02 Control de Acceso Basado en Roles

Intro

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

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 also enables client authorization.

Once access control is enabled, users must authenticate themselves.

Roles

A role grants privileges to perform the specified actions on resource. Each privilege is either specified explicitly in the role or inherited from another role or both.

Privileges

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

Resource

A resource is 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.

Action

An action specifies the operation allowed on the resource. For available actions see Privilege Actions.

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.

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.

Users and Roles

You can assign roles to users during the user creation. You can also update existing users to grant or revoke roles.

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.

Built-In Roles and User-Defined Roles

MongoDB provides built-in roles 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.
Built-In Roles

Database user roles

https://docs.mongodb.com/manual/reference/built-in-roles/#database-user-roles

read

Práctica

```
MacBook-Pro-de-Pedro:~ pedro$ mongo --authenticationDatabase "admin" -u
"superAdmin" -p

> use clinica
switched to db clinica
> db.createUser({
... user: "juan73",
... pwd: "juan4321",
... roles: ["read"]
... })
Successfully added user: { "user" : "juan73", "roles" : [ "read" ] }
```

Abrimos otro mongo con:

```
MacBook-Pro-de-Pedro:~ pedro$ mongo --authenticationDatabase "clinica" -u
"juan73"
```

Y podremos leer pero no escribir en clínica:

```
> show dbs
clinica  0.000GB
> use clinica
switched to db clinica
> db.pacientes.find()
{ "_id" : ObjectId("5e3078860a4428bb0b3d74eb"), "nombre" : "Juan Pedro",
"apellidos" : "Pérez" }
{ "_id" : ObjectId("5e3078960a4428bb0b3d74ec"), "nombre" : "María", "apellidos" :
"López" }
{ "_id" : ObjectId("5e307a450a4428bb0b3d74ee"), "nombre" : "Pepe" }

> db.pacientes.insert({nombre: "Laura"})
WriteCommandError({
	"ok" : 0,
	"errmsg" : "not authorized on clinica to execute command { insert:
\"pacientes\", ordered: true, lsid: { id:
UUID(\"771ca775-52ff-42b3-b0ed-f89f110e16a9\") }, $db: \"clinica\" }",
	"code" : 13,
	"codeName" : "Unauthorized"
})
```

readWrite

Práctica

Podemos cambiar el rol del usuario anterior a readWrite.

En la shell del administrador lanzamos:

```
> db.runCommand({
... usersInfo: "juan73",
... showPrivileges: true
... })
{
        "users" : [
                {
                        "_id" : "clinica.juan73",
                        "userId" : UUID("a38a167f-c10c-4bec-9422-05e66e917bd6"),
                        "user" : "juan73",
                        "db" : "clinica",
                        "mechanisms" : [
                                "SCRAM-SHA-1",
                                "SCRAM-SHA-256"
                        ],
                        "roles" : [
                                {
                                        "role" : "read",
                                        "db" : "clinica"
                                }
                        ],
...
```

Ahora podemos modificar el rol de la siguiente manera:

```
> db.runCommand({
... updateUser: "juan73",
... roles: [{role: "readWrite", db: "clinica"}]
... })
{ "ok" : 1 }
```

Si ahora repetimos la operación de escritura en la shell del usuario podemos comprobar:

```
> db.pacientes.insert({nombre: "Laura"})
WriteResult({ "nInserted" : 1 })
```

Database Administration Roles

https://docs.mongodb.com/manual/reference/built-in-roles/#database-administration-roles

dbAdmin

Idem readWrite + ops sobre system.profile

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.

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.

Práctica

En la shell de administrador lanzamos

```
> use clinica
switched to db clinica
> db.createUser({
... user: "laura73",
... pwd: "laura4321",
... roles: ["userAdmin"]
... })
Successfully added user: { "user" : "laura73", "roles" : [ "userAdmin" ] }
```

Nos logueamos en una shell con este nuevo usuario:

```
MacBook-Pro-de-Pedro:~ pedro$ mongo --authenticationDatabase "clinica" -u
"laura73"
```

Y podemos crear usuarios en esta base de datos

```
> use clinica
switched to db clinica
> db.createUser({user: "manuel90", pwd: "manuel4321", roles: ["read"]})
Successfully added user: { "user" : "manuel90", "roles" : [ "readWrite" ] }
```

También podemos eliminar usuarios de esta base de datos.

```
> db.runCommand( {    dropUser: "juan73" } )
{ "ok" : 1 }
```

Cluster Administration Roles

https://docs.mongodb.com/manual/reference/built-in-roles/#cluster-administration-roles

clusterAdmin

clusterManager
clusterMonitor
hostManager

Backup and Restoration Roles

https://docs.mongodb.com/manual/reference/built-in-roles/#backup-and-restoration-roles

backup
restore

All-Database Roles

https://docs.mongodb.com/manual/reference/built-in-roles/#all-database-roles

readAnyDatabase

Práctica

Desde el superadministrador vamos añadir este rol al usuario manuel90:

```
> db.runCommand({ usersInfo: "manuel90", showPrivileges: true })

> db.runCommand({
      grantRolesToUser: "manuel90",
      roles: [
         {db: "admin", role: "readAnyDatabase"}]
})
{ "ok" : 1 }
```

Si nos loqueamos con este usuario comprobamos que puede leer cualquier base de datos pero solo escribir en clinica:

readWriteAnyDatabase
userAdminAnyDatabase
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.

Superuser Roles

https://docs.mongodb.com/manual/reference/built-in-roles/#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 role, when scoped to the admin database
- userAdmin role, when scoped to the admin database
- userAdminAnyDatabase role

The following role provides full privileges on all resources:

root

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

readWriteAnyDatabase
dbAdminAnyDatabase
userAdminAnyDatabase
clusterAdmin
restore
backup

Also provides the validate privilege action on system. collections.

User-Defined Roles

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

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.

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.

The db.createRole() method accepts the following arguments:

Parameter	Type	Description

role	document	A document containing the name of the role and the role definition.
writeConcern	document	Optional. The level of write concern to apply to this operation.

The role document has the following form:

```
{
  role: "<name>",
  privileges: [
     { resource: { <resource> }, actions: [ "<action>", ... ] },
     ...
  ],
  roles: [
     { role: "<role>", db: "<database>" } | "<role>",
      ...
  ],
  authenticationRestrictions: [
    {
      clientSource: ["<IP>" | "<CIDR range>", ...],
      serverAddress: ["<IP>" | "<CIDR range>", ...]
    },
    ...
  ]
}
```

Práctica

Desde la shell del superadministrador:

```
> use clinica

> db.createRole({
... role: "onlyWritePacientesClinica",
... privileges: [
... {resource: {db: "clinica", collection: "pacientes"}, actions:
                                          ["find","update","insert","remove"]}
... ],
... roles: [
 { role: "read", db: "clinica" }
]
... })
```

Ahora creamos un usuario con ese rol en clinica:

```
> db.createUser({
... user: "maria78",
... pwd: "maria4321",
... roles: ["onlyWritePacientesClinica"]
... })
Successfully added user: { "user" : "maria85", "roles" : [ "onlyWritePacientesClinica" ] }
```

Nos logueamos con ese usuario:

```
MacBook-Pro-de-Pedro:~ pedro$ mongo --authenticationDatabase "clinica" -u
"maria78"

> use clinica
switched to db clinica
> db.empleados.find()
{ "_id" : { "zona" : "sur", "sector" : 1 }, "nombre" : "María" }
{ "_id" : 2 }
> db.empleados.insert({nombre: "Juan"})
WriteCommandError({
	"ok" : 0,
	"errmsg" : "not authorized on clinica to execute command { insert:
\"empleados\", ordered: true, lsid: { id:
UUID(\"2ff3af9d-fce7-4e7f-a0e5-5967a93f136f\") }, $db: \"clinica\" }",
	"code" : 13,
	"codeName" : "Unauthorized"
})
> db.pacientes.insert({nombre: "Juan"})
WriteResult({ "nInserted" : 1 })
```