MongoDB NHTT Security

Release 3.2

MongoDB, Inc.

August 15, 2016

Contents

1 Security

1.1 Authorization

Learning Objectives

Upon completing this module, students should be able to:

- Outline MongoDB's authorization model
- List authorization resources
- Describe actions users can take in relation to resources
- Create roles
- Create privileges
- Outline MongoDB built-in roles
- Grant roles to users

Authorization vs Authentication

Authorization and Authentication are generally confused and misinterpreted concepts:

- Authorization defines the rules by which users can interact with a given system:
  - Which operations can they perform
  - Over which resources
- Authentication is the mechanism by which users identify and are granted access to a system:
  - Validation of credentials and identities
  - Controls access to the system and operational interfaces

Authorization Basics

- MongoDB enforces a role-based authorization model.
- A user is granted roles that determine the user's access to database resources and operations.

The model determines:

- Which roles are granted to users
- Which privileges are associated with roles
- Which actions can be performed over different resources

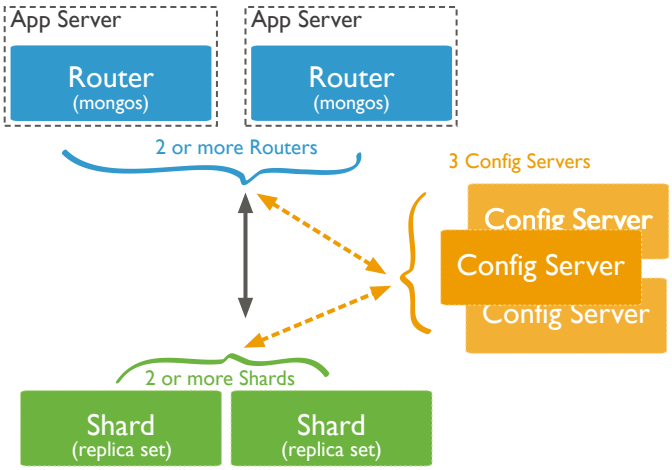
What is a resource?

- Databases?
- Collections?
- Documents?
- Users?
- Nodes?
- Shard?
- Replica Set?

Authorization Resources

- Databases
- Collections
- But that is not all. See next several slides.

Cluster Resources



Types of Actions

Given a resource, we can consider the available actions:

- Query and write actions
- Database management actions
- Deployment management actions
- Replication actions
- Sharding actions
- Server administration actions
- Diagnostic actions
- Internal actions

Specific Actions of Each Type

Query / Write	Database Mgmt	Deployment Mgmt
find	enableProfiler	planCacheRead
insert	createIndex	storageDetails
remove	createCollection	authSchemaUpgrade
update	changeOwnPassword	killop
	...	...

See the complete list of actions[1] in the MongoDB documentation.

Authorization Privileges

A privilege defines a pairing between a resource as a set of permitted actions.

Resource:

```
{"db": "yourdb", "collection": "mycollection"}
```

Action: `find`

Privilege:

```
{
  resource: {"db": "yourdb", "collection": "mycollection"},
  actions: ["find"]
}
```

[1] https://docs.mongodb.com/manual/reference/privilege-actions/

Authorization Roles

MongoDB grants access to data through a role-based authorization system:

- Built-in roles: pre-canned roles that cover the most common sets of privileges users may require
- User-defined roles: if there is a specific set of privileges not covered by the existing built-in roles you are able to create your own roles

Built-in Roles

Database Admin	Cluster Admin	All Databases
dbAdmin	clusterAdmin	readAnyDatabase
dbOwner	clusterManager	readWriteAnyDatabase
userAdmin	clusterMonitor	userAdminAnyDatabase
	hostManager	dbAdminAnyDatabase

Database User	Backup & Restore
read	backup
readWrite	restore

Superuser	Internal
root	__system

Built-in Roles

To grant roles while creating an user:

```
use admin
db.createUser(
  {
    user: "myUser",
    pwd: "$up3r$3cr7",
    roles: [
      {role: "readAnyDatabase", db: "admin"},
      {role: "dbOwner", db: "superdb"},
      {role: "readWrite", db: "yourdb"}
    ]
  }
)
```

Built-in Roles

To grant roles to existing user:

```
use admin
db.grantRolesToUser( {
  "reportsUser",
  [
    { role: "read", db: "accounts" }
  ]
} )
```

User-defined Roles

- If no suitable built-in role exists, we can can create a role.
- Define:
  - Role name
  - Set of privileges
  - List of inherit roles (optional)

```
use admin
db.createRole({
  role: "insertAndFindOnlyMyDB",
  privileges: [
    {resource: { db: "myDB", collection: "" }, actions: ["insert", "find"]}
  ],
  roles: []})
```

Role Privileges

To check the privileges of any particular role we can get that information using the `getRole` method:

```
db.getRole("insertAndFindOnlyMyDB", {showPrivileges: true})
```

1.2 Lab: Administration Users

Premise

Security roles often span different levels:

- Superuser roles
- DBA roles
- System administration roles
- User administration roles
- Application roles

In this lab we will look at several types of administration roles.

User Administration user

- Generally, in complex systems, we need someone to administer users.
- This role should be different from a `root` level user for a few reasons.
- `root` level users should be used has last resort user
- Administration of users is generally related with security officers

Create User Admin user

Create a user that will administer other users:

```
db.createUser(
{
  user: "securityofficer",
  pwd: "doughnuts",
  customData: { notes: ["admin", "the person that adds other persons"] },
  roles: [
    { role: "userAdminAnyDatabase", db: "admin" }
    ]
})
```

Create DBA user

DBAs are generally concerned with maintenance operations in the database.

```
db.createUser(
{
  user: "dba",
  pwd: "i+love+indexes",
  customData: { notes: ["admin", "the person that admins databases"] },
  roles: [
    { role: "dbAdmin", db: "X" }
    ]
})
```

If want to make sure this DBA can administer all databases of the system, which role(s) should he have? See the MongoDB documentation[2].

[2] https://docs.mongodb.com/manual/reference/built-in-roles/

Create a Cluster Admin user

Cluster administration is generally an operational role that differs from DBA in the sense that is more focussed on the deployment and cluster node management.

For a team managing a cluster, what roles enable individuals to do the following?

- Add and remove replica nodes
- Manage shards
- Do backups
- Cannot read data from any application database

1.3 Lab: Create User-Defined Role (Optional)

Premise

- MongoDB provides a set of built-in roles.
- Please consider those before generating another role on your system.
- Sometimes it is necessary to create roles match specific the needs of a system.
- For that we can rely on user-defined roles that system administrators can create.
- This function should be carried by `userAdmin` level administration users.

Define Privileges

- Roles are sets of privileges that a user is granted.
- Create a role with the following privileges:
  - User can read user details from database `brands`
  - Can list all collections of database `brands`
  - Can update all collections on database `brands`
  - Can write to the collection `automotive` in database `brands`

Create the JSON array that describes the requested set of privileges.

Create Role

- Given the privileges we just defined, we now need to create this role specific to database `brands`.
- The name of this role should be `carlover`
- What command do we need to issue?

Grant Role: Part 1

We now want to grant this role to the user named `ilikecars` on the database `brands`.

```
use brands;
db.createUser(
{
  user: "ilikecars",
  pwd: "ferrari",
  customData: {notes: ["application user"]},
  roles: [
    {role: "carlover", db: "brands"}
    ]
})
```

Grant Role: Part 2

- We now want to grant greater responsibility to our recently created `ilikecars`!
- Let's grant the `dbOwner` role to the `ilikecars` user.

Revoke Role

- Let's assume that the role `carlover` is no longer valid for user `ilikecars`.
- How do we revoke this role?

1.4 Authentication

Learning Objectives

Upon completing this module, you should understand:

- Authentication mechanisms
- External authentication
- Native authentication
- Internal node authentication
- Configuration of authentication mechanisms

Authentication

- Authentication is concerned with:
  - Validating identities
  - Managing certificates / credentials
  - Allowing accounts to connect and perform authorized operations
- MongoDB provides native authentication and supports X509 certificates, LDAP, and Kerberos as well.

Authentication Mechanisms

MongoDB supports a number of authentication mechanisms:

- SCRAM-SHA-1 (default >= 3.0)
- MONGODB-CR (legacy)
- X509 Certificates
- LDAP (MongoDB Enterprise)
- Kerberos (MongoDB Enterprise)

Internal Authentication

For internal authentication purposes (mechanism used by replica sets and sharded clusters) MongoDB relies on:

- Keyfiles
  - Shared password file used by replica set members
  - Hexadecimal value of 6 to 1024 chars length
- X509 Certificates

Simple Authentication Configuration

To get started we just need to make sure we are launching our mongod instances with the `--auth` parameter.

```
mongod --dbpath /data/db --auth
```

For any connections to be established to this mongod instance, the system will require a username and password.

```
mongo -u user -p
Enter password:
```

1.5 Lab: Secure mongod

Premise

It is time for us to get started setting up our first MongoDB instance with authentication enabled!

Launch `mongod`

Let's start by launching a `mongod` instance:

```
mkdir /data/secure_instance_dbpath
mongod --dbpath /data/secure_instance_dbpath --port 28000
```

At this point there is nothing special about this setup. It is just an ordinary `mongod` instance ready to receive connections.

Root level user

Create a `root` level user:

```
mongo --port 28000 admin  // Puts you in the _admin_ database
```

```
use admin
db.createUser( {
  user: "maestro",
  pwd: "maestro+rules",
  customData: { information_field: "information value" },
  roles: [ {role: "root", db: "admin" } ]
} )
```

Enable Authentication

Launch `mongod` with `auth` enabled

```
mongo admin --port 28000 --eval 'db.shutdownServer()'
mongod --port 28000 --dbpath /data/secure_instance_dbpath --auth
```

Connect using the recently created `maestro` user.

```
mongo --port 28000 admin -u maestro -p
```

1.6 Auditing

Learning Objectives

Upon completing this module, you should be able to:

- Outline the auditing capabilities of MongoDB
- Enable auditing
- Summarize auditing configuration options

Auditing

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

Audit Events

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

- Schema
- Replica set and sharded cluster
- Authentication and authorization
- CRUD operations (DML, off by default)

Auditing Configuration

The following are command-line parameters to `mongod`/`mongos` used to configure auditing.

Enable auditing with `--auditDestination`.

- `--auditDestination`: where to write the audit log
  - syslog
  - console
  - file
- `--auditPath`: audit log path in case we define "file" as the destination

Auditing Configuration (cont'd)

- `--auditFormat`: the output format of the emitted event messages
  - BSON
  - JSON
- `--auditFilter`: an expression that will filter the types of events the system records

By default we only audit DDL operations but we can also enable DML (requires auditAuthorizationSuccess set to true)

Auditing Message

The audit facility will launch a message every time an auditable event occurs:

```
{
  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>
}
```

Auditing Configuration

If we want to configure our audit system to generate a *JSON* file we would need express the following command:

```
mongod --auditDestination file --auditPath /some/dir/audit.log --auditFormat JSON
```

If we want to capture events from a particular user *myUser*:

```
mongod --auditDestination syslog --auditFilter '{"users.user": "myUser"}'
```

To enable DML we need to set a specific parameter:

```
mongod --auditDestination console --setParameter auditAuthorizationSuccess=true
```

1.7 Encryption

Learning Objectives

Upon completing this module, students should understand:

- The encryption capabilities of MongoDB
- Network encryption
- Native encryption
- Third party integrations

Encryption

MongoDB offers two levels of encryption

- Transport layer
- Encryption at rest (MongoDB Enterprise >=3.2)

Network Encryption

- MongoDB enables TLS/SSL for transport layer encryption of traffic between nodes in a cluster.
- Three different network architecture options are available:
  - Encryption of application traffic connections
  - Full encryption of all connections
  - Mixed encryption between nodes

Native Encryption

MongoDB Enterprise comes with a encrypted storage engine.

- Native encryption supported by WiredTiger
- Encrypts data at rest
  - AES256-CBC: 256-bit Advanced Encryption Standard in Cipher Block Chaining mode (default)
    - symmetric key (same key to encrypt and decrypt)
  - AES256-GCM: 256-bit Advanced Encryption Standard in Galois/Counter Mode
  - FIPS is also available
- Enables integration with key management tools

Encryption and Replication

- Encryption is not part of replication:
  - Data is not natively encrypted on the wire
    - Requires transport encryption to ensure secured transmission
  - Encryption keys are not replicated
    - Each node should have their own individual keys

Third Party Integration

- Key Management Interoperability Protocol (KMIP)
  - Integrates with Vormetric Data Security Manager (DSM) and SafeNet KeySecure
- Storage Encryption
  - Linux Unified Key Setup (LUKS)
  - IBM Guardium Data Encryption
  - Vormetric Data Security Platform
    - Also enables Application Level Encryption on per-field or per-document
  - Bitlocker Drive Encryption

1.8 Lab: Secured Replica Set - KeyFile (Optional)

Premise

Security and Replication are two aspects that are often neglected during the Development phase to favor usability and faster development.

These are also important aspects to take in consideration for your Production environments, since you probably don't want to have your production environment **Unsecured** and without **High Availability**!

This lab is to get fully acquainted with all necessary steps to create a secured replica set using the `keyfile` for cluster authentication mode

Setup Secured Replica Set

A few steps are required to fully setup a secured Replica Set:

1. Instantiate one `mongod` node with no `auth` enabled
2. Create a `root` level user
3. Create a `clusterAdmin` user
4. Generate a keyfile for internal node authentication
5. Re-instantiate a `mongod` with `auth` enabled, `keyfile` defined and `replSet` name
6. Add Replica Set nodes

We will also be basing our setup using MongoDB configuration files[3]

[3] https://docs.mongodb.org/manual/reference/configuration-options/

Instantiate `mongod`

This is a rather simple operation that requires just a simple instruction:

```
$ pwd
/data
$ mkdir -p /data/secure_replset/{1,2,3}; cd secure_replset/1
```

Then go to this yaml file[4] and copy it into your clipboard

```
$ pbpaste > mongod.conf; cat mongod.conf
```

Instantiate `mongod` (cont'd)

```
systemLog:
  destination: file
  path: "/data/secure_replset/1/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/secure_replset/1"
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
net:
  port: 28001
processManagement:
  fork: true
# setParameter:
#   enableLocalhostAuthBypass: false
# security:
#   keyFile: /data/secure_replset/1/mongodb-keyfile
```

Instantiate `mongod` (cont'd)

After defining the basic configuration we just need to call `mongod` passing the configuration file.

```
mongod -f mongod.conf
```

Create root user

We start by creating our typical `root` user:

```
$ mongo admin --port 28001
```

```
> use admin
> db.createUser(
{
  user: "maestro",
  pwd: "maestro+rules",
  roles: [
    { role: "root", db: "admin" }
    ]
})
```

[4] https://github.com/thatnerd/work-public/blob/master/mongodb_trainings/secure_replset_config.yaml

Create clusterAdmin user

We then need to create a clusterAdmin user to enable management of our replica set.

```
$ mongo admin --port 28001
```

```
> db.createUser(
{
  user: "pivot",
  pwd: "i+like+nodes",
  roles: [
    { role: "clusterAdmin", db: "admin" }
    ]
})
```

Generate a keyfile

For internal Replica Set authentication we need to use a keyfile.

```
openssl rand -base64 741 > /data/secure_replset/1/mongodb-keyfile
chmod 600 /data/secure_replset/1/mongodb-keyfile
```

Add keyfile to the configuration file

Now that we have the *keyfile* generated it's time to add that information to our configuration file. Just un-comment the last few lines.

```
systemLog:
  destination: file
  path: "/data/secure_replset/1/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/secure_replset/1"
net:
  port: 28001
processManagement:
  fork: true
setParameter:
  enableLocalhostAuthBypass: false
security:
  keyFile: /data/secure_replset/1/mongodb-keyfile
```

Configuring Replica Set

- Now it’s time to configure our Replica Set
- The desired setup for this Replica Set should be named “VAULT”
- It should consist of 3 data bearing nodes