

MD002 - Infraestructuras de computación

Trabajo Práctico: ETL Project II

Perez Copello, Mateo

2 de diciembre de 2023



1. **Introducción**
2. **Proceso ETL**
 - 2.1. **Indicadores**
3. **Construcción del dockerfile**
 - 3.1. **Código python**
4. **Construcción del docker-compose**
5. **Output y conclusión**
6. **Anexo**

1. Introducción

El trabajo concentra su desarrollo en la realización de un script de python, que contenga todo el proceso de extracción, transformación y carga de los datos, la construcción de un dockerfile y la construcción de un docker compose. El objetivo final es lograr un acceso a la información extraída, transformada y cargada, desde diversos ordenadores.

El proceso se diagramó pensado desde un punto de vista decisorio, orientando nuestro trabajo específicamente entre nuestro input y nuestro output, generando un valor informativo en pos de poder tomar acciones de la misma con insights de mejor calidad.

La serie de videojuegos FIFA que EA Sports ha estado desarrollando desde la década de 1990 ha sido nuestro disparador práctico. Esta compañía ha lanzado nuevas ediciones del videojuego con una periodicidad anual, incorporando mejoras gráficas y mejoras de jugabilidad.

Para esto último, fueron desarrollando cada vez más y mejores estadísticas tanto del negocio futbolístico, como así también de los jugadores. No es extraño pensar que en busca de mejorar las características realistas del videojuego, las estadísticas que determinan a cada jugador sean cada vez más precisas. Es esto lo que nos conduce al gran supuesto que tiene el desarrollo de todo nuestro trabajo: tomamos las estadísticas realizadas como representativas de la realidad.

El hilo conductor de nuestro trabajo y por el cual fuimos tomando las decisiones técnicas pertinentes, fue el Barcelona Futbol Club (BFC). La demanda por parte de esta institución consiste en tener la capacidad de analizar diariamente el rendimiento actual de todos los jugadores profesionales del mundo, para brindar mejores insights al área deportiva de detección de jóvenes talentos. El BFC tiene scouts repartidos en distintos países estratégicos para la institución, y quieren tener la capacidad de poder observar de manera prematura los futuros talentos. Además, nos remarcaron que la restricción presupuestaria es totalmente operativa en este

área, con lo cual tanto la valuación de mercado y como los flujos futuros estimados de estos jugadores “promesas”, terminan siendo sumamente relevantes a la hora de realizar una nueva incorporación.

Nuestro trabajo se basa en la construcción de un indicador de actualización diaria para que puedan ejecutar los propios scouts, que se encuentran en diversas zonas geográficas estratégicas, representando un input de alto impacto, ya que les permite no solo analizar los aspectos futbolísticos, sino también los aspectos económicos.

2. Proceso ETL

El desarrollo del trabajo ETL Project II presentó varios desafíos desde su comienzo. En primer lugar, pensar respecto a la actualización diaria de las mismas. Al basarnos en los datos del rendimiento actual de los jugadores, nos vimos en la necesidad de tener un esquema dinámico a la hora de obtener los datos. Si bien los números estadísticos son públicos, obtener los mismos en una base de datos estructurada no es algo de fácil acceso, y menos aún actualizados de manera diaria. Es por ello, que planteamos una extracción de datos a través del método de *web scraping* a través de Sofifa.

¿Qué es Sofifa? Es un sitio web que proporciona datos y estadísticas detalladas sobre jugadores de fútbol para el juego de video FIFA, desarrollado por EA Sports. Aunque no hay información detallada sobre quién creó Sofifa, el sitio ha sido ampliamente utilizado por la comunidad de jugadores de FIFA desde su creación.

La finalidad principal de Sofifa es ofrecer a los jugadores de FIFA información actualizada sobre los atributos y estadísticas de los jugadores que pueden ser utilizados en el modo Ultimate Team y otros modos de juego. Estos datos incluyen detalles como la velocidad, la habilidad de disparo, la capacidad defensiva, entre otros aspectos que son fundamentales en el juego, potencial, entre otros.

2.1. Indicadores

Nuestro trabajo se concentra en tres variables generales: potencial, valor de mercado y salario de cada jugador, para la construcción de un indicador final que utilizaremos como referencia decisiva.

En el proceso de transformación de nuestras tres variables encontramos diversas problemáticas. Para el caso del valor de mercado y el salario, las variables eran obtenidas en *string*, por ejemplo: 10K, lo que representa una complicación a la hora de utilizar las mismas como medidas. Como solución, se llevó a cabo un proceso de manipulación de las variables para poder transformarlas tanto a un formato numérico, como a una métrica común.

i- Potencial: Si bien nuestra fuente de datos es Sofifa, en la realización de este trabajo uno de los indicadores claves fue el potencial. Este atributo es establecido por EA Sports, el desarrollador del juego, el cuál representa el potencial máximo que un jugador puede alcanzar en su carrera en el juego. EA Sports determina estos valores basándose en su evaluación de las habilidades y el rendimiento de los jugadores en el mundo real. Tal como dijimos, Sofifa recopila y presenta esta información proporcionada por EA Sports en su base de datos. Por lo tanto, cuando extraemos este atributo en Sofifa, estamos viendo la evaluación estimada de EA Sports sobre el máximo potencial que un jugador puede alcanzar en el juego/vida real. Este atributo, es un estadístico altamente elaborado que nos brinda un resumen del rendimiento actual y esperado de los jugadores profesionales de todo el mundo.

ii- Valor de mercado: el valor de mercado representa la cotización diaria de cada jugador en el mercado de pases. Está determinado por el rendimiento del jugador a la hora de firmar su contrato.

iii- Valor del sueldo: el valor del sueldo está determinado por el rendimiento del jugador a la hora de firmar el contrato.

Para armar estos indicadores cogimos las tres variables descritas anteriormente y las dividimos por cuartiles. Cada observación se encuentra dentro de un cuartil estadístico que determina su categorización final:

| Cuartil | Potencial | Valor de Mercado | Valor de Salario |
|---------|-----------|------------------|------------------|
| Q1 | Muy bajo | Muy bajo | Muy bajo |
| Q2 | Bajo | Bajo | Bajo |
| Q3 | Alto | Alto | Alto |
| Q4 | Muy Alto | Muy Alto | Muy Alto |

Luego de obtener cada categorización por variable procedimos a armar un único indicador que contenga un resumen de estos tres, quedando por ejemplo:

| Nombre | Potencial | Valor de Mercado | Valor de Salario |
|-------------------|-----------|------------------|------------------|
| Lionel Messi | Muy alto | Muy alto | Muy alto |
| Cristiano Ronaldo | Muy bajo | Muy alto | Muy alto |

| Nombre | Indicador |
|-------------------|---|
| Lionel Messi | Muy alto potencial, muy alto valor, muy alto salario. |
| Cristiano Ronaldo | Muy bajo potencial, muy alto valor, muy alto salario. |

3. Construcción del dockerfile

1- FROM python:3.11-slim

Para la construcción del dockerfile procedimos a diseñar una imagen que contenga "python:3.11-slim". La decisión de escoger una versión de python específica se basa en tener una versión que se actualice, pero que no sea la última disponible, ya que estas pueden llegar a tener algunos errores de lanzamiento. Por otro lado, la versión

slim se escogió en busca de eficientizar los recursos, ya que estas cuentan con las funcionalidades esenciales para el funcionamiento.

```
2- RUN apt-get update && apt-get -y install git && rm -rf  
/var/lib/apt/lists/*
```

Luego decidimos correr un apt-get update para poder contener todos nuestros paquetes actualizados a medida que se ejecuta el contenedor, y el instalador del paquete git. Esta decisión se tomó en base a dos cuestiones. La primera razón es que tener nuestro código en git permite una fácil actualización, sin tener que volver a enviar el mismo a cada scout cada vez que se actualice. Y la segunda, porque nuestro grupo de desarrolladores trabaja con una modalidad híbrida, con lo cual tener el código en git permite un acceso remoto y actualizado de manera constante.

```
3- ADD . /app
```

Esta línea permite copiar todos los archivos del dominio en docker, es totalmente funcional a nuestra estructura.

```
4- WORKDIR /app
```

Luego de copiar todos nuestros archivos a docker, corremos la función *workdir* para indicar cuál será nuestro directorio de ahora en más.

```
5- RUN pip install -r requirements.txt
```

Ejecutamos el código run con un pip install para instalar en nuestra receta de imagen todas las librerías de python que necesitamos utilizar para este proceso ETL.

Para simplificar el código construimos un archivo txt llamado "requirements.txt" que contiene todas las librerías utilizadas en nuestro código de ejecución. Este código cuenta con los siguientes paquetes: prequests, beautifulsoup4, pandas, matplotlib, plotly, psycpg2 y sqlalchemy. Estos paquetes nos habilitan a descargar y ejecutar las siguientes librerías:

Requests: Esta librería es necesaria para realizar solicitudes HTTP.

Soup: Esta librería es necesaria para procesar la respuesta HTML.

Pandas: Esta librería es necesaria para manipular y analizar los datos.

Matplotlib: Esta librería es necesaria para crear gráficos estáticos, animados e interactivos en Python.

Plotly express: Esta librería es necesaria para crear gráficos interactivos.

También usamos librerías básicas dentro del paquete de python que no necesitan ser instaladas mediante el txt.

Time: Esta librería es necesaria para añadir un retraso entre las solicitudes.

Random: Esta librería es necesaria para generar un retraso aleatorio.

```
6- RUN mkdir -p /jugadores_promesa
```

Esta función es totalmente elemental para la construcción de todo nuestro proceso ETL, ya que es la encargada de crear un directorio donde se guarden todos los outputs que tenemos en nuestra aplicación.

```
7- RUN apt-get update \  
    && apt-get install -y postgresql-client \  
    && apt-get clean \  
    && rm -rf /var/lib/apt/lists/*
```

Esta línea se corresponde con una de nuestras funcionalidades otorgadas en el proceso ETL, donde creamos un repositorio de postgresql para cargar una tabla final que contenga la información actualizada de todos nuestros jugadores. Se realiza esto en función de tener la capacidad de conectar la tabla a algún cuadro de mando que permita visualizar la información de manera gráfica. Aunque este desarrollo ya corresponderá a cada scout. Además, actualiza y limpia los paquetes, con la finalidad de reducir el tamaño de la imagen.

```
8- CMD ["python", "Scrapping.py"]
```


Por último, tenemos el comando CMD. Luego de hacer todo este recorrido de recolección, actualización, limpieza y guardado, el dockerfile finaliza con un código que deberá correr. Este código, llamado “Scraping.py”, contiene todo nuestro proceso ETL.

3.1. Código Python

Tal como lo anticipamos, acá vimos un gran desafío. Si bien las estadísticas brindadas por la compañía EA Sports son de público acceso, la captura de las mismas no es una labor sencilla, menos aún cuando se propone una aplicación dinámica que brinde datos actualizados de manera diaria. Para ello utilizamos la librería *soup* contenida dentro del paquete *beautifulsoup4*. Una de las partes más imperiosas del código y que más trabas presentó al principio fue la creación de un simulador de ordenador con extracciones temporales randomizadas entre 0 y 10 segundos, dado que la solicitud sin esto era bloqueada.

```
for offset in range(0, 2):
    url = url_base + str(offset * 61)

    # Añadir un retraso aleatorio entre las solicitudes
    delay = random.uniform(1, 10)
    time.sleep(delay)

    # Configurar encabezados HTTP para simular un navegador real
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
    }

    # Realizar la solicitud con encabezados y retraso
    p_html = requests.get(url, headers=headers)

    # Verificar si la solicitud fue exitosa antes de procesar la respuesta
    if p_html.status_code == 200:
        p_soup = p_html.text
        data = Soup(p_soup, 'html.parser')
        table = data.find('tbody')
print(table)
```

Luego encontramos que al estar paginada la url, y poseer únicamente 60 jugadores por página, debíamos cambiar la url fuente para poder hacer contar con una cantidad de jugadores significativa para este ejercicio. Es por ello que nos vimos obligados a reiterar el código repetidas veces. Conformamos una base de 420 jugadores, con la cual trabajamos en su transformación con el objetivo concreto de nuestra carga final.

Como explicamos anteriormente, luego de la extracción, procedimos con la construcción de nuestro indicador. Este proceso se basó en la transformación de las variables de interés adaptándolas a nuestro objetivo final: reglas estadísticas para la categorización. Necesitamos que todas las variables contenidas dentro del mismo estén en formato numérico, para poder clasificarlas en cuartiles, quedándonos de la siguiente manera:

```
# Calcula cuartiles para Value, Wage y Potential
value_quartiles = pd.qcut(df_nuevo['Value_Millones'], q=[0, 0.25, 0.5, 0.75, 1], labels=['Muy Bajo', 'Bajo', 'Alto', 'Muy Alto'])
wage_quartiles = pd.qcut(df_nuevo['Wage_Miles'], q=[0, 0.25, 0.5, 0.75, 1], labels=['Muy Bajo', 'Bajo', 'Alto', 'Muy Alto'])
potential_quartiles = pd.qcut(df_nuevo['Potential'], q=[0, 0.25, 0.5, 0.75, 1], labels=['Muy Bajo', 'Bajo', 'Alto', 'Muy Alto'])

# Combina las etiquetas para obtener categorías combinadas
df_nuevo['indicador'] = (
    value_quartiles.astype(str) + ' Value, ' +
    wage_quartiles.astype(str) + ' Wage, ' +
    potential_quartiles.astype(str) + ' Potential'
)

# Nos quedamos con las variables de interes
columnas_seleccionadas = ['Name', 'Value_Millones', 'Wage_Miles', 'Potential',
                           'indicador', 'Age', 'Team', 'picture', 'ID', 'flag', 'Position',
                           'Team_image']
df_nuevo = df_nuevo[columnas_seleccionadas]
```

Por ultimo, habilitamos un puerto de posgreSQL para hacer la carga de nuestro data frame final:

```
# Parámetros de conexión a la base de datos PostgreSQL
db_params = {
    'host': 'postgres',
    'port': '5432',
    'user': 'scout_barcelona',
    'password': 'mesqueunclub',
    'database': 'tabla_jugadores'
}

# Establecer la conexión con la base de datos
connection = psycopg2.connect(**db_params)

# Crear un motor SQLAlchemy usando la conexión PostgreSQL
engine = create_engine(f"postgresql+psycopg2://{db_params['user']}:{db_params['password']}@{db_params['host']}:{db_params['port']}/{db_params['database']}")

# Nombre de la tabla en la base de datos PostgreSQL
tabla_jugadores = 'tabla_jugadores'

# Cargar el DataFrame en la tabla existente o crear una nueva
df_nuevo.to_sql(tabla_jugadores, engine, if_exists='replace', index=False)

# Cerrar la conexión
connection.close()
```

4. Construcción del docker-compose

La construcción del docker-compose tiene como objetivo dar de alta un servicio de PgAdmin en el localhost creado en docker, al cual se pueda acceder con usuario y contraseña a la base de datos final resultante del script de python, llamada "tabla_jugadores". Como pgadmin tiene cargada su imagen en docker hub, este interpreta y descarga automáticamente la misma.

```
pgadmin:
  image: dpage/pgadmin4
  ports:
    - "5050:80"
  environment:
    - PGADMIN_DEFAULT_EMAIL=scout@bfc.com
    - PGADMIN_DEFAULT_PASSWORD=mesqueunclub
```

Luego, levantamos el servicio para nuestra aplicacion python, y le indicamos un build context. Este ultimo es muy importante, ya que le permitira contextualizar el dominio de ejecucion para coger el dockerfile correspondiente. Se configura los volumenes para buscar en el dominio el codigo de python que se debe ejecutar. Por ultimo, el environment configura las variables de entorno para la aplicación python, especificando la información de conexión a la base de datos PostgreSQL.

```
python_app:
  build:
    context: .
  volumes:
    - ./Scrapping.py:/app/Scrapping.py
    - ./data:/app/data
  environment:
    - POSTGRES_HOST=postgres
    - POSTGRES_USER=scout_barcelona
    - POSTGRES_PASSWORD=mesqueunclub
    - POSTGRES_DB=tabla_jugadores
```

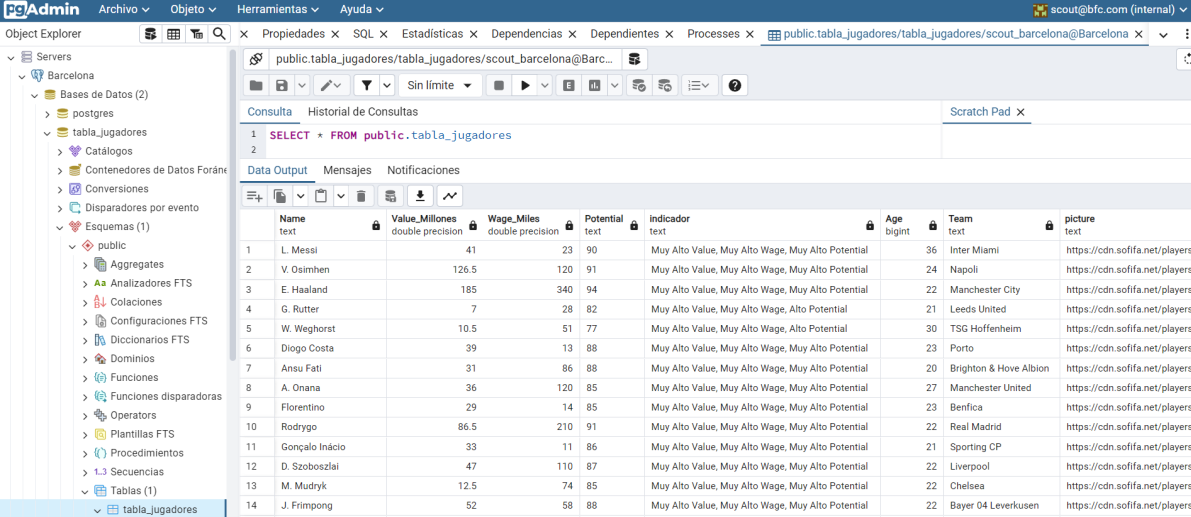
Por ultimo, damos de alta el servicio de PostgreSQL, que al igual que el servicio de PgAdmin, cuenta con su imagen oficial cargada en docker hub. Configuramos los accesos de nuestro usuario, y picamos un depends_on: - python_app: para asegurarnos que el servicio python_app se inicie antes de iniciar el servicio postgres.

```
postgres:
  image: postgres:latest
  environment:
    - POSTGRES_USER=scout_barcelona
    - POSTGRES_PASSWORD=mesqueunclub
    - POSTGRES_DB=tabla_jugadores
  depends_on:
    - python_app
```

5. Output y conclusión¹

El objetivo de nuestra propuesta fue cumplido, hemos podido desarrollar un proceso de ETL mediante la utilización de docker de manera exitosa. Hemos creado una infraestructura de desarrollo que incluye una base de datos PostgreSQL, una interfaz de administración pgAdmin, y la aplicación Python que interactúa con la base de datos. Además, hemos situado un csv al cual se puede acceder desde el docker con toda la información, sin necesidad de tener que abrir el PgAdmin para realizar la visualización. Tal como se planteó al principio, la idea es brindarle este soporte a los scouts para que puedan tener el acceso actualizado a la información de los jugadores. En la tabla agregamos datos del jugador, tal como su imagen, su club, su posición, lo que les va a permitir desarrollar informes estructurados al momento de ir a observar las condiciones técnicas del mismo. Que tengan la posibilidad de acceder a su imagen es muy importante, ya que al estar buscando talento los jugadores no suelen ser conocidos y la imagen facilita su distinción. También el scout podrá conectar un cuadro de mando a esta aplicación para tener los datos actualizados. En el siguiente print pueden ver que desde el localhost, al cual se ingresa con mail y contraseña configuradas, vemos la tabla_jugadores.

2



| | Name | Value_Millones | Wage_Miles | Potential | Indicador | Age | Team | picture |
|----|----------------|------------------|------------------|-----------|---|--------|------------------------|--------------------------------|
| | text | double precision | double precision | text | | bigint | text | text |
| 1 | L. Messi | 41 | 23 | 90 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 36 | Inter Miami | https://cdn.sofifa.net/players |
| 2 | V. Osimhen | 126.5 | 120 | 91 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 24 | Napoli | https://cdn.sofifa.net/players |
| 3 | E. Haaland | 185 | 340 | 94 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 22 | Manchester City | https://cdn.sofifa.net/players |
| 4 | G. Rutter | 7 | 28 | 82 | Muy Alto Value, Muy Alto Wage, Alto Potential | 21 | Leeds United | https://cdn.sofifa.net/players |
| 5 | W. Weghorst | 10.5 | 51 | 77 | Muy Alto Value, Muy Alto Wage, Alto Potential | 30 | TSG Hoffenheim | https://cdn.sofifa.net/players |
| 6 | Diogo Costa | 39 | 13 | 88 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 23 | Porto | https://cdn.sofifa.net/players |
| 7 | Ansu Fati | 31 | 86 | 88 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 20 | Brighton & Hove Albion | https://cdn.sofifa.net/players |
| 8 | A. Onana | 36 | 120 | 85 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 27 | Manchester United | https://cdn.sofifa.net/players |
| 9 | Florentino | 29 | 14 | 85 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 23 | Benfica | https://cdn.sofifa.net/players |
| 10 | Rodrygo | 86.5 | 210 | 91 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 22 | Real Madrid | https://cdn.sofifa.net/players |
| 11 | Gonçalo Inácio | 33 | 11 | 86 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 21 | Sporting CP | https://cdn.sofifa.net/players |
| 12 | D. Szoboszlai | 47 | 110 | 87 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 22 | Liverpool | https://cdn.sofifa.net/players |
| 13 | M. Mudryk | 12.5 | 74 | 85 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 22 | Chelsea | https://cdn.sofifa.net/players |
| 14 | J. Frimpong | 52 | 58 | 88 | Muy Alto Value, Muy Alto Wage, Muy Alto Potential | 22 | Bayer 04 Leverkusen | https://cdn.sofifa.net/players |

¹ <https://youtu.be/Tt8MpeueSVY> a través de este video se puede ver la interacción con el entorno

² <https://youtu.be/JBIQ7phZgiM> a través de este video se puede acceder a un gráfico interactivo

6. Anexo

The screenshot shows the Docker Desktop interface with the 'etl2_muds' container selected. The container is running and displaying a list of 240 rows x 19 columns of data. The data includes player names, team names, and various statistics. The interface shows the container's logs and the Docker Desktop status bar at the bottom.

3

The screenshot shows the Docker Desktop interface with the 'etl2_muds-python_app-1' container selected. The container is running and displaying a file explorer view of the container's filesystem. The files and directories are listed with their names, sizes, and last modified dates. The interface shows the container's logs and the Docker Desktop status bar at the bottom.

```
PS C:\Users\Mateo> cd "C:\Users\Mateo\Documents\Mateo\Data Science - La Salle\MD002 - Infraestructuras de computacion\Trabajos Practicos\TP2\ETL2\ETL2_MUDS"
PS C:\Users\Mateo\Documents\Mateo\Data Science - La Salle\MD002 - Infraestructuras de computacion\Trabajos Practicos\TP2\ETL2\ETL2_MUDS> docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
88bae5c5b02a   postgres:late  "docker-entrypoint.s..." About an hour ago Up About an hour 5432/tcp       etl2_muds-postgres-1
b3fe923d7cc5   dpape/pgadmin4 "/entrypoint.sh"        About an hour ago Up About an hour 443/tcp, 0.0.0.0:5050->80/tcp etl2_muds-pgadmin-1
PS C:\Users\Mateo\Documents\Mateo\Data Science - La Salle\MD002 - Infraestructuras de computacion\Trabajos Practicos\TP2\ETL2\ETL2_MUDS>
```

³ El servicio de python se cierra luego de su ejecución para liberar recursos.