Backup System Guide

Please make sure that your database is protected with authentication and password. If it is not, scroll down to the section **Protect Your Database with Password** to set up it first.

Configuring and Setting Up the Backup System

Setting up your server for the backup system

Our system is Ubuntu 16.04 LTS and you will need git, npm and Node.JS on the back up system. Please look at our deployment guide on how to set up your server and install the necessary tools.

In addition, install `mongodump` by running the following commands:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/3.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.4.list

sudo apt-get update
sudo apt-get install mongodb-org-tools
```

Deploying the backup system from GitHub

1. Clone the repository from GitHub by running

```
  git clone https://github.com/DanSun18/plm-backup-system.git
  cd plm-backup-system
```

1. Checkout the server version by running

```
  git fetch
  git checkout server
```

1. Make sure that your password won't get accidentally uploaded to GitHub by running (adopted from https://stackoverflow.com/questions/4348590/how-can-i-make-git-ignore-future-revisions-to-a-file)

```
git update-index --skip-worktree server/dbOptions.config.js
git update-index --skip-worktree server/email.config.js
```

1. Change necessary configurations. Provide your actual password for the email account and database user in `server/dbOptions.config.js` and `server/email.config.js`. In addition, if you wish to use a different email, make changes to `server/email.config.js` as well as the `sendEmail` function in `server/backup.server.controller.js`.
2. Run `npm install` to install all dependencies.

3. Use `npm run start-dev` to verify that nothing goes wrong.
4. Use `nohup npm run start-dev >/dev/null 2>&1 &` to run the server even after you log out the ssh session or close the terminal. The backup routine will run at 10pm New York Time everyday.
5. Alternatively, you could use the scripts `startNonStoppingServer.sh` and `shutdownServer.sh` provided in the repository to start or shutdown the server, respectively.

Restoring from a backup

1. Log on to the back up system, and navigate to the directory of the backup system (e.g. `cd plm-backup-system`)
2. Determine the backup you want to restore to the original system. For example, if you want the backup on 2018-3-24, the `path-to-backup` could be `backup/database-backup/mongodump-daily/2018-3-24/`
3. On the backup system, run the command `mongorestore --host <address-of-your-server> --username "plmUser" --password "<plmUserPassword>" --authenticationDatabase plm <path-to-backup> --drop --objcheck`. The `--drop` opiton will drop the database before attempt to restore, and `objecheck` will check for validity while attempting to restore.
4. You will see command line outputs that indicate whether the restore was successful. In case of schema unmatch, you will also be notified on the command line.

Check for validity

We recommend checking the validity of the backup before restoring it to your system. This means first try to restore it to the test server. After restoring data on the test server, validate your data by running `db.plm.validate({full:true})` in your mongo shell. If you see `valid: true` in the output, the data is valid to use, and you can follow the same steps to restore your data to your production server.

Protect Your Database with Password

We outline the steps to make the database password protected here. If your database is already password protected, you may skip this section.

On the server

(adopted from https://ianlondon.github.io/blog/mongodb-auth/) We assume that mongo is running as a service on your service without password protection. For how to start mongo as a service, see our development guide. If your database is already username-password protected, you may want to skip this part.

1. `ssh` into the server, and start the mongo shell by typing `mongo`.
2. Create a user for your database. In our case, we want to add a user `plmUser` to our database `plm`. The user will have `readWrite` role in the `plm` database.

```
use plm
db.createUser(
  {
    user: "plmUser",
    pwd: <plmPassword>,
```

```
    roles: [ { role: "readWrite", db: "plm" } ]
  }
)
```

<plmPassword> is a the password for your user, and it is a string such as "abc123".

1. Enable Auth and Open MongoDB access

Edit your MongoDB config file for the mongodb service. First, find out the the path of your config file by viewing the script to start your mongodb service. On our system, we typed `vi /etc/init.d/mongodb` . Inside the file you will see a line `CONF=<path-to-config-file>`. In our case `<path-to-config-file>` is `/etc/mongodb.conf`. Open the file by typing `sudo vi <path-to-config-file>`. Inside the file, uncomment the line `auth = true`, and change the line specifying `bind_ip` to `bind_ip = 0.0.0.0`.

1. If you have not already, open port 27017 on your server by typing `sudo ufw allow 27017`
2. Restart the mongo daemon by typing `sudo systemctl restart mongodb`. Make sure you can still log in with mongo while ssh'd into the server.

Note: you can try to log in as the newly created `plmUser` by typing `mongo -u "plmUser" -p <plmPassword> --authenticationDatabase "plm"` on the server.

Locally

(adotped from https://docs.mongodb.com/manual/tutorial/enable-authentication/)

1. Start the MongoDB without access control by navigating to the `mongodb` directory and run

```
sudo ./bin/mongod
```

1. In a separate terminal window, connect a mongo shell to the instance by by typing

```
sudo ./bin/mongo
```

1. Create the user administrator.

We want to create a user with the `userAdminAnyDatabase` role in the `admin` database. This user has privileges over creating users in other databases. Switch to the `admin` database by typing `use admin` in the mongo shell. Then, type the following command in the mongo shell:

```
db.createUser(
  {
    user: <username>,
    pwd: <password>,
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

<username> is a string such as "myUserAdmin", and <password> is a string such as "abc123".

(Note: You can check for existing users in the database by typing db.getUsers(). If you would like to delete any existing user, use db.dropUser(<username>))

Disconnet the Mongo Shell.

1. Re-start the MongoDB instance with access control.

In our case, this means running

```
sudo ./bin/mongod --auth
```

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.

1. Connect and authenticate as the user administrator

Start the mongo shell by typing

```
sudo ./bin/mongo -u <username> -p <password> --authenticationDatabase "admin"
```

where <username> and <password> are the same as you set in step 3 (Do not forget the quotation marks to mark them as string).

1. Create users for your database

Once logged into the mongo shell as your newly created user administrator, you can use db.createUser() to create additional users. In our case, we want to add a user plmUser to our database plm. The user will have readWrite role in the plm database.

```
use plm
db.createUser(
  {
    user: "plmUser",
    pwd: <plmPassword>,
    roles: [ { role: "readWrite", db: "plm" } ]
  }
)
```

1. Verify by connecting and authenticating as plmUser

Start another mongo shell by typing

```
sudo ./bin/mongo -u "plmUser" -p <plmPassword> --authenticationDatabase "plm"
```

Perform some database operations to verify that you can do it.