How to run multiple applications on nginx in a single container and port with a different extension, along with three other containers postgres, mongo and redis.

Introduction:

Over the past few years, Docker has become a frequently used solution for deploying applications thanks to how it simplifies running and deploying applications in ephemeral containers. When you are using an application stack, for example, with vue.js, node, python, Nginx, postgres, redis, mongo and the Laravel framework, Docker can significantly streamline the setup process.

Docker Compose has further simplified the development process by allowing developers to define their infrastructure, including application services, networks, and volumes, in a single file. Docker Compose offers an efficient alternative to running multiple docker container create and docker container run commands.

Here we will build applications like python, nodejs, vue js and laravel with Nginx as the web server all inside Docker containers. We will define the entire stack configuration in a docker-compose file, along with configuration files for postgres, mongo, redis, and Nginx.

Pre-requisites

One Ubuntu 20.04 server.

- Docker installed.
- Docker Compose installed

Cloning the projects.

Note: only applicable if the respective projects are not on your local machine.

Make a directory with

- mkdir directory_name

Cd into the directory we just created

- Cd directory_name_thah_we_created

Create a directory for laravel, node, vue and python and clone it into the directories from the below links respectively.

https://github.com/Mohsin-Yousaf/Laravel_database.git

https://github.com/Mohsin-Yousaf/Node_database.git

https://github.com/Mohsin-Yousaf/Python_database.git

Create the vue project with the following command:

npm install -g @vue/cli
vue create my-vue-app
cd my-vue-app

Created a Docker file with the following command in the main directory pro where the projects are cloned.

- touch docker-compose.yml
- nano docker-compose.yml

The configuration uses Docker Compose to orchestrate the deployment and management of multiple interconnected containers. The depends_on directive ensures that the web container starts after the specified services (mysql, mongodb, postgres, and redis) are up and running.

In the docker compose write the following commands

In the docker-compose.yml we will first define the version, here we are taking 3.

Services: This section defines the different containers that will be part of the environment.

```yaml
version: '3'
services:
  mongodb:
    image: mongo
    container_name: mongodb
    ports:
      - "27017:27017"
    volumes:
      - mymongo-data:/data/db
```

1.1 Docker-compose

mongodb: This service uses the official MongoDB Docker image. It maps the host's port 27017 to the container's port 27017 and sets up a volume named mymongo-data for persisting MongoDB data.

```yaml
  postgres:
    image: postgres:latest
    container_name: postgres_db
    environment:
      POSTGRES_PASSWORD: adnan
      POSTGRES_DB: adnan
      POSTGRES_USER: adnan
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

1.2 Docker-compose

postgres: This service uses the official PostgreSQL Docker image. It maps the host's port 5432 to the container's port 5432. It also specifies environment variables for configuring the PostgreSQL database, such as the password, database name, and username. A volume named postgres_data is created for persisting PostgreSQL data.

```yaml
  redis:
    image: redis:latest
    container_name: redis_db
    ports:
      - "6379:6379"
```

1.3 Docker-compose

redis: This service uses the official Redis Docker image. It maps the host's port 6379 to the container's port 6379.

```
web:
  container_name: web
  depends_on:
    - mysql
    - mongodb
    - postgres
    - redis
  links:
    - mongodb:mongodb
    - postgres:postgres
    - redis:redis
  build: .
  restart: "always"
  ports:
    - "80:80"
```

1.4 Docker-compose

web: This service represents a custom web application container. It depends on the other services (mysql, mongodb, postgres, and redis). It's built using a Dockerfile located in the same directory as the docker-compose.yml file. The container is linked to the other services (mongodb, postgres, and redis) and restarts automatically. It maps the host's port 80 to the container's port 80.

```
volumes:
  postgres_data:
  mymongo-data:
```

1.5 Docker-compose

Volumes: This section defines named volumes that will be used for persisting data generated by the containers. Each service that uses a volume mounts the volume to a specific path inside the container. This ensures that data persists across container restarts.

postgres_data: This volume is used by the postgres service to persist PostgreSQL data.

mymongo-data: This volume is used by the mongodb service to persist MongoDB data.

Dockerfile:

Create a docker file in the same directory with nano Dockerfile.

Create a docker file in the same directory with the following commands.

- nano Dockerfile

2.1 Dockerfile

Below are the explanation of the above commands in the Dockerfile.

Each command is elaborated individually

Base Image (FROM ubuntu:20.04):

FROM ubuntu:20.04

The FROM instruction specifies the base image that your Docker image will be built upon. In this case, the base image is Ubuntu 20.04, a popular Linux distribution.

Environment Variable (ENV DEBIAN_FRONTEND=noninteractive):

```
ENV DEBIAN_FRONTEND=noninteractive
```

This line sets an environment variable named DEBIAN_FRONTEND to the value noninteractive. This variable is often used during package installations to prevent interactive prompts, making the installation non-interactive.

Package Installation (RUN apt-get update && apt-get install -y ...):

```
RUN apt-get update && apt-get install -y \ software-properties-common \ nginx \ composer \ curl \ nano
```

This sequence of commands updates the package repository (apt-get update) and then installs several software packages (software-properties-common, nginx, composer, curl, and nano) using apt-get install -y. The -y flag answers "yes" to any prompts during installation.

PHP Installation (RUN apt install -y php7.4-fpm ... php7.4-zip):

```
RUN apt install -y php7.4-fpm ... php7.4-zip
```

This series of apt install commands installs various PHP extensions and modules, such as php7.4-fpm, php7.4-bcmath, php7.4-cli, etc., which are necessary for PHP-based applications to work properly.

Python and Node.js Installation (RUN apt install -y python3-pip and RUN apt install -y npm):

```
RUN apt install -y python3-pip
```

```
RUN apt install -y npm
```

These commands install Python 3 and npm (Node.js package manager), which are commonly used for backend and frontend development, respectively.

Composer Installation (RUN curl -sS https://getcomposer.org/installer | php and related commands):

```
RUN curl -sS https://getcomposer.org/installer | php
```

```
 RUN mv composer.phar /usr/local/bin/composer
```

```
RUN composer global require laravel/installer
```

These commands download the Composer installation script using curl, then execute it using php. Composer is a popular dependency management tool for PHP. The subsequent commands move the Composer executable to a global location and install the Laravel installer using Composer.

Working Directory (WORKDIR /app):

```
WORKDIR /app
```

This command sets the working directory within the image to /app. Subsequent instructions will be executed in this directory.

Nginx Configuration (RUN rm /etc/nginx/sites-enabled/default and COPY ./nginx/default.conf /etc/nginx/sites-enabled/default.conf):

```
RUN rm /etc/nginx/sites-enabled/default COPY ./nginx/default.conf
/etc/nginx/sites-enabled/default.conf
```

These commands remove the default Nginx configuration and replace it with a custom Nginx configuration file copied from the local directory.

Copying Files (COPY . .):

```
COPY . .
```

This command copies the entire contents of the local directory (where the Dockerfile is located) into the /app directory within the image.

Permissions (RUN chmod -R 777 /app):

```
RUN chmod -R 777 /app
```

This command grants full read, write, and execute permissions to all files and directories in the /app directory. However, such permissive permissions can pose security risks and should be used with caution.

Python Dependencies (RUN pip3 install -r /app/py/requirements.txt):

```
RUN pip3 install -r /app/py/requirements.txt
```

This command installs Python dependencies specified in the requirements.txt file located in the /app/py directory.

Node.js and Vue.js (RUN curl -sL https://deb.nodesource.com/setup_16.x | bash - ...):

RUN curl -sL https://deb.nodesource.com/setup_16.x | bash - && \ apt-get update && \ apt-get install nodejs -y WORKDIR /app/vuee

```
RUN npm install
```

```
RUN npm run build
```

These commands install Node.js 16.x using the provided setup script and package manager. Subsequent commands navigate to the /app/vuee directory, install Node.js dependencies, and build a Vue.js application located in that directory.

Exposing Ports (EXPOSE 80):

```
EXPOSE 80
```

This instruction exposes port 80 within the container, allowing external communication on that port.

Startup Command (CMD ["sh", "-c", "..."]):

```
CMD ["sh", "-c", "service php7.4-fpm start & cd /app/py & python3 /app/py/app.py & cd /app/node_pro & npm install & node /app/node_pro/index.js & vuee cd /app/vuee & npm run serve & nginx -g 'daemon off;'"]
```

This command performs and combines multiple actions using shell commands. It starts several services and processes:

- It starts the PHP-FPM service.
- It navigates to the /app/py directory and runs a Python application.
- It navigates to the /app/node_pro directory, installs Node.js dependencies, and runs a Node.js application.
- It navigates to the /app/vuee directory, installs Vue.js dependencies, and runs a Vue.js application.
- Finally, it starts Nginx in the foreground to serve the application.

Nginx configuration:

Create a directory named nginx with the following command

- Mkdir nginx
- Cd nginx
- Nano default.conf

This Nginx configuration file defines various URL-based routing rules to direct incoming requests to different parts of your application stack. It listens for requests on port 80 and IPv6 port 80, responds to the hostname "localhost", and serves files from the specified root directory. The configuration includes reverse proxy settings for routing requests to PHP-FPM, Python, Node.js, and Vue.js application servers based on their respective paths. Keep in mind that Nginx configurations can vary greatly based on the specifics of your application and deployment environment.

Server Listening and Basic Settings:

```
listen 80;
listen [::]:81 default_server;
server_name localhost;
root /app/lara/public;
index index.php index.html index.htm;
```

3.1 Nginx

listen 80; and listen [::]:80 default_server;:

listen 80; listen

[::]:80 default_server;

server_name localhost;

These lines indicate that the server should listen on port 80 for IPv4 connections and on port 81 for IPv6 connections (default_server).

server_name localhost;: This defines the server name as "localhost", meaning this server block will respond to requests targeting the hostname "localhost".

root /app/lara/public;:

root /app/lara/public;

Specifies the root directory for this server block. This is the location where the files to be served are located. In this case, it's set to the /app/lara/public directory, which is commonly used for Laravel applications.

index index.php index.html index.htm;

index index.php index.html index.htm;: Lists the order of index files to be searched for

when a directory is accessed. In this case, it prioritizes index.php followed by other HTML-related files.

```
location /php {
    try_files $uri $uri/ /index.php?$query_string;
}

location ~ \.php$ {
    include fastcgi_params;
    fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
}
```

3.2 Nginx

location /php { ... }:

This block handles URLs starting with /php. If the requested file is not found, it forwards the request to index.php with the query string.
location ~ \.php$ { ... }: This block uses a regular expression to match URLs ending with .php. It includes FastCGI parameters for passing requests to PHP-FPM for processing.
Python and Node.js Proxies:

location /py { ... }: For URLs starting with /py, this block sets up a reverse proxy to the Python application running at http://127.0.0.1:5000/.

```
location /py {
    proxy_pass http://127.0.0.1:5000/;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Prefix /;
}
```

3.3 Nginx

location /node { ... }:

```
location /node {
    proxy_pass http://127.0.0.1:3000/;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Prefix /;
}
```

3.4 Nginx

For URLs starting with /node, this block sets up a reverse proxy to the Node.js application running at http://127.0.0.1:3000/.
Vue.js Application Proxy:

location /v { ... }:

```
location /v {
    proxy_pass http://127.0.0.1:8080/;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Prefix /;
}
```

3.5 Nginx

location ~ /\.ht { deny all; } Requests to URLs starting with /v are proxied to the Vue.js application server running at http://127.0.0.1:8080/.
Denying Access to Hidden Files:

location ~ /\.ht { deny all; }:

```
location ~ /\.ht {
    deny all;
}
```

3.6 Nginx

This location block uses a regular expression to match URLs containing .ht, which are typically used for server configuration files. Access to these files is denied.

Overall, the Nginx configuration is designed to route requests to different backend services based on their paths. It's forwarding requests to PHP-FPM, Python, Node.js, and Vue.js servers using proxy_pass. This configuration helps to properly distribute requests to the relevant parts of your application stack.

Note: This configuration assumes specific ports (e.g., 5000 for Python, 3000 for Node.js, 8080 for Vue.js), and you should ensure that these ports match the ports your services are actually running on. Additionally, Nginx configurations can be complex and are often tailored to the specific needs of the application, so adjustments may be necessary depending on your exact setup and requirements.