las etapas básicas en la creación de un Dockerfile.

Un Dockerfile es un script de texto plano que contiene instrucciones para construir una imagen de Docker.

Parte A.

Aquí hay algunas etapas comunes:

1. \*\*Seleccionar una imagen base:\*\*

- Elige una imagen base para tu aplicación. Puedes seleccionar una imagen oficial desde el

[Docker Hub](https://hub.docker.com/) según las necesidades de tu aplicación (por ejemplo, una imagen de Node.js, Python, Alpine Linux, etc.).

```Dockerfile

FROM node:14

```

2. \*\*Configurar el entorno:\*\*

- Configura variables de entorno necesarias para tu aplicación.

```Dockerfile

ENV NODE\_ENV=production

```

3. \*\*Copiar el código fuente:\*\*

- Copia los archivos de tu aplicación al sistema de archivos de la imagen.

```Dockerfile

WORKDIR /usr/src/app

COPY . .

```

4. \*\*Instalar dependencias:\*\*

- Ejecuta comandos para instalar las dependencias necesarias.

```Dockerfile

RUN npm install

```

5. \*\*Exponer puertos:\*\*

- Especifica los puertos en los que tu aplicación estará escuchando.

```Dockerfile

EXPOSE 3000

```

6. \*\*Comandos de inicio:\*\*

- Define el comando que se ejecutará cuando el contenedor se inicie.

```Dockerfile

CMD ["npm", "start"]

```

7. \*\*Construir la imagen:\*\*

- Utiliza el comando `docker build` para construir la imagen a partir del Dockerfile.

```bash

docker build -t nombre-de-tu-imagen:tag .

```

8. \*\*Ejecutar el contenedor:\*\*

- Utiliza el comando `docker run` para ejecutar el contenedor a partir de la imagen que has construido.

```bash

docker run -p 3000:3000 nombre-de-tu-imagen:tag

```

9. \*\*Optimizar la imagen (opcional):\*\*

- Puedes realizar pasos adicionales para optimizar la imagen, como reducir capas, limpiar archivos temporales, etc.

```Dockerfile

FROM node:14 as builder

WORKDIR /usr/src/app

COPY . .

RUN npm install

RUN npm run build

FROM node:14

WORKDIR /usr/src/app

COPY --from=builder /usr/src/app/dist ./dist

COPY package\*.json ./

RUN npm install --only=production

EXPOSE 3000

CMD ["npm", "start"]

```

Estas son solo las etapas básicas para tu Dockerfile

ParteA1.

Información sobre el uso de puertos

Puedes utilizar varios comandos y herramientas para verificar si un puerto está siendo utilizado por otro servicio en un sistema basado en Unix, como Ubuntu.

A continuación se de describen algunas opciones:

### 1. Comando `netstat`:

El comando `netstat` muestra información sobre conexiones de red, tablas de enrutamiento, estadísticas de interfaz, masquerading, etc. Para verificar si un puerto está ocupado, puedes utilizar:

``` terminal

sudo netstat -tulpn | grep :PUERTO

```

Reemplaza "PUERTO" con el número de puerto que deseas verificar. Este comando mostrará la información sobre el proceso que está utilizando ese puerto.

### 2. Comando `lsof`:

El comando `lsof` (list open files) muestra información sobre los archivos abiertos por procesos. Puedes usarlo para verificar el proceso que está utilizando un puerto específico:

``` terminal

sudo lsof -i :PUERTO

```

### 3. Comando `ss`:

El comando `ss` (socket statistics) proporciona información sobre sockets en Linux. Para verificar el estado de un puerto, puedes usar:

``` terminal

sudo ss -ltn | grep :PUERTO

```

### 4. Comando `fuser`:

El comando `fuser` identifica procesos que están utilizando archivos o sockets. Puedes usarlo para encontrar el proceso que utiliza un puerto:

``` terminal

sudo fuser PUERTO/tcp

```

### 5. Herramienta `nc` (netcat):

Puedes usar `nc` para verificar la conexión a un puerto específico:

``` terminal

nc -zv localhost PUERTO

```

Esto intentará establecer una conexión con el puerto y mostrará si está abierto.

### 6. Herramienta `telnet`:

Si está instalado, puedes usar `telnet` para verificar la conexión a un puerto:

```terminal

telnet localhost PUERTO

```

Parte B.

**Laboratorio 3:**

**Creación de un Dockerfile, Construcción de una Imagen, Publicación en Docker Hub y Gestión de Contenedores**

#### **Paso 1: Preparación (**opcional**)**

Comprobar la instalación de Docker en las máquinas de clase.

docker --version

#### **Paso 2: Crear un Directorio para el Proyecto**

Cada estudiante crea un nuevo directorio para su proyecto.

mkdir mi\_proyecto\_docker

cd mi\_proyecto\_docker

#### **Paso 3: Crear Archivos HTML**

Dentro del directorio del proyecto, cada estudiante crea tres archivos HTML con nombres propios

(por ejemplo, `index\_andrei.html`, `index\_elena.html`, `index\_arturo.html`, opcional incluye imágenes de mascotas favoritas, utiliza un sitio web libre de derechos para descargar una imagen, ejemplo pixabay.com o unsplash.com).

#### **Paso 4: Crear el Dockerfile**

En el mismo directorio, cada estudiante crea un archivo llamado `Dockerfile` con el siguiente contenido:

```Dockerfile

FROM nginx:latest

COPY index\_juan.html /usr/share/nginx/html/index.html

COPY index\_maria.html /usr/share/nginx/html/index\_maria.html

COPY index\_pedro.html /usr/share/nginx/html/index\_pedro.html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

#### Paso 5: Construir la Imagen

Cada estudiante ejecuta el siguiente comando para construir la imagen:

```terminal

docker build -t nombre-de-usuario/mi\_proyecto\_docker:tag .

```

El ejemplo de mi terminal de VSC, para este paso:

El comando:

docker build -t duscam23/prueba-crear-imagen-andr\_docker:0.1 "C:\Users\andri\Desktop\23\_24\_Benigasló\Entornos de Desarrollo\Proyecto1DockerfileAnd23"

Donde para el comando

1. `**docker build`:**Este es el comando principal de Docker para construir imágenes.

2. `-t duscam23/prueba-crear-imagen-andr\_docker:0.1`: Aquí, usas la opción `**-t**` para etiquetar (tag) la imagen que estás construyendo. En este caso, estás asignando el nombre `duscam23/prueba-crear-imagen-andr\_docker` a la imagen y la etiqueta `0.1` para identificar la versión.

**‘duscam23’**- mi nombre de usuario del dockerhub

3. "**C:\Users\andri\Desktop\23\_24\_Benigasló\Entornos de Desarrollo\Proyecto1DockerfileAnd23**" -

Esta es *la ruta* al contexto de construcción. El contexto es el conjunto de archivos y directorios que Docker enviará al daemon para construir la imagen.

En este caso, estoy especificando la ruta del directorio que contiene el Dockerfile y otros archivos necesarios para construir la imagen. Están en mi carpeta.

Finalmente, con este comando estoy instruyendo a Docker para construir una imagen utilizando el Dockerfile ubicado en el directorio especificado y etiquetarla con el nombre `**duscam23/prueba-crear-imagen-andr\_docker`** y la versión `**0.1**`. Este es un paso fundamental en el proceso de construcción de imágenes Docker antes de ejecutar contenedores basados en esas imágenes.

Por ejemplo, si este Dockerfile se usa para construir una imagen y luego se ejecuta un contenedor basado en esa imagen, el contenedor iniciará Nginx con la configuración especificada por el comando `CMD`.

Además

El comando `CMD` en un Dockerfile se utiliza para especificar el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en la imagen. La forma básica del comando `CMD` es:

```archivo dockerfile

CMD [comando, argumento1, argumento2, ...]

```

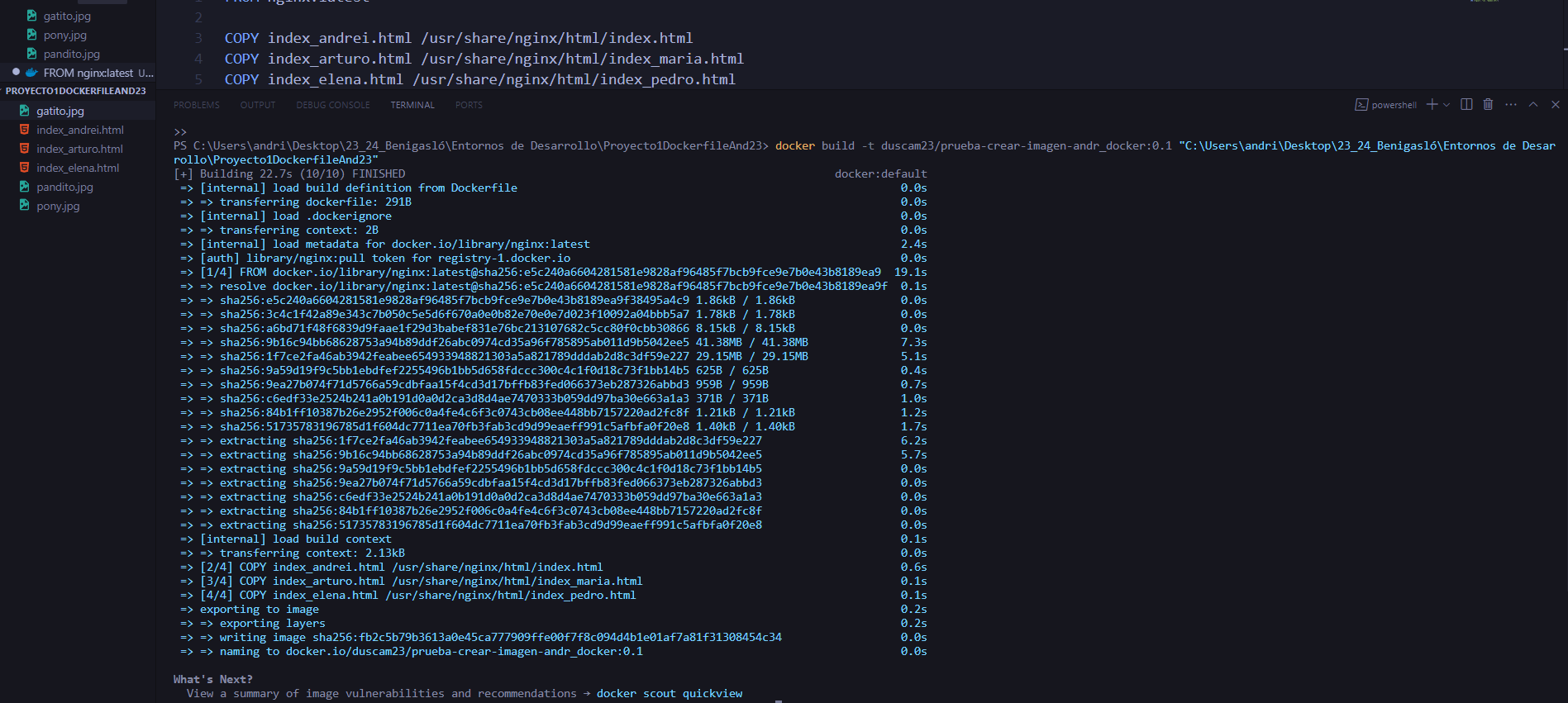
En mi caso específico, el comando `CMD` está configurado para iniciar el servidor Nginx en el contenedor con los argumentos "-g daemon off;". Desglosemos cada parte del comando:

1. \*\*`nginx`:\*\* Este es el comando principal que se ejecutará en el contenedor. En este caso, se está iniciando el servidor web Nginx.

2. \*\*`-g`:\*\* Este es un argumento para el comando Nginx. En este contexto, se utiliza para proporcionar una directiva de configuración al servidor.

3. \*\*`daemon off;`:\*\* Esta es la directiva de configuración específica que se pasa al servidor Nginx. En este caso, se establece `daemon` en `off`, lo que significa que el servidor Nginx se ejecutará en el primer plano (foreground) en lugar de en el fondo (background). Esto es común en entornos de contenedor donde los procesos deben ejecutarse en el primer plano para que el contenedor no salga cuando el proceso principal finaliza.

La respuesta:



Importante: asegúrate que tu archivo dockerfile no tiene extensión:

Si quieres cambiar la extensión del archivo "Untitled-1.dockerfile" a "Dockerfile". Puedes hacerlo utilizando el comando Rename-Item en PowerShell. Aquí está el comando que puedes usar:

Terminal visual studio code

*Rename-Item -Path .\Untitled-1.dockerfile -NewName Dockerfile*

Este comando cambiará el nombre del archivo "Untitled-1.dockerfile" a "Dockerfile"

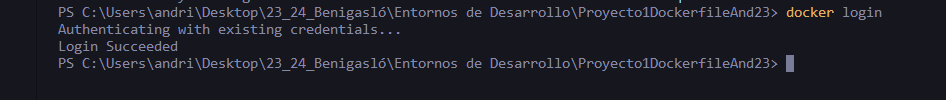
#### Paso 6: Iniciar Sesión en Docker Hub

Cada estudiante inicia sesión en Docker Hub desde la línea de comandos:

```terminal de visual estudio code:

docker login

```

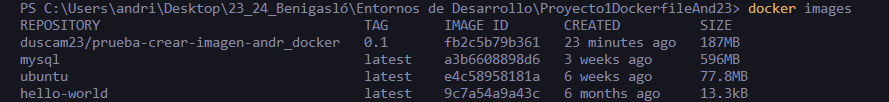


Paso 7: Subir la Imagen a Docker Hub

Cada alumno/a sube su imagen al repositorio en Docker Hub:

```terminal de visual estudio code:

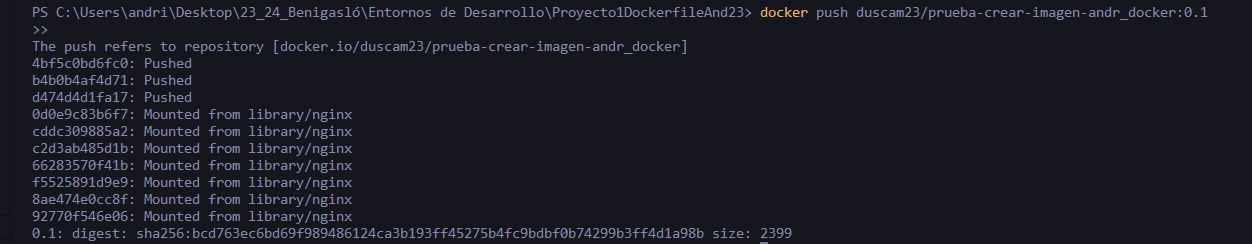
Paso opcional, comprueba las imágenes:



docker push nombre-de-usuario/mi\_proyecto\_docker:tag

```

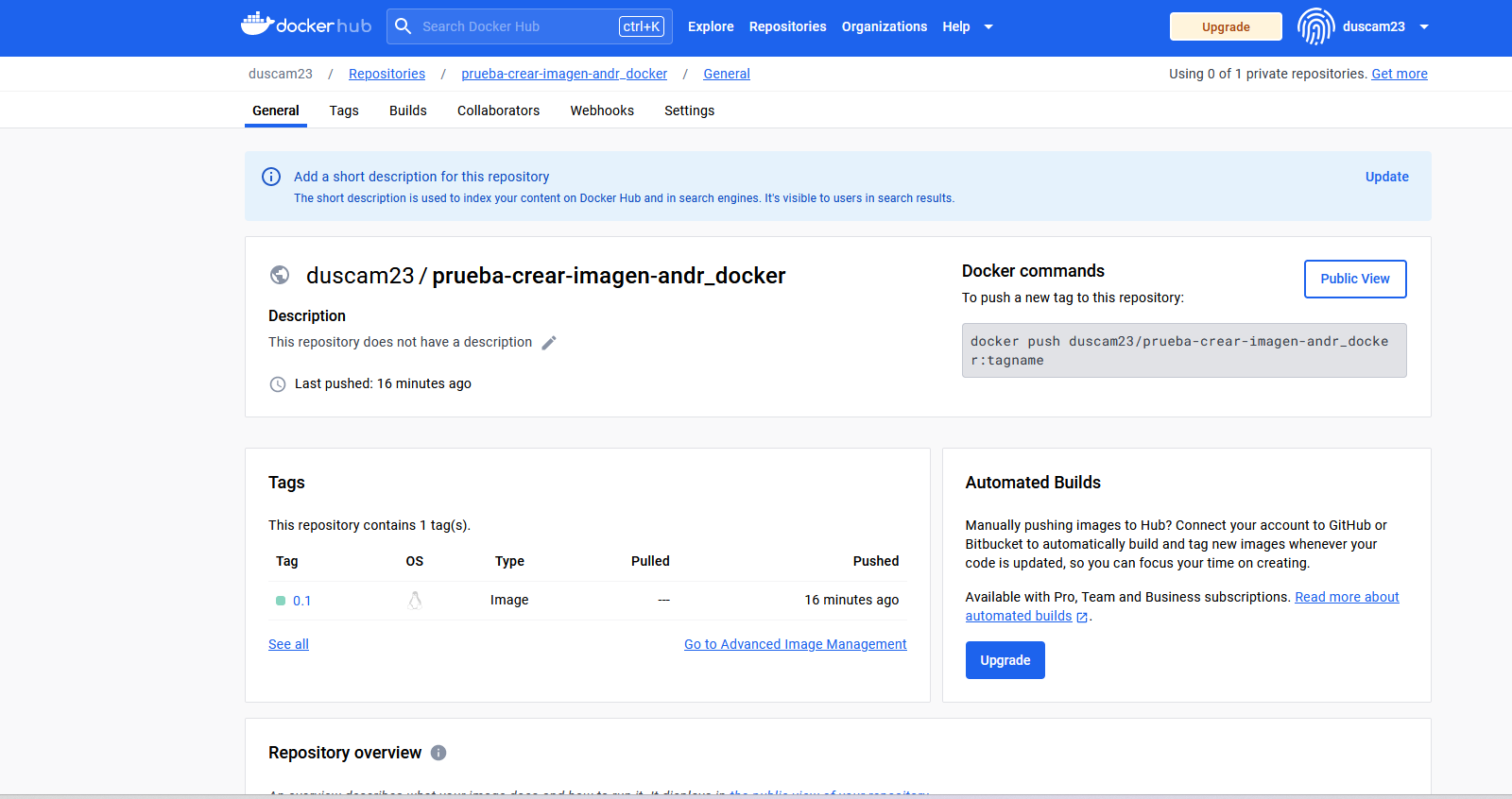
***docker push duscam23/prueba-crear-imagen-andr\_docker:0.1***



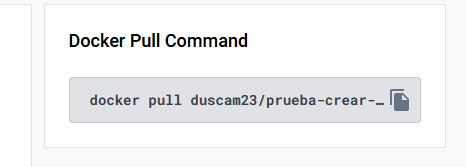
Comprobar en dockerHub si esta tu imagen.

Veo que tengo un proyecto: **prueba-crear-imagen-andr\_docker**

y sí le damos clic:



Vamos a utilizar este comando para crear contenedores y realizar operaciones de volúmenes y de mapeado:



Paso 8: Iniciar Contenedores

Cada estudiante inicia 4-5 contenedores usando su imagen:

```terminal

docker run -d -p 8081:80 nombre-de-usuario/mi\_proyecto\_docker:tag

docker run -d -p 8082:80 nombre-de-usuario/mi\_proyecto\_docker:tag

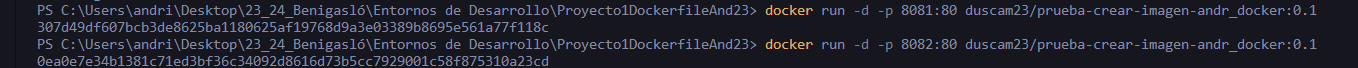
# ... continuar según sea necesario

```

***docker run -d -p 8081:80 duscam23/prueba-crear-imagen-andr\_docker:0.1***

***docker run -d -p 8082:80 duscam23/prueba-crear-imagen-andr\_docker:0.1***

***docker run -d -p 3000:80 --name prueba-1-docker duscam23/prueba-crear-imagen-andr\_docker:0.1***



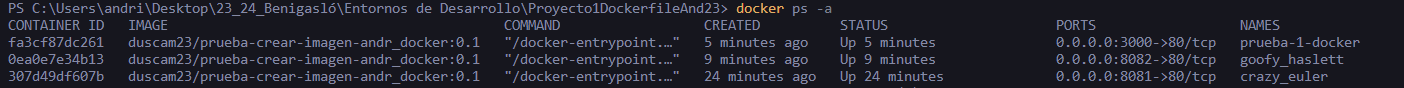


Paso 9: Verificar Contenedores en Ejecución

Cada estudiante verifica que los contenedores están en ejecución:

```terminal

docker ps -a



```

Está es la distribución es decir el mapeo de puertos para cada contenedor:

* prueba-1-docker: Mapea el puerto 3000 de tu máquina host al puerto 80 del contenedor.
* goofy\_haslett: Mapea el puerto 8082 de tu máquina host al puerto 80 del contenedor.
* crazy\_euler: Mapea el puerto 8081 de tu máquina host al puerto 80 del contenedor.

Esto significa *que, por ejemplo, si accedes a http://localhost:3000 desde tu navegador, deberías poder ver el contenido del contenedor prueba-1-docker.*

Comprobamos en nuestro pc en localhost:3000



#### Paso 10: Personalizar Contenido Visual

Cada estudiante personaliza el contenido visual de sus páginas HTML para hacer la práctica más representativa.

a) incluir texto

b) incluir imágenes

#### Paso 11: Gestionar Contenedores

- Cambiar el nombre de un contenedor:

```terminal

docker rename nombre-actual nuevo-nombre

```

- Detener un contenedor:

``` terminal

docker stop nombre-del-contenedor

```

- Eliminar un contenedor:

``` terminal

docker rm nombre-del-contenedor

```

Paso 12: Gestionar Imágenes

- Listar imágenes en la máquina:

``` terminal

docker images

```

- Eliminar una imagen:

``` terminal

docker rmi nombre-de-usuario/mi\_proyecto\_docker:tag

```

- Eliminar todas las imágenes no utilizadas:

``` terminal

docker image prune -a

```

Parte C.

.

### Práctica Guiada: Creación de un Dockerfile y Operaciones Básicas con Contenedores en Visual Studio Code

#### Paso 1: Preparación

Asegúrate de que Docker y Visual Studio Code estén instalados en las máquinas de los estudiantes.

#### Paso 2: Crear un Proyecto en Visual Studio Code

Cada estudiante crea un nuevo directorio para su proyecto y lo abre en Visual Studio Code.

```bash

mkdir mi\_proyecto\_docker

code mi\_proyecto\_docker

```

#### Paso 3: Crear el Dockerfile

En Visual Studio Code, dentro del directorio del proyecto, crea un nuevo archivo llamado `Dockerfile` y añade el siguiente contenido:

```Dockerfile

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

#### Paso 4: Construir la Imagen

Utilizando la extensión de Docker en Visual Studio Code, abre el Dockerfile, haz clic derecho y selecciona "Build Image". Proporciona un nombre y una etiqueta para la imagen.

#### Paso 5: Abrir una Terminal Integrada

Desde Visual Studio Code, abre una terminal integrada para ejecutar comandos de Docker directamente desde el entorno de desarrollo.

#### Paso 6: Operaciones Básicas con Contenedores

- \*\*Crear Contenedores con Puertos Mapeados:\*\*

- Utilizando la extensión de Docker en Visual Studio Code, inicia contenedores con puertos mapeados:

```bash

docker run -d -p 8080:80 nombre-de-usuario/mi\_proyecto\_docker:tag

docker run -d -p 3000:80 nombre-de-usuario/mi\_proyecto\_docker:tag

docker run -d -p 4000:80 -p 5000:80 nombre-de-usuario/mi\_proyecto\_docker:tag

```

- \*\*Verificar Contenedores en Ejecución:\*\*

- Utilizando la extensión de Docker en Visual Studio Code, verifica que los contenedores están en ejecución.

#### Paso 7: Intentar Borrar Imágenes o Contenedores

- Intenta borrar una imagen o contenedor que esté en ejecución. Explica el error y la necesidad de detener el contenedor antes de eliminarlo.

```bash

# Esto dará un error si intentas borrar un contenedor en ejecución

docker rm nombre-del-contenedor

# Primero detén el contenedor y luego bórralo

docker stop nombre-del-contenedor

docker rm nombre-del-contenedor

```

#### Paso 8: Listar Imágenes

- Utiliza la extensión de Docker en Visual Studio Code para listar imágenes.

```bash

docker images -q .

```

- Explica el significado de cada columna en el resultado.

–Resultados Esperados en la práctica de martes 21/11/23

* Al finalizar esta práctica, deberían haber creado un Dockerfile simple,
* construido una imagen personalizada,
* iniciado contenedores con diferentes puertos mapeados y aprendido sobre la necesidad de detener un contenedor antes de intentar borrarlo.
* También deberían haber practicado el uso del comando `docker images` y entender el significado de las columnas en la salida.