Installing MySQL on AWS

Cheat Sheet

INET2005 Web App Programming I
Sean Morrow

While phpMyAdmin is a nice solution, installing and administrating MySQL via the command line is a useful skill. With AWS it is the easiest way to set MySQL up and work with it.

This cheat sheet assumes that an EC2 instance is up and running and all other steps of document CHEAT SHEET - Publishing ASP.NET Core to AWS.pdf are complete.

Step 1:
Connecting to the EC2 Instance with SSH and key pair

- click on the running EC2 instance and click the connect button for detailed instructions on how to connect to your server via SSH
- open a terminal (or windows bash / or putty) in same folder as the pem key pair file and run commands:

```
chmod 400 dotnetcoreServer.pem
ssh -i " dotnetcoreServer.pem" ubuntu@[public DNS of EC2
instance]
```

- -i is identity file and ubuntu is the username while after @ is public DNS of the EC2 instance (found in the description of your EC2 instance)
- access should be granted and you will see a flashing cursor – you can now run commands on your server!

Step 2:
Install MySQL via APT

- run commands:

```
sudo apt-get update
sudo apt-get install mysql-server
```

- start up the MySQL server with command:

```
sudo systemctl start mysql
```

Step 3:
Login to MySQL shell without a password and add root user password

- run command to get into MySQL shell (indicated by a mysql> prompt):

```
sudo mysql -u root
```

- ubuntu sets up MySQL to use auth_socket plugin for the user credentials. Need to change this to use mysql_native_password plugin instead. Run SQL commands:

```
USE mysql;
UPDATE user SET plugin='mysql_native_password' WHERE
User='root';
FLUSH PRIVILEGES;
exit;
```

- restart the MySQL server with command:

```
sudo systemctl stop mysql
sudo systemctl start mysql
```

- log back into the MySQL shell with command:
  ```
  sudo mysql -u root
  ```
- change root password to something better than blank (default) with command:
  ```
  ALTER USER 'root'@'localhost' IDENTIFIED WITH
  mysql_native_password BY 'new-password';
  ```
- exit the MySQL shell again with command:
  ```
  exit
  ```

Step 4:
Add a user / password your web app will use as outlined in your connection string

- login with newly configured root username / password with command:
  ```
  mysql -u root -p
  ```
- run command to add new user:
  ```
  CREATE USER 'w0090347'@'localhost' IDENTIFIED BY 'Forrester308';
  ```
- create a new database as outlined in your connection string. Run command:
  ```
  CREATE DATABASE sean_dotnetcoreSamples;
  ```
- grant full privileges to this database to our new user w0090347 with command:
  ```
  GRANT ALL PRIVILEGES ON sean_dotnetcoreSamples.* TO
  'w0090347'@'localhost';
  ```
- exit out of MySQL shell

Step 5:
Add data into the database via a SQL script and test:

- upload the SQL script file to the server and navigate to the folder it is located
- run command to run SQL script with MySQL server:
  ```
  mysql -u w0090347 -p < [SQL Script Name].sql
  ```
- note that you must include a USE sean_dotnetcoreSamples; in your SQL Script for it to know what database it is running on
- To test connect to MySQL shell and run usual SQL commands:
  ```
  mysql -u=w0090347 -p
  SHOW DATABASES;
  USE [database name];
  SHOW TABLES;
  SELECT * FROM [tablename]
  ```

References:

https://support.rackspace.com/how-to/installing-mysql-server-on-ubuntu/
https://stackoverflow.com/questions/39281594/error-1698-28000-access-denied-for-user-rootlocalhost
https://linuxize.com/post/how-to-create-mysql-user-accounts-and-grant-privileges/

Step 6:
Upgrade .net core to version 2.2

- run command on your server:

```
sudo apt install dotnet-sdk-2.2
```

Step 7:
Setup Security Group:

- in the AWS EC2 dashboard - click on instance and scroll to the right to find out what security group the EC2 instance belongs to
- go to Network and Security / Security Groups link on left of page
- click inbound tab and edit button to add a new rule (by default only port 22 allowed through which is what SSH terminal is using – we need to be able to hit this server with HTTP (a browser request!))
- The new rule specs are: http / 80 / 0.0.0.0/0

Step 8:
Publish ASP.NET Core web app

- in web app project folder on your computer run commands:

```
dotnet restore
dotnet publish --output ".\published" --configuration release
```

Step 9:
Copy web app files to ubuntu server on AWS

- there are a few ways to get your files from your computer up to your server on AWS. You can set it up to pull your files directly from GitHub, for instance. In order to clearly see and understand the process we will be using FTP
- open filezilla or WinSCP and setup a connection:
  Protocol: SFTP
  Port: 22
  Host: public DNS of your EC2 Instance
  Logon type: key file (target .pem keypair for SSH)
  User: ubuntu
- in terminal or putty on your server, create a new folder for your web app to be uploaded to – use command:

  ```
  mkdir [name of app folder]
  ```

- upload all contents of the /published folder in your web app project folder to the newly created folder on your server

Step 10:
Install Apache2 server

- even though .NET Core is pre-installed on our server, there is no HTTP server!
- install with commands:
```
sudo apt update
sudo apt install apache2
```

Step 11:
Configure Apache2 Server

- This Apache2 server needs to receive requests and pass them on to a locally running kestrel server which will be running our web app. For this to happen we need to configure the Apache2 server to be a reverse proxy server.
- To enable run commands:
```
sudo a2enmod proxy
sudo a2enmod proxy_http
```
- Add a configuration file using the nano text editor with command:
```
sudo nano /etc/apache2/sites-enabled/000-default.conf
```
and modify it to:
```
<VirtualHost *:80>
      ProxyPreserveHost On
      ProxyPass / http://127.0.0.1:5000/
      ProxyPassReverse / http://127.0.0.1:5000/

      ServerName [PUBLIC DNS FROM AWS CONSOLE]
      ServerAlias [PUBLIC DNS FROM AWS CONSOLE]

      ErrorLog ${APACHE_LOG_DIR}/error.log
      CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- What is this? It sets up the apache server to take requests to the ServerName and pass them along to the kestrel server running at http://localhost:5000 (which is 127.0.0.1:5000)
- Apache2 server runs as a service. This mens it will continue to run in the background all the time, even when we disconnect from the machine
- We haven't started the Apache2 server yet – but will as a final step

Step 12:
Setup Kestrel server to run as a service

- Much like the Apache2 server, we need the Kestrel server to also run as a service all the time – this needs to be done more manually through a script file
- run command (pick a name for your service like "kestrel"):
```
sudo nano /etc/systemd/system/[YOUR SERVICE NAME].service
```
- services in /etc/systemd/system/ will run automatically

- Enter the following into the empty file using nano text editor:

```
[Unit]
Description=.NET Web App running on CentOS 7

[Service]
WorkingDirectory=/home/ubuntu/[YOUR PROJECT FOLDER]
ExecStart=/usr/share/dotnet/dotnet /home/ubuntu/[YOUR PROJECT FOLDER]/[YOUR PROJECT'S DLL FILE].dll
Restart=always
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=dotnet-example
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

- Enable the service with command:
  `sudo systemctl enable [YOUR SERVICE NAME].service`

Step 13:
Time to start it all up!

- Start up Apache2 server with command:
  `sudo service apache2 start`
- Start up Kestrel server service with command:
  `sudo systemctl start [YOUR SERVICE NAME].service`
- Hit your web app with a browser using the public DNS URL outlined in EC2 instance dashboard

Other useful commands:

- to stop Kestrel server service:
  `sudo systemctl stop [YOUR SERVICE NAME].service`
- to test status of Kestrel server service:
  `sudo systemctl status [YOUR SERVICE NAME].service`
  there should be no errors!
- to shut down Apache2 server:
  `sudo service apache2 stop`
- to test status of Apache2 server:
  `sudo systemctl status apache2`