Setting Up Docker, Nginx, PHP, and MySQL in a Docker Container on EC2

This guide will walk you through the process of setting up a Docker container with Nginx, PHP, and MySQL on an Amazon EC2 instance running Ubuntu 20.04.

0. Pre-requisites

- An existing domain name configured, in our case: www.wizardly-wilbur.cloud
- An existing EC2 server with ports 80 and 443 opened.
- Docker is installed on your EC2 server.

1. Stop Existing Services (Optional)

If you have existing services like Nginx, PHP, and MySQL running on your EC2 instance and want to stop them, use the following commands:

```
sudo systemctl stop nginx
sudo systemctl disable nginx
sudo systemctl stop php7.4-fpm
sudo systemctl disable php7.4-fpm
sudo systemctl stop mysql
sudo systemctl disable mysql
```

2. Install Docker

Follow the official Docker installation guide for Ubuntu to install Docker:

[Install Docker on Ubuntu]

Additionally, you can remove any conflicting packages by running:

```
for pkg in docker.io docker-doc docker-compose podman-docker containerd runc; do
    sudo apt-get remove $pkg
done
```

3. Add Docker's Official GPG Key and Repository

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
        /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo "deb [arch=\$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
        https://download.docker.com/linux/ubuntu \$(. /etc/os-release && echo
        \${VERSION_CODENAME}) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
        /dev/null

sudo apt-get update
```

4. Install Docker Packages

Install Docker packages:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
        compose-plugin -y
```

5. Verify Docker Installation

Check the Docker version to verify the installation:

```
sudo docker --version
```

6. Pull Ubuntu 20.04 Image

Pull the Ubuntu 20.04 image from the official Docker repository:

```
sudo docker pull ubuntu:20.04
```

7. Create a Directory for Mounting

Create a directory to mount with the Docker container later:

```
cd
mkdir web_docker_mount_directory
```

8. Create the Docker Container

Create the Docker container with a mounted directory:

```
sudo docker run -it \
--name Silver_Link_WebApp \
-d -p 443:443 -p 80:80 \
--shm-size=1g \
--ulimit memlock=-1 \
--ulimit stack=67108864 \
-v ~/web_docker_mount_directory:/web_docker_mount_directory \
ubuntu:20.04
```

9. Access the Docker Container

Access the Docker container via the terminal:

```
sudo docker exec -it Silver_Link_WebApp /bin/bash
```

10. Update and Upgrade Packages

Update and upgrade the packages inside the Docker container:

```
apt-get update
apt-get upgrade -y
```

11. Set Timezone

Set the timezone to Asia/Singapore:

```
apt-get install -y tzdata
ln -sf /usr/share/zoneinfo/Asia/Singapore /etc/localtime
dpkg-reconfigure -f noninteractive tzdata
```

12. Install Nginx

Install Nginx inside the Docker container:

```
apt-get install nginx -y
service nginx start
service nginx status
```

13. Install PHP and PHP-MySQL Extension

Install PHP and the PHP-MySQL extension:

```
apt-get install php-fpm php-mysql -y
service php7.4-fpm start
```

14. Configure Nginx for PHP

Copy the default Nginx server block configuration and edit it:

```
cp -a -v /etc/nginx/sites-available/default /etc/nginx/sites-available/www.wizardly-
        wilbur.cloud
apt-get install nano
nano /etc/nginx/sites-available/www.wizardly-wilbur.cloud
```

Edit the following lines in the Nginx configuration file:

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

        root /var/www/html;

        index index.php index.html index.htm index.nginx-debian.html;

        server_name www.wizardly-wilbur.cloud;

        location / {
                try_files $uri $uri/ =404;
        }

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/run/php/php7.4-fpm.sock;
        }
}
```

Enable the new server block and disable the default configuration:

```
ln -s /etc/nginx/sites-available/www.wizardly-wilbur.cloud /etc/nginx/sites-enabled/
rm /etc/nginx/sites-enabled/default
```

Validate the Nginx configuration:

```
nginx -t
```

Reload Nginx:

```
service nginx reload
```

15. Change Permissions (Optional/Not recommended)

Change permissions for the web directory:

```
chmod -R 777 /var/www/html
```

16. Create a PHP Test File

Create a simple PHP test file in Nginx’s web hosting directory:

```
echo "<?php phpinfo(); ?>" >> /var/www/html/info.php
```

17. Access PHP Info

Visit `http://www.wizardly-wilbur.cloud/info.php` in your browser to see the PHP info page. If you see the page, it means both Nginx and PHP services are up and running.

18. Install net-tools (Optional)

You can install net-tools if needed:

```
apt-get install net-tools
ifconfig
```

19. Set Up HTTPS (Optional)

To set up HTTPS using Let’s Encrypt, follow these steps:

```
apt-get install certbot -y
apt-get install certbot python3-certbot-nginx
certbot --nginx -d www.wizardly-wilbur.cloud
```

Answer the questions by certbot. #### 20. Configure Nginx for HTTP/2 (Optional)

Edit the Nginx server block configuration to enable HTTP/2:

```
nano /etc/nginx/sites-available/www.wizardly-wilbur.cloud
```

Edit the server block to include HTTP/2:

```
server {
    server_name www.wizardly-wilbur.cloud;
    location / {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    listen [::]:443 ssl http2 ipv6only=on; # managed by Certbot
    listen 443 ssl http2; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/www.wizardly-wilbur.cloud/fullchain.pem; #
managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/www.wizardly-wilbur.cloud/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
}
```

Validate the Nginx configuration:

```
nginx -t
```

Reload Nginx:

```
systemctl reload nginx
```

21. Enabling HTTP Strict Transport Security (HSTS) (Optional)

To enable HTTP Strict Transport Security (HSTS), open the Nginx main configuration file `/etc/nginx/nginx.conf` and add the following line:

```
http {
...
add_header Strict-Transport-Security "max-age=15768000; includeSubDomains" always;
}
```

Run the following commands to validate and reload the Nginx configuration:

```
nginx -t
systemctl reload nginx
```

22. Installing MySQL

Install MySQL server:

```
apt-get install mysql-server -y
service mysql start
mysql_secure_installation
```

Follow the prompts to configure MySQL's security settings. Remember the password you set for the MySQL root user.

23. Set MySQL Password Authentication Method If Previous Step Did Not Prompt For One(Optional)

You may need to set the authentication method for the MySQL root user to `mysql_native_password`. Connect to MySQL as the root user and run the following SQL commands:

```
mysql -u root
ALTER USER 'root'@'localhost' IDENTIFIED WITH 'mysql_native_password' BY 'your-password';
FLUSH PRIVILEGES;
exit;
```

24. Connecting to MySQL from Outside the Docker Container via MySQL Workbench (Deprecated at the moment, coz idk how to port-forward)

:(

25. Using a `db-config.ini` File (Optional)

You can store database connection details in a `db-config.ini` file to keep them separate from your code. Create the file and add your database configuration:

```
cd /var/www
mkdir private
nano private/db-config.ini
```

Edit the `db-config.ini` file:

```
[database]
servername = "localhost"
username = "root"
password = "your-password"
dbname = "<your-database-name>"
```

26. Connecting to the Database from PHP (Optional)

In your PHP code, you can use the `mysqli` extension to connect to the MySQL database. Here's an example code snippet:

```
<?php
// Create database connection.
$config = parse_ini_file('../private/db-config.ini');
$conn = new mysqli($config['servername'], $config['username'], $config['password'],
        $config['dbname']);

// Check connection.
if ($conn->connect_error) {
    $errorMsg = "Connection failed: " . $conn->connect_error;
    $success = false;
} else {
    $success = true;

    // Query to fetch records from VolunteerDetails table (example).
    $sql = "SELECT ID, Name, Phone_Number, Birth_Date FROM VolunteerDetails";
    $result = $conn->query($sql);

    // Check if there are records.
    if ($result->num_rows > 0) {
        echo "<table border='1'>";
        echo "<tr><th>ID</th><th>Name</th><th>Phone Number</th><th>Birth Date</th></tr>";

        // Output data of each row.
        while ($row = $result->fetch_assoc()) {
            echo "<tr>";
            echo "<td>" . $row["ID"] . "</td>";
            echo "<td>" . $row["Name"] . "</td>";
            echo "<td>" . $row["Phone_Number"] . "</td>";
            echo "<td>" . $row["Birth_Date"] . "</td>";
            echo "</tr>";
        }

        echo "</table>";
    } else {
        echo "No records found in VolunteerDetails table.";
    }

    // Close the database connection.
    $conn->close();
}
?>
```

27. Start Docker Services (If Restarted)

If you restart your Docker container, you can start the services inside the container using these commands:

```
sudo docker start Silver_Link_WebApp
sudo docker exec -it Silver_Link_WebApp /bin/bash
service nginx start
service php7.4-fpm start
service mysql start
```

Docker Troubleshooting Guide

List Images

To list Docker images on your system:

```
sudo docker image ls
```

List Containers

To list all Docker containers, both running and stopped:

```
sudo docker ps -a
```

Remove Images

To remove a Docker image, replace `<repository>` with the actual repository name or image ID:

```
sudo docker rmi <repository>
```

Remove Containers

To remove a Docker container, specify the name or container ID:

```
sudo docker rm <container_name>
```

List Running Containers

To list only running Docker containers:

```
docker ps
```

Start an Existing Container

To start an existing Docker container:

```
docker start -a <container_name>
```

Restart a Running Container

To restart a running Docker container:

```
docker restart <container_name>
```

Stop a Running Container

To stop a running Docker container:

```
docker stop <container_name>
```

Access the Terminal of a Running Container

To access the terminal of a running Docker container:

```
docker exec -it <container_name> /bin/bash
```

Monitor GPU Status

To monitor the status of NVIDIA GPUs within a Docker container:

```
nvidia-smi
```

Create a New Directory in a Docker Container

To create a new directory inside a running Docker container:

```
docker exec -it <container_name> mkdir /path/to/new/directory
```

List Directories in a Container

To list directories inside a running Docker container:

```
docker exec <container_name> ls <directory_path>
```

For example:

```
docker exec my_container ls /
```

Remove a Container (When Not Running)

To remove an existing Docker container, but only when it’s not running:

```
docker rm -v <container_name>
```

Copy Files Between Host and Container

To copy files or directories from the host machine to a Docker container:

```
docker cp /path/to/local/file_or_directory container_name:/path/inside/container
```

To copy files or directories from a Docker container to the host machine:

```
docker cp container_name:/path/inside/container /path/to/local/directory
```

Check Container Ports

To check the exposed ports of a running Docker container:

```
docker port <container_name>
```