**Syed Ali Mehdi**

**Documentation of Docker project**

Basic Steps followed:

1. Running the Ubuntu as a base image.
2. Installing basic packages for smooth usage of our applications basic of python and php
3. Then installing node package directly from the link using curl.
4. Then installing nodejs
5. Then installing composer and creating project
6. Then copying nginx conf file and other peojects to the working directory
7. Installing flask and express and vue library.
8. Making directories executable and exposing ports.
9. From the end points running all the services.
10. Nginx configuration also done and they are working perfectly with applications.

Explanation of “Dockerfile” is listed below;

1. **FROM ubuntu:22.04**

The foundational instruction in a Dockerfile, a script-like configuration used to build Docker container images. In this context, it signifies the base image upon which the new container image will be built.

1. **ENV DEBIAN\_FRONTEND=noninteractive**

command typically used in Linux-based Dockerfiles or shell scripts to set an environment variable named "DEBIAN\_FRONTEND" to the value "noninteractive". In the context of Debian-based package management systems like apt, this variable influences the behavior of package installation and configuration prompts. When set to "noninteractive", it indicates that no interactive user interface should be used during package installation, preventing any prompts that might require user input. This is particularly useful in automated environments or scripts where manual intervention is not desired. Essentially, this line ensures that package installations proceed without any user interaction, making the process smoother and suitable for unattended setups.

1. **RUN apt-get update && apt-get install -y \**

**php7.4-fpm php7.4-cli php7.4-mbstring php7.4-xml php7.4-zip php7.4-curl \**

**nginx composer curl nano python3**

The Ubuntu will get its updated packages from the server, after that we will be installing mandatory packages of php from the server –y flag means yes to all options asked during installation. Nginx is also being installed here which is the web server curl is we often use for testing either service is serving or not curl actually makes request to server and nano is the file editor and python3 is python package as we are running python service in our container.

1. **RUN apt-get -y install pip**

Here, "apt-get" is a package management command in Debian-based systems, and "-y" flag is employed to automatically answer "yes" to any prompts for confirmation. The "install" option specifies that the command is being used to install packages, and "pip" is the package being installed. Pip is a tool used to install and manage Python packages, which are pre-built code libraries, making it easier to incorporate existing code functionality into Python projects. Therefore, this command ensures that the "pip" package manager is installed within the container or system, enabling the seamless addition of external Python libraries and modules as needed.

1. **RUN curl -sL https://deb.nodesource.com/setup\_16.x | bash -**

Here, "curl" is used to retrieve the script from the specified URL, and the "-sL" flags ensure a silent download while following redirects. The retrieved script is then piped ("|") to the "bash" interpreter for execution. This script from NodeSource adds the appropriate package repository configuration to the system, allowing the system's package manager (such as "apt-get" on Debian-based systems) to install Node.js version 16.x and its associated npm (Node Package Manager) tool. This command facilitates the installation of a specific version of Node.js within the container or system, making it ready for running JavaScript applications on the server-side.

1. **RUN apt update**

This command is used to update the packages of Ubuntu fetch all the packages from the server.

1. **RUN apt-get install -y nodejs**

Here the package installation is being carried out, and "nodejs" is the specific package being installed. Node.js is a runtime environment that allows the execution of JavaScript code on the server-side, enabling developers to build scalable and performant network applications. By running this command, Node.js is added to the system, making it possible to develop and run JavaScript-based applications, such as web servers or APIs, within the environment of the container or system.

1. **RUN sed -i 's/;clear\_env = no/clear\_env = no/' /etc/php/7.4/fpm/pool.d/www.conf**

The text within single quotes indicates a search-and-replace operation: it searches for the line containing ";clear\_env = no" and replaces it with "clear\_env = no". This modification likely alters the PHP-FPM configuration to prevent the clearing of environment variables before processing requests, which can be useful for maintaining certain environment settings across PHP requests. By employing this command, the PHP-FPM configuration is adjusted according to the specified change, ensuring that the desired behaviour is implemented when the Docker container or system is running PHP applications.

1. **RUN composer create-project --prefer-dist laravel/laravel /app/laravel**

Here, "composer" refers to Composer, a PHP dependency management tool used to manage libraries and packages. The "create-project" command creates a new project based on a specified package or template. In this case, "--prefer-dist" instructs Composer to prioritize downloading pre-built distribution archives rather than cloning from a version control system. The package being used is "laravel/laravel," which is the Laravel framework's official template for creating new applications. The command concludes by specifying the target directory "/app/laravel," where the new Laravel application will be set up. This line automates the process of creating a fresh Laravel application within the container, making it easier to deploy Laravel-based web projects.

1. **COPY ./nginx/nginx-site.conf /etc/nginx/sites-available/default**

The command facilitates the configuration of the Nginx server by replacing the default configuration with the content of the provided "nginx-site.conf" file, enabling custom settings for the web server's behaviour and virtual host setup. This ensures that the Nginx server within the container or system will use the specified configuration when serving web content

1. **RUN composer create-project laravel/laravel:^9.0 example-app**

Here, "composer" refers to the Composer tool, and "create-project" is a Composer command that generates a new project based on a specified package or template. The package being utilized is "laravel/laravel," which is the Laravel framework's official template for creating applications. The ":^9.0" version constraint indicates that Composer should install a version of Laravel 9.0 or higher, allowing for compatibility with Laravel's new features and improvements. The command concludes by specifying "example-app" as the project directory where the new Laravel application will be set up. This line automates the creation of a Laravel application within the container, making it convenient to deploy and develop Laravel-based web projects using the desired version of the framework.

1. **RUN pip3 install Flask**

Here, "pip3" is the Python package installer for Python 3, and "install" is the command used to install Python packages. The package being installed is "Flask," which is a popular web framework used to develop web applications in Python. By using this command, the Flask framework is added to the container or system's Python environment, enabling developers to create web applications using Flask's features and capabilities.

1. **RUN chmod 777 -R /var**

Here, "chmod" is a command that changes permissions on files and directories, and "777" is a permission setting that grants read, write, and execute permissions to all users. The "-R" flag indicates that the operation should be applied recursively to all subdirectories and files within "/var".

1. **WORKDIR /app**

Changing the workspace to “/app” location so we will be working in that directory.

1. **COPY . .**

This will copy all the content which is present around the docker file and copy to the workspace or location which we have selected and working in it.

1. **RUN chmod 777 -R /app**

Here, "chmod" is a command that changes permissions on files and directories, and "777" is a permission setting that grants read, write, and execute permissions to all users. The "-R" flag indicates that the operation should be applied recursively to all subdirectories and files within "/app".

1. **WORKDIR /app/vueapp**

Changing the workspace to “/app/vueapp” location so we will be working in that directory.

1. **RUN npm install**

The "install" command is an npm command used to install project dependencies listed in the "package.json" file, creating the "node\_modules" directory and populating it with the required packages. This command is a crucial step in setting up a Node.js application, ensuring that all necessary libraries are available for the application to run correctly.

1. **RUN npm install vue@3.2.26**

Here, "npm" refers to the Node Package Manager, used for managing JavaScript packages and libraries. In this case, it installs version 3.2.26 of the Vue.js framework, a popular JavaScript framework for building user interfaces.

1. **RUN npm run build**

This will build the vue project and then we will map this with nginx web server.

1. **EXPOSE 80 3000 5000**

Here ports of container will be exposed to application so that they can use and say and nginx can listen complete progress of application will be done.

1. **CMD ["sh", "-c", "service php8.1-fpm start & python3 /app/app.py & node /app/node/index.js & nginx -g 'daemon off;'"]**

This is endpoint of a container where the command is a complex shell command that starts multiple services concurrently within the container.

**Docker Compose file**

postgres:

    image: postgres:13

    container\_name: mypostgres

    environment:

      POSTGRES\_DB: mydb

      POSTGRES\_USER: myuser

      POSTGRES\_PASSWORD: mypassword

    networks:

      - mynetwork

    ports:

      - 5432:5432

    volumes:

      - postgres\_data:/var/lib/postgresql/data

The provided configuration is written in YAML format and defines a Docker container for a PostgreSQL database using the "postgres:13" image. The container is given the name "mypostgres" and is associated with specific environment variables. The "POSTGRES\_DB" variable is set to "mydb," indicating the name of the database that will be created inside the PostgreSQL instance. Similarly, "POSTGRES\_USER" is set to "myuser" and "POSTGRES\_PASSWORD" is set to "mypassword," specifying the credentials for accessing the database.

The configuration is linked to a network named "mynetwork," which facilitates communication between containers on the same network. The container's port 5432 is mapped to the host system's port 5432, allowing external access to the PostgreSQL database. Additionally, a volume named "postgres\_data" is defined and mounted to the "/var/lib/postgresql/data" directory within the container, ensuring that the database data persists even if the container is stopped or deleted.

mongodb:

    image: mongo:latest

    container\_name: mymongodb

    ports:

      - 27017:27017

    environment:

      MONGO\_INITDB\_ROOT\_USERNAME: admin

      MONGO\_INITDB\_ROOT\_PASSWORD: adminpassword

    networks:

      - mynetwork

    volumes:

      - mongo\_data:/data/db

The provided YAML configuration outlines the setup of a Docker container for a MongoDB database using the "mongo:latest" image. The container is named "mymongodb" and is configured with the latest version of the MongoDB image available. The configuration includes port mapping, with the container's port 27017 being mapped to the host system's port 27017, enabling external access to the MongoDB instance.

Environment variables are defined to configure the initial setup of the MongoDB database. The "MONGO\_INITDB\_ROOT\_USERNAME" is set to "admin," designating the root username for administrative access. Similarly, "MONGO\_INITDB\_ROOT\_PASSWORD" is set to "adminpassword," specifying the corresponding password for administrative authentication.

The container is linked to the "mynetwork" network, which facilitates communication between containers within the same network. Additionally, a volume named "mongo\_data" is defined and mounted to the "/data/db" directory within the container. This ensures that the database data persists even if the container is stopped or removed, maintaining data integrity and preventing data loss.

redis:

    image: "redis:latest"

    container\_name: redis\_db

    networks:

      - mynetwork

    ports:

      - "6379:6379"

The provided YAML configuration outlines the setup of a Docker container for a Redis database using the "redis:latest" image. The container is given the name "redis\_db" and is configured to use the latest version of the Redis image available. The configuration defines the container's connection to the "mynetwork" network, enabling communication between containers within the same network.

The Redis container's port mapping is established, with the container's port 6379 being mapped to the host system's port 6379. This mapping allows external access to the Redis database service. By default, Redis uses port 6379 for client communication, and this configuration ensures that clients outside the container can interact with the Redis instance hosted within it.

volumes:

   mongo\_data:

   mysql\_data:

   postgres\_data:

networks:

  mynetwork:

    external: true

The provided YAML snippet defines three named volumes, "mongo\_data," "mysql\_data," and "postgres\_data," which are intended to store persistent data for MongoDB, MySQL, and PostgreSQL containers, respectively. These volumes allow data to persist beyond the lifecycle of containers, ensuring data integrity and availability.

Additionally, a network named "mynetwork" is defined with the "external: true" attribute. This indicates that the network is not created within the current Docker Compose configuration, but rather it is expected to be pre-existing and available for use. By connecting containers to this external network, they can communicate with each other securely and efficiently.

**Nginx Configuration file**

server\_name  localhost;

    root /app/vueapp/dist;

    index index.php index.html index.htm;

    location / {

        try\_files $uri $uri.html $uri/ /index.html;

    }

In this nginx will go to the vueapp and show the vue.js default page and this will show the page present on default url localhost

location /python/ {

        proxy\_pass http://0.0.0.0:5000/;

    }

This will redirect to the flask service which is running on the port 5000 of the container

 location /node/ {

        proxy\_pass http://0.0.0.0:3000/;

    }

This will redirect to the node service which is running on the port 3000 of the container

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/php8.1-fpm.sock;

                fastcgi\_index index.php;

                fastcgi\_param SCRIPT\_FILENAME /app/laravel/public/index.php;

                include fastcgi\_params;

            }

    }

The provided configuration block is part of an Nginx server setup and handles requests directed to a "/php/" URL path. It uses the "alias" directive to point to a Laravel application's "public" directory for serving PHP content. The "index" directive sets "index.php" as the default index file. The "try\_files" directive helps handle URL routing by attempting to find matching files or forwarding the request to Laravel's "index.php" for processing.

For PHP files, the configuration employs a nested location block. It specifies "fastcgi\_pass" to connect to PHP-FPM using the specified socket path. The "fastcgi\_param" directive defines the PHP script's filename and includes common FastCGI parameters. This configuration allows Nginx to correctly route PHP requests to the Laravel application's entry point while using PHP-FPM for processing PHP code.