Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Generación de pipe-lines para transformación de datos en cloud

Trabajo Fin de Máster

**Máster Universitario en Ingeniería Informática**

**Autor**: Ignacio Davó Escribá

**Tutor**: Jon Ander Gómez Adrián

2019/2020

# Resumen

Este Trabajo Final de Master consiste en la creación de un software para la generación de forma automática de pipelines de datos en sistemas distribuidos desplegados en la nube. La herramienta dispondrá de un interfaz web para la interacción del usuario, donde este podrá subir un fichero de muestra de sus datos en formato CSV y seleccionar algunas de las opciones disponibles y el sistema generará las instrucciones necesarias para crear un pipeline de datos de forma automatizada.

**Palabras clave:** pipeline de datos, nube, AWS, big data, generación de pipelines, ficheros CSV.

# Abstract

This Master Final Thesis consists in the creation of a software tool for the automatic generation of data pipelines in distributed systems deployed in cloud. The tool will have a web interface for user interaction, where the user can upload a sample file of their data in CSV format and select some of the available options and the system will generate the necessary instructions to create a data pipeline in an automated way.

**Keywords:** data pipeline, cloud, AWS, big data, pipeline generation, CSV files.

# Índice

# 1. Introducción

*"Done is better than perfect" (Unknown)*

Hace unos años realicé mi Trabajo Fin de Grado sobre la "*Instalación y configuración de herramientas software para Big Data*", especialmente sobre los sistemas del proyecto Apache, Hadoop[1] y Spark[2] (Davó Escribá, 2016). En ese trabajo describía los pasos necesarios para la instalación en un clúster de 32 ordenadores con el que trabajar con estas herramientas orientadas al tratamiento masivo de datos.

Con el paso del tiempo, la realización de este Master Universitario en Ingeniería Informática y mi desarrollo profesional en el terreno del Big Data y el tratamiento de datos, he ido comprobando cómo este tipo de sistemas se pueden desplegar de forma bastante sencilla con distribuciones comerciales de algunas empresas (Cloudera, Hortonworks, MapR Technologies, etc.) [3], con muy poco esfuerzo y disponiendo además de un soporte especializado y de herramientas de gestión exclusivas.

Además de los proveedores de distribuciones de Hadoop ya mencionados, existen en la actualidad proveedores para despliegues *on cloud*, es decir en la nube, de estas tecnologías. En este tipo de despliegues, no solo podemos "desentendernos" de la instalación y configuración de los sistemas, sino que además podemos confiar el hardware subyacente, la gestión de este, su mantenimiento y los problemas que podría generarnos la *escalabilidad*[4] del sistema a un tercero. Estos proveedores *on cloud* sí que son capaces, entonces, de ofrecernos la posibilidad de desplegar un sistema para tratamiento de big data con unos pocos clics de ratón y en cuestión de pocos minutos. Entre los principales proveedores de infraestructura *on cloud*, se encuentran los gigantes de la tecnología Amazon Web Services, Microsoft Azure y Google Cloud [5], además de una miríada de proveedores más pequeños que no ofrecen la misma cantidad de servicios, pero que sí están muy especializados en algún campo en concreto.

Dado que en la actualidad parece tremendamente "sencillo" desplegar nuestros sistemas de tratamiento de grandes volúmenes de datos, parece sensato centrarnos en otro de los grandes retos a los que se enfrentan empresas y organizaciones, es decir, la adquisición, la limpieza y el tratamiento previo de esos datos. Son los llamados procesos de E.T.L (extraer, transformar y cargar) [6], que nos permiten extraer los datos de diferentes fuentes y orígenes, limpiarlos y transformarlos de acuerdo con nuestras

[1] Apache Hadoop es un *framework* de software que permite el procesamiento, tratamiento y almacenamiento distribuido de grandes conjuntos de datos.
[2] Apache Spark es un motor analítico para el procesamiento de grandes conjuntos de datos.
[3] Cloudera Inc. (https://es.cloudera.com/), Hortonworks Inc. (https://es.hortonworks.com/), MapR Technologies Inc (https://mapr.com/)
[4] Escalabilidad: capacidad de un sistema de crecer en magnitud.
[5] Amazon Web Services, AWS (https://aws.amazon.com/), Microsoft Azure (https://azure.microsoft.com/), Google Cloud (https://cloud.google.com/)
[6] ETL del inglés Extract, Transform and Load

necesidades y, como paso final, cargarlos en los diferentes sistemas (normalmente *data warehouse*[7] o *data mart*[8]) para que puedan ser utilizados para análisis o procesos de *Business Intelligence* o *Machine Learning*.

Hoy en día está cobrando una importancia extraordinaria, para las empresas e instituciones pertenecientes a cualquier sector y tamaño, la captura, transformación y almacenamiento de datos de todo tipo. No solo datos empresariales o de negocio, sino también datos de sensores, dispositivos, sanitarios, etc.

Cada día que pasa, utilizamos más y avanzamos más en los procesos de ETL, para organizar y clasificar las ingentes cantidades de datos que capturamos. La utilización de estos procesos de ETL, que tienden a ser repetitivos, se destina básicamente a aplicar sobre ellos las nuevas técnicas de *machine learning* y de *deep learning*, además de otras técnicas de aprendizaje automático (*clustering*, *discovery*, etc.) con el fin de detectar comportamientos pasados de nuestros sistemas o también con el de predecir posibles comportamientos futuros de los mismos.

Existe una tarea para los *data scientists* de descubrimiento (que algunos pueden calificar de tediosa) de los tipos de datos que existen en un dataset a partir de una muestra reducida del mismo. Una vez realizada esta, otra parte de su trabajo consiste en establecer un criterio y dividir los datos en conjuntos más pequeños para poder utilizar una parte de estos para el *entrenamiento* de los modelos y otra parte para *testear* y comprobar hasta qué punto dichos modelos han resultado eficaces y cuál de los diferentes modelos utilizados ofrece un mejor resultado.

Además, todos estos datos necesitan ser puestos a disposición de los ingenieros y aplicaciones de *machine learning* en una ubicación “sencilla” y “altamente disponible” que permita el acceso a ella desde cualquier ubicación.

En este Trabajo Fin de Master, me centraré en la generación de los *pipelines* para captura y transformación de datos y su posterior almacenamiento, realizado todo ello en sistemas *on cloud* y mediante el uso las tecnologías y servicios disponibles en alguno de los múltiples proveedores de infraestructura en la nube que existen.

[7] Data warehouse: Almacén de datos, colección o colecciones de datos de una empresa para ayudar en los procesos de toma de decisiones.
[8] Data Mart: Subconjunto de datos dentro de un *data warehouse*.

## 1.1 Objetivo

El objetivo de este trabajo es la creación de un sistema para generación de *pipelines* de entrada de datos de forma automatizada. El sistema permitirá la lectura de un fichero de ejemplo de los datos a generar (en formato CSV[9]). De este fichero se leerán y analizarán sus características y las características de sus columnas.

En función de la información leída, el sistema generará de forma autónoma un pipe-line de datos para automatizar la lectura de posteriores versiones de ese fichero (normalmente más pobladas y con números de líneas muy superiores) sobre un sistema *cloud*, permitiendo generar datasets de datos para realizar las diferentes labores de entrenamiento y test para algoritmos de *machine learning*.

El sistema, además de detectar de forma automatizada los tipos de datos de cada una de las columnas, determinará si dichas columnas pueden ser anonimizadas o pueden ser enmascarables. Las diferencias entre los mismos son:

- Anonimizable: son datos, generalmente de tipo texto, que no deben ser conocidos por la persona que los gestiona, por ejemplo, los nombres de las personas o los números de D.N.I. en un fichero de movimientos bancarios. Estos datos se pueden anonimizar con alguna función de encriptación. En general, no son datos que se puedan utilizar para procesos de aprendizaje o inferencia, sino exclusivamente para relacionar unos registros con otros.
- Enmascarables: son datos, generalmente de tipo numérico, que no es conveniente que se conozcan para evitar posibles deducciones de nuestros datos, por ejemplo, la edad de nuestros clientes. Puede no interesarnos que se sepa la edad de nuestros clientes para que no se pueda establecer un perfil demográfico de los mismos, sin embargo, puede ser necesario que la edad mantenga unas relaciones de manera que podamos utilizarlo en algún proceso de aprendizaje. El problema con el que nos encontramos para el enmascarado de datos es que debemos tener un conocimiento de todas las muestras antes de enmascarar, ya que necesitamos conocer sus valores máximo, mínimo, etc. antes de poder establecer algún mecanismo de "normalización"

Uno de los mayores problemas con los que se encuentran los ingenieros de *machine learning* es la partición de los datos en conjuntos de entrenamiento, de desarrollo y de test. Por mi experiencia en este campo, creo que en muchas ocasiones es conveniente hacer esa partición de manera que toda la información relacionada con un cliente esté en el subconjunto de entrenamiento o en el de test, pero que no se mezcle. Por ejemplo, en un sistema de gestión de imágenes sanitarias para detección de enfermedades, deberíamos utilizar todas las imágenes de un mismo paciente para *test* o para *train*,

[9] CSV: del inglés *comma-separated values*, o valores separados por comas

independientemente de si en la imagen existe algún problema médico o no. El sistema permitirá elegir a partir de que columna se realiza el particionado.

Una vez seleccionadas las diferentes opciones, el sistema generará los datos necesarios en base de datos para permitir que nuestro sistema cloud reciba ficheros de este tipo, realice las transformaciones necesarias y las almacene en nuestro sistema de almacenamiento donde podrán ser reutilizados por otros equipos de análisis de datos.

De esta forma, podríamos tener un sistema en el que, subiendo una pequeña muestra de nuestros datos, los analizara, transformara y cargara en sistemas de almacenamiento y, posteriormente, (con procesos que no van a ser tratados en este trabajo), se les aplicaran diferentes técnicas y tecnologías de inteligencia artificial para ver cuales obtienen mejores resultados de predicción y poder recomendar al usuario una tecnología u otra.

# 2. Conceptos

## 2.1 Procesos ETL

Los ya mencionados procesos de ETL permiten a las organizaciones realizar las acciones necesarias para llevar a cabo los tres pasos fundamentales:

- *Extracción*: obtención de los datos originales, en muchos casos, de fuentes diferentes y heterogéneas. Las empresas suelen tener los datos en bases de datos relacionales, pero en muchas ocasiones los tienen en otro tipo de estructuras como ficheros de hojas de cálculo, ficheros CSV[10], otros tipos de bases de datos, ficheros de logs. Incluso en la mayor parte de las ocasiones, es necesario obtener datos de fuentes externas a la organización, como pueden ser datos meteorológicos o climatológicos, ratios o coeficientes de uso, estadísticas generales, etc.
- *Transformación*: Estos datos extraídos de alguna fuente (interna o externa a la organización), deben ser, en general, limpiados, transformados y adaptados a nuestras necesidades. Por ejemplo, si estamos obteniendo datos de ficheros de logs para el cálculo de consumos de tiempos de procesos de un sistema, debemos decidir qué hacemos si algún valor nos da negativo (el tiempo de proceso no puede ser negativo). Si estamos obteniendo datos históricos de temperaturas en la comunidad valenciana, debemos decidir qué hacer si algún valor es, por ejemplo, inferior a -10 º C o superior a 60 º C. En definitiva, debemos decidir qué datos vamos a considerar erróneos o anómalos y que acciones vamos a tomar en esos casos (descartar el dato, limpiarlo y corregirlo, etc.). Además, como parte del proceso de transformación, debemos decidir qué datos y de qué manera los vamos a utilizar (con el objetivo de simplificar procesos y cálculos posteriores), por ejemplo, si estamos obteniendo los datos de ventas de una cadena de grandes supermercados a partir de sus tickets de venta, es posible que nos interese agregar y sumar los datos de ventas de artículos diarios por cada uno de los centros ya que el detalle por ticket no nos aporta nada y tener precalculados los datos por día y centro pueden simplificar trabajos posteriores.
- *Carga*: En esta fase, el objetivo es almacenar los datos ya extraídos, transformados y normalizados a otro sistema donde sean fácilmente accesibles por nuestras aplicaciones. En función de la utilidad que vayamos a dar a estos datos, estos sistemas pueden ser bases de datos relacionales, bases de datos clave-valor, o cualquier otro tipo de sistema que sea de utilidad a la organización (normalmente, *Datawarehouse* o *Datamarts*).

[10] CSV: del inglés comma-separated values, tipo de documente de texto que representa los datos en forma de tablas separando las columnas por comas y las filas por saltos de línea.

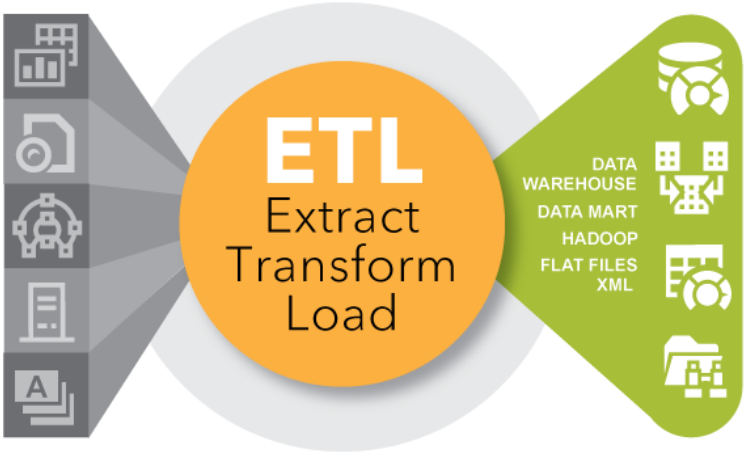


*Imagen 1 - Procesos ETL con varios orígenes y varios destinos*

Una vez finalizada la fase de ETL, tendremos nuestros datos listos en algún sistema que podrá ser accedido de forma "sencilla" por nuestras aplicaciones de *Business Intelligence, Machine Learning* o algún otro tipo de sistema para poder realizar análisis de lo que ha ocurrido en el pasado en nuestra organización o para predecir cuales van a ser los posibles comportamientos de nuestros datos en el futuro.

En los sistemas ETL, las tres fases descritas son importantes, ya que si alguna de ellas no funciona de forma correcta la totalidad del sistema se resiente. Sin embargo, existe un elemento adicional que cobra especial importancia hoy en día con los sistemas Big Data para análisis de grandes volúmenes de datos. El dimensionamiento y la escalabilidad del sistema deben ser establecidos durante el análisis de este, pero que, es posible, que no necesiten disponer de un tamaño constante durante toda la vida útil del sistema. Es decir, es posible que nuestro sistema reciba grandes cantidades de información (y necesite mucha capacidad de proceso) solo los días laborables, pero que se pueda dimensionar al mínimo durante los días festivos y fines de semana ya que en estos períodos no recibe grandes volúmenes de datos.

Además, podría ocurrir que el tiempo del que disponemos para realizar la extracción de algunos datos de origen podría cambiar, lo que implicaría necesariamente incrementar la capacidad de cálculo para procesar la misma cantidad de datos en menos tiempo. Es en este apartado donde cobran importancia los sistemas *on cloud*, que nos permiten los redimensionamientos de los sistemas de forma rápida y sencilla.

Actualmente, en algunos foros se habla también de ETL (extracción, carga y transformación) donde la diferencia es apenas un matiz de en qué orden y como se realizan algunos procesos. En las ETL, se extraen los datos de su origen, se cargan en los nuevos sistemas (normalmente sistemas orientados a *Big Data*) y luego se utiliza la capacidad de proceso de estos sistemas para realizar las transformaciones de los datos. Esto incrementaría la velocidad del proceso ya que este se produce dónde están los datos.

## 2.2 Sistemas Cloud

La computación en la nube[11] es un paradigma de sistemas donde se ofrecen una serie de servicios o infraestructuras informáticas en la red internet, que son proporcionados por algún proveedor de servidor en la nube que los aloja físicamente en sus centros de proceso de datos.

Estos proveedores disponen de infraestructuras (de proceso, almacenamiento, etc.) dinámicas y configurables que nos permiten acceder a ellas de forma modular y escalable sin necesidad de adquisición de equipos o software y con la posibilidad de pago por uso y un uso determinado por nuestras necesidades de servicio y no por la vida útil de los sistemas. Es decir, podemos desplegar un sistema de potentes servidores durante un mes y luego desecharlos cuando ya no los necesitemos. Además, pagamos solo por el tiempo que los hemos utilizado (o han estado en funcionamiento) y no por el tiempo del contrato).

Entre los diferentes tipos de servicios que podemos encontrar en los sistemas cloud se encuentran los siguientes:

- SaaS[12] Software como servicio: Se encuentra en la capa más alta, y ofrece un producto o solución software como un servicio. Estas aplicaciones son accesibles vía web y normalmente son utilizadas por varios clientes. Nosotros no disponemos de ningún tipo de software en nuestros sistemas y nos beneficiamos de las economías de escala, ya que no pagamos el producto sino el uso que hacemos de él. Actualmente existen infinidad de productos de software como servicio, para los más utilizados son servicios de correo (Gmail, Hotmail, Outlook, etc.), servicios de redes sociales e incluso sistemas de productos de ofimática como los nuevos productos de Office 365 de Microsoft.
- PaaS[13] Plataforma como servicio. Está en un nivel inferior al software como servicio, y lo que hace es proporcionarnos algún tipo de funcionalidad concreta (con todos los módulos y desarrollos de software intermedios que conlleva) sobre la que podemos desplegar nuestros sistemas. Un caso muy frecuente son los sistemas de bases de datos en cloud, donde el proveedor del servicio se encarga tanto del dimensionamiento del sistema subyacente como del software del SGBD, así como de sus actualizaciones y parches.
- IaaS[14], la infraestructura como servicio está en el nivel más bajo, y nos proporciona el equivalente al hardware como servicio, proporcionándonos tanto espacio de almacenamiento como capacidad de cómputo. Normalmente estas

[11] Del inglés cloud computing
[12] SaaS Software as a Service. Software como servicio
[13] PaaS Platform as a service. Plataforma como servicio
[14] IaaS Infrastructure as a service. Infraestructura como servicio

soluciones se basan en la virtualización de servidores, proporcionándonos algunas características básicas que podemos dimensionar (como RAM, tipo y numero de CPU´s, espacio en Gb y tipo de disco, etc.). Sobre estos sistemas se instala un sistema operativo que podremos configurar y adaptar a nuestras necesidades.

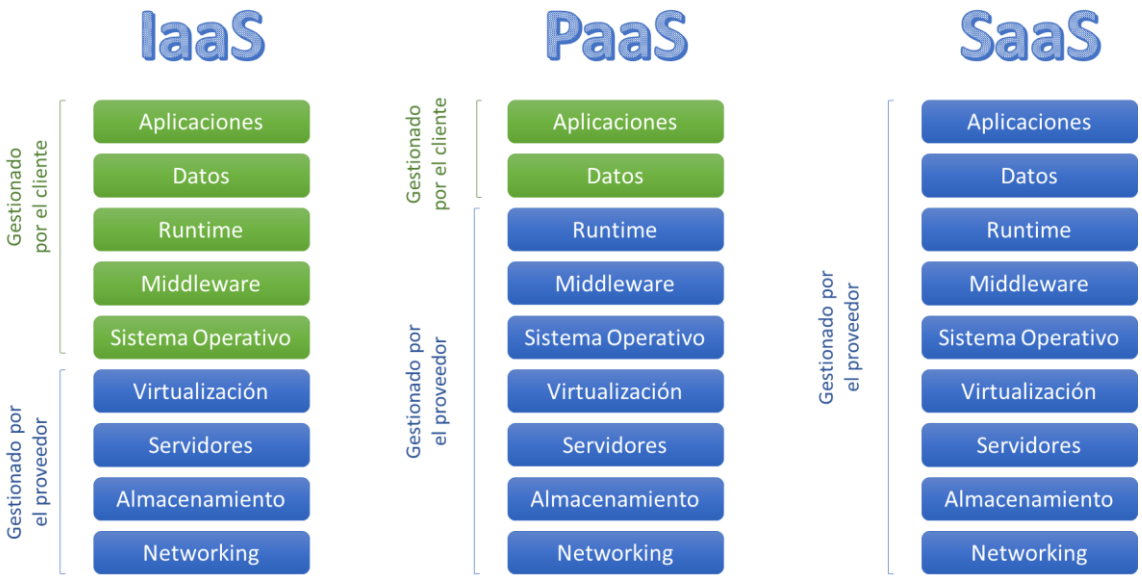


*Imagen 2 - Diferencias IaaS, PaaS y SaaS*

Los sistemas desplegados utilizando infraestructuras *on cloud*, ya sean SaaS, PaaS o IaaS, en cualquiera de los proveedores que nos las pueden ofrecer, tienen una serie de ventajas e inconvenientes comunes. Entre los primeros nos encontramos con:

- Uso eficiente de la energía ya que los data center destinados a cloud optimizan su utilización.
- Se prescinde de la instalación y mantenimiento de software de terceros, ya que es el proveedor de la infraestructura quien lo gestiona.
- Baja inversión inicial, ya que no es necesario adquirir equipamiento o licencias.
- Nula dependencia de sistemas que pueden sufrir fallos o quedarse obsoletos.
- Pago por uso, si un sistema no lo utilizamos no pagamos por él.
- Escalabilidad: en general, el redimensionamiento tanto vertical como horizontal puede ser cuestión de unos pocos clics de ratón.

Entre las principales desventajas tenemos las siguientes:

- Excesiva dependencia de nuestro proveedor de cloud.
- La información de la empresa (o de nuestros clientes) no está alojada en servidores y sistemas que podamos controlar al 100%, por lo que es necesario

revisar si nuestro proveedor y sus sistemas cumplen con los requisitos de la GPDR[15].

- Escalabilidad: la escalabilidad a medio y largo plazo puede verse comprometida por la disponibilidad de sistemas de nuestro proveedor.

[15] GPDR: General Data Protection Regulation: Reglamento General de Protección de Datos

# 3. Estado del arte

Actualmente existen diferentes sistemas desarrollados por grandes compañías para la generación de *pipelines* de datos y su tratamiento. Entre otras muchas podemos encontrar Apache Airflow[16], Apache Beam[17], Amazon Data Pipeline[18], Amazon Athena[19] y otras muchas plataformas comerciales destinadas al tratamiento y/o generación de *pipelines* de datos de forma automatizada.

Sin embargo, todos ellos requieren de amplios conocimientos de tecnología y personal experto para su puesta en marcha y funcionamiento. Uno de los motivos de la complejidad es precisamente su amplitud y eficacia. Son plataformas que permiten realizar infinidad de operaciones diferentes y en multitud de sistemas diferentes, y ello incrementa de forma notable tanto la complejidad del sistema como la curva de aprendizaje de este.

De hecho, algunos de ellos son librerías de código destinadas a integrarse y ser llamadas desde otros programas desarrollados por ingenieros expertos en programación. Estos programas se pueden desarrollar en diferentes lenguajes (Java, Python, etc.) pero requieren conocimientos avanzados de programación.

La idea de este proyecto no es competir con ninguno de estos sistemas ya desarrollados por grandes equipos de ingenieros y con unos niveles de calidad extraordinarios, sino proporcionar una plataforma que, de una manera sencilla, sea capaz de generar un pipeline de datos para extracción, transformación y carga y que estos datos luego puedan estar disponibles para otras aplicaciones u otros módulos de la misma aplicación (no desarrollados en este TFM).

De esta forma podríamos desarrollar un "completo" y sencillo sistema de *machine learning* en el que un usuario sin especiales conocimientos sobre la materia podría, a partir de un simple fichero en formato CSV, crear una configuración y generación automática de un pipeline para que sus datos se fueran adaptando a sus necesidades y posteriormente aplicarle procesos de IA solo seleccionando algunas opciones de menú y sin necesidad de realizar complejos y costosos desarrollos de software.

[16] Apache Airflow: (https://airflow.apache.org/) es una plataforma del proyecto Apache para crear, programar y monitorizar flujos de trabajo mediante programación en python.
[17] Apache Beam es un *framework* del proyecto Apache (https://beam.apache.org/), para el procesamiento de trabajos por lotes y streaming mediante programación en Java, Python o Go.
[18] Amazon Data Pipeline es un servicio de AWS (https://aws.amazon.com/es/datapipeline/) para la generación vía web de *pipelines* de datos.
[19] Amazon Athena es un servicio de AWS (https://aws.amazon.com/es/athena/) para la realización de consultas sobre ficheros utilizando el lenguaje SQL.

## 3.1 Apache Airflow

Es una plataforma en la que podemos crear, planificar y monitorizar *workflows*[20] de trabajo por medio de programación en Python. Un *workflow*, para el sistema Airflow, es una secuencia de tareas que son disparadas por algún tipo de evento o por una planificación detallada y que se utilizan para manejar *pipelines* de datos.

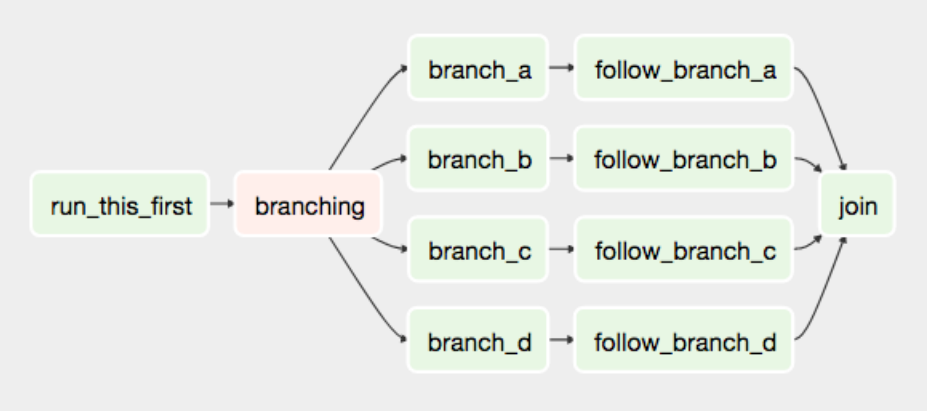


*Imagen 3 - Ejemplo de pipeline de Airflow*

Entre sus principales ventajas están:

- Dinámico: Los *pipelines* de Airflow están programados en Python, por lo que es sencillo escribir código adicional para generar los *pipelines* de forma dinámica. Además, al estar en Python, es posible realizar cualquier tipo de transformación necesaria a los datos de forma "sencilla".
- Extensible: De la misma manera, es fácil extender y ampliar las librerías con nuevas operaciones definidas por el usuario, para estandarizar y reutilizar algunas de las funciones que realicemos.
- Escalable: Airflow utiliza una arquitectura modular, utilizando colas de mensajes para gestionar un numero arbitrario de trabajadores (*workers*), lo que nos permite una gran escalabilidad horizontal.

## 3.2 Apache Beam

Es un *framework* para la definición de líneas de procesado de datos (tanto en modelo *batch* como en *streaming*), y un conjunto de librerías específico para construir *pipelines* y ejecutarlos en diferentes lenguajes de programación (Go, Java y Python). Los sistemas

[20] Workflows, flujos de trabajo.

de Beam pueden ser ejecutados en diferentes entornos de computación distribuida como Apache Flinx, Apache Spark, Google Cloud Dataflow, Hazelcast Jet, etc.

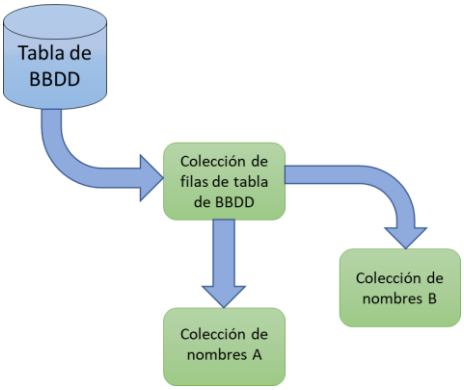


*Imagen 4 - Workflow de Apache Beam*

Está orientado a tres tipos de usuarios diferentes:

- Usuarios finales: Programan *pipelines* utilizando uno de los SDK[21] (*software development kit*) existentes y ejecutándolo sobre alguno de los ejecutores existentes. Se centran en programar la lógica de la aplicación.
- Desarrolladores de SDK: Son expertos en algún lenguaje de programación y desarrollan un SDK para una comunidad de usuarios específica y para un lenguaje específico.
- Desarrolladores de ejecutores: Desarrollan modelos y sistemas de ejecución de trabajos para entornos distribuidos (tanto en modelo *batch* como en *streaming*).

## 3.3 AWS Athena

El servicio de Amazon Web Services, Athena, es un servicio de consultas interactivo que permite realizar diferentes tipos de análisis de datos en almacenados en el servicio S3[22] de Amazon, con diferentes tipos de formatos de ficheros y, utilizando para ello el lenguaje estándar SQL[23]. Athena no utiliza un servidor dedicado para prestar el servicio, sino que es un servicio puro en la nube y utiliza la infraestructura propia de AWS para su ejecución, sin que el usuario tenga que preocuparse de configuraciones ni infraestructuras, además, el pago se realiza en base a las consultas que se ejecutan.

[21] SDK kit de desarrollo de software
[22] S3, del inglés Simple Storage Service, servicio sencillo de almacenamiento, es el servicio de Amazon para el almacenamiento de todo tipo de ficheros en la nube.
[23] SQL del inglés *structured query language*, lenguaje de consultas estructurado, se utiliza para consultas en bases de datos.

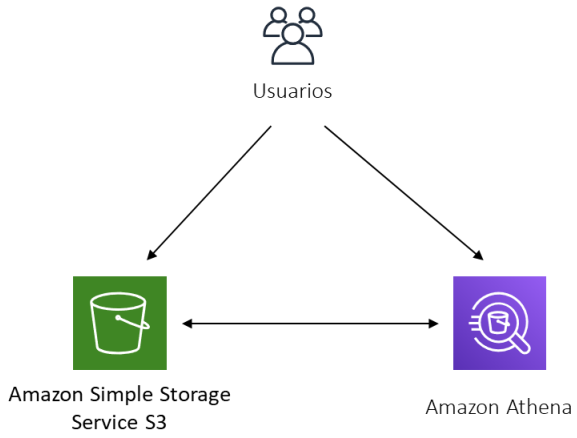


*Imagen 5 - Esquema de uso de Amazon Athena*

Con AWS Athena, tras una definición y configuración del sistema donde básicamente definimos los tipos de archivos que el sistema va a leer y cuál es su estructura. Podemos utilizar ficheros con diferentes formatos como CSV, Parquet [24], JSON, etc. y posteriormente le definimos al sistema que estructura tienen dichos ficheros (columnas, tipos de datos, etc.). Una vez definido, el resto del proceso es sencillo, subimos los ficheros de datos a un bucket [25] de S3 y lanzamos consultas en formato SQL a la plataforma de Amazon Athena.

La plataforma utilizará sus propios sistemas para localizar la información dentro de los ficheros que hemos subido a S3 y mostrarnos los resultados de forma rápida y eficaz.

## 3.4 AWS Data Pipeline

Es un servicio web que permite procesar y transformar datos entre diferentes plataformas de almacenamiento y computación dentro del propio entorno de AWS. puede acceder fácilmente a los datos desde la ubicación donde se almacenan, transformarlos y procesarlos a escala, y transferir los resultados de manera eficiente a los servicios de AWS como Amazon S3, Amazon RDS, Amazon DynamoDB y Amazon EMR. Le permite crear cargas de trabajo complejas de procesamiento de datos que son tolerantes a fallos, repetibles y altamente disponibles.

[24] Parquet: Formato de almacenamiento del ecosistema Hadoop para ficheros orientados a columnas.
[25] Bucket, cubo, espacio privado (o público) de almacenamiento de ficheros de una cuenta en S3.

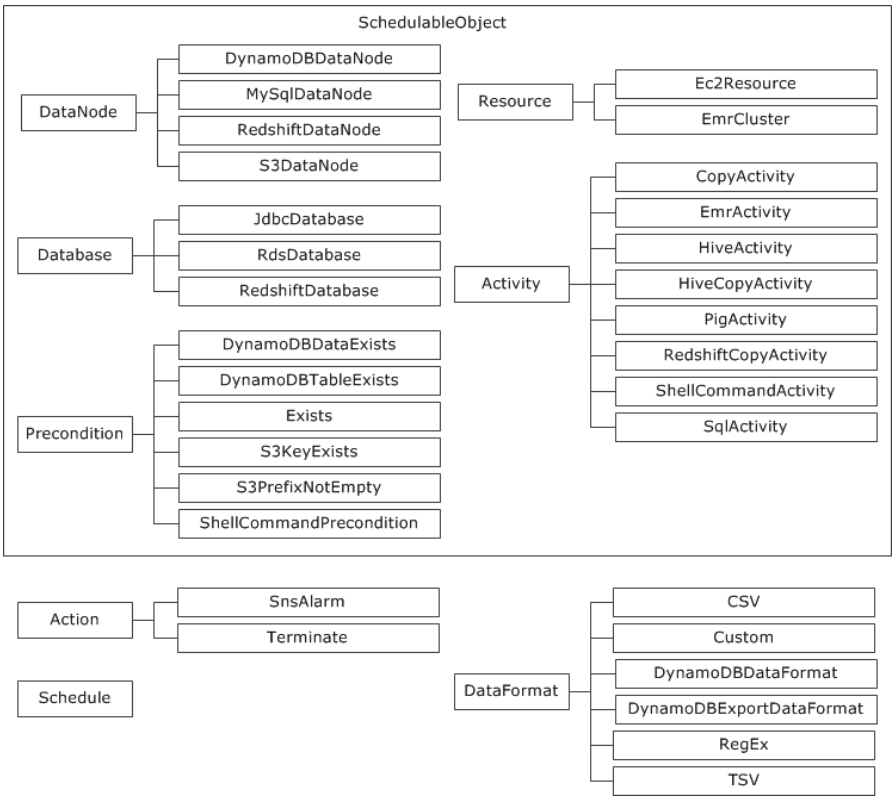


*Imagen 6 - Jerarquía de objetos de AWS Pipeline*

Entre sus ventajas podemos encontrar:

- Proporciona una consola *drag-and-drop*[26] en la interface de AWS.
- Está desarrollada sobre una infraestructura distribuida con alta disponibilidad y tolerante a fallos.
- Es igual de fácil ejecutar el trabajo en una máquina o en varias.

[26] Drag-and-drop seleccionar un objeto de una ubicación y arrastrarlo a otra diferente.

# 4. Planificación

Entre los elementos más importantes de cualquier proyecto se encuentra su planificación, que es donde se detalla que tareas son las que van a ser realizadas, con sus costes y duraciones estimadas. Además, se deben detallar para cada una de ellas quien o quienes son los responsables de ejecutarlas y llevarlas a cabo. En este caso, al tratarse de un proyecto unipersonal, no tiene sentido la asignación de tareas ya que todas corresponden a la misma persona.

En circunstancias normales, una buena planificación del proyecto puede ser básica y nos ayuda a mantener el control sobre la evolución del proyecto y su ejecución y, de esta forma, detectar la mayoría de las desviaciones incluso antes de que se produzcan. De esta forma, podremos aplicar las medidas necesarias (tanto en la faceta temporal como en la económica) con el fin de evitar "sorpresas" de última hora.

La planificación inicial del proyecto está compuesta por cinco tareas básicas que he desglosado en la siguiente tabla. Los tiempos de ejecución de las tareas se han estimado en horas. La conversión a días no es posible ya que me dedicación al desarrollo de este proyecto es variable debido a mis obligaciones tanto profesionales como personales, por lo que la duración en días es muy variable. Existen dos tareas con una carga muy superior de trabajo (como era de esperar) y son las de desarrollo de la aplicación y la de redacción de esta memoria.

| Tarea | | Duración |
|---|---|---|
| **Tarea 1** | **Preparación proyecto** | **15** |
| | Preparación proyecto | 10 |
| | Planificación del proyecto | 5 |
| **Tarea 2** | **Obtención documentación inicial** | **55** |
| | Sobre Pipe Lines | 10 |
| | Sobre Amazon Web Services | 30 |
| | Sobre otras tecnologías | 15 |
| **Tarea 3** | **Desarrollo aplicación** | **170** |
| | Desarrollo módulos BBDD | 20 |
| | Desarrollo Python | 100 |
| | Desarrollo PHP | 50 |
| **Tarea 4** | **Integración y pruebas de sistema** | **80** |
| | Despliegue sobre AWS | 20 |
| | Pruebas caso de uso Titanic | 20 |
| | Resolución y correción errores | 40 |
| **Tarea 5** | **Redacción Memoria** | **150** |
| | Documentación adicional | 25 |
| | Imágenes y Tablas | 20 |
| | Escritura memoria | 80 |
| | Revisión y corrección | 25 |

*Tabla 1 - Tabla de tareas*

Para la planificación del proyecto, se han obviado todos los temas relacionados con costes económicos debido a los motivos siguientes:

- Todos los productos que se van a utilizar (tanto de software como de lenguajes de programación) son de software libre (MySQL, PHP, Python etc.), y no es necesario realizar ningún desembolso por su utilización.
- Se van a utilizar algunos servicios de Amazon Web Services (AWS) en la nube que, inicialmente, tienen un coste establecido, sin embargo, entre las ventajas de AWS se encuentra el disponer de una capa de uso gratuito para todos sus productos y servicios, lo que hace que utilizarlos para pruebas pueda ser realizado sin coste. En cualquier caso, los servicios que se implementen lo serán exclusivamente durante el tiempo necesario para realizar las pruebas, por lo que, al disponer de pagos por uso, estos serán muy reducidos.
- En cuanto a los costes de la parte de recursos humanos, elemento fundamental en cualquier proyecto relacionado con el desarrollo de software, no se han tenido en cuenta ya que serán exclusivamente las horas de dedicación del autor, que al tener intereses concretos en el desarrollo del proyecto no será computado como coste.
- Otro elemento que sería necesario computar como coste son las horas de dirección del proyecto, en este caso a cargo de Jon Ander Gómez como director de este, en formato de reuniones, tutorías y correcciones del trabajo. A pesar de que este apartado es siempre fundamental en un proyecto, tampoco serán contabilizadas por tratarse de un trabajo académico.

No se ha realizado un diagrama de Gantt, ya que el calendario de ejecución del proyecto ha ido variando mucho a medida que se ha realizado. Tal y como ya he comentado, debido a mis obligaciones familiares y profesionales, no ha sido posible comprometerme a realizar un determinado número de horas de trabajo diarias o semanales, por lo que alguna tarea de pocas horas me ha llevado algún mes y otras las he resuelto en unos pocos días. La ejecución práctica del proyecto ha dependido mucho del resto de mis responsabilidades, además de algunos factores externos (por todos conocidos y que han generado retrasos e inconvenientes en múltiples actividades y sectores), por lo que el tiempo de ejecución real ha sido muy superior al que determinan las horas de dedicación.

.

# 5. Recursos

Para el desarrollo de este Trabajo Fin de Master, he tenido que decidir entre la selección de diferentes herramientas y proveedores para el desarrollo, despliegue y pruebas de la solución. Este TFM no pretende ser ni realizar un análisis de las diferentes herramientas y de los diferentes proveedores de servicios en cloud, por lo que no voy a entrar a detallar las diferencias entre los mismos, sus servicios ni sus características, pero sí que expondré los principales motivos que me han llevado a tomar una decisión tecnológica de entre las diferentes posibles.

## 5.1 Amazon Web Services (AWS)

Entre los diferentes proveedores posibles de soluciones en cloud, me he decantado directamente por los servicios ofrecidos por Amazon Web Services (AWS), frente a otros posibles como Google Cloud o Microsoft Azzure.

Y, tal y como he dicho anteriormente, este TFM no pretende ser ni realizar un análisis de los diferentes proveedores de servicios en cloud, pero sí que expondré los principales motivos de mi elección:

- En el pasado, he realizado algún curso avanzado de arquitectura de aplicaciones y sistemas sobre Amazon Web Services.
- A nivel profesional, trabajo y despliego servicios y sistemas en AWS con cierta frecuencia.
- Tengo una cuenta personal en AWS con la que ya he realizado diferentes proyectos.
- Disponen de un eficaz y completo sistema para interactuar con sus servicios a través de la línea de comandos (CLI), también se puede interactuar a través de su interface web y disponen de librerías SDK para Python (***boto3***) de fácil uso y que permiten la inclusión de comandos en código Python.
- Disponen de una capa gratuita de uso que, en general, es suficiente para el desarrollo de este trabajo.
- Disponen de uno de los mayores catálogos de servicios existentes en la nube, donde podemos encontrar absolutamente de todo, desde sencillas máquinas de un *core* y 512 Mb. de RAM (EC2), sistemas de bases de datos relacionales gestionados (RDS), despliegues de grandes clústeres con servicios Map Reduce (EMR), bases de datos de grafos (Neptune), bases de datos NoSQL (DynamoDB), balanceadores de carga, gestores de trabajos *batch*[27], etc.

[27] Batch: trabajos por lotes.

### 5.1.1 Amazon EC2

Amazon Elastic Cloud Computing (Amazon EC2) es uno de los pilares de los servicios en la nube ofrecidos por Amazon Web Services. Permite generar ordenadores y sistemas virtuales en los que se puede ejecutar las aplicaciones de los usuarios, según el modelo IaaS (infraestructuras como servicio) comentados en la introducción.

A través de una interfaz de servicios web muy sencilla de utilizar, AWS permite configurar y desplegar diferentes tipos de ordenadores, con diferentes tipos de capacidades, características y sistemas operativos en función de las necesidades del usuario.

Además de la flexibilidad y facilidad para el despliegue de sistemas, cuenta con las siguientes ventajas:

- Escalabilidad: todos sus sistemas son fácilmente escalables.
- Inmediatez: cualquier sistema desplegado es accesible de forma prácticamente inmediata (en cuestión de pocos minutos en el peor de los casos).
- Seguros y fiables: los servicios EC2 de AWS disponen de un acuerdo de nivel de servicio (SLA[28]) del 99.99%. Además, su arquitectura de red basada en redes privadas está diseñada para proporcionar la máxima seguridad a sus sistemas.
- Coste: el pago de los servicios EC2 es por uso, lo que quiere decir que, si desplegamos un equipo virtual y lo utilizamos durante unas pocas horas al mes, solo pagaremos por esas horas (siempre que en el resto de horas este apagado). Este modelo es perfecto para proyectos de corta duración o proyectos que tienen una gran variabilidad en el uso.

| Nombre | CPU Virtual | Memoria GiB | Precio/Hora Linux |
|---|---|---|---|
| t3.nano | 2 | 0,5Gib | 0,0057 USD |
| t3.micro | 2 | 1 GiB | 0,0114 USD |
| t3.small | 2 | 2 GiB | 0,0228 USD |
| t3.medium | 2 | 4 Gib | 0,0456 USD |
| t3.large | 2 | 8 Gib | 0,0912 USD |
| t3.xlarge | 4 | 16 GiB | 0,1824 USD |
| t3.2xlarge | 8 | 32 Gib | 0,3648 USD |

*Tabla 2 - Tabla precios AWS EC2 del CD de Irlanda*

[28] SLA: Service Level Agreement)

Según la tabla de precios anterior, si dispusiéramos de un equipo t3.large en Linux utilizado durante 2 horas diarias durante los 30 días del mes, su coste total sería 5.472 $ o su equivalente en euros.

### 5.1.2 Dynamo DB

Amazon Dynamo DB es uno de los servicios de AWS de base de datos NoSQL (de clave-valor y documental) que ofrece un alto rendimiento en la nube. Es una base de datos de fácil despliegue y completamente administrada, lo que libera al usuario de las complejas tareas de administración y gestión.

Además, en caso necesario se puede desplegar como sistema distribuido y como clúster en cloud. Entre sus ventajas se encuentran:

- No es necesario desplegar un servidor para alojar esta base de datos. El servicio se autodimensiona en función de las necesidades.
- Soporta transacciones ACID[29]
- Datos cifrados de origen
- Copias de seguridad automatizadas.
- Pago por uso como en el resto de los servicios de AWS

## 5.2 Python 3.7

Python es un lenguaje de programación multiparadigma, que soporta programación funcional, programación orientada a objetos (OOP) y programación imperativa. Es un lenguaje interpretado, además es multiplataforma, por lo que puede ser ejecutado en Windows, Mc Os y Linux. Además, su asignación de tipos es dinámica y es de tipado fuerte.

Es, además, un lenguaje de código abierto. Entre las razones que me han llevado a elegirlo están las siguientes:

- Tal y como comentaba en el punto anterior referido a los servicios de AWS, a nivel profesional también trabajo diariamente con Python 3.7, por lo que estoy muy familiarizado con el lenguaje.
- El código fuente está delimitado por el sistema de identación (las funciones y bloques se separan por espacios al principio de cada línea, en lugar de por las

[29] ACID (referido a bases de datos) Del inglés Atomicity, Consistency, Isolation and Durability y en castellano Atomicidad, Consistencia, Aislamiento y Durabilidad

tradicionales llaves {}). De esta forma, el código Python se caracteriza por ser un código ordenado y limpio.
- Como ya he dicho antes, es multiparadigma y de código abierto. Además, existen multitud de herramientas y librerías para Python también de código libre.
- Los principios de la filosofía Python que priman la simplicidad y legibilidad.
- Es un lenguaje muy portable por lo que permite desarrollar en cualquier plataforma y ejecutar en cualquier otra sin necesidad de realizar cambios al código.
- Además, es uno de los lenguajes preferidos en los campos de la ciencia de datos y el *machine learning*.

## 5.3 Pandas 1.0.2

Pandas es una biblioteca de software desarrollada para el lenguaje Python y ampliamente utilizada en entornos de Ciencia de Datos ya que es muy rápida y potente. Ofrece las estructuras y operaciones necesarias para el manejo de múltiples tipos de datos y series temporales. Entre sus características más destacadas se encuentran:

- Utiliza un tipo de datos *dataframe*, que es una estructura de datos muy utilizada consistente en vectores de datos de diferentes tipos.
- Tiene herramientas para lectura/escritura desde estructuras de ficheros como CSV, Excel, Bases de datos SQL, etc.
- Herramientas para el manejo de datos faltantes o incorrectos.
- Permite la mezcla, unión y agrupaciones de datos.
- Amplias capacidades de manejo de series temporales.

## 5.4 PHP 7

El PHP[30] es un lenguaje de programación del lado del servidor, utilizado para el desarrollo web con contenido dinámico. Entre los motivos que me han llevado a seleccionar PHP para la programación del lado de servidor están:

- Fácil de aprender.
- Su código no es visible en el navegador web del cliente, y se ejecuta en la parte del servidor.
- Módulos para conexión con diferentes sistemas de bases de datos, en especial con MySQL.
- Es un software de código abierto.

[30] PHP del inglés Hypertext Preprocessor (preprocesador de hipertexto)

## 5.5 MySql 5.7

MySQL es un sistema de gestión de bases de datos relacionales (SGBDR) de código abierto que dispone de una licencia de uso pública general (además de una comercial de Oracle Corporation).

Actualmente es una de las bases de datos más extendidas ya que dispone de módulos de conexión en casi todos los lenguajes de programación y se integra de forma excepcional con los sistemas Linux, Python y PHP. Entre los motivos para seleccionarla se encuentran:

- Es *open source* (licencia GPL)
- Es uno de los gestores de bases de datos libres con mejor rendimiento.
- Puede ser ejecutado en máquinas de bajos recursos.
- Muy seguro, baja probabilidad de corrupción de datos.
- Facilidad de instalación, configuración y mantenimiento.

## 5.6 Bitbucket GIT

GIT es un sistema de control de versiones de código abierto que fue desarrollado inicialmente por Linus Torvalds y que está muy extendido en las comunidades de desarrollo de software para controlar versiones en equipos de trabajo.

Bitbucket es un sistema de gestión de código y versiones GIT *on cloud* y que es gratuito para uso particular.

He decidido utilizar este sistema de versionado en la nube por varios motivos:

- Es un sistema estandarizado y de fácil uso.
- Aloja todo nuestro software *on cloud*, por lo que nos independizamos de la máquina en que desarrollamos.
- Es gratuito para uso personal.
- Estoy familiarizado con el sistema ya que lo utilizo a nivel profesional (en su versión de pago para empresas).

# 6. Desarrollo del proyecto

## 6.1 Arquitectura hardware

Para el desarrollo y despliegue del sistema, he elegido una instancia EC2[31] en Amazon Web Services. La instancia que he elegido dispone de las siguientes características:

- Instancia t3a.medium
- Utiliza 2 CPUs virtuales
- Dispone de 4 GiB de RAM
- Está instalada la versión 18.04 de Ubuntu
- Como SGBD he instalado MySQL 5.7
- Además, tiene instalado Python 3.7 (para el backend) y Apache2 2.4 y PhP 7 para el *frontend*.

No es necesario para este proyecto desplegar máquinas independientes para el *frontend* web, para el *backend* y para la base de datos, ya que por el volumen de trabajo se puede integrar perfectamente en un único sistema, aunque está diseñado de manera que se puedan separar para instalar cada una de las partes en sistemas independientes y que de esta manera sea fácilmente escalable tanto de forma vertical como de forma horizontal.

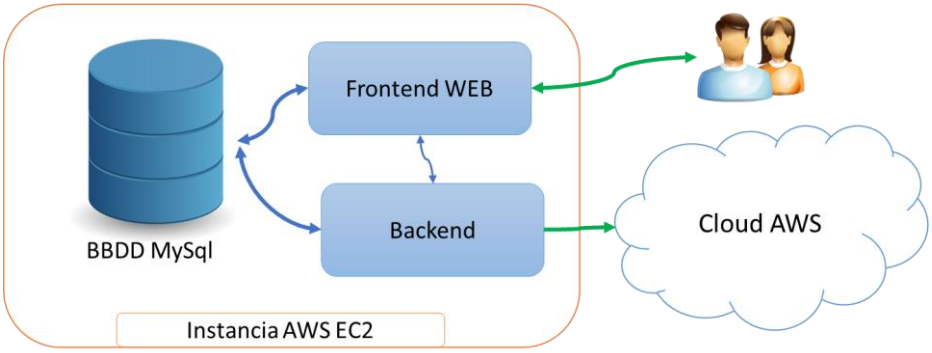


*Imagen 7 - Esquema del sistema*

En nuestro sistema, los usuarios interactuarán con el *frontend* web desarrollado, donde podrán establecer las características y patrones de comportamiento necesarios del sistema. Estas características definidas por el usuario serán almacenadas en nuestra base de datos, de manera que puedan ser recuperadas y reutilizadas en cualquier momento.

[31] EC2 de Elastic Computer Cloud

Desde este sistema web, se darán las instrucciones de ejecución necesarias a nuestro sistema *backend* para que realice el análisis de nuestros datos, y genere las instrucciones necesarias (en forma de *script*[32]) para poder ser ejecutadas en nuestro sistema cloud.

## 6.2 Diseño base de datos

Para el desarrollo de todo el sistema, tal y como se ha comentado antes, se ha utilizado el sistema de gestión de bases de datos MySQL. En este sistema se han definido una serie de tablas que nos permitirán mantener la persistencia de la información relacionada con cada uno de los procesos realizados y su estado.

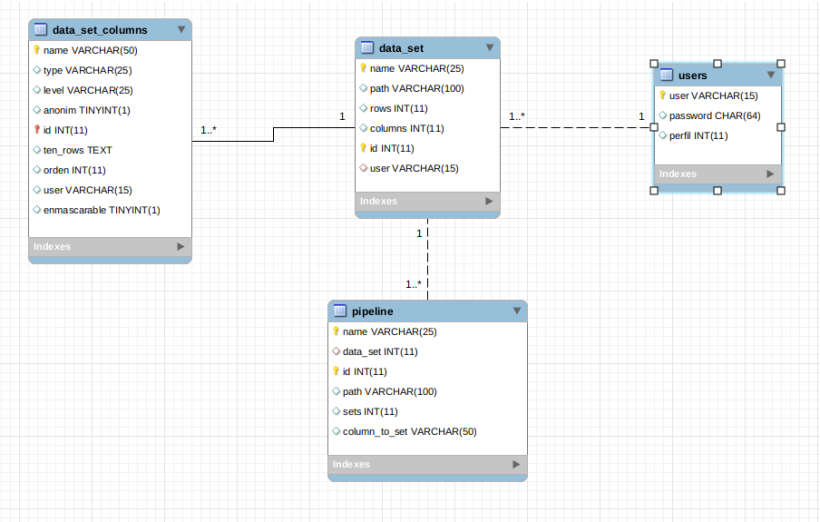


*Imagen 8 - Esquema de Base de Datos*

En cuanto a la descripción detallada de las tablas y su uso, tenemos las siguientes:

- Tabla **Users**: tabla donde se almacenan los datos de los usuarios con permisos de acceso al sistema. Aunque podría contener muchos más datos, por simplicidad la he limitado a los datos necesarios.

| Columna | Tipo | Observaciones |
|---|---|---|
| User | Varchar(15) | Nombre de usuario para identificación en el sistema |
| Password | Char(64) | Password hasheada con la función de PHP password_hash() |
| Perfil | Int | Perfil para el caso de que queramos establecer diferentes niveles de acceso |

*Tabla 3 - Tabla users*

[32] Script: archivo de ordenes o archivo de procesamiento por lotes

- Tabla **data_set**: es la tabla donde se almacena la información relacionada con cada uno de los data sets que hemos subido a la plataforma. Contienen información de cada uno de los diferentes ficheros CSV subidos a la plataforma y son reutilizables para otros usos con características diferentes:

| Columna | Tipo | Observaciones |
|---|---|---|
| Id | Int | Identificador único del data_set. Autoincrementado |
| Name | Varchar | Nombre del data set |
| Path | Varchar | Path con la ubicación completa del archivo CSV que hemos utilizado para generar el data set |
| Rows | Int | Numero de filas presentes en el data set |
| Columns | Int | Numero de columnas presentes en el dataset |
| User | Varchar(15) | Clave ajena que identifica al usuario propietario |

*Tabla 4 - Tabla dataset*

- Tabla **data_set_columns**: en esta tabla almacenaremos la información de cada una de las columnas correspondientes a los diferentes data sets, entre ellas, sus características y tratamientos para anonimizar o enmascarar. Además, conservaremos en un campo de texto el contenido de las 10 primeras filas para esta columna. El valor está desnormalizado de forma consciente, ya que no será necesario dar un tratamiento posterior a estos datos y simplifica bastante tanto la estructura de la base de datos como su tratamiento.

| Columna | Tipo | Observaciones |
|---|---|---|
| Id | Int | Clave ajena que identifica al data set que corresponde el dato. |
| Name | Varchar | Nombre de la columna. |
| Type | Varchar | Tipo que corresponde a los datos de la columna. |
| Level | Varchar | Subtipo que corresponde a la columna. |
| Anonim | Tinyint | 1/0 en función de que se tenga que anonimizar. |
| Enmascarable | Tinyint | 1/0 si se debe enmascarar el dato. |
| Orden | Int | Numero de orden de la columna dentro del data set. |
| Ten_rows | Text | Campo de texto con el valor para esta columna de las 10 primeras filas separados por comas. |

*Tabla 5 - Tabla data_set_columns*

- Tabla **pipeline**: contiene datos relacionados con la generación del *pipeline* concreto para un *data set*. Entre ellos, el número de conjuntos de datos que generaremos (entrenamiento, desarrollo y/o test) y que porcentaje de los datos (de forma aproximada) utilizaremos para cada uno de los conjuntos.

| Columna | Tipo | Observaciones |
|---|---|---|
| Id | Int | Identificador del pipeline. Autoincrementado |
| Name | Varchar | Nombre del pipeline. |
| Dataset | Int | Clave ajena que identifica el id del dataset |
| Path | Varchar | Path completo donde se encuentra el fichero con los comandos para ejecutar el pipeline. |
| Sets | Int | Número de conjuntos de datos que se deben generar (entrenamiento, desarrollo y test) |
| Colum_to_set | Varchar | Identificador de la columna utilizada para realizar la partición del dataset |

*Tabla 6 - Tabla pipeline*

## 6.3 Configuración

El sistema dispone de una carpeta *config* en la que he ubicado un fichero de configuración *database.ini* en formato estandarizado de manera que el software sea capaz de *parsear*[33] los datos contenidos. Este fichero será utilizado tanto por el *frontend* (en la parte PHP) como por el *backend* (en la parte Python) para, entre otras cosas, obtener los datos necesarios para la conexión a la base de datos. El fichero dispone de tres secciones diferenciadas:

- MySQL: Sección destinada a contener los datos de configuración de la base de datos de MySQL.
- Email: Datos de configuración para el envío de emails por parte del sistema.
- Globales: Datos globales de la aplicación, entre ellos los datos personales del autor.

De esta manera, si fuera necesario cambiar datos de configuración del sistema, no sería necesario modificar el código de este, sino únicamente modificar los datos en el fichero de configuración. Este fichero queda de la siguiente manera (se han eliminado datos de

[33] Parsear: analizar sintácticamente mediante un programa informático

teléfonos y contraseñas exclusivamente por motivos de seguridad y de confidencialidad):

```
[mysql]
host=sistema_host
db=tfm
user=nacho
passwd=password

[email]
servidor=smtp.office365.com
puerto=587
desde=igdaes@inf.upv.es
password=password

[globales]
copyright="Nacho Davo TFM"
nombre="Nacho Davo TFM"
telefono="(+34) 658 XXX XXX"
email=igdaes@inf.upv.es
server="TFM"
```

*Imagen 9 - Fichero de configuración*

## 6.4 Desarrollo frontend

El *frontend* de nuestro sistema se ha desarrollado utilizando una combinación de las siguientes tecnologías:

- HTML y CSS: Son tecnologías estándar de desarrollo web.
- Bootstrap: Es un *framework* de código abierto para sitios y páginas web. Dispone de múltiples elementos de diseño web basados en HTML y CSS. Además, ofrece a las páginas web la posibilidad de hacer su desarrollo *responsive*[34] sin demasiado esfuerzo.
- JavaScript y Ajax También son tecnologías estándar para desarrollo web y se han utilizado para la realización, en la parte cliente del frontal web, de algunas funciones que HTML no permite.
- PHP: Se ha utilizado para toda la parte de la página web que debe ser ejecutada sobre el servidor, de forma especial los accesos a base de datos y las solicitudes de ejecuciones de procesos de *backend*.

[34] Responsive: Diseño web adaptable, permite adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas

La finalidad del código *frontend* de nuestro sistema (como en la mayoría de las ocasiones) es proporcionar una interface sencilla al usuario donde pueda interactuar con nuestra aplicación sin necesidad de conocimientos informáticos.

El sistema está pensado para proporcionar una página web al usuario donde poder subir sus ficheros CSV y que le permita lanzar sus procesos de *backend* (desarrollados en Python) sin conocer ni donde se encuentran estos scripts de código ni cómo se lanzan. Para ello utilizará combinaciones de botones, *checkboxes*, listas desplegables y demás elementos disponibles en HTML para la gestión de páginas web.

Para la entrada al sistema, será necesario identificarse con usuario y contraseña válidos que estarán almacenados en nuestra base de datos de MySQL. Una vez identificado correctamente, tendremos acceso al menú de la aplicación donde podremos subir nuevos ficheros CSV (Datasets) o lanzar nuevos *pipelines* sobre los *datasets* ya creados.

Entre los *scripts* de código generados, los más destacados son los siguientes (con descripción de sus funciones más importantes y propósito):

- **funciones.php:** Este es un fichero que alberga algunas funciones estándar y comunes en PhP para el resto del sistema. El resto de los módulos harán uso de él incluyéndolo en su código. Entre las funciones que están desarrolladas se encuentran:
    - **abrir_db():** esta función está destinada a abrir la base de datos y devolver el objeto que la controla. Lee los parámetros de la base de datos desde el fichero de configuración.

```
function abrir_db() {
    $ruta_ini="config/database.ini";
    $array_ini = parse_ini_file($ruta_ini, true)["mysql"];

    $DB_HOST=$array_ini["host"];
    $DB_USER=$array_ini["user"];
    $DB_PASS=$array_ini["passwd"];
    $DB_NAME=$array_ini["db"];

    $mysqli = new mysqli($DB_HOST, $DB_USER, $DB_PASS, $DB_NAME);
    if (mysqli_connect_errno()) {
        session_destroy();
        $_SESSION["mensaje"]="Connection failed: ".mysqli_connect_error();
        $mensaje="Error de conexión de base de datos";
        header( "Location: " . 'index.php?mensaje='.$mensaje );
        exit(0);
    }
    return $mysqli;
}
```

*Imagen 10 - Función Abrir_db*

- **access_user()**: Función destinada a comprobar si el usuario y contraseña introducidos son correctos y si el usuario tiene acceso a la aplicación. Para la comprobación de la password introducida, se han utilizado las funciones internas de PHP password_hash (para generar la password) y password_verify (para comprobar que la password es correcta). De esta manera, la password se almacena encriptada sin que, incluso con acceso a la base de datos, se pueda conocer la password de un usuario.

```php
function access_user($usuario, $password){
    $mysqli = abrir_db();

    $respuesta=[0,"",9];
    if ($stmt = $mysqli->prepare("SELECT user, password,perfil
                                  FROM users WHERE user=?;")) {
        $stmt->bind_param("s", strtolower($usuario));
        $stmt->execute();
        $stmt->store_result();
        $stmt->bind_result($name,$realpass,$perfil);
        $stmt->bind_result($name,$realpass,$perfil);
        if ($stmt->num_rows>0) {
            $stmt->fetch();
            if (password_verify($password,$realpass)) {
                $respuesta=[1,$name,(int) $perfil];
            }
        }
        $stmt->close();
    }
    $mysqli->close();
    return $respuesta;
}
```

*Imagen 11 - Función access_user()*

- **show_header()**: Función destinada a visualizar correctamente los menús en cada una de las páginas de la aplicación. Debido a su extensión, no la reproduzco en este documento.

- **jsfunctions.js:** Este es un fichero que alberga algunas funciones de uso común para el código JavaScript. Entre ellas, se pueden destacar:
  - **String.prototype.hashCode**: prototipado[35] de la función hasCode para los objetos de tipo *string*. Esta función se utiliza para obtener un código *hash* de ejemplo para los valores de tipo *string* que sean hasheables.

[35] Prototipado: Sistema de generación de clases y métodos para JavaScript

```
String.prototype.hashCode = function() {
    var hashed_code = 0, i, character;
    if (this.length === 0) return hashed_code;
    for (i = 0; i < this.length; i++) {
      character   = this.charCodeAt(i);
      hashed_code  = ((hashed_code << 5) -
                     hashed_code) + character;
      hashed_code |= 0;
    }
    return hashed_code;
  };
```

*Imagen 12 - Prototype función hashCode*

- **anonimizar(elemento)**: función que realiza el anonimizado de las columnas seleccionas haciendo uso del prototipado de la función anterior. Esta función obtiene un *hash* de una cadena de *string* (por ejemplo, de un nombre), sustituyendo su valor de manera que quede protegido y el usuario no pueda reconocerlo. Se utiliza para columnas que no sean susceptibles de ser utilizadas en procesos de *machine learning*.

```
function anonimizar(elemento){
      var checkBox=document.getElementById("id-ch"+elemento);
      if (checkBox.checked == true) {
          for (i = 0; i < 10; i++){
            fila="tb"+i;fila=fila+elemento;
            valor=document.getElementById(fila).getAttribute("name");
            document.getElementById(fila).innerHTML=valor.hashCode();
          }
      } else {
          for (i = 0; i < 10; i++){
              fila="tb"+i;fila=fila+elemento;
              document.getElementById(fila).innerHTML=
                  document.getElementById(fila).getAttribute("name");
          }
     }
}
```

*Imagen 13 - Función anonimizar*

- **ofuscar_array()**: Función para enmascarar datos. Esta función se utiliza normalmente para datos de tipo numérico, cuando queremos que el dato no sea reconocible pero que se siga pudiendo utilizar en procesos de *machine learning* (por ejemplo, para la edad).

```
function ofuscar_array(arr){
    mean = math.mean(arr);
    std = math.std(arr);
    arr = arr.map( function(value) {return value - mean } );
    arr = arr.map( function(value) {if (std!=0)
          {return Math.round(value / std * 10000)/10000} else {return 0} } );
    return arr;
}
```

*Imagen 14 - Función ofuscar_array*

- **ofuscar()**: función utilizada para ofuscar los datos de una columna utilizando la función de *ofuscar_array()* anterior:

```
function ofuscar(column){
    var checkBox=document.getElementById("id-ch-of"+column);
    var elementos=[];
    if (checkBox.checked == true) {
        for (i = 0; i < 10; i++){
            fila="tb"+i;fila=fila+column;
            elementos.push(document.getElementById(fila).getAttribute("name"));
        }
        elementos=ofuscar_array(elementos);
        for (i = 0; i < 10; i++){
            fila="tb"+i;fila=fila+column;
            document.getElementById(fila).innerHTML=elementos[i];
        }
    } else {
         for (i = 0; i < 10; i++){
            fila="tb"+i;fila=fila+column;
            document.getElementById(fila).innerHTML=
                    document.getElementById(fila).getAttribute("name");
        }
    }
```

*Imagen 15 - Función ofuscar*

Tal y como he comentado anteriormente, el enmascarado (ofuscado), consiste en ocultar los valores reales de los datos, pero haciendo que mantengan un sistema de proporciones (normalizado) de manera que utilizándolos en determinados procesos de *machine learning*, los resultados obtenidos sean los mismos que con los valores originales. El problema es que, para aplicar esta normalización, es necesario conocer el *dataset* completo, conociendo sus valores máximos, mínimos, medias, etc. Para la visualización de ejemplo en la pantalla, se utilizan los datos del *dataset* reducido subido al sistema.

- Además, existen una serie de ficheros PHP encargados de la obtención de datos en nuestro servidor web y la generación del código HTML correspondiente para la página web. Estos ficheros utilizan las propiedades de la tecnología PHP para obtener los datos necesarios desde nuestro servidor de MySQL y generar, de forma dinámica y en base a los resultados obtenidos, el correspondiente código HTML que nos permita navegar por nuestro sistema.

  El código PHP también es el encargado (utilizando funciones AJAX) de actualizar en nuestra base de datos la información nueva o modificada introducida por el usuario en la página web.

## 6.5 Desarrollo backend

Para el desarrollo del *backend* de nuestro sistema se ha elegido el lenguaje de programación Python en su versión 3.7. El módulo *backend* será llamado por el módulo *frontend* (PhP) a petición del usuario para realizar dos tareas diferenciadas:

- La primera tarea por realizar es la de lectura del fichero CSV subido por el usuario y el reconocimiento de todos sus campos. El fichero subido puede ser una muestra reducida (en cuanto a número de filas) de un fichero real, pero es cierto que cuantas más filas tenga, más probable será que la detección de algunas características de las columnas sea correcta. Por ejemplo, si disponemos de un fichero con 2 filas y para una columna tenemos en una fila el valor 0 y en otra el valor 1, el sistema no puede distinguir si se trata de un valor *booleano*[36] (0/1) y si es una entero. Sin embargo, si en ese mismo fichero disponemos de 100.000 líneas, si en todas las líneas aparecen los valores 0/1, lo tratará como un booleano.
- La segunda tarea es la de, una vez analizado el tipo de ficheros CSV, y con la información procedente de ellos y almacenada en nuestra base de datos MySQL, realizar la lectura de los correspondientes ficheros CSV con el grueso de datos para ir traspasando la información a nuestras tablas en la base de datos clave/valor *on cloud* DynamoDB.

### 6.5.1 Lectura CSV

El código Python para la lectura de ficheros CSV es llamado desde la función de PhP. Este código tiene que realizar varias funciones previas a la generación de los correspondientes *pipelines*. Lo primero que necesita realizar es la lectura del fichero y el paso a una estructura de datos manejable.

Para poder realizar el análisis de los ficheros CSV subidos por el usuario, he realizado una clase, llamada **Datos**, que contendrá información de la configuración del sistema. Para que el sistema sea capaz de reconocer de forma correcta los tipos de algunas columnas, se han generado ficheros con valores obtenidos de internet:

- Nombres: se han obtenido desde internet 105.000 nombres propios en diferentes idiomas y se han almacenado en un fichero de tipo *pickle* [37] llamado **namelist.pickle**.
- Ciudades: se ha generado un fichero con 10.000 nombre de poblaciones de diferentes países, también de tipo *pickle*. **Townlist.pickle**.

[36] Booleano: Tipo de dato con valor de lógica binaria, es decir con 2 valores posibles *(true/false*, 0/1, etc.)
[37] Pickle: Librería de Python para almacenamiento en fichero de objetos de memoria.

- Países: Se ha generado un fichero con los nombres (en diferentes idiomas) de los países del mundo. **Countrylist.pickle.**

Estos tres ficheros generados de forma previa, nos permitirá detectar de una forma "sencilla", algunas columnas y sus tipos de datos al poder detectar, por comparación, las columnas que contengan nombres propios, países o ciudades.

Lo primero que realiza el código es la creación de una clase llamada Datos, que será el tipo de objeto donde almacenaremos la información leída desde el fichero CSV. Al generar una instancia de la clase Datos, se cargan, por un lado, las listas de los ficheros de nombres, poblaciones y países que he comentado anteriormente. Por otro lado, se cargan algunos parámetros de configuración (que pueden ser modificados) y que determinarán el comportamiento posterior del sistema.

```
# ratio mínimo de nombres/municipios/pais para considerar la característica como tal
self.rationombres = 0.85
self.ratiomuni = 0.7
self.ratiopais = 0.8
```

*Imagen 16 - Configuración ratios*

Con estos ratios de configuración, podemos establecer, por ejemplo, que una columna la consideraremos que son nombres propios cuando el 85% de sus filas coincidan con un nombre propio, o una columna la consideraremos como nombres de países cuando el 80% de sus filas corresponda a uno de los nombres de países definidos en nuestros ficheros de referencia.

Este sistema nos permitiría, en el futuro, añadir nuevos datos que consideremos estandarizados y que deben corresponder a algún tipo de característica concreto, como pueden ser los códigos postales, profesiones, códigos CNAE de actividades, etc.

Para la lectura del fichero CSV, utilizamos la librería de pandas y su función *read_csv* sobre la instancia de la clase datos generada:

```
try:
    self.data = pd.read_csv(csvpath)
except IOError as e:
    print('Error al cargar csv.', e)
    return
```

*Imagen 17 - Lectura fichero CSV*

De esta forma, sobre el atributo *data* de nuestra instancia de la clase Datos, tendríamos el Dataframe leído desde el fichero CSV con todos nuestros datos ya organizados en columnas. Una vez leído nuestro dataset, procedemos a realizar algunas estadísticas de este, para ver cuáles son los tipos de datos que contiene en cada una de las columnas y su distribución. Esto es realizado por la función de análisis de características:

```
def _analisis_features(self):
    self.feature_set = set(self.data.columns)
    describe = self.data.describe().T
    type = pd.Series(self.typedict).rename('type')

    card = pd.Series(self.data.apply(lambda x: x.value_counts().shape[0])).rename('card')
    nans = self.data.isnull().sum().rename('nans')

    nUnbaDistval = pd.Series(self.data.apply(
        lambda x: next(i + 1 for i, v in enumerate(accumulate(x.value_counts(normalize=True)))
        if v >= 1 - self.ratiomindistintosmoda)))
    nUnbaDistval.name = 'nUnbaDistval'

    nOneHotval = pd.Series(self.data.apply(
        lambda x: next(i + 1 for i, v in enumerate(accumulate(x.value_counts(normalize=True)))
        if v >= 1-self.ratiomaxrestantes)))
    nOneHotval.name = 'nOneHotval'

    info = pd.concat([describe, type, card, nans, nUnbaDistval, nOneHotval], axis=1)
    return info
```

*Imagen 18 - Análisis de características*

Este nos permite determinar si en una columna existen datos numéricos o de caracteres, así como su tipo exacto de dato (entero, decimal, fecha, etc.) y además podemos establecer, gracias a sus distribuciones estadísticas, si una columna está compuesta de ordinales, intervalos, datos categóricos, binarios, etc.

```
def _detecta_nombre(self):
    f_contienenombre = lambda x: any(subcadena in self.namelist for subcadena in trata_nombres(x))
    l_caract_nombre = []
    stringfeatures = [feature for feature in self.data.columns if self.typedict.get(feature, None) is str]
    for feature in stringfeatures:
        serie = self.data[feature]
        numnombres = sum(serie[serie.notnull()].apply(f_contienenombre))
        if numnombres / sum(serie.notnull()) > self.rationombres:
            l_caract_nombre.append(feature)
    self.meta.loc[l_caract_nombre, 'level'] = 'name'
    self.meta.loc[l_caract_nombre, 'keep'] = False
```

*Imagen 19 - Función detección nombres*

Una vez realizado el análisis de características a nuestro *dataset*, realizamos un análisis con los filtros establecidos previamente para nombres, poblaciones y países, de forma que podemos detectar estos valores en cualquiera de nuestras columnas. La función utilizada para ello es la siguiente (únicamente se muestra la función de detección de nombres, ya que las otras son iguales):

Por último, solo sería necesario determinar cuáles de nuestras columnas son susceptibles de ser anonimizadas (las de tipo carácter y que no son categóricas) y que columnas de ser enmascaradas (ofuscables) (que son las de tipo numérico). Una vez determinado estos tipos podemos guardar la información procedente del análisis en nuestro sistema de base de datos, para su uso posterior. La información sería almacenada en las tablas ***data_set*** y ***data_set_columns*** descritas en el punto 4.2 de esta memoria.

### 6.5.2 Generación pipeline

La segunda tarea para nuestro código Python, es la generación del código necesario para, a partir de la información leída en nuestro fichero CSV, generar las correspondientes tablas de datos on line en DynamoDB. Para ello he utilizado la librería *Boto3*[38] de Amazon.

El sistema puede recibir uno o varios ficheros CSV para analizar su información y trasladarla a nuestra base de datos de DynamoDB. Las tareas que debe realizar el sistema son las siguientes:

- Lectura de la base de datos de MySQL para leer la información relacionada con las columnas del fichero CSV y sus características. Esto lo podemos hacer de forma fácil con una simple consulta a la base de datos:

```
def get_pipeline_data(id):
    conn = open_BBDD()
    cursor = conn.cursor()
    query = "SELECT name, data_set,path,sets,column_to_set " \
          "from pipeline where id=%s"
    cursor.execute(query,(id,))
    data=cursor.fetchall()
    cursor.close()
    conn.close()
    return data
```

*Imagen 20 - Consulta datos pipeline*

[38] Boto3: Librería SDK de Amazon que permite a los desarrolladores Python crear, configurar y gestionar los diferentes servicios de AWS a través de una API de servicio.

- A continuación, una vez leídos los datos de características, necesitamos crear la tabla en DynamoDB donde serán almacenados todos los datos. Como el proceso puede ser ejecutado en diferentes ocasiones y con diferentes ficheros CSV, si la tabla ya existe no será necesario crearla.

```
client = boto3.client('dynamodb', aws_access_key_id=ACCESS_KEY,
                aws_secret_access_key=SECRET_KEY, region_name = 'us-east-1')
AttributeDefinitions = [{'AttributeName': 'id', 'AttributeType': 'N'},
                 {'AttributeName': 'datasetgroup', 'AttributeType': 'S'}]
KeySchema = [{'AttributeName': 'id',
     'KeyType': 'HASH'}]
KeySchemaIndex = [{'AttributeName': 'datasetgroup',
     'KeyType': 'HASH'}]
try:
   response = client.create_table(AttributeDefinitions=AttributeDefinitions,
                        TableName=nombre_tabla,
                        KeySchema =KeySchema,
                        GlobalSecondaryIndexes=[
                          {
                            'IndexName': 'datasetIndex',
                            'KeySchema': KeySchemaIndex,
                            'Projection': {'ProjectionType': 'ALL'},
                            'ProvisionedThroughput': {
                              'ReadCapacityUnits': 10,
                              'WriteCapacityUnits': 10
                            }
                          },
                        ],
                        ProvisionedThroughput={
                          'ReadCapacityUnits': 10,
                          'WriteCapacityUnits': 10
                        } )
except:
   print('La tabla ya existe')
```

*Imagen 21 - Creación tabla en DynamoDB*

Para la creación con Boto3 de la tabla, necesitamos proporcionar los parámetros de ACCESS_KEY y SECRET_KEY que, en mi caso, están definidas en variables de entorno de mi sistema. He definido un campo ***id*** que será utilizado como campo de identificación único del sistema y clave principal del mismo. Este id se genera por la unión del *timestamp*[39] del sistema en el momento de la ejecución y un número aleatorio de 8 dígitos para evitar coincidencias. Se realiza con la siguiente instrucción de Python:

```
id = str(round(datetime.now().timestamp())) + str(random.randint(10000000, 99999999))
```

- Podemos leer nuestro fichero CSV a una estructura de datos tipo *dataframe* utilizando la librería Pandas. Además, como ventaja adicional, esta librería nos permite leer el fichero a trozos (*chunks*), por lo que, aunque el fichero sea muy grande y no quepa en memoria en nuestro sistema podremos procesarlo correctamente. Estableciendo un valor para el parámetro *chunksize*, el sistema lee las líneas correspondientes de nuestro fichero y genera un objeto sobre el que podemos iterar. Además, con Pandas podemos establecer innumerables parámetros para leer correctamente nuestro fichero de datos. La instrucción utilizada es:

```
pd.read_csv(file_to_read, sep=',', quotechar='"', chunksize=chunksize)
```

- Para establecer a que dataset debemos asignar cada muestra (fila en nuestro CSV), utilizamos una función *hash* sobre la columna que teníamos que utilizar para el particionado. Esta función nos devuelve un valor numérico y obtenemos su resto de dividir ese valor por 10. De esta forma, obtenemos para cada valor de la columna, un valor entre 0 y 9, distribuidos de forma uniforme (para grandes volúmenes de datos) y que asigna siempre el mismo valor para el mismo valor de la columna. De esta forma aseguramos que todas las muestras que tengan el mismo valor en la columna de reparto irán destinadas al mismo conjunto de datos. Para la asignación, he creado un diccionario con 10 elementos (del 0 al 9) y cuyos valores son los nombres de los *datasets* (*train*, *test*, *development*) y aparecen tantas veces como porcentaje queramos repartir, es decir, si queremos un 70% de nuestros datos a *train*, este valor aparecerá 7 veces; si queremos un 20% para test, este valor aparecerá 2 veces y, por último, el otro 10% será para development, por lo que aparecerá una sola vez. En mi sistema de pruebas, para un dataset de 892 muestras, y repartiendo por la columna nombre (*Name*) en 2 grupos (train y test) con un reparto de 70% y 30%, el sistema de reparto automático repartió 277 muestras para el grupo de *test* (un 31%) y 615 muestras para el grupo de *train* (un 69%).

- A continuación, procedemos a iterar sobre las líneas leídas de nuestro fichero CSV. Debemos comprobar si alguna de las columnas debe ser anonimizada, para lo que utilizamos una función *hash* procedente de la librería *hashlib* de Python y

[39] Timestamp: Marca de tiempo medida en segundos transcurridos desde el 1 de enero de 1970

utilizamos el algoritmo *sha 512*. Con estos datos, generamos la línea de información que requiere el sistema boto3 para insertar una línea en DynamoDB y lanzamos la consulta para generarla (con la función *put_item*):

```
for _,datos in csv.iterrows():
    id = str(round(datetime.now().timestamp())) + \
         str(random.randint(10000000, 99999999))
    Item={'id':{'N':str(id)}}
    for _,column in df.iterrows():
        if column['anonim']:
            hash_object = hashlib.sha512(datos[column['name']]
                                         .encode('ISO-8859-1'))
            cadena = hash_object.hexdigest()
        else:
            cadena = str(datos[column['name']])
        if column['name'] == column_to_set:
            dataset = hash(cadena) % 10
            Item['datasetgroup']={'S':conjuntos[dataset]}
        Item[column['name']]={datatypes[column['type']]:cadena}
    response = client.put_item(TableName=nombre_tabla, Item=Item)
```

*Imagen 22 - Escritura en DynamoDB*

Por supuesto, en esta función, y antes de insertar los datos en el diccionario Item (utilizado como parámetro para el envío de datos a DynamoDB), podemos aplicar cualquier función de transformación de estos, o añadir algún elemento adicional que no estuviera en nuestro fichero CSV y, por ejemplo, estuviera alojado en alguna tabla de alguna base de datos auxiliar, etc.

# 7. Despliegue

El despliegue del Sistema ha sido bastante fácil al realizarse todo *on cloud* sobre la máquina descrita en el punto 4.1 de esta memoria. Después de generar la máquina e instalar todo el software, sus actualizaciones y dependencias, el primer paso ha sido la creación de las tablas en el sistema MySQL. Las tablas se han creado con las siguientes instrucciones:

```
CREATE TABLE `users` (
    `user` varchar(15) NOT NULL,
    `password` char(64) DEFAULT NULL,
    `perfil` int(11) DEFAULT NULL,
    PRIMARY KEY (`user`)
);

CREATE TABLE `data_set` (
    `name` varchar(25) NOT NULL,
    `path` varchar(100) DEFAULT NULL,
    `rows` int(11) DEFAULT NULL,
    `columns` int(11) DEFAULT NULL,
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `user` varchar(15) DEFAULT NULL,
    PRIMARY KEY (`id`, `name`),
    KEY `fk_data_set_1_idx` (`user`),
    CONSTRAINT `fk_data_set_1` FOREIGN KEY (`user`) REFERENCES `users` (`user`)
            ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE `data_set_columns` (
    `name` varchar(50) NOT NULL,
    `type` varchar(25) DEFAULT NULL,
    `level` varchar(25) DEFAULT NULL,
    `anonim` tinyint(1) DEFAULT '0',
    `id` int(11) NOT NULL,
    `ten_rows` text,
    `orden` int(11) DEFAULT NULL,
    `user` varchar(15) DEFAULT NULL,
    `enmascarable` tinyint(1) DEFAULT '0',
    PRIMARY KEY (`id`, `name`),
    CONSTRAINT `fk_data_set_rows_1` FOREIGN KEY (`id`) REFERENCES `data_set`
            (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE `pipeline` (
    `name` varchar(25) NOT NULL,
    `data_set` int(11) DEFAULT NULL,
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `path` varchar(100) DEFAULT NULL,
    `sets` int(11) DEFAULT '1',
    `column_to_set` varchar(50) DEFAULT NULL,
    PRIMARY KEY (`id`, `name`),
    KEY `fk_pipeline_1_idx` (`data_set`),
    CONSTRAINT `fk_pipeline_1` FOREIGN KEY (`data_set`) REFERENCES `data_set`
          (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

*Imagen 23 - Creación tablas BBDD*

Una vez creadas las tablas en la base de datos, se ha procedido a crear los correspondientes usuarios y permisos en las mismas. Después de ello se ha desplegado el software en la carpeta */var/www/html* utilizando para ello el sistema GIT y haciendo un *clone* del repositorio del software.

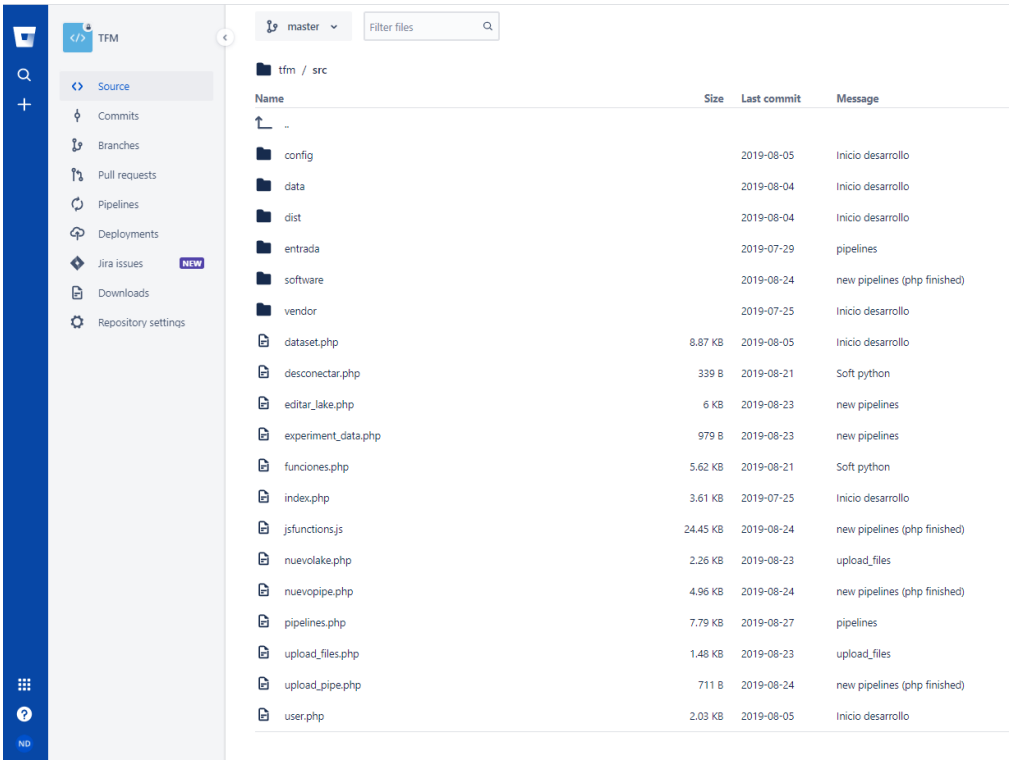


*Imagen 24- Pantalla Bitbucket*

Para la realización del clone del sistema se ha utilizado la instrucción proporcionada por Bitbucket para ello:

```
ubuntu@tfm:/var/www/html$ git clone https://nachodavo@bitbucket.org/nachodavo/tfm.git
```

Como el repositorio contiene todo el código tanto de *backend* como de *frontend*, una vez clonado ya disponemos de todo el código en nuestra máquina. Cualquier cambio de versión posterior podrá ser desplegado de forma inmediata utilizando la instrucción ***git pull***, que actualizará nuestro despliegue con la última versión.

Por último, solo nos queda configurar nuestro servidor web Apache y sus ficheros de *sites* para que redirija a nuestros visitantes a la carpeta donde hemos desplegado nuestro software. Además, deberemos configurar los permisos de acceso a carpetas para que ningún usuario malintencionado pueda acceder a información o ficheros no autorizados.

Para un despliegue escalable y para grupos de usuarios elevados, lo adecuado sería desplegar la base de datos MySQL en un sistema propio (por ejemplo, en el sistema

RDS [40] de AWS). Además, el código de la web podría estar en máquinas EC2 pertenecientes a un grupo de auto escalado que se auto dimensionara en caso de necesidad. Para el sistema de DynamoDB no sería necesario, ya que el mismo se auto dimensiona y escala en función de las necesidades.

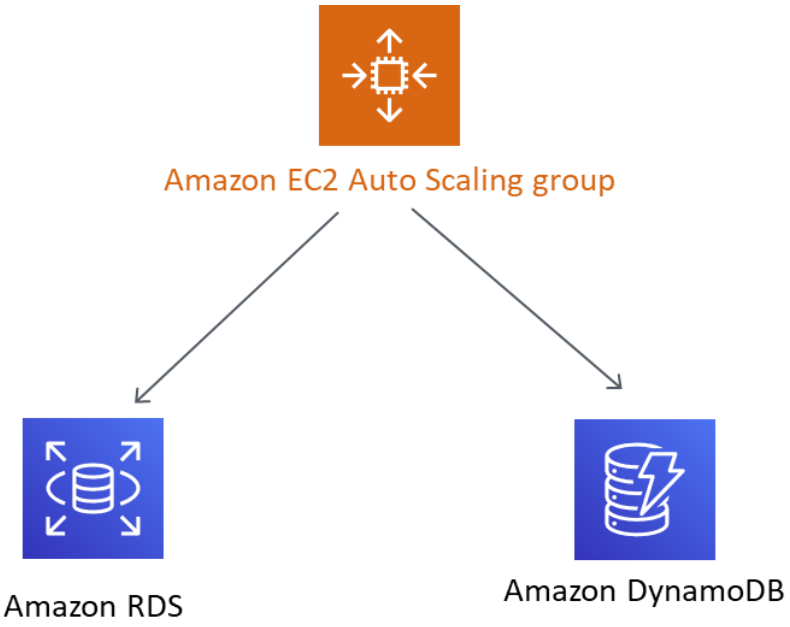


*Imagen 25 - AWS Auto scaling*

Como una evolución adicional, podríamos separar el sistema *backend* del sistema *frontend* y alojar cada uno de ellos en grupos de auto escalado diferentes. Esta opción sería mucho más razonable ya que, previsiblemente, los sistemas que más carga de trabajo deberían tener son los sistemas *backend*, con procesos más pesados y largos. Además, nos permitiría configurar máquinas *ligeras* (de poca capacidad) para el sistema web y máquinas más grandes para el sistema de *backend*. Así pues, nuestro sistema definitivo para un despliegue real *on cloud* podría ser algo similar a esto:

[40] RDS: Relational Database System, servicio de bases de datos relacionales de AWS.

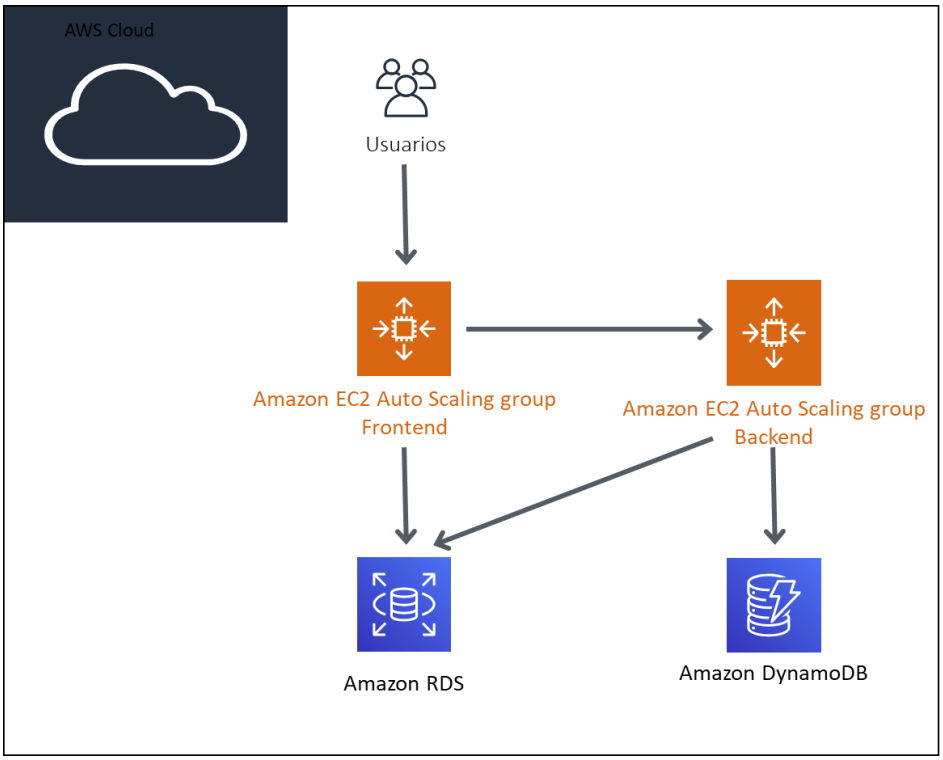


*Imagen 26 - Sistema auto escalado final*

# 8. Caso de uso Titanic

Para la prueba del sistema se ha utilizado el fichero de la competición de *machine learning* propuesta en Kaggle[41] para predecir la supervivencia de los pasajeros del Titanic. Este fichero de datos, que es de uso público, se puede descargar desde la URL https://www.kaggle.com/c/titanic/data

En la aplicación web existen dos opciones diferenciadas, la primera opción es la de Dataset, que permite subir nuevos ficheros CSV al sistema para ser analizados. La segunda es la opción de *pipelines*, que permite definir cómo vamos a generar nuestro *pipeline* a partir de un *dataset*.

## 8.1 Opción de Dataset

Una vez accedemos a la aplicación vía web (no utilizamos una URL concreta sino su dirección IP, ya que no se ha creado un dominio para ello) y tras loguearnos con nombre de usuario y contraseña correctos, aparecemos en la pantalla de inicio del sistema.


*Imagen 27 - Pantalla de inicio*

En esta pantalla podemos ver una tabla con los diferentes datasets (ficheros CSV) que hemos subido en la aplicación. En nuestro caso podemos ver dos, el de los ficheros del Titanic y otros de una POC (*Proof of Copcept*, prueba de concepto) de un cliente. Las columnas que aparecen en dicha tabla son las siguientes:

- **Nombre**: el nombre que hemos dado a nuestro *dataset*.

[41] https://www.kaggle.com/

- **Columnas**: número de columnas que hemos encontrado en el fichero de datos.
- **Filas**: número de filas diferentes de nuestro *dataset*.
- **Anonimizable**: si existe alguna columna que pueda ser anonimizable, es decir, sustituir sus valores por valores "anónimos" de manera que no puedan ser reconocidos (por ejemplo, los nombres o números de identificación).
- **Enmascarable**: si existe alguna columna con características para ser enmascarable, es decir, que sus valores puedan ser sustituidos por otros pero que deban mantener relaciones de proporcionalidad para poder ser utilizados en procesos de *machine learning* (por ejemplo, la edad).
- ***Pipelines***: número de *pipelines* de datos que hemos generado con este dataset.
- Además, existen dos columnas, una de ellas para visualizar los datos del *dataset* y la otra para eliminar el *dataset* en caso de necesidad.

Existe también un botón *Nuevo* para que podamos subir un nuevo fichero CSV y generar así su correspondiente dataset. Una vez introducimos el nombre, el sistema activa de forma automática el botón para subir archivo.

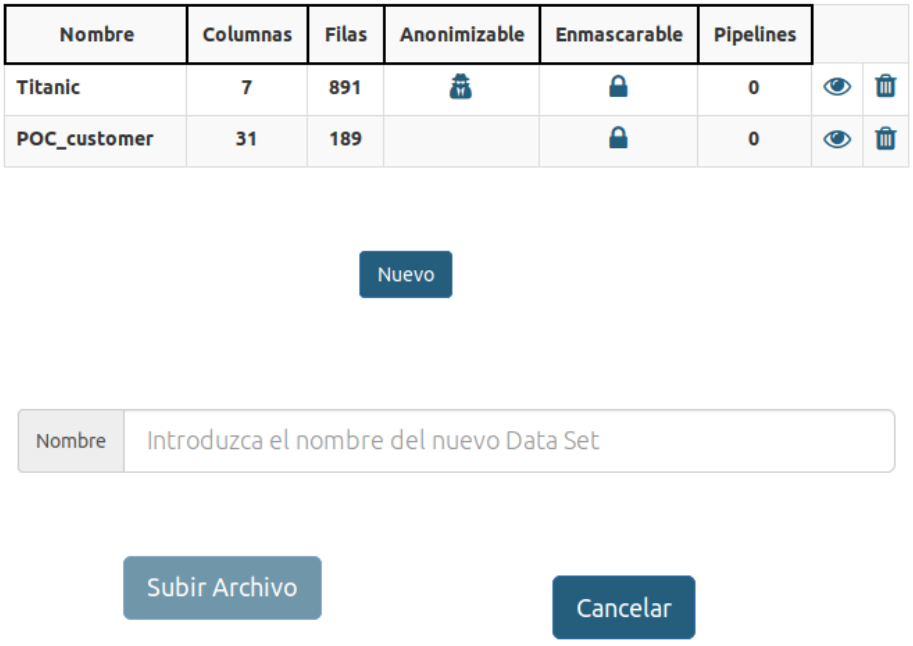


*Imagen 28 - Nuevo data set*

Una vez pulsado dicho botón, se lanzan los procesos de la parte *backend* del sistema para analizar las diferentes características del fichero que hemos subido. Este proceso puede llevar unos minutos en función del fichero y de su extensión y características.

Cuando finalice el proceso, los datos principales serán almacenados en la base de datos de MySQL y este nuevo dataset nos aparecerá en la lista de la pantalla de inicio, donde podremos ver sus columnas y las características detectadas.

Si en la pantalla principal seleccionamos la opción visualizar el dataset, el sistema abre la siguiente pantalla donde se pueden ver los datos concretos de este conjunto de datos. Además de los datos de columnas y filas, podemos ver dos tablas, una en el lado izquierdo y otra en el lado derecho.

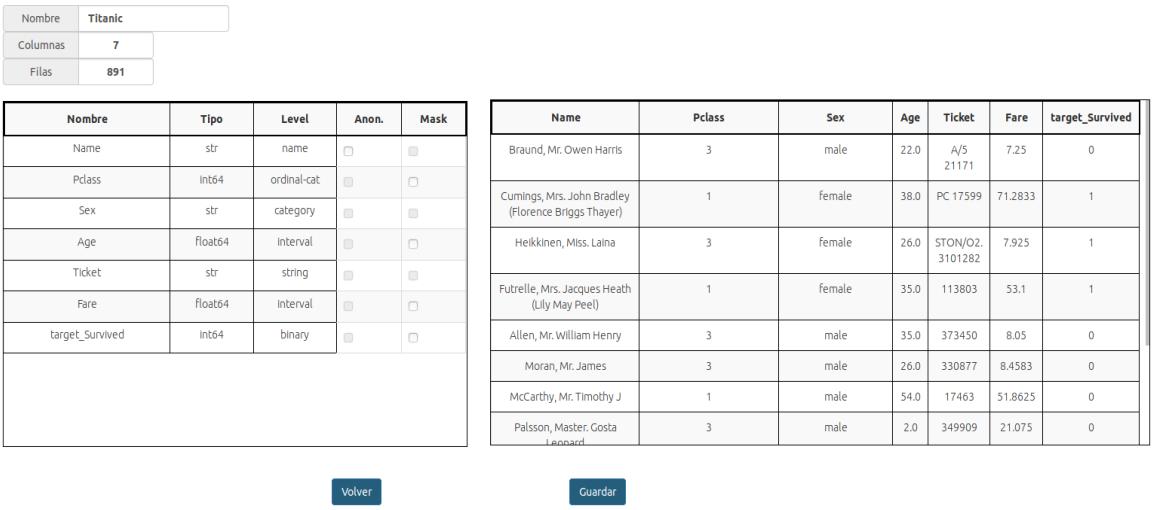


*Imagen 29 - Columnas data set*

- En la tabla de la izquierda, cada una de las filas representa una de las columnas que han sido encontradas en nuestro fichero CSV (en nuestro ejemplo 7 columnas) donde, además, establece que tipo de datos se ha encontrado en este fichero de datos para esa columna. Además, aparece un subtipo de datos (*Level*) que representa como están organizados esos datos. De esta forma, para la columna *Sex (*que contiene el sexo*)*, el sistema ha detectado que es un tipo *str* (*string*) y un subtipo *category*, es decir que es una cadena de caracteres, pero con un número de valores limitado o fijo (en este caso *male/female*).
- En la tabla del lado derecho, están representadas las 10 primeras filas del fichero CSV subido, de manera que podamos visualizarlas y ver físicamente los datos que contienen. De esta forma podemos comprobar que en la primera fila aparece un individuo con nombre Braund, Mr. Owen Harris, varón, de 22 años y que viajaba en tercera categoría y que, además, no sobrevivió al hundimiento.

Además, en la tabla de la izquierda, existen dos columnas que nos sirven para anonimizar/enmascarar los datos. En función del tipo de datos detectado en una

columna del fichero CSV, el sistema determina si es susceptible de ser anonimizada/enmascarada, y a estas columnas susceptibles les habilita el *checkbox* correspondiente en su casilla, de manera que podamos activarla para esos campos.

Si seleccionáramos para el campo *Name* la opción de anonimizar, el sistema generaría un código hash para todos los valores de ese campo (tal y como se muestra en la Imagen 30), donde podemos ver que el *Name* de la primera línea (en la tabla de la derecha) ha sido substituido por su código hash (269354413).

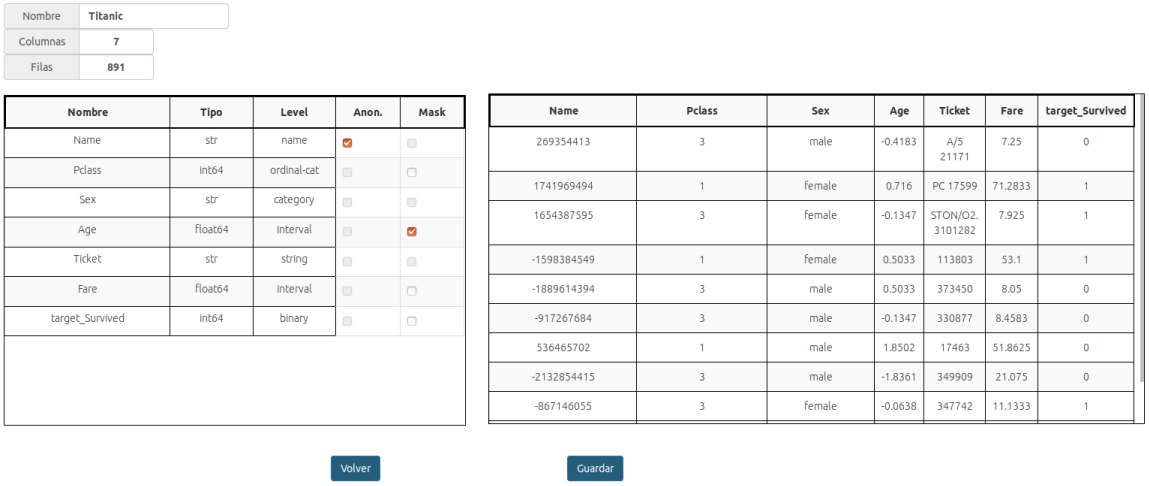


*Imagen 30 - Columnas dataset anonimizadas*

De la misma manera, si en la tabla de la izquierda seleccionamos el *checkbox* de enmascarar para la columna *Age*, podemos ver en la Imagen 30, como modifica el valor de toda la columna para tener, en la primera línea, el valor -0.4183, correspondiente a la función de ofuscación que hemos definido. Esto permite utilizar dichos valores para procesos de *machine learning*, pero no permiten hacerse una idea a la persona que vea los datos de cuál es la distribución de edades de nuestros datos de origen.

## 8.2 Opción de Pipelines

Accediendo a la opción de *pipelines*, podemos generar un nuevo pipeline a partir de un dataset (fichero CSV) ya subido anteriormente. Al seleccionar esta opción, el sistema nos presenta una pantalla con los diferentes *pipelines* que hayamos creado, pudiendo crear nuevos o visualizar los ya existentes.

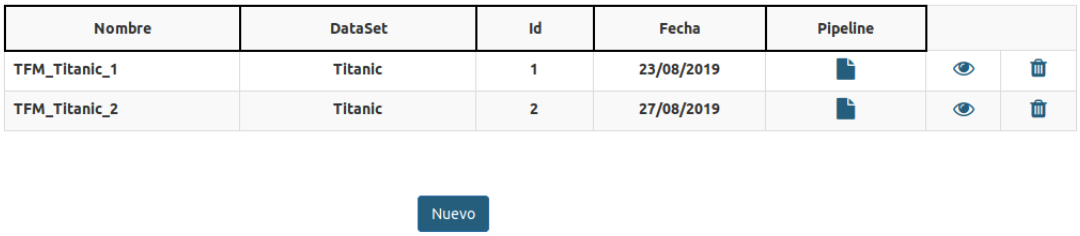


*Imagen 31 - Pantalla de Pipelines*

Los campos que podemos ver en la tabla son muy básicos:

- **Nombre**: El nombre que le dimos en su momento al *pipeline*.
- **Dataset**: el dataset con el que se ha creado
- **Id**: Número de identificación
- **Fecha**: fecha de creación del *pipeline*

Dentro de los *pipelines* ya existentes, podemos seleccionar el icono del *pipeline* para ver las diferentes características de dicho *pipeline*. También podemos, por supuesto, borrar el *pipeline* si ya no nos es de utilidad o visualizar que opciones le dimos al crearlo.

Además, pulsando el botón de nuevo podemos crear un nuevo *pipeline*, en el que además de poder establecer su nombre, podremos elegir desde cuál de los *dataset* queremos crearlo, así como que columnas del *dataset* queremos leer y cuales no (podría darse el caso de que algún dato no nos fuera necesario).

El sistema también puede ir generando de forma automática las particiones necesarias para nuestros procesos de *machine learning*. Podemos seleccionar desde que columna particionaremos y cuantos subconjuntos vamos a realizar, desde uno (solo *training*), dos (*training* y *test*) o tres (*training*, *development* y *test*).

Una de las ventajas del modelo es que genera las particiones a través de un hash de división entera, por lo que, una vez seleccionada una columna, todas las muestras con el mismo valor para una columna irán a parar al mismo conjunto. Por ejemplo, si hemos decidido particionar por la columna DNI, al utilizar este tipo de hash conseguiríamos que todas las muestras con el DNI 55.555.555 fueran a parar al conjunto de *train* (p.e.).

De esta manera, en un hipotético fichero de movimientos bancarios, todos los movimientos de pertenecientes a un cliente irían al dataset de *train* y los del otro irían al dataset de *test*, pero nunca mezclaríamos datos de un cliente en test y en train. Por supuesto, podríamos seleccionar para dicho particionado cualquiera de nuestras columnas, con lo que, si seleccionamos la columna de fecha (utilizando el mes y el año) podríamos hacer que unos meses fueran al dataset de train y unos pocos meses al dataset de test (útil para muchas series temporales):

| DNI | Fecha | Concepto | Debe | Haber | SET |
|---|---|---|---|---|---|
| 11,111,111 | 01/08/2019 | Nomina | | 1000,00 | Train |
| 11,111,111 | 01/08/2019 | Pago Alquiler | 625,00 | | Train |
| 11,111,111 | 01/08/2019 | Pago Colegio | 175,00 | | Train |
| 22,222,222 | 01/08/2019 | Nomina | | 950,00 | Test |
| 11,111,111 | 03/08/2019 | Tarjeta VISA | 115,00 | | Train |
| 22,222,222 | 03/08/2019 | Pago Alquiler | 625,00 | | Test |
| 11,111,111 | 03/08/2019 | Tarjeta VISA | 85,00 | | Train |

*Tabla 7 - Tabla reparto train/test*

Esto soluciona un problema recurrente en los sistemas de *machine learning*, donde se utilizan muestras de un mismo individuo tanto para entrenamiento como para *test*, lo que puede desvirtuar algunos resultados y algunos valores de error. Con el tiempo, la experiencia y algunos sabios consejos, he llegado a la conclusión de que, en general, todas las muestras de un mismo individuo se deberían utilizar para un mismo fin (*train*/*test*) y no mezclarlas en diferentes sets.

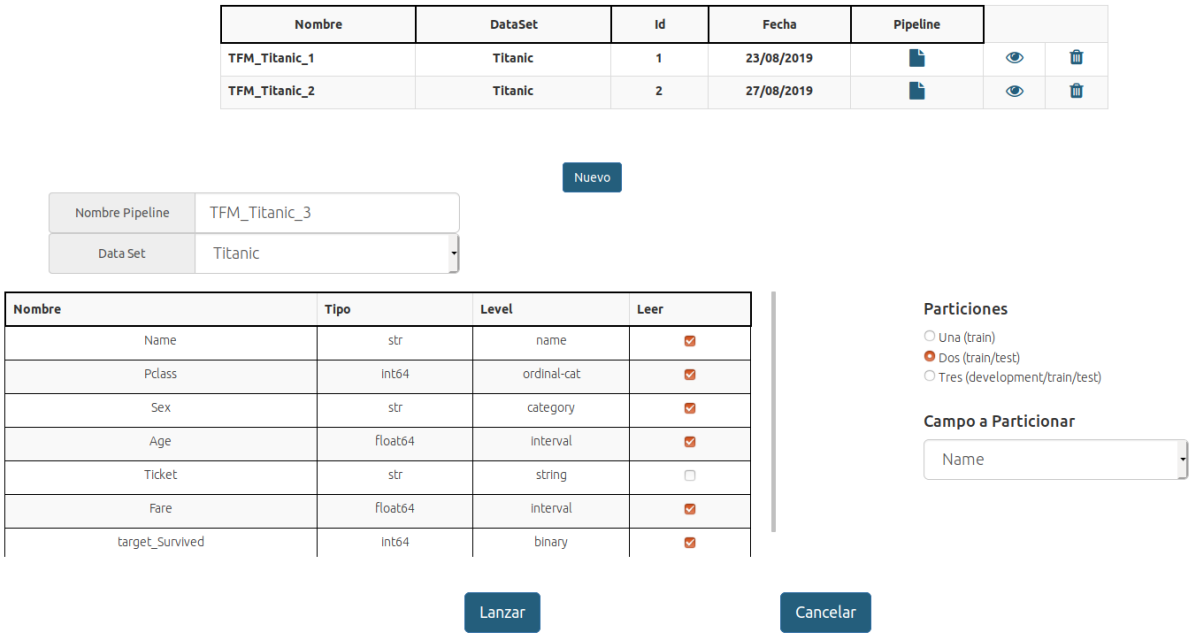


*Imagen 32 - Nuevo Pipeline*

En la Imagen 32 - Nuevo Pipeline se puede observar cómo en nuestro caso de ejemplo, hemos seleccionado para realizar nuestro pipeline el *dataset* del Titanic, seleccionando que no queremos leer e importar la columna de Ticket (puesto que pensamos que dicha columna no aportará ningún valor a nuestro modelo), y, además, que particione nuestro dataset en dos conjuntos, uno de *train* y otro de *test*. Estos conjuntos los generará desde la columna *Name*, por lo que todas las muestras que existieran pertenecientes a un mismo individuo irían a parar a un mismo conjunto de datos (en el ejemplo del Titanic esto no se produce ya que solo existe una muestra para cada individuo, pero he duplicado alguna línea del conjunto de datos para poder reproducirlo).

Una vez seleccionadas las opciones necesarias para la generación de nuestro *pipeline*, pulsando el botón **lanzar,** se ejecutan los procesos necesarios (con las correspondientes llamadas al código *Python* de nuestro backend*)* para la generación de nuestro *pipeline*. Al finalizar el proceso, podemos entrar en la consola de panel de control de DynamoDB de AWS para comprobar los resultados:

- Tras ejecutar el proceso completo, independientemente del tamaño de nuestro fichero, podemos observar en el panel de control de AWS correspondiente a DynamoDB, los datos introducidos en la tabla TFM_Titanic_3. Entre estos datos se puede observar como el *id* ha sido generado para cada elemento y como ha sido asignado a uno de los *datasets* (*train*, *test* o *development*) en función de la columna Name. Además, observamos como dicha columna ha sido anonimizada utilizando un *hash.*

Examen: [Tabla] TFM_Titanic_3: id — Mostrando

| id | Age | Fare | Name | Pclass | Sex | Ticket | datasetgroup | target_Survived |
|---|---|---|---|---|---|---|---|---|
| 159794468247358865 | 22 | 7.25 | b6f3ffac31b386b66e6e4d2b1571c58380ba0d801eb2f8be4b19862b506e09… | 3 | male | A/5 21171 | train | 0 |
| 159794468259990481 | 38 | 71.2833 | 8ed6428503c155da24adf97ca92bf7499711116ea268289aa40de0c352d74… | 1 | female | PC 17599 | train | 1 |
| 159794468294049036 | 22 | 7.25 | b6f3ffac31b386b66e6e4d2b1571c58380ba0d801eb2f8be4b19862b506e09… | 3 | male | A/5 21171 | train | 0 |
| 159794468312756594 | 54 | 51.8625 | 04e22a5547e3bcc602928ec6b8aa0ba7e38a9193881e36ea46d597ba26b2… | 1 | male | 17463 | test | 0 |
| 159794468316686144 | 27 | 11.1333 | d4517f08a650b4444a795a6394a65f1f64f7b57370016d67ec92a0bedcf0e5… | 3 | female | 347742 | test | 1 |
| 159794468330254238 | 35 | 53.1 | f8cfbe6b5355864f34043bdc0b71b1af58c7c9f8ec37bfe110e8ee8ef52a37a… | 1 | female | 113803 | train | 1 |
| 159794468344468317 | 28 | 8.4583 | 66da565ce2c8efa49b15c58146773179166d692f79c54a6711cd4241e7bdd… | 3 | male | 330877 | train | 0 |
| 159794468371515689 | 2 | 21.075 | 1cf1c67a11450c93bb9a0fee2451303ddf75abec762674c59becfe731fc6a80… | 3 | male | 349909 | train | 0 |
| 159794468374574838 | 26 | 7.925 | be921ac4f9d87f6b8f791a017ae36ab1edef979e370974f8d9ce27a0484cbf8… | 3 | female | STON/O2. 3101282 | test | 1 |
| 159794468394708407 | 35 | 8.05 | 705f3961d16e03deb41a0c4b8ad7bd33284b6655d06c44b37dbae36aca07… | 3 | male | 373450 | test | 0 |
| 159794468421407583 | 14 | 7.8542 | 8ef94e21d6988ccd5f49d30884098da4e79540f6eb713bdd6fc489666f461c… | 3 | female | 350406 | test | 0 |
| 159794468440169212 | 55 | 16 | 1d1f8776fe43c1992bd641de24e34ba51d0d8f91315b0f47836c7206201078… | 2 | female | 248706 | train | 1 |
| 159794468457229990 | 20 | 8.05 | c609d7ca42306ae84c4cc4063c436127888c0fe1204d8c56535b014dfd7b5… | 3 | male | A/5. 2151 | train | 0 |
| 159794468461853269 | 4 | 16.7 | 1d73a9be73adbc73e61423caf6c56db16b0aac4f938222843d5ed2d9de27a… | 3 | female | PP 9549 | test | 1 |
| 159794468472695153 | 58 | 26.55 | abb0ad976aff7a752f6c941d2b0962d9228567f2caaf928e63644ce2de701e… | 1 | female | 113783 | train | 1 |
| 159794468485792541 | 14 | 30.0708 | d8e72d78cc3ef1cf90e3c51ca700cf51c14587b8c24e2062ebd4d4c1f81dfe3… | 2 | female | 237736 | train | 1 |
| 159794468491064808 | 39 | 31.275 | 1365ebcc8e79ac25c8658c653b2c6d8ee8ae5f880de140c8442b2d5fa0fb45… | 3 | male | 347082 | train | 0 |

*Imagen 33 - Consola de datos en DynamoDB*

- También podemos observar cómo, las dos primeras líneas, correspondientes al mismo valor para la columna *Name*, han sido asignadas al mismo conjunto de datos de *train*:

| id | Age | Fare | Name | Pclass | Sex | Ticket | datasetgroup |
|---|---|---|---|---|---|---|---|
| 159794319576712609 | 22 | 7.25 | b6f3ffac31b386b66e6e4d2b1571... | 3 | male | A/5 21171 | train |
| 159794319594893402 | 22 | 7.25 | b6f3ffac31b386b66e6e4d2b1571... | 3 | male | A/5 21171 | train |

*Imagen 34 - Detalle DynamoDB*

Al finalizar el proceso, ya tenemos en nuestra base de datos de tipo clave-valor (DynamoDB) *on cloud*, todo nuestro fichero de datos CSV con toda la información correspondiente. Además, filtrando por la columna *datasetgroup*, podemos obtener datasets ya preparados tanto para nuestros procesos de *train* como de *test* o *development* sin necesidad de hacer particionados adicionales.

# 9. Conclusiones

Tal y como comentaba en la introducción y en los objetivos de este trabajo, son solo una pequeña parte de las tareas necesarias las que he pretendido automatizar y simplificar en este Trabajo Fin de Master, aunque, por supuesto, sin intentar dar por resuelto completamente el problema de los *pipelines* y ETLs de datos.

Para ello, he realizado la implementación de un sistema "sencillo" que permita realizar algunas de las tareas más comunes en los procesos de ETL, como es el descubrimiento inferido de los tipos de datos presentes en un fichero CSV, a partir de los propios datos proporcionados por una pequeña muestra de este y su particionado en diferentes conjuntos de datos.

Estos conjuntos de datos serán clasificados y almacenados en un sistema de bases de datos clave-valor *on cloud*, de manera que luego estarán disponibles de forma fácil, rápida y desde cualquier lugar para nuestros sistemas y procesos, no solo de inteligencia artificial, sino de cualquier otro tipo que los necesiten.

Al funcionar todo el sistema vía web, y al estar además diseñado para ser utilizado en la nube, podría ser desplegado y utilizado como un *software as a service* (SaaS), de manera que nuestros usuarios no requirieran de ninguna infraestructura ni de ningún tipo de software (a excepción de conexión a internet y de un navegador) para poder hacer uso del sistema. Además, y de cara a una posible comercialización de este, se podría facturar el servicio por el uso efectivamente realizado, esto es, por el número de ficheros subidos, por el número de *pipelines* generado y/o, incluso, por el número de líneas de datos contenidas en los propios ficheros.

Realmente, la idea del sistema va mucho más allá de la propia generación de los procesos para transformar y almacenar datos. El proyecto podríamos decir que se engloba y forma una pequeña parte de otro proyecto muy superior que consiste en generar un sistema fácil de *machine learning* para el usuario.

Este sistema tendría, por un lado, la parte de ETL para procesar los datos y llevarlos a algún almacén *on cloud* para su gestión. Por otro lado, el sistema debería establecer un dato como objetivo (target) y una serie de modelos y técnicas de *machine learning* a aplicar a nuestro dataset.

El sistema propuesto generaría los modelos y luego realizaría los correspondientes test de estos para poder ofrecer una recomendación de las técnicas más adecuadas y que mejor rendimiento obtienen para aplicar a los datos.

Para el desarrollo del Trabajo Fin de Master, he utilizado muchas tecnologías y disciplinas diferentes, de la misma manera que realizo en mi entorno profesional, y de la misma forma que cualquier Ingeniero en Informática necesita utilizar servicios en la

nube, junto con bases de datos relacionales SQL y bases de datos NOSQL, programando en diferentes lenguajes de programación y utilizando múltiples librerías y herramientas proporcionadas por terceros.

Por último, no creo que los procesos de ETL y transformación de datos vayan a llegar a desaparecer nunca, pero sí que es verdad que, a medida que avanza la tecnología, podemos automatizar determinadas tareas para centrarnos en detalles a los que, muchas veces, no hemos prestado la debida o quizás necesaria atención.

# Índice de imágenes



# Índice de tablas

# Bibliografía

**Amazon Web Services, Inc. 2020.** *Amazon Elastic Compute Cloud.* 2020.

**—. 2020.** *Amazon EMR: Management Guide.* 2020.

**—. 2019.** *Amazon Redshift: Guía de administración de clústeres.* 2019.

**—. 2020.** *AWS Athena: User Guide.* 2020.

**—. 2020.** *AWS Data Pipeline: Guía para desarrolladores.* 2020.

**—. 2020.** *AWS: Developers Documentation.* 2020.

**Apache Software Foundation. 2019.** Apache Airflow. [En línea] 2019. [Citado el: 15 de Junio de 2019.] https://airflow.apache.org/ui.html.

**—. 2019.** Apache Beam. [En línea] 2019. [Citado el: 16 de Junio de 2019.] https://beam.apache.org/.

**Atlassian Support. 2020.** BitBucket Support. [En línea] 2020. https://confluence.atlassian.com/bitbucketserver/using-bitbucket-server-776639769.html.

**Davó Escribá, Ignacio. 2016.** *Instalación y configuración de herramientas software para Big Data.* Trabajo Fin de Grado, Universidad Politécnica de Valencia. 2016.

**INBEST.Cloud.** [En línea] [Citado el: 13 de Enero de 2019.] https://www.inbest.cloud/aws-data-pipeline.

**Mozilla y colaboradores individuales.** Mozilla Developer Network (MDN). [En línea] [Citado el: 15 de Junio de 2019.] https://developer.mozilla.org/en-US/docs/Web/HTML.

**Oracle Corporation and/or its affiliates. 2019.** MySQL™ Reference Manual. [En línea] 2019. https://dev.mysql.com/doc/refman/5.7/en/.

**Python Software Foundation. 2019.** The Python Programming Language. [En línea] 2019. https://www.python.org/.

**Talend Inc.** *Talend.* [En línea] [Citado el: 22 de Abril de 2019.] https://es.talend.com/resources/what-is-etl/.

**The PHP Group. 2019.** [En línea] 2019. [Citado el: 15 de Junio de 2019.] https://php.net/.

**University Of Washington. Department of Biostatistics. 2018.** *Analysis Pipeline on the Cloud.* 2018.