Modelos diferentes de dockerfile:

Cuatro ejemplos de Dockerfiles para ilustrar diferentes casos de uso y conceptos básicos en Docker.

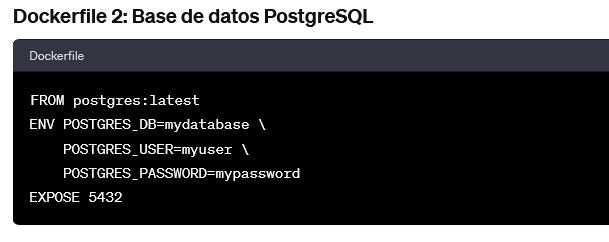


**Explicación:**

* Este Dockerfile crea un contenedor para una aplicación web simple escrita en Python con el framework Flask.
* Se utiliza la imagen base de Python 3.8.
* Las dependencias se instalan utilizando un archivo requirements.txt.
* Se expone el puerto 5000 para acceder a la aplicación web.
* Se copian los archivos de la aplicación al contenedor y se ejecuta la aplicación al iniciar el contenedor.

Atentos en conclusión:

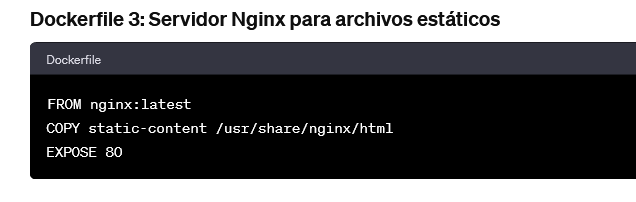
1. **FROM python:3.8**: Utiliza la imagen base oficial de Python versión 3.8 desde Docker Hub.
2. **WORKDIR /app**: Establece el directorio de trabajo dentro del contenedor en /app.
3. **COPY requirements.txt .**: Copia el archivo requirements.txt del directorio local al directorio /app en el contenedor.
4. **RUN pip install --no-cache-dir -r requirements.txt**: Instala las dependencias especificadas en el archivo requirements.txt. La opción --no-cache-dir evita la caché de pip para reducir el tamaño de la imagen.
5. **COPY . .**: Copia el contenido del directorio local (que contiene el código fuente de la aplicación) al directorio /app en el contenedor.
6. **EXPOSE 5000**: Documenta que la aplicación dentro del contenedor escuchará en el puerto 5000.
7. **CMD ["python", "app.py"]**: Especifica el comando a ejecutar cuando se inicie el contenedor. En este caso, se ejecutará la aplicación Python llamada app.py.



**Explicación:**

* Este Dockerfile crea un contenedor para una instancia de base de datos PostgreSQL.
* Utiliza la imagen base oficial de PostgreSQL.
* Define variables de entorno para configurar la base de datos.
* Expone el puerto 5432, que es el puerto por defecto de PostgreSQL.

1. **FROM postgres:latest**: Utiliza la imagen base oficial de PostgreSQL desde Docker Hub.
2. **ENV POSTGRES\_DB=mydatabase \ POSTGRES\_USER=myuser \ POSTGRES\_PASSWORD=mypassword**: Define variables de entorno que serán utilizadas por el contenedor de PostgreSQL para configurar la base de datos.
3. **EXPOSE 5432**: Documenta que el contenedor estará escuchando en el puerto 5432, que es el puerto por defecto de PostgreSQL.



**Explicación:**

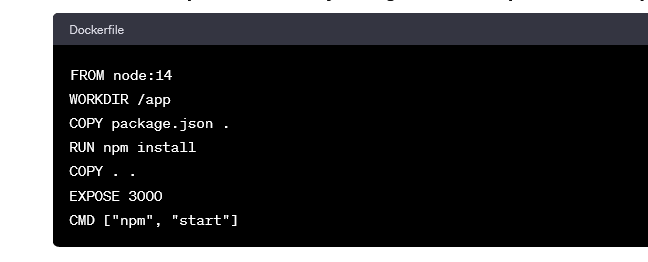
* Este Dockerfile crea un contenedor con un servidor Nginx para servir archivos estáticos.
* Utiliza la imagen base oficial de Nginx.
* Copia los archivos estáticos al directorio de contenido de Nginx.
* Expone el puerto 80 para acceder al servidor.

Es decir

1. **FROM nginx:latest**: Esta línea indica que estamos utilizando la imagen base oficial de Nginx desde Docker Hub. nginx:latest se refiere a la última versión disponible de la imagen de Nginx. Esta línea crea la primera capa de nuestro contenedor.
2. **COPY static-content /usr/share/nginx/html**: Esta línea copia el contenido del directorio local llamado static-content al directorio /usr/share/nginx/html dentro del contenedor. En este caso, estamos asumiendo que el directorio local static-content contiene los archivos estáticos que queremos servir. Esta línea crea una nueva capa en el contenedor.
3. **EXPOSE 80**: Esta línea especifica que el contenedor estará escuchando en el puerto 80. Sin embargo, ten en cuenta que esta instrucción no publica el puerto en el host. Simplemente documenta que el contenedor espera tráfico en el puerto 80. La exposición real del puerto debe hacerse al ejecutar el contenedor con la opción -p o --publish desde la línea de comandos.

Atentos en conclusión, este Dockerfile construye un contenedor que utiliza la imagen base oficial de Nginx, copia archivos estáticos locales al directorio de contenido de Nginx en el contenedor, y expone el puerto 80 para que pueda recibir tráfico. Este contenedor sería adecuado para servir contenido estático a través de un servidor Nginx.





**Explicación:**

* Este Dockerfile crea un contenedor para una aplicación Node.js con una gestión de dependencias separada.
* Utiliza la imagen base de Node.js 14.
* Copia solo el archivo package.json para aprovechar la caché de capas de Docker durante la instalación de dependencias.
* Luego, copia el resto de los archivos de la aplicación y realiza la instalación de dependencias.
* Expone el puerto 3000 y especifica el comando para ejecutar la aplicación al iniciar el contenedor.

En conclusión:

* **FROM node:14**: Utiliza la imagen base de Node.js versión 14 desde Docker Hub.
* **WORKDIR /app**: Establece el directorio de trabajo dentro del contenedor en /app.
* **COPY package.json .**: Copia únicamente el archivo package.json del directorio local al directorio /app en el contenedor.
* **RUN npm install**: Instala las dependencias utilizando npm.
* **COPY . .**: Copia el resto de los archivos de la aplicación desde el directorio local al directorio /app en el contenedor.
* **EXPOSE 3000**: Documenta que la aplicación dentro del contenedor escuchará en el puerto 3000.
* **CMD ["npm", "start"]**: Especifica el comando a ejecutar cuando se inicie el contenedor. En este caso, se ejecutará el script de inicio definido en el archivo package.json a través de npm start.