MERN Deployment on Azure VM

I : Deploying a MERN Application on Azure VM using Nginx **on the same VM :**

Ensure the Vite Application is running on the server is running properly on a specific port and allowing specific IP's. Also ensure proper Nginx and Firewall Configurations.

1 : Create a MERN Application in VsCode which is rendering the UI and saving the data to MongoDb.

- we should make sure the app is listening to all the IP's be it public or private by passing '0.0.0.0' to app.listen
    - *ideally, the node application should run on 0.0.0.0 and not on localhost as listening on localhost just give the access to the machine the application is running on and not outside sources which will defeat the purpose.*

- on the frontend side, we can make sure the react application is ideally requesting to the Load Balancer/ API Gateway IP and in case of testing, to the VM's public IP when making axios requests.
    - These requests should ideally be https

when trying it run considering security :

Run the application on 0.0.0.0 making it listen to all the IP's byu default but later we should allow traffic from only Load balancers and API Gateway or run it behind a proxy. Use **Private Endpoint.**

2 : Create a VM on Azure. Login using ssh and key downloaded from Azure**.** We can also generate a new key-pair using **ssh-keygen.** Also give necessary permissions to the key we will use to login into the server with command :

chmod 600/700 <dirname/file name>

The ssh command is :
ssh -i <path to the key> <server user name>:<server public IP>

After logging into the server, follow the following steps :

- *update all the packages : sudo apt update && sudo apt upgrade -y*
- *add inbound rule to allow http/https to the VM or a specific port/protocol access, as we just have ssh access by default when creating the vm.*

3 : Inorder to deploy the application on the VM, we usually shouldn't do it as root. We will add a user to the Server that

too in the sudo group which will give elevated permissions to the user.

The steps to add the user are :

- adduser <username>

- Add the user to the sudo group to grant administrative privileges:

  command : usermod -aG sudo <username>

- Switch to the new user :
  - *su - <username>*

- Test sudo access :
  - *sudo whoami*

We should disable the root login by going into ssh_config file and changing 2 params :

- *PermitRootLogin no*
- *PasswordAuthentication no*

Restart the service :

- *sudo systemctl restart ssh*

4 : Then logout out of root and login as the user just created by command :

ssh <username>@<server IP>

For below point : In ssh directory inside a User Dir, we have ssh_config.d and sshd.config.d along with other files. ssh denotes ssh client config and sshd denotes ssh server config.

By default, SSH client settings are configured in:

- */etc/ssh/ssh_config (global settings)*
- *~/.ssh/config (user-specific settings)*

5 : Usually, after logging in as a new user, we will disable root login and password login as these are not good practices. Using system as a root can cause bad-consequences which can upset the system and logging-in using a password can make a system vulnerable to attacks.

make changes to /etc/ssh/sshd_config.d

- *change port : Port <port number>*
- *PasswordAuthentication : no*
- *Disable root login : PermitRootLogin no*
- *Allow only specific users : AllowUsers your_user*
- *Limit Login Attempts :* MaxAuthTries 3

Restart the service : sudo systemctl restart ssh

6 : After making the changes to the configuration, we can ssh into the vm with the command :

ssh <username>:<Server IP> -p <new port number>

7 : Make some firewall changes using **ufw** to allow sshing only to the new ssh port, allow http and https connections.

- We can make some additional changes to the ufw configuration by going into /etc/ufw/before.rules

8 : We can first clone the Application from the repo and then build it on the Server itself and not locally. But for that we need to setup the Nodejs environment and other packages for the application dependencies to be installed. The steps are :

- sudo apt update && sudo apt upgrade -y

- curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
- sudo apt install -y nodejs

For testing purpose, we can access the application from our local machine, by adding our local machine public IP and a

port our react application is running on to the VM's inbound rule.

9 : If the DB is hosted on Atlas, we need to add our Server PIP to the Atlas DB Network settings.

10 : We need to install pm2 and configure it to run our node.js application even if we close the terminal.

Detailed NGINX Config : Below is detailed description and why do we need it :

In the absence of Nginx, users can directly access the nodejs and react application on their respective ports via the Vm's <Public IP>:<port no>. This exposes the application to the public internet. Hence we setup Nginx which acts as a reverse proxy.

We setup Nginx in such a way that it works as a reverse proxy which means it receives the requests on behalf of our backend application for security purposes. The requests are then forwarded to our backend application on the port we configure it to run on (port 3000 for react). Even the React Application which usually runs on port 3000 is actually accessed via Nginx's port 80 (or whichever port is configured). We configure Nginx in a way it serves the React build files which are static files.

11 : Installing and Configuring NGINX which is very important.

- We can either point the Nginx to serve the static website files by pointing it to the directory we have generated in the react project by running the npm run build command or copy the html, css and js files from local machine (wsl2 in this case) to **/var/www/html** location on the remote server where Nginx can server the website static files to the users as this is the default location where web servers like apache and nginx look for, when they serve the web pages.

- **Not the best approach :**
  Inorder to make the changes, we have to open the */etc/nginx/sites-available* directory and default file and change :

- root field to point to the directory where we save the static files
- server_name : points to our server's public IP or domain name
- location block / : is inside the server block indicating to server index.html on home route
  - in proxy_pass directive : If our frontend application is hitting '/' route, then we need to pass 'http://localhost:<backend port>/' so that reaching / will proxy user to the home page.

Sample default file inside sites-enabled :

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # root /var/www/bookstore;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name 4.213.170.209;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.

                root /var/www/bookstore;
                try_files $uri $uri /index.html;
        }

        location /books {
                proxy_pass http://127.0.0.1:5555/books;
                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;
```

}

- later if the react application is searching for ‘/api’ , then we should add the ‘location /api block’ under the ‘location / block’ and add ‘proxy_pass' directive to the /api block which will be ‘http://localhost:5555/api’

- make sure if api endpoints start with /api. if they do then, proxy_pass should be **localhost:5000/api** and if not then **localhost:5000/api/**. here the trailing / will remove api from the endpoint.

- add a location block preferably **/api** and rarely /api/: add localhost url and the port out application is running on.
  - http://localhost:5555;
    - avoid the trailing /

- **Proper Approach :**

  Always make a separate file for configurations inside **/etc/nginx/conf.d** directory. Every config file should ideally be named after the domain its configured for.

  eg. mern-bookstore.com.conf

  Understanding Nginx Behavior When Serving a Static Website :

When you configure Nginx to serve a website from **/var/www/bookstore** using the root directive, it works as follows:

1. Accessing bookstore/ (Homepage)

If you access http://yourdomain.com/bookstore/, Nginx will look for an index file inside /var/www/bookstore.

- It will check for files like index.html or index.htm (as per your index directive in Nginx configuration).
- If an index file exists, it will be served.
- If no index file exists and autoindex on; is not enabled, Nginx will return a 403 Forbidden or 404 Not Found.

2. Accessing bookstore/book

Nginx will look for a file or directory named book inside **/var/www/bookstore/.**

- If book is a file (e.g., /var/www/bookstore/book.html), it will serve that file.
- If book is a directory (/var/www/bookstore/book/), it will check for an index.html inside /var/www/bookstore/book/.
- If no file or index is found, it returns a 404 Not Found.

3. Accessing bookstore/about or bookstore/product

Nginx follows the same logic:

- /var/www/bookstore/about.html → If exists, serves the file.
- /var/www/bookstore/about/index.html → If about/ is a directory, serves the index.
  Otherwise, returns 404.

- After making these changes, we can run our application on our Server's Public IP on port 80 (http) and not on our application's port as Nginx will forward the User Request to the our application's port securing the Application .

- Understanding directives and contexts :

  - if we dont have a trailing / in proxy pass after 5000, nginx will assume the url to be http://myapp.com/books/123 and forwards it to http://localhost:5000/books/123 keeping the book in the url.

    ```
    location /books/ {
        proxy_pass http://localhost:5000;
    }
    ```

  - conversely, if we have a trailing / in proxy pass, nginx removes /books/ and forwards it as: http://localhost:5000/123

```
location /books/ {
        proxy_pass http://localhost:5000/;
}
```

12 : We will now have to buy a domain name and then map our Server's Public IP to it. This will run our Application on <my-website>.com.

Inorder to check whether out application is running successfully on the domain name we created, we can check the header named **host** under chrome tools => network tab

13 : We can later enable https in this process by going to *letsencrypt.org* and use their certbot tool to install a SSL Certificate.

Install Certbot :

- sudo apt update
- sudo apt install certbot python3-certbot-nginx -y

Run Certbot to obtain an SSL Certificate :

- sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com

Enter the email Id and choose to redirect traffic to https rather than http.

Certbot will configure Nginx to run on https.

Ensure https connections are allowed on the VM by :

- sudo ufw allow 'Nginx Full'
- sudo systemctl restart nginx