Enable Auth

On this page

Overview

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

For authentication, MongoDB supports various Authentication Mechanisms.

The following tutorial enables access control on a standalone `mongod` instance and uses the default authentication mechanism.

Replica sets and sharded clusters

Replica sets and sharded clusters require internal authentication between members when access control is enabled. For more details, please see Internal Authentication.

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.

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

Procedure

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

1 Start MongoDB without access control.

For example, the following starts a standalone `mongod` instance without access control.

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

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`.

3 Create the user administrator.

In the `admin` database, add a user with the `userAdminAnyDatabase` role. For example, the following creates the user `myUserAdmin` in the `admin` database:

> **NOTE:**
> The database where you create the user (in this example, `admin`) is the user's authentication database. Although the user would authenticate to this database, the user can have roles in other databases; i.e. the user's authentication database does not limit the user's privileges.

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

Disconnect the `mongo` shell.

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
```

Clients that connect to this instance must now authenticate themselves as a MongoDB user. Clients can only perform actions as determined by their assigned roles.

5 Connect and authenticate as the user administrator.

Using the `mongo` shell, you can:

- Connect with authentication by passing in user credentials, or
- Connect first withouth authentication, and then issue the `db.auth()` method to authenticate.

To authenticate during connection

Start a `mongo` shell with the `-u <username>`, `-p <password>`, and the `--authenticationDatabase <database>` command line options:

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

To authenticate after connecting

Connect the `mongo` shell to the `mongod`:

```
mongo --port 27017
```

Switch to the authentication database (in this case, `admin`), and use `db.auth(<username>, <pwd>)` method to authenticate:

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

6

Create additional users as needed for your deployment.

Once authenticated as the user administrator, use `db.createUser()` to create additional users. You can assign any built-in roles or user-defined roles to the users.

The `myUserAdmin` user only has privileges to manage users and roles. As `myUserAdmin`, if you attempt to perform any other operations, such as read from a `foo` collection in the `test` database, MongoDB returns an error.

The following operation adds a user `myTester` to the `test` database who has `readWrite` role in the `test` database as well as `read` role in the `reporting` database.

> **NOTE:**
> The database where you create the user (in this example, `test`) is that user's authentication database. Although the user would authenticate to this database, the user can have roles in other databases; i.e. the user's authentication database does not limit the user's privileges.

```
use test
db.createUser(
  {
    user: "myTester",
    pwd: "xyz123",
    roles: [ { role: "readWrite", db: "test" },
             { role: "read", db: "reporting" } ]
  }
)
```

7

Connect and authenticate as `myTester`.

To authenticate during connection

Start a `mongo` shell with the `-u <username>`, `-p <password>`, and the `--authenticationDatabase <database>` command line options:

```
mongo --port 27017 -u "myTester" -p "xyz123" --authenticationDatabase "test"
```

To authenticate after connecting

Connect the `mongo` shell to the `mongod`:

```
mongo --port 27017
```

Switch to the authentication database (in this case, `test`), and use `db.auth(<username>, <pwd>)` method to authenticate:

```
use test
db.auth("myTester", "xyz123" )
```

Insert into a collection as `myTester`.

As `myTester`, you have privileges to perform read and write operations in the `test` database (as well as perform read operations in the `reporting` database). For example, you can peform the following insert operation in the `test` database:

```
db.foo.insert( { x: 1, y: 1 } )
```

SEE ALSO:

Manage Users and Roles.