Los algoritmos de ML se remontan a los años 70, pero no había la cantidad de datos ni la capacidad de computación para desarrollar una IA hasta en los últimos años.

Elegir enfocarse en Data Scientist o Data Engineer.

Hay cosas como en acciones (stop loss) donde se procesan los datos en streaming, al momento, y esa información solo es valiosa en ese instante, por lo que no se almacenan los datos.

Crecimiento Orgánico y Big Data

Con Big Data: Ver a un conjunto de ordenadores como a uno solo (a lo cual se le llama ‘cluster’). A eso se le llama crecimiento o escalamiento horizontal (voy poniendo nuevos pequeños al lado de pequeño tamaño); lo contrario sería vertical (tiro el antiguo y compro otro más grande con mayor capacidad).

Ley de Moore, en relación a la velocidad de procesamiento computacional, la cual es exponencial: cada 18 meses aproximadamente se duplica.

Un ‘data warehouse’ es una base de datos que incluye bases de datos. Dependerá de la casuística será mejor esto o un ‘**data lake**’. Un data lake se apoya en una tecnología llamada HDFS, que son una serie de ficheros distribuidos donde se guarda información: un cluster que está formado por nodos. Cloudera y Hortonworks están aquí.

En el otro modelo (del ‘data warehouse’) están Oracle o SQL server.

Datos como un activo imprescindible de la empresa, no solo como un reflejo del estado de la empresa. Ser proactivos y predecir el futuro en vez de reactivos y analizar el pasado. Conseguir transformar los datos en información útil para la empresa. Datos es diferente a información, por tanto.

Captura y Recogida de Datos

**Procesos ETL**: extract, transform & load.

SFTP: Secure File Transfer Protocol. Un Cliente SFTP es FileZilla.

En las bases de datos relacionales existen el protocolo ODBC y JDBC: consulto de una base de datos externa y extraigo el dato o datos que quiera y me los llevo. Hago consultas SQL.

Para replicar una base de datos entera es CDC, es un agente que monitoriza activamente la base de datos y cada vez que cambia algo allí me lo notifica. Se sincroniza en tiempo real.

Colas o tópicos en cuanto a mensajería.

Batch junta, por lotes. Programo las horas del día en las que quiero que se ejecuten procesos, es una manera de automatizar. Tengo información actualizada desde la última vez que lancé el proceso. Me espero x tiempo para procesar todo lo que lleve hasta ese momento. Micro-batching es hacerlo cada minuto o cinco minutos, para hacerlo más rápido, casi streaming, pero no es automático. En el 90% de los casos serviría. Pero hay veces que necesito inmediatez (operaciones bursátiles), para eso están las streaming sources, que es conforme el dato llega, como puede ser Kafka.

Bases de Datos Relacionales

Para almacenar datos estructurados, organizados son las más comunes (SQL). Hay otras, son las NOSQL. Nosotros trabajaremos con MySQL entre las primeras. SQL es el lenguaje formal para acceder a esos datos de la base de datos.

Las entidades son como las cosas: coche, persona, mesa. Y luego los atributos son las características (rojo, alto, rota), o las columnas. Y las relaciones es entre las distintas entidades. La **cardinalidad** puede ser 0, 1 o N (varios).

Mejor tener entidades separadas, no padre-hijo heredadas ni agregación, en la práctica.

Mirar ‘Internet Sales Model ejemplo’ para aclararse.

https://github.com/echiner/edem-mda-relational-databases, para el Ejercicio 1, por ejemplo, para dibujar el Diagrama.

Las entidades se traducen en tablas, pero no todas las tablas serán entidades, hay veces que se necesitarán tablas intermedias. Not null es que no quieres que ese campo esté vacío. Default es para asignar un valor por defecto en caso de que no se rellene ese campo. Cuantas más restricciones (constraints) la base de datos es más consistente o sólida (integridad), pero también es más lenta. Una primary key será not null y unique por definición.

La integridad referencial es muy importante en una base de datos para por ejemplo corroborar que el cliente existe en la misma.

Para hacer Ejercicio 2 descargarse el GitHub a local y levantar el docker en el sitio correcto y seguir las instrucciones.

Open Data:

https://www.europeandataportal.eu/es

https://datos.gob.es

http://gobiernoabierto.valencia.es/es/

Son interesantes: World Bank y Gapminder.

Data Provenance es trazabilidad también.

El componente, el que ejercita la acción es el procesador, y el FlowFile es la unidad mínima de datos en **NiFi**.

Procesadores pueden ser fuentes de datos (sources) y destinos donde guardar los datos (sinks).

Ejercicio 1: con los procesadores getfile (leer, obtenerlo) y putfile. En propiedades indicar directorio correcto (/tmp/in o /tmp/out) y en putfile en settings pincharle a failure y success. Luego ya darle al play a los dos y ya funcionará cuando cree un archivo en in se irá directamente a out. Para crear la conexión es simplemente arrastrando. Por tanto, creamos un archivo y lo metemos en un sitio (get it and put it there, en out).

Ejercicio 2: Leer (llamas) de una web y guardas en out. Dejar el invokeHTTP solo response. Y en la conexión solo response también. Además, en invoke rellenar con la url desde donde se lean los datos (el CSV del Ayuntamiento, por ejemplo). En ‘run schedule’ pones cada cuánto tiempo quieres que te haga el proceso: cada 30 segs., 1 vez al día… (batch), en función de las necesidades del proyecto. El resto, igual que en el Ejercicio 1.

Ejercicio 3: Añades dos procesadores adicionales intermedios. Para renombrar darle a ‘+’ y poner filename y luego en texto X. Trocea el fichero en 1000 ficheritos (splits).

SQL

Escribes consultas o ‘queries’ desde un cliente; en **BigQuery** se puede, desde Big Data en Google Cloud. No se puede desde SQL (crear una instancia de PostgreSQL, por ejemplo), eso solo es una base de datos para almacenar datos, no es cliente para hacer queries.

SELECT: dame

INSERT: toma algo nuevo

UPDATE: actualiza algo ya existente

DELETE: borra

Las queries no son sensibles a mayúsculas en toda la parte de código. Nullable significa que no es not null, puedes no poner nada y no pasa nada. Si no hay propiedades se pueden repetir los datos, no pasa nada (no pone UNIQUE ni nada).

En las relaciones entre tablas se pone FK. “Una PK en una tabla suele ser lo que es una FK en una tabla que se relaciona con ella”. UNIQUE es un valor que no se puede repetir, pero no hace falta que sea PK. En BigQuery no hay constraints porque está pensado para análisis. Null no es que haya un 0 o un texto en blanco, sino que no hay nada, no hay información, es desconocido.

CREATE TABLE Alquileres (

ID serial PRIMARY KEY,

Mes VARCHAR (10) NOT NULL,

Cantidad VARCHAR (15) NOT NULL,

Descripción VARCHAR (1000) CHECK (Descripción > 250)

);

La tabla de la izquierda es la más grande. Solo se puede hacer joins entre tablas que tienen una relación entre sí.

Interesante en BigQuery ver dentro de una tabla > exportar > examinar con Data Studio.

Tenemos que hacer ejercicio de la última diapositiva en el editor o gestor de bases de datos que pone en el README de sql en edem2021.

ETLs

Extraemos lo que queremos, lo transformamos y cargamos los datos, en ese orden. La ‘staging area’ o ‘zona de landing’ es una fase previa, donde el dato llega casi sin modificar desde el origen, se guarda ahí para ahorrar recursos. En esa zona se hace la limpieza de datos, el enriquecimiento y la trazabilidad. Y luego ya mediante otra ETL se modifica de verdad y se cargan los datos. Las ETL están pensadas más para un proceso batch (cada hora saco los datos y me los llevo, normalmente el incremental). Tarda más al principio al meterlo, pero luego es más rápido cuando nos piden el dato porque ya está transformado (DWH). Funciona al contrario con las ELT (data lake) donde metemos todo tal cual y luego cuando nos piden ya transformamos; los tiempos se invierten.

Lo bueno es que en las ELT el cliente te pide algo y como no lo has transformado aun puedes hacerlo como él quiera; con las ETL ya está hecha la transformación y hay lo que hay, el cliente lo coge como esté.

Donde más problemas hay es en la extracción, por la diversidad. Para ello, **Talend** por ejemplo tiene más de 100 conectores para diferentes fuentes de datos. Y otro tanto de conectores de salida para diferentes herramientas de BI.

Fase de transformación: Conjunto de operaciones donde los datos de origen se encuentran con las reglas de negocio.

Carga/load: Rolling es lo que ya has calculado día a día, lo agrupas y tienes la semana. La acumulación simple es cogerlo todo de golpe y agruparlo. Para hacer rolling bien necesito que los datos estén en la granularidad más baja para luego poder agruparlos como quiera.

ETL (tengo un DWH y el sistema ya lo prepara de antes para que sea más fácil de leer) vs ingestión (tengo un data lake tengo todos los datos a nivel de segundo y si me pides por año cojo y los agrego, y así sucesivamente).

¡Cuadrante mágico de Gartner!

Con Talend en el primer ejercicio queremos que los datos vayan de un fichero de entrada (CSV) a otro de salida (JSON). Cada cosa que se coge de la derecha es una tarea. Y tenemos un job con dos tareas, una dice de dónde tiene que leer y la otra dónde tiene que escribir. El fichero json tiene el contenido del csv pero en json. Hay que leer desde metadata (file delimited) el csv, sino saldrá el json pero vacío; y el output sí se coge de la derecha (json).

En el segundo estamos haciendo otra transformación, del tipo ‘replace’. Si hay problemas con columnas darle a sync columns en la transformación; si en el que coge está bien hecho entonces se arreglará. Acordarse de poner cabecera-1 en el que coge (CSV) para que salgan los nombres de las columnas como queremos.

En el tercero hacemos un filtrado por filas, donde nos quedamos únicamente con aquellas donde el país sea Brasil (9 filas al final, de un total de 152 que había).

Y en el último hacemos un agregado, un ‘count’, cuántos alumnos hay por país. Y nos quedamos con solo dos columnas para el de salida: país y total. De 5 pasamos a 2.

Diferencia entre importar metadatos a un fichero y a una base de datos. Creamos conexión a la base de datos ahora y se da a extraer esquema.

https://www.udemy.com/course/aws-certified-developer-associate-step-by-step/?couponCode=LEARNAWS: **Curso Amazon AWS**.

*Nota*: docker ps -a ID\_contenedor es para parar por comandos un contenedor.

Almacenamiento de datos (Big Data) y bases de datos NOSQL

Antes era NoSQL (no), ahora es NOSQL (not only), pues muchos también incluyen.

Bases de datos de grafos: Te interesan las relaciones más que la información en sí (el caso de los papeles de Panamá).

En memoria: El acceso es más rápido (no está en disco), pero difícil de manejar (se utiliza como caché o en aplicaciones de procesado en tiempo real).

Las BBDD NOSQL más generalistas son MongoDB, Elasticsearch y Cassandra.

**Teorema de CAP**: se pueden tener 2/3.

* Consistencia: todo el mundo verá lo mismo.
* Disponibilidad: seguirá funcionando la BD pese a si hay un fallo en los nodos.
* Partitioning: seguirá funcionando la BD pese si hay un fallo de red.

… salvo Google, que permite nodos distribuidos por regiones sin perder la consistencia propia de las BBDD relacionales.

**Cassandra:**

Transacciones: cada vez que haces un commit es una transacción y NOSQL no permite.

CQL: Cassandra Query Language, parecido a SQL, y donde hay filas de columnas.

El KeySpace es una forma de agrupar las tablas. Se tiene que crear antes de crear cualquier tabla.

Normalmente se usan las Compound Primary Key. La Primary Key se compone de una *Partition Key* y de una *Clustering Key*. La primera agrupa por nodos dentro del cluster (asigna un nodo u otro en función de los valores para repartir dentro del cluster, en los nodos y particiones: cómo se van a distribuir los datos dentro del cluster); y la segunda ordena. Debe ser única en conjunto. Se define a nivel de tabla. En caso de que no haya clustering key (solo hay un campo) es la partition key. Si no defines claramente y hay más de dos entonces la primera es la PK y el resto CK.

Si se cae un nodo no quieres perder los datos, para eso está la replicación. De todo lo que metes en la BD se copia (n factor veces). Si la estrategia es simple (una sola BD) si se cae la BD entonces se fastidia, si se utilizan varias entonces es más complejo, pero más seguro ante esa posibilidad. Se define a nivel de keyspace.

Podría haber cierta inconsistencia en un momento dado, pero al milisegundo. Está pensado más para escribir que para leer. No está pensado para actualizar. Sí está pensado para cuando hay muchos datos.

Map: clave + valor; set: conjunto o lista de n elementos únicos; list: es un set, pero ordenado (es la diferencia).

En el ejercicio en Zeppelin de crear una tabla es mejor que group sea partition key.

Hay otro ejercicio que mezcla NiFi con Cassandra y Zeppelin.

Gobierno del Dato

Empresa **data-driven**: datos como asset, no como final. Primero hay que ver cómo estamos, y luego definir una estrategia, una visión, una planificación. Una estrategia de datos se suele hacer a 4 años, como planificación, en una empresa grande. Los Data Principles es asegurar la consistencia de los datos, el cuidado, evitar duplicidades, etc. Luego, alinearlo con las decisiones de negocio, con la estructura de negocio.

¿Cómo organizar el asalto a los datos? DEFINIR BIEN LOS ROLES

* Consumidor: puede ser la aplicación web. Las interfaces tienen que minimizar la lógica de negocio. No eres el dueño de los datos, un consumidor no modifica datos, solo consume. Hay otra persona responsable del contenido…
* Content Owner: el responsable de la información, de los datos, del contenido, de la definición, de los datos que faltan, etc. Puede haber uno por área.
* Platform Owner: dimensionar los datos, que sean accesibles en la plataforma. Es el jefe de la parte tecnológica: encargado del modelo físico, de la seguridad, etc.
* Data Steward: el soldado raso tecnológico. Encuentra los errores.

El modelo lógico cambiará según las necesidades de negocio; el físico irá por detrás (parte más IT) para respaldarlo y atenderlo desde abajo (base datos, entorno Big Data…). Puede haber varios modelos físicos, pero un solo modelo lógico. El modelo lógico tiene que ser comprendido por negocio; y el modelo conceptual es el modelo más abstracto o superior que hay en la organización (te quedas solo con los atributos más importantes y los nombres de las entidades).

Vistas: Tablas virtuales que desde fuera se ven como una tabla real. Ejemplo: CREATE TABLE (select… y haces la query). El modelo lógico trabaja con vistas (encapsula las queries que hay por debajo en una tabla). Esa vista es lectura, pero no existe en el modelo físico. Así, si cambio por debajo el modelo físico eso será invisible para negocio porque la vista seguirá igual, porque seguirá haciendo la misma query de las tablas del modelo físico que sí han cambiado.

Ejercicio table + view: VND es el vendedor, que apuntará en otra tabla a su PK (aquí es FK), aunque también podría ser un varchar (50), por ejemplo; timestamp es hasta el milisegundo. AMT no puede ser int.

CREATE TABLE SLS\_ACCT (

ID int PRIMARY KEY,

VND int FOREIGN KEY,

FVenta timestamp NOT NULL,

AMT double NOT NULL

);

Dato es la información (el valor de la celda) y el metadato te da más detalles (qué tipo de dato es, fecha, a qué tabla pertenece, formato archivo…).

**Metadata Lineage** (no se puede de datos): se sabe que D=A+B+C (metadatos), pero no se sabe que 4=1+1+2 (datos).

Hacer un glosario de términos de tu negocio, que tienen que ser validados y comunes para toda la empresa. Para que no haya malentendidos ni decisiones contradictorias. De esto se encarga el Content Owner.

En nuestro ejercicio en grupo puede haber un content owner por cada área: uno para personas (RRHH), otro para proyectos y tareas (Project Manager) y otro para pagos (finanzas). Si hay problema notifica el steward y avisa al content owner (que entiende de su área de negocio específica) y este habla con el platform owner para que modifique lo que sea a nivel IT si es necesario.

KAFKA: sistema de mensajería distribuido.

Topics son como tablas en bases de datos, sitios donde mandamos mensajes, los mensajes se envían y se leen a/de un topic en concreto. Ordenación global es orden perfecto.

Escalabilidad vertical es poner más capacidad, mayor memoria y tal a un ordenador; la horizontal es poner muchos ordenadores (la aplicación que desarrollamos correrá en distintas máquinas, las cuales deberán comunicarse entre sí, necesitan coordinarse). Necesita alta capacidad de computación (alto rendimiento). Desacoplamiento significa que hay esa separación mediante el sistema intermedio.

En Kafka se guarda todo muy ordenado para encontrar luego la información rápido. Por ejemplo, todos los que tengan una misma llave irán a la misma partición.

Scew data: balanceo entre los distintos ordenadores, para que no haya sobre/infra utilización de unos sobre otros. Para distribuir mejor (más equilibrado y homogéneo entre las distintas particiones) puede usarse ‘hash’, para que sea aleatorio y no haya diferencias ni sobre/infra utilización. Kafka hace esto solo automáticamente.

SPoF: single point of failure (es malo tener).

Avro es un “contrato” informático escrito en JSON para uniformizar todo y entenderse bien para que nos entendamos (poniendo condiciones, órdenes y límites). Serializar es transformar el mensaje en bites de 0 y 1. Y al que le llega lo deserializa para leer. Y avro ayuda a esto. Kafka no entiende que es un JSON ni nada, solo entiende de 0 y 1.

Un broker es un nodo, un servidor, una máquina.

En Ejercicio\_Lab2 hay que hacer cada uno de los tres en una consola diferente y todo desde dentro de la carpeta kafka-streams-example (donde está el proyecto). Y hacerlo por orden: primero productor, luego el procesador y luego el consumidor.

https://github.com/rusansor/edem2021

Cada broker es un PC, un servidor. Los mensajes se envían al topic en función de la materia, y cada topic se divide en particiones, que se guardan (replicadas) en diferentes brokers para tener mayor rapidez y por si se fastidia un servidor, que no pase nada. Dentro de cada topic están los índices o offset.

Paralelizar (más de un consumidor) para tener mayor rapidez, rendimiento. Hay que saber cuántos topics tengo y particiones para saber cómo los voy a leer (consumidores).

Borrar mensajes: no se puede en Kafka, solo a nivel administrativo, no 1 a 1:

* Por antigüedad: borro los más viejos.
* Por tamaño: cuando vaya a llegar a X tamaño borro los antiguos.
* Por compactación: para ahorrar espacio cuando se actualiza información, me quedo con la más nueva.

Se pueden hacer joins de diferentes topics. Zookeeper decide de dónde pueden leer cada uno de los consumidores o grupos de consumidores.

Confluent Kafka Tutorials: https://kafka-tutorials.confluent.io/

Apache SPARK

KAFKA Streams es la competencia. Es un framework de procesamiento distribuido (procesamiento entre distintos servidores de las distintas n divisiones de los datos: no hay límite, como sí lo había con la computación monolítica). “Divide y vencerás”.

Pero dividirlo tampoco sale gratis, sino que hay que juntarlo y pasar información por red (que es más lento) entre los servidores (minimizar ese tráfico por la red).

**HDFS** es clave en el procesamiento distribuido. Un sistema de ficheros distribuido que divide todo en bloques y lo distribuye entre distintos servidores. Pero además replica esos bloques, para que no se pierda el fichero si se fastidia un servidor (por defecto 3 veces). También se podría replicar el cluster entero con sus nodos (servidores) y enviar alguna de esas replicas ahí, para mayor seguridad (aunque sería más lento leer de ahí).

El Master (NameNode) tiene los metadatos, sabe dónde está cada cosa y sabe unirlo.

La base de la programación distribuida: llevarse la computación donde están los datos. Pongo cada uno de los nodos con su servidor en cada bloque de datos.

Hay que evitar el tráfico de red, pero a veces no te queda otra. Y para coordinarlo está **MapReduce** (para batch). Alta latencia: desde que empieza hasta que termina cuánto tarda (por eso no sirve para stream). Alto throughput (rapidez de mb/seg) porque tengo a muchos, es distribuido. Map: ejecuta una función (\*sueldo+100, p.e.) a cada uno de los elementos que nos interese, en todos los nodos, en paralelo, sobre cada uno de los registros (sumo 100 euros al sueldo de cada uno de los empleados). Y el nuevo fichero distribuido al que se le aplica la función determinada también se distribuirá en bloques y se replicará y todo, mediante HDFS (dos procesos distintos). Shuffle: para agrupar por nodos, para distribuirlo, por departamentos, por ejemplo, ello implica tráfico de red para llevar a cabo ese proceso. Reduce: agrega los resultados (ahora son 2 bloques, cuando antes entraron 3 “Input Split” o bloques, cada uno con un nodo).

No se puede modificar un fichero, se tiene que crear uno nuevo con la modificación, y luego si quieres borrar el antiguo.

Puedes encadenarlo con sucesivos maps, aplicarle otra función y obtener un nuevo output, y así sucesivamente. Map escribe en disco duro y lee y escribe y lee, en vez de guardártelo en memoria (RAM). Estas operaciones cuando se encadenan maps son muy lentas. Además de que había alta latencia. Así surge **Apache SPARK**.

Un cluster con muchos nodos irá más rápido; muchos clusters con pocos nodos irá más lento, pero será más seguro.

*Nota*: Hadoop es un ecosistema que engloba aplicaciones de Big Data en entornos distribuidos, e incluye: MapReduce, HDFS o Apache SPARK.

Trabaja en memoria RAM, en vez de escribir en disco.

Grafo acíclico dirigido: O (load) -> O (filter) -> O (map). El Driver lo construye y le dice a todos los executors en cada uno de sus nodos con la información que tengan que los ejecuten.

Por defecto, 1 tarea – 1 CPU – 1 bloque (cachito de datos).

**RDD**: Colección de datos distribuida e inmutable (solo lectura, no se modifica; si se modifica el resultado es otro nuevo RDD), el cual está particionado (es único, pero internamente está particionado). Cada tarea se ejecuta sobre una única partición. Una partición es la unidad mínima; una misma partición no puede estar partida en diferentes nodos. Normalmente es necesario tantas particiones como CPUs, pero no siempre: si hay problema de memoria puedo hacer más particiones, por ejemplo.

* Funciones de transformación: sobre un RDD1 te devuelve otro RDD2.
* Funciones de acción: sobre un RDD1 te devuelve un resultado.

Ejercicios:

Lo hacemos en Google Colab. Siempre habrá que ejecutar primero los prerrequisitos para tener el programa en la nube, en una VM. Y obviamente bajarse los datasets.

TextFile es un RDD que contiene internamente ese fichero de texto. Si hubiera varios nodos los ejecutaría en paralelo.

En el segundo, meto la colección de Python (array) en un RDD (distribuido) y aplico la función esa al RDD (no a la colección de Python).

La escritura en disco, en Kafka o en cualquier otro sitio es una acción también. Hasta que no se topa con la acción no ejecuta nada.

Lambda: le aplica una función o método a cada uno de los elementos del RDD (saco con elementos). En el reduceByKey hace un shuffling y distribuye por nodos: aplica una función de agregación por ‘key’.

**SPARK SQL**: lo realmente importante.

DataFrame=Dataset: colección de datos distribuida (RDD) + Esquema

Hive: es un lenguaje muy completo de SQL.

Compilar: pasar de código escrito a código máquina (0 y 1).

DataFrame: una colección de objetos ‘row’.

Los DataFrames son como los RDDs, son inmutables y no se pueden modificar.

ELK:

Se incluye dentro de toda la estructura en almacenamiento de datos (Elasticsearch), pero también en ingestión (Logstash) y visualización (Kibana).

En vez de buscar por el nombre, buscas por el contenido del documento (índice invertido), como Google. No como buscas como en tu ordenador un documento (eso sería índice normal). Buscas por el texto de dentro de los documentos, y te da los documentos que contienen esas palabras. Es un sistema Big Data, que funciona muy rápido la búsqueda, escalable, donde el almacenamiento es un poco costoso.

Es un framework de procesamiento distribuido, donde hay clusters de ordenadores (escalabilidad horizontal): no hay límite. Un nodo es una máquina, un cluster es una colección de máquinas.

A través de RESTful APIs. Y encaja en el diagrama donde es rápido y con grandes volúmenes, como Google (no preciso). La información al día siguiente puede que cambie el orden o el contenido, pero nos sirve para funcionar. Y es código abierto, gratis.

Si en Elastic tengo datos almacenándose en tiempo real, en Kibana en dashboard y en el resto se actualizará todo también en tiempo real. Kibana está pensada únicamente para Elasticsearch, mientras que Tableau o PowerBI son de propósito general, caben diferentes fuentes de datos.

Todo en Elastic se guarda como JSON y cada documento que se guarda tiene un ID único. Cada fila de un CSV sería un documento diferente.

Un índice tiene por una parte los metadatos o esquema y por otra el donde se encuentran (cluster, nodo, partición). Indexar en Elastic es guardar (insert) o almacenar. Si te da igual el ID concreto, no te lo inventas tú sino que lo hace Elastic, entonces es POST en vez de PUT. Para traerte documentos se hacen queries mediante JSONs, en vez de SQL. Si el ID ya existiera la inserción fallará, no creará el documento (con el create). Si quieres meterlo nuevo y borrar lo que tenías entonces lo haces indexando, ahí entonces sí se guardará.

El 200 significa que ha ido todo bien.

El ‘score’ es un número para comparar los documentos y ordenarlos a ver cuáles se parecen más a la búsqueda o a la consulta. Cuanto más alto es que más se parece.

El resultado es el mismo, aunque esté puesto en mayúsculas o en minúsculas. No case-sensitive.

<https://github.com/rlopezherrero/GFT-EDEM-MasterData/tree/master/AlmacenamientoProcesamiento>