**MongoDB Security**

MongoDB provides various features, such as such as authentication, access control, encryption etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Authentication | Authorization | TLS/SSL | Enterprise Only |
| authentication | Role Based Access Control | TLS/SSL(transport encryption) | Kerberos authentication |
| SCRAM | Enable Access Control | Configure mongod and mongos for TLS/SSL encryption | LDAP proxy authentication |
| x.509 | Manage Users and Roles | TLS/SSL configuration for clients | Encrypt at rest |
|  |  |  | Auditing |

**Security CheckList :**

MongoDB also provides the Security Checklist for a list of recommended actions to protect a MongoDB deployment.

Pre-production Checklist/Considerations

* **Enable Access Control and Enforce Authentication**

Enable access control and specify the authentication mechanism. You can use MongoDB’s SCRAM or x.509 authentication mechanism or integrate with your existing Kerberos/LDAP infrastructure. Authentication requires that all clients and servers provide valid credentials before they can connect to the system.

* **Configure Role-Based Access Control**

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

* **Encrypt Communication (TLS/SSL)**

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 deployment as well as between all applications and MongoDB.

Starting in version 4.0, MongoDB uses the native TLS/SSL OS libraries:

Windows Secure Channel (Schannel)

Linux/BSD OpenSSL

macOS Secure Transport

* **Encrypt and Protect Data**

Starting with MongoDB Enterprise 3.2, you can encrypt data in the storage layer with the WiredTiger storage engine’s native Encryption at Rest.

* **Limit Network Exposure**

Ensure that MongoDB runs in a trusted network environment and configure firewall or security groups to control inbound and outbound traffic for your MongoDB instances.

Disable direct SSH root access.

* **Audit System Activities**

Track access and changes to database configurations and data. MongoDB Enterprise 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. You can set up filters to record specific events, such as authentication events.

* **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.

* **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.

1. **Enable Access Control** :

Enabling access control on MongoDB deployment enforces authentication, it requires users to identify themselves. When accessing a MongoDB deployment that has access control enabled, users can only perform actions as determined by their roles.

In this documents, enables access control on a standalone mongod instance and uses the default authentication mechanism.

User Administrator :

With access control enabled, ensure you have a user with userAdmin or userAdminAnyDatabase role in the admin database. This user can administrate user and roles such as: create users, grant or revoke roles from users, and create or modify customs roles.

First add a user administrator to mongoDB instance without access control and then enable access control.

1. Start MongoDB without access control

mongod --port 27017 --dbpath E:\MongoData

1. Connect to the instance

mongo --port 27017

1. Create the user administrator

From the mongo shell, add a user with the userAdminAnyDatabase role in the admin database. Include additional roles as needed for this user. For example, the following creates the user adm in the admin database with the userAdminAnyDatabase role and the readWriteAnyDatabase role.

For example :

> use admin;

switched to db admin

> db.createUser(

... {

... user: "adm",

... pwd: "adm123",

... roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDat

abase" ]

... }

... );

Successfully added user: {

"user" : "adm",

"roles" : [

{

"role" : "userAdminAnyDatabase",

"db" : "admin"

},

"readWriteAnyDatabase"

]

}

1. Re-start the MongoDB instance with access control

Shutdown mongodb instance with the following command

db.adminCommand( { shutdown: 1 } )

1. Exit the mongo shell
2. Start the mongod with access control enabled

mongod --auth --port 27017 --dbpath E:\MongoData

If you start the mongod using a configuration file, add the security.authorization   
 configuration file setting:

security:

authorization: enabled

1. Connect and authenticate as the user administrator

mongo --port 27017 --authenticationDatabase "admin" -u "adm" –p

Enter your password when prompted

After connection you can also authenticate as below command.

db.auth("myUserAdmin", passwordPrompt());

1. Create additional users as needed for your deployment

use test

db.createUser(

{

user: "test",

pwd: passwordPrompt(), // or cleartext password

roles: [ { role: "readWrite", db: "test" },

{ role: "read", db: "mydb" } ]

}

)

1. Connect to the instance and authenticate as test

mongo --port 27017 -u "test" --authenticationDatabase "test" –p

Enter the password for the user when prompted

1. **Authentication**

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

Authentication Methods :

To authenticate using the mongo shell, either:

Use the mongo command-line authentication options (--username, --password, and --   
 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.

Authentication Mechanisms:

MongoDB supports a number of authentication mechanisms that clients can use to   
 verify their identity.

MongoDB supports multiple authentication mechanisms:

SCRAM (Default)

x.509 Certificate Authentication.

MongoDB Enterprise also supports the following mechanisms:

LDAP proxy authentication, and

Kerberos authentication.

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 to their respective  
 replica set or sharded cluster. See Internal/Membership Authentication for more   
 information.

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

1. **MongoDB Users and Roles**

Before you enable access control, you should create a user that can then create users and assign roles to them once access control is enabled.

This user-admin will then be used to create and maintain other users and roles, so needs to be assigned a suitable role to enable it to do so.

In case you do not create this user-admin, you will not be able to log in, or create new users and roles when access control is enabled.

**Localhost Exception :**

If you were to enable access control without previously having created at least a user-admin user, then you would not be able to log in.

The localhost exception is here to avoid this problem by allowing you to create the first user after enabling access control.

You can try to do this with following steps

If you were to enable access control without previously having created at least a user-admin user, then you would not be able to log in.

1-Enable access control

2-Connect to the localhost interface

3-Create the first user in the admin database that must have enough permissions to manage other users and roles.

**How to drop a User :**

>use admin

> db.dropUser('<userName>');

**Where are Users Stored :**

To check a user, you must change your context to the database in which the user was created, such as the Admin database.

> use admin

Then you can use either db.system.users.find(); or db.getUsers();

But, if you only want to ask for a specific user, use this command:

>db.getUser('<userName>');

**How to logout :**

>use admin

>db.logout();

**MongoDB Roles :**

As you already know, a role is a privilege granted to a user for making actions over resources.

The role defines the tasks that the role member is allowed to do and the resources where those tasks can be done.

MongoDB offers **built-in** roles for the most common purposes. But, also, allows us to create our own roles depending on our specific needs.

Each role is scoped to the database in which it has been created.

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 resource, and can inherit from roles in other databases as well as the admin database.

So, if you need to inherit from a role created in another database you will have to create your new role in the admin database.

**Where are Roles Stored :**

I have explained before that you can create roles in the admin database or in any other.

Therefore, if you want to check them, you must do it in the database where they were defined

>use '<dbName>';

to get all the roles of a database, use

db.system.roles.find() or db.getRoles()

If you only want to ask for a specific role, you will use this command:

> use '<dbName>'

> db.getRole('<roleName>');

**Built-in Roles :**

MongoDB have build in roles are as follows

1-Database User Roles

2-Database Administration Roles

3-Cluster Administration Roles

4-Backup and Restoration Roles

5-All-Database Roles

6-Superuser Roles

7-Internal Role

**1-Database User Roles:**

The roles available at the database level are:

read – Read data on all non-system collections

readWrite – Include all ‘read’ role privileges and the ability to write data on all non-system collections

**2-Database Administration Roles**

Every The database administration roles we can use are the following:

dbAdmin – Grant privileges to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics

userAdmin – Allows you to create and modify users and roles on the current database

dbOwner – This role combines the following:

readWrite

dbAdmin

userAdmin

**4-Cluster Administration Roles:**

Roles at the admin database for administering the whole system.

clusterMonitor – Provides read-only access to monitoring tools

clusterManager – For management and monitoring actions on the cluster

hostManager – To monitor and manage servers

clusterAdmin – Combines the other three roles plus dropDatabase action.

**4-Backup and Restoration Roles**:

This role belongs to the admin database.

backup – Provides the privileges needed for backing up data

restore – Provides the privileges needed to restore data from backups.

**5- All-Database Roles**  
These roles lie on the admin database and provide privileges which apply to all databases.

readAnyDatabase – The same as ‘read’ role but applies to all databases

readWriteAnyDatabase – The same as ‘readWrite’ role but applies to all databases

userAdminAnyDatabase – The same as ‘userAdmin’ role but applies to all databases

dbAdminAnyDatabase – The same as ‘dbAdmin’ role but applies to all databases.

**6- Superuser Roles**  
The following roles are not superuser roles directly but are able to assign any user any privilege on any database, also themselves.

userAdmin

dbOwner

userAdminAnyDatabase

The root role provides full privileges on all resources:

root

How to Check the Privileges of a Role

If you need to know the privileges (inherited from other roles or not) of a role, you can activate the ‘showPrivileges’ field:

> use '<dbName>'

> db.getRole('<roleName>', { showPrivileges : true });

**Summary of Roles**:

In my opinion, the following roles are typically the most useful:

userAdminAnyDatabase

clusterManager

clusterMonitor

backup

restore

dbAdmin

readWrite

read

**How to Grant a Role to a User**

You can grant a role as you create the user, or retrospectively.

The next command is valid for assigning a role at the same time you create the user:

> use '<dbName>'

> db.createUser(

{

user: "<userName>",

pwd: "<password>",

roles: [ { role: "<roleName>", db: "<dbName>" } ]

})

And you can use this command to do it later:

> use '<dbName>'

> db.grantRolesToUser(

'<userName>',

[ { role : '<roleName>', db : '<dbName>' }, '<roleName>', … ]

)

**How to Revoke a Role from a User**

> use '<dbName>'

> db.revokeRolesFromUser(

'<userName>',

[ { role : '<roleName>', db : '<dbname>' } | '<roleName>' ]

);

**User-defined Roles**

How to Create a Role

> use '<dbName>'

> db.createRole({

role: "<roleName>",

privileges: [

{ resource: { db : “<dbName>”,

collection : “<collectionName>” },

actions: [ '<actionName>' ]

}

],

roles: [ { role : '<fatherRoleName>', db : '<dbName>'} | '<roleName>' ]

})

**How to Drop a Role**

> use '<dbName>'

> db.dropRole('<roleName>')

How to Grant or Revoke Privileges to/from a Role

These commands grant or revoke privileges to a user-defined role.

> use '<dbName>'

> db.grantPrivilegesToRole(

'<roleName>',

[

{ resource : { db : '<dbName>', collection : '<collectionName'> },

actions : [ '<actionName>',... ]

},

...

]

)

> db.revokePrivilegesFromRole(

'<roleName>',

[

{ resource : { db : '<dbName>', collection : '<collectionName'> },

actions : [ '<actionName>',... ]

},

…

]

)

**How to Grant or Revoke Roles to/from a Role**

> use '<dbName>'

> db.grantRolesToRole(

'<roleName>',

[ { role : '<roleName>', db : '<dbName>' } | <roles> ]

)

> db.revokeRolesFromRole(

'<roleName>',

[ { role : '<roleName>', db : '<dbName>' } | <roles> ]

)

**How to Update a Role**

Be careful! As the documentation says: “An update to the privileges or roles array completely replaces the previous array’s values”.

> use '<dbName>'

> db.updateRole(

'<roleName>',

{

privileges : [

{

resource : { db : '<dbName>', collection : '<collectionName>' },

actions : [ '<actionName>' ]

},...

],

roles : [ { role : '<roleName>', db : '<dbName>' } | '<roleName>' ]

}

)