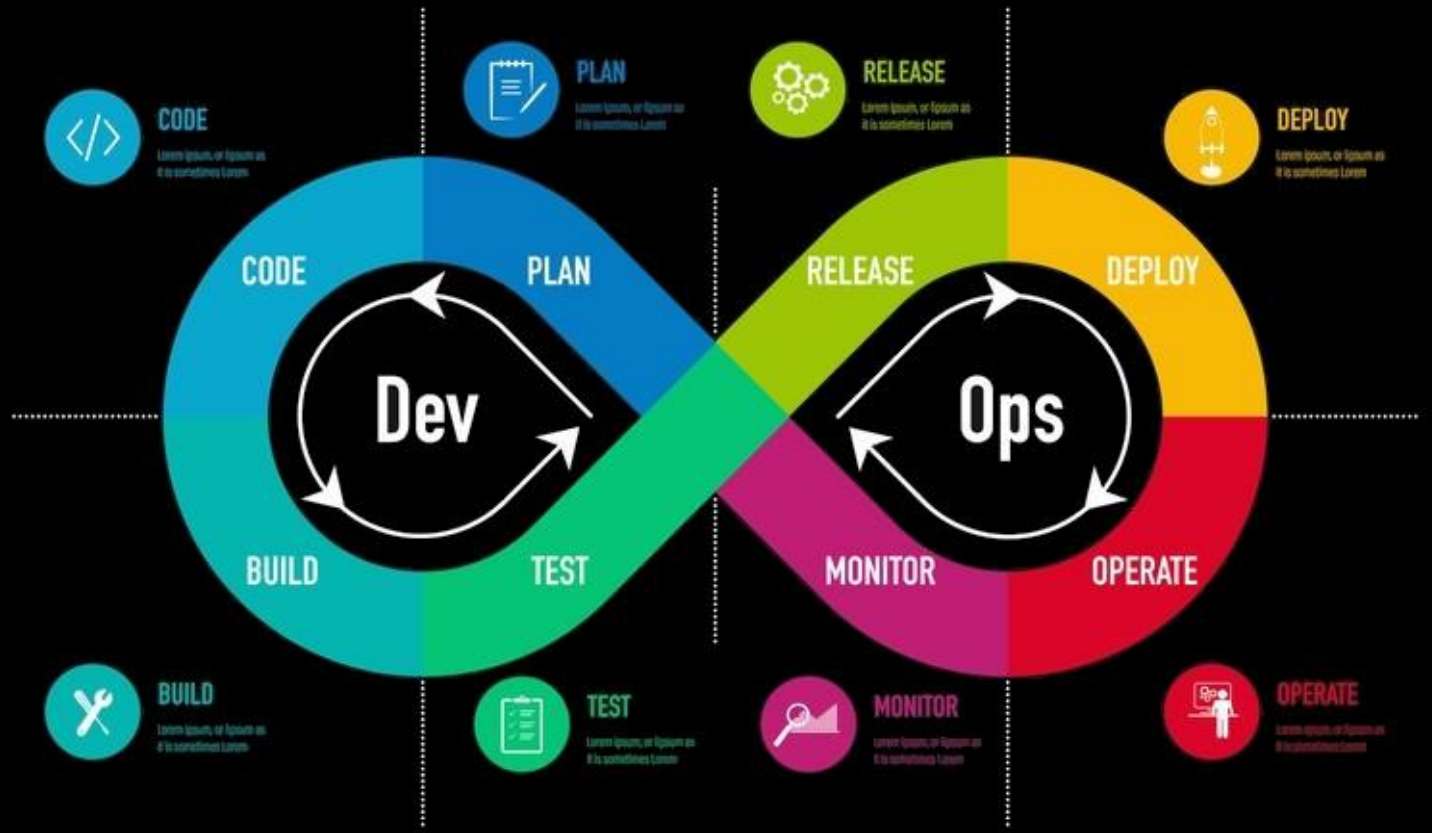
CFGS DAM

IES MONTE NARANCO



UT- 3 DevOps y Cloud Computing

Fundamentos de contenedores

Práctica 6 – Uso de del fichero Dockerfile

Archivos Dockerfile

- Un **Dockerfile** es un archivo de texto que contiene una lista de **instrucciones** que Docker sigue para construir una imagen personalizada.

- Sintáxis instrucciones Dockerfile:
  - https://docs.docker.com/reference/dockerfile/

Archivos Dockerfile - Nomenclatura

- El nombre de archivo predeterminado para un Dockerfile es **Dockerfile**, sin extensión y con la D en mayúscula.
- Algunos proyectos pueden requerir Dockerfiles con distinta nomenclatura para fines específicos.
  - Una convención común es nombrarlos con el formato **<nombre>.Dockerfile**

Archivos Dockerfile – Instrucciones más comunes

Instrucción	Descripción
FROM <image>	Define una base para nuestra imagen.
RUN <command>	Ejecuta cualquier comando en una nueva capa sobre la imagen actual y guarda el resultado. **RUN** También dispone de una interfaz de línea de comandos para ejecutar comandos.
WORKDIR <directory>	Establece el directorio de trabajo para cualquier instrucción **RUN**, **CMD**, **ENTRYPOINT**, **COPYADD** que le siga en el Dockerfile.
COPY <src> <dest>	Copia nuevos archivos o directorios desde **<src>** y los agrega al sistema de archivos del contenedor en la ruta **<dest>**.
CMD <command>	Permite definir el programa predeterminado que se ejecuta al iniciar el contenedor basado en esta imagen. Cada Dockerfile solo tiene una instancia **CMD**, y en caso de tener varias solo se utiliza la última.

Concepto de Dockerizar

- Dockerizar es el proceso de empaquetar una aplicación y todas sus dependencias (librerías, configuraciones, entorno de ejecución) dentro de un contenedor Docker.
- El objetivo es que la aplicación funcione exactamente igual en cualquier lugar: en tu computadora, en la de un colega o en un servidor en la nube, eliminando el clásico problema de "*en mi máquina sí funciona*".

Pasos para Dockerizar una aplicación

1. **Escribir un Dockerfile**: Es un archivo de texto con las instrucciones para construir la imagen (qué sistema operativo usar, qué instalar, qué archivos copiar).

2. **Construir la Imagen (Build)**: Docker lee el archivo y genera una "plantilla" estática llamada imagen.

3. **Ejecutar el Contenedor (Run)**: Se crea una instancia viva de esa imagen.

Ejemplo Dockerización

Tenemos un archivo index.html que vamos a dockerizar

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi App Dockerizada</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            background-color: #f0f2f5;
        }
        .container {
            text-align: center;
            padding: 50px;
            background: white;
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
        h1 {
            color: #007bff;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Ejemplo de dockerización</h1>
        <p>¡Este sitio está corriendo dentro de un contenedor!</p>
    </div>
</body>
</html>
```

Ejemplo Dockerización

- El archivo Dockerfile tendrá el siguiente contenido

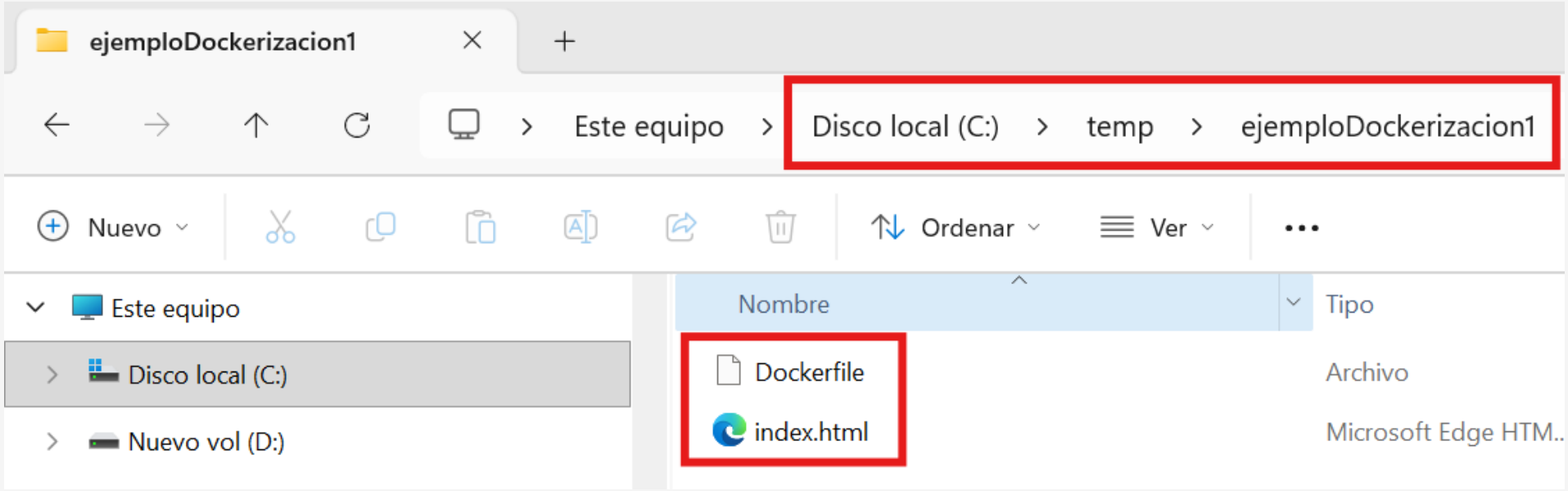
```
# 1. Elegimos la imagen base
FROM nginx:alpine

# 2. Copiamos nuestra web dentro del
contenedor
COPY . /usr/share/nginx/html

# 3. Exponemos el puerto 80
EXPOSE 80
```

Pasos para Dockerizar una aplicación

- En una carpeta de nuestro equipo por ejemplo: c:\temp\ejemploDockerizacion1
  - Creamos el archivo index.html
  - Creamos el archivo Dockerfile



Pasos para Dockerizar una aplicación

- En una ventana de msdos nos situamos en la carpeta donde hemos copiado los archivos y generamos la imagen a partir del fichero Dockerfile:
  - **docker build -t mi-web-docker .**

```
C:\temp\ejemploDockerizacion1>docker build -t mi-web-docker .
[+] Building 1.8s (8/8) FINISHED                                docker:desktop-linux
 => [internal] load build definition from Dockerfile                             0.0s
 => => transferring dockerfile: 209B                                             0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                  1.6s
 => [auth] library/nginx:pull token for registry-1.docker.io                     0.0s
 => [internal] load .dockerignore                                                0.0s
 => => transferring context: 2B                                                  0.0s
 => [internal] load build context                                                0.0s
 => => transferring context: 62B                                                 0.0s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:b0f7830b6bfaa1258f45d94c240  0.0s
 => => resolve docker.io/library/nginx:alpine@sha256:b0f7830b6bfaa1258f45d94c240  0.0s
 => CACHED [2/2] COPY . /usr/share/nginx/html                                    0.0s
 => exporting to image                                                           0.1s
 => => exporting layers                                                          0.0s
 => => exporting manifest sha256:407fd386fddf88f3b0a8b02e5c1d598a2dc9ee3caf693e2  0.0s
 => => exporting config sha256:815241a730c8ccea775a4f16f591e5210b056a7efdb3a1fc8  0.0s
 => => exporting attestation manifest sha256:8116ce14f7a93cb8a8fe313f2ad85b35f5d  0.0s
 => => exporting manifest list sha256:0dd1a07b512b17821bb4ef17eae1eec843055d8b2a  0.0s
 => => naming to docker.io/library/mi-web-docker:latest                          0.0s
 => => unpacking to docker.io/library/mi-web-docker:latest                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/hdit0a
zfd4ytuymdwuo4wg021
```

Pasos para Dockerizar una aplicación

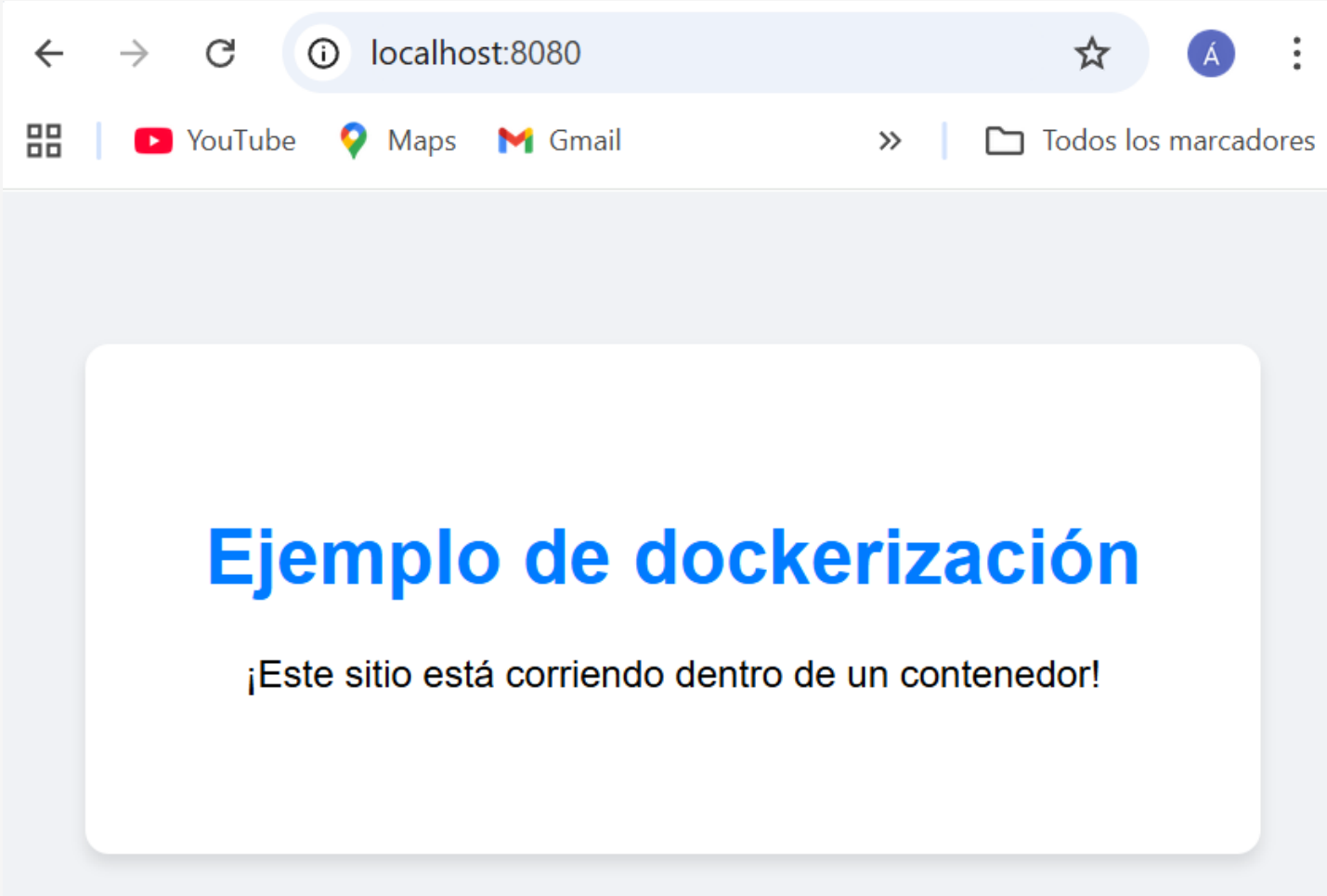
- Ejecutamos un contenedor a partir de la imagen creada:
  - **docker run -d -p 8080:80 mi-web-docker**

```
C:\temp\ejemploDockerizacion1>docker run -d -p 8080:80 mi-web-docker
112a8939c797ef9aabc408cb28155d9745e2f49dc2c2305f174ec2b24fcc20b1

C:\temp\ejemploDockerizacion1>
```

Pasos para Dockerizar una aplicación

- Desde el navegador accedemos a la página web que está en el contenedor:



Ejemplo de dockerizar programa Python

El siguiente fragmento de código muestra una pequeña aplicación "Hola Mundo" escrita en Python, utilizando el framework Flask.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Dockerizar programa Python - Dockerfile

```
# syntax=docker/dockerfile:1
FROM ubuntu:22.04

# install app dependencies
RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip install flask==3.0.*

# install app
COPY hello.py /

# final configuration
ENV FLASK_APP=hello
EXPOSE 8000
CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```

Dockerizar programa Python - Dockerfile

```
# syntax=docker/dockerfile:1
```

- Esta línea es opcional y le indica al compilador de Docker qué sintaxis usar al analizar el Dockerfile
- La directiva de análisis sintáctico debe aparecer antes de cualquier otro comentario, espacio en blanco o instrucción del Dockerfile, y debe estar en la primerl línea.

Nota: se recomienda usar **docker/dockerfile:1**, que siempre apunta a la última versión de la sintaxis de la versión 1.

Dockerizar programa Python - Dockerfile

```
FROM ubuntu:22.04
```

- La instrucción FROM establece la imagen base en la versión 22.04 de Ubuntu.
- Todas las instrucciones siguientes se ejecutan en esta imagen base: un entorno Ubuntu.
- El nombre de la imagen ubuntu:22.04 sigue la notación nombre:etiqueta que es el estándar para nombrar imágenes Docker.

Configuración del entorno

```
# install app dependencies
RUN apt-get update && apt-get install -y python3 python3-pip
```

- La instrucción RUN ejecuta un shell en Ubuntu que actualiza el índice de paquetes APT e instala las herramientas de Python en el contenedor.

Comentarios

```
# install app dependencies
```

- La línea # install app dependencies es un comentario.
- Los comentarios en Dockerfiles comienzan con el símbolo #

Nota: los comentarios se indican con el mismo símbolo que la directiva de sintaxis en la primera línea del archivo. Este símbolo solo se interpreta como una directiva si el patrón coincide con una directiva y aparece al principio del Dockerfile. De lo contrario, se trata como un comentario.

Instalación de dependencias

```
RUN pip install flask==3.0.*
```

- La segunda instrucción RUN instala la dependencia flask requerida por la aplicación Python.
- Un requisito previo para esta instrucción es que pip esté instalado en el contenedor. El primer comando RUN se encargó de instalarlo, lo que garantiza que podamos usar el comando para instalar el framework web Flask.

Copiado de archivos

```
COPY hello.py /
```

- Se utiliza la instrucción **COPY** para copiar el archivo hello.py desde el build context local al directorio raíz de nuestra imagen.
- Un **build context** es el conjunto de archivos a los que se puede acceder en las instrucciones de **Dockerfile**, como COPY y ADD.
- El **build context** se especifica cuando se ejecuta el **Dockerfile** y lo más habitual es utilizar la carpeta en la cual se encuentra el fichero **Dockerfile** indicándolo con el directorio **.**

Configuración de variables de entorno

```
ENV FLASK_APP=hello
```

- Si nuestra aplicación utiliza variables de entorno, podemos configurarlas utilizando la instrucción **ENV**.
- Esto define una variable de entorno de Linux que necesitaremos más adelante.
- **Flask**, el framework utilizado en este ejemplo, usa esta variable para iniciar la aplicación.
- Sin ella, **Flask** no sabría dónde encontrar nuestra aplicación para poder ejecutarla.

Configuración de variables de entorno

```
CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```

- La instrucción CMD establece el comando que se ejecuta cuando se inicia un basado en esta imagen.
- En este caso inicia el servidor de desarrollo de Flask, que escucha en todas las direcciones del puerto 8000.

Configuración de variables de entorno

```
CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```

- Los elementos entre los corchetes deben ir entre comillas dobles separados por comas.
- El primer elemento (en este caso “flask”) Docker lo toma como el comando a ejecutar y el resto los considera como los parámetros para el comando.

Generación de la imagen a partir del Dockerfile

```
docker build -t test:latest .
```

- La opción **-t test:latest** especifica el nombre y la etiqueta de la imagen.
- El punto (.) al final del comando establece el contexto de compilación en el directorio actual. Esto significa que la compilación espera encontrar el Dockerfile y el archivo hello.py en el directorio donde se ejecuta el comando. Si estos archivos no existen, la compilación fallará.

Generación de la imagen a partir del Dockerfile

```
docker run -p 127.0.0.1:8000:8000 test:latest
```