (page 129) on the role's database.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `myUserAdmin` created in *Enable Client Access Control* (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

Step 2: Identify the privileges granted by a role. For a given role, use the `db.getRole()` method, or the `rolesInfo` command, with the `showPrivileges` option:

For example, to view the privileges granted by `read` role on the `products` database, use the following operation, issue:

```
use products
db.getRole("read", { showPrivileges: true })
```

In the returned document, the `privileges` and `inheritedPrivileges` arrays. The `privileges` lists the privileges directly specified by the role and excludes those privileges inherited from other roles. The `inheritedPrivileges` lists all privileges granted by this role, both directly specified and inherited. If the role does not inherit from other roles, the two fields are the same.

```
...
"privileges" : [
  {
    "resource": { "db" : "products", "collection" : "" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  },
  {
    "resource" : { "db" : "products", "collection" : "system.js" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  }
]
```

```

    }
  ],
  "inheritedPrivileges" : [
    {
      "resource": { "db" : "products", "collection" : "" },
      "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
    },
    {
      "resource" : { "db" : "products", "collection" : "system.js" },
      "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
    }
  ]
}

```

7.3.2 Change Your Password and Custom Data

Changed in version 2.6.

On this page

- [Overview](#) (page 77)
- [Considerations](#) (page 77)
- [Prerequisites](#) (page 77)
- [Procedure](#) (page 78)

Overview

Users with appropriate privileges can change their own passwords and custom data. [Custom data](#) (page 126) stores optional user information.

Considerations

To generate a strong password for use in this procedure, you can use the `openssl` utility's `rand` command. For example, issue `openssl rand` with the following options to create a base64-encoded string of 48 pseudo-random bytes:

```
openssl rand -base64 48
```

Prerequisites

To modify your own password and custom data, you must have privileges that grant `changeOwnPassword` (page 130) and `changeOwnCustomData` (page 130) *actions* (page 129) respectively on the user's database.

Step 1: Connect as a user with privileges to manage users and roles.

Connect to the `mongod` or `mongos` with privileges to manage users and roles, such as a user with `userAdminAnyDatabase` (page 121) role. The following procedure uses the `myUserAdmin` created in [Enable Client Access Control](#) (page 40).

```
mongo --port 27017 -u myUserAdmin -p abc123 --authenticationDatabase admin
```

Step 2: Create a role with appropriate privileges.

In the admin database, create a new role with `changeOwnPassword` (page 130) and `changeOwnCustomData` (page 130).

```
use admin
db.createRole(
  { role: "changeOwnPasswordCustomDataRole",
    privileges: [
      {
        resource: { db: "", collection: "" },
        actions: [ "changeOwnPassword", "changeOwnCustomData" ]
      }
    ],
    roles: []
  }
)
```

Step 3: Add a user with this role.

In the test database, create a new user with the created "changeOwnPasswordCustomDataRole" role. For example, the following operation creates a user with both the built-in role `readWrite` (page 115) and the user-created "changeOwnPasswordCustomDataRole".

```
use test
db.createUser(
  {
    user:"user123",
    pwd:"12345678",
    roles:[ "readWrite", { role:"changeOwnPasswordCustomDataRole", db:"admin" } ]
  }
)
```

To grant an existing user the new role, use `db.grantRolesToUser()`.

Procedure

Step 1: Connect with the appropriate privileges.

Connect to the `mongod` or `mongos` as a user with appropriate privileges.

For example, the following operation connects to MongoDB as `user123` created in the *Prerequisites* (page 77) section.

```
mongo --port 27017 -u user123 -p 12345678 --authenticationDatabase test
```

To check that you have the privileges specified in the *Prerequisites* (page 77) section as well as to see user information, use the `usersInfo` command with the `showPrivileges` option.

Step 2: Change your password and custom data.

Use the `db.updateUser()` method to update the password and custom data.

For example, the following operation changes the user's password to `KN1ZmiaNUp0B` and custom data to `{ title: "Senior Manager" }`:

```
use test
db.updateUser(
  "user123",
  {
    pwd: "KN1ZmiaNUp0B",
    customData: { title: "Senior Manager" }
  }
)
```

7.4 Network

The following tutorials provide information on handling network security for MongoDB.

Configure mongod and mongos for TLS/SSL (page 79) Configure MongoDB to support TLS/SSL.

TLS/SSL Configuration for Clients (page 84) Configure clients to connect to MongoDB instances that use TLS/SSL.

Upgrade a Cluster to Use TLS/SSL (page 87) Rolling upgrade process to use TLS/SSL.

Configure MongoDB for FIPS (page 88) Configure for Federal Information Processing Standard (FIPS).

Configure Linux iptables Firewall for MongoDB (page 90) Basic firewall configuration patterns and examples for `iptables` on Linux systems.

Configure Windows netsh Firewall for MongoDB (page 94) Basic firewall configuration patterns and examples for `netsh` on Windows systems.

7.4.1 Configure mongod and mongos for TLS/SSL

On this page

- [Overview \(page 79\)](#)
- [Prerequisites \(page 80\)](#)
- [Procedures \(page 81\)](#)

Overview

This document helps you to configure MongoDB to support TLS/SSL. MongoDB clients can use TLS/SSL to encrypt connections to `mongod` and `mongos` instances. MongoDB TLS/SSL implementation uses OpenSSL libraries.

Note: Although TLS is the successor to SSL, this page uses the more familiar term SSL to refer to TLS/SSL.

These instructions assume that you have already installed a build of MongoDB that includes SSL support and that your client driver supports SSL. For instructions on upgrading a cluster currently not using SSL to using SSL, see [Upgrade a Cluster to Use TLS/SSL \(page 87\)](#).

Changed in version 2.6: MongoDB's SSL encryption only allows use of strong SSL ciphers with a minimum of 128-bit key length for all connections.

Prerequisites

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

MongoDB Support

New in version 3.0: Most MongoDB distributions now include support for SSL.

Certain distributions of MongoDB¹⁹ do **not** contain support for SSL. To use SSL, be sure to choose a package that supports SSL. All MongoDB Enterprise²⁰ supported platforms include SSL support.

Client Support

See *TLS/SSL Configuration for Clients* (page 84) to learn about SSL support for Python, Java, Ruby, and other clients.

Certificate Authorities

For production use, your MongoDB deployment should use valid certificates generated and signed by a single certificate authority. You or your organization can generate and maintain an independent certificate authority, or use certificates generated by a third-party SSL vendor. Obtaining and managing certificates is beyond the scope of this documentation.

.pem File

Before you can use SSL, you must have a .pem file containing a public key certificate and its associated private key.

MongoDB can use any valid SSL certificate issued by a certificate authority, or a self-signed certificate. If you use a self-signed certificate, although the communications channel will be encrypted, there will be *no* validation of server identity. Although such a situation will prevent eavesdropping on the connection, it leaves you vulnerable to a man-in-the-middle attack. Using a certificate signed by a trusted certificate authority will permit MongoDB drivers to verify the server's identity.

In general, avoid using self-signed certificates unless the network is trusted.

Additionally, with regards to *authentication among replica set/sharded cluster members* (page 19), in order to minimize exposure of the private key and allow hostname validation, it is advisable to use different certificates on different servers.

For *testing* purposes, you can generate a self-signed certificate and private key on a Unix system with a command that resembles the following:

```
cd /etc/ssl/  
openssl req -newkey rsa:2048 -new -x509 -days 365 -nodes -out mongodb-cert.crt -keyout mongodb-cert.key
```

This operation generates a new, self-signed certificate with no passphrase that is valid for 365 days. Once you have the certificate, concatenate the certificate and private key to a .pem file, as in the following example:

¹⁹<http://www.mongodb.org/downloads?jmp=docs>

²⁰<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

```
cat mongodb-cert.key mongodb-cert.crt > mongodb.pem
```

See also:

Use x.509 Certificates to Authenticate Clients (page 48)

Procedures**Set Up `mongod` and `mongos` with SSL Certificate and Key**

To use SSL in your MongoDB deployment, include the following run-time options with `mongod` and `mongos`:

- `net.ssl.mode` set to `requireSSL`. This setting restricts each server to use only SSL encrypted connections. You can also specify either the value `allowSSL` or `preferSSL` to set up the use of mixed SSL modes on a port. See `net.ssl.mode` for details.
- `PEMKeyfile` with the `.pem` file that contains the SSL certificate and key.

Consider the following syntax for `mongod`:

```
mongod --sslMode requireSSL --sslPEMKeyFile <pem>
```

For example, given an SSL certificate located at `/etc/ssl/mongodb.pem`, configure `mongod` to use SSL encryption for all connections with the following command:

```
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem
```

Note:

- Specify `<pem>` with the full path name to the certificate.
- If the private key portion of the `<pem>` is encrypted, specify the passphrase. See *SSL Certificate Passphrase* (page 83).

You may also specify these options in the `configuration` file, as in the following examples:

If using the YAML configuration file format:

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/ssl/mongodb.pem
```

Or, if using the older `older` configuration file format²¹:

```
sslMode = requireSSL
sslPEMKeyFile = /etc/ssl/mongodb.pem
```

To connect, to `mongod` and `mongos` instances using SSL, the `mongo` shell and MongoDB tools must include the `--ssl` option. See *TLS/SSL Configuration for Clients* (page 84) for more information on connecting to `mongod` and `mongos` running with SSL.

See also:

Upgrade a Cluster to Use TLS/SSL (page 87)

²¹<https://docs.mongodb.org/v2.4/reference/configuration-options>

Set Up `mongod` and `mongos` with Certificate Validation

To set up `mongod` or `mongos` for SSL encryption using an SSL certificate signed by a certificate authority, include the following run-time options during startup:

- `net.ssl.mode` set to `requireSSL`. This setting restricts each server to use only SSL encrypted connections. You can also specify either the value `allowSSL` or `preferSSL` to set up the use of mixed SSL modes on a port. See `net.ssl.mode` for details.
- `PEMKeyfile` with the name of the `.pem` file that contains the signed SSL certificate and key.
- `CAFile` with the name of the `.pem` file that contains the root certificate chain from the Certificate Authority.

Consider the following syntax for `mongod`:

```
mongod --sslMode requireSSL --sslPEMKeyFile <pem> --sslCAFile <ca>
```

For example, given a signed SSL certificate located at `/etc/ssl/mongodb.pem` and the certificate authority file at `/etc/ssl/ca.pem`, you can configure `mongod` for SSL encryption as follows:

```
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem --sslCAFile /etc/ssl/ca.pem
```

Note:

- Specify the `<pem>` file and the `<ca>` file with either the full path name or the relative path name.
 - If the `<pem>` is encrypted, specify the passphrase. See *SSL Certificate Passphrase* (page 83).
-

You may also specify these options in the `configuration` file, as in the following examples:

If using the YAML configuration file format:

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/ssl/mongodb.pem
    CAFile: /etc/ssl/ca.pem
```

Or, if using the older `configuration` file format²²:

```
sslMode = requireSSL
sslPEMKeyFile = /etc/ssl/mongodb.pem
sslCAFile = /etc/ssl/ca.pem
```

To connect, to `mongod` and `mongos` instances using SSL, the `mongo` tools must include the both the `--ssl` and `--sslPEMKeyFile` option. See *TLS/SSL Configuration for Clients* (page 84) for more information on connecting to `mongod` and `mongos` running with SSL.

See also:

Upgrade a Cluster to Use TLS/SSL (page 87)

Block Revoked Certificates for Clients To prevent clients with revoked certificates from connecting, include the `sslCRLFile` to specify a `.pem` file that contains revoked certificates.

For example, the following `mongod` with SSL configuration includes the `sslCRLFile` setting:

```
mongod --sslMode requireSSL --sslCRLFile /etc/ssl/ca-crl.pem --sslPEMKeyFile /etc/ssl/mongodb.pem --
```

²²<https://docs.mongodb.org/v2.4/reference/configuration-options>

Clients with revoked certificates in the `/etc/ssl/ca-crl.pem` will not be able to connect to this `mongod` instance.

Validate Only if a Client Presents a Certificate In most cases it is important to ensure that clients present valid certificates. However, if you have clients that cannot present a client certificate, or are transitioning to using a certificate authority you may only want to validate certificates from clients that present a certificate.

If you want to bypass validation for clients that don't present certificates, include the `allowConnectionsWithoutCertificates` run-time option with `mongod` and `mongos`. If the client does not present a certificate, no validation occurs. These connections, though not validated, are still encrypted using SSL.

For example, consider the following `mongod` with an SSL configuration that includes the `allowConnectionsWithoutCertificates` setting:

```
mongod --sslMode requireSSL --sslAllowConnectionsWithoutCertificates --sslPEMKeyFile /etc/ssl/mongod
```

Then, clients can connect either with the option `--ssl` and **no** certificate or with the option `--ssl` and a **valid** certificate. See *TLS/SSL Configuration for Clients* (page 84) for more information on SSL connections for clients.

Note: If the client presents a certificate, the certificate must be a valid certificate.

All connections, including those that have not presented certificates are encrypted using SSL.

Disallow Protocols

New in version 3.0.7.

To prevent MongoDB servers from accepting incoming connections that use specific protocols, include the `--sslDisabledProtocols` option, or if using the configuration file the `net.ssl.disabledProtocols` setting.

For example, the following configuration uses `--sslDisabledProtocols` option to prevent `mongod` from accepting incoming connections that use either `TLS1_0` or `TLS1_1`:

```
mongod --sslMode requireSSL --sslDisabledProtocols TLS1_0,TLS1_1 --sslPEMKeyFile /etc/ssl/mongodb.pem
```

If using the YAML configuration file format:

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/ssl/mongodb.pem
    CAFile: /etc/ssl/ca.pem
    disabledProtocols: TLS1_0,TLS1_1
```

For more information, including the protocols recognized by the option, see `net.ssl.disabledProtocols` or the `--sslDisabledProtocols` option for `mongod` and `mongos`.

SSL Certificate Passphrase

The PEM files for `PEMKeyfile` and `ClusterFile` may be encrypted. With encrypted PEM files, you must specify the passphrase at startup with a command-line or a configuration file option or enter the passphrase when prompted.

Changed in version 2.6: In previous versions, you can only specify the passphrase with a command-line or a configuration file option.

To specify the passphrase in clear text on the command line or in a configuration file, use the `PEMKeyPassword` and/or the `ClusterPassword` option.

To have MongoDB prompt for the passphrase at the start of `mongod` or `mongos` and avoid specifying the passphrase in clear text, omit the `PEMKeyPassword` and/or the `ClusterPassword` option. MongoDB will prompt for each passphrase as necessary.

Important: The passphrase prompt option is available if you run the MongoDB instance in the foreground with a connected terminal. If you run `mongod` or `mongos` in a non-interactive session (e.g. without a terminal or as a service on Windows), you cannot use the passphrase prompt option.

Run in FIPS Mode

Note: FIPS-compatible SSL is available only in [MongoDB Enterprise²³](#). See [Configure MongoDB for FIPS](#) (page 88) for more information.

See [Configure MongoDB for FIPS](#) (page 88) for more details.

7.4.2 TLS/SSL Configuration for Clients

On this page

- [mongo Shell SSL Configuration](#) (page 84)
- [MongoDB Cloud Manager and Ops Manager Monitoring Agent](#) (page 86)
- [MongoDB Drivers](#) (page 86)
- [MongoDB Tools](#) (page 87)

Clients must have support for TLS/SSL to work with a `mongod` or a `mongos` instance that has TLS/SSL support enabled.

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

Note: Although TLS is the successor to SSL, this page uses the more familiar term SSL to refer to TLS/SSL.

See also:

[Configure mongod and mongos for TLS/SSL](#) (page 79).

mongo Shell SSL Configuration

For SSL connections, you must use the `mongo` shell built with SSL support or distributed with MongoDB Enterprise. New in version 3.0: Most MongoDB distributions now include support for SSL.

The `mongo` shell provides various `mongo-shell-ssl` settings, including:

- `--ssl`

²³<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

- `--sslPEMKeyFile` with the name of the `.pem` file that contains the SSL certificate and key.
- `--sslCAFile` with the name of the `.pem` file that contains the certificate from the Certificate Authority (CA).

Changed in version 3.0: When running `mongo` with the `--ssl` option, you must include either `--sslCAFile` or `--sslAllowInvalidCertificates`.

This restriction does not apply to the MongoDB tools. However, running the tools without `--sslCAFile` creates the same vulnerability to invalid certificates.

Warning: For SSL connections (`--ssl`) to `mongod` and `mongos`, if the `mongo` shell (or *MongoDB tools* (page 87)) runs without the `--sslCAFile <CAFile>` option (i.e. specifies the `--sslAllowInvalidCertificates` instead), the `mongo` shell (or *MongoDB tools* (page 87)) will not attempt to validate the server certificates. This creates a vulnerability to expired `mongod` and `mongos` certificates as well as to foreign processes posing as valid `mongod` or `mongos` instances. Ensure that you *always* specify the CA file to validate the server certificates in cases where intrusion is a possibility.

- `--sslPEMKeyPassword` option if the client certificate-key file is encrypted.

For a complete list of the `mongo` shell's SSL settings, see *mongo-shell-ssl*.

Connect to MongoDB Instance with SSL Encryption

To connect to a `mongod` or `mongos` instance that requires *only a SSL encryption mode* (page 81), start `mongo` shell with `--ssl` and include the `--sslCAFile` to validate the server certificates.

```
mongo --ssl --sslCAFile /etc/ssl/ca.pem
```

Changed in version 3.0: When running `mongo` with the `--ssl` option, you must include either `--sslCAFile` or `--sslAllowInvalidCertificates`.

This restriction does not apply to the MongoDB tools. However, running the tools without `--sslCAFile` creates the same vulnerability to invalid certificates.

Connect to MongoDB Instance that Requires Client Certificates

To connect to a `mongod` or `mongos` that requires *CA-signed client certificates* (page 82), start the `mongo` shell with `--ssl`, the `--sslPEMKeyFile` option to specify the signed certificate-key file, and the `--sslCAFile` to validate the server certificates.

```
mongo --ssl --sslPEMKeyFile /etc/ssl/client.pem --sslCAFile /etc/ssl/ca.pem
```

Changed in version 3.0: When running `mongo` with the `--ssl` option, you must include either `--sslCAFile` or `--sslAllowInvalidCertificates`.

This restriction does not apply to the MongoDB tools. However, running the tools without `--sslCAFile` creates the same vulnerability to invalid certificates.

Connect to MongoDB Instance that Validates when Presented with a Certificate

To connect to a `mongod` or `mongos` instance that *only requires valid certificates when the client presents a certificate* (page 83), start `mongo` shell either:

- with the `--ssl`, `--sslCAFile`, and **no** certificate or
- with the `--ssl`, `--sslCAFile`, and a **valid** signed certificate.

Changed in version 3.0: When running `mongo` with the `--ssl` option, you must include either `--sslCAFile` or `--sslAllowInvalidCertificates`.

This restriction does not apply to the MongoDB tools. However, running the tools without `--sslCAFile` creates the same vulnerability to invalid certificates.

For example, if `mongod` is running with weak certificate validation, both of the following `mongo` shell clients can connect to that `mongod`:

```
mongo --ssl --sslCAFile /etc/ssl/ca.pem
mongo --ssl --sslPEMKeyFile /etc/ssl/client.pem --sslCAFile /etc/ssl/ca.pem
```

Important: If the client presents a certificate, the certificate must be valid.

MongoDB Cloud Manager and Ops Manager Monitoring Agent

The MongoDB Cloud Manager Monitoring agent will also have to connect via SSL in order to gather its statistics. Because the agent already utilizes SSL for its communications to the MongoDB Cloud Manager servers, this is just a matter of enabling SSL support in MongoDB Cloud Manager itself on a per host basis. See the [MongoDB Cloud Manager documentation](#)²⁴ for more information about SSL configuration.

For Ops Manager, see [Ops Manager documentation](#)²⁵.

MongoDB Drivers

The MongoDB Drivers support for connection to SSL enabled MongoDB. See:

- [C Driver](#)²⁶
- [C++ Driver](#)²⁷
- [C# Driver](#)²⁸
- [Java Driver](#)²⁹
- [Node.js Driver](#)³⁰
- [Perl Driver](#)³¹
- [PHP Driver](#)³²
- [Python Driver](#)³³
- [Ruby Driver](#)³⁴
- [Scala Driver](#)³⁵

²⁴<https://docs.cloud.mongodb.com/>

²⁵<https://docs.opsmanager.mongodb.com/current/>

²⁶<http://api.mongodb.org/c/current/advanced-connections.html>

²⁷<https://github.com/mongodb/mongo-cxx-driver/wiki/Configuring%20the%20Legacy%20Driver>

²⁸<http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/ssl/>

²⁹<http://mongodb.github.io/mongo-java-driver/3.0/driver/reference/connecting/ssl/>

³⁰http://mongodb.github.io/node-mongodb-native/2.0/tutorials/enterprise_features/

³¹<https://metacpan.org/pod/MongoDB::MongoClient#ssl>

³²<http://php.net/manual/en/mongo.connecting.ssl.php>

³³<http://api.mongodb.org/python/current/examples/tls.html>

³⁴<http://docs.mongodb.org/ecosystem/tutorial/ruby-driver-tutorial/#mongodb-x509-mechanism>

³⁵<http://mongodb.github.io/mongo-scala-driver/1.1/reference/connecting/ssl/>

MongoDB Tools

Changed in version 2.6.

Various MongoDB utility programs supports SSL. These tools include:

- mongodump
- mongoexport
- mongofiles
- mongoimport
- mongooplog
- mongorestore
- mongostat
- mongotop

To use SSL connections with these tools, use the same SSL options as the `mongo` shell. See *mongo Shell SSL Configuration* (page 84).

7.4.3 Upgrade a Cluster to Use TLS/SSL

Changed in version 3.0: Most MongoDB distributions now include support for TLS/SSL. See *Configure mongod and mongos for TLS/SSL* (page 79) and *TLS/SSL Configuration for Clients* (page 84) for more information about TLS/SSL and MongoDB.

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

Changed in version 2.6.

The MongoDB server supports listening for both TLS/SSL encrypted and unencrypted connections on the same TCP port. This allows upgrades of MongoDB clusters to use TLS/SSL encrypted connections.

To upgrade from a MongoDB cluster using no TLS/SSL encryption to one using *only* TLS/SSL encryption, use the following rolling upgrade process:

1. For each node of a cluster, start the node with the option `--sslMode` set to `allowSSL`. The `--sslMode allowSSL` setting allows the node to accept both TLS/SSL and non-TLS/non-SSL incoming connections. Its connections to other servers do not use TLS/SSL. Include other *TLS/SSL options* (page 79) as well as any other options that are required for your specific configuration. For example:

```
mongod --replSet <name> --sslMode allowSSL --sslPEMKeyFile <path to TLS/SSL Certificate and key
```

Upgrade all nodes of the cluster to these settings.

You may also specify these options in the configuration file. If using a YAML format configuration file, specify the following settings in the file:

```
net:
  ssl:
    mode: <disabled|allowSSL|preferSSL|requireSSL>
    PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
    CAFile: <path to root CA PEM file>
```

Or, if using the older configuration file format³⁶:

```
sslMode = <disabled|allowSSL|preferSSL|requireSSL>
sslPEMKeyFile = <path to TLS/SSL certificate and key PEM file>
sslCAFile = <path to root CA PEM file>
```

2. Switch all clients to use TLS/SSL. See *TLS/SSL Configuration for Clients* (page 84).
3. For each node of a cluster, use the `setParameter` command to update the `sslMode` to `preferSSL`.³⁷ With `preferSSL` as its `net.ssl.mode`, the node accepts both TLS/SSL and non-TLS/non-SSL incoming connections, and its connections to other servers use TLS/SSL. For example:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "preferSSL" } )
```

Upgrade all nodes of the cluster to these settings.

At this point, all connections should be using TLS/SSL.

4. For each node of the cluster, use the `setParameter` command to update the `sslMode` to `requireSSL`.¹ With `requireSSL` as its `net.ssl.mode`, the node will reject any non-TLS/non-SSL connections. For example:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "requireSSL" } )
```

5. After the upgrade of all nodes, edit the `configuration` file with the appropriate TLS/SSL settings to ensure that upon subsequent restarts, the cluster uses TLS/SSL.

7.4.4 Configure MongoDB for FIPS

On this page

- [Overview](#) (page 88)
- [Prerequisites](#) (page 88)
- [Considerations](#) (page 89)
- [Procedure](#) (page 89)

New in version 2.6.

Overview

The Federal Information Processing Standard (FIPS) is a U.S. government computer security standard used to certify software modules and libraries that encrypt and decrypt data securely. You can configure MongoDB to run with a FIPS 140-2 certified library for OpenSSL. Configure FIPS to run by default or as needed from the command line.

Prerequisites

Important: A full description of FIPS and TLS/SSL is beyond the scope of this document. This tutorial assumes prior knowledge of FIPS and TLS/SSL.

³⁶<https://docs.mongodb.org/v2.4/reference/configuration-options>

³⁷ As an alternative to using the `setParameter` command, you can also restart the nodes with the appropriate TLS/SSL options and values.

Only the [MongoDB Enterprise³⁸](#) version supports FIPS mode. See <https://docs.mongodb.org/manual/administration> to download and install [MongoDB Enterprise³⁹](#) to use FIPS mode.

Your system must have an OpenSSL library configured with the FIPS 140-2 module. At the command line, type `openssl version` to confirm your OpenSSL software includes FIPS support.

For Red Hat Enterprise Linux 6.x (RHEL 6.x) or its derivatives such as CentOS 6.x, the OpenSSL toolkit must be at least `openssl-1.0.1e-16.el6_5` to use FIPS mode. To upgrade the toolkit for these platforms, issue the following command:

```
sudo yum update openssl
```

Some versions of Linux periodically execute a process to *prelink* dynamic libraries with pre-assigned addresses. This process modifies the OpenSSL libraries, specifically `libcrypto`. The OpenSSL FIPS mode will subsequently fail the signature check performed upon startup to ensure `libcrypto` has not been modified since compilation.

To configure the Linux prelink process to not prelink `libcrypto`:

```
sudo bash -c "echo '-b /usr/lib64/libcrypto.so.*' >>/etc/prelink.conf.d/openssl-prelink.conf"
```

Considerations

FIPS is property of the encryption system and not the access control system. However, if your environment requires FIPS compliant encryption *and* access control, you must ensure that the access control system uses only FIPS-compliant encryption.

MongoDB's FIPS support covers the way that MongoDB uses OpenSSL for network encryption and X509 authentication. If you use Kerberos or LDAP Proxy authentication, you must ensure that these external mechanisms are FIPS-compliant. MONGODB-CR authentication is *not* FIPS compliant.

Procedure

Configure MongoDB to use TLS/SSL

See *Configure mongod and mongos for TLS/SSL* (page 79) for details about configuring OpenSSL.

Run mongod or mongos instance in FIPS mode

Perform these steps after you *Configure mongod and mongos for TLS/SSL* (page 79).

Step 1: Change configuration file. To configure your `mongod` or `mongos` instance to use FIPS mode, shut down the instance and update the configuration file with the following setting:

```
net:
  ssl:
    FIPSMode: true
```

³⁸<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

³⁹<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

Step 2: Start `mongod` or `mongos` instance with configuration file. For example, run this command to start the `mongod` instance with its configuration file:

```
mongod --config /etc/mongod.conf
```

Confirm FIPS mode is running

Check the server log file for a message FIPS is active:

```
FIPS 140-2 mode activated
```

7.4.5 Configure Linux `iptables` Firewall for MongoDB

On this page

- [Overview](#) (page 90)
- [Patterns](#) (page 91)
- [Change Default Policy to DROP](#) (page 93)
- [Manage and Maintain `iptables` Configuration](#) (page 93)

On contemporary Linux systems, the `iptables` program provides methods for managing the Linux Kernel's `netfilter` or network packet filtering capabilities. These firewall rules make it possible for administrators to control what hosts can connect to the system, and limit risk exposure by limiting the hosts that can connect to a system.

This document outlines basic firewall configurations for `iptables` firewalls on Linux. Use these approaches as a starting point for your larger networking organization. For a detailed overview of security practices and risk management for MongoDB, see [Security](#) (page 1).

See also:

For MongoDB deployments on Amazon's web services, see the [Amazon EC2⁴⁰](#) page, which addresses Amazon's Security Groups and other EC2-specific security features.

Overview

Rules in `iptables` configurations fall into chains, which describe the process for filtering and processing specific streams of traffic. Chains have an order, and packets must pass through earlier rules in a chain to reach later rules. This document addresses only the following two chains:

INPUT Controls all incoming traffic.

OUTPUT Controls all outgoing traffic.

Given the default ports of all MongoDB processes, you must configure networking rules that permit *only* required communication between your application and the appropriate `mongod` and `mongos` instances.

Be aware that, by default, the default policy of `iptables` is to allow all connections and traffic unless explicitly disabled. The configuration changes outlined in this document will create rules that explicitly allow traffic from specific addresses and on specific ports, using a default policy that drops all traffic that is not explicitly allowed. When you have properly configured your `iptables` rules to allow only the traffic that you want to permit, you can [Change Default Policy to DROP](#) (page 93).

⁴⁰<https://docs.mongodb.org/ecosystem/platforms/amazon-ec2>

Patterns

This section contains a number of patterns and examples for configuring `iptables` for use with MongoDB deployments. If you have configured different ports using the `port` configuration setting, you will need to modify the rules accordingly.

Traffic to and from `mongod` Instances

This pattern is applicable to all `mongod` instances running as standalone instances or as part of a *replica set*.

The goal of this pattern is to explicitly allow traffic to the `mongod` instance from the application server. In the following examples, replace `<ip-address>` with the IP address of the application server:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27017 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27017 -m state --state ESTABLISHED -j ACCEPT
```

The first rule allows all incoming traffic from `<ip-address>` on port 27017, which allows the application server to connect to the `mongod` instance. The second rule, allows outgoing traffic from the `mongod` to reach the application server.

Optional

If you have only one application server, you can replace `<ip-address>` with either the IP address itself, such as 198.51.100.55. You can also express this using CIDR notation as 198.51.100.55/32. If you want to permit a larger block of possible IP addresses you can allow traffic from a /24 using one of the following specifications for the `<ip-address>`, as follows:

```
10.10.10.10/24
10.10.10.10/255.255.255.0
```

Traffic to and from `mongos` Instances

`mongos` instances provide query routing for *sharded clusters*. Clients connect to `mongos` instances, which behave from the client's perspective as `mongod` instances. In turn, the `mongos` connects to all `mongod` instances that are components of the sharded cluster.

Use the same `iptables` command to allow traffic to and from these instances as you would from the `mongod` instances that are members of the replica set. Take the configuration outlined in the *Traffic to and from `mongod` Instances* (page 91) section as an example.

Traffic to and from a MongoDB Config Server

Config servers, host the *config database* that stores metadata for sharded clusters. Each production cluster has three config servers, initiated using the `mongod --configsvr` option.⁴¹ Config servers listen for connections on port 27019. As a result, add the following `iptables` rules to the config server to allow incoming and outgoing connection on port 27019, for connection to the other config servers.

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27019 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27019 -m state --state ESTABLISHED -j ACCEPT
```

Replace `<ip-address>` with the address or address space of *all* the `mongod` that provide config servers.

⁴¹ You also can run a config server by using the `configsvr` value for the `clusterRole` setting in a configuration file.

Additionally, config servers need to allow incoming connections from all of the `mongos` instances in the cluster *and* all `mongod` instances in the cluster. Add rules that resemble the following:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27019 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Replace `<ip-address>` with the address of the `mongos` instances and the shard `mongod` instances.

Traffic to and from a MongoDB Shard Server

For shard servers, running as `mongod --shardsvr`⁴² Because the default port number is 27018 when running with the `shardsvr` value for the `clusterRole` setting, you must configure the following `iptables` rules to allow traffic to and from each shard:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27018 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27018 -m state --state ESTABLISHED -j ACCEPT
```

Replace the `<ip-address>` specification with the IP address of all `mongod`. This allows you to permit incoming and outgoing traffic between all shards including constituent replica set members, to:

- all `mongod` instances in the shard's replica sets.
- all `mongod` instances in other shards.⁴³

Furthermore, shards need to be able make outgoing connections to:

- all `mongod` instances in the config servers.

Create a rule that resembles the following, and replace the `<ip-address>` with the address of the config servers and the `mongos` instances:

```
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27018 -m state --state ESTABLISHED -j ACCEPT
```

Provide Access For Monitoring Systems

1. The `mongostat` diagnostic tool, when running with the `--discover` needs to be able to reach all components of a cluster, including the config servers, the shard servers, and the `mongos` instances.
2. If your monitoring system needs access the HTTP interface, insert the following rule to the chain:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 28017 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Replace `<ip-address>` with the address of the instance that needs access to the HTTP or REST interface. For *all* deployments, you should restrict access to this port to *only* the monitoring instance.

Optional

For config server `mongod` instances running with the `shardsvr` value for the `clusterRole` setting, the rule would resemble the following:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 28018 -m state --state NEW,ESTABLISHED -j ACCEPT
```

For config server `mongod` instances running with the `configsrv` value for the `clusterRole` setting, the rule would resemble the following:

⁴² You can also specify the shard server option with the `shardsvr` value for the `clusterRole` setting in the configuration file. Shard members are also often conventional replica sets using the default port.

⁴³ All shards in a cluster need to be able to communicate with all other shards to facilitate *chunk* and balancing operations.

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 28019 -m state --state NEW,ESTABLISH
```

Change Default Policy to DROP

The default policy for `iptables` chains is to allow all traffic. After completing all `iptables` configuration changes, you *must* change the default policy to `DROP` so that all traffic that isn't explicitly allowed as above will not be able to reach components of the MongoDB deployment. Issue the following commands to change this policy:

```
iptables -P INPUT DROP
```

```
iptables -P OUTPUT DROP
```

Manage and Maintain `iptables` Configuration

This section contains a number of basic operations for managing and using `iptables`. There are various front end tools that automate some aspects of `iptables` configuration, but at the core all `iptables` front ends provide the same basic functionality:

Make all `iptables` Rules Persistent

By default all `iptables` rules are only stored in memory. When your system restarts, your firewall rules will revert to their defaults. When you have tested a rule set and have guaranteed that it effectively controls traffic you can use the following operations to you should make the rule set persistent.

On Red Hat Enterprise Linux, Fedora Linux, and related distributions you can issue the following command:

```
service iptables save
```

On Debian, Ubuntu, and related distributions, you can use the following command to dump the `iptables` rules to the `/etc/iptables.conf` file:

```
iptables-save > /etc/iptables.conf
```

Run the following operation to restore the network rules:

```
iptables-restore < /etc/iptables.conf
```

Place this command in your `rc.local` file, or in the `/etc/network/if-up.d/iptables` file with other similar operations.

List all `iptables` Rules

To list all of currently applied `iptables` rules, use the following operation at the system shell.

```
iptables -L
```

Flush all `iptables` Rules

If you make a configuration mistake when entering `iptables` rules or simply need to revert to the default rule set, you can use the following operation at the system shell to flush all rules:

```
iptables -F
```

If you've already made your `iptables` rules persistent, you will need to repeat the appropriate procedure in the *Make all iptables Rules Persistent* (page 93) section.

7.4.6 Configure Windows `netsh` Firewall for MongoDB

On this page

- [Overview](#) (page 94)
- [Patterns](#) (page 95)
- [Manage and Maintain *Windows Firewall* Configurations](#) (page 96)

On Windows Server systems, the `netsh` program provides methods for managing the *Windows Firewall*. These firewall rules make it possible for administrators to control what hosts can connect to the system, and limit risk exposure by limiting the hosts that can connect to a system.

This document outlines basic *Windows Firewall* configurations. Use these approaches as a starting point for your larger networking organization. For a detailed over view of security practices and risk management for MongoDB, see *Security* (page 1).

See also:

[Windows Firewall](#)⁴⁴ documentation from Microsoft.

Overview

Windows Firewall processes rules in an ordered determined by rule type, and parsed in the following order:

1. Windows Service Hardening
2. Connection security rules
3. Authenticated Bypass Rules
4. Block Rules
5. Allow Rules
6. Default Rules

By default, the policy in *Windows Firewall* allows all outbound connections and blocks all incoming connections.

Given the default ports of all MongoDB processes, you must configure networking rules that permit *only* required communication between your application and the appropriate `mongod.exe` and `mongos.exe` instances.

The configuration changes outlined in this document will create rules which explicitly allow traffic from specific addresses and on specific ports, using a default policy that drops all traffic that is not explicitly allowed.

You can configure the *Windows Firewall* with using the `netsh` command line tool or through a windows application. On Windows Server 2008 this application is *Windows Firewall With Advanced Security* in *Administrative Tools*. On previous versions of Windows Server, access the *Windows Firewall* application in the *System and Security* control panel.

The procedures in this document use the `netsh` command line tool.

⁴⁴<http://technet.microsoft.com/en-us/network/bb545423.aspx>

Patterns

This section contains a number of patterns and examples for configuring *Windows Firewall* for use with MongoDB deployments. If you have configured different ports using the `port` configuration setting, you will need to modify the rules accordingly.

Traffic to and from `mongod.exe` Instances

This pattern is applicable to all `mongod.exe` instances running as standalone instances or as part of a *replica set*. The goal of this pattern is to explicitly allow traffic to the `mongod.exe` instance from the application server.

```
netsh advfirewall firewall add rule name="Open mongod port 27017" dir=in action=allow protocol=TCP l
```

This rule allows all incoming traffic to port 27017, which allows the application server to connect to the `mongod.exe` instance.

Windows Firewall also allows enabling network access for an entire application rather than to a specific port, as in the following example:

```
netsh advfirewall firewall add rule name="Allowing mongod" dir=in action=allow program=" C:\mongodb\l
```

You can allow all access for a `mongos.exe` server, with the following invocation:

```
netsh advfirewall firewall add rule name="Allowing mongos" dir=in action=allow program=" C:\mongodb\l
```

Traffic to and from `mongos.exe` Instances

`mongos.exe` instances provide query routing for *sharded clusters*. Clients connect to `mongos.exe` instances, which behave from the client's perspective as `mongod.exe` instances. In turn, the `mongos.exe` connects to all `mongod.exe` instances that are components of the sharded cluster.

Use the same *Windows Firewall* command to allow traffic to and from these instances as you would from the `mongod.exe` instances that are members of the replica set.

```
netsh advfirewall firewall add rule name="Open mongod shard port 27018" dir=in action=allow protocol=
```

Traffic to and from a MongoDB Config Server

Configuration servers, host the *config database* that stores metadata for sharded clusters. Each production cluster has three configuration servers, initiated using the `mongod --configsvr` option.⁴⁵ Configuration servers listen for connections on port 27019. As a result, add the following *Windows Firewall* rules to the config server to allow incoming and outgoing connection on port 27019, for connection to the other config servers.

```
netsh advfirewall firewall add rule name="Open mongod config svr port 27019" dir=in action=allow prot
```

Additionally, config servers need to allow incoming connections from all of the `mongos.exe` instances in the cluster and all `mongod.exe` instances in the cluster. Add rules that resemble the following:

```
netsh advfirewall firewall add rule name="Open mongod config svr inbound" dir=in action=allow protoc
```

Replace `<ip-address>` with the addresses of the `mongos.exe` instances and the shard `mongod.exe` instances.

⁴⁵ You also can run a config server by using the `configsvr` value for the `clusterRole` setting in a configuration file.

Traffic to and from a MongoDB Shard Server

For shard servers, running as `mongod --shardsvr`⁴⁶ Because the default port number is 27018 when running with the `shardsvr` value for the `clusterRole` setting, you must configure the following *Windows Firewall* rules to allow traffic to and from each shard:

```
netsh advfirewall firewall add rule name="Open mongod shardsvr inbound" dir=in action=allow protocol=TCP localport=27018
netsh advfirewall firewall add rule name="Open mongod shardsvr outbound" dir=out action=allow protocol=TCP localport=27018
```

Replace the `<ip-address>` specification with the IP address of all `mongod.exe` instances. This allows you to permit incoming and outgoing traffic between all shards including constituent replica set members to:

- all `mongod.exe` instances in the shard's replica sets.
- all `mongod.exe` instances in other shards.⁴⁷

Furthermore, shards need to be able make outgoing connections to:

- all `mongos.exe` instances.
- all `mongod.exe` instances in the config servers.

Create a rule that resembles the following, and replace the `<ip-address>` with the address of the config servers and the `mongos.exe` instances:

```
netsh advfirewall firewall add rule name="Open mongod config svr outbound" dir=out action=allow protocol=TCP localport=27018
```

Provide Access For Monitoring Systems

1. The `mongostat` diagnostic tool, when running with the `--discover` needs to be able to reach all components of a cluster, including the config servers, the shard servers, and the `mongos.exe` instances.
2. If your monitoring system needs access the HTTP interface, insert the following rule to the chain:

```
netsh advfirewall firewall add rule name="Open mongod HTTP monitoring inbound" dir=in action=allow protocol=TCP localport=27018
```

Replace `<ip-address>` with the address of the instance that needs access to the HTTP or REST interface. For *all* deployments, you should restrict access to this port to *only* the monitoring instance.

Optional

For config server `mongod` instances running with the `shardsvr` value for the `clusterRole` setting, the rule would resemble the following:

```
netsh advfirewall firewall add rule name="Open mongos HTTP monitoring inbound" dir=in action=allow protocol=TCP localport=27018
```

For config server `mongod` instances running with the `configsvr` value for the `clusterRole` setting, the rule would resemble the following:

```
netsh advfirewall firewall add rule name="Open mongod configsvr HTTP monitoring inbound" dir=in action=allow protocol=TCP localport=27018
```

Manage and Maintain *Windows Firewall* Configurations

This section contains a number of basic operations for managing and using `netsh`. While you can use the GUI front ends to manage the *Windows Firewall*, all core functionality is accessible from `netsh`.

⁴⁶ You can also specify the shard server option with the `shardsvr` value for the `clusterRole` setting in the configuration file. Shard members are also often conventional replica sets using the default port.

⁴⁷ All shards in a cluster need to be able to communicate with all other shards to facilitate *chunk* and balancing operations.

Delete all *Windows Firewall* Rules

To delete the firewall rule allowing `mongod.exe` traffic:

```
netsh advfirewall firewall delete rule name="Open mongod port 27017" protocol=tcp localport=27017
```

```
netsh advfirewall firewall delete rule name="Open mongod shard port 27018" protocol=tcp localport=27018
```

List All *Windows Firewall* Rules

To return a list of all *Windows Firewall* rules:

```
netsh advfirewall firewall show rule name=all
```

Reset *Windows Firewall*

To reset the *Windows Firewall* rules:

```
netsh advfirewall reset
```

Backup and Restore *Windows Firewall* Rules

To simplify administration of larger collection of systems, you can export or import firewall systems from different servers) rules very easily on Windows:

Export all firewall rules with the following command:

```
netsh advfirewall export "C:\temp\MongoDBfw.wfw"
```

Replace `"C:\temp\MongoDBfw.wfw"` with a path of your choosing. You can use a command in the following form to import a file created using this operation:

```
netsh advfirewall import "C:\temp\MongoDBfw.wfw"
```

7.5 Encryption

The following tutorials provide

[Configure Encryption \(page 97\)](#) Configure MongoDB for encryption and key management.

[Rotate Encryption Keys \(page 100\)](#) Rotate out encryption key.

7.5.1 Configure Encryption

On this page

- [Overview \(page 98\)](#)
- [Key Manager \(page 98\)](#)
- [Local Key Management \(page 99\)](#)

New in version 3.2.

Overview

Enterprise Feature

Available in MongoDB Enterprise only.

Important: Available for the WiredTiger Storage Engine Only.

MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine. With storage encryption, the secure management of the encryption keys is critical.

Only the system key is external to the server and requires external management. To manage the system key, MongoDB's encrypted storage engine supports two key management options:

- Integration with a third party key management appliance via the Key Management Interoperability Protocol (KMIP). **Recommended**
- Use of local key management via a keyfile.

The following tutorial outlines the procedures to configure MongoDB for encryption and key management.

Key Manager

MongoDB Enterprise supports secure transfer of keys with compatible key management appliances. Using a key manager allows for the keys to be stored in the key manager.

MongoDB Enterprise supports secure transfer of keys with Key Management Interoperability Protocol (KMIP) compliant key management appliances. While any appliance vendor that provides support for KMIP is expected to be compatible, MongoDB has certified against SafeNet KeySecure and Vormetric Data Security Manager (DSM).

Recommended

Using a key manager meets regulatory key management guidelines, such as HIPAA, PCI-DSS, and FERPA, and is recommended over the local key management.

Prerequisites

- Your key manager must support the KMIP communication protocol, such as Vormetric DSM and Safenet Key-Secure.
- For Vormetric or Safenet, you must have a valid certificates issued by the specific appliance vendor in order to authenticate MongoDB to the KMIP server.

Encrypt Using a New Key

To create a new key, connect `mongod` to the key manager by starting `mongod` with the following options:

- `--enableEncryption,`
- `--kmipServerName <KMIP Server Hostname>,`
- `--kmipServerCAFile <path to KMIP Server's CA File>, and`
- `--kmipClientCertificateFile <path to valid client certificate>.`

Include any other options specific to your configuration.

```
mongod --enableEncryption --kmipServerName <KMIP Server HostName> \
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem
```

This operation creates a new system key in your key manager for use by the `mongod` to wrap the keys `mongod` generates for each database.

To verify that the key creation and usage was successful, check the log file. If successful, the process will log the following messages:

```
[initandlisten] Created KMIP key with id: <UID>
[initandlisten] Encryption key manager initialized using system key with id: <UID>
```

See also:

encryption-key-management-options,

Encrypt Using an Existing Key

You can use an existing system key created and managed by your KMIP. To use an existing key, connect `mongod` to the key manager by starting `mongod` with the following options:

- `--enableEncryption`,
- `--kmipServerName <KMIP Server Hostname>`,
- `--kmipServerCAFile <path to KMIP Server's CA File>`,
- `--kmipClientCertificateFile <path to valid client certificate>`, and
- `--kmipKeyIdentifier <UID>`.

Include any other options specific to your configuration.

```
mongod --enableEncryption --kmipServerName <KMIP Server HostName> \
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem \
  --kmipKeyIdentifier <UID>
```

Important: If data is already encrypted with a key, you must specify that key's `<UID>` for the `--kmipKeyIdentifier` option. Otherwise, MongoDB will not start and log an error.

See also:

encryption-key-management-options

Local Key Management

Important: Using the keyfile method does not meet most regulatory key management guidelines and requires users to securely manage their own keys.

The safe management of the keyfile is critical.

To encrypt using a keyfile, you must have a base64 encoded keyfile that contains a 16 or 32 character string. The keyfile must only be accessible by the owner of the `mongod` process.

1. Create the base64 encoded keyfile with the 16 or 32 character string. You can generate the encoded keyfile using any method you prefer. For example,


```
openssl rand -base64 32 > mongodb-keyfile
```

2. Update the file permissions.

```
chmod 600 mongodb-keyfile
```

3. To use the key file, start `mongod` with the following options:

- `--enableEncryption`,
- `--encryptionKeyFile <path to keyfile>`,

```
mongod --enableEncryption --encryptionKeyFile mongodb-keyfile
```

4. Verify if the encryption key manager successfully initialized with the keyfile. If the operation was successful, the process will log the following message:

```
[initandlisten] Encryption key manager initialized using system key with id:
```

See also:

[encryption-key-management-options](#)

7.5.2 Rotate Encryption Keys

On this page

- [Rotate a Member of Replica Set](#) (page 100)
- [KMIP Master Key Rotation](#) (page 101)

Most regulatory requirements mandate that a managed key used to decrypt sensitive data must be rotated out and replaced with a new key once a year.

MongoDB provides two options for key rotation. You can rotate out the binary with a new instance that uses a new key. Or, if you are using a KMIP server for key management, you can rotate the master key.

Rotate a Member of Replica Set

For a replica set, to rotate out a member:

1. Start a new `mongod` instance, configured to use a new key. Include the `--replSet` option with the name of the replica set as well as any other options specific to your configuration, such as `--dbpath`.

```
mongod --replSet myReplSet --enableEncryption --kmpServerName  
<KMIP Server HostName> \ --kmpServerCAFile ca.pem  
--kmpClientCertificateFile client.pem
```

2. Connect a `mongo` shell to the replica set's primary.
3. Add the instance to the replica set.

```
rs.add("<hostname>:<port>")
```

During the initial sync process, the re-encryption of the data with an entirely new set of database keys as well as a new system key occurs.

4. Once the new node completes its initial sync process, remove the old node from the replica set and delete all its data. For instructions, see <https://docs.mongodb.org/manual/tutorial/remove-replica-set-member>

KMIP Master Key Rotation

If you are using a KMIP server for key management, you can rotate the master key, the only externally managed key. With the new master key, the internal keystore will be re-encrypted but the database keys will be otherwise left unchanged. This obviates the need to re-encrypt the entire data set.

1. Rotate the master key for the *secondary* members of the replica set one at a time.
 - (a) Restart the secondary, including the `--kmipRotateMasterKey` parameter. Include any other options specific to your configuration. If the member already includes the `--kmipKeyIdentifier` option, either update the `--kmipKeyIdentifier` option with the new key to use or omit to request a new key from the KMIP server:

```
mongod --enableEncryption --kmipRotateMasterKey \
  --kmipServerName <KMIP Server HostName> \
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem
```

If using a configuration file, include the `security.kmip.rotateMasterKey`.

- (b) Upon successful completion of the master key rotation and re-encryption of the database keystore, the mongod will exit.
 - (c) Restart the secondary without the `--kmipRotateMasterKey` parameter. Include any other options specific to your configuration.

```
mongod --enableEncryption --kmipServerName <KMIP Server HostName> \
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem
```

If using a configuration file, remove the `security.kmip.rotateMasterKey` setting.

2. Step down the replica set primary.

Connect a mongo shell to the primary and use `rs.stepDown()` to step down the primary and force an election of a new primary:

```
rs.stepDown()
```

3. When `rs.status()` shows that the primary has stepped down and another member has assumed PRIMARY state, rotate the master key for the stepped down member:

- (a) Restart the stepped-down member, including the `--kmipRotateMasterKey` parameter. Include any other options specific to your configuration. If the member already includes the `--kmipKeyIdentifier` option, either update the `--kmipKeyIdentifier` option with the new key to use or omit.

```
mongod --enableEncryption --kmipRotateMasterKey \
  --kmipServerName <KMIP Server HostName> \
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem
```

If using a configuration file, include the `security.kmip.rotateMasterKey`.

- (b) Upon successful completion of the master key rotation and re-encryption of the database keystore, the mongod will exit.
 - (c) Restart the stepped-down member without the `--kmipRotateMasterKey` parameter. Include any other options specific to your configuration.

```
mongod --enableEncryption --kmipServerName <KMIP Server HostName> \  
  --kmipServerCAFile ca.pem --kmipClientCertificateFile client.pem
```

If using a configuration file, remove the `security.kmip.rotateMasterKey` setting.

7.6 Auditing

The following tutorials provide instructions on how to enable auditing for system events and specify which events to audit.

Configure Auditing (page 102) Enable and configure MongoDB Enterprise system event auditing feature.

Configure Audit Filters (page 104) Specify which events to audit.

7.6.1 Configure Auditing

On this page

- [Enable and Configure Audit Output \(page 102\)](#)

New in version 2.6.

MongoDB Enterprise⁴⁸ supports *auditing* (page 33) of various operations. A complete auditing solution must involve **all** `mongod` server and `mongos` router processes.

The audit facility can write audit events to the console, the *syslog* (option is unavailable on Windows), a JSON file, or a BSON file. For details on the audited operations and the audit log messages, see *System Event Audit Messages* (page 134).

Enable and Configure Audit Output

Use the `--auditDestination` option to enable auditing and specify where to output the audit events.

Warning: For sharded clusters, if you enable auditing on `mongos` instances, you must enable auditing on all `mongod` instances in the cluster, i.e. shards and config servers.

Output to Syslog

To enable auditing and print audit events to the syslog (option is unavailable on Windows) in JSON format, specify `syslog` for the `--auditDestination` setting. For example:

```
mongod --dbpath data/db --auditDestination syslog
```

Warning: The syslog message limit can result in the truncation of the audit messages. The auditing system will neither detect the truncation nor error upon its occurrence.

You may also specify these options in the `configuration` file:

⁴⁸<https://www.mongodb.com/products/mongodb-enterprise-advanced?jmp=docs>

```
storage:
  dbPath: data/db
auditLog:
  destination: syslog
```

Output to Console

To enable auditing and print the audit events to standard output (i.e. `stdout`), specify `console` for the `--auditDestination` setting. For example:

```
mongod --dbpath data/db --auditDestination console
```

You may also specify these options in the configuration file:

```
storage:
  dbPath: data/db
auditLog:
  destination: console
```

Output to JSON File

To enable auditing and print audit events to a file in JSON format, specify `file` for the `--auditDestination` setting, `JSON` for the `--auditFormat` setting, and the output filename for the `--auditPath`. The `--auditPath` option accepts either full path name or relative path name. For example, the following enables auditing and records audit events to a file with the relative path name of `data/db/auditLog.json`:

```
mongod --dbpath data/db --auditDestination file --auditFormat JSON --auditPath data/db/auditLog.json
```

The audit file rotates at the same time as the server log file.

You may also specify these options in the configuration file:

```
storage:
  dbPath: data/db
auditLog:
  destination: file
  format: JSON
  path: data/db/auditLog.json
```

Note: Printing audit events to a file in JSON format degrades server performance more than printing to a file in BSON format.

Output to BSON File

To enable auditing and print audit events to a file in BSON binary format, specify `file` for the `--auditDestination` setting, `BSON` for the `--auditFormat` setting, and the output filename for the `--auditPath`. The `--auditPath` option accepts either full path name or relative path name. For example, the following enables auditing and records audit events to a BSON file with the relative path name of `data/db/auditLog.bson`:

```
mongod --dbpath data/db --auditDestination file --auditFormat BSON --auditPath data/db/auditLog.bson
```

The audit file rotates at the same time as the server log file.

You may also specify these options in the `configuration` file:

```
storage:
  dbPath: data/db
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
```

To view the contents of the file, pass the file to the MongoDB utility `bsondump`. For example, the following converts the audit log into a human-readable form and output to the terminal:

```
bsondump data/db/auditLog.bson
```

See also:

[Configure Audit Filters](#) (page 104), [Auditing](#) (page 33), [System Event Audit Messages](#) (page 134)

7.6.2 Configure Audit Filters

On this page

- [--auditFilter Option](#) (page 104)
- [Examples](#) (page 104)

MongoDB Enterprise⁴⁹ supports *auditing* (page 33) of various operations. When *enabled* (page 102), the audit facility, by default, records all auditable operations as detailed in *Audit Event Actions, Details, and Results* (page 135). To specify which events to record, the audit feature includes the `--auditFilter` option.

--auditFilter Option

The `--auditFilter` option takes a string representation of a query document of the form:

```
{ <field1>: <expression1>, ... }
```

- The `<field>` can be *any field in the audit message* (page 134), including fields returned in the *param* (page 135) document.
- The `<expression>` is a *query condition expression*.

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

To specify the audit filter in a `configuration` file, you must use the YAML format of the configuration file.

Examples

Filter for Multiple Operation Types

The following example audits only the `createCollection` (page 130) and `dropCollection` (page 130) actions by using the filter:

```
{ atype: { $in: [ "createCollection", "dropCollection" ] } }
```

⁴⁹<https://www.mongodb.com/products/mongodb-enterprise-advanced?jmp=docs>

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auditDestination file --auditFilter '{ atype: { $in: [ "createCollection",
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ atype: { $in: [ "createCollection", "dropCollection" ] } }'
```

Filter on Authentication Operations on a Single Database

The <field> can include *any field in the audit message* (page 134). For authentication operations (i.e. `atype: "authenticate"`), the audit messages include a `db` field in the `param` document.

The following example audits only the `authenticate` operations that occur against the `test` database by using the filter:

```
{ atype: "authenticate", "param.db": "test" }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ atype: "authenticate", "param
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
security:
  authorization: enabled
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ atype: "authenticate", "param.db": "test" }'
```

To filter on all `authenticate` operations across databases, use the filter `{ atype: "authenticate" }`.

Filter on Collection Creation and Drop Operations for a Single Database

The <field> can include *any field in the audit message* (page 134). For collection creation and drop operations (i.e. `atype: "createCollection"` and `atype: "dropCollection"`), the audit messages include a `namespace ns` field in the `param` document.

The following example audits only the `createCollection` and `dropCollection` operations that occur against the `test` database by using the filter:

Note: The regular expression requires two backslashes (`\\`) to escape the dot (`.`).

```
{ atype: { $in: [ "createCollection", "dropCollection" ] }, "param.ns": /^test\\.\/ } }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ atype: { $in: [ "createCollection"
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
security:
  authorization: enabled
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ atype: { $in: [ "createCollection", "dropCollection" ] }, "param.ns": /^test\\.\/ } }'
```

Filter by Authorization Role

The following example audits operations by users with `readWrite` (page 115) role on the `test` database, including users with roles that inherit from `readWrite` (page 115), by using the filter:

```
{ roles: { role: "readWrite", db: "test" } }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ roles: { role: "readWrite",
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
security:
  authorization: enabled
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ roles: { role: "readWrite", db: "test" } }'
```

Filter on Read and Write Operations

To capture read and write operations in the audit, you must also enable the audit system to log authorization successes using the `auditAuthorizationSuccess` parameter.⁵⁰

Note: Enabling `auditAuthorizationSuccess` degrades performance more than logging only the authorization failures.

The following example audits the `find()`, `insert()`, `remove()`, `update()`, `save()`, and `findAndModify()` operations by using the filter:

```
{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "findandmodify"
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

⁵⁰ You can enable `auditAuthorizationSuccess` parameter without enabling `--auth`; however, all operations will return success for authorization checks.

```
mongod --dbpath data/db --auth --setParameter auditAuthorizationSuccess=true --auditDestination file
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
security:
  authorization: enabled
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "f
setParameter: { auditAuthorizationSuccess: true }
```

Filter on Read and Write Operations for a Collection

To capture read and write operations in the audit, you must also enable the audit system to log authorization successes using the `auditAuthorizationSuccess` parameter. ¹

Note: Enabling `auditAuthorizationSuccess` degrades performance more than logging only the authorization failures.

The following example audits the `find()`, `insert()`, `remove()`, `update()`, `save()`, and `findAndModify()` operations for the collection `orders` in the database `test` by using the filter:

```
{ atype: "authCheck", "param.ns": "test.orders", "param.command": { $in: [ "find", "insert", "delete
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --setParameter auditAuthorizationSuccess=true --auditDestination file
```

To specify the audit filter in a configuration file, you must use the YAML format of the configuration file.

```
storage:
  dbPath: data/db
security:
  authorization: enabled
auditLog:
  destination: file
  format: BSON
  path: data/db/auditLog.bson
  filter: '{ atype: "authCheck", "param.ns": "test.orders", "param.command": { $in: [ "find", "inse
setParameter: { auditAuthorizationSuccess: true }
```

See also:

Configure Auditing (page 102), *Auditing* (page 33), *System Event Audit Messages* (page 134)

7.7 Miscellaneous

The following pages provide additional information regarding handling security.

Implement Field Level Redaction (page 108) Procedure to set up different access levels for the document content.

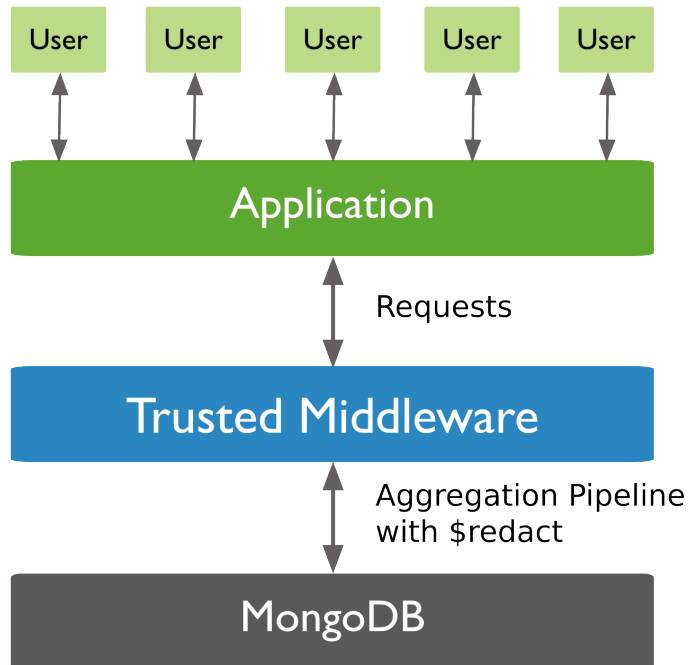
Create a Vulnerability Report (page 110) Report a vulnerability in MongoDB.

7.7.1 Implement Field Level Redaction

On this page

- [Procedure](#) (page 108)

The `$redact` pipeline operator restricts the contents of the documents based on information stored in the documents themselves.



To store the access criteria data, add a field to the documents and embedded documents. To allow for multiple combinations of access levels for the same data, consider setting the access field to an array of arrays. Each array element contains a required set that allows a user with that set to access the data.

Then, include the `$redact` stage in the `db.collection.aggregate()` operation to restrict contents of the result set based on the access required to view the data.

For more information on the `$redact` pipeline operator, including its syntax and associated system variables as well as additional examples, see `$redact`.

Procedure

For example, a `forecasts` collection contains documents of the following form where the `tags` field determines the access levels required to view the data:

```
{
  _id: 1,
  title: "123 Department Report",
```

```

tags: [ [ "G" ], [ "FDW" ] ],
year: 2014,
subsections: [
  {
    subtitle: "Section 1: Overview",
    tags: [ [ "SI", "G" ], [ "FDW" ] ],
    content: "Section 1: This is the content of section 1."
  },
  {
    subtitle: "Section 2: Analysis",
    tags: [ [ "STLW" ] ],
    content: "Section 2: This is the content of section 2."
  },
  {
    subtitle: "Section 3: Budgeting",
    tags: [ [ "TK" ], [ "FDW", "TGE" ] ],
    content: {
      text: "Section 3: This is the content of section3.",
      tags: [ [ "HCS"], [ "FDW", "TGE", "BX" ] ]
    }
  }
]
}

```

For each document, the `tags` field contains various access groupings necessary to view the data. For example, the value `[["G"], ["FDW", "TGE"]]` can specify that a user requires either access level `["G"]` or both `["FDW", "TGE"]` to view the data.

Consider a user who only has access to view information tagged with either `"FDW"` or `"TGE"`. To run a query on all documents with year 2014 for this user, include a `$redact` stage as in the following:

```

var userAccess = [ "FDW", "TGE" ];
db.forecasts.aggregate(
  [
    { $match: { year: 2014 } },
    { $redact:
      {
        $cond: {
          if: { $anyElementTrue:
            {
              $map: {
                input: "$tags" ,
                as: "fieldTag",
                in: { $setIsSubset: [ "$$fieldTag", userAccess ] }
              }
            }
          },
          then: "$$DESCEND",
          else: "$$PRUNE"
        }
      }
    }
  ]
)

```

The aggregation operation returns the following “redacted” document for the user:

```

{ "_id" : 1,
  "title" : "123 Department Report",

```

```
"tags" : [ [ "G" ], [ "FDW" ] ],
"year" : 2014,
"subsections" :
  [
    {
      "subtitle" : "Section 1: Overview",
      "tags" : [ [ "SI", "G" ], [ "FDW" ] ],
      "content" : "Section 1: This is the content of section 1."
    },
    {
      "subtitle" : "Section 3: Budgeting",
      "tags" : [ [ "TK" ], [ "FDW", "TGE" ] ]
    }
  ]
}
```

See also:

\$map, \$setIsSubset, \$anyElementTrue

7.7.2 Create a Vulnerability Report

On this page

- [Create the Report in JIRA](#) (page 110)
- [Information to Provide](#) (page 110)
- [Send the Report via Email](#) (page 111)
- [Evaluation of a Vulnerability Report](#) (page 111)
- [Disclosure](#) (page 111)

If you believe you have discovered a vulnerability in MongoDB or have experienced a security incident related to MongoDB, please report the issue to aid in its resolution.

To report an issue, we strongly suggest filing a ticket in the [SECURITY](#)⁵¹ project in JIRA. MongoDB, Inc responds to vulnerability notifications within 48 hours.

Create the Report in JIRA

Submit a [Ticket](#)⁵² in the [Security](#)⁵³ project on our JIRA. The ticket number will become the reference identification for the issue for its lifetime. You can use this identifier for tracking purposes.

Information to Provide

All vulnerability reports should contain as much information as possible so MongoDB's developers can move quickly to resolve the issue. In particular, please include the following:

- The name of the product.
- *Common Vulnerability* information, if applicable, including:
- CVSS (Common Vulnerability Scoring System) Score.

⁵¹<https://jira.mongodb.org/browse/SECURITY>

⁵²<https://jira.mongodb.org/secure/CreateIssue!default.jspa?project-field=%22Security%22>

⁵³<https://jira.mongodb.org/browse/SECURITY>

- CVE (Common Vulnerability and Exposures) Identifier.
- Contact information, including an email address and/or phone number, if applicable.

Send the Report via Email

While JIRA is the preferred reporting method, you may also report vulnerabilities via email to security@mongodb.com⁵⁴.

You may encrypt email using MongoDB's public key at <https://docs.mongodb.org/10gen-security-gpg-key.asc>.

MongoDB, Inc. responds to vulnerability reports sent via email with a response email that contains a reference number for a JIRA ticket posted to the [SECURITY](#)⁵⁵ project.

Evaluation of a Vulnerability Report

MongoDB, Inc. validates all submitted vulnerabilities and uses Jira to track all communications regarding a vulnerability, including requests for clarification or additional information. If needed, MongoDB representatives set up a conference call to exchange information regarding the vulnerability.

Disclosure

MongoDB, Inc. requests that you do *not* publicly disclose any information regarding the vulnerability or exploit the issue until it has had the opportunity to analyze the vulnerability, to respond to the notification, and to notify key users, customers, and partners.

The amount of time required to validate a reported vulnerability depends on the complexity and severity of the issue. MongoDB, Inc. takes all required vulnerabilities very seriously and will always ensure that there is a clear and open channel of communication with the reporter.

After validating an issue, MongoDB, Inc. coordinates public disclosure of the issue with the reporter in a mutually agreed timeframe and format. If required or requested, the reporter of a vulnerability will receive credit in the published security bulletin.

⁵⁴security@mongodb.com

⁵⁵<https://jira.mongodb.org/browse/SECURITY>

Security Reference

On this page

- [Security Methods in the mongo Shell \(page 113\)](#)
- [Security Reference Documentation \(page 114\)](#)

The following lists the security related methods available in the mongo shell as well as additional *security reference material* (page 114).

8.1 Security Methods in the mongo Shell

8.1.1 User Management and Authentication Methods

Name	Description
<code>db.auth()</code>	Authenticates a user to a database.
<code>db.createUser()</code>	Creates a new user.
<code>db.updateUser()</code>	Updates user data.
<code>db.changeUserPassword()</code>	Changes an existing user's password.
<code>db.removeUser()</code>	Deprecated. Removes a user from a database.
<code>db.dropAllUsers()</code>	Deletes all users associated with a database.
<code>db.dropUser()</code>	Removes a single user.
<code>db.grantRolesToUser()</code>	Grants a role and its privileges to a user.
<code>db.revokeRolesFromUser()</code>	Removes a role from a user.
<code>db.getUser()</code>	Returns information about the specified user.
<code>db.getUsers()</code>	Returns information about all users associated with a database.

8.1.2 Role Management Methods

Name	Description
<code>db.createRole()</code>	Creates a role and specifies its privileges.
<code>db.updateRole()</code>	Updates a user-defined role.
<code>db.dropRole()</code>	Deletes a user-defined role.
<code>db.dropAllRoles()</code>	Deletes all user-defined roles associated with a database.
<code>db.grantPrivilegesToRole()</code>	Assigns privileges to a user-defined role.
<code>db.revokePrivilegesFromRole()</code>	Removes the specified privileges from a user-defined role.
<code>db.grantRolesToRole()</code>	Specifies roles from which a user-defined role inherits privileges.
<code>db.revokeRolesFromRole()</code>	Removes inherited roles from a role.
<code>db.getRole()</code>	Returns information for the specified role.
<code>db.getRoles()</code>	Returns information for all the user-defined roles in a database.

8.2 Security Reference Documentation

Built-In Roles (page 114) Reference on MongoDB provided roles and corresponding access.

system.roles Collection (page 123) Describes the content of the collection that stores user-defined roles.

system.users Collection (page 125) Describes the content of the collection that stores users' credentials and role assignments.

Resource Document (page 127) Describes the resource document for roles.

Privilege Actions (page 129) List of the actions available for privileges.

System Event Audit Messages (page 134) Reference on system event audit messages.

8.2.1 Built-In Roles

On this page

- Database User Roles (page 115)
- Database Administration Roles (page 116)
- Cluster Administration Roles (page 117)
- Backup and Restoration Roles (page 120)
- All-Database Roles (page 121)
- Superuser Roles (page 122)
- Internal Role (page 122)

MongoDB grants access to data and commands through *role-based authorization* (page 23) and provides built-in roles that provide the different levels of access commonly needed in a database system. You can additionally create *user-defined roles* (page 27).

A role grants privileges to perform sets of *actions* (page 129) on defined *resources* (page 127). A given role applies to the database on which it is defined and can grant access down to a collection level of granularity.

Each of MongoDB's built-in roles defines access at the database level for all *non-system* collections in the role's database and at the collection level for all `system` collections.

MongoDB provides the built-in *database user* (page 115) and *database administration* (page 116) roles on *every* database. MongoDB provides all other built-in roles only on the `admin` database.

This section describes the privileges for each built-in role. You can also view the privileges for a built-in role at any time by issuing the `rolesInfo` command with the `showPrivileges` and `showBuiltinRoles` fields both set to `true`.

Database User Roles

Every database includes the following client roles:

read

Provides the ability to read data on all *non-system* collections and on the following system collections: `system.indexes`, `system.js`, and `system.namespaces` collections. The role provides read access by granting the following *actions* (page 129):

- `collStats` (page 133)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `find` (page 129)
- `killCursors` (page 130)
- `listIndexes` (page 134)
- `listCollections` (page 134)

readWrite

Provides all the privileges of the `read` (page 115) role plus ability to modify data on all *non-system* collections and the `system.js` collection. The role provides the following actions on those collections:

- `collStats` (page 133)
- `convertToCapped` (page 132)
- `createCollection` (page 130)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `dropCollection` (page 130)
- `createIndex` (page 130)
- `dropIndex` (page 133)
- `emptycapped` (page 130)
- `find` (page 129)
- `insert` (page 129)
- `killCursors` (page 130)
- `listIndexes` (page 134)
- `listCollections` (page 134)
- `remove` (page 129)
- `renameCollectionSameDB` (page 133)
- `update` (page 129)

Database Administration Roles

Every database includes the following database administration roles:

dbAdmin

Provides the following *actions* (page 129) on the database's `system.indexes`, `system.namespaces`, and `system.profile` collections:

- `collStats` (page 133)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `find` (page 129)
- `killCursors` (page 130)
- `listIndexes` (page 134)
- `listCollections` (page 134)
- `dropCollection` (page 130) and `createCollection` (page 130) on `system.profile` *only*

Changed in version 2.6.4: `dbAdmin` (page 116) added the `createCollection` (page 130) for the `system.profile` collection. Previous versions only had the `dropCollection` (page 130) on the `system.profile` collection.

Provides the following actions on all *non-system* collections. This role *does not* include full read access on non-system collections:

- `bypassDocumentValidation` (page 129)
- `collMod` (page 132)
- `collStats` (page 133)
- `compact` (page 132)
- `convertToCapped` (page 132)
- `createCollection` (page 130)
- `createIndex` (page 130)
- `dbStats` (page 133)
- `dropCollection` (page 130)
- `dropDatabase` (page 132)
- `dropIndex` (page 133)
- `enableProfiler` (page 130)
- `indexStats` (page 134)
- `reIndex` (page 133)
- `renameCollectionSameDB` (page 133)
- `repairDatabase` (page 133)
- `storageDetails` (page 131)
- `validate` (page 134)

dbOwner

The database owner can perform any administrative action on the database. This role combines the privileges granted by the `readWrite` (page 115), `dbAdmin` (page 116) and `userAdmin` (page 117) roles.

userAdmin

Provides the ability to create and modify roles and users on the current database. This role also indirectly provides *superuser* (page 122) access to either the database or, if scoped to the `admin` database, the cluster. The `userAdmin` (page 117) role allows users to grant any user any privilege, including themselves.

The `userAdmin` (page 117) role explicitly provides the following actions:

- `changeCustomData` (page 130)
- `changePassword` (page 130)
- `createRole` (page 130)
- `createUser` (page 130)
- `dropRole` (page 130)
- `dropUser` (page 130)
- `grantRole` (page 130)
- `revokeRole` (page 130)
- `viewRole` (page 131)
- `viewUser` (page 131)

Cluster Administration Roles

The `admin` database includes the following roles for administering the whole system rather than just a single database. These roles include but are not limited to *replica set* and *sharded cluster* administrative functions.

clusterAdmin

Provides the greatest cluster-management access. This role combines the privileges granted by the `clusterManager` (page 117), `clusterMonitor` (page 118), and `hostManager` (page 119) roles. Additionally, the role provides the `dropDatabase` (page 132) action.

clusterManager

Provides management and monitoring actions on the cluster. A user with this role can access the `config` and `local` databases, which are used in sharding and replication, respectively.

Provides the following actions on the cluster as a whole:

- `addShard` (page 132)
- `applicationMessage` (page 132)
- `cleanupOrphaned` (page 131)
- `flushRouterConfig` (page 132)
- `listShards` (page 132)
- `removeShard` (page 132)
- `replSetConfigure` (page 131)
- `replSetGetStatus` (page 131)
- `replSetStateChange` (page 131)
- `resync` (page 131)

Provides the following actions on *all* databases in the cluster:

- `enableSharding` (page 132)
- `moveChunk` (page 132)
- `splitChunk` (page 132)
- `splitVector` (page 132)

On the `config` database, provides the following actions on the `settings` collection:

- `insert` (page 129)
- `remove` (page 129)
- `update` (page 129)

On the `config` database, provides the following actions on all configuration collections and on the `system.indexes`, `system.js`, and `system.namespaces` collections:

- `collStats` (page 133)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `find` (page 129)
- `killCursors` (page 130)

On the `local` database, provides the following actions on the `replset` collection:

- `collStats` (page 133)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `find` (page 129)
- `killCursors` (page 130)

clusterMonitor

Provides read-only access to monitoring tools, such as the [MongoDB Cloud Manager](https://cloud.mongodb.com/?jmp=docs)¹ and [Ops Manager](https://docs.opsmanager.mongodb.com/current/)² monitoring agent.

Provides the following actions on the cluster as a whole:

- `connPoolStats` (page 133)
- `cursorInfo` (page 133)
- `getCmdLineOpts` (page 134)
- `getLog` (page 134)
- `getParameter` (page 133)
- `getShardMap` (page 132)
- `hostInfo` (page 133)
- `inprog` (page 131)
- `listDatabases` (page 134)
- `listShards` (page 132)

¹<https://cloud.mongodb.com/?jmp=docs>

²<https://docs.opsmanager.mongodb.com/current/>

- [netstat](#) (page 134)
- [replSetGetStatus](#) (page 131)
- [serverStatus](#) (page 134)
- [shardingState](#) (page 132)
- [top](#) (page 134)

Provides the following actions on *all* databases in the cluster:

- [collStats](#) (page 133)
- [dbStats](#) (page 133)
- [getShardVersion](#) (page 132)

Provides the [find](#) (page 129) action on all `system.profile` collections in the cluster.

Provides the following actions on the `config` database's configuration collections and `system.indexes`, `system.js`, and `system.namespaces` collections:

- [collStats](#) (page 133)
- [dbHash](#) (page 133)
- [dbStats](#) (page 133)
- [find](#) (page 129)
- [killCursors](#) (page 130)

hostManager

Provides the ability to monitor and manage servers.

Provides the following actions on the cluster as a whole:

- [applicationMessage](#) (page 132)
- [closeAllDatabases](#) (page 132)
- [connPoolSync](#) (page 132)
- [cpuProfiler](#) (page 131)
- [diagLogging](#) (page 133)
- [flushRouterConfig](#) (page 132)
- [fsync](#) (page 133)
- [invalidateUserCache](#) (page 131)
- [killop](#) (page 131)
- [logRotate](#) (page 133)
- [resync](#) (page 131)
- [setParameter](#) (page 133)
- [shutdown](#) (page 133)
- [touch](#) (page 133)
- [unlock](#) (page 130)

Provides the following actions on *all* databases in the cluster:

- [killCursors](#) (page 130)

- [repairDatabase](#) (page 133)

Backup and Restoration Roles

The admin database includes the following roles for backing up and restoring data:

backup

Provides minimal privileges needed for backing up data. This role provides sufficient privileges to use the [MongoDB Cloud Manager](#)³ backup agent, [Ops Manager](#)⁴ backup agent, or to use `mongodump` to back up an entire mongod instance.

Provides the following *actions* (page 129) on the `mms.backup` collection in the admin database:

- [insert](#) (page 129)
- [update](#) (page 129)

Provides the [listDatabases](#) (page 134) action on the cluster as a whole.

Provides the [listCollections](#) (page 134) action on all databases.

Provides the [listIndexes](#) (page 134) action for all collections.

Provides the [bypassDocumentValidation](#) (page 129) action for collections that have document validation.

Provides the [find](#) (page 129) action on the following:

- all *non*-system collections in the cluster
- all the following system collections in the cluster: `system.indexes`, `system.namespaces`, and `system.js`
- the `admin.system.users` and `admin.system.roles` collections
- legacy `system.users` collections from versions of MongoDB prior to 2.6

Changed in version 3.2.1: The [backup](#) (page 120) role provides additional privileges to back up the `system.profile` collections that exist when running with *database profiling*. Previously, users required an additional `read` access on this collection.

restore

Provides privileges needed to restore data from backups that do not include `system.profile` collection data. This role is sufficient when restoring data with `mongorestore` without the `--oplogReplay` option.

- If the backup data includes `system.profile` collection data and the target database does not contain the `system.profile` collection, `mongorestore` attempts to create the collection even though the program does not actually restore `system.profile` documents. As such, the user requires additional privileges to perform [createCollection](#) (page 130) and [convertToCapped](#) (page 132) actions on the `system.profile` collection for a database.

The built-in roles [dbAdmin](#) (page 116) and [dbAdminAnyDatabase](#) (page 122) provide the additional privileges.

- If running `mongorestore` with `--oplogReplay`, the [restore](#) (page 120) role is insufficient to replay the oplog. To replay the oplog, create a *user-defined role* (page 72) that has [anyAction](#) (page 134) on *anyResource* (page 129) and grant only to users who must run `mongorestore` with `--oplogReplay`.

³<https://cloud.mongodb.com/?jmp=docs>

⁴<https://docs.opsmanager.mongodb.com/current/>

Provides the following actions on all *non*-system collections and `system.js` collections in the cluster; on the `admin.system.users` and `admin.system.roles` collections in the `admin` database; and on legacy `system.users` collections from versions of MongoDB prior to 2.6:

- `collMod` (page 132)
- `createCollection` (page 130)
- `createIndex` (page 130)
- `dropCollection` (page 130)
- `insert` (page 129)

Provides the `listCollections` (page 134) action on all databases.

Provides the following *additional* actions on `admin.system.users` and legacy `system.users` collections:

- `find` (page 129)
- `remove` (page 129)
- `update` (page 129)

Provides the `find` (page 129) action on all the `system.namespaces` collections in the cluster.

Although, `restore` (page 120) includes the ability to modify the documents in the `admin.system.users` collection using normal modification operations, *only* modify these data using the *user management methods*.

All-Database Roles

The `admin` database provides the following roles that apply to all databases in a `mongod` instance and are roughly equivalent to their single-database equivalents:

readAnyDatabase

Provides the same read-only permissions as `read` (page 115), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 134) action on the cluster as a whole.

readWriteAnyDatabase

Provides the same read and write permissions as `readWrite` (page 115), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 134) action on the cluster as a whole.

userAdminAnyDatabase

Provides the same access to user administration operations as `userAdmin` (page 117), except it applies to *all* databases in the cluster. The role also provides the following actions on the cluster as a whole:

- `authSchemaUpgrade` (page 131)
- `invalidateUserCache` (page 131)
- `listDatabases` (page 134)

The role also provides the following actions on the `admin.system.users` and `admin.system.roles` collections on the `admin` database, and on legacy `system.users` collections from versions of MongoDB prior to 2.6:

- `collStats` (page 133)
- `dbHash` (page 133)
- `dbStats` (page 133)
- `find` (page 129)

- `killCursors` (page 130)
- `planCacheRead` (page 131)

Changed in version 2.6.4: `userAdminAnyDatabase` (page 121) added the following permissions on the `admin.system.users` and `admin.system.roles` collections:

- `createIndex` (page 130)
- `dropIndex` (page 133)

The `userAdminAnyDatabase` (page 121) role does not restrict the permissions that a user can grant. As a result, `userAdminAnyDatabase` (page 121) users can grant themselves privileges in excess of their current privileges and even can grant themselves *all privileges*, even though the role does not explicitly authorize privileges beyond user administration. This role is effectively a MongoDB system *superuser* (page 122).

dbAdminAnyDatabase

Provides the same access to database administration operations as `dbAdmin` (page 116), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 134) action on the cluster as a whole.

Superuser Roles

Several roles provide either indirect or direct system-wide superuser access.

The following roles provide the ability to assign any user any privilege on any database, which means that users with one of these roles can assign *themselves* any privilege on any database:

- `dbOwner` (page 116) role, when scoped to the `admin` database
- `userAdmin` (page 117) role, when scoped to the `admin` database
- `userAdminAnyDatabase` (page 121) role

The following role provides full privileges on all resources:

root

Provides access to the operations and all the resources of the `readWriteAnyDatabase` (page 121), `dbAdminAnyDatabase` (page 122), `userAdminAnyDatabase` (page 121), `clusterAdmin` (page 117) roles, `restore` (page 120) *combined*.

Changed in version 3.0.7: The `root` (page 122) has `validate` (page 134) action on `system.` collections. Previously, `root` (page 122) does **not** include any access to collections that begin with the `system.` prefix.

The `root` (page 122) role includes privileges from the `restore` (page 120) role.

Internal Role

__system

MongoDB assigns this role to user objects that represent cluster members, such as replica set members and `mongos` instances. The role entitles its holder to take any action against any object in the database.

Do not assign this role to user objects representing applications or human administrators, other than in exceptional circumstances.

If you need access to all actions on all resources, for example to run `applyOps` commands, do not assign this role. Instead, *create a user-defined role* (page 72) that grants `anyAction` (page 134) on `anyResource` (page 129) and ensure that only the users who need access to these operations have this access.

8.2.2 `system.roles` Collection

New in version 2.6.

On this page

- `system.roles` Schema (page 123)
- Examples (page 124)

The `system.roles` collection in the `admin` database stores the user-defined roles. To create and manage these user-defined roles, MongoDB provides *role management commands*.

`system.roles` Schema

The documents in the `system.roles` collection have the following schema:

```
{
  _id: <system-defined id>,
  role: "<role name>",
  db: "<database>",
  privileges:
    [
      {
        resource: { <resource> },
        actions: [ "<action>", ... ]
      },
      ...
    ],
  roles:
    [
      { role: "<role name>", db: "<database>" },
      ...
    ]
}
```

A `system.roles` document has the following fields:

`admin.system.roles.role`

The `role` (page 123) field is a string that specifies the name of the role.

`admin.system.roles.db`

The `db` (page 123) field is a string that specifies the database to which the role belongs. MongoDB uniquely identifies each role by the pairing of its name (i.e. `role` (page 123)) and its database.

`admin.system.roles.privileges`

The `privileges` (page 123) array contains the privilege documents that define the *privileges* (page 23) for the role.

A privilege document has the following syntax:

```
{
  resource: { <resource> },
  actions: [ "<action>", ... ]
}
```

Each privilege document has the following fields:

`admin.system.roles.privileges[n].resource`

A document that specifies the resources upon which the privilege `actions` (page 124) apply. The document has one of the following form:

```
{ db: <database>, collection: <collection> }
```

or

```
{ cluster : true }
```

See *Resource Document* (page 127) for more details.

`admin.system.roles.privileges[n].actions`

An array of actions permitted on the resource. For a list of actions, see *Privilege Actions* (page 129).

`admin.system.roles.roles`

The `roles` (page 124) array contains role documents that specify the roles from which this role *inherits* (page 24) privileges.

A role document has the following syntax:

```
{ role: "<role name>", db: "<database>" }
```

A role document has the following fields:

`admin.system.roles.roles[n].role`

The name of the role. A role can be a *built-in role* (page 114) provided by MongoDB or a *user-defined role* (page 27).

`admin.system.roles.roles[n].db`

The name of the database where the role is defined.

Examples

Consider the following sample documents found in `system.roles` collection of the `admin` database.

A User-Defined Role Specifies Privileges

The following is a sample document for a user-defined role `appUser` defined for the `myApp` database:

```
{
  _id: "myApp.appUser",
  role: "appUser",
  db: "myApp",
  privileges: [
    { resource: { db: "myApp" , collection: "" },
      actions: [ "find", "createCollection", "dbStats", "collStats" ] },
    { resource: { db: "myApp", collection: "logs" },
      actions: [ "insert" ] },
    { resource: { db: "myApp", collection: "data" },
      actions: [ "insert", "update", "remove", "compact" ] },
    { resource: { db: "myApp", collection: "system.js" },
      actions: [ "find" ] },
  ],
  roles: []
}
```

The `privileges` array lists the five privileges that the `appUser` role specifies:

- The first privilege permits its actions ("find", "createCollection", "dbStats", "collStats") on all the collections in the myApp database *excluding* its system collections. See *Specify a Database as Resource* (page 128).
- The next two privileges permits *additional* actions on specific collections, logs and data, in the myApp database. See *Specify a Collection of a Database as Resource* (page 127).
- The last privilege permits actions on one system collections in the myApp database. While the first privilege gives database-wide permission for the find action, the action does not apply to myApp's system collections. To give access to a system collection, a privilege must explicitly specify the collection. See *Resource Document* (page 127).

As indicated by the empty roles array, appUser inherits no additional privileges from other roles.

User-Defined Role Inherits from Other Roles

The following is a sample document for a user-defined role appAdmin defined for the myApp database: The document shows that the appAdmin role specifies privileges as well as inherits privileges from other roles:

```
{
  _id: "myApp.appAdmin",
  role: "appAdmin",
  db: "myApp",
  privileges: [
    {
      resource: { db: "myApp", collection: "" },
      actions: [ "insert", "dbStats", "collStats", "compact", "repairDatabase" ]
    }
  ],
  roles: [
    { role: "appUser", db: "myApp" }
  ]
}
```

The privileges array lists the privileges that the appAdmin role specifies. This role has a single privilege that permits its actions ("insert", "dbStats", "collStats", "compact", "repairDatabase") on all the collections in the myApp database *excluding* its system collections. See *Specify a Database as Resource* (page 128).

The roles array lists the roles, identified by the role names and databases, from which the role appAdmin inherits privileges.

8.2.3 system.users Collection

Changed in version 2.6.

On this page

- [system.users Schema](#) (page 126)
- [Example](#) (page 126)

The system.users collection in the admin database stores user *authentication* (page 7) and *authorization* (page 23) information. To manage data in this collection, MongoDB provides *user management commands*.

system.users Schema

The documents in the `system.users` collection have the following schema:

```
{
  _id: <system defined id>,
  user: "<name>",
  db: "<database>",
  credentials: { <authentication credentials> },
  roles: [
    { role: "<role name>", db: "<database>" },
    ...
  ],
  customData: <custom information>
}
```

Each `system.users` document has the following fields:

admin.system.users.user

The `user` (page 126) field is a string that identifies the user. A user exists in the context of a single logical database but can have access to other databases through roles specified in the `roles` (page 126) array.

admin.system.users.db

The `db` (page 126) field specifies the database associated with the user. The user's privileges are not necessarily limited to this database. The user can have privileges in additional databases through the `roles` (page 126) array.

admin.system.users.credentials

The `credentials` (page 126) field contains the user's authentication information. For users with externally stored authentication credentials, such as users that use *Kerberos* (page 55) or x.509 certificates for authentication, the `system.users` document for that user does not contain the `credentials` (page 126) field.

admin.system.users.roles

The `roles` (page 126) array contains role documents that specify the roles granted to the user. The array contains both *built-in roles* (page 114) and *user-defined role* (page 27).

A role document has the following syntax:

```
{ role: "<role name>", db: "<database>" }
```

A role document has the following fields:

admin.system.users.roles[n].role

The name of a role. A role can be a *built-in role* (page 114) provided by MongoDB or a *custom user-defined role* (page 27).

admin.system.users.roles[n].db

The name of the database where role is defined.

When specifying a role using the *role management* or *user management* commands, you can specify the role name alone (e.g. "readWrite") if the role that exists on the database on which the command is run.

admin.system.users.customData

The `customData` (page 126) field contains optional custom information about the user.

Example

Changed in version 3.0.0.

Consider the following document in the `system.users` collection:

```
{
  _id : "home.Kari",
  user : "Kari",
  db : "home",
  credentials : {
    "SCRAM-SHA-1" : {
      "iterationCount" : 10000,
      "salt" : "nkHYXEZTTYmn+hrY994y1Q==",
      "storedKey" : "wxWGN3ElQ25WbPjACeXdUmN4nNo=",
      "serverKey" : "h7vBq5tACT/BtrIElY2QTm+pQzM="
    }
  },
  roles : [
    { role: "read", db: "home" },
    { role: "readWrite", db: "test" },
    { role: "appUser", db: "myApp" }
  ],
  customData : { zipCode: "64157" }
}
```

The document shows that a user Kari is associated with the home database. Kari has the `read` (page 115) role in the home database, the `readWrite` (page 115) role in the test database, and the `appUser` role in the myApp database.

8.2.4 Resource Document

On this page

- [Database and/or Collection Resource](#) (page 127)
- [Cluster Resource](#) (page 128)
- [anyResource](#) (page 129)

The resource document specifies the resources upon which a privilege permits actions.

Database and/or Collection Resource

To specify databases and/or collections, use the following syntax:

```
{ db: <database>, collection: <collection> }
```

Specify a Collection of a Database as Resource

If the resource document species both the `db` and `collection` fields as non-empty strings, the resource is the specified collection in the specified database. For example, the following document specifies a resource of the `inventory` collection in the `products` database:

```
{ db: "products", collection: "inventory" }
```

For a user-defined role scoped for a non-admin database, the resource specification for its privileges must specify the same database as the role. User-defined roles scoped for the `admin` database can specify other databases.

Specify a Database as Resource

If only the `collection` field is an empty string (""), the resource is the specified database, excluding the system collections. For example, the following resource document specifies the resource of the `test` database, excluding the system collections:

```
{ db: "test", collection: "" }
```

For a user-defined role scoped for a non-admin database, the resource specification for its privileges must specify the same database as the role. User-defined roles scoped for the `admin` database can specify other databases.

Note: When you specify a database as the resource, system collections are excluded, unless you name them explicitly, as in the following:

```
{ db: "test", collection: "system.js" }
```

System collections include but are not limited to the following:

- `<database>.system.profile`
- `<database>.system.js`
- *system.users Collection* (page 125) in the `admin` database
- *system.roles Collection* (page 123) in the `admin` database

Specify Collections Across Databases as Resource

If only the `db` field is an empty string (""), the resource is all collections with the specified name across all databases. For example, the following document specifies the resource of all the `accounts` collections across all the databases:

```
{ db: "", collection: "accounts" }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

Specify All Non-System Collections in All Databases

If both the `db` and `collection` fields are empty strings (""), the resource is all collections, excluding the system collections, in all the databases:

```
{ db: "", collection: "" }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

Cluster Resource

To specify the cluster as the resource, use the following syntax:

```
{ cluster : true }
```

Use the `cluster` resource for actions that affect the state of the system rather than act on specific set of databases or collections. Examples of such actions are `shutdown`, `replSetReconfig`, and `addShard`. For example, the following document grants the action `shutdown` on the `cluster`.

```
{ resource: { cluster : true }, actions: [ "shutdown" ] }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

anyResource

The internal resource `anyResource` gives access to every resource in the system and is intended for internal use. **Do not** use this resource, other than in exceptional circumstances. The syntax for this resource is `{ anyResource: true }`.

8.2.5 Privilege Actions

New in version 2.6.

On this page

- [Query and Write Actions](#) (page 129)
- [Database Management Actions](#) (page 130)
- [Deployment Management Actions](#) (page 131)
- [Replication Actions](#) (page 131)
- [Sharding Actions](#) (page 132)
- [Server Administration Actions](#) (page 132)
- [Diagnostic Actions](#) (page 133)
- [Internal Actions](#) (page 134)

Privilege actions define the operations a user can perform on a *resource* (page 127). A MongoDB *privilege* (page 23) comprises a *resource* (page 127) and the permitted actions. This page lists available actions grouped by common purpose.

MongoDB provides built-in roles with pre-defined pairings of resources and permitted actions. For lists of the actions granted, see *Built-In Roles* (page 114). To define custom roles, see *Create a User-Defined Role* (page 72).

Query and Write Actions

find

User can perform the `db.collection.find()` method. Apply this action to database or collection resources.

insert

User can perform the `insert` command. Apply this action to database or collection resources.

remove

User can perform the `db.collection.remove()` method. Apply this action to database or collection resources.

update

User can perform the `update` command. Apply this action to database or collection resources.

bypassDocumentValidation

New in version 3.2.

User can bypass document validation on commands that support the `bypassDocumentValidation` option. For a list of commands that support the `bypassDocumentValidation` option, see *3.2-rel-notes-document-validation*. Apply this action to database or collection resources.

Database Management Actions

changeCustomData

User can change the custom information of any user in the given database. Apply this action to database resources.

changeOwnCustomData

Users can change their own custom information. Apply this action to database resources. See also *Change Your Password and Custom Data* (page 77).

changeOwnPassword

Users can change their own passwords. Apply this action to database resources. See also *Change Your Password and Custom Data* (page 77).

changePassword

User can change the password of any user in the given database. Apply this action to database resources.

createCollection

User can perform the `db.createCollection()` method. Apply this action to database or collection resources.

createIndex

Provides access to the `db.collection.createIndex()` method and the `createIndexes` command. Apply this action to database or collection resources.

createRole

User can create new roles in the given database. Apply this action to database resources.

createUser

User can create new users in the given database. Apply this action to database resources.

dropCollection

User can perform the `db.collection.drop()` method. Apply this action to database or collection resources.

dropRole

User can delete any role from the given database. Apply this action to database resources.

dropUser

User can remove any user from the given database. Apply this action to database resources.

emptycapped

User can perform the `emptycapped` command. Apply this action to database or collection resources.

enableProfiler

User can perform the `db.setProfilingLevel()` method. Apply this action to database resources.

grantRole

User can grant any role in the database to any user from any database in the system. Apply this action to database resources.

killCursors

User can kill cursors on the target collection.

revokeRole

User can remove any role from any user from any database in the system. Apply this action to database resources.

unlock

User can perform the `db.fsyncUnlock()` method. Apply this action to the `cluster` resource.

viewRole

User can view information about any role in the given database. Apply this action to database resources.

viewUser

User can view the information of any user in the given database. Apply this action to database resources.

Deployment Management Actions**authSchemaUpgrade**

User can perform the `authSchemaUpgrade` command. Apply this action to the `cluster` resource.

cleanupOrphaned

User can perform the `cleanupOrphaned` command. Apply this action to the `cluster` resource.

cpuProfiler

User can enable and use the CPU profiler. Apply this action to the `cluster` resource.

inprog

User can use the `db.currentOp()` method to return pending and active operations. Apply this action to the `cluster` resource.

invalidateUserCache

Provides access to the `invalidateUserCache` command. Apply this action to the `cluster` resource.

killop

User can perform the `db.killOp()` method. Apply this action to the `cluster` resource.

planCacheRead

User can perform the `planCacheListPlans` and `planCacheListQueryShapes` commands and the `PlanCache.getPlansByQuery()` and `PlanCache.listQueryShapes()` methods. Apply this action to database or collection resources.

planCacheWrite

User can perform the `planCacheClear` command and the `PlanCache.clear()` and `PlanCache.clearPlansByQuery()` methods. Apply this action to database or collection resources.

storageDetails

User can perform the `storageDetails` command. Apply this action to database or collection resources.

Replication Actions**appendOplogNote**

User can append notes to the oplog. Apply this action to the `cluster` resource.

replSetConfigure

User can configure a replica set. Apply this action to the `cluster` resource.

replSetGetStatus

User can perform the `replSetGetStatus` command. Apply this action to the `cluster` resource.

replSetHeartbeat

User can perform the `replSetHeartbeat` command. Apply this action to the `cluster` resource.

replSetStateChange

User can change the state of a replica set through the `replSetFreeze`, `replSetMaintenance`, `replSetStepDown`, and `replSetSyncFrom` commands. Apply this action to the `cluster` resource.

resync

User can perform the `resync` command. Apply this action to the `cluster` resource.

Sharding Actions

addShard

User can perform the `addShard` command. Apply this action to the `cluster` resource.

enableSharding

User can enable sharding on a database using the `enableSharding` command and can shard a collection using the `shardCollection` command. Apply this action to database or collection resources.

flushRouterConfig

User can perform the `flushRouterConfig` command. Apply this action to the `cluster` resource.

getShardMap

User can perform the `getShardMap` command. Apply this action to the `cluster` resource.

getShardVersion

User can perform the `getShardVersion` command. Apply this action to database resources.

listShards

User can perform the `listShards` command. Apply this action to the `cluster` resource.

moveChunk

User can perform the `moveChunk` command. In addition, user can perform the `movePrimary` command provided that the privilege is applied to an appropriate database resource. Apply this action to database or collection resources.

removeShard

User can perform the `removeShard` command. Apply this action to the `cluster` resource.

shardingState

User can perform the `shardingState` command. Apply this action to the `cluster` resource.

splitChunk

User can perform the `splitChunk` command. Apply this action to database or collection resources.

splitVector

User can perform the `splitVector` command. Apply this action to database or collection resources.

Server Administration Actions

applicationMessage

User can perform the `logApplicationMessage` command. Apply this action to the `cluster` resource.

closeAllDatabases

User can perform the `closeAllDatabases` command. Apply this action to the `cluster` resource.

collMod

User can perform the `collMod` command. Apply this action to database or collection resources.

compact

User can perform the `compact` command. Apply this action to database or collection resources.

connPoolSync

User can perform the `connPoolSync` command. Apply this action to the `cluster` resource.

convertToCapped

User can perform the `convertToCapped` command. Apply this action to database or collection resources.

dropDatabase

User can perform the `dropDatabase` command. Apply this action to database resources.

dropIndex

User can perform the `dropIndexes` command. Apply this action to database or collection resources.

fsync

User can perform the `fsync` command. Apply this action to the `cluster` resource.

getParameter

User can perform the `getParameter` command. Apply this action to the `cluster` resource.

hostInfo

Provides information about the server the MongoDB instance runs on. Apply this action to the `cluster` resource.

logRotate

User can perform the `logRotate` command. Apply this action to the `cluster` resource.

reIndex

User can perform the `reIndex` command. Apply this action to database or collection resources.

renameCollectionSameDB

Allows the user to rename collections on the current database using the `renameCollection` command. Apply this action to database resources.

Additionally, the user must either *have* `find` (page 129) on the source collection or *not have* `find` (page 129) on the destination collection.

If a collection with the new name already exists, the user must also have the `dropCollection` (page 130) action on the destination collection.

repairDatabase

User can perform the `repairDatabase` command. Apply this action to database resources.

setParameter

User can perform the `setParameter` command. Apply this action to the `cluster` resource.

shutdown

User can perform the `shutdown` command. Apply this action to the `cluster` resource.

touch

User can perform the `touch` command. Apply this action to the `cluster` resource.

Diagnostic Actions**collStats**

User can perform the `collStats` command. Apply this action to database or collection resources.

connPoolStats

User can perform the `connPoolStats` and `shardConnPoolStats` commands. Apply this action to the `cluster` resource.

cursorInfo

User can perform the `cursorInfo` command. Apply this action to the `cluster` resource.

dbHash

User can perform the `dbHash` command. Apply this action to database or collection resources.

dbStats

User can perform the `dbStats` command. Apply this action to database resources.

diagLogging

User can perform the `diagLogging` command. Apply this action to the `cluster` resource.

getCmdLineOpts

User can perform the `getCmdLineOpts` command. Apply this action to the `cluster` resource.

getLog

User can perform the `getLog` command. Apply this action to the `cluster` resource.

indexStats

User can perform the `indexStats` command. Apply this action to database or collection resources.

Changed in version 3.0: MongoDB 3.0 removes the `indexStats` command.

listDatabases

User can perform the `listDatabases` command. Apply this action to the `cluster` resource.

listCollections

User can perform the `listCollections` command. Apply this action to database resources.

listIndexes

User can perform the `ListIndexes` command. Apply this action to database or collection resources.

netstat

User can perform the `netstat` command. Apply this action to the `cluster` resource.

serverStatus

User can perform the `serverStatus` command. Apply this action to the `cluster` resource.

validate

User can perform the `validate` command. Apply this action to database or collection resources.

top

User can perform the `top` command. Apply this action to the `cluster` resource.

Internal Actions

anyAction

Allows any action on a resource. **Do not** assign this action except for exceptional circumstances.

internal

Allows internal actions. **Do not** assign this action except for exceptional circumstances.

8.2.6 System Event Audit Messages

On this page

- [Audit Message](#) (page 135)
- [Audit Event Actions, Details, and Results](#) (page 135)

Note: Available only in [MongoDB Enterprise](#)⁵.

⁵<http://www.mongodb.com/products/mongodb-enterprise?jmp=docs>

Audit Message

The *event auditing feature* (page 33) can record events in JSON format. To configure auditing output, see *Configure Auditing* (page 102)

The recorded JSON messages have the following syntax:

```
{
  atype: <String>,
  ts : { "$date": <timestamp> },
  local: { ip: <String>, port: <int> },
  remote: { ip: <String>, port: <int> },
  users : [ { user: <String>, db: <String> }, ... ],
  roles: [ { role: <String>, db: <String> }, ... ],
  param: <document>,
  result: <int>
}
```

field string atype Action type. See *Audit Event Actions, Details, and Results* (page 135).

field document ts Document that contains the date and UTC time of the event, in ISO 8601 format.

field document local Document that contains the local `ip` address and the `port` number of the running instance.

field document remote Document that contains the remote `ip` address and the `port` number of the incoming connection associated with the event.

field array users Array of user identification documents. Because MongoDB allows a session to log in with different user per database, this array can have more than one user. Each document contains a `user` field for the username and a `db` field for the authentication database for that user.

field array roles Array of documents that specify the *roles* (page 23) granted to the user. Each document contains a `role` field for the name of the role and a `db` field for the database associated with the role.

field document param Specific details for the event. See *Audit Event Actions, Details, and Results* (page 135).

field integer result Error code. See *Audit Event Actions, Details, and Results* (page 135).

Audit Event Actions, Details, and Results

The following table lists for each `atype` or action type, the associated `param` details and the `result` values, if any.

atype	param	result
authenticate	<pre>{ user: <user name>, db: <database>, mechanism: <mechanism> }</pre>	0 - Success 18 - Authentication Failed
Continued on next page		

Table 8.1 – continued from previous page

atype	param	result
authCheck	<pre>{ command: <name>, ns: <database>.<collection>, args: <command object> }</pre> <p>ns field is optional. args field may be redacted.</p>	<p>0 - Success 13 - Unauthorized to perform the operation. By default, the auditing system logs only the authorization failures. To enable the system to log authorization successes, use the auditAuthorizationSuccess parameter.⁶</p>
createCollection (page 130)	<pre>{ ns: <database>.<collection> }</pre>	0 - Success
createDatabase	<pre>{ ns: <database> }</pre>	0 - Success
createIndex (page 130)	<pre>{ ns: <database>.<collection>, indexName: <index name>, indexSpec: <index specification> }</pre>	0 - Success
renameCollection	<pre>{ old: <database>.<collection>, new: <database>.<collection> }</pre>	0 - Success
dropCollection (page 130)	<pre>{ ns: <database>.<collection> }</pre>	0 - Success
dropDatabase (page 132)	<pre>{ ns: <database> }</pre>	0 - Success
dropIndex (page 133)	<pre>{ ns: <database>.<collection>, indexName: <index name> }</pre>	0 - Success

Continued on next page

⁶ Enabling auditAuthorizationSuccess degrades performance more than logging only the authorization failures.

Table 8.1 – continued from previous page

atype	param	result
<code>createUser</code> (page 130)	<pre>{ user: <user name>, db: <database>, customData: <document>, roles: [{ role: <role name>, db: <database> }, ...] }</pre> <p>The <code>customData</code> field is optional.</p>	0 - Success
<code>dropUser</code> (page 130)	<pre>{ user: <user name>, db: <database> }</pre>	0 - Success
<code>dropAllUsersFromDatabase</code>	<pre>{ db: <database> }</pre>	0 - Success
<code>updateUser</code>	<pre>{ user: <user name>, db: <database>, passwordChanged: <boolean>, customData: <document>, roles: [{ role: <role name>, db: <database> }, ...] }</pre> <p>The <code>customData</code> field is optional.</p>	0 - Success
<code>grantRolesToUser</code>	<pre>{ user: <user name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }</pre>	0 - Success

Continued on next page

Table 8.1 – continued from previous page

atype	param	result
revokeRolesFromUser	<pre>{ user: <user name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }</pre>	0 - Success
createRole (page 130)	<pre>{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...], privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] }</pre> <p>The roles and the privileges fields are optional. For details on the resource document, see <i>Resource Document</i> (page 127). For a list of actions, see <i>Privilege Actions</i> (page 129).</p>	0 - Success

Continued on next page

Table 8.1 – continued from previous page

atype	param	result
updateRole	<pre>{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...], privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] }</pre> <p>The roles and the privileges fields are optional. For details on the resource document, see <i>Resource Document</i> (page 127). For a list of actions, see <i>Privilege Actions</i> (page 129).</p>	0 - Success
dropRole (page 130)	<pre>{ role: <role name>, db: <database> }</pre>	0 - Success
dropAllRolesFromDatabase	{ db: <database> }	0 - Success
grantRolesToRole	<pre>{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }</pre>	0 - Success

Continued on next page

Table 8.1 – continued from previous page

atype	param	result
revokeRolesFromRole	<pre>{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }</pre>	0 - Success
grantPrivilegesToRole	<pre>{ role: <role name>, db: <database>, privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] }</pre> <p>For details on the resource document, see <i>Resource Document</i> (page 127). For a list of actions, see <i>Privilege Actions</i> (page 129).</p>	0 - Success
revokePrivilegesFromRole	<pre>{ role: <role name>, db: <database name>, privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] }</pre> <p>For details on the resource document, see <i>Resource Document</i> (page 127). For a list of actions, see <i>Privilege Actions</i> (page 129).</p>	0 - Success

Continued on next page

Table 8.1 – continued from previous page

atype	param	result
replSetReconfig	<pre>{ old: <configuration>, new: <configuration> }</pre> <p>Indicates membership change in the replica set. The <code>old</code> field is optional.</p>	0 - Success
enableSharding (page 132)	<pre>{ ns: <database> }</pre>	0 - Success
shardCollection	<pre>{ ns: <database>.<collection>, key: <shard key pattern>, options: { unique: <boolean> } }</pre>	0 - Success
addShard (page 132)	<pre>{ shard: <shard name>, connectionString: <hostname>:<port>, maxSize: <maxSize> }</pre> <p>When a shard is a replica set, the <code>connectionString</code> includes the replica set name and can include other members of the replica set.</p>	0 - Success
removeShard (page 132)	<pre>{ shard: <shard name> }</pre>	0 - Success
shutdown (page 133)	<pre>{ }</pre> <p>Indicates commencement of database shutdown.</p>	0 - Success
applicationMessage (page 132)	<pre>{ msg: <custom message string> }</pre> <p>See <code>logApplicationMessage</code>.</p>	0 - Success

Additional Resources

- [Making HIPAA Compliant MongoDB Applications¹](#)
- [Security Architecture White Paper²](#)
- [Webinar: Securing Your MongoDB Deployment³](#)

¹<https://www.mongodb.com/blog/post/making-hipaa-compliant-applications-mongodb?jmp=docs>

²<https://www.mongodb.com/lp/white-paper/mongodb-security-architecture?jmp=docs>

³<http://www.mongodb.com/presentations/webinar-securing-your-mongodb-deployment?jmp=docs>