**Aplicación para gestión de capacitación a usuarios**

# Índice

Contenido

[Índice 1](#_Toc194028344)

[Requisitos de la aplicación 2](#_Toc194028345)

[Arbol de directorios 3](#_Toc194028346)

[Creación del Archivo docker-compose.yml 4](#_Toc194028347)

[Creación de un archivo .env 4](#_Toc194028348)

[Creación de Dockerfile 5](#_Toc194028349)

[Scripts SQL 7](#_Toc194028350)

[Creacion\_de\_tablas.sql 7](#_Toc194028351)

[Generación de datos para las tablas. 9](#_Toc194028352)

[Verificaciones 9](#_Toc194028353)

[Diagrama final de la base de datos 10](#_Toc194028354)

[Creación manual de datos 10](#_Toc194028355)

[Aclaraciones 11](#_Toc194028356)

[Otros Comandos Docker 11](#_Toc194028357)

[Scripts de Python 11](#_Toc194028358)

[Elasticsearch y Kibana 13](#_Toc194028359)

[Fuentes 17](#_Toc194028360)

## Requisitos de la aplicación

**Docker:** Tecnología de la organización en contenedores que posibilita la creación y el uso de los contenedores de Linux. Esto es lo único que tenemos que tener instalado en nuestra maquina para que pueda ser ejecutada nuestra aplicación sin inconvenientes.

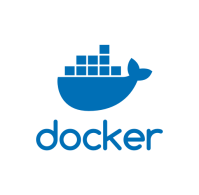
**Ubuntu:** Distribución de GNU/Linux basada en Debian, que incluye principalmente software libre y de código abierto

**Python:** Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes.

**Mysql:** Sistema de gestión de bases de datos relacionales (RDBMS) de código abierto que se utiliza para almacenar y gestionar datos.

**Kibana:** La plataforma de código abierto Kibana se utiliza para analizar los datos recogidos del motor de búsqueda Elasticsearch. Kibana es una plataforma de análisis y visualización de código abierto basada en el navegador.

**Elasticsearch:** Motor de búsqueda y analítica de RESTful distribuido, un almacén de datos escalable y una base de datos de vectores capaz de abordar un número creciente de casos de uso.



## Arbol de directorios

Este árbol de directorios debe respetarse, es la estructura escencial para que podamos ejecutar nuestros comandos de Docker sin inconvenientes, simplemente estando en la ruta donde alojemos el “Desafio Meli”, y entrando en la carpeta Aplicacion

Desafio Meli\_/

  |\_ Aplicacion/

      |\_ .env

|\_ Recursos

      |\_ docker-compose.yml

      |\_ Dockerfile

      |\_ requirements.txt

      |\_ app/

          |\_ calcular\_metricas.py

          |\_ calcular\_metricas\_date\_utils.py

          |\_ calcular\_metricas\_metrics\_app.py

          |\_ calcular\_metricas\_metrics\_calculator.py

          |\_ utils.py

          |\_ main.py

          |\_ generar\_data.py

          |\_ generar\_data\_insert\_trainings.py

          |\_ generar\_data\_insert\_users.py

          |\_ generar\_data\_insert\_user\_trainings.py

          |\_ generar\_data\_random\_date.py

|\_ transferir\_a\_elasticsearch.py

|\_ start.sh

      |\_ data/

      |\_ init-scripts/

          |\_ creacion\_de\_tablas.sql

## Creación del Archivo docker-compose.yml

Este archivo es un archivo de configuración que define y ejecuta aplicaciones con Docker, unificando en este único archivo los servicios, redes y volúmenes necesarios.

Utilizando el comando **docker-compose up** se puede levantar todo el entorno de manera automatizada.

Este archivo nos ayudará a:

1. **Levantar la base de Datos MySQL**

La base de datos nos ayudara a tener un entorno para almacenar los datos de los usuarios, capacitaciones o métricas. Con esto, tenemos la ventaja de que la BBDD este siempre disponible y configurada de la misma manera sin importar el Sistema operativo

1. **Gestionar la comunicación entre servicios**

Componentes como MySQL y Elasticsearch puedan interactuar entre sí; creando una red compartida en la que los contenedores se comunican usando nombres de servicio como hostnames.

1. **Asegurar la portabilidad y reproducibilidad**

Hacer que el entorno completo sea fácil de replicar en cualquier host o Sistema Operativo, esto es posible a que todas las configuraciones quedan en un único archivo. Usando el comando **docker-compose up**, cualquiera puede levantar el proyecto sin instalar manualmente cada componente.

## Creación de un archivo .env

Este archivo nos permitirá manejar configuraciones sensibles y personalizadas de manera segura y eficiente. En el archivo se necesitara definir variables sensibles como passwords y claves APIs.

Como practica de seguridad, este archivo se pondría en el **Github** como **.env.example;** dicho archivo incluiría las variables necesarias pero sin los valores sensibles, luego simplemente se renombra a **.env** y cada uno le puede agregar sus propios valores. Dado que este es un Challenge y tiene que estar operativo, el .env quedara tal cual en el Github.

En este archivo se podrán alojar

* Credenciales y configuraciones de bases de datos
* Tokens y credenciales de autenticación
* Configuraciones de servicios externos
* Variables específicas de entorno
* Configuraciones de redes

## Creación de Dockerfile

Este archivo de texto actúa como el paso a paso para construir una imagen en Docker; contiene todo lo que nuestra aplicación necesita para funcionar dentro de un contenedor Docker, como las dependencias, archivos del proyecto y las instrucciones para ejecutarla.

Con este archivo tenemos la ventaja de :

1. Crear un entorno específico para la aplicación
2. Asegurar consistencias entre entornos
3. Hacen la aplicación “portátil”
4. Simplifica el despliegue

Este Dockerfile crea una imagen utilizando una base ligera de Python (Python :3.9-slim), instala todo lo necesario como dependencias del sistema y Python, organiza los códigos y los datos. La parte fundamental a la hora de crear este Dockerfile fue la configuración para un rendimiento eficiente. No se utiliza el usuario root para garantizar buenas prácticas de seguridad. Además no tenemos que ejecutar ningún Script de generación de datos, solo tenemos que aguardar un momento para que Elasticsearch este disponible para crear los dashboards

**Archivo Dockerfile**

#Imagen base para nuestro contenedor, Python slim como base.

FROM python:3.9-slim

#Directorio de trabajo en nuestro contenedor

WORKDIR /app

# Instalación de dependencias del sistema: paquetes, compilador de C que es gcc, la biblioteca para conectar a mysql y eliminacion de archivos temporales

RUN apt-get update && \

    apt-get install -y --no-install-recommends \

    gcc \

    default-libmysqlclient-dev \

    && rm -rf /var/lib/apt/lists/\*

# Establecer variables de entorno para la compilación, el primero nos sirve para que Python no genere archivos .pyc, el segundo desactiva el almacenamiento en el

#buffer, lo que permite mostrar en tiempo real  los logs y los mensajes.

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Copiar requirements y instalar dependencias, evitando que se guarde en la caché local

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Crear directorio para los datos

RUN mkdir -p /data

# Copiar el código de la aplicación a esta direccion dentro de esta direccion

COPY ./app /app

# Crear un usuario no root para la aplicación, lo que permite leer y escribir

RUN adduser --disabled-password --gecos '' appuser

RUN chown -R appuser:appuser /app /data

# Cambiar al usuario no root, con la intencion de que el comando siguiente sea ejecutado con este usuario

USER appuser

# Comando para ejecutar la aplicación de inicializacion

COPY ./app /app

COPY --chmod=0755 ./app/start.sh /app/

CMD ["/bin/bash", "/app/start.sh"]

**¿Por qué decidí usar Docker?**

Considero que Docker es una solución práctica y eficiente para gestionar el entorno de desarrollo y del despliegue. Docker nos permite encapsular servicios como la base de datos MySQL, Elasticsearch y Kibana, lo que garantiza consistencia y portabilidad en diferentes máquinas y Sistemas Operativos, reduciendo errores, agregando escalabilidad y facilidad de mantenimiento.

## Scripts SQL

Los scripts SQL desempeñan un papel crucial en la configuración y gestión de las base de datos.

Tenemos un script SQL llamados **creacion\_de\_tablas.sql**, el cual se encarga de crear las cuatro tablas que necesitamos: **usuarios, capacitaciones, capacitaciones\_por\_usuarios y métricas\_de\_capacitaciones**. Cada tabla está definida con las columnas y sus configuraciones correspondientes.

### Creacion\_de\_tablas.sql

-- 01\_create\_users\_table.sql

-- Script para crear la tabla de usuarios

CREATE TABLE IF NOT EXISTS usuarios (

    ID INT AUTO\_INCREMENT PRIMARY KEY,

    USERNAME VARCHAR(50) NOT NULL UNIQUE,

    START\_DATE DATE NOT NULL,

    END\_DATE DATE DEFAULT NULL,

    BUSINESS\_UNIT VARCHAR(100) NOT NULL,

    MANAGER VARCHAR(50) NOT NULL,

    LAST\_UPDATE DATE NOT NULL,

    IS\_EXTERNAL BOOLEAN NOT NULL,

    INDEX idx\_username (USERNAME),

    INDEX idx\_business\_unit (BUSINESS\_UNIT),

    INDEX idx\_end\_date (END\_DATE)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

-- 02\_create\_trainings\_table.sql

-- Script para crear la tabla de capacitaciones

CREATE TABLE IF NOT EXISTS capacitaciones (

    ID INT AUTO\_INCREMENT PRIMARY KEY,

    NAME VARCHAR(100) NOT NULL,

    LINK VARCHAR(255) NOT NULL,

    CREATION\_DATE DATE NOT NULL,

    INDEX idx\_name (NAME)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

-- 03\_create\_user\_trainings\_table.sql

-- Script para crear la tabla de relación entre usuarios y capacitaciones

CREATE TABLE IF NOT EXISTS capacitaciones\_por\_usuarios (

    ID INT AUTO\_INCREMENT PRIMARY KEY,

    FK\_USERNAME VARCHAR(50) NOT NULL,

    FK\_TRAINING INT NOT NULL,

    END\_DATE DATE DEFAULT NULL,

    ASSIGNMENT\_DATE DATE NOT NULL,

    INDEX idx\_fk\_username (FK\_USERNAME),

    INDEX idx\_fk\_training (FK\_TRAINING),

    INDEX idx\_end\_date (END\_DATE),

    FOREIGN KEY (FK\_USERNAME) REFERENCES usuarios(USERNAME) ON DELETE CASCADE,

    FOREIGN KEY (FK\_TRAINING) REFERENCES capacitaciones(ID) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

-- 04\_create\_metrics\_table.sql

-- Script para crear la tabla de métricas históricas

CREATE TABLE IF NOT EXISTS metricas\_de\_capacitaciones (

    ID INT AUTO\_INCREMENT PRIMARY KEY,

    BUSINESS\_UNIT VARCHAR(100) NOT NULL,

    MONTH DATE NOT NULL,

    COMPLETION\_PERCENTAGE DECIMAL(5,2) NOT NULL,

    TOTAL\_USERS INT NOT NULL,

    COMPLETED\_TRAININGS INT NOT NULL,

    CALCULATION\_DATE TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    INDEX idx\_business\_unit (BUSINESS\_UNIT),

    INDEX idx\_month (MONTH)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

Las columnas con **INDEX** crean índices en columnas específicas de las tablas, como lo vemos en el código SQL. Esto permite hacer consultas con WHERE, JOIN y ORDER BY de una manera mucho más rápida y ayuda a gestionar recursos.

Las columnas que tienen **CALCULATION\_DATE** almacenan la fecha y la hora exacta en que se insertó la métrica y el **DEFAULT CURRENT\_TIMESTAMP** inserta automáticamente la fecha y hora actual cuando se agrega un nuevo registro.

El **ENGINE=InnoDB** define las configuraciones de la tabla, **InnODB** es el motor de almacenamiento más modero en MySQL el cual soporta transacciones y claves foráneas.

**UTF8mb4** permite almacenar texto en cualquier idioma

**Unicode\_ci** hace que las comparaciones de texto sean insensibles a mayúsculas y tildes

## Instalación

**Requisitos previos comunes**

* Asegurarse de tener Docker y Docker Compose instalados y funcionando correctamente
* Git instalado o posibilidad de descargar el repositorio como ZIP
* Puertos disponibles: 3306, 9200, 25601

### Para usuarios de Linux

Clonar el repositorio:

**git clone https://github.com/Ghostsecure/Desafio-Meli.git**

**cd Desafio-Meli**

### Para usuarios de Windows

Clonar el repositorio:

Utilizando Git Bash:

**git clone https://github.com/Ghostsecure/Desafio-Meli.git**

**cd Desafio-Meli**

O descargar como ZIP desde GitHub y descomprimirlo

### Requisitos específicos para Windows:

* Tener Docker Desktop para Windows instalado
* Configurar Docker Desktop en modo WSL 2 o Hyper-V
* Asegurarse de que Docker Desktop esté en ejecución
* Iniciar el entorno (desde PowerShell o CMD):

## Generación de datos para las tablas.

Primero que nada tenemos que generar el contenedor correspondiente en docker, para ello tenemos que ejecutar el siguiente comando en la terminal de **Docker**, o en PowerShell de nuestro Windows, donde tenemos nuestros arhivos de docker**(.env, docker-compose.yml, Dockerfile y requirements.txt**).

**docker-compose up -d**

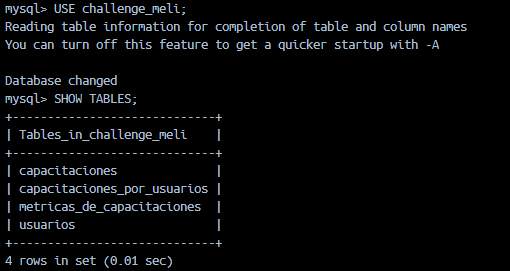
Eso es todo lo que tenemos que hacer, no tenemos que ejecutar ningún script ya que a la hora de levantar los contenedores todo se ejecutará haciendo que sea transparente para nosotros y nos centremos en la creación de los dashboards en Kibana. Si esperemos que los servicios se levanten de manera correcta, dejar al menos dos minutos para que MySQL y Elasticsearch se terminen de ejecutar y cargar sus entornos.

## Verificaciones

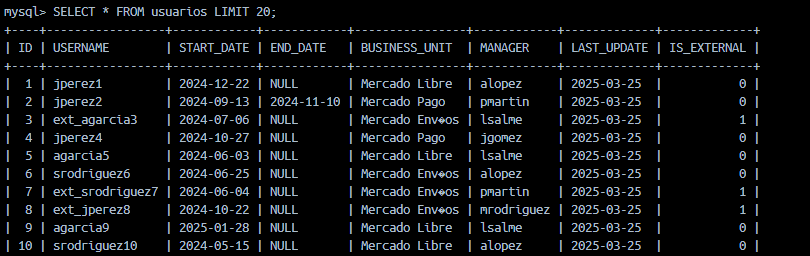
Para verificar una correcta creación de los datos podemos acceder a la base de datos de nuestro contenedor usando el siguiente comando, el cual luego, nos pedirá la contraseña que seteamos el nuestro archivo .env bajo la variable **MYSQL\_PASSWORD**:

**docker exec -it meli\_mysql mysql -u meli\_user –p**

Seleccionamos la base de datos **challenge\_meli** y ejecutamos una consulta para que nos muestre las tablas de la base de datos, la consulta es “**SHOW TABLES;**”:



Si queremos echar un vistazo a las tablas podemos ejecutar diferentes consultas para ver sus datos, como por ejemplo **SELECT \* FROM usuarios LIMIT 20;**



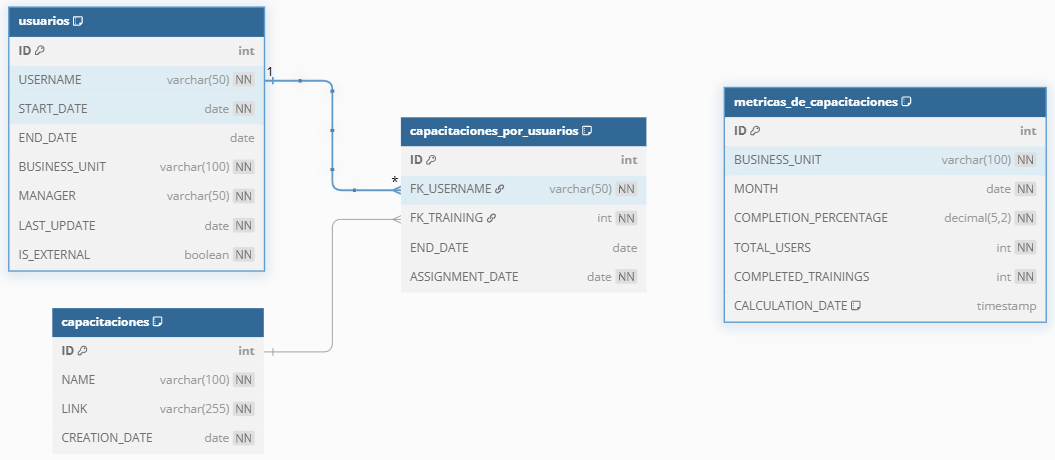
Si queremos ver las demás tablas simplemente cambiamos el **usuario** de la consulta anterior por **capacitaciones, capacitaciones\_por\_usuarios o metricas\_de\_capacitaciones.** Nos quedarían asi:

**SELECT \* FROM capacitaciones LIMIT 20;**

**SELECT \* FROM capacitaciones\_por\_usuarios LIMIT 20;**

**SELECT \* FROM metricas\_de\_capacitaciones LIMIT 20;**

## Diagrama final de la base de datos



## Creación manual de datos

Si en algún punto queremos crear nuevos datos y calcular nuevas métricas cuando ya todo el sistema está en ejecución, tenemos que ejecutar comandos específicos, estos pueden ser:

# Para generar nuevos datos

**docker exec -it meli\_app python /app/generar\_data.py**

# Para calcular nuevas métricas basadas en esos datos

**docker exec -it meli\_app python /app/calcular\_metricas.py**

# Para transferir las nuevas métricas a Elasticsearch

**docker exec -it meli\_app python /app/transferir\_a\_elasticsearch.py**

#limpiar datos existentes antes de generar nuevos, tenemos que contectarnos a MySQL

**docker exec -it meli\_mysql mysql -u meli\_user -p -e "USE challenge\_meli; DELETE FROM metricas\_de\_capacitaciones; DELETE FROM capacitaciones\_por\_usuarios; DELETE FROM usuarios; DELETE FROM capacitaciones;"**

### Aclaraciones

Podemos modificar los scripts para generar datos diferentes o calcular nuevas métricas con solo editar los scripts de nuestra carpeta /app local y estos cambios se verán reflejados automáticamente en el contenedor gracias al volumen montado.

Con la ejecución del script de envío de datos a Elasticsearch, lo único que debemos hacer es actualizar nuestros dashboards en Kibana y tendremos todo disponible para visualizar; en caso de que hayamos creado nuevos registros en nuestras tablas, necesitaremos actualizar los patrones de índices.

## Otros Comandos Docker

# Verificar el estado de los contenedores

**docker-compose ps**

# Verificar los logs de la aplicación

**docker logs meli\_app**

## Scripts de Python

Se decidió usar Python dado la utilidad del lenguaje y la usabilidad para este tipo de proyectos. Se desarrolló n scripts los cuales se pensaron para mantener un código organizado, utilizando la programación modular y con la idea que se pueda reutilizar.

Los scripts de Python que tiene nuestra aplicación son 11 en total, cada uno con una tarea en específica, la razón por la que sean 11 es porque se decidió usar la programación modular y que cada tarea se haga por script.

**Scripts Principales**

1. **main.py**: este es el script principal de toda la aplicación, es el encargado de ejecutar y programar los procesos de generación de datos y cálculo de métricas.
2. **generar\_data.py:** es el encargado de coordinar todos los datos de prueba, llama a los otros scripts de generación de usuarios, capacitación y asignaciones.
3. **calcular\_metricas.**py: inicia el proceso de cálculo de métricas de las capacitaciones y es el punto de entrada para el análisis de los datos.
4. **Start.sh:** script de tipo bash que inicia junto con los contenedores, en el se verfica que los servicios estén en ejecución para generar datos, calcule las métricas y transfiera los datos con la ejecución de los demás scripts que fueron creados para dicho propósito. Finaliza ejecutando el **main.py**

**Scripts de generación de datos**

1. **generar\_data\_insert\_users.py:** crea usuarios de pruebas internos y externos a los cuales les asigna las unidades de negocio y managers.
2. **generar\_data\_insert\_trainings.py:** crea las capacitaciones disponibles en el Sistema.
3. **generar\_data\_insert\_user\_training.py:** asigna las capacitaciones a los usuarios y también determina cuales están completadas y cuáles no.
4. **generar\_data\_random\_date.py:** genera fechas aleatorias.

**Scripts para cálculo de métricas**

1. **calcular\_metricas\_metrics\_app.py:** controla el flujo de cálculo de métricas y configura conexiones y maneja errores.
2. **calcular\_metricas\_metrics\_calculator.py:** contiene la lógica real del cálculo de métricas.
3. **calcular\_metricas\_date\_utils.py:** proporciona funciones para manejar fechas en el cálculo de métricas y obtiene estos periodos de tiempo para la generación de reportes.

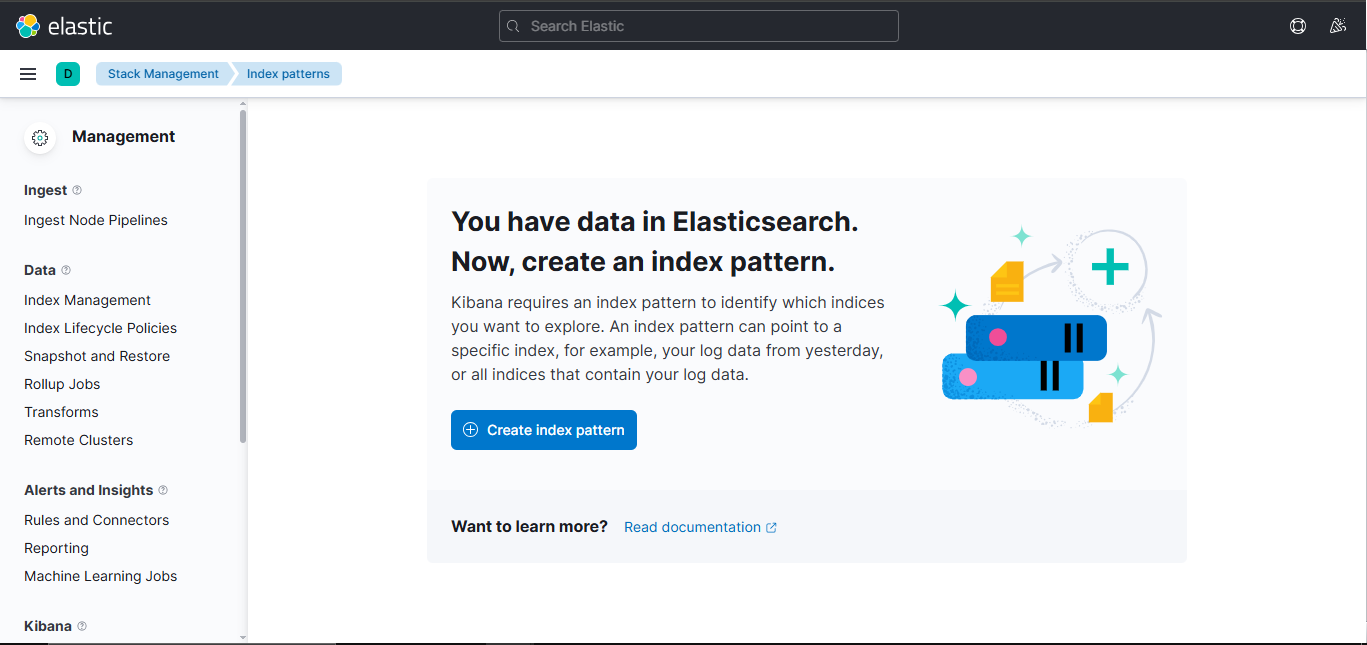
**Otros scripts**

1. **utils.py** contiene funciones de utilidad con la finalidad de que sea reutilizable por estos o más scripts, en él se manejan los loggings, conexiones a la base de datos y variables de entorno
2. **transferir\_a\_elasticsearch.py:** transfiere datos de MySQL a Elasticsearch, permite que Kibana visualice la información; tambien formatea los datos para que sean compatibles con Elasticsearch y crea índices para cada tabla de la base de datos.

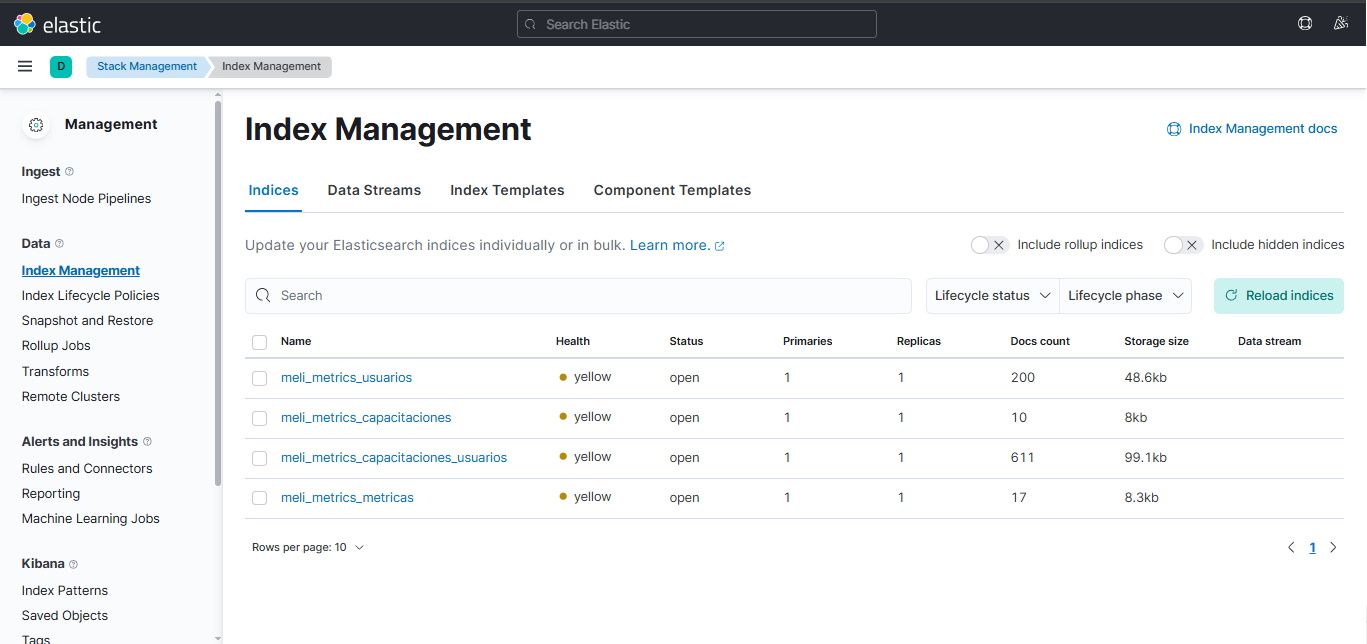
## Elasticsearch y Kibana

Podemos acceder a Elastic ejecutando Chrome y escribiendo en el navegador: **http://127.0.0.1:25601**

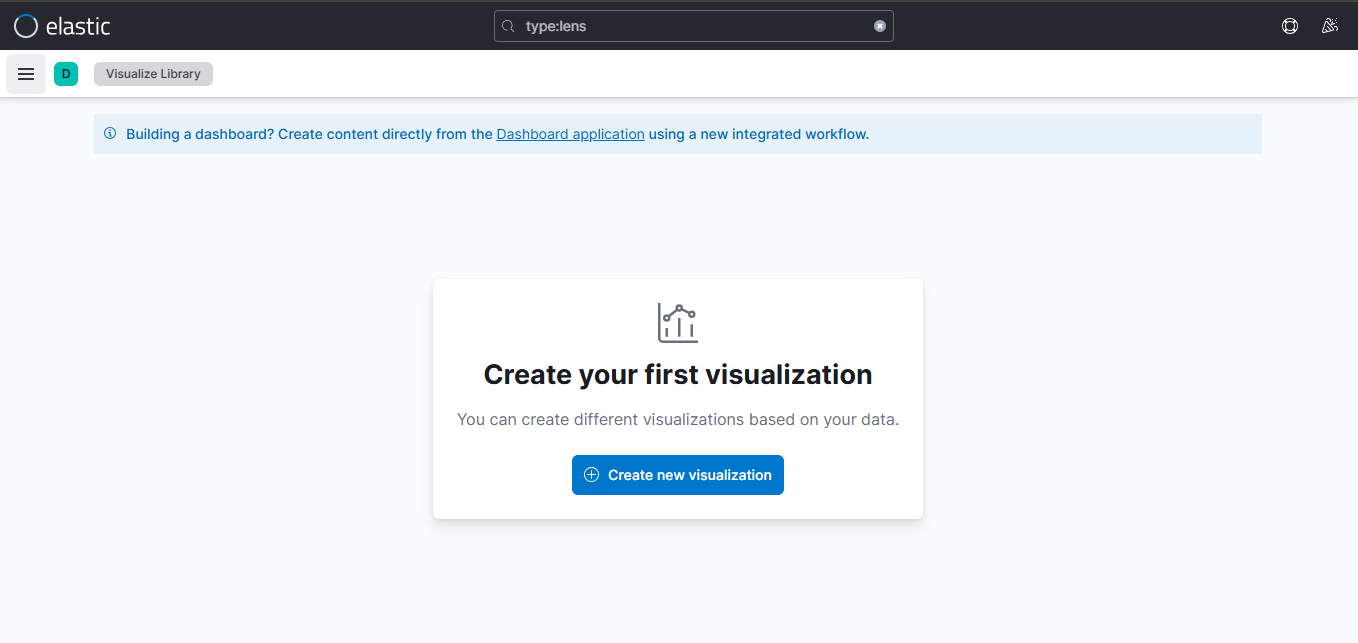
Con Elasticsearch y Kibana podemos crear nuestros propios **Dashboards** utilizando Elascticsearch y Kibana, por lo que primero que tendremos que hacer es crear un índice, podemos buscarlo en la barra de búsqueda de Elastic escribiendo **“Index patterns”,** como vemos que hay información, procedemos a hacer click en **“Create index pattern”**

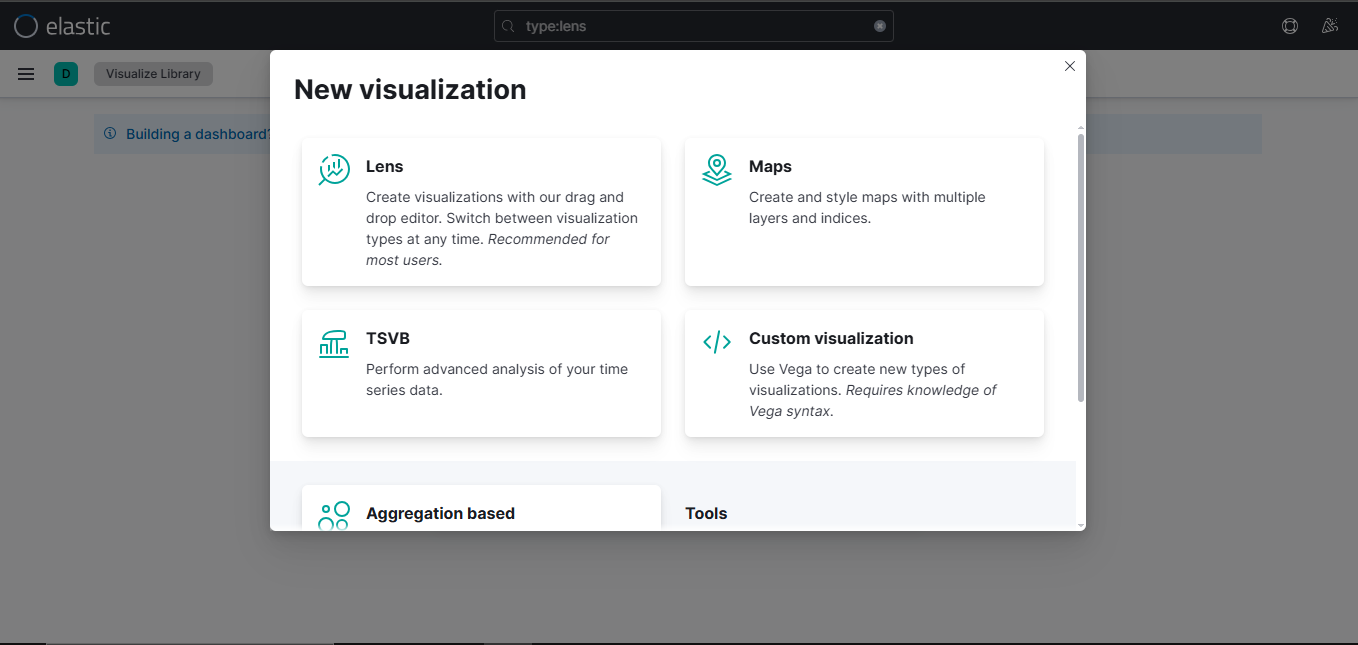


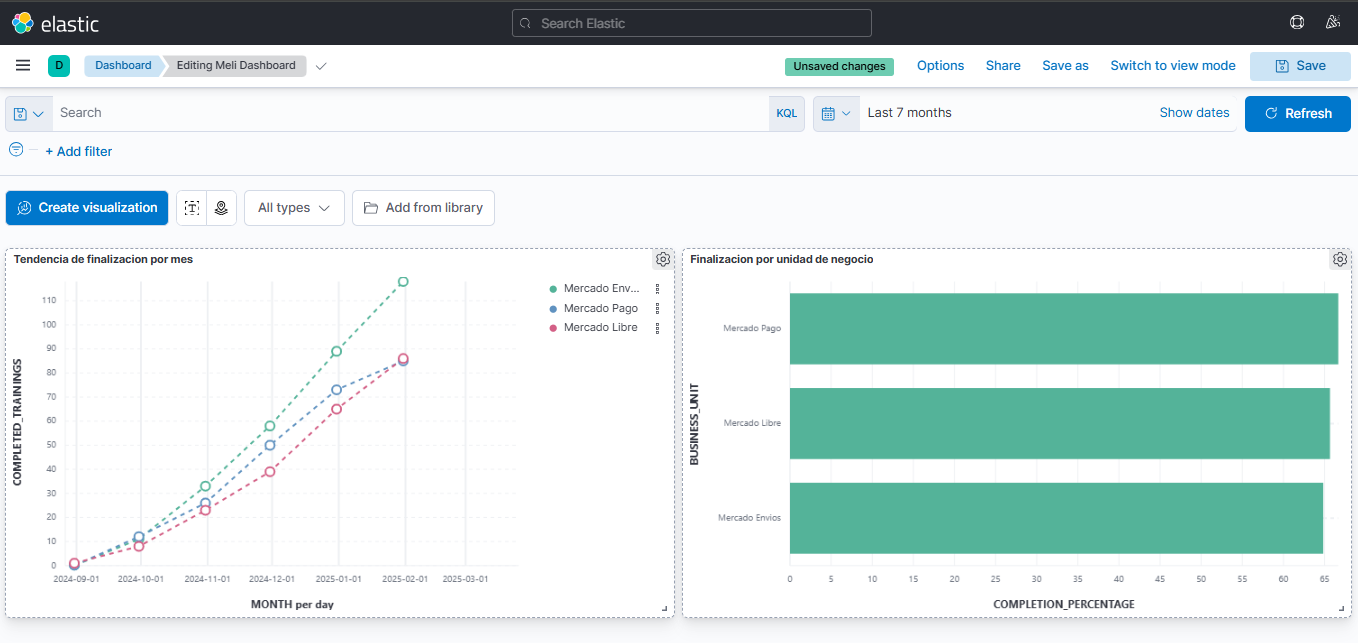
Estos son unos ejemplos de los Indices que creamos, podemos crear 1 por cada tabla de nuestra Base de Datos, 1 indice que contemple todas las tablas, etc.



En Visualize Library podemos comenzar a crear nuestras visualizaciones de nuestros datos utilizando todas las herramientas que nos aporta Kibana

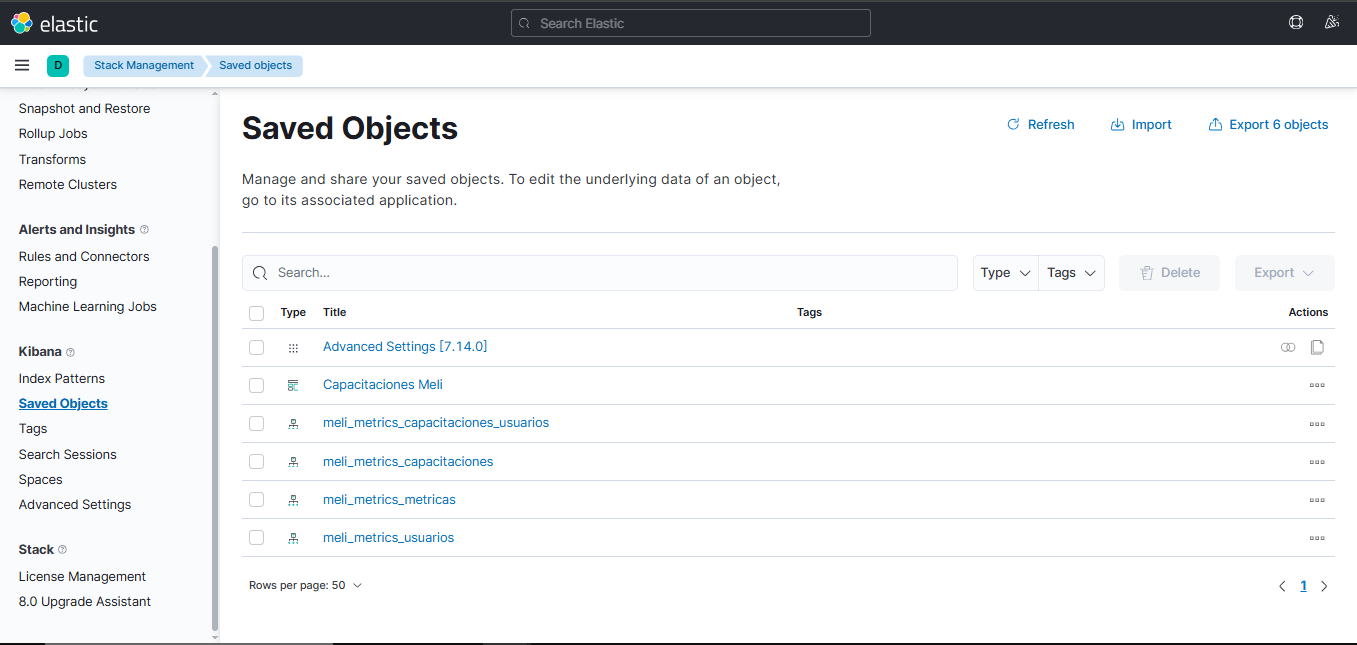




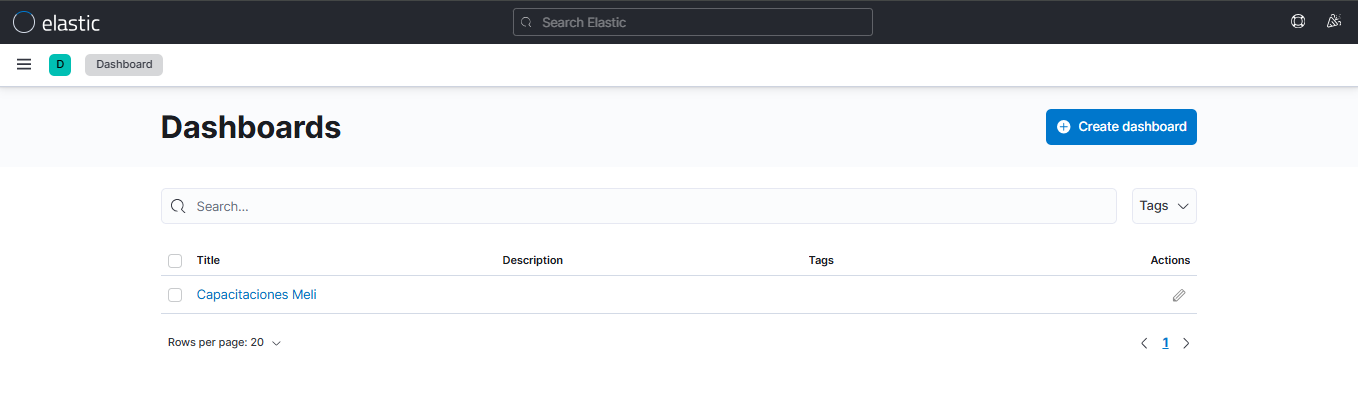


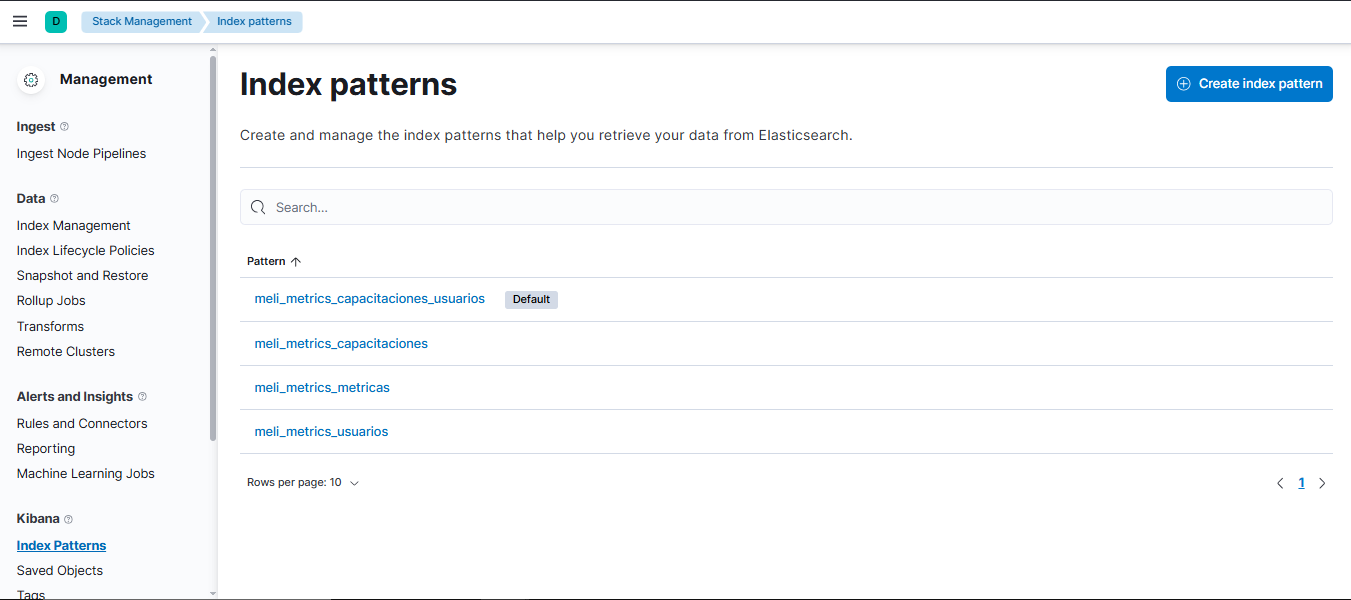
**Recursos de Kibana**

En el Github se encuentra un dashboard e índices ya creados para que estos puedan ser utilizados de una manera más sencilla y rápida, los mismos deben ser importados a partir de **Management > Stack Management > Saved Objects**. Una vez aquí hacemos click donde dice **Import**  e importamos estos recursos.



Luego al Dashboard lo vemos simplemente yendo a **Analytics > Dashboards;** y a los **Index Patterns** en **Management > Index Patterns** en la sección de Kibana





El Dashboard tiene diferentes gráficos y contadores que corresponden a:

* Porcentaje de Completitud Global: Un medidor que muestre el porcentaje promedio de completitud de todas las capacitaciones.
* Total de Usuarios Activos: Contador de usuarios sin fecha de finalización.
* Total de Capacitaciones Disponibles: Contador del número total de capacitaciones.
* Porcentaje de completitud por unidad de negocio.
* Evolución del porcentaje de completitud a lo largo del tiempo por unidad de negocio.
* Usuarios con más capacitaciones asignadas/completadas.

## Fuentes

Docker: <https://www.redhat.com/es/topics/containers/what-is-docker>

Ubuntu: <https://es.wikipedia.org/wiki/Ubuntu>

Python: <https://aws.amazon.com/es/what-is/python/#:~:text=Python%20es%20un%20lenguaje%20de,ejecutar%20en%20muchas%20plataformas%20diferentes>.

Mysql: <https://www.oracle.com/ar/mysql/what-is-mysql/>

Kibana: <https://www.dgtls.com/es/paginas-de-destino/kibana#:~:text=La%20plataforma%20de%20c%C3%B3digo%20abierto,el%20motor%20de%20b%C3%BAsqueda%20Elasticsearch>.

Elasticsearch: <https://www.elastic.co/es/elasticsearch>