CRM Server Side Documentation

How to Setup and Configure a Server and Deploy the Application on it

In this part of the series of documentation, I outline the steps I took to setup a production ready server, configure it and deploy a built MERN stack client side application.

TODO

- Automate deployment process using AWS CLI.
- Add Testing and CI/CD.

This part covers the following:

1. Creating a new Ubuntu Server on AWS EC2 (Elastic Cloud Compute).
2. Connecting to the Ubuntu EC2 instance via SSH.
3. Setting up Node.js, MongoDB and NGINX on the running server.
4. Deploying the Node.js and MongoDB back-end API.
5. Deploying the MERN front-end application.
6. Configuring NGINX to serve the Node.js API and React front-end
7. Testing the application on a browser.

These steps should only be carried out for the first time.

Creating a new Ubuntu Server on AWS EC2

1. First, sign in to the AWS Management console at https://aws.amazon.com/console/.
2. Go to Services and select EC2 from the "Compute" section.
3. Click the "Launch Instance" button.
4. **Chose AMI** - Check the "Free tier only" checkbox. Type in the search box "Ubuntu" and select the "Ubuntu Server 18.04" Amazon Machine Image (AMI) from the filtered results.
5. **Choose Instance Type** - Select the "t2.micro" (Free tier eligible) instance type and click the "Configure Security Group" tab at the top. If you can't see this, make sure your browser display is maximum width.
6. **Configure Security Group** - Add a new rule to allow HTTP traffic then click "Review and Launch"/

7. **Review** - Click Launch.
8. Select "Create a new key pair", enter a name for the key pair (e.g "crm-key") and click "Download Key Pair" to download the private key. This is the key that will be use to connect to the server via SSH. (Alternatively, you can use an existing key, but don't mix or lose keys for the projects you are working on)

Connecting to the Ubuntu EC2 Instance via SSH

If the steps in the section above were followed correctly, the EC2 Instance should be up and running. (Look for a "Running" status in green on the EC2 dashboard in the AWS Console). This means you can connect to it via SSH using the private key that you downloaded in the last step.

To connect to the EC2 instance via SSH:

On Windows: Follow the instructions here to connect to the EC2 Instance via SSH.

On Mac/Linux: 1. Open a terminal window and update the permissions of the private key file with the command `chmod 400 <path-to-key-file>` e.g. `chmod 400 ~/Downloads/my-aws-key.pem`, the key must not be publicly viewable for SSH to work. 2. Copy the "Public DNS (IPv4)" property from the instance description tab in the AWS Console, then connect to the instance from the terminal window with the command `ssh -i <path-to-key-file> ubuntu@<domain name>` e.g. `ssh -i ~/Downloads/my-aws-key.pem ubuntu@ec2-52-221-185-40.ap-southeast-2.compute.amazonaws.com`
3. Enter yes to the prompt "Are you sure you want to continue connecting (yes/no)?" to add the url to your list of known hosts.

Setting Up Node.js, MongoDB and NGINX on the Running Server

The below command executes a script to automatically setup and configure a production ready MERN Stack web server on Ubuntu that includes Node.js, MongoDB, PM2, NGINX and UFW.

For more details about how the script works see Setup Node.js + MongoDB Production Server on Ubuntu.

While connected to the new AWS EC2 instance in the terminal window, run the following command: `curl https://gist.githubusercontent.com/Chizaram-Igolo/f1277f40e14b597af5cd | sudo bash`

Deploying the Node.js and MongoDB back-end API

Follow these steps to setup the Node.js API on the server and configure NGINX to enable access to it.

1. Clone the Node.js + MongoDB API project into the /opt/back-end directory with the command sudo `git clone https://github.com/Juvenix/crm-backend.git /opt/crm-backend`
2. Navigate into the back-end directory and install all required npm packages with the command `cd /opt/crm-backend && sudo npm i -g yarn && sudo yarn`
3. Start the API using the PM2 process manager with command `sudo pm2 start server.js`

The API is now running on Node.js under the PM2 process manager and listening on port 4000. Only port 80 (HTTP) is publicly accessible on the server so you can't hit the API yet, this will be possible after you've configured NGINX as a reverse proxy to pass through HTTP traffic to the api (more on this shortly).

Deploying the MERN front-end application

Follow these steps to setup the React application on the server. Make sure you have already built the React app for deployment.

1. Clone the React project into the **/opt/crm** directory with the command `sudo git clone https://github.com/Juvenix/crm.git /opt/crm`
2. ~~Navigate into the front-end directory and install all required npm packages with the command~~ `cd /opt/crm && sudo yarn`
3. ~~Build the front-end app with the command~~ `sudo npm run build`

   Because you are using a free tier EC2 instance, you only get 1GB of RAM on that AMI. This won't be enough to build the project so you will need to built it first on your local machine before carrying out steps 1. You only need to carry out step 1. Make sure that the static files are built into a `build` folder in the root level of the React project.

If you did the above correctly, the React app is now ready to be served from the `/opt/crm/build` directory. In the next step, you'll configure the NGINX web server to enable access to it.

Configuring NGINX to serve the Node.js API and React front-end

Since the MERN Stack application is made up of two separate projects that both need to be accessed via the same port (HTTP on port 80), you're going to use NGINX as the public facing web server to receive requests for both the front-end and back-end, and decide where to send each request based on its path. Requests beginning with the path /api/* will be proxied through to the Node.js

api running on port 4000, while other requests will serve the React front-end app and associated files (js/css/images).

Follow these steps to configure NGINX for the MERN stack app.

1. Delete the default NGINX site config file with the command `sudo rm /etc/nginx/sites-available/default`
2. Launch the nano text editor to create an new default site config file with `sudo nano /etc/nginx/sites-available/default`
3. Paste in the following config:

```
server {
  listen 80 default_server;
  server_name _;

  # react app & front-end files
  location / {
    root /opt/crm/build;
    try_files $uri /index.html;
  }

  # node api reverse proxy
  location /api/ {
    proxy_pass http://localhost:4000/;
  }
}
```

4. Save the file by pressing `Ctrl` + `x` and selecting Yes to save.
5. /Restart NGINX with the command `sudo systemctl restart nginx`

NGINX Config Reference

`server { ... }` defines a server block which contains the configuration for a virtual server within NGINX.

`listen 80 default_server;` uses the listen directive to configure the virtual server to accept requests on port 80 and sets it as the default virtual server on this NGINX server.

`server_name _;` uses the server_name directive to set the server name to an underscore (_) to make this server block a catch-all block that matches any domain name that doesn't match another more specific server block. Since this example has only one server block it will match all domain names.

`location / { ... }` defines a location block which contains the configuration for requests that have a URI beginning with a forward slash (/), unless the request URI matches another more specific location block.

`root /opt/front-end/dist;` uses the root directive to set the root directory to the front end dist folder (/opt/front-end/dist) for requests matching this location block.

`try_files $uri /index.html;` uses the try_files directive to first check for the existence of a file matching the request URI (`$uri`) and returning it if there is one. If no file matches the request URI then it defaults to returning `/index.html`.

`location /api/ { ... }` defines a location block which contains the configuration for requests that have a URI beginning with `/api/`.

`proxy_pass http://localhost:4000/;` uses the proxy_pass directive to proxy requests beginning with `/api/` through to the Node.js API running at `http://localhost:4000/`.

Testing the Application on a Browser

Enter the hostname of the AWS EC2 instance in a browser to access and test the deployed MERN stack application.

The hostname is the "Public DNS (IPv4)" property located on the instance description tab in the AWS Console.