

MD002 - Infraestructuras de computación

Proyecto 2 – Contenedores (Docker)



KENNEDY, Leonard

Diciembre 2023

Índice

1. Introducción	2
2. Objetivos	2
3. Marco Teórico.....	2
3.1. Containerisation	2
3.2. Docker.....	4
4. ETL	6
4.1. Extraction	6
4.2. Transformation.....	7
4.3. Load	8
5. Docker.....	9
5.1. Instalación	9
5.2. Dockerfile	11
5.3. Docker Image.....	12
5.4. Docker Container	14
6. Conclusiones	16
7. REFERENCIAS.....	17

1. Introducción

En el contexto de la era digital en la que nos encontramos, tecnologías como la “Containerisation o containerización” han permitido que la información sea compartida bajo estándares que simplifican la forma en que se genera valor y conocimiento. Docker, con su enfoque basado en contenedores, cambia la manera en que se conceptualizan y se ejecutan las aplicaciones.

Este informe explora con un ejemplo práctico, la funcionalidad y los beneficios de usar Docker como herramienta de Containerisation. Con lenguaje Python se creó una ETL usando datos de una API que contiene datos históricos de diferentes tipos automóviles. Posteriormente se creó un Dockerfile que fue utilizado para la creación de la imagen. Con esta imagen se creó el contenedor que contiene los recursos necesarios para correr el programa de Python.

2. Objetivos

- Definir el concepto de Containerisation y la plataforma Docker
- Crear una ETL que obtenga los datos de una API, utilizando lenguaje Python.
- Utilizar Docker para generar el contenedor que contenga los recursos necesarios para desplegar el código Python y ejecutar la ETL creada.

3. Marco Teórico

3.1. Containerisation

La tecnología de contenedores, a diferencia de la virtualización con hipervisor, permite desplegar contenedores en el entorno del usuario, encima del kernel del sistema operativo. Por esta razón, también es conocida como virtualización a nivel de sistema operativo.

En lugar de iniciar un sistema operativo completo en el host, los contenedores comparten el kernel con el sistema operativo, lo que disminuye los costos operativos y proporciona aislamiento entre las aplicaciones. Estas características de los contenedores hacen posible enviar un contenedor

pequeño que actúa como un sistema operativo completo, encapsulando solo aquellos archivos necesarios para ejecutar las aplicaciones deseadas.

La virtualización por otro lado despliega una o varias máquinas virtuales usando el hardware disponible, utilizando una capa de intermediación.

La siguiente tabla comparativa permite observar las diferencias entre el concepto de contenedores y máquinas virtuales.

Característica	Máquinas Virtuales	Contenedores
Aislamiento del entorno de trabajo	Proporcionan un aislamiento completo. Se ejecutan un sistema operativo completo independiente del anfitrión	Menor aislamiento pues los contenedores comparten el mismo kernel del sistema operativo del anfitrión,
Uso de Recursos	Pueden consumir más recursos, porque cada máquina virtual utiliza su propio sistema operativo y una cantidad significativa de recursos dedicados.	Eficiencia en el uso de recursos pues utilizan el mismo sistema operativo y si son diseñados de manera correcta, incluyen las bibliotecas y dependencias necesarias para desplegar la aplicación o código
Velocidad de arranque	Arranque lento pues debe inicializar por completo el sistema operativo	Tienen un arranque mucho más rápido, ya que no requieren iniciar un sistema operativo completo.

Tabla 1 Comparación de características de máquinas virtuales y contenedores

Según (Turnbull, 2019) Los contenedores suelen considerarse una tecnología eficiente porque requieren menos recursos. A diferencia de la virtualización tradicional, no necesitan una capa de emulación o una capa de hipervisor para ejecutarse, y en cambio utilizan la interfaz normal de llamadas al sistema del sistema operativo. Esto reduce la cantidad de recursos necesarios para ejecutar contenedores y permite tener una mayor densidad de contenedores en un host.

3.2. Docker

Docker es una plataforma de código abierto que permite crear, gestionar y publicar contenedores. La gran ventaja de Docker es que, si hay un proyecto de software dentro de un contenedor, y este se comparte, funcionará exactamente igual sin importar el sistema operativo que se tenga o la versión del lenguaje de programación con la que fue hecho. Es un lugar en donde se empaqueta toda la solución, guardándola y dejándola lista para que cualquier persona la pueda desplegar sin problemas. Como explica (Meadusani, 2018), Docker fue creado para proporcionar un entorno liviano y rápido en el cual ejecutar el código de los usuarios con un flujo de trabajo eficiente, permitiendo llevar ese código desde la computadora del usuario al entorno de prueba y luego al entorno de producción.

La arquitectura de Docker tiene elementos como Docker Daemon, que puede ser considerado como el núcleo de Docker, ya que interactúa con el sistema operativo para posibilitar el uso, escalado, creación, configuración y eliminación de contenedores. El Docker Daemon interactúa con el sistema operativo para lograr esas funciones.

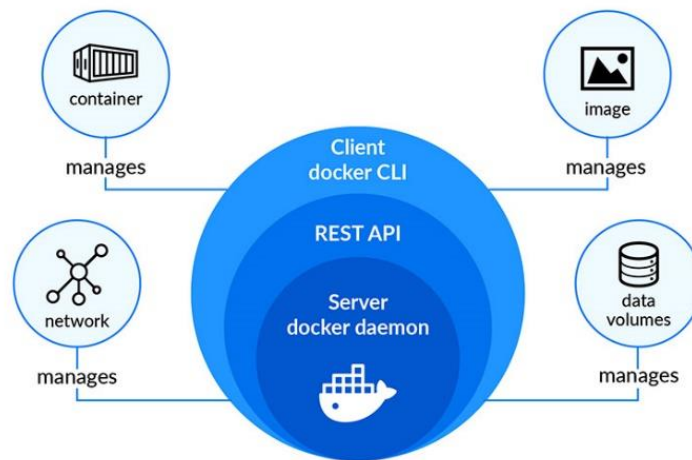


Ilustración 1Arquitectura de Docker (Tomado de <https://thinkpalm.com/blogs/what-is-docker-how-does-it-simplify-testing/>)

La forma en que Docker Daemon expone una interfaz para comunicarse con él es mediante el protocolo HTTP de la REST API. Es posible comunicarse con Docker Daemon desde nuestra propia máquina a través de HTTP, así como desde una máquina remota. También es posible exponer nuestro Docker Daemon, permitiendo que una máquina ubicada en otra red se comunique con la nuestra y le dé instrucciones al Docker

Por encima de la REST API se tiene el Client Docker CLI que es la línea de comandos, la interfaz que se utiliza para interactuar con el sistema Docker. Este cliente se comunica a través del REST API para ejecutar comandos, lo que permite gestionar el ciclo de vida de nuestras imágenes y contenedores. Se utilizan instrucciones específicas en la línea de comandos para realizar acciones con los contenedores (como las descritas anteriormente), así como la construcción y administración de imágenes.

Las imágenes (al igual que los contenedores, volúmenes de datos, y redes), hacen parte de los componentes de Docker. El concepto de imagen es explicado por (Docs, s.f.), como una plantilla de formato de solo lectura con instrucciones para crear un contenedor de Docker. Por lo general, una imagen se basa en otra imagen, con algunos cambios adicionales.

La diferencia entre Docker y un archivo .Zip o una máquina virtual está en la capacidad del Docker Daemon para contener proyectos de manera eficiente. Con Docker, es posible containerizar los proyectos y luego ajustar aspectos como redes, volúmenes de datos o imágenes, ya sea antes o después de la creación del contenedor. Esto da mayor flexibilidad en el proceso de desarrollo y despliegue de la aplicación, código o solución.

4. ETL

La información que es usada para realizar esta ETL se tomó de la website <https://api-ninjas.com/>. Tienen una gran variedad de api que pueden ser utilizadas. En este caso se utilizó una api de autos que arroja bastante información acerca de el tipo de transmisión, configuración del motor y consumo del automóvil entre otras. El objetivo de esta ETL es determinar mediante un KPI dado, cual es el vehículo que tiene mayor ratio Cilindros/Consumo, para determinar así cuales son los autos con mayor potencia del motor y menos consumo. La ETL arroja un documento formato .csv que contiene organizados de mayor a menor los automóviles según su KPI.

Cabe destacar que, observando los datos obtenidos de la API, pareciera que la información del consumo dada en millas por galón no está correcta. Comprobar la veracidad de esos datos se encuentra fuera del alcance de este reporte. Se da por sentado que esos datos obtenidos están correctos.

4.1. Extraction

Los datos son extraídos de la website utilizando la librería request

```
url = f'https://api.api-ninjas.com/v1/cars?limit=50&year={year}'
url_response = requests.get(url, headers={'X-API-Key': 'J8q/mbRgr6+d0Zj5S0waeg==AyIvEPXnblRz5u8z'})
```

Ilustración 2 Uso librería request

Es posible indicar el año del que se quiere sacar la información. Desde 1985 a 2023

```
import requests
import pandas as pd

## Extract (API data import)
# Request the user to import a valid year (Data from 1984 to 2023)
while True:
    try:
        year = int(input("Ingrese el año (entre 1984 y 2023): "))
        if 1984 <= year <= 2023:
            break
        else:
            print("Error: Por favor, ingrese un año válido dentro del rango dado.")
    except ValueError:
        print("Error: Por favor, ingrese un año válido (número entero).")

# Do the API request with the obtained year from the user
url = f'https://api.api-ninjas.com/v1/cars?limit=50&year={year}'
url_response = requests.get(url, headers={'X-API-Key': 'J8q/mbRgr6+d0Zj5S0waeg==AyIvEPXnblRz5u8z'})
#API key given from the api-ninjas.com website after you register
```

Ilustración 3 Datos de automóviles desde 1985 a 2023

4.2. Transformation

La transformación de los datos incluye la creación de una nueva columna que es el promedio del consumo de gasolina en ciudad y carretera. Y otra columna que utiliza este promedio para hacer un KPI de consumo que es una ratio entre cilindrada del auto y el consumo promedio. Se eliminan columnas que no son relevantes en el estudio y se organiza el dataset de mayor a menor según el KPI de consumo. Arriba de la tabla estarán los autos de una mayor ratio de potencia y consumo.

```
# Transform (Modify the data set, delete unnecessary columns and add KPI)
# Use pandas to make information a DataSet
df = pd.DataFrame(information)

# Delete columns that won't be used because the year is given and we are calculating the average fuel
consumption. Drop deletes the specified columns
columns_to_delete = ['combination_mpg', 'year']
df_without_columns = df.drop(columns_to_delete, axis=1)

# Create a new column for the avg fuel consumption called avg_mpg
df_without_columns['avg_mpg'] = (df_without_columns['city_mpg'] + df_without_columns['highway_mpg']) /
2

# Create a new column for our base KPI of # cylinders / consumo promedio, the higher the value the
better. Rate between power and fuel consumption.
df_without_columns['kpi_consumption'] = (df_without_columns['cylinders'] /
df_without_columns['avg_mpg'])

# Reorganise column to give you the most relevant information first
column_order = ['make', 'model', 'class', 'kpi_consumption', 'transmission', 'fuel_type', 'drive',
'cylinders', 'displacement', 'highway_mpg', 'avg_mpg', 'city_mpg']
df_ordered = df_without_columns[column_order]

# Organise the table from max to min to identify the best vehicles based on the defined KPI
df_ordered = df_ordered.sort_values(by='kpi_consumption', ascending=False)

# Create a new index column
df_ordered = df_ordered.reset_index(drop=True)
```

Ilustración 4 Código de la transformación de los datos

4.3. Load

Por último, la carga genera un archivo.csv con los datos transformados.

```
#Load (Export the transformed data)
#Export the data in a .csv file type
path_csv = 'datos_automoviles.csv'
df_ordered.to_csv(path_csv, index=False)
print(f'DataFrame exportado a {path_csv}')
```

Ilustración 5 Creación archivo .csv versión para python local

Versión para Docker

```
#Ask the user if they want to download the DataFrame
download_choice = input("¿Desea descargar el DataFrame? (Yes or No): ").lower()

# Load - Docker .csv file creation
output_dir = '/output_dir/'
output = output_dir + 'cars_db.csv'

# Check user's choice and proceed accordingly
if download_choice == 'yes':
    df_ordered.to_csv(output, index=False)
    print('El DataFrame ha sido exportado a: ' + output)
else:
    print('No se descargará el DataFrame.')
```

Ilustración 6 Exportar .csv para Docker

5. Docker

Una vez creada la ETL, procedemos a utilizar la plataforma de Docker para crear un container y poder compartirla. Para ello, se instala Docker en la versión de Ubuntu que se encuentra en la máquina virtual de “Virtual Box” creada sobre Windows en el ejercicio anterior. Visual Studio Code permite crear el Dockerfile que es usado para crear la imagen usando la terminal de comandos. Luego esta imagen permite hacer la creación del contenedor que despliega el código de la ETL hecho en Python, con los recursos y librerías necesarias para su ejecución.

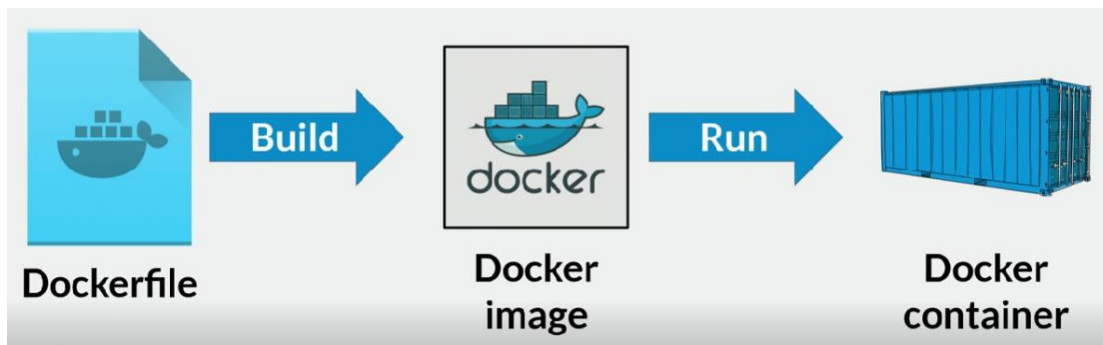


Ilustración 7 Secuencia de creación de Contenedor desde el Dockerfile. Tomado de (Platzi.com/Docker)

5.1. Instalación

Para la instalación de Docker en el entorno Ubuntu, inicialmente se verificaron variables de entorno relacionadas con el entorno de escritorio actual para que sean gnome con el comando

```
echo $XDG_CURRENT_DESKTOP
```

Luego se instaló curl para descargar el script de instalación de Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
l30k3nn@UbuntuLK: ~  
l30k3nn@UbuntuLK:~$ curl -fsSL https://get.docker.com -o get-docker.sh  
Command 'curl' not found, but can be installed with:  
sudo snap install curl # version 8.1.2, or  
sudo apt install curl # version 7.81.0-1ubuntu1.14  
See 'snap info curl' for additional versions.  
l30k3nn@UbuntuLK:~$ sudo snap install curl # version 8.1.2  
curl 8.1.2 from Wouter van Bommel (woutervb) installed
```

Ilustración 8 Instalación de curl

Luego se ejecuta el script de instalación:

```
sudo sh get-docker.sh
```

Para confirmar la correcta instalación se puede usar:

```
sudo service docker status
```

Donde se ve un mensaje indicando que el servicio está activo (running) y en ejecución.

```
~  
...skipping...  
● docker.service - Docker Application Container Engine  
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)  
   Active: active (running) since Thu 2023-11-30 14:00:47 CET; 9min ago  
 TriggeredBy: ● docker.socket  
     Docs: https://docs.docker.com  
    Main PID: 903 (dockerd)  
       Tasks: 10  
      Memory: 105.3M  
         CPU: 5.725s  
    CGroup: /system.slice/docker.service  
            └─903 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Ilustración 9 Comprobación del estado de servicio de Docker

En este caso, la versión de Docker Instalada fue 24.0.7

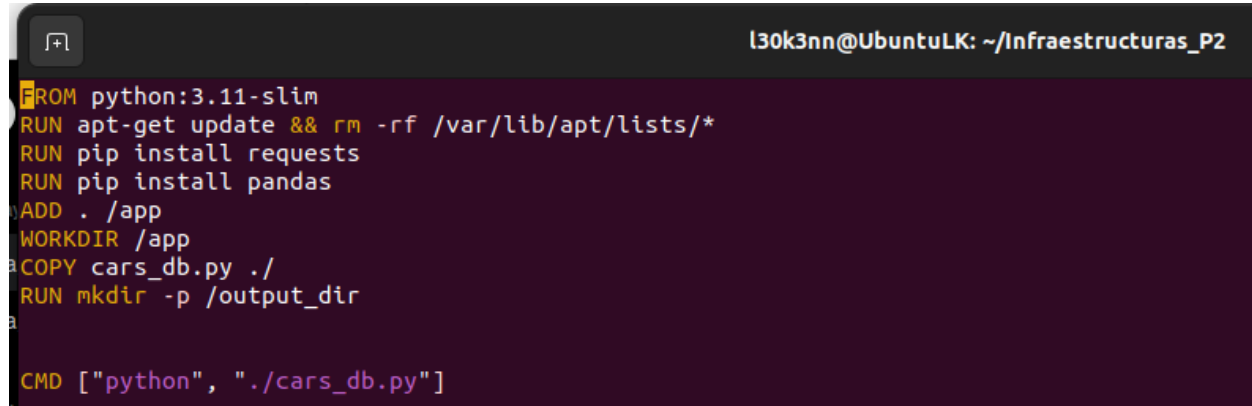
```
l30k3nn@UbuntuUK:~$ docker --version  
Docker version 24.0.7, build afdd53b  
l30k3nn@UbuntuUK:~$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
719385e32844: Pull complete  
Digest: sha256:c79d06dffd3d3eb04cafd0dc2bacab0992ebc243e083cabe208bac4dd7759e0  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (and64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
  
l30k3nn@UbuntuUK:~$ docker info  
Client: Docker Engine - Community  
Version: 24.0.7  
Context: default  
Debug Mode: false  
Plugins:  
 buildx: Docker Buildx (Docker Inc.)  
   Version: v0.11.2  
   Path: /usr/libexec/docker/cli-plugins/docker-buildx  
 compose: Docker Compose (Docker Inc.)  
   Version: v2.21.0  
   Path: /usr/libexec/docker/cli-plugins/docker-compose  
  
Server:  
Containers: 1  
 Running: 0  
  Paused: 0  
 Stopped: 1  
Images: 1  
Server Version: 24.0.7  
Storage Driver: overlay2  
 Backing Filesystem: extfs  
 Supports d_type: true  
 Using metaCopy: false  
 Native Overlay Diff: true  
userxattr: false
```

Ilustración 10 Verificación de la versión de Docker instalada

5.2. Dockerfile

El Dockerfile es la receta que se debe seguir para cocinar la imagen que incluye las dependencias necesarias para ejecutar el código Python y establece la configuración necesaria dentro de la imagen para la ejecución del código.

La creación del Dockerfile se hizo en Visual Studio Code. El comando Vim Dockerfile nos muestra lo siguiente:



```
FROM python:3.11-slim
RUN apt-get update && rm -rf /var/lib/apt/lists/*
RUN pip install requests
RUN pip install pandas
ADD . /app
WORKDIR /app
COPY cars_db.py ./
RUN mkdir -p /output_dir

CMD ["python", "./cars_db.py"]
```

Ilustración 11 Dockerfile

Mirando línea a línea el Dockerfile tenemos:

FROM python:3.11-slim: Esta línea especifica la imagen base que se utilizará para construir esta imagen. En este caso, se está utilizando una imagen ligera de Python 3.11. El parámetro *slim* permite mantener el tamaño de la imagen lo más pequeño posible.

El primer **RUN apt-get update && rm -rf /var/lib/apt/lists/*** actualiza los paquetes en la imagen y limpia el cache de paquetes. Este comando se utiliza para asegurarse de que el sistema de paquetes de la imagen esté actualizado y, al mismo tiempo, reduce el tamaño de la imagen al eliminar los archivos de lista de paquetes descargados.

El siguiente **RUN pip install requests:** Instala el paquete de Python **requests** utilizando pip. para realizar solicitudes HTTP necesaria para obtener los datos de la API

El siguiente run se utiliza para instalar pandas para la manipulación del dataframe recibido en formato Json

ADD . /app: Copia el contenido del directorio actual, dado por el “.”, donde se tiene guardado el código al directorio **/app** dentro de la imagen.

WORKDIR /app: Establece el directorio de trabajo dentro de la imagen en **/app**. Los comandos que sigan se ejecutarán en ese directorio.

COPY cars_db.py ./: Copia el archivo python **cars_db.py** desde el directorio actual donde esta guardado el código, al directorio de trabajo (**/app**) dentro de la imagen.

RUN mkdir -p /output_dir: Crea un directorio llamado **output_dir** dentro de la imagen. La opción **-p** permite crear directorios padres si no existen, para poder guardar el archivo .csv que contiene el reporte final de los vehículos del año seleccionado.

CMD ["python", "./cars_db.py"]: Define el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en esta imagen. En este caso, ejecutará el código **cars_db.py** utilizando Python.

5.3. Docker Image

Una vez creado el Dockerfile, se procede a construir la imagen con el comando

docker build -t carsdb .

Donde docker build es el comando que se utiliza para crear la imagen y -t es la etiqueta o “tag” que nombra la imagen.

Como se puede observar en la zona derecha de la imagen a continuación, ha tardado bastante tiempo en crear la imagen pues es la primera vez que se creaba y no había caché.

```
● l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker build -t carsdb .
[+] Building 106.2s (13/13) FINISHED                                docker:default
=> [internal] load .dockerignore                                     0.1s
=> transferring context: 2B                                          0.0s
=> [internal] load build definition from Dockerfile                 0.1s
=> transferring dockerfile: 261B                                     0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim  4.6s
[1/8] FROM docker.io/library/python:3.11-slim@sha256:23f52205321f806c2cc742cefbf837e0d25101388c043e860c7817985230565c 30.3s
=> resolve docker.io/library/python:3.11-slim@sha256:23f52205321f806c2cc742cefbf837e0d25101388c043e860c7817985230565c 0.1s
=> sha256:eb7a9f47cd21f68c04356a841389326a9cefcdd2dbf53ed788b13641af5aa4 1.25kB / 1.25kB 0.0s
=> sha256:7ba9869f7b1ec033cbl8d3eed51ebd34286f6dc7e551baeeab3b603cab18 6.91kB / 6.91kB 0.0s
=> sha256:23f52205321f806c2cc742cefbf837e0d25101388c043e860c7817985230565c 4.20kB / 4.20kB 0.0s
=> sha256:1f7ce2fa46ab3942feabee654933948821383a5a821789dddb2d8c3df59e227 29.15MB / 29.15MB 9.6s
=> sha256:442c5d63eafd969b755d7c8e0914319677ca8d5112a84e851b16677a4fcd1d89 3.32MB / 3.32MB 2.2s
=> sha256:c3aa3af0d5819a2242bfec83abe997d35d5d295c8f3ef3fc40d2celac796b1bd 12.83MB / 12.83MB 6.1s
=> sha256:4bfbe15b3e0135fc110b1a7b2dcecc2d9564b611698e7a9c36cb28774d0b1581 233B / 233B 2.8s
=> sha256:848d19a36773ad77a8144dab4d0f16d8c7a7794c58c8f91d1b7c06bd662c2673 3.19MB / 3.19MB 5.2s
=> extracting sha256:1f7ce2fa46ab3942feabee654933948821383a5a821789dddb2d8c3df59e227 11.7s
=> extracting sha256:442c5d63eafd969b755d7c8e0914319677ca8d5112a84e851b16677a4fcd1d89 0.7s
=> extracting sha256:c3aa3af0d5819a2242bfec83abe997d35d5d295c8f3ef3fc40d2celac796b1bd 3.5s
=> extracting sha256:4bfbe15b3e0135fc110b1a7b2dcecc2d9564b611698e7a9c36cb28774d0b1581 0.0s
=> extracting sha256:848d19a36773ad77a8144dab4d0f16d8c7a7794c58c8f91d1b7c06bd662c2673 2.6s
[internal] load build context                                       0.3s
=> transferring context: 3.40kB                                     0.1s
[2/8] RUN apt-get update && rm -rf /var/lib/apt/lists/*           12.7s
[3/8] RUN pip install requests                                    11.8s
[4/8] RUN pip install pandas                                     37.1s
[5/8] ADD . /app                                                    0.6s
[6/8] WORKDIR /app                                                  0.2s
[7/8] COPY cars_db.py ./                                           0.3s
[8/8] RUN mkdir -p /output_dir                                     1.2s
=> exporting to image                                              6.5s
=> exporting layers                                                6.4s
=> writing image sha256:d6bac3b57cce57d3ada37ee750ada975653dfe78bc5382fec4ad8fb58c50e4cf 0.0s
=> naming to docker.io/library/carsdb                               0.0s
```

Ilustración 12 Creación de la imagen

El comando ***docker image ls*** se utiliza para listar las imágenes de Docker que están actualmente almacenadas localmente en el sistema e incluye el tag, un identificador único de la imagen, cuándo se creó, y el tamaño de la imagen.

```
See 'docker run --help'.
● l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker image ls
REPOSITORY      TAG         IMAGE ID      CREATED        SIZE
carsdb           latest      d6bac3b57cce  11 minutes ago 327MB
hello-world      latest      9c7a54a9a43c  6 months ago  13.3kB
```

Ilustración 13 Lista de imágenes almacenadas localmente

5.4. Docker Container

El comando ***docker container run --name containercars -it carsdb*** se utiliza para crear y ejecutar el contenedor a partir de la imagen dada. Se da el nombre que se le quiere poner al contenedor y la referencia de la imagen con la que se quiere interactuar.

Como se puede ver a continuación, el contenedor también da paso a la ejecución del código de la ETL

```
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker container run --name containercars -it carsdb
Ingrese el año (entre 1984 y 2023): 1999
Datos obtenidos exitosamente.
¿Desea descargar el DataFrame? (Yes or No): No
No se descargará el DataFrame.
```

Ilustración 14 Creación y ejecución del contenedor

La ETL da como resultado un archivo tipo .csv. Para poder ver si el archivo fue creado y su ubicación primero reiniciamos el contenedor con el comando ***docker start***, y se ejecuta el contenedor con el comando ***docker exec -it contenedorcar /bin/bash***.

Luego se navega a la carpeta con el comando ***cd /output_dir/***, y con ***ls*** se puede ver el contenido de la carpeta. La siguiente imagen muestra los comandos utilizados para encontrar el archivo .csv

```
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker container run --name containercar -it cardb
docker: invalid reference format.
See 'docker run --help'.
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker container run -name containercar -it cardb
unknown shorthand flag: 'n' in -name
See 'docker container run --help'.
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker container run --name containercar -it cardb
Ingrese el año (entre 1984 y 2023): 1986
Datos obtenidos exitosamente.
¿Desea descargar el DataFrame? (Yes or No): Yes
El DataFrame ha sido exportado a: /output_dir/cars_db.csv
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS      PORTS      NAMES
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker exec -it containercar /bin/bash
Error response from daemon: Container eeda838068496819fbf86e980abc0798061009b6871380c21aeafe0aa33a07c2 is not running
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker start containercar
containercar
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS      PORTS      NAMES
eeda83806849   cardb     "python ./cars_db.py"    10 minutes ago    Up 11 seconds      containercar
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ cd /output_dir/
bash: cd: /output_dir/: No such file or directory
l30k3nn@UbuntuLK:~/Infraestructuras_P2$ docker exec -it containercar /bin/bash
root@eeda83806849:/app# cd /output_dir/
root@eeda83806849:/output_dir# pwd
/output_dir
root@eeda83806849:/output_dir# ls
cars_db.csv
root@eeda83806849:/output_dir#
```

Ilustración 15 Comandos para encontrar Docker Container

Finalmente, el comando *cat cars_db.csv* permite visualizar el archivo generado con la información relevante de los vehículos

```
cars_db.csv
root@eeda83806849:/output_dir# cat cars_db.csv
make,model,class,kpi_consumption,transmission,fuel_type,drive,cylinders,displacement,highway_mpg,avg_mpg,city_mpg
lamborghini,countach,two seater,1.5,m,gas,rwd,12,5.2,10,8.0,6
ferrari,testarossa,two seater,1.0434782608695652,m,gas,rwd,12,4.9,14,11.5,9
texas coach company,500sl,two seater,0.64,a,gas,rwd,8,5.0,14,12.5,11
mercedes-benz,560sl,two seater,0.5714285714285714,a,gas,rwd,8,5.6,16,14.0,12
ferrari,mondial/cabriolet,minicompact car,0.5714285714285714,m,gas,rwd,8,3.2,17,14.0,11
ferrari,328 gts/gtb,two seater,0.5714285714285714,m,gas,rwd,8,3.2,17,14.0,11
maserati,biturbo spyder,two seater,0.46153846153846156,a,gas,rwd,6,2.5,14,13.0,12
maserati,biturbo,minicompact car,0.46153846153846156,a,gas,rwd,6,2.5,14,13.0,12
porsche,928 s,minicompact car,0.4444444444444444,a,gas,rwd,8,5.0,21,18.0,15
autokraft limited,a.c.mkiv,two seater,0.4444444444444444,m,gas,rwd,8,4.9,21,18.0,15
chevrolet,corvette convertible,two seater,0.43243243243243246,m,gas,rwd,8,5.7,22,18.5,15
chevrolet,corvette convertible,two seater,0.43243243243243246,a,gas,rwd,8,5.7,22,18.5,15
chevrolet,corvette,two seater,0.43243243243243246,m,gas,rwd,8,5.7,22,18.5,15
chevrolet,corvette,two seater,0.43243243243243246,a,gas,rwd,8,5.7,22,18.5,15
porsche,911,minicompact car,0.34285714285714286,m,gas,rwd,6,3.3,20,17.5,15
tvr engineering ltd,tvr 280i/350i coupe,two seater,0.32432432432432434,a,gas,rwd,6,2.8,22,18.5,15
tvr engineering ltd,tvr 280i/350i convertible,two seater,0.32432432432432434,a,gas,rwd,6,2.8,22,18.5,15
nissan,300zx,two seater,0.3157894736842105,a,gas,rwd,6,3.0,23,19.0,15
porsche,911,minicompact car,0.3157894736842105,m,gas,rwd,6,3.2,23,19.0,15
nissan,300zx,two seater,0.3076923076923077,m,gas,rwd,6,3.0,23,19.5,16
pontiac,fiero,two seater,0.3076923076923077,a,gas,rwd,6,2.8,23,19.5,16
nissan,300zx,two seater,0.3076923076923077,a,gas,rwd,6,3.0,23,19.5,16
nissan,300zx,two seater,0.3,m,gas,rwd,6,3.0,24,20.0,16
alfa romeo,gtv,minicompact car,0.2926829268292683,m,gas,rwd,6,2.5,24,20.5,17
pontiac,fiero,two seater,0.2926829268292683,m,gas,rwd,6,2.8,25,20.5,16
pontiac,fiero,two seater,0.2926829268292683,m,gas,rwd,6,2.8,24,20.5,17
```

Ilustración 16 Visualización del archivo .csv creado

6. Conclusiones

Docker, con su enfoque basado en contenedores, es una herramienta muy innovadora al cambiar la conceptualización y ejecución de aplicaciones. Facilita el desarrollo, prueba y despliegue de aplicaciones al ofrecer un entorno autosuficiente para compartir y ejecutar soluciones.

Los contenedores pueden ser vistos como Jim Carrey en la película “The Truman Show”, donde viven en su mundo y tienen todo lo que necesitan, pero que a su vez están dentro de un sistema operativo que los contiene.

7. REFERENCIAS

Docs, D., s.f. *docs.docker.com.* [En línea]
Available at: <https://docs.docker.com/get-started/overview/#docker-architecture>
[Último acceso: 28 11 2023].

Meadusani, S. R., 2018. Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications. *CULMINATING PROJECTS IN INFORMATION ASSURANCE*.

Turnbull, J., 2019. *The Docker Book: Containerization Is the New Virtualization*. V18.09.2 ed. s.l.:s.n.

APPENDIX A

```
import requests
import pandas as pd

## Extract (API data import)
# Request the user to import a valid year (Data from 1984 to 2023)
while True:
    try:
        year = int(input("Ingrese el año (entre 1984 y 2023): "))
        if 1984 <= year <= 2023:
            break
        else:
            print("Error: Por favor, ingrese un año válido dentro del rango dado.")
    except ValueError:
        print("Error: Por favor, ingrese un año válido (número entero).")

# Do the API request with the obtained year from the user
url = f'https://api.api-ninjas.com/v1/cars?limit=50&year={year}'
url_response = requests.get(url, headers={'X-API-Key': 'J8q/mbRgr6+d0Zj5S0waeg==AyIvEPXnblRz5u8z'})
#API key given from the api-ninjas.com website after you register

# Verify the API response using the value 200
if url_response.status_code == 200:
    information = url_response.json()
    # Print if it was successful
    print("Datos obtenidos exitosamente.")
else:
    print(f"Error al obtener datos. Código de estado: {url_response.status_code}")

## Transform (Modify the data set, delete unnecesary columns and add KPI)
#Use pandas to make information a DataSet
df = pd.DataFrame(information)

# Delete columns that wont be used because the year is given and we are calculating the average fuel
consumption. Drop deletes the specified columns
columns_to_delete = ['combination_mpg', 'year']
df_without_columns = df.drop(columns_to_delete, axis=1)

# Create a new column for the avg fuel consumption called avg_mpg
df_without_columns['avg_mpg'] = (df_without_columns['city_mpg'] + df_without_columns['highway_mpg']) / 2

# Create a new column for our base KPI of # cylinders / consumo promedio, the higher the value the
better. Rate between power and fuel consumption.
df_without_columns['kpi_consumption'] = (df_without_columns['cylinders'] /
df_without_columns['avg_mpg'])

# Reorganise column to give you the most relevant information first
column_order = ['make', 'model', 'class', 'kpi_consumption', 'transmission', 'fuel_type', 'drive',
'cylinders', 'displacement', 'highway_mpg', 'avg_mpg', 'city_mpg']
df_ordered = df_without_columns[column_order]

#Organise the table from max to min to identify the best vehicles based on the defined KPI
df_ordered = df_ordered.sort_values(by='kpi_consumption', ascending=False)

#Create a new index column
df_ordered = df_ordered.reset_index(drop=True)
#print(df_ordered)

#Ask the user if they want to download the DataFrame
download_choice = input("¿Desea descargar el DataFrame? (Yes or No): ").lower()

# Load - Docker .csv file creation
output_dir = '/output_dir/'
output = output_dir + 'cars_db.csv'

# Check user's choice and proceed accordingly
if download_choice == 'yes':
    df_ordered.to_csv(output, index=False)
    print('El DataFrame ha sido exportado a: ' + output)
else:
    print('No se descargará el DataFrame.')

#Load (Export the transformed data)
#Export the data in a .csv file type
path_csv = 'datos_automoviles.csv'
df_ordered.to_csv(path_csv, index=False)
print(f'DataFrame exportado a {path_csv}')
```