Dockerized Web Development Stack with Nginx, PHP, Laravel, Python, Node.js and Vue.js

We start with our docker file that contains

Base Image

The Dockerfile starts with the ubuntu:20.04 base image, which is based on Ubuntu 20.04 LTS.

```
FROM ubuntu:20.04
```

Environment Variables

Sets the DEBIAN_FRONTEND environment variable to noninteractive to prevent prompts during package installations.

```
ENV DEBIAN_FRONTEND=noninteractive
```

Update and Install Dependencies

Update the package list and install essential software packages including Nginx, Composer, Curl, and Nano.

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

Install PHP and Extensions

Install PHP 7.4 and a variety of PHP extensions required for web development.

```
RUN apt install -y php7.4-fpm php7.4-bcmath php7.4-cli php7.4-common php7.4-curl
php7.4-dev php7.4-fpm php7.4-gd php7.4-imagick
RUN apt install -y php7.4-opcache php7.4-pgsql php7.4-pspell php7.4-readline
php7.4-snmp php7.4-sqlite3 php7.4-ssh2 php7.4-xml
   php7.4-xsl php7.4-zip php7.4-mbstring php7.4-memcachephp7.4-mongodb
php7.4-mysql php7.4-redis
```

Install Python 3 and pip

Install Python 3 and pip for Python application development.

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

Install npm

Install npm (Node Package Manager) for managing Node.js packages.

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

Install Composer Globally

Download and install Composer globally for managing PHP dependencies.

```
RUN curl -sS https://getcomposer.org/installer | php
RUN mv composer.phar /usr/local/bin/composer
```

Set the Working Directory

Set the working directory inside the container to /app where all our projects like node.js,vue.js,laravel and python reside.

```
WORKDIR /app
```

Create Laravel Project

Create a new Laravel project using Composer.

```
RUN composer create-project --prefer-dist laravel/laravel
```

Install Laravel Installer Globally

Install the Laravel installer globally using Composer.

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

Nginx Configuration

Remove the default Nginx configuration and replace it with custom configuration files from the /config directory.

```
RUN rm -rf /etc/nginx/sites-enabled/*
COPY ./config/* /etc/nginx/sites-enabled/
```

Copy Application Files

Copy all files from the current directory (your project files) into /app directory.

```
COPY . .
```

Set Permissions

Set permissions to allow read, write, and execute access to the /app directory.

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

Install Python Dependencies

Install Python dependencies listed in requirements.txt for python project.

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

Set Working Directory for Vue.js

Change the working directory to /app/vue to work with Vue.js.

```
WORKDIR /app/vue
```

Install Vue.js Dependencies and Build

Install Vue.js dependencies and build the application.

```
RUN npm install
RUN npm run build
```

Expose Ports

Expose port 80 for Nginx to serve web content.

```
   EXPOSE 80
```

Start Services

Start various services including PHP-FPM, Python application, Node.js application, and Nginx web server.

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

This Dockerfile provides a comprehensive environment for web application development, combining multiple technologies and components into a single container.

Nginx Configuration file set for running   Node.js,Python,Vue.js and Laravel in nginx server

Server Block Initialization

The server block starts by initializing Nginx to listen on port 80 and respond to requests with the server name "localhost."

```
  server {
      listen 80;
      server_name localhost;
    }
```

listen 80: Configures Nginx to listen on port 80 for incoming HTTP requests.
server_name localhost: Sets the server name to "localhost," indicating that this block applies to requests directed at "localhost."

Setting Root and Default Index

The configuration specifies the root directory for serving content and sets the default index files.

```
  root /app/vue/dist/;
  index index.php index.html index.htm;
```

root: Sets the root directory for serving content to /app/vue/dist/.
index: Defines the order in which index files are checked when a directory is accessed.

PHP Location Block

This section configures how requests to paths starting with /php/ are handled.

```
  location /php/ {
      alias /app/laravel/public/;
      index index.php;
```

```
        try_files $uri $uri/ /index.php?$query_string;
   location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME /app/laravel/public/index.php;
        include fastcgi_params;
      }
   }
```

location /php/: Defines how requests to URLs starting with /php/ are processed.
alias: Maps requests to /php/ to the Laravel public directory.
location ~ \.php$: Handles PHP files within this location.
fastcgi_pass: Sets the PHP-FPM socket path.
Other directives handle PHP execution.

Vue.js Location Block

This section configures how requests to paths starting with /vue/ are handled.

```
   location /vue/ {
       alias /app/vue/dist/;
       index index.html;
       try_files $uri $uri/ /index.html?$query_string;
   }
```

location /vue/: Defines how requests to URLs starting with /vue/ are processed.
alias: Maps requests to /vue/ to the Vue.js distribution directory.
Other directives handle Vue.js application routing.

Node.js Location Block

This section configures how requests to paths starting with /node/ are handled.

```
   location /node/ {
       proxy_pass http://localhost:3000/;
       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;
   }
```

location /node/: Defines how requests to URLs starting with /node/ are processed.
proxy_pass: Proxies requests to a Node.js application running on localhost:3000.
Other directives are related to proxy settings.

Python Location Block

This section configures how requests to paths starting with /python/ are handled.

```
 location /python/ {
     proxy_pass http://localhost:5000/;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 }
```
location /python/: Defines how requests to URLs starting with /python/ are processed.
proxy_pass: Proxies requests to a Python application running on localhost:5000.
Other directives handle proxy headers.

This Nginx configuration allows you to host multiple web applications under different paths (/php/, /vue/, /node/, /python/) on a single Nginx server, with each path configured to route requests to the appropriate backend application or serve static files.

Docker Compose configuration
It defines a multi-container environment with services for MongoDB, PostgreSQL, Redis, and a custom web application of our docker file.

Docker Compose Version
The Docker Compose configuration starts by specifying the version of Docker Compose being used.
```
   version: '3'
```
version: '3': Indicates that this configuration is written in Docker Compose version 3 format.
Define Services
The services section defines individual containers and their configurations.
MongoDB Service
```
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db
```
mongodb: Defines a service named "mongodb."
image: mongo: Specifies the Docker image to use for MongoDB.
ports: Maps port 27017 from the host to the container.
volumes: Defines a volume named "mongo-data" for persisting MongoDB data.

PostgreSQL Service

```
  postgres:
    image: postgres:latest
    container_name: postgres
    environment:
      POSTGRES_PASSWORD: db_1122
      POSTGRES_DB: database
      POSTGRES_USER: user
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

postgres: Defines a service named "postgres."
image: postgres:latest: Specifies the Docker image to use for PostgreSQL.
environment: Sets environment variables for PostgreSQL.
ports: Maps port 5432 from the host to the container.
volumes: Defines a volume named "postgres_data" for persisting PostgreSQL data.

Redis Service

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

redis: Defines a service named "redis."
image: redis:latest: Specifies the Docker image to use for Redis.
ports: Maps port 6379 from the host to the container.

Web Service

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

web: Defines a service named "web." for our docker file that contain node.js ,vue.js , python and laravel application running on nginx server
depends_on: Specifies dependencies on other services (mongodb, postgres, redis).
links: Links the service to other containers for communication.
build: .: Builds the container using the Dockerfile in the current directory.
restart: always: Restarts the container automatically if it stops unexpectedly.
ports: Maps port 80 from the host to the container.

Define Volumes

The volumes section defines named volumes for persisting data.

```
volumes:
  postgres_data:
  mongo-data:
```

postgres_data, mongo-data: Named volumes for persisting data associated with MySQL, PostgreSQL, and MongoDB.

This Docker Compose configuration defines a multi-container environment for running MongoDB, PostgreSQL, Redis, and a custom web application, ensuring that they can communicate with each other.