Securing MongoDB

MongoDB Authentication and Authorization

- While authentication and authorization are closely connected, it is important to note that authentication is distinct from authorization. The purpose of authentication is to verify the identity of a user, while authorization determines the verified user's access to resources and operations.

Creating Administrator User in MongoDB

- Start mongoDB, without authentication enabled.
- $ mongo --port 27017

```
-> use admin
-> db.createUser(
 {
   user: "myUserAdmin",
   pwd: passwordPrompt(), // or cleartext password
   roles: [ { role: "userAdminAnyDatabase", db: "admin" },
"readWriteAnyDatabase" ]
 }
)
```

- Start mongoDB with authentication Enabled

Creating Administrator User in MongoDB

- Start mongoDB, without authentication enabled.
- $ mongo --port 27017 // here grant root privilege to user dba

```
-> use admin
-> db.createUser(
{user:"dba",pwd:"oracle", roles:["root"]}
)
```

- Start mongoDB with authentication Enabled

Enabling Authentication

- Enabling Authentication.
- To enable authentication, change the authorization=enabled in /etc/mogodb.conf
- eg

```
security:
        authorization: enabled
```

- Also , in the startup command of mongod , specify --auth option, as in this example

```
$ mongod --auth
```

Connecting to MongoDB with authentication Enabled

- Authentication Methods

- To authenticate as a user, you must provide a username, password, and the authentication database associated with that user.
- 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.

- $ mongo -u dba -p oracle --authenticationDatabase <db_name>
- $ mongo 10.17.57.12/admin -u dba -p oracle
- $ mongo
  -> use admin
  -> db.auth(“dba”,”oracle”)

Finding as what user connected to the database

-> use admin
-> db.runCommand({connectionStatus:1})

```
MongoDB Enterprise > db.runCommand({connectionStatus:1})
{
	"authInfo" : {
		"authenticatedUsers" : [
			{
				"user" : "dba",
				"db" : "admin"
			}
		],
		"authenticatedUserRoles" : [
			{
				"role" : "root",
				"db" : "admin"
			}
		]
	},
	"ok" : 1
}
MongoDB Enterprise >
```

Creating a Regular User in MongoDB

- -> use admin // This will be the authentication database for the user
- -> db.createUser(

```
{
   user: "reportsUser",
   pwd: passwordPrompt(),  // or cleartext password // here the password will be prompted
   roles: [
     { role: "read", db: "reporting" },
     { role: "read", db: "products" },
     { role: "read", db: "sales" },
     { role: "readWrite", db: "accounts" }
   ]
 }
)
```

Changing the password of the user

- -> use admin // This will be the authentication database for the user
- -> db.changeUserPassword("accountUser", passwordPrompt())

Listing the details of the users

Listing the details of the users

-> use admin

-> db.getUsers()

Listing the details of a specific user

-> use admin

-> db.getUsers({filter:{user:"reportsUser"}});

Or

-> use admin

-> db.getUser("dba")

Assigning/Managing Roles to a User

- **To grant roles to user**

  -> db.grantRolesToUser(
  "reportsUser",[{role:"readWrite",db:"local"},{role:"readWrite",db:"test"}])

- **To update the roles of an existing user**

  -> db.updateUser(
  "reportsUser",{roles:[{role:"readWrite", db: "admin"}]})

  -> db.updateUser(
  "dba",
  {roles:[{role:"userAdminAnyDatabase",db:"admin"},{role:"readWriteAnyDatabase",db:
  "admin"},{role:"readWrite",db:"local"}]})

Removing Roles from a user

```
-> db.revokeRolesFromUser(
... "reportsUser",[{role:"readWrite",db:"test"}]
... )
```

Listing roles assigned to a user

Listing the details of the users

-> use admin

-> db.getUsers()

Listing the details of a specific user

-> use admin

-> db.getUsers({filter:{user:"reportsUser"}});

Or

-> use admin

-> db.getUser("dba")

Dropping a user

-> db.dropUser("reportsUser")

Or

-> db.removeUser("reportsUser")

Creating User Defined Roles

```
db.createRole(
{
role:"NewReadWrite",
privileges:[],
roles:[{role:"readWrite",db:"test"},{role:"read",db:"admin"}]
}
)
```

Details of the roles

- **To get the details about a role**

  -> db.getRole("NewReadWrite")

- **To list all user Defined Roles**

  -> db.getRoles()

  this command will list all user defined role.

Privilege Management

- **Granting Privileges to a Role**

```
db.grantPrivilegesToRole
(
"NewReadWrite",
[
{
resource:{db:"test",collection:""},
action:["insert","read"]
}
]
)
```

Privilege Management

- **Granting Privileges to a Role**

```
db.grantPrivilegesToRole
(
"NewReadWrite",
[
{
resource:{db:"test",collection:""},
action:["insert","read"]
}
]
)
```

Privilege Management

To revoke privileges from a role

```
db.revokePrivilegeFromRole
(
"NewReadWrite",
[
{
resource:{db:"test",collections:""},
action:["insert"]
}
]
)
```

Privilege Management

To list the privileges given for a role

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

To list all the user defined roles with privileges

```
db.getRoles(
   {
     rolesInfo: 1,
     showPrivileges:true,
     showBuiltinRoles: false
   }
)
```

Listing built in Roles

```
-> Use admin
-> db.getRoles(
    {
      rolesInfo: 1,
      showPrivileges:true,
      showBuiltinRoles: true
    }
)
```

Granting Roles to Roles

```
-> use admin
-> db.grantRolesToRole(
"NewReadWrite",
[
{role:"read",db:"local"}
]
)
```

Revoking Roles from Roles

```
-> use admin
-> db.revokeRolesFromRole(
     "NewReadWrite",
     [
     {role:"read",db:"local"}
     ]
     )
```

How to change own password

- To change own password , user should have changeOwnPassword and changeOwnCustomData privileges
- Steps
  - Create a user defined role with the privilegss changeOwnPassword and changeOwnCustomData privileges
  - Create the user with the role or grant this role to the existing user.

How to change own password

```
db.createRole(
{
role:"passwdRole",
privileges:
[{resource:{db:"admin",collection:""},actions:["changeOwnPassword",
"changeOwnCustomData"]}
],
roles:[]
}
)
```

How to change own password

- Create the user defined role.

```
db.createRole(
{
role:"passwdRole",
privileges:
[{resource:{db:"admin",collection:""},actions:["changeOwnPassword",
"changeOwnCustomData"]}
],
roles:[]
}
)
```

- db.grantRolesToUser("dba", [{role:"passwdRole",db:"admin"}])

Built-in Roles in MonoDB

Database User Roles

Every database includes the following client roles:

- read
  - Provides the ability to read data on all non-system collections and the system.js collection.
- readWrite
  - Provides all the privileges of the read role plus ability to modify data on all non-system collections and the system.js collection.

Database Administrator Roles

Every database includes the following database administration roles:

- dbAdmin
  - Provides the ability to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics. This role does not grant privileges for user and role management.
- userAdmin
  - Create and modify roles and users on the current database
- dbOwner
  - The database owner can perform any administrative action on the database. This role combines the privileges granted by the readWrite, dbAdmin and userAdmin roles.

userAdmin

- Provides the ability to create and modify roles and users on the current database. Since the userAdmin role allows users to grant any privilege to any user, including themselves, the role also indirectly provides superuser access to either the database or, if scoped to the admin database, the cluster.

The userAdmin role explicitly provides the following actions:

- changeCustomData
- changePassword
- createRole
- createUser
- dropRole
- dropUser
- grantRole
- revokeRole
- setAuthenticationRestriction
- viewRole
- viewUser

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.

- clusterManger
  - 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.
- clusterMonitor
  - Provides read-only access to monitoring tools, such as the MongoDB Cloud Manager and Ops Manager monitoring agent.
- hostManager
  - Provides the ability to monitor and manage servers.
- clusterAdmin
  - Provides the greatest cluster-management access. This role combines the privileges granted by the clusterManager, clusterMonitor, and hostManager roles. Additionally, the role provides the dropDatabase action.

Backup and Restore 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 backup agent, Ops Manager backup agent, or to use mongodump to back up an entire mongod instance.
- Restore
  - Provides privileges needed to restore data from backups.

All Database Roles

- The following roles are available on the admin database and provide privileges which apply to all databases except local and config
- readAnyDatabase
  - Provides the same read-only privileges as read on all databases except local and config. The role also provides the listDatabases action on the cluster as a whole
- readWriteAnyDatabase
  - Provides the same privileges as readWrite on all databases except local and config. The role also provides the listDatabases action on the cluster as a whole.
- userAdminAnyDatabase
  - Provides the same access to user administration operations as userAdmin on all databases except local and config.
- dbAdminAnyDatabase
  - Provides the same privileges as dbAdmin on all databases except local and config. The role also provides the listDatabases action on the cluster as a whole.

Super User Roles

- root

  Provides access to the operations and all the resources of the following roles combined:

  readWriteAnyDatabase
  dbAdminAnyDatabase
  userAdminAnyDatabase
  clusterAdmin
  restore
  backup

Super User Roles

- 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 role, when scoped to the admin database
  - userAdmin role, when scoped to the admin database
  - userAdminAnyDatabase role

How to assign privileges to a user

- As of version 4.0 MongoDB doesn’t support assigning privileges directly to the user
- Steps
  1. Create a user defined role
  2. Assign the required privileges to the role
  3. Assign the role to the user