Procesamiento de Datos Masivos

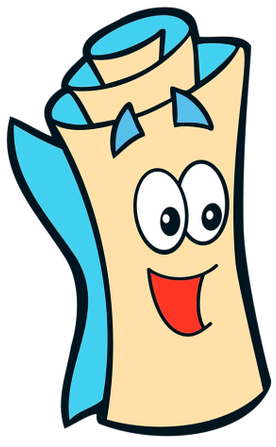
Clase 02 - Data Warehousing y Pipelines de Datos

Recapitulemos un poco el viaje de hacer ciencia de datos

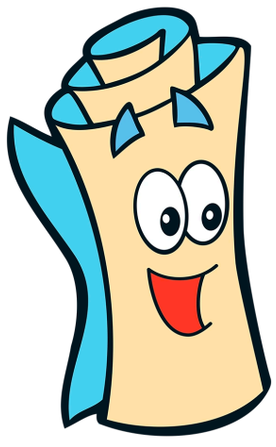

Quiero sacarle valor
a los datos

Pero no sé por donde partir :(

Pero no sé por donde partir :(



Soy el mapa, yo te ayudo!



En general, el ciclo de vida de los datos pasa por las siguientes etapas:

- El lago de datos
- El *data warehouse*
- *El procesamiento de datos*
- El análisis exploratorio de datos
- Modelos de Machine Learning
- Cómo evaluar nuestros modelos
- La puesta en producción

Un pipeline de datos mueve los datos a través de estas etapas

En esta clase

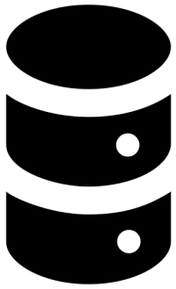
- Aprender sobre el ciclo de vida de los datos
- Entender el rol de un Data Engineer - ML Engineer
- Entregar la noción sobre cómo armar un pipeline de datos
- Entender cómo los modelos de ML pasan a producción

¿Por qué miramos los datos?

Una aplicación típica

En general, al crear una aplicación buscamos resolver un problema del mundo real

- Una aplicación para ahorrar dinero/invertir
- Una aplicación para hacer compras con envío a domicilio
- Una plataforma de aprendizaje en línea
- ...



Generamos datos de las transacciones de los usuarios

Análisis de datos

Cuando generamos datos, hay **conocimiento explícito** que podemos obtener a partir de los datos

En general, podemos obtener este conocimiento con un lenguaje de consultas como SQL

Ej. "Dame todos los usuarios que han comprado el producto **X** en el último mes"

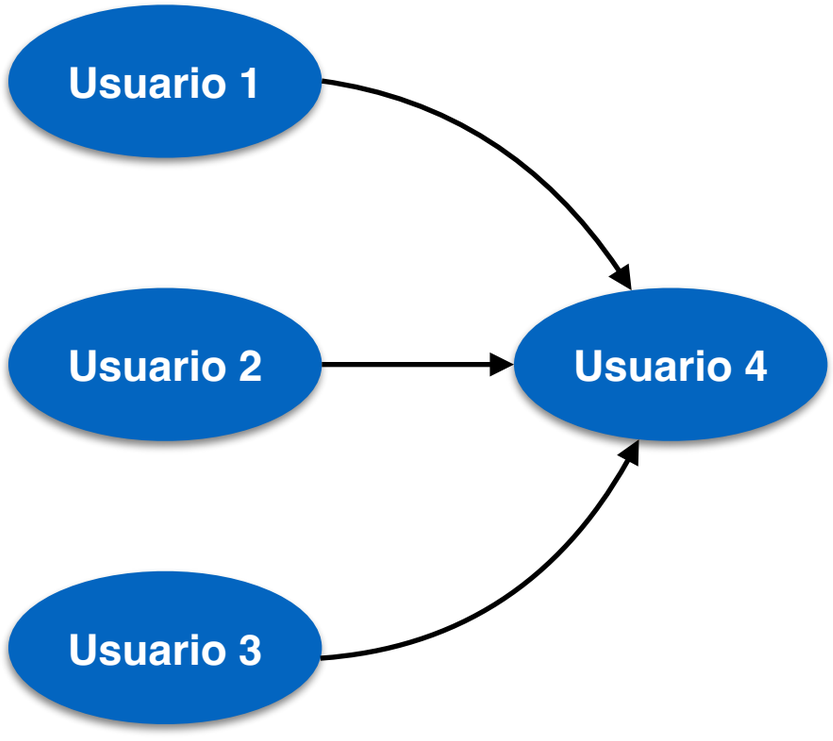
Análisis de datos

Pero hay mucho conocimiento que no está almacenado de forma explícita

Ese contenido lo vamos a extraer haciendo análisis de datos

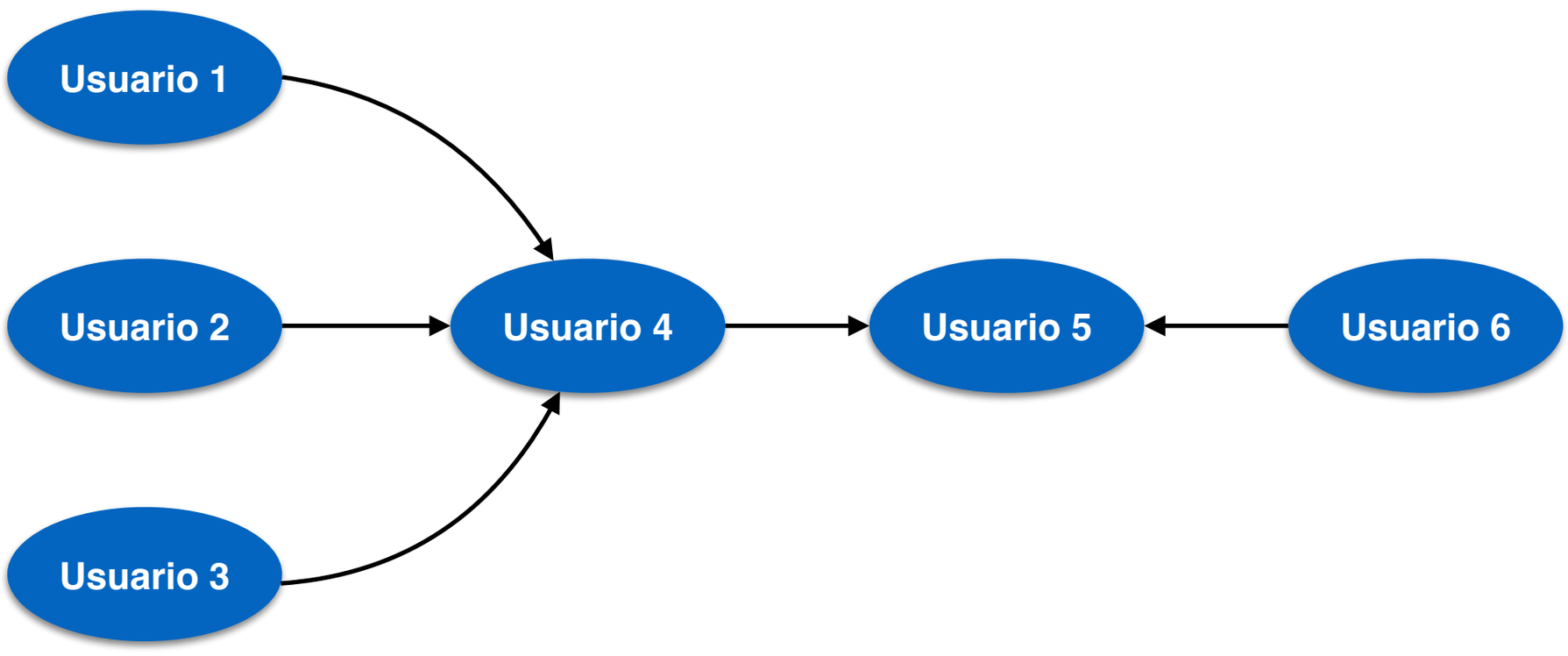
Ejemplo: medidas de centralidad

¿Cuál es el usuario más importante de esta red?



Ejemplo: medidas de centralidad

¿Y en esta red?



Análisis de datos

En el ejemplo anterior, podemos obtener mucha información de los datos explícitos

Pero saber el usuario más importante no es algo que podamos preguntar con un lenguaje de consultas

Análisis de datos

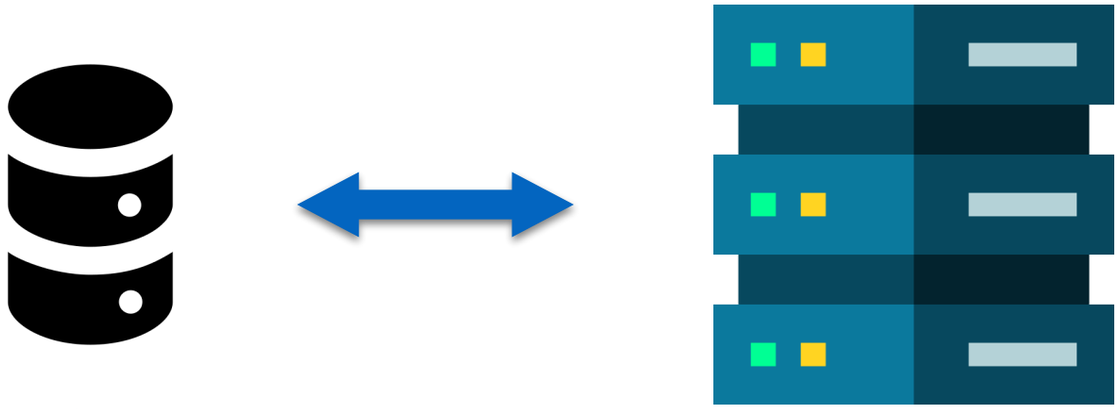
¿Cómo extraemos conocimiento que viene implícito en los datos?

- Data Mining / ML
- Statistical Learning
- IA
- Análisis de redes
- ...

Pero para sacar valor de los datos necesitamos la infraestructura adecuada

La infraestructura para poder analizar datos

Partamos por la aplicación más sencilla mencionada antes



Una base de datos de una aplicación suele ser transaccional

Una BD transaccional

Una BD transaccional está pensada para hacer muchas operaciones livianas al día:

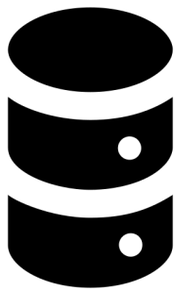
- Inserción, actualización y eliminación de tuplas
- Búsquedas simples: traer el usuario **x**, buscar los productos de la compra **y**, ...
- Además queremos soporte para propiedades ACID

Análisis de datos

Pero en general:

- Estos sistemas no están pensados para realizar consultas pesadas de análisis de datos
- No es buena idea hacer análisis directamente en la base de datos de producción

Además en el futuro tendremos más fuentes de datos



Análisis de datos

Vamos a querer hacer análisis de datos en sistemas especializados para esto: BigQuery, Redshift, Snowflake

Estos sistemas no son transaccionales, ya que están pensadas para hacer pocas consultas por día, pero cada una requiere hartos recursos computacionales

Aquí buscaremos consolidar todos los datos necesarios para los analistas

Data Lake

Un Data Lake es el lugar donde guardo todos los datos generados por mi producto:

- Datos de la base de datos del producto
- Datos del tracking de los usuarios
- Datos de nuestros experimentos (ej. AB testing)

En general, es **información cruda** sin estructura: archivos de distinto tipo, de distintos dominios, etc

Pero en algún momento vamos a procesar estos datos

Data Warehouse

Un Data Warehouse es el lugar en el que guardamos información procesada

En general, son datos puestos a disposición de los analistas para que obtengan valor de los datos

Ejemplo

Una compañía puede tener un repositorio de archivos (como S3 o Cloud Storage) con mucha información cruda; este sería el **lago de datos**

Podemos tener *jobs* que procesen estos archivos y le den estructura y sentido

La información procesada la guardamos en una base de datos de análisis (ej. BigQuery); este lugar con datos estructurados será nuestro **Data Warehouse**

Ingeniería de Datos

Ingeniería de Datos

La Ingeniería de Datos se trata de diseñar e implementar la infraestructura para trabajar con datos

Perfil ligado al desarrollo, pero con conocimiento en datos

Pensemos en las etapas de un pipeline de datos

Data Lake

Data Warehouse

Data Processing

EDA

Modelos de ML

Evaluar Modelos

ML Producción

¿Qué corresponde a ingeniería de datos vs ciencia de datos?

Ingeniería de Datos

Ciencia de Datos

Data Lake

Data Warehouse

Data Processing

EDA

Modelos de ML

Evaluar Modelos

ML Producción

¿Cómo construimos el pipeline?

Pipeline de datos

Un pipeline de datos suele usar el proceso ETL (Extract-Transform-Load) o ELT (Extract-Load-Transform)

Como el almacenamiento es bastante barato hoy en día, optamos por ELT

Requisitos

En general, para construir un pipeline de datos, se recomienda tener las siguientes habilidades:

- Programar muy bien (ojalá en un lenguaje como Python)
- Lo anterior incluye tener nociones sobre control de versiones (ej. GIT), servidores, uso de terminal, ...
- Saber sobre BDs relacionales, tanto de modelación como SQL
- Tener conocimiento sobre sistemas en la nube (ej. GCP)

Stack de tecnologías

Un stack moderno para construir un pipeline de datos incluye las siguientes tecnologías:

- Airflow (u otro framework de orquestación de *jobs*)
- Docker y Kubernetes
- Herramientas en la nube (GCS, BigQuery, Cloud Composer, GKE)

Apache Airflow

Apache Airflow es una herramienta para orquestar tareas

Estas tareas son encapsuladas en operadores (ej. PythonOperator, BashOperator, KubernetesPodOperator)

Algunas personas lo definen como "cron on steroids"

Apache Airflow

Se basa en la idea de definir un DAG (grafo dirigido acíclico) de tareas; este DAG se define mediante código Python

```python
from datetime import datetime

from airflow import models
from airflow.operators.python_operator import PythonOperator

with models.DAG(
    dag_id="dag_example_pythonop",
    schedule_interval="@once",
    start_date=datetime(2022, 1, 1),
    catchup=False,
    tags=["example"],
) as dag:

    def return_constant():
        return 42

    python_op_example = PythonOperator(
        task_id="python_op_example",
        python_callable=return_constant,
    )

    python_op_example
```

```
with models.DAG(
    dag_id="dag_example_pythonop_gcs",
    schedule_interval="@once",
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=["example"],
) as dag:

    def return_constant():
        return 42

    python_op_example = PythonOperator(
        task_id="python_op_example",
        python_callable=return_constant,
    )

    list_bucket = GCSListObjectsOperator(
        task_id="list_bucket",
        bucket=GCS_BUCKET
    )

    python_op_example >> list_bucket
```

Docker

Docker es una herramienta que nos permite encapsular código en imágenes, y ejecutarlas como contenedores

Nos permite tener un entorno de ejecución uniforme en todos los lugares en que el código corra

Ejemplo de un Dockerfile

```dockerfile
FROM python:alpine

# Install GCC
RUN apk add gcc musl-dev libffi-dev

# Upgrade pip
RUN pip install --no-cache-dir --upgrade pip

# Install poetry
RUN pip install poetry

# Installing dependencies
RUN mkdir /app
WORKDIR /app

COPY pyproject.toml /app/

RUN poetry install --no-dev

COPY . /app

CMD ["poetry", "run", "python", "/app/src/main.py"]

```

Docker: algunos detalles

Un Dockerfile nos permite definir una imagen de Docker; podemos definir una imagen a partir de otra

Una imagen es un "computador base" que está listo para correr un comando

Las instancias de una imagen se conocen como contenedores

Docker: algunos detalles

En el ejemplo anterior:

- Tomamos un "computador base" con una distribución mínima de Linux (Alpine) que viene con Python instalado
- Copiamos nuestro código a ese computador base
- Corremos el comando `poetry run`...

Arquitectura del pipeline

En general, nuestros jobs van a hacer imágenes de docker que cumplen funciones específicas:

- Copiar datos hacia el data lake
- Cargar los datos en el *data warehouse*
- Realizar un análisis sobre los datos del *warehouse*

Estas imágenes van a ser orquestadas por Airflow

¿Y las imágenes en qué lugar corren?

Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications

Kubernetes es un tópico en si mismo, pero estudiemos las nociones necesarias para construir un pipeline de datos

Kubernetes

La idea a grandes rasgos es que vamos a tener un *cluster* de Kubernetes que va a tener los recursos computacionales necesarios para correr nuestras imágenes

Algunas cosas que necesitaremos hacer:

- Crear namespaces
- Crear configmaps y secretos
- Monitorear nuestros pods

Airflow y Kubernetes

La forma más típica de hacer *deploy* de Airflow es sobre un *cluster* de Kubernetes

Airflow puede usar el KubernetesPodOperator para ejecutar contenedores en este mismo *cluster*

Los servicios en la nube para correr Airflow ofrecen una instancia corriendo sobre Kubernetes, con posibilidad de ejecutar muchos *jobs* sobre el mismo cluster

Un poco más de Airflow

Si bien Airflow tiene un operador para correr código Python, los cómputos pesados no se deberían realizar en Airflow

Por eso, uno delega las tareas pesadas a un *cluster* de Kubernetes que pueda escalar a medida que es necesario

Data Warehousing

Data Warehouse

Ya sabemos que un Data Warehouse es el lugar donde guardamos información procesada, lista para que los analistas puedan consumirla

Un Data Warehouse moderno se construye sobre un sistema en la nube como BigQuery, Snowflake o Redshift

En este curso vamos a aprender a usar Big Query, pero todos estos sistemas tienen los mismos principios

Big Query

BigQuery is Google's fully managed, ***serverless data warehouse*** *that enables scalable analysis over petabytes of data*

Serverless

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers

Serverless

En un modelo serverless:

- No manejamos servidores, solo una interfaz de alto nivel
- No nos preocupamos de hacer escalar nuestro servicio, el proveedor de la nube lo hace solo
- En general mientras más recursos usamos más pagamos ($)

Data Warehouse en la Nube

Cuando usamos una herramienta de Data Warehouse en la nube:

- Tenemos espacio "ilimitado" (en la medida que podamos pagar)
- Tenemos recursos para ejecutar consultas de analítica **muy** pesadas
- Tenemos integraciones con diversos lenguajes/ programas

Data Warehouse en la Nube

Arquitectura

La arquitectura de un sistema de Data Warehouse en la nube se basa en un almacenamiento distribuido

Al hacer consultas comunicamos datos a un cluster de cómputo de alta disponibilidad

Al hacer una consulta disponemos de **muchos** "computadores pequeños" para responderla

¿Cómo podemos responder una consulta utilizando varios computadores pequeños?

Algoritmos en BigQuery

Big Query divide una consulta en varias consultas pequeñas

Cada sub-consulta se envía a un *slot* de trabajo distinto

En general, las consultas hacen hashing a los distintos servidores

Algoritmos en BigQuery

Joins (de la documentación de BQ)

Broadcast joins

- When joining a large table to a small table, BigQuery creates a broadcast join where the small table is sent to each slot processing the large table
- Even though the SQL query optimizer can determine which table should be on which side of the join, it is recommended to order joined tables appropriately

Algoritmos en BigQuery

Joins (de la documentación de BQ)

Hash joins

- When joining two large tables, BigQuery uses hash and shuffle operations to shuffle the left and right tables so that the matching keys end up in the same slot to perform a local join. This is an expensive operation since the data needs to be moved.

Algoritmos en BigQuery

Agregación

¿Podemos usar Hashing para responder consultas de agregación?

Algoritmos en BigQuery

Big Query no usa índices típicos para responder consultas

Cada vez que hacemos una consulta, cada tabla mencionada se carga en memoria completa al menos una vez

Estos sistemas se basan en la "capacitad ilimitada" de cómputo

Big Query y SQL

Big Query usa SQL para responder consultas

Tiene algunos comandos específicos, pero en general es igual al SQL de toda la vida

¿Qué comandos de SQL son especialmente útiles en un Data Warehouse?

Big Query y SQL

Además de hacer joins y agregaciones, existe un tipo de comandos muy importante en Data Warehousing

Estos comandos son las **Window Functions**, funciones que nos permiten hacer cierto tipo de analítica

Estudiaremos estas funciones con algunos ejemplos prácticos

Procesamiento de Datos Masivos

Clase 02 - Data Warehousing y Pipelines de Datos