Next.js deployment project

25/02/2023

In this project I will deploy Next.js(just frontend) to the Linux Ubuntu server. Application will be dockerized, created registry for image and Nginx will be used as reverse proxy, also I will register domain on Dynu.com and https will be created via certbot. Firewall will be activated for traffic protection and Ansible will be used to automate deployment process at the end.

This project will be decomposed to stages.

Stage 1

- Deploying Ubuntu server on linode platform and installing all the requirements(Docker, git, nginx) and also update it.
- Configuring Firewall

Stage 2

- Dockerizing next.js app
- Setting up local registry and nginx as reverse proxy

Stage 3

- Registering a domain and configuring HTTPS with certbot.
- Configure nginx to redirect http traffic to https
- Set up firewall on server to restrict access only to necessary ports and services

While creating this project I will take notes about problems that I encounter with and solution to those by stages.

Stage 1

I have deployed Ubuntu server on Linode platform, now I need to install git, Docker and nginx.

```
root@143.42.24.103
```

Nginx and git are installed, now I need to install Docker and installing docker is a little bit different and it depends from distro to distro. Our distribution on Linux is Ubuntu 20.04

I have installed everything now I need to learn more about setting firewall on linode, and after that to set on my machine.

I have set firewall only to accept inbound traffic from my IPv4 address and also limited SSH access to my IP address.

Tips for hardening access to server

1. You should always keep your Linux server up to date.

```
apt update


apt-dist upgrade -y
```

First command `apt update` updates all the packets that can be upgraded, and checks if there are some new packets and command which install updates is `apt dist-upgrade -y` .

Now we will automate installing updates so we do not need to do that manually. For this purpose we will need to install one package

```
apt install unattended-upgrades -y
```

I am already having this package so I only need to turn on this feature with command

```
dpkg-reconfigure --priority=low unattended-upgrades
```

2. Add a limited user account

```
useradd -m -s /bin/bash ognjen && passwd ognjen
```

With `useradd` command we are adding new user(-m/assigning user home directory -s/adding bash shell) and setting password.

After that add user to sudo group

```
usermod -aG sudo ognjen
```

Stage 2

Now I am trying to build effective dockerfile and by effective I mean to build an app faster and more reliable. So considering this I came up to interesting information about two different package managers for Node.js and those are yarn and npm.

The main difference between Yarn and npm is that yarn is more faster and reliable, especially when it comes to resolving package dependencies. Yarn also has some additional features such as the ability to install packages offline.

This is my initial Dockerfile

```
FROM node:16



WORKDIR /app



COPY . .



RUN npm install



EXPOSE 3000



CMD [ "npm", "run", "dev" ]
```

Since I am not a developer and I now this can be optimized I will use chatGPT for help.

What I get suggested is multi-stage build where I can separate *build environment* where all the dependencies are installed and *production environment* where runtime dependencies are installed.

This is optimized Dockerfile

```
#Build environment

FROM node:18-alpine AS build

WORKDIR /app

COPY package*.json ./

RUN npm ci

RUN npm run Build



#Production environment

FROM node:18-alpine AS Production

WORKDIR /app

COPY --from=build /app/package*.josn ./

RUN npm ci --only=production

COPY --from=build /app/.next ./.next

EXPOSE 3000
```

```
CMD ["npm","run","start"]
```

With stage build we separated build and production environment.

In the build env we copy the package.json file, install dependencies using `npm ci`, copy the rest of the app code, and run the build script.

In the second stage, we copy only the production dependencies from the build environment, copy the `.next` directory (which contains the built Next.js app), and start the app using `npm run start`.

Now using npm ci instead of npm install: The command `npm ci` is designed specifically for continuous integration (CI) environments where you want to ensure that exact same dependencies are installed evertime.

`npm ci` read package-lock.json file and installs the exact dependencies listed here.

I have totally changed the Dockerfile in comparison to one that chatGPT generates. This is my new Dockerfile which is 515MB of size.

```
# First stage #

FROM node:18-alpine as builder

ENV NEXT_PUBLIC_FOURSQUARE_API_KEY=fsq3snxM24jfqamDg26HA89t8YESgTqhC7kkE40jrM0ZjuY=

ENV NEXT_PUBLIC_UNSPLASH_ACCESS_KEY=8p9lC4mKntiOa0q50srgCcspkNBLFoXELHcdfvTBM4k

ENV AIRTABLE_API_KEY=keyieL30tupx8ItBM

ENV AIRTABLE_BASE_KEY=appEQyo6tgjzfYWIz



WORKDIR /app
```

```
COPY package*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Second stage #
FROM gcr.io/distroless/nodejs:18

WORKDIR /app

COPY --from=builder /app /app

ENV NEXT_PUBLIC_FOURSQUARE_API_KEY=fsq3snxM24jfqamDg26HA89t8YESgTqhC7kkE40jrM0ZjuY=
```

```
ENV NEXT_PUBLIC_UNSPLASH_ACCESS_KEY=8p9lC4mKntiOa0q50srgCcspkNBLFoXELHcdfvTBM4k

ENV AIRTABLE_API_KEY=keyieL30tupx8ItBM

ENV AIRTABLE_BASE_KEY=appEQyo6tgjzfYWIz



EXPOSE 3000



USER 1000



CMD ["./node_modules/next/dist/bin/next", "start"]
```

Yesterday I had problem with ENV variables that I needed to pass in during build of Docker image, firstly I used ENV because env stores values such as secrets, API keys, and database URL's. Also ENV should be present in second stage in order to be able to preform an api call.

I have placed them right under the parent image in Dockerfile and image and container are built without problem and it works. Now I will try to hide these ENV variables via Docker secrets.

Tips for securing Docker containers

1. Do not execute as ROOT user.
    1. By default many containers are configured to execute as root. Root user is needed because of the full permissions in order to install all the dependencies and build the app but we can change it after all the configuration is done. To change the user we can use command `USER node` towards the end of a Dockerfile.
2. Use a multi-stage build and distroless-image
    1. Since my container is base on alpine linux distro It is likely to have shell installed which opens space for attacker if he gain access to container he can use that shell to execute broader attack.Solution to this is using distroless image and we can

apply this by using multi stage build. Since user is no logner defined in distroless image we need to use numerical-user ID.
2. In the second stage I have changed the first layer and USER and way of running an app.

```
FROM gcr.io/distroless/nodejs:18

USER 1000

CMD ["./node_modules/next/dist/bin/next", "start"]
```

These two tips are about increasing security within contaier.

One important tip is that I needed to customize port rule so server can accept incoming request from my ip address on server port 3000.
Also I need to say that my image size is 505MB.

My next aim is to understand how local docker registry can imporve security of docker image and to set up local registry and configure nginx as reverse proxy.

Creating local docker registry

By default docker image is stored on the local Docker image cache on that server. The docker image cache is local storage area that docker uses to store all the images that have been pulled or built on that server.

```
docker run -d -p 80:80 --name nginx \
-v /etc/nginx2/nginx.conf : /etc/nginx/nginx.conf:ro \
-v /etc/nginx2/.htpasswd : /etc/nginx/conf.d/.htpasswd:ro \
--link registry:registry \
nginx
```

I have spin up two containers, one was for registry that was running on port 5000 and another one is for nginx server that was running on 8080 port, and I have done setup in nginx.conf file and later mount that to nginx.conf file in nginx container

So basically in steps it was like this :

- Spin up a registry container :

```
sudo docker run -d -p 5000:5000 --restart=always --name registry \
-v /var/lib/docker-registry:/var/lib/registry \
registry:2
```

Prior to this I have created directory docker-registry in /var/lib/

- Created on server nginx2 directory and inside I have created nginx.conf file where I did setup for nginx.reverse proxy :

```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    access_log /var/log/nginx/access.log;

    upstream registry {
        server registry:5000;
    }

    server {
        listen 8080;
        server_name coffeconnoisseur.top;

        location / {
            auth_basic "Restricted";
            auth_basic_user_file /etc/nginx/conf.d/.htpasswd;
            proxy_pass http://registry;
```

```
        }
    }
```

- Installed net tools

```
apt install net-tools
```

and then run this command for encrypting password

```
htpasswd -Bbn USERNAME PASSWORD > .htpasswd
```

Of course instead of username I have inserted arbitrary username and same goes for password

Those will be credentials for accessing docker registry, once I have finished this I started with nginx container

- Spin up nginx container :

```
docker run -d -p 8080:8080 --name nginx \
-v /etc/nginx2/nginx.conf : /etc/nginx/nginx.conf:ro \
-v /etc/nginx2/.htpasswd : /etc/nginx/conf.d/.htpasswd:ro \
--link registry:registry \
nginx
```

And later just tested from VM if I can push or pull and it works, of course it prompts for credentials before those actions

Stage 3

Domain register

I have registerd domain coffeeconnoisseur.top on NameSilo registrar. And I have added DNS A record for the root domain(@). Since It did not worked I needed to change the NS record or name server on Name silo to

- ns1.linode.com
- ns2.linode.com

- ns3.linode.com

and then to set domain in Linode console and everything now works fine, so when I hit coffeeconnoisseur.top:3000 it previews me the web app.

Certificate issude for HTTPS via Certbot

In order to finish certificate issue for https I needed to follow these steps :

1. Install snapd
   - Since I am using Ubuntu 20.04 distro, snapd comes pre installed on this distribution
2. Ensure that the snapd version is up to date by running next command
   - `snap install core`
3. Remove certbot-auto and any CertbotOS Packages
   - Since one way to use Certbot packages is to use the one provided by our OS package manager, while another way is to use Certbot snap package, we should remove Certbot packages provided by our OS since we are planning to use Certbot snap packages. We can remove by running command `apt-get remove certbot`
4. Install Certbot
   - We will install Certbot by running command `snap install --classic certbot`
5. Prepare the Certbo command
   - By executing this command we are ensuring that certbot command can be executed `ln -s /snap/bin/certbot /usr/bin/certbot`
6. Now by running this command we will get certificate and have Certbot edit our nginx configuration automatically to serve it, turning on HTTPS access in a single step
   - certbot --nginx
   - There is also option just to get certificate and make changes by hand to our nginx conf by running command `certbot certonly --nginx`

```
certbot --nginx -d coffeeconnoisseur.top
```

Still there shoul be done some settings in nginx.conf file

```
upstream coffeeconnoisseur.top {

    server 143.42.24.103:3000;

}



server {

    listen 443 ssl;

    root /root/next-coffee-stores;



    server_name coffeeconnoisseur.top;



    ssl_certificate  /etc/letsencrypt/live/coffeeconnoisseur.top/fullchain.pem;

    ssl_certificate_key  /etc/letsencrypt/live/coffeeconnoisseur.top/privkey.pem;



    location / {

    proxy_pass http://coffeeconnoisseur.top/;

    proxy_set_header Host $host;

    proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        }



    }
```

We are here creating a reverse proxy pass setup in nginx.conf file.

In the first context upstream I set directive server to the IP of my server and port on which docker container is running.

Next context is server with some directives like listen set to port 443, root directory of where our page is located and then server_name directive with domain name

```
ssl_certificate  /etc/letsencrypt/live/coffeeconnoisseur.top/fullchain.pem;

ssl_certificate_key  /etc/letsencrypt/live/coffeeconnoisseur.top/privkey.pem;
```

These two lines are telling us the location of ssl certificate,

and the location context, final one, is having / that indicates on root path and it is show us what is going to be preformed when we set the location of /.

first option is proxy_pass and it will redirect us to the IP add of our server serving on port 3000 so clients does not need to type coffeeconnoisseur.top:3000 in the url to reach our website.

And final three directives are :

1. proxy_set_header Host $host;
2. proxy_set_header X-Real-IP $remote_addr;
3. proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

So lets break this, basically those variables hold values that are being sent in the request header and it uses them to facilitate the job for server that handle the request.

So without host the upstream server might not be able to correctly handle the request if it is expecting request with specific hostname.

The $remote_addr variable hold value of IP of client that is sending the request and this can be useful for server especially if the request has passed through one or more proxies.

$proxy_add_x_forwarded_for variable - This header is used by the upstream server to determine the complete chain of IP addresses that the request has passed through.