AWS Deployment

Please read these instructions very carefully! Anywhere you see double curly brackets `{{ }}` , these are placeholders for your project-specific information. You will need to replace them and everything contained within them with your data - eg, file names, IP addresses, etc.

Part 1: Get prepared

Core Setup Requirements

. An AWS account

. A GitHub or bitbucket account

. An Internet connection

. Git should be installed locally

Get going

. Create a local, functional, full MEAN project

. Keep your project version controlled via git - good practice anyway.

. Create a .gitignore file in your project

```
touch .gitignore
```

. Add the following to the .gitignore file

```
.DS_Store
.idea
.vagrant
bootstrap.sh
VagrantFile
bower_components/
node_modules
package-lock.json
```

Make a GitHub/bitbucket repository

. Push your project to that GitHub/bitbucket repository

Part 2: Set up AWS

. Enter AWS, and click launch new instance.

. Select Ubuntu 16.04 LTS

. Select t2.micro

. Set security settings:

- `ssh 0.0.0.0, (Anywhere or myIP)`
- `http 0.0.0.0 (Anywhere)`

- `https 0.0.0.0 (Anywhere, or don't set it)`

Download a `.pem` key from AWS or use an existing key

Move the `.pem` file to an appropriate folder on your system

From the folder containing your `.pem` file, change its user permission with this command:

```
chmod 400 {{file_name}}.pem
```

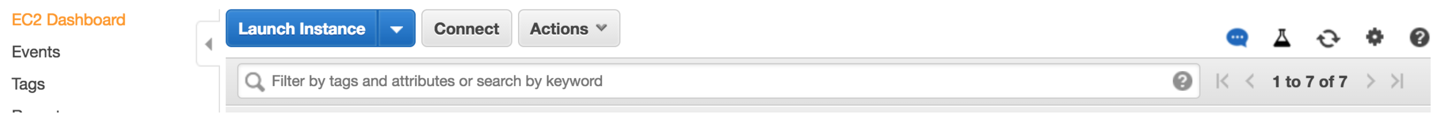
For PC users, this command may require kitty, putty, or bash terminal.

We are now ready to enter the cloud server!

Part 3: Enter the cloud server

Navigate to the directory where your .pem file is!

Copy the command to SSH into your EC2 instance by clicking the Connect button on AWS.



A pop-up will show you how to connect to your instance. All you need is the line of code underneath where it says **Example**, which will follow this basic format: `ssh -i {{mypem}}.pem ubuntu@{{yourAWS.ip}}`

For PC users, this command may require kitty, putty, or bash terminal.

From the directory where your .pem file is, run the command you copied from AWS to SSH into your EC2 instance.

When prompted, type `yes` !

You are now in the EC2 server!

Part 4: Install dependencies

In the ubuntu terminal: These establish some basic dependencies for deployment and the Linux server.

```
sudo apt-get update
sudo apt-get install -y build-essential openssl libssl-dev pkg-config
```

Type the following lines one at a time because they require confirmation. The first two commands install basic node and npm. The third line forcibly cleans the cache, which will give you an interesting comment. :)

```
sudo apt-get install -y nodejs nodejs-legacy
sudo apt-get install npm -y
sudo npm cache clean -f
```

Note: In case the first command does not work, try `sudo apt install nodejs-legacy` instead.

. Install the node package manager **n** and updated node.

```
sudo npm install -g n
sudo n stable
```

. Install the Angular CLI

```
sudo npm install -g @angular/cli
```

. Install NGINX and git:

```
sudo apt-get install nginx git -y
```

Part 5: Clone your project

. Navigate to /var/www

```
cd /var/www
```

If `/var/www` does not exist, then run `sudo mkdir /var/www` , after which you may navigate to it.

. Clone your project

```
sudo git clone {{your project file path on github/bitbucket}}
```

At this point, you should be able to navigate into your project and make sure everything is looking just as you remember it.

Part 6: Set up NGINX

. Go to nginx's sites-available directory

```
cd /etc/nginx/sites-available
```

. Create a file using vim and name it after your cloned repo

```
sudo vim {{your cloned repo's name}}
```

vim is a terminal-based text editor. For more info see: vim-adventures.com/ or other vim learning tools. The key commands for us are

- **i**, which allows us to type, or **insert** text
- **esc**, which turns off insert
- **:wq**, which we use to **write** (also known as save) and **quit**

. Add the following code to the file you just made by using vim. Enter insert mode by clicking **i**. Change the two placeholders inside of double curly brackets `{{ }}` to match your specifications.

```
server {
    listen 80;
    location / {
        proxy_pass http://{{PRIVATE-IP}}:{{NODE-PROJECT-PORT}};
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

This code says: have the reverse proxy server (nginx) listen at port 80. When going to root /, listen for http requests as though you were actually http://<your private ip> and the port where your server is listening e.g @8000 or @6789 etc.

Learn more from nginx: http://nginx.org/en/docs/http/ngx_http_proxy_module.html

. Remove the default from nginx's sites-available directory

```
sudo rm default
```

. Create a symbolic link from sites-enabled to sites-available:

```
sudo ln -s /etc/nginx/sites-available/{{repo name}} /etc/nginx/sites-enabled/{{repo name}}
```

. Remove the default from nginx's sites-enabled diretory

```
sudo rm /etc/nginx/sites-enabled/default
```

Part 7: Project Dependencies and pm2

. Install pm2 globally

```
sudo npm install pm2 -g
```

pm2 is a production process manager that allows us to run node processes in the background. (https://www.npmjs.com/package/pm2.5) (https://www.npmjs.com/package/pm2).

. Navigate back to your project and change its permissions

```
cd /var/www
sudo chown -R ubuntu {{repo name}}
```

. Navigate into the project, install the needed node modules, and build the dist folder

```
cd {{repo name}}
npm install
cd {{angular project name, we usually call it public}}
```

```
npm install
ng build
```

Part 8: MongoDB

. Import GPG key

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A22
91FA4AD5
```

Ubuntu checks that software packages are authentic by checking that they are signed with a GPG key, which is used for encryption. The code above imports the key for the official MongoDB repository. You may also check the documentationfor the most updated key here.

. Define where to download the packages

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6 mul
tiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list
```

The above command creates a list file for MongoDB. Again, check the documentation for the most recent command.

. Re-update to integrate Mongo

```
sudo apt-get update
```

. Install MongoDB

```
sudo apt-get install -y mongodb
```

Use `sudo apt-get install -y mongodb-org` if the above command does not work.

. Create /data/db

```
sudo mkdir /data
sudo mkdir /data/db
```

.

```
sudo service mongod start
```

This command will run mongoDB as a daemon, or as a background process.

If you're having trouble getting mongod to start with `service`, try running it manually with `sudo mongod` and check for error messages.

. Check the status of your service. Use `ctrl C` when you are done.

```
sudo service mongod status
```

. Now we want to automatically start Mongo when the system starts.

```
sudo systemctl enable mongod && sudo systemctl start mongod
```

Part 9: Start your server!

. Navigate to your project

```
cd /var/www/{{repo name}}
```

. Start your pm2 project

```
pm2 start server.js
```

. Restart nginx

```
sudo service nginx stop && sudo service nginx start
```

At this point, you should be able to navigate to your AWS public IP and see your live project!

Use `pm2 logs` to see the logs of your different pm2 instances.

`pm2 show {{ pm2 instance id }}` will give you details of that instance.