Securing a local MongoDB server with a user

By default, MongoDB is not authenticated. For local development, this is not a big deal if you're not sharing a computer, but it's still best to secure your database.

We can do this by turning on access control and creating a user. We do this by creating an admin user.

Add users to local MongoDB

Create an admin user

1. Open your CLI and enter `mongosh` at the prompt to start the Mongo Shell.
2. To create an admin user, we need to add a user to the **admin** database. The **admin** database is the default user database for MongoDB.
   a. Switch to **admin** database:

      ```
      ➢ use admin
      ```

   b. Create a new username and password and give it the special `userAdminAnyDatabase` role. This gives permissions to create databases and users **but does not give permission to view database data**. This makes sense for the main admin user (you can specify any username you want for your admin user) security-wise. This alone does not give you read/write permissions, though it allows you to create more users and set permissions on any user (i.e. administer the database). Set an appropriate username and password.

      ```
      ➢ db.createUser({ user: "admin", pwd: "Test123!!", roles: [ { role:
        "userAdminAnyDatabase", db: "admin" } ] });
      ```

      The `userAdminAnyDatabase` role allows the admin to create/drop users, create/drop databases, and assign users to databases but does not allow the admin user to access the data within the DB. (Consider: This is like a landlord/tenant. Landlords can rent out to a tenant but should not just enter a tenant's premises whenever they want. It's about layers of security.)

      If you want a super user, with read/write on any database and user/database admin privileges, use the "root" role.
3. Stop MongoDB.
   a. **Windows**: If installed as a service, open the Services dialog (search for "Services" in Windows), look for MongoDB in the list of services, then stop the MongoDB service.
   b. **Mac**: If Homebrew was used, `brew services stop mongodb-community@8.0`
4. Exit the shell with **Ctrl+C**.
5. Enable user authentication then restart MongoDB.

Windows

If you've installed MongoDB as a service (i.e. if you click on Start then type "Services" and open the **Services** dialog, you find MongoDB in the services list), you can tweak the config file then start using the Services dialog box.

1. Go to the ***bin*** folder where MongoDB has been installed and look for a ***.cfg*** file. This is the config file. Depending on the platform, the config file may be found in an ***etc*** folder instead.
2. Open ***mongod.cfg*** in a code editor.
3. Uncomment the **security** line edit to look like the following (if it's not there or parts are missing, you need to type it out):

```
security:
  authorization: enabled
```

4. Save the file and restart MongoDB using the **Services** dialog.

If the authentication fails after all the steps (below as well), you can turn off authorization control by commenting out the lines in the Mongo config file from step 3 above (security), restart the MongoDB server, then connect unauthenticated. You can then get rid of the users added and start over.

For Mac

1. Edit the conf file. In your terminal, type:

```
cd <base_path_from_below>/etc/mongod.conf
```

   If an M chip, your base path should be: **`/opt/homebrew`**
   If an Intel chip, your base path should be: **`/usr/local`**

   (https://github.com/mongodb/homebrew-brew#default-paths-for-the-mongodb-community-formula)

   For example, if you have an M1 chip, **`cd /opt/homebrew/etc`**

2. You can now open up the folder in Finder so that you can easily open the config file in a code editor. In the etc folder, type in the terminal: **`open .`**
3. Open up the *mongod.conf* file in a code editor.
4. Add the following to the end of the file **with the exact formatting (don't skip spaces or line breaks)**:

```
security:
  authorization: enabled
```

5. Save the file.
6. Restart mongo. (If MongoDB is running because you didn't stop it before, you can use the word "restart" instead of "start".)

```
brew services start mongodb-community@8.0
```

6. If you have not yet exited Mongo Shell, make sure to do so now by typing `exit` at the prompt.
7. Now log in to shell again but this time using the admin user (type the whole line below). By specifying `-p` without a value, this will prompt the shell to ask for a password. Using a value will leave your password exposed in your CLI history which is insecure. You can specify an authentication database if you created a user in a different DB. You can do this using the `--authenticationDatabase` flag. The below line tells shell to check the "admin" database for the login user. Enter your admin password at the prompt

   ```
   ➢ mongosh --port 27017 -u "admin" --authenticationDatabase "admin" -p
   ```

   By default, users are assumed to be created in the admin DB so if "admin" is where you created your user, you don't need to specify `--authenticationDatabase` (see below).

   ```
   ➢ mongosh --port 27017 -u "admin" -p
   ```

Create a DB-specific user and test

1. Now that you're logged in to shell, you can create a user who has full permissions on the testdb database only. For our example, we're creating a user for a database named "testdb", you can create a user for any database you want. You can create all your users in the **admin** DB.
   a. Switch to the database you want to add a user for. (If you're already in the admin DB, you can go to step b below.)
      ```
      ➢ use admin
      ```
   b. Create a new user which has the dbOwner role (this has full permissions for a database, hence "owner"). There are other built-in roles such as "readWrite" if you want more restrictive permissions for security. Notice that the roles key takes an array. This is because you can set multiple roles using multiple { role: … } each { role: … } sets a role and the database on which your setting those role permissions for the particular user.

      ```
      ➢ db.createUser({ user: "testdbuser", pwd: "Test123!!", roles:
        [ { role: "dbOwner", db: "testdb" } ] } );
      ```

      Built-in roles

      Take a look at the documentation for a list of all built-in user roles.

      https://docs.mongodb.com/manual/reference/built-in-roles/#std-label-built-in-roles

2. Test your new users out (you can try Compass using the steps below):
   a. Open MongoDB Compass.
   b. Under "New Connection", if you don't know the connection string click on the **Fill in connection fields individually** link.
   c. Select "Username/password" from the "Authentication" dropdown.
   d. Test the admin user by entering the admin user credentials and typing "admin" as the **Authentication Database**. Click **Connect** to connect right away. If you want to see the connection string, click on the **Paste connection string** link instead then **Connect**.

e. Test the testdbuser user by entering the testdbuser credentials and typing "testdb" as the **Authentication Database** (the Authentication Database should be the database where you created the user who's currently logging in).

Using a connection string with authentication

In the test step 2 above, you can see the format of the connection string though MongoDB Compass hides the password. The actual format you need to connect in your Node app is:

```
mongodb://<db_user_name>:<db_user_pwd>@127.0.0.1:27017/<db_name>?authSource=<user's_db>
```

The authSource is only used if you created a user outside of the admin database. If you created your user within admin, you can omit the `?authSource=<db_name>` part.

This means for our previous Node app example with database name "testdb" and DB user "testdbuser", we can use the following connection string. (Change values accordingly if your user, password and add authSource if the DB name where the user was created was other than **admin**.)

```
mongodb://testdbuser:Test123!!@127.0.0.1:27017/testdb
```

Set up username and password as environment variables

Having the DB user and password connection string in plain text in the code is not secure especially if you want to push to your repo. A better choice is to set up the values as environment variables (in a file) and then add to gitignore. (https://www.coderrocketfuel.com/article/store-mongodb-credentials-as-environment-variables-in-nodejs)

1. Create a file named ***.env*** and add the following lines to it (we'll use the testdbuser above).

   ```
   DB_USER=testdbuser
   DB_PWD=Test123!!
   ```

2. Save the file and add to ***.gitignore*** file. If you don't have a ***.gitignore*** file, create one add following (also ignore ***node_modules*** folder):

   .env
   node_modules/

3. Install the Dotenv Node package. This package allows you to load the variables in ***.env*** and make them available for use in your code.

   ➢ `npm install --save dotenv`

4. Add the require line in the file using the environment variables and initialize the environment variables. You need this loaded as soon as possible so put these lines at the top of the *index.js* file.

   ```
   const dotenv = require("dotenv");
   dotenv.config();
   ```

5. Now modify the connection string to use the environment variables. Use the back ticks since we're using template literals for the environment variables.

   ```
   const dbUrl =
   `mongodb://${process.env.DB_USER}:${process.env.DB_PWD}@127.0.0.1:27017/testdb
   `;
   ```

6. Run the Node app and test. You should still see the database connected and working.