



# **Practicum: Análisis y Gobierno de Datos**

## **ENSANUT 2018**

**Julio Cesar Velázquez Corona**

Universidad Anáhuac Campus Norte

Enero - Mayo 2026



*Asesora: Dra. María del Carmen Villar Patiño*

# Índice general

<b>Capítulo 1 Preparación</b>	<b>1</b>
1.1 Requisitos de Software . . . . .	1
1.1.1 Instalación de Docker . . . . .	1
1.1.2 Instalación de Ubuntu 20.04.6 LTS (Solo para Windows) . . . . .	1
1.1.3 Instalación de OpenMetadata . . . . .	2
1.2 Desarrollo Previo de Automatización de Creación y Llenado de Tablas . . . . .	4
1.2.1 Lectura de un Diccionarios de Datos . . . . .	4
1.2.2 Automatización de Lectura de Diccionarios de Datos . . . . .	7
1.2.3 Creación de Tablas en PostgreSQL a partir de Archivos JSON . . . . .	8
1.3 Gobernanza de Datos con OpenMetadata . . . . .	9
1.3.1 Glosario . . . . .	9
1.3.2 Clasificación . . . . .	10
1.3.3 Métricas . . . . .	12
1.4 OpenMetadata y otros Softwares . . . . .	13
<b>Glosario</b>	<b>14</b>
<b>Bibliografía</b>	<b>15</b>

# Capítulo 1 Preparación

## 1.1 Requisitos de Software

Un proyecto de ciencia de datos requiere diferentes tipos de software para llevarse a cabo y en el caso de nuestro proyecto, es necesario contar con los siguientes:

- Docker: es una plataforma para empaquetar y ejecutar aplicaciones en contenedores que llevan consigo las dependencias para que corran igual en distintas máquinas.
- Ubuntu 20.04.6 LTS: es un entorno Linux dentro de Windows que corre dentro de WSL (Windows Subsystem for Linux). Es básicamente el backend de Docker.
- OpenMetadata: es una plataforma unificada para el descubrimiento, la observabilidad y la gobernanza, impulsada por un repositorio central de metadatos, lineaje detallado y una colaboración fluida del equipo.

### 1.1.1 Instalación de Docker

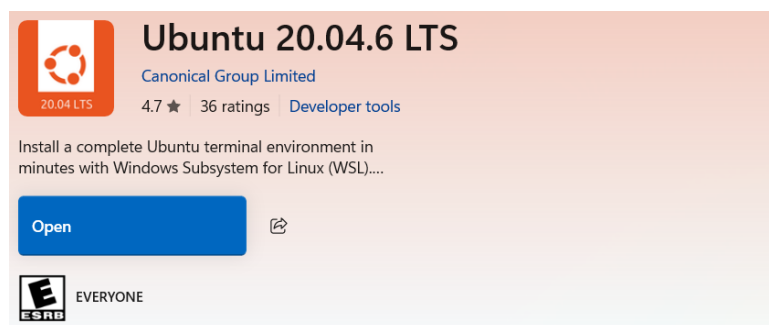
Accedemos a la [página de Docker](#) y damos click en Download Docker Desktop, luego aparecerá la opción de elegir el instalador que queremos dependiendo de nuestro SO. Ya con el instalador descargado, lo ejecutamos y seguimos los pasos que nos muestra. Cuando se indique, asegurarse de utilizar WLS 2 en lugar de Hyper-V. Por último, es necesario asignar como mínimo 6 GiB de memoria y 4 vCPUs. Para asegurarnos de que todo se instaló bien podemos correr en Powershell los siguientes comandos:

```
> docker --version
```

```
> docker compose version
```

### 1.1.2 Instalación de Ubuntu 20.04.6 LTS (Solo para Windows)

Puede descargarse directamente desde la Microsoft Store.



Ya instalado lo corremos y pedirá crear un usuario y una contraseña, luego entramos a Ubuntu e instalamos lo siguiente:

```
> sudo apt update
```

```
> sudo apt install python3-pip python3-venv
```

### 1.1.3 Instalación de OpenMetadata

En nuestro entorno Ubuntu en el caso de Windows o en la Docker CLI en el caso de Mac, colocar los siguientes comandos:

```
> mkdir openmetadata-docker && cd openmetadata-docker
> curl -sL -o docker-compose-postgres.yml https://github.com/open-metadata/
OpenMetadata/releases/download/1.11.3-release/docker-compose-postgres.yml
```

En el caso de Windows, entrar a Docker Desktop e ir a Settings ¿ Resources ¿ WSL integration y asegurarse de que todo esté activado de la siguiente manera:

#### Resources

Advanced File sharing Proxies Network WSL integration

Configure which WSL 2 distros you want to access Docker from.

☒ Enable integration with my default WSL distro

Enable integration with additional distros:

☒ Ubuntu-20.04

Refetch distros

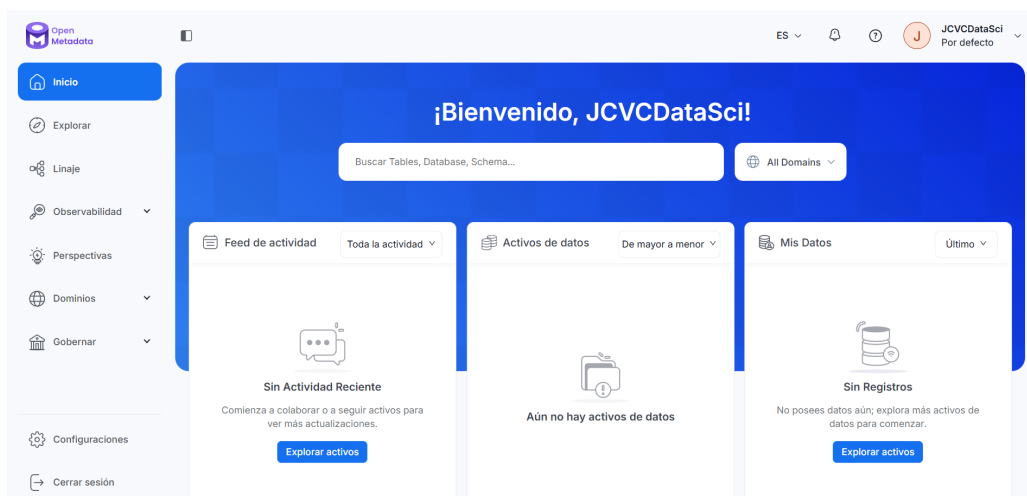
Colocamos el siguiente comando y al correrlo, reiniciamos la terminal:

```
> sudo usermod -aG docker $USER
```

Al volver a abrir la terminal corremos lo siguiente:

```
> docker compose -f docker-compose-postgres.yml up --detach
```

Al correr lo anterior se levanta el servicio de Docker Compose para OpenMetadata y podemos acceder al servicio desde <http://localhost:8585>. Ya dentro se nos pedirá crear una cuenta y ya podemos usar OpenMetadata.



Si posteriormente queremos volver a utilizar OpenMetadata hay que tener corriendo Docker Desktop en el fondo y correr los comandos que se presentarán a continuación:

```
> cd openmetadata-docker/  
> docker compose -f docker-compose-postgres.yml down  
> docker compose -f docker-compose-postgres.yml up -d
```

Finalmente acceder nuevamente a <http://localhost:8585>.








## 1.2 Desarrollo Previo de Automatización de Creación y Llenado de Tablas

El proceso que se debe llevar a cabo es el siguiente: **Automatizar la creación y llenado de las tablas, a una base de datos PostgreSQL y Oracle**. Se va a plantear cómo se tendría que realizar este proceso y de ahí se creará un diagrama de flujo para facilitar la reproducibilidad.






Nuestros datos se dividen en dos carpetas:

 nutricion	26/12/2025 11:28 a. m.	Carpeta de archivos
 salud	26/12/2025 11:29 a. m.	Carpeta de archivos

Cada una de esas carpetas tiene carpetas raíz con la siguiente estructura:

 conjunto_de_datos_cn_alimentos_adu_en...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_alimentos_com_en...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_alimentos_esc_ens...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_alimentos_prees_e...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_antropometria_en...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_cat_alimentos_ens...	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos_cn_des_inf_ensanut_2...	26/12/2025 11:28 a. m.	Carpeta de archivos

Cada carpeta raíz contiene lo siguiente:

 catalogos	26/12/2025 11:28 a. m.	Carpeta de archivos
 conjunto_de_datos	26/12/2025 11:28 a. m.	Carpeta de archivos
 diccionario_de_datos	26/12/2025 11:28 a. m.	Carpeta de archivos
 metadatos	26/12/2025 11:28 a. m.	Carpeta de archivos
 modelo_entidad_relacion	26/12/2025 11:28 a. m.	Carpeta de archivos

La carpeta `diccionario_de_datos` contiene un archivo CSV con una descripción de cómo está estructurada la tabla que tiene los datos, la idea es que desde ahí se empiece la automatización de creación de tablas y por lo tanto nuestro primer paso es leer los diccionarios de datos.

### 1.2.1 Lectura de un Diccionarios de Datos

La información leída debe de almacenarse y asignarse para cada carpeta raíz, mi idea para eso es que se cree un programa que busque, almacene y mande esa información estructurada correctamente a un archivo JSON. De esa forma, las tablas que queremos crear automáticamente podrán usar ese archivo como referencia. Para convertir de un CSV a JSON con Python, encontré tres opciones:

- Usar los módulos csv y json: es la forma más básica, estandar y es útil para procesos pequeños y datasets pequeños.
- Usar pandas: esta librería de manipulación de datos es un buen candidato.
- Usar jsonlines: esta librería escribe JSON línea por línea y es bastante útil para datasets grandes.

Las opciones que tengo como tentativa a usar son los módulos o jsonlines. Cabe destacar que la ruta de donde se obtendrán los datos será la que se utiliza localmente en mi computadora, sin embargo, en la documentación se elegirá una ruta genérica. A continuación se presenta un ejemplo básico con jsonlines de lo que se busca hacer:

```
# Extraccion del primer diccionario de datos

import csv
import jsonlines
from pathlib import Path
import re

# Path del primer CSV de salud
csv_path = 'C:\\Personal\\practicum\\rawdata\\salud\\conjunto_de_datos_cs_ensanut_2018_csv\\conjunto_de_datos_cs_act_fis_ado_ensanut_2018\\diccionario_de_datos\\diccionario_datos_cs_act_fis_ado_ensanut_2018.csv'

# Path para que se guarde en la carpeta actual
out_path = out_path = Path(__file__).resolve().parent / 'output.jsonl'

# Nombre de la tabla
m = re.search(r'diccionario_datos_(.+?)_ensanut_2018', csv_path)
nombre = m.group(1) if m else None

# Generamos JSON
with open(csv_path, mode='r', newline='', encoding='utf-8-sig') as csvfile, \
    jsonlines.open(out_path, mode='w') as writer:
    for row in csv.DictReader(csvfile):
        writer.write(row)
```

Este programa funciona, se guarda un JSON con toda la información del diccionario de datos y tengo planeado que para tener más orden y facilitar una automatización, se cree un JSON para cada tabla, por lo tanto, **nos conviene utilizar los módulos en lugar de jsonlines.**

La estructura que se manejó en el JSON es demasiado plana y no conviene para que pueda ser utilizada por PostgreSQL u Oracle, se tiene que corregir eso. Vamos a enfocarnos por el momento en PostgreSQL, en ese software existe una función llamada JSON\_TABLE la cual realiza una query en un JSON para poder crear una tabla a partir de esos datos. Se debe ajustar la estructura del JSON para que la función pueda trabajar bien con los datos.

```
# Extraccion de diccionario de datos con el primer conjunto de datos de la carpeta salud

import csv
import json
```

```

import re
from pathlib import Path

# Ruta donde se ubica el archivo .csv
csv_path = 'C:\\Personal\\practicum\\rawdata\\salud\\conjunto_de_datos_cs_ensanut_2018_csv\\conjunto_de_datos_cs_act_fis_ado_ensanut_2018\\diccionario_de_datos\\diccionario_datos_cs_act_fis_ado_ensanut_2018.csv'

# Extraccion del nombre de la tabla
m = re.search(r'diccionario_datos_(.+?)_ensanut_2018', csv_path)
nombre = m.group(1) if m else 'tabla'

# Para que el JSON se guarde en la misma carpeta donde esta este programa
out_path = Path(__file__).resolve().parent / f'{nombre}.json'

# Funcion anonima que convierte numeros de caracter a numeros int
to_int = lambda x: int(x) if (x := (x or '').strip()).isdigit() else None

# Funcion anonima que convierte los '' a null
to_none = lambda x: (x := (x or '').strip()) or None

# Lectura del CSV
with open(csv_path, 'r', encoding='utf-8-sig', newline='') as f:
    campos = []
    for row in csv.DictReader(f):
        # Quitamos un espacio extra en los encabezados si es que hay
        row = {k.strip(): v for k, v in row.items()}
        # Extraemos informacion con nuestras funciones anonimas
        campos.append({
            'nombre_campo': to_none(row.get('nombre_campo')),
            'longitud': to_int(row.get('longitud')),
            'tipo': to_none(row.get('tipo')),
            'nemonico': to_none(row.get('nemónico')) or to_none(row.get('nemonico')),
            'catalogo': to_none(row.get('catálogo')) or to_none(row.get('catalogo')),
            'rango_claves': to_none(row.get('rango_claves')),
        })

# Creamos un 'contenedor' tipo JSON donde se guardara la informacion
payload = {'nombre_tabla': nombre, 'campos': campos}

# Creamos JSON
with open(out_path, 'w', encoding='utf-8') as f:
    # Cuidamos por si hay caracteres mexicanos
    json.dump(payload, f, ensure_ascii=False, indent=2)

```

El programa funcionó bien y ya tenemos listo el algoritmo que se utilizará para obtener la estructura de las demás tablas.



### 1.2.2 Automatización de Lectura de Diccionarios de Datos

Lo que tenemos que hacer ahora es darle retoques al programa que hicimos anteriormente para que almacene la estructura de los demás diccionarios de datos de la carpeta llamada salud. Mi idea para hacerlo fue la siguiente:

- Obtener la ruta donde se encuentran nuestros datos.
- Crear nuestras funciones anónimas.
- Sacamos los directorios donde estan todos los diccionarios para después alimentarlo al programa.
- Extraemos los nombres otra vez para asignarlos a cada JSON respectivamente.
- Indicamos que queremos que la carpeta se cree en el directorio donde está el programa.
- Recorremos nuestra variable con los directorios, sacamos los CSV, los leemos.
- Creamos un payload y metemos la información ahí, todo dentro de un ciclo.

```
# Extracción de todos diccionario de datos de la carpeta salud
import csv
import json
import re
from pathlib import Path

# Ruta donde se ubican las carpetas raíz
root_path =
↳ Path('C:\\Personal\\practicum\\rawdata\\salud\\conjunto_de_datos_cs_ensanut_2018_csv')

# Función anónima que convierte números de caracter a números int
to_int = lambda x: int(x) if (x := (x or '').strip()).isdigit() else None

# Función anónima que convierte los '' a null
to_none = lambda x: (x := (x or '').strip()) or None

# Sacar directorios con un CSV que empiece con 'diccionario'
dirs_with_csv = sorted({f.parent for f in root_path.rglob('diccionario*.csv')})
for d in dirs_with_csv:
    print(d)

# Extraemos el nombre de la tabla desde el nombre del archivo
nombres = []
pat = re.compile(r'diccionario_datos_(.+?)_ensanut_2018')
for d in dirs_with_csv:
    f = sorted(d.glob('diccionario*.csv'))[0]
    m = pat.search(f.stem)
    nombres.append(m.group(1))

# Indicamos que queremos que la carpeta con los jsons se creen aqui
out_dir = Path(__file__).resolve().parent / 'jsons'
# Pero se crea si no existe la carpeta
out_dir.mkdir(exist_ok=True)

# Recorremos nuestra variable con los directorios
```

```

for d, nombre in zip(dirs_with_csv, nombres):
    # Guardamos los CSV por orden alfabetico para mantener orden
    in_csv = sorted(d.glob('diccionario*.csv'))[0]
    # Leemos cada CSV
    campos = []
    with open(in_csv, 'r', encoding='utf-8-sig', newline='') as f:
        for row in csv.DictReader(f):
            # Quitamos un espacio extra en los encabezados si es que hay
            row = {k.strip(): v for k, v in row.items()}
            # Extraemos información de cada CSV con nuestras funciones anónimas
            campos.append({
                'nombre_campo': to_none(row.get('nombre_campo')),
                'longitud': to_int(row.get('longitud')),
                'tipo': to_none(row.get('tipo')),
                'nemonico': to_none(row.get('nemónico')) or to_none(row.get('nemonico')),
                'catalogo': to_none(row.get('catálogo')) or to_none(row.get('catalogo')),
                'rango_claves': to_none(row.get('rango_claves')),
            })
    # Contenedor de como debe estar la estructura
    payload = {'nombre_tabla': nombre, 'campos': campos}
    # Ruta de salida del JSON y asignamos a cada uno el nombre indicado
    out_path = out_dir / f'{nombre}.json'
    # Creamos JSON
    with open(out_path, 'w', encoding='utf-8') as f:
        json.dump(payload, f, ensure_ascii=False, indent=2)
    print('JSON creado:', out_path)

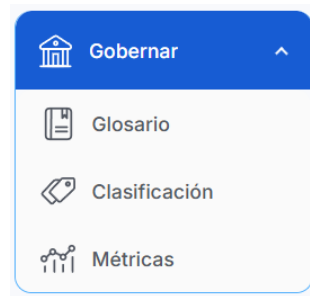
```

Ahora, nos queda hacer interacción con PostgreSQL.

### 1.2.3 Creación de Tablas en PostgreSQL a partir de Archivos JSON

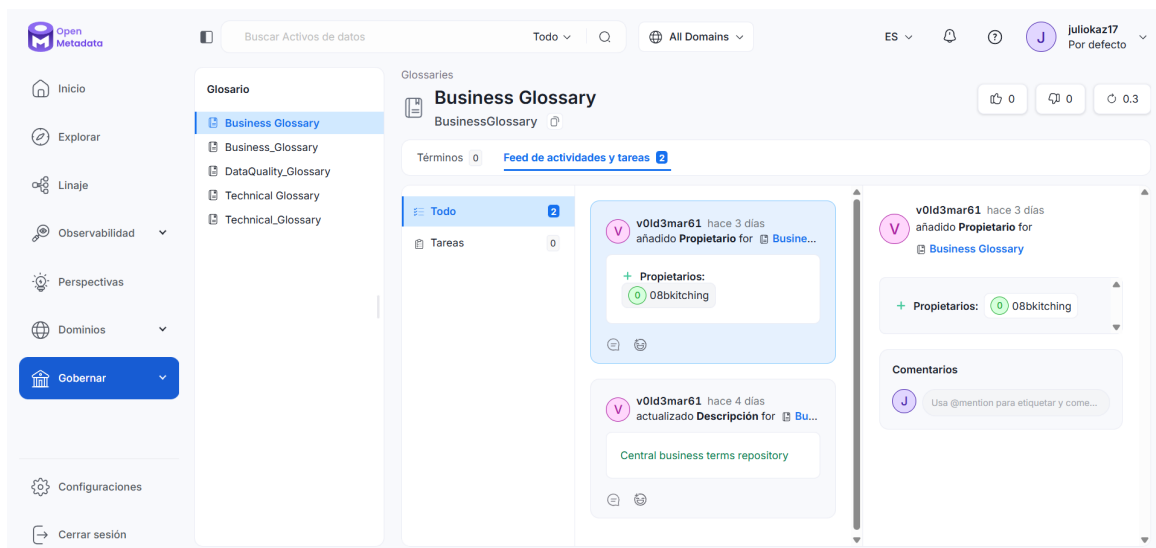
## 1.3 Gobernanza de Datos con OpenMetadata

El gobierno de datos es el conjunto de decisiones, reglas y prácticas con las que definimos y hacemos cumplir cómo se crean, se describen, se usan, se comparten, se protegen y se mantienen los datos dentro de una organización, para que sean confiables, seguros, consistentes y útiles para el negocio. En OpenMetadata tenemos tres artefactos que nos permiten llevar a cabo lo anterior y se encuentran en la sidebar del menú principal.



### 1.3.1 Glosario

En Glosario es donde se aloja el repositorio de los conceptos de los datos, eso quiere decir que ayuda a definir qué significa cada dato y también conectarlos con activos como tablas o columnas. En la sandbox nos encontramos con lo siguiente:



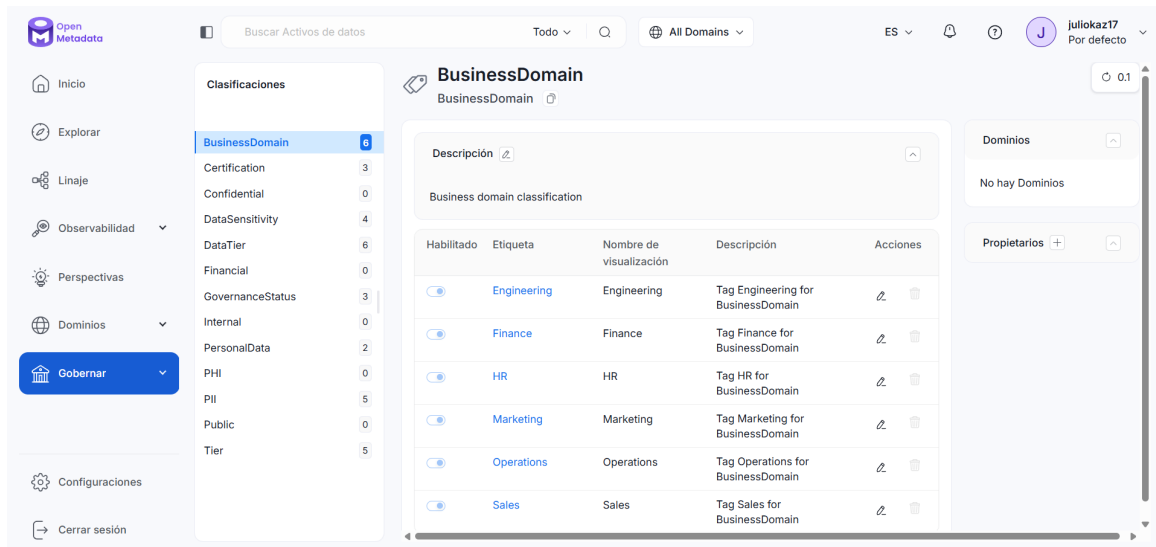
En el panel izquierdo podemos ver que hay diferentes glosarios y en este caso tenemos seleccionado el Business Glossary. A la derecha aparece nuestro título y vemos que en la parte de abajo dice Términos 0, eso quiere decir que aún no tenemos definiciones como tal de los datos. En la otra pestaña llamada Feed de actividades y tareas hay dos eventos registrados:

- El usuario v0ld3mar61 añadió "Propietario" a nuestro glosario y el propietario asignado es 08bkitching.
- Ese mismo usuario actualizó la descripción del glosario y la dejó como: *Central business terms repository*.

En la parte de la derecha aparece el cambio seleccionado con una sección de comentarios por si se quiere hablar del cambio.

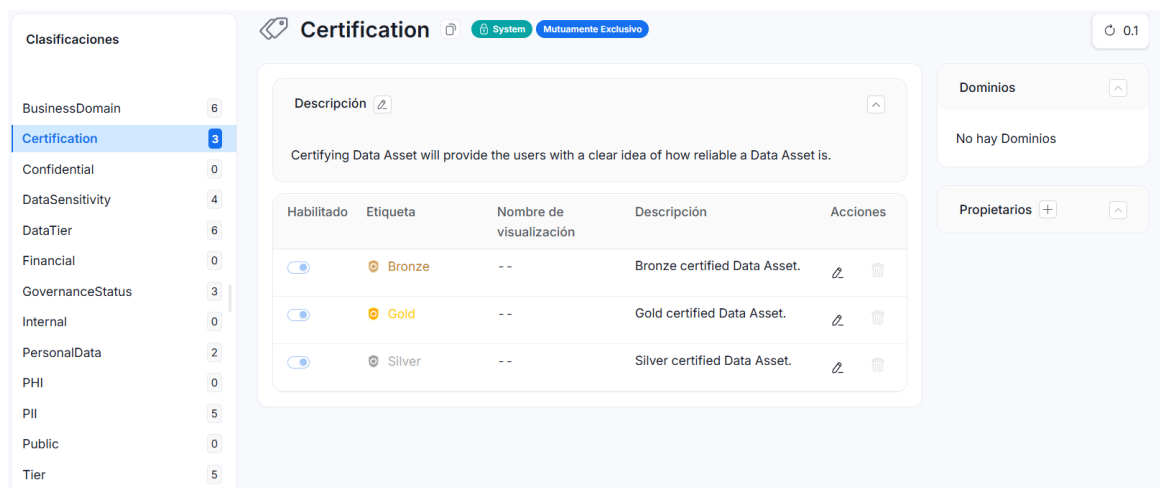
### 1.3.2 Clasificación

El siguiente artefacto es **Clasificación** y sirve para definir y administrar los datos que tenemos. No definimos el significado como tal, más bien se define qué tipo de dato es, cómo debe tratarse, si es PII o no PII, etc. La sandbox lo muestra de la siguiente forma:



En el sidebar debajo de Clasificaciones es donde se encuentra lo importante: son las etiquetas en las que pueden entrar nuestros datos. Algunas son creadas por default en el sistema y otras son creadas por usuarios. Para nuestro interes vale la pena describir las etiquetas creadas por el sistema:

- Certification: aquí se nos muestra si un dato es confiable o no.



- PersonalData: aquí se marcan datos que son personales.

**Clasificaciones**

- BusinessDomain: 6
- Certification: 3
- Confidential: 0
- DataSensitivity: 4
- DataTier: 6
- Financial: 0
- GovernanceStatus: 3
- Internal: 0
- PersonalData: 2**
- PHI: 0
- PII: 5
- Public: 0
- Tier: 5

**PersonalData** System Mutuamente Exclusivo

Descripción: Tags related classifying **Personal data** as defined by **GDPR**.  
 \_Note to Legal - This tag category is provided as a starting point. Please review and update the tags based on your company policy. Also, add a reference to your GDPR policy document in this description.\_

Habilitado	Etiqueta	Nombre de visualización	Descripción	Acciones
<input checked="" type="checkbox"/>	Personal	--	Data that can be used to directly or indirectly ... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	SpecialCategory	--	GDPR special category data is personal ... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>

**Dominios**: No hay Dominios

**Propietarios**: + -

- PII: aquí se etiqueta información personal que es capaz de identificar a una persona.

**Clasificaciones**

- BusinessDomain: 6
- Certification: 3
- Confidential: 0
- DataSensitivity: 4
- DataTier: 6
- Financial: 0
- GovernanceStatus: 3
- Internal: 0
- PersonalData: 2
- PHI: 0
- PII: 5**
- Public: 0
- Tier: 5

**PII** System Mutuamente Exclusivo

Descripción: Personally identifiable Information that, when used alone or with other relevant data, can identify an individual.  
 \_Note to Legal - This tag category is provided as a starting point. Please review and update the tags based on your company policy. Also, add a reference to your PII policy document in this description.\_

Habilitado	Etiqueta	Nombre de visualización	Descripción	Acciones
<input checked="" type="checkbox"/>	Email	Email	Customer email address	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	Name	Name	Customer name	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	None	--	Non PII	<a href="#">🔗</a> <a href="#">🗑️</a>

**Dominios**: No hay Dominios

**Propietarios**: + -

- Tier: en esta etiqueta se clasifican nuestros activos de datos en capas que van del Tier1 al Tier5 y mientras más alta es la capa, más importante es el dato.

**Clasificaciones**

- BusinessDomain: 6
- Certification: 3
- Confidential: 0
- DataSensitivity: 4
- DataTier: 6
- Financial: 0
- GovernanceStatus: 3
- Internal: 0
- PersonalData: 2
- PHI: 0
- PII: 5
- Public: 0
- Tier: 5**

**Tier** System Mutuamente Exclusivo

Descripción: Tags related to tiering of the data. Tiers capture the business importance of data. When a data asset is tagged with Tier tag, all the upstream data assets used for producing it will also be labeled with the same tag. This will help upstream data asset owners to understand the critical purposes their data is being used.

Habilitado	Etiqueta	Nombre de visualización	Descripción	Acciones
<input checked="" type="checkbox"/>	Tier1	--	Critical Source of Truth business data assets of ... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	Tier2	--	Important business datasets for your ... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	Tier3	--	Department/group level datasets that are typical... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>
<input checked="" type="checkbox"/>	Tier4	--	Team level datasets that are typically non-... <a href="#">Ver más</a>	<a href="#">🔗</a> <a href="#">🗑️</a>

**Dominios**: No hay Dominios

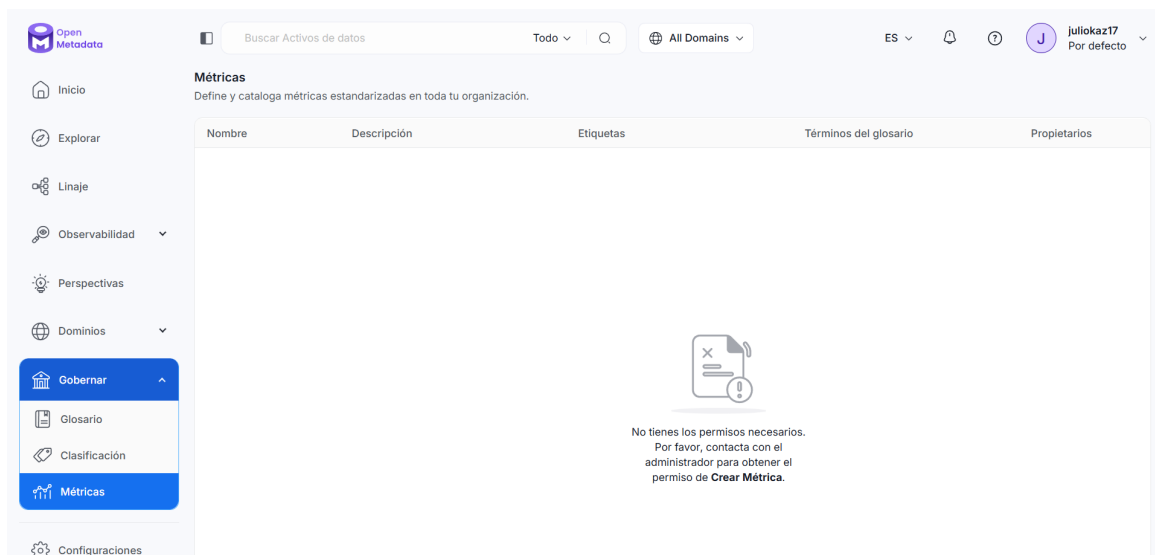
**Propietarios**: + -

### 1.3.3 Métricas

Por último, en métricas es donde se define explícitamente con qué lógica opera una métrica definida. Por ejemplo: si tenemos un KPI llamado `monthly_arpu` (ingreso promedio por usuario al mes), podemos definir lo siguiente:

- Nombre o ID.
- Descripción.
- Tipo o unidad.
- Granularidad.
- Responsable.
- Glosarios vinculados.
- Tags.

En la sandbox desafortunadamente no podemos observar cómo es que se utiliza este artefacto porque no hay métricas definidas y aparte aparezo registrado como un usuario que no cuenta con los permisos necesarios para crear una.



## 1.4 OpenMetadata y otros Softwares

Hasta ahora, hemos descrito bastante OpenMetadata, sin embargo, es de nuestro interés estar conscientes de que existe una competencia y que OpenMetadata no es la única opción. Los softwares más relevantes para lo que queremos son: Amazon Datazone, Oracle Cloud Infrastructure Data Catalog, DataKitchen y Alteryx. A continuación se mostrará un cuadro comparativo de interés:

Software	OpenMetadata	Amazon Datazone	Oracle Cloud Infrastructure	DataKitchen	Alteryx
Precio	Gratis	Pay As You Go	Pay As You Go	USD \$150,000 al año	Pay As You Go
Acceso a PostgreSQL DB	Sí	No	Sí	Sí	Sí
Acceso a Oracle DB	Sí	No	Sí	Sí	Sí
Artefactos de Gobierno	Glosario Clasificación Métricas	Management console Blueprints	Data catalog Harvest jobs Business glosary Data lineage Tagging	DataOps automation software	Roles Glosario Data lineage

# Glosario

- AWS IAM** Es un servicio que permite controlar de forma segura quién tiene acceso a qué recursos de AWS.
- Data Lineage** Es la trazabilidad de un dato; consiste en ver de dónde vino, qué transformaciones sufrió y de qué pipeline provino.
- DDL** Son las sentencias SQL para definir la estructura de la base de datos. O sea, crear, modificar y borrar objetos.
- FKs** Las llaves foráneas son restricciones que hacen que un campo en una tabla apunte a una fila existente en otra tabla, para mantener integridad referencial.
- Harvesting** Es el proceso de extraer, traer y actualizar metadatos desde las fuentes de datos hacia un catálogo o herramienta de gobierno.
- JSON** Es un tipo de archivo que utiliza la JavaScript Object Notation para almacenar información.
- Metadato** Información que describe a un dato para que se pueda entender.



# Bibliografía

- [1] Alteryx. *Connect to an Oracle DB*. <https://help.alteryx.com/current/en/designer/data-sources/oracle/connect-to-an-oracle-database.html>, 2026.
- [2] Alteryx. *Glossary*. <https://help.alteryx.com/current/en/connect/use-connect/understand-assets/glossary.html>, 2026.
- [3] Alteryx. *Maintain Data Lineage*. <https://help.alteryx.com/current/en/connect/administer-connect/maintain-data-lineage.html>, 2026.
- [4] Alteryx. *PostgreSQL*. <https://help.alteryx.com/current/en/designer/data-sources/postgresql.html>, 2026.
- [5] Alteryx. *User Roles and Permissions*. <https://help.alteryx.com/current/en/server/administer-alteryx-server/user-and-group-management/user-roles-and-permissions.html>, 2026.
- [6] Oxycon Blog. *Leveraging JSON in PostgreSQL 17*. <https://blog.oxyconit.com/leveraging-json-in-postgresql-17>, 2026.
- [7] crunchydata. *Convert JSON into Columns and Rows*. [https://www.crunchydata.com/blog/easily-convert-json-into-columns-and-rows-with/json\\_table](https://www.crunchydata.com/blog/easily-convert-json-into-columns-and-rows-with/json_table), 2026.
- [8] DataKitchen. *Automate Data Governance*. <https://datakitchen.io/solutions/datagovops>, 2026.
- [9] Amazon DataZone. *Authorization*. <https://docs.aws.amazon.com/datazone/latest/userguide/security-authorization.htm>, 2026.
- [10] Amazon DataZone. *AWS Glue Data Catalog*. <https://docs.aws.amazon.com/datazone/latest/userguide/create-glue-data-source.html>, 2026.
- [11] Amazon DataZone. *Built-in blueprints*. <https://docs.aws.amazon.com/datazone/latest/userguide/working-with-blueprints.html>, 2026.
- [12] Amazon DataZone. *Manage inventory and curate assets*. <https://docs.aws.amazon.com/datazone/latest/userguide/update-metadata.html>, 2026.
- [13] GeeksforGeeks. *Convert CSV to JSON*. <https://www.geeksforgeeks.org/python/convert-csv-to-json-using-python>, 2026.
- [14] Oracle Cloud Infrastructure. *Data Asset Properties*. <https://docs.oracle.com/en-us/iaas/Content/data-catalog/using/create-da-properties.htm>, 2026.
- [15] Oracle Cloud Infrastructure. *Data Lineage Overview*. <https://docs.oracle.com/en-us/iaas/Content/data-catalog/using/view-data-lineage-dis.htm>, 2026.
- [16] Oracle Cloud Infrastructure. *Managing a Business Glossary*. <https://docs.oracle.com/en-us/iaas/Content/data-catalog/using/manage-glossary.htm>, 2026.
- [17] Oracle Cloud Infrastructure. *Managing Jobs and Schedules*. <https://docs.oracle.com/en-us/iaas/Content/data-catalog/using/manage-jobs.htm>, 2026.
- [18] Oracle Cloud Infrastructure. *Overview of Data Catalog*. <https://docs.oracle.com/en-us/iaas/Content/data-catalog/using/overview.htm>, 2026.
- [19] Oracle Cloud Infrastructure. *Overview of Tagging*. <https://docs.oracle.com/en-us/iaas/Content/Tagging/Concepts/tagging-overview.htm>, 2026.

- [20] OpenMetadata. *OpenMetadata Documentation*. <https://docs.open-metadata.org/latest>, 2025.
- [21] C. V. Patiño. *Apuntes de Machine Learning*. Universidad Anáhuac México Campus Norte, 2025.
- [22] PostgreSQL. *JSON Functions and Operators*. <https://www.postgresql.org/docs/current/functions-json.html>, 2026.
- [23] PostgreSQL. *Populating a Database*. <https://www.postgresql.org/docs/current/populate.html>, 2025.
- [24] Python. *pathlib, Object-oriented filesystem paths*. <https://docs.python.org/3/library/pathlib.html>, 2026.