AWS Docker Setup and EC2 Instance Deployment Guide

Prerequisites:

1. **Docker** must be installed on your server or local machine. If not, follow the installation instructions from Docker.
   https://docs.docker.com/get-started/get-docker/

2. **Docker Compose** is required to manage multi-container Docker applications. Install it from here.

   https://docs.docker.com/compose/install/

1. Create a Dockerfile

The **Dockerfile** defines the instructions to build your Docker container. It specifies the base image, installs dependencies, copies your application files, builds the application, and sets up the production environment. Below is an example Dockerfile for setting up an application with a React.js frontend and Nginx as the web server.

Example:

```
# Build stage

FROM node:20-alpine AS build

WORKDIR /app

# Copy package files and install dependencies

COPY package.json package-lock.json ./
```

```
RUN npm ci


# Copy the rest of the application code

COPY . .

# Build the application

RUN npm run build

# Production stage

FROM nginx:stable-alpine AS production

# Copy the built files from the build stage

COPY --from=build /app/dist /usr/share/nginx/html

# Copy custom nginx configuration

COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expose port 80

EXPOSE 80

# Start nginx

CMD ["nginx", "-g", "daemon off;"]
```

2. Create an Nginx Configuration File

Below is an example nginx.conf file, which configures Nginx for serving a single-page application (SPA) with gzip compression, caching, and security headers.

Example:

```
server {

    listen 80;

    server_name xpectrum-ai.com www.xpectrum-ai.com;
    root /usr/share/nginx/html;

    index index.html;

    # Gzip compression

    gzip on;

    gzip_types text/plain text/css application/json 
application/javascript text/xml application/xml 
application/xml+rss text/javascript;

    gzip_comp_level 6;

    gzip_min_length 1000;

    # Cache static assets

    location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg)$ {

        expires 30d;

        add_header Cache-Control "public, max-age=2592000";

    }

    # Handle SPA routing

    location / {

        try_files $uri $uri/ /index.html;

    }

    # Security headers

    add_header X-Content-Type-Options nosniff;

    add_header X-XSS-Protection "1; mode=block";

    add_header X-Frame-Options SAMEORIGIN;

}
```

3. Create a Docker Compose File

A **docker-compose.yml** file is used to define and run multi-container Docker applications. This file will define how to build and run your application using Docker Compose.

Example:

```
version: '3.8'


services:

  xpectrum-app:

    build:

      context: .

      dockerfile: Dockerfile

    container_name: xpectrum-app

    restart: unless-stopped

    ports:

      - "8081:80"

    environment:

       - NODE_ENV=production


    # Uncomment and configure if you need to mount volumes

    # volumes:

    #   - ./some-local-path:/app/some-container-path


    # Uncomment for health checks

    # healthcheck:

    #   test: ["CMD", "wget", "--spider", "-q",
"http://localhost:80"]

    #   interval: 30s

    #   timeout: 10s

    #   retries: 3

    #   start_period: 40s
```

4. Create a .dockerignore File

A `.dockerignore` file prevents unnecessary files from being included in your Docker image, making the build faster and cleaner.

5. Build and Run Your Application with Docker Compose

Once you've set up your files, follow these steps to build and run your Docker container:

1. **Navigate to your project directory**:

   ```
   cd /path/to/your/project
   ```

2. **Build and start the container**:

   ```
   docker-compose up -d
   ```

3. **Verify the container is running**:

   ```
   docker ps
   ```

4. **Check the logs to ensure everything started correctly**:

   ```
   docker-compose logs -f
   ```

5. **Rebuild and restart if necessary**:

   ```
   docker-compose down

   docker-compose up -d --build
   ```

EC2 Instance Deployment Setup

1. Install Node.js and NPM

Start by installing Node.js and npm on your EC2 instance.

```
curl -sL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install nodejs
node --version
```

2. Generate SSH Key for GitHub

Generate an SSH key that will be used to authenticate with GitHub:

```
ssh-keygen -t rsa -b 4096 -C "your-email@example.com"
```

- **This command creates a new SSH key pair for your EC2 instance.**
- **Replace `"your-email@example.com"` with your email address.**

Display the SSH public key:

```
cat ~/.ssh/id_rsa.pub
```

3. Add SSH Key to GitHub

To add the SSH key to GitHub, follow these steps:

1. Copy the public key output from the cat ~/.ssh/id_rsa.pub command.
2. Go to GitHub Settings > SSH and GPG Keys > New SSH Key.
3. Paste the copied key and save.

Test SSH connection with GitHub:

```
ssh -T git@github.com
```

4. Clone Your GitHub Repository

To clone your repository via SSH:

```
git clone git@github.com:Xpectrum-AI/main-website.git
```

5. Set Up Firewall

Enable and configure the firewall using UFW (Uncomplicated Firewall):

```
sudo ufw enable

sudo ufw status

sudo ufw allow ssh    # Allow SSH (Port 22)

sudo ufw allow http   # Allow HTTP (Port 80)

sudo ufw allow https  # Allow HTTPS (Port 443)
```

- ufw enable: Enables the firewall.

- ufw allow: Opens the required ports for SSH, HTTP, and HTTPS.

6. Install NGINX and Configure

Install NGINX on the EC2 instance:

```
sudo apt install nginx
```

Configure NGINX to proxy requests to your Node.js application:

```
sudo nano /etc/nginx/sites-available/default
```

Inside the configuration file, add the following configuration:

Example:

```
server {

    server_name xpectrum-ai.com www.xpectrum-ai.com;

    location / {

        proxy_pass http://localhost:8081;  # or
whatever port your app uses

        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;
    }


    listen 443 ssl; # managed by Certbot

    ssl_certificate
/etc/letsencrypt/live/xpectrum-ai.com/fullchain.pem;
# managed by Certbot

    ssl_certificate_key
/etc/letsencrypt/live/xpectrum-ai.com/privkey.pem; #
managed by Certbot

    include /etc/letsencrypt/options-ssl-nginx.conf;
# managed by Certbot

    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #
managed by Certbot

}


server {

    if ($host = www.xpectrum-ai.com) {

        return 301 https://$host$request_uri;

    } # managed by Certbot


    if ($host = xpectrum-ai.com) {

        return 301 https://$host$request_uri;

    } # managed by Certbot
    listen 80;
    server_name xpectrum-ai.com www.xpectrum-ai.com;
    return 404; # managed by Certbot
}
```

7. Check NGINX Configuration

To ensure the NGINX configuration is correct, run:

```
sudo nginx -t
```

8. Restart NGINX

Reload NGINX to apply the configuration changes:

```
sudo nginx -s reload
```

9. Install Docker and Docker Compose

Install Docker and Docker Compose:

```
sudo apt update
sudo apt install docker.io docker-compose -y
```

Enable Docker to start automatically on boot:

```
sudo systemctl enable docker
```

Add your user to the Docker group to avoid using sudo every time:

```
sudo usermod -aG docker $USER
```

10. Start Your Application with Docker Compose

Start the application using Docker Compose:

```
docker-compose up -d
```

11. Verify Docker Container

To check if your Docker container is running, use:

```
docker ps
```

Check if your app is accessible:

```
curl http://localhost:8081
```

12. Set Up SSL with Certbot

To set up SSL using Certbot, follow these steps:

```
sudo add-apt-repository ppa:certbot/certbot

sudo apt-get update
sudo apt-get install python3-certbot-nginx
```

Run Certbot to obtain and configure the SSL certificates:

```
sudo certbot --nginx -d xpectrum-ai.com -d
www.xpectrum-ai.com
```